PAPER

Efficient Storage and Querying of Horizontal Tables Using a PIVOT Operation in Commercial Relational DBMSs*

Sung-Hyun SHIN[†], *Nonmember*, **Yang-Sae MOON**[††a)], *Member*, **Jinho KIM**[††], *and* **Sang-Wook KIM**[†], *Nonmembers*

SUMMARY In recent years, a *horizontal table* with a large number of attributes is widely used in OLAP or e-business applications to analyze multidimensional data efficiently. For efficient storing and querying of horizontal tables, recent works have tried to transform a horizontal table to a traditional *vertical table*. Existing works, however, have the drawback of not considering an optimized *PIVOT* operation provided (or to be provided) in recent commercial RDBMSs. In this paper we propose a formal approach that exploits the optimized PIVOT operation of commercial RDBMSs for storing and querying of horizontal tables. To achieve this goal, we first provide an overall framework that stores and queries a horizontal table using an equivalent vertical table. Under the proposed framework, we then formally define 1) a method that stores a horizontal table in an equivalent vertical table and 2) a PIVOT operation that converts a stored vertical table to an equivalent horizontal view. Next, we propose a novel method that transforms a user-specified query on horizontal tables to an equivalent *PIVOT-included* query on vertical tables. In particular, by providing transformation rules for all five elementary operations in relational algebra as theorems, we prove our method is theoretically applicable to commercial RDBMSs. Experimental results show that, compared with the earlier work, our method reduces storage space significantly and also improves average performance by several orders of magnitude. These results indicate that our method provides an excellent framework to maximize performance in handling horizontal tables by exploiting the optimized PIVOT operation in commercial RDBMSs.

key words: *PIVOT, horizontal tables, relational algebra, query transformation, OLAP*

1. Introduction

On-Line Analytical Processing (OLAP) provides a multidimensional analysis method to extract a variety of useful information from a large amount of data stored in data warehouses [3], [6], [11]. To analyze these multidimensional data efficiently, we generally store or represent the original data as various forms of transformed data [13]. One of these representation methods is a *horizontal table* [1]. To represent complex multidimensional data in a relatively simple format, a horizontal table forms a two-dimensional table whose columns correspond to values of dimension attributes [8], [13]. Figure 1 (a) shows an example of the horizontal table *SalesH* that manages 'sales' quantities for every electronic product and for every shop at the same time. As shown in the figure, we can easily recognize or summarize sales quantities for each shop or for each product. Thus, we can say that a horizontal table provides an easy view to perform multidimensional data analysis efficiently. Like a PIVOT table provided in Microsoft Excel and a cross table used in representing statistics data, a horizontal table has a horizontal form of schema structures consisting of a large number of columns (i.e., attributes) [1], [9], [15], [17], [18].

The traditional relational DBMSs (RDBMSs), however, cannot support a horizontal table efficiently. This is because the traditional RDBMSs have focused on handling a *vertical table* with a large number of tuples (rows) but a small number of attributes (columns). Figure 1 (b) shows an example of the vertical table *SalesV*. Thus, most RDBMSs have an explicit limitation on the number of attributes to be supported. For example, Microsoft SQL Server 2005 [5], [12] and Oracle 9i [14], which are representative commercial RDBMSs, limit the maximum number of attributes in a table to 1,024. However, OLAP or e-business applications must support a horizontal table with thousands or tens of thousands of attributes [1]. Therefore, we need to maintain a horizontal table for multidimensional data analysis as an equivalent vertical table for commercial RDBMSs.

To solve the problem of storing horizontal tables in RDBMSs, Agrawal et al. [1] have proposed a novel method that uses a vertical table for a horizontal view. For this, they first convert a horizontal table to an equivalent vertical table and store the vertical table in relational databases. They then transform a horizontal table-based query to an equiv-

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

(a) A horizontal table (*SalesH*)

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

(b) A vertical table (*SalesV*)

Fig. 1 An example of horizontal and vertical tables.

Manuscript received March 16, 2007.
Manuscript revised November 26, 2007.

[†]The authors are with the College of Information and Communications, Hanyang University, Korea.
[††]The authors are with the Department of Computer Science, Kangwon National University, Korea.

*This work was partially supported by the Ministry of Science and Technology (MOST)/Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc). Also, this work was partially supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Advancement) under grant IITA-2007-C1090-0701-0040.

a) E-mail: ysmoon@kangwon.ac.kr

DOI: 10.1093/ietisy/e91–d.6.1719

alent vertical table-based query. More specifically, their method transforms a relational algebra operation on horizontal tables to an equivalent operation on vertical tables. Through experiments, they have also shown that processing transformed queries on vertical tables is more efficient than processing the original queries on horizontal tables. In recent years, however, a *PIVOT* operation [4], [5] is provided (or to be provided) in commercial RDBMSs to support a horizontal view of a stored vertical table [12], [14]. Here, PIVOT is the operation of converting the roles of rows and columns in a relation table, and it is used for transforming attribute values of a vertical table into attribute names of the corresponding horizontal table. Agrawal et al.'s approach did not consider this optimized PIVOT operation since their method was introduced before commercial RDBMSs provided a PIVOT operation as the internal feature. Also, they have not provided formal proofs for query transformation rules in relational algebra. In contrast, in this paper we propose an efficient method of storing tables and processing queries by fully exploiting this optimized PIVOT operation. That is, the major difference between our work and Agrawal et al.'s work is whether to use a PIVOT operation or not. Using the PIVOT operation, however, causes many technical problems in query processing frameworks, query transformation rules, and implementation details, and our solution is quite different from Agrawal et al.'s. Moreover, by exploiting the recent PIVOT operation of commercial RDBMSs, our solution can be widely and practically used in many e-business applications.

In this paper, we propose a formal approach that uses the optimized PIVOT operation of commercial RDBMSs for storing and querying of horizontal tables. To achieve this goal, we first provide an overall framework that stores and queries a horizontal table using an equivalent vertical table. Under the framework, we then formally define 1) a method that stores a horizontal table in an equivalent vertical table and 2) a PIVOT operation that transforms a stored vertical table to an equivalent horizontal view. Next, we propose a novel method that uses the PIVOT operation in transforming an original horizontal table-based query to an equivalent vertical table-based query. That is, we propose a systematic way of transforming a user-specified query on horizontal tables to an equivalent *PIVOT-included* query on vertical tables. In particular, to achieve completeness of query transformation rules, we consider all of the five elementary operations in relational algebra [16] as theorems and formally prove the theorems. That is, we propose transformation rules for projection (π), selection (σ), set union ($\cup$), set difference ($-$), and the Cartesian product ($\times$), respectively, and prove correctness of the rules.

The proposed method is much more practical and gives better performance than the earlier work. First, our method is easy to implement compared with the earlier work by Agrawal et al. [1]. It is because we simply exploit the PIVOT operation in transforming queries while they have to rewrite the corresponding SQL statements line by line against the complex algebra operations. Thus, our method provides a practical implementation mechanism that enables normal users as well as experts to handle horizontal tables in commercial RDBMSs. Second, our method improves performance compared with Agrawal et al.'s one. It is because we use the optimized PIVOT operation while they have to execute complex SQL statements step by step. That is, the recent commercial RDBMSs consider various optimization techniques to support a PIVOT operation as its internal feature [5], and we use this optimized PIVOT operation to improve performance. Experimental results show that, compared with the earlier work, our method reduces storage space significantly and also improves performance by several orders of magnitude.

The rest of this paper is organized as follows. Section 2 describes related work. Section 3 explains an overall framework for storing and querying of horizontal tables. Section 4 presents query transformation rules for all five elementary operations in relational algebra. Section 5 shows the results of performance evaluation. Section 6 summarizes and concludes the paper.

2. Related Work

Several previous researches identified usefulness of horizontal tables in OLAP or e-business applications [1], [2], [8], [17]. Gray et al. [8] and Witkowski et al. [17] presented a spreadsheet and a cross table to easily represent the multidimensional data for naive users. These spreadsheet and cross table have the similar concept with a horizontal table. However, they focused on efficiently storing and querying of a horizontal table itself, but they did not try to solve the problem of a horizontal table or to transform the table to an equivalent vertical table. Next, Agrawal et al. [1] insisted that horizontal tables were practically important in many e-business applications since a lot of e-business data could be viewed as the form of horizontal tables. They then pointed out the problem of storing a horizontal table in the traditional table structure and proposed a novel method of representing a horizontal table as an equivalent vertical table. Recently, Beckmann et al. [2] argued that the proper way to handle sparse data was not to use vertical tables, but rather to extend the RDBMS tuple storage format to allow the representation of sparse attributes as interpreted fields. To use their approach in a commercial RDBMS, however, we have to modify major functions of its internal engine, which is not feasible in practice. Thus, we do not consider this storage format modification approach, but focus on Agrawal et al.'s vertical table approach.

Lakshmanan et al. [10] first proposed a transformation method that converts a horizontal table to an equivalent vertical table. They presented four reorganization operations, *unfold*, *fold*, *split*, and *unite*, to efficiently store a horizontal table with a large number of attributes. Here, *unfold* and *fold* are the operations of transforming a horizontal table to a vertical table, and vice versa. Based on this concept, Agrawal et al. [1] proposed a novel and systematic method that stores and queries a horizontal table using an equiv-

Table 1 Summary of notation.

Symbols	Definitions
H	a horizontal table
V	a vertical table
Oid	an attribute for tuple (or object) identifiers
O_i	a value for the tuple identifier (the i-th tuple in the horizontal table H)
A, M	attributes in the vertical table V (A_i is a value of A; $M_{i,j}$ a value of M)
A_j	the j-th attribute in the horizontal table H (= a value that A in V has)
$M_{i,j}$	a value of the attribute A_j in the horizontal table H where $Oid = O_i$ (= a value of the attribute M in the vertical table V where $Oid = O_i$ and $A = A_j$)

alent vertical table. That is, to provide a horizontal view for users, they stored its equivalent vertical table internally. By storing a vertical table rather than a horizontal table, Agrawal et al. were able to overcome the limitation on the number of attributes in RDBMSs. Also, they maximized storage utilization and improved performance by not storing *null values*, which were a critical problem of horizontal tables. Agrawal et al.'s solution, however, has the drawback of not considering the PIVOT operation optimized in commercial RDBMSs. The reason is their solution was introduced before commercial RDBMSs provided a PIVOT operation as their internal feature. Also, they have not provided formal proofs for query transformation rules in relational algebra.

In addition to the works above, many researchers have studied on a horizontal table. First, Witkowski et al. [17], [18] defined multidimensional data as a horizontal table in the form of a spreadsheet and proposed a manipulation method for the horizontal table. Through defining and manipulating the horizontal table, they tried to enable naive users to more easily use analysis functions of OLAP. Second, Cunningham et al. [5] formally defined PIVOT and UNPIVOT in the form of relational algebra. Like *unfold* and *fold*, their PIVOT and UNPIVOT are the operations of transforming a horizontal table to a vertical table, and vice versa. They then presented SQL statements to implement the algebra operations and proposed query optimization techniques to use the PIVOT/UNPIVOT operations together with other elementary operations. Third, Chen et al. [4] provided *GPIVOT* as generalization of PIVOT/UNPIVOT, and proposed an incremental maintenance algorithm for the GPIVOT-based materialized views. All of the earlier solutions, however, were introduced without considering the PIVOT operation optimized in commercial RDBMSs. Thus, our solution differs from the earlier ones in that we fully exploit the optimized PIVOT operation in order to maximize performance for storing and querying of horizontal tables.

3. The Proposed Framework with a PIVOT Operation

In this section we explain an overall framework that handles a horizontal table using an equivalent vertical table. We first summarize in Table 1 the notation to be used throughout the paper. According to the notation in Table 1, we then represent a horizontal table and a vertical table as in Fig. 2. As shown in the figure, the horizontal table H has the schema of $(Oid, A_1, A_2, \ldots, A_n)$, and each tuple in H is

Table H

Oid	A_1	A_2	...	A_j	...	A_n
:	:	:	...	:	...	:
O_i	$M_{i,1}$	$M_{i,2}$		$M_{i,j}$	...	$M_{i,n}$
:	:	:	...	:	...	:

(where $M_{i,j} = \perp$ or $\neq \perp$)

(a) Representation of a horizontal table

Table V

Oid	A	M
:	:	:
O_i	A_1	$M_{i,1}$
O_i	A_2	$M_{i,2}$
:	:	:
O_i	A_j	$M_{i,j}$
:	:	:
O_i	A_n	$M_{i,n}$
:	:	:

(where $M_{i,j} \neq \perp$)

(b) Representation of a vertical table

Fig. 2 Representation of horizontal and vertical tables.

represented as $(O_i, M_{i,1}, M_{i,2}, \ldots, M_{i,n})$. The vertical table V, which is transformed from the horizontal table H, has the schema of (Oid, A, M), and each tuple in V is represented as $(O_i, A_j, M_{i,j})$. Here, we note that $M_{i,j}$ in H can be either null ($=\perp$) or not null while $M_{i,j}$ in V cannot be null.

We now explain how we can store the horizontal table H in the equivalent vertical table V. As shown in Fig. 2, we map a tuple in the horizontal table H to several tuples in the vertical table V. That is, by changing roles of rows and columns, we can convert a horizontal table to an equivalent vertical table. Agrawal et al. [1] and Chen et al. [4] have already defined this converting procedure in the form of relational algebra. In this paper we adopt Chen et al. [4]'s relational algebra representation since it is more recent work. The following Eq. (1) shows a formula to store the horizontal table H in the vertical table V without any information loss [4].

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

As shown in Eq. (1), for each tuple $(O_i, M_{i,1}, M_{i,2}, \ldots, M_{i,n})$ in the horizontal table H, we generate a tuple $(O_i, A_j, M_{i,j})$ $(1 \leq j \leq n)$ in the vertical table V if $M_{i,j}$, the value of the attribute A_j, is not null. According to Eq. (1), the vertical table does not store any null value while the horizontal table may store null values. Thus, storage space for a horizontal table with m rows and n columns is proportional to $m \cdot n$, while that for an equivalent vertical table is proportional to $((m \cdot n) - (n \cdot m \cdot \alpha))$. Here, α $(0 \leq \alpha \leq 1)$ is a null density [1]. And accordingly, storage space for a vertical table becomes much smaller than that for a horizontal table as α increases. That is, converting a horizontal table to its equivalent vertical table is a very efficient way of storing e-business tables that generally have a lot of null values [1], [10].

Next, we explain the PIVOT operation that represents the vertical table V as the horizontal table H, i.e., provides the horizontal view H instead of the stored vertical table V. Commercial RDBMSs such as Microsoft SQL Server 2005 [12] and Oracle 9i also provide (or are about to provide) this PIVOT operation for an easy and simple analysis. As an inverse operation of the storing procedure in Eq. (1), we formally define the PIVOT operation as the following Eq. (2) [4].

$$H = \text{PIVOT}^{[A_1,\ldots,A_n]}_{A \text{ on } M}(V) = \left[\rightthreetimes^{n}_{j=1} \pi_{Oid,M}(\sigma_{A='A_j'}(V))\right] \quad (2)$$

As shown in Eq. (2), PIVOT on V against $[A_1, \ldots, A_n]$ generates the equivalent horizontal view H. We explain the right-hand side of Eq. (2) in more detail as follows: we first make temporary results by extracting tuples $(O_i, A_j, M_{i,j})$ from the vertical table V for every value A_j of the attribute A; we then integrate the results through the full outer join ($\rightthreetimes$); and we finally provide the result as the horizontal view H for the vertical table V. In contrast, we note that the PIVOT operation in commercial RDBMSs may provide the best performance since the RDBMSs usually use various optimization techniques as their internal feature [5]. Therefore, in order to efficiently handle the horizontal tables in commercial RDBMSs, we simply use the PIVOT operation ($= \text{PIVOT}^{[A_1,\ldots,A_n]}_{A \text{ on } M}(V)$) itself rather than a quite complex relational algebra operation ($= \left[\rightthreetimes^{n}_{j=1} \pi_{Oid,M}(\sigma_{A='A_j'}(V))\right]$) in Eq. (2).

Last, we explain an overall framework that processes a query on the horizontal table H using the equivalent vertical table V. Figure 3 shows the overall framework for transforming a query between horizontal and vertical tables. As depicted in the figure, we transform a query *Query* on the horizontal table H to an equivalent query *Query'* on the vertical table V. In the left side of the figure, we obtain *Result* that by evaluating the original query on the horizontal table H while, in the right side, we obtain *Result'* by evaluating the transformed query on the vertical table V. As shown in the figure, if we use the transformation rules to be proposed, *Result* of the left side will be the same as *Result'* of the right side. Through transforming a horizontal table-based query to a vertical table-based query, therefore, we can obtain the same query result without storing the horizontal table. In particular, we use the optimized PIVOT operation to maximize performance in transforming a query between horizontal and vertical tables. For this, we present the PIVOT-included query transformation rules for all five elementary operations in relational algebra, and formally prove the rules.

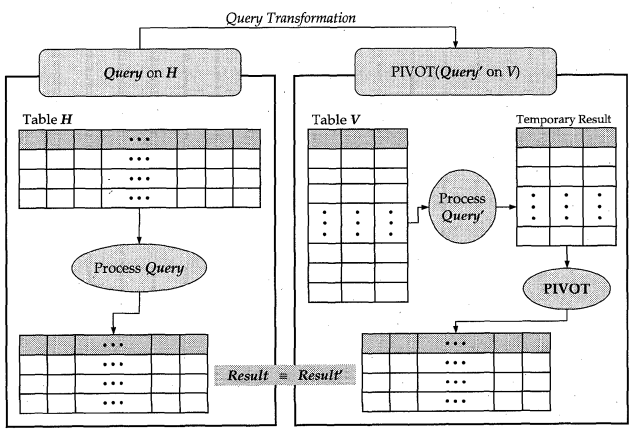


Fig. 3 An overall framework for transforming a query between horizontal and vertical tables.

4. PIVOT-Included Query Transformation Rules

In this section we present the query transformation rules that convert a horizontal table-based query to an equivalent vertical table-based query. In particular, we will enforce that the transformed query contains the PIVOT operation to maximize the performance. To guarantee completeness of query transformation rules, we deal with all of the five elementary operations in relational algebra [16]. We provide the rules for the projection (π) in Sect. 4.1, the selection (σ) in Sect. 4.2, and the set operations ($\cup, -, \times$) in Sect. 4.3, respectively.

4.1 Projection

We can transform a projection query on the horizontal view to an equivalent PIVOT-included query on the stored vertical table. The following Theorem 1 presents this query transformation rule.

Theorem 1: The projection on the horizontal table H can be transformed to the equivalent PIVOT-included operation on the vertical table V as in Eq. (3):

$$\pi_{Oid,B_1,\ldots,B_k}(H) \left(= \left[\rightthreetimes^{k}_{j=1} \pi_{Oid,M}(\sigma_{A='B_j'}(V))\right]\right) = \text{PIVOT}^{[B_1,\ldots,B_k]}_{A \text{ on } M}(V) \quad (3)$$

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

Proof: We first prove that $\pi_{Oid,B_1,\ldots,B_k}(H)$ is identical to $\left[\rightthreetimes^{k}_{j=1} \pi_{Oid,M}(\sigma_{A='B_j'}(V))\right]$ using the mathematical induction. Then, Eq. (3) holds by the PIVOT definition in Eq. (2). For the more detailed proof, refer to Appendix. □

According to Theorem 1, the projection that extracts the given attributes $(B_1, \ldots, B_k)$ from the horizontal table H is identical to the PIVOT operation against the attributes on the vertical table V. It is because, as we explained in Sect. 3, the attribute A_j and its value $M_{i,j}$ in the horizontal table H is transformed to a tuple $(O_i, A_j, M_{i,j})$ in the vertical table V. Here, we note that Eq. (3) differs from Eq. (2) in that Eq. (3) is performed against the given attributes $(B_1, \ldots, B_k)$ while Eq. (2) against all of the attributes $(A_1, \ldots, A_n)$.

Example 1: Recall *SalesH* and *SalesV* in Fig. 1. Figure 4 shows an example of transforming a projection on the horizontal table *SalesH* to an equivalent PIVOT-included

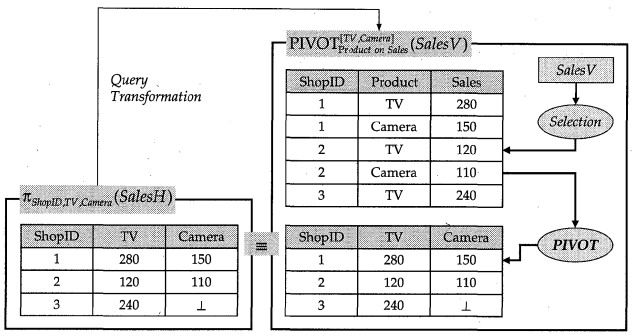


Fig. 4 An example of transforming a projection query.

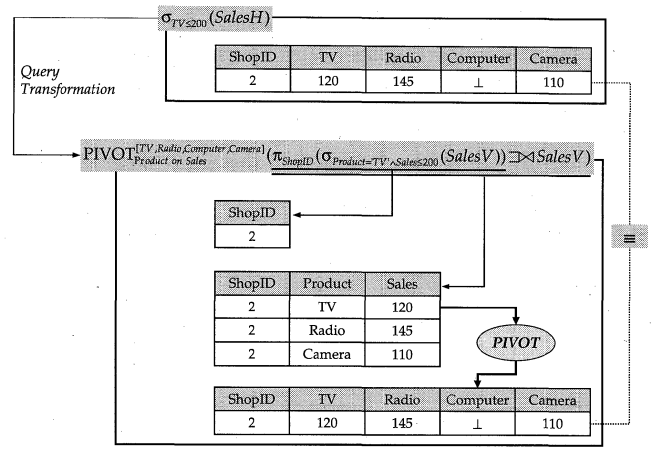


Fig. 5 An example of transforming a selection query.

operation on the vertical table *SalesV*. In Fig. 4, the projection on *SalesH*, $\pi_{ShopID,TV,Camera}(SalesH)$, is transformed to an equivalent PIVOT-included operation on *SalesV*, $\text{PIVOT}^{[TV,Camera]}_{Product\ on\ Sales}(SalesV)$, based on Theorem 1. As depicted in the right part of Fig. 4, the selection on the vertical table V is internally performed first, and then columns and rows are changed by the PIVOT definition†. And accordingly, the result from H will be identical to that from V. □

4.2 Selection

We can transform a selection query on the horizontal view to an equivalent PIVOT-included selection and projection query on the stored vertical table. To derive the transformation rule, we first explain the notation used in [1], [4]. Each predicate on a selection query has the form of $A_i\theta M_i$, and several predicates are concatenated by the logical AND operator ($\wedge$). Here, A_i is an attribute name of H, and M_i is a constant value that A_i has. And, θ is one of the comparison operators ($=, \neq, <, \leq, >, \geq$). Based on this predicate notation, we present a query transformation rule for the selection as follows.

Theorem 2: The selection on the horizontal table H can be transformed to the equivalent PIVOT-included selection and projection operation on the vertical table V as in Eq. (4):

$$\begin{aligned}
&\sigma_{\wedge_{i=1}^{k}(A_i\theta M_i)}(H) \\
&\quad (= [\rtimes\!\!\ltimes_{j=1}^{n} \pi_{Oid,M}(\sigma_{A='A_j'}(\\
&\qquad (\cap_{i=1}^{k}\pi_{Oid}(\sigma_{A='A_i'\wedge M\theta'M_i'}(V))) \rtimes V))]) \\
&\quad = \text{PIVOT}^{[A_1,\ldots,A_n]}_{A\ on\ M} \\
&\qquad ((\cap_{i=1}^{k}\pi_{Oid}(\sigma_{A='A_i'\wedge M\theta'M_i'}(V))) \rtimes V) \qquad (4)
\end{aligned}$$

Proof: By using the mathematical induction, we first prove that $\sigma_{\wedge_{i=1}^{k}(A_i\theta M_i)}(H)$ is identical to the intermediate formula $(= [\rtimes\!\!\ltimes_{j=1}^{n} \cdots \rtimes V])$. Then, Eq. (4) holds by the PIVOT definition in Eq. (2). For the more detailed proof, refer to Appendix. □

According to Theorem 2, the selection that extracts the tuples satisfying the given predicates from the horizontal table H is identical to the following operations: 1) extract *Oid*'s that satisfy the predicates from the vertical table V; 2) perform the left outer join ($\rtimes$) between the *Oid*'s and the original V; and 3) use the PIVOT operation to convert the vertical form into the equivalent horizontal form. Here, the reason why we use the left outer join for *Oid*'s is to select all of the tuples that satisfy the selection predicates from the vertical table V. It means that we first select the tuples satisfying the predicates from the vertical table, and then convert the result into the horizontal form using the PIVOT operation.

Example 2: Recall *SalesH* and *SalesV* in Fig. 1. Figure 5 shows an example of transforming a selection on the horizontal table *SalesH* to an equivalent PIVOT-included operation on the vertical table *SalesV*. That is, the selection on *SalesH*, $\sigma_{TV\leq 200}(SalesH)$, is transformed to an equivalent PIVOT-included selection and projection operation on *SalesV*, $\text{PIVOT}^{[TV,Radio,Computer,Camera]}_{Product\ on\ Sales}(\cdots)$, based on Theorem 2. As shown in the lower part of Fig. 5, the transformed query is executed as follows. 1) The tuples that satisfy the given predicates are selected from *SalesV*, and their *Oid*'s are extracted into a temporary result $(= \pi_{Oid}(\sigma_{Product='TV'\wedge Sales\leq 200}(SalesV)))$. Let the result be O. 2) The left outer join between the result O and the original vertical table V is performed to include every tuple whose *Oid* is contained in O $(= O \rtimes SalesV)$. 3) Columns and rows are changed by the PIVOT definition $(= \text{PIVOT}^{[TV,Radio,Computer,Camera]}_{Product\ on\ Sales}(O \rtimes SalesV))$. And accordingly, the result from H will be identical to that from V. □

4.3 Set Operations

4.3.1 Set Union

To formally derive the transformation rule for the set union, let H_i denote a horizontal table and V_i be its equivalent vertical table. In order to use the set union for two or more tables, the tables should have the same schema [16], thus we

†Actually, the implementation details of the PIVOT operation may differ among RDBMSs. For easy understanding, however, we describe the intermediate results based on the PIVOT definition in Eq. (2).

assume all of the horizontal tables H_i have the same schema structure. The following Theorem 3 presents a query transformation rule for the set union.

Theorem 3: The set union among m horizontal tables H_i ($1 \leq i \leq m$) can be transformed to the equivalent PIVOT-included set union operation among m vertical tables V_i as in Eq. (5):

$$\begin{aligned} &\cup_{i=1}^{m}(H_i) \\ &\quad (= \left[\rtimes\!\!\sqsubset_{j=1}^{n} \pi_{Oid,M}(\sigma_{A=`A_j'}(\cup_{i=1}^{m}(V_i))\right]) \\ &\quad = \mathrm{PIVOT}_{A\ on\ M}^{[A_1,\ldots,A_n]}(\cup_{i=1}^{m}(V_i)) \end{aligned} \tag{5}$$

Proof: The following Eq. (6) trivially holds according to Eq. (1) and the query optimization rules [7].

$$\begin{aligned} &\cup_{i=1}^{m}(V_i) \\ &\quad = \cup_{i=1}^{m}([\cup_{j=1}^{n}\pi_{Oid,`A_j',A_j}(\sigma_{A_j\neq\perp}H_i)]) \\ &\quad = [\cup_{j=1}^{n}\pi_{Oid,`A_j',A_j}(\sigma_{A_j\neq\perp}(\cup_{i=1}^{m}(H_i))]) \end{aligned} \tag{6}$$

Then, Eq. (6) can be represented as Eq. (5) by the PIVOT definition in Eq. (2). This completes the proof. □

According to Theorem 3, the set union among several horizontal tables H_i is identical to the PIVOT operation on the result obtained by the set union among the corresponding vertical tables V_i.

4.3.2 Set Difference

Except that the order of tables should be preserved, a transformation rule for the set difference is the same as that for the set union. The following Theorem 4 presents a query transformation rule for the set difference.

Theorem 4: The set difference among m horizontal tables H_i ($1 \leq i \leq m$) can be transformed to the equivalent PIVOT-included set difference operation among m vertical tables V_i as in Eq. (5). Here, $-_{i=1}^{m}(H_i) = H_1 - H_2 - \cdots - H_m$.

$$\begin{aligned} &-_{i=1}^{m}(H_i) \\ &\quad (= \left[\rtimes\!\!\sqsubset_{j=1}^{n} \pi_{Oid,M}(\sigma_{A=`A_j'}(-_{i=1}^{m}(V_i))\right]) \\ &\quad = \mathrm{PIVOT}_{A\ on\ M}^{[A_1,\ldots,A_n]}(-_{i=1}^{m}(V_i)) \end{aligned} \tag{7}$$

Proof: We can easily prove the theorem by replacing the set union as the set difference in Theorem 3. Thus, we omit the details. □

According to Theorem 4, the set difference among several horizontal tables H_i is identical to the PIVOT operation on the result obtained by the set difference among the corresponding vertical tables.

4.3.3 The Cartesian Product

We can also transform the Cartesian product query among the horizontal tables to an equivalent PIVOT-included Cartesian product query among the vertical tables. Unlike the set union or the set difference, in case of the Cartesian product, each horizontal table H_i can have its own different schema [16]. The following Theorem 5 presents a query transformation rule for the Cartesian product.

Theorem 5: The Cartesian product among m horizontal tables H_i ($1 \leq i \leq m$) can be transformed to the equivalent PIVOT-included Cartesian product operation among m vertical tables V_i as in Eq. (8). Here, $\times_{i=1}^{m}(H_i) = H_1 \times H_2 \times \cdots \times H_m$, Ai and Mi are the attributes of the vertical table V_i (i.e., Ai and Mi in V_i can be seen as A and M in V).

$$\begin{aligned} &\times_{i=1}^{m}(H_i) \\ &\quad (= [\rtimes\!\!\sqsubset_{j=1}^{n} \pi_{(Oid1,\ldots,Oidm),(M1,\ldots,Mm)} \\ &\qquad (\sigma_{(A1,\ldots,Am)=(`A1_j',\ldots,`Am_j')}(\times_{i=1}^{m}(V_i))]) \\ &\quad = \mathrm{PIVOT}_{(A1,\ldots,Am)\ on\ (M1,\ldots,Mm)}^{[(A1_1,\ldots,Am_1),\ldots,(A1_n,\ldots,Am_n)]} \\ &\qquad (\times_{i=1}^{m}(V_i)) \end{aligned} \tag{8}$$

Proof: We can easily prove the theorem by replacing 1) the set difference as the Cartesian product, 2) Oid as $(Oid1,\ldots,Oidm)$, 3) A as $(A1,\ldots,Am)$, and 4) M as $(M1,\ldots,Mm)$, respectively, in Theorem 3. Thus, we omit the details. □

According to Theorem 5, the Cartesian product among several horizontal tables is identical to applying the PIVOT operation to the result obtained by the Cartesian product among the corresponding vertical tables. That is, the transformation rule is very similar to that of the set union or the set difference. However, the Cartesian product differs from the two operations in the following points. As the result of the Cartesian product among m vertical tables, the new identifier of the form $(Oid1,\ldots,Oidm)$ rather than Oid is generated. Also, as the result of the PIVOT operation, the new tuples of the schema $[(Oid1,\ldots,Oidm),\ (A1_1,\ldots,Am_1),\ \ldots,\ (A1_n,\ldots,Am_n)]$ rather than $(Oid, A_1,\ldots,A_n)$ are generated. Likewise, the result of the Cartesian product among horizontal tables is the same as that of the PIVOT operation against new identifiers and new attributes after performing the Cartesian product among vertical tables.

5. Performance Evaluation

In this section we present the experimental results for the proposed query transformation rules. We describe the experimental data and environment and explain the results of storage space comparison in Sect. 5.1, and present the results of performance evaluation in Sect. 5.2.

5.1 Experimental Data and Environment

We have implemented three methods: 1) the method of using horizontal tables (we call it *H-Method*), 2) the method by Agrawal et al. [1] (we call it *ASX* by taking authors' initials), and 3) the proposed method (we call it *V-Method*). These three experimental methods can be summarized as follows:

- H-Method: We store a horizontal table itself directly

in databases and process the given query on the stored horizontal table without any transformation.
- ASX: As the method by Agrawal et al., we transform a horizontal table to an equivalent vertical table and store it in databases. In this method, however, we do not use the PIVOT operation. That is, we convert each relation algebra operation to the equivalent SQL statements and to execute the statements step by step.
- V-Method: Like ASX, we transform a horizontal table to an equivalent vertical table and store it in databases. But, unlike ASX, we use the PIVOT operation provided in SQL Server 2005 DBMS.

We generate the following schemes for the horizontal table H and the vertical table V, respectively:

$$H = (Oid\ \texttt{integer}, A_1\ \texttt{float}, A_2\ \texttt{float}, \ldots, A_n\ \texttt{float})$$
$$V = (Oid\ \texttt{integer}, A\ \texttt{char(4)}, M\ \texttt{float})$$

As shown in the schemes above, we set the type of the tuple identifier *Oid* in H (or V) as the `integer` type, those of the attributes A_i and M in H as the `float` type, and that of the attribute A in V as the `char` type. To improve performance in processing the queries, we set *clustering* on the attribute *Oid* for each table.

The hardware platform for the experiment is a PC equipped with an Intel Pentium IV 1.70 GHz CPU, 1 GB RAM, and an 80 GB hard disk. The software platform is Microsoft Windows XP operating system and SQL Server 2005 DBMS. Like Agrawal et al.'s experiment [1], we have measured the elapsed time by varying the data set, the null density, and the selectivity value. We first generated data in horizontal format and then transformed it into its equivalent vertical format. We kept the size of a table (number of rows $\times$ number of columns in a row) constant by adjusting the number of rows as we varied the number of columns, and generated four horizontal tables, $200 \times 100K$, $400 \times 50K$, $800 \times 25K$, and $1000 \times 20K$. We used 90% and 95% as the null density [1], and varied the selectivity according to the query types. Finally, we perform the TPC-H benchmark. We use two queries, Q1 and Q6, among 22 queries provided in TPC-H benchmark. We select these Q1 and Q6 queries since they are easily modeled with five elementary operations presented in our work (By the way, modeling other 20 queries except Q1 and Q6 with elementary operations is very difficult since they include aggregations and complex joins).

The proposed V-Method significantly reduces the storage space compared with H-Method: V-Method reduces the storage space to $\frac{1}{5}$ of that for H-Method on the average when the null density is 90%; $\frac{1}{10}$ on the average when the null density is 95%. This space reduction is due to that V-Method does not store any null value in the vertical table while H-Method needs to store null values in the horizontal table [1]. That is, the null density makes a large effect on the storage space of H-Method since null values are stored in H-Method; in contrast, it does not make any effect on the storage space of V-Method since null values are not stored in V-Method.

5.2 Performance Results

In this section we explain the performance evaluation results for 1) the projection, 2) the selection, and 3) the mixed operation of projection and selection. Among the five transformation rules presented in Sect. 4, we focus on projection and selection, but exclude set operations. It is because selection and projection are very often used in real query environment while the set operations are hardly used. Also, the performance difference among three methods for set operations is similar to that for selection and projection. Thus, we focus on selection and projection only in the experiments.

5.2.1 Projection

Figure 6 shows the experimental results of three methods for the projection queries. Figure 6 (a) shows the case where the null density is 90%, and Fig. 6 (b) the case where the null density is 95%. For each table, we measure the elapsed time by varying the number of attributes to 5, 10, 20, and 40.

As shown in Fig. 6 (a), V-Method outperforms ASX as well as H-Method in all cases. The reason why V-Method outperforms H-Method is trivial. It is because the PIVOT operation used in V-Method selects only a few tuples from the vertical table while the projection used in H-Method have to retrieve all of the tuples from the horizontal table. V-Method also outperforms ASX since it uses the PIVOT operation optimized in Microsoft SQL Server 2005 [12]. That is, ASX needs to perform the outer join to combine the intermediate tuples selected from the vertical table while our

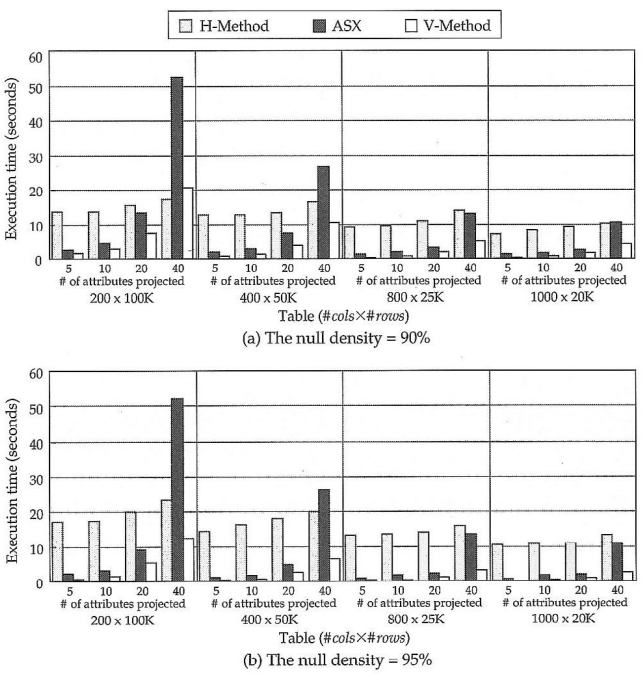


Fig. 6 Performance comparison of H-Method, ASX, and V-Method for projection queries.

V-Method simply exploits the optimized PIVOT operation to do it. As shown in the figure, as the number of attributes increases, performance is not changed in H-Method, but degraded in both V-Method and ASX. The reason is that H-Method retrieves all of the tuples from the horizontal table regardless of the number of attributes projected while V-Method and ASX retrieve more tuples from the vertical table as the number of attributes increases. In particular, if the ratio of "the number of all the attributes" to "the number of attributes projected" is large (i.e., in the case where the number of attributes is 40 in the table $200 \times 100K$ of Fig. 6 (a)), V-Method shows a little bit worse performance than H-Method. This degradation means that even if V-Method uses the optimized PIVOT operation, its performance may become worse than H-Method as the number of attributes projected rapidly increases. In summary of the results in Fig. 6 (a), our V-Method improves performance 4.0 times over H-Method and 2.4 times over ASX on the average.

The performance difference between V-Method and H-Method in Fig. 6 (b) (the case where the null density is 95%) is much larger than that in Fig. 6 (a) (the case where the null density is 90%). It is because the null density affects the number of tuples retrieved in V-Method. That is, as the null density increases, the number of tuples retrieved from the horizontal table is not changed, but that from the vertical table decreases. In summary of the results in Fig. 6 (b), our V-Method improves performance 8.0 times over H-Method and 3.6 times over ASX on the average.

5.2.2 Selection

Figure 7 shows the experimental results for the selection queries. Fig. 7 (a) shows the case where the null density is 90%, and Fig. 7 (b) the case where the null density is 95%. For each table, we measure the elapsed time by varying the selectivity to 0.5%, 1%, 5%, and 10%.

As shown in Figs. 7 (a) and 7 (b), our V-Method outperforms both ASX and H-Method regardless of the null density. We note that H-Method shows the worst performance. This worst performance is due to that storage space for H-Method is much larger than that for V-Method (or ASX). That is, we need to retrieve all of the tuples stored in the table in order to confirm whether each tuple satisfies the given predicates or not, and thus, H-Method with the largest storage space shows the worst performance. As shown in the figure, V-Method slightly outperforms ASX since it exploits the optimized PIVOT operation. In summary of the results for selection queries, our V-Method improves performance 19.9 times over H-Method and 1.2 times over ASX on the average.

5.2.3 Mixed Operation

Figure 8 shows the experimental results for the mixed operation queries. As shown in Fig. 8 (a), we first measure the elapsed time by changing the selectivity to 0.5%, 1%, 5%, and 10%. Here, we use $200 \times 100K$ as the horizontal table and 90% as the null density. We then perform the experiment by changing the number of attributes projected to 5, 10, 20, and 40 in Fig. 8 (b). In Fig. 8 (b), we use 95% as the null density and 5% as the selectivity. According to Figs. 8 (a) and 8 (b), our V-Method significantly improves performance over H-Method and ASX in all the experimental cases. In summary of the results for mixed queries, our V-Method improves performance 5.9 times over H-Method and 1.8 times over ASX on the average.

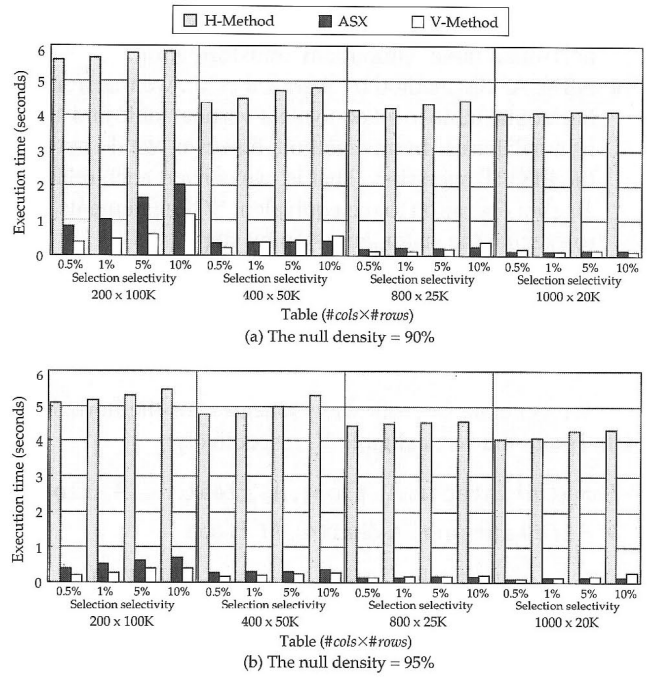


Fig. 7 Performance comparison of H-Method, ASX, and V-Method for selection queries.

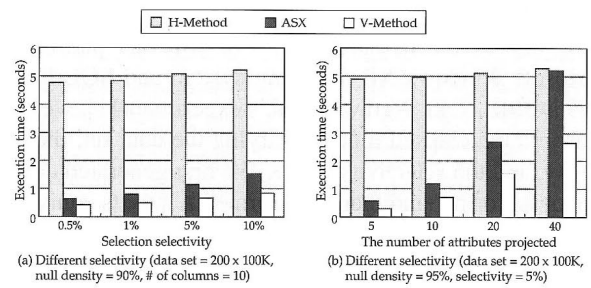


Fig. 8 Performance comparison of H-Method, ASX, and V-Method for mixed queries.

5.2.4 TPC-H Benchmark

Figure 9 shows the TPC-H benchmark results of three methods. We performed this TPC-H benchmark in order to evaluate three methods in a more objective manner and to confirm the superiority of our approach in a well known environment. Figure 9 (a) shows the experimental results of Q1, the pricing summary report query, and Fig. 9 (b) those of Q6,

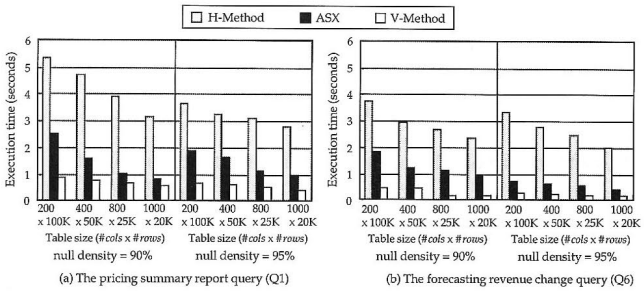


Fig. 9 Experiment results on the TPC-H benchmark.

the forecasting revenue change query. In this benchmark, we used the same data sets and null densities with the previous experiments. As shown in Fig. 9, our V-Method still outperforms ASX and H-Method in all cases. In summary of the TPC-benchmark, V-Method improves performance 8.8 times over H-Method and 3.0 times over ASX on the average. In conclusion, our V-Method significantly improves average performance over ASX as well as H-Method even in the TPC-H benchmark, which simulates a realistic query environment.

6. Conclusions

In OLAP or e-business applications, a horizontal table with a large number of attributes is widely used to analyze multidimensional data efficiently. The traditional RDBMSs, however, cannot support a horizontal table efficiently since they have an explicit limitation on the number of attributes to be supported. To solve this problem, several solutions have been proposed to transform a horizontal table (a horizontal table-based query) to an equivalent vertical table (an equivalent vertical table-based query). All of the earlier solutions, however, have been introduced without considering a PIVOT operation optimized in commercial RDBMSs. In recent years, the optimized PIVOT operation [5], [12] has been considered an important operation to provide the horizontal view of the stored vertical table. Therefore, we have proposed a formal approach that uses the best optimized PIVOT operation of commercial RDBMSs in storing and querying horizontal tables.

Contributions of the paper can be summarized as follows. First, we have provided an overall framework that stores and queries a horizontal table using an equivalent vertical table. Second, we have formally defined 1) a method that stores a horizontal table in an equivalent vertical table and 2) a PIVOT operation that provides a horizontal view for the stored vertical table. Third, we have proposed a novel method that transforms the given queries on horizontal tables to the equivalent PIVOT-included queries on vertical tables. In particular, by providing transformation rules for all five elementary operations in relational algebra as theorems, we have proven our method is theoretically applicable to commercial RDBMSs. Fourth, we have presented that our method reduces storage space significantly, by up to $\frac{1}{10}$, compared with the naive method. Fifth, through extensive experiments, we have shown our method improves average performance by several orders of magnitude compared with earlier ones.

These results indicate that our method provides an excellent framework to maximize performance in handling horizontal tables by exploiting the PIVOT operation optimized in commercial RDBMSs. As future works, we will extend the transformation rules to supporting join or aggregate operations and show superiority of the proposed method for these join or aggregate operations through experiments.

References

[1] R. Agrawal, A. Somani, and Y. Xu, "Storage and querying of e-commerce data," Proc. 27th Int'l Conf. on Very Large Data Bases, pp.149–158, Roma, Italy, Sept. 2001.

[2] J.L. Beckmann, A. Halverson, R. Krishnamurthy, and J.F. Naughton, "Extending RDBMSs to support sparse datasets using an interpreted attribute storage format," Proc. 22nd Int'l Conf. on Data Engineering, IEEE, p.58, Atlanta, GA, April 2006.

[3] S. Chaudhuri and U. Dayal, "An overview of data warehousing and technology," ACM SIGMOD Record, vol.26, no.1, pp.65–74, March 1997.

[4] S. Chen and E.A. Rundensteiner, "GPIVOT: Efficient incremental maintenance of complex ROLAP views," Proc. 21st Int'l Conf. on Data Engineering (ICDE), IEEE, pp.552–563, Tokyo, Japan, April 2005.

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

[6] J.-P. Dittrich, D. Kossmann, and A. Kreutz, "Bridging the gap between OLAP and SQL," Proc. 31st Int'l Conf. on Very Large Data Bases, pp.1031–1042, Trondheim, Norway, Aug. 2005.

[7] R. Elmasri and S. Navathe, Fundamentals of Database Systems, 3rd ed., Addison-Wesley, 2000.

[8] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, "Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals," Data Mining and Knowledge Discovery, vol.1, no.1, pp.29–53, March 1997.

[9] L.V.S. Lakshmanan, F. Sadri, and S.N. Subramanian, "SchemaSQL — A language for querying and restructuring multi-database systems," Proc. 22nd Int'l Conf. on Very Large Data Bases, pp.239–250, Bombay, India, Sept. 1996.

[10] L.V.S. Lakshmanan, F. Sadri, and S.N. Subramanian, "On efficiently implementing schemaSQL on an SQL database system," Proc. 25th Int'l Conf. on Very Large Data Bases, pp.471–482, Edinburgh, Scotland, Sept. 1999.

[11] C. Li, B.C. Ooi, A.K.H. Tung, and S. Wang, "DADA: A data cube for dominant relationship analysis," Proc. Int'l Conf. on Management of Data, ACM SIGMOD, pp.659–670, Chicago, Illinois, June 2006.

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

[13] M. Mohania, S. Samtani, J. Roddick, and Y. Kambayashi, "Advances and research directions in data warehousing technology," Australian Journal of Information Systems, vol.7, no.1, pp.41–59, Dec. 1999.

[14] Oracle 9i Database, http://www.oracle.com/database/

[15] S.S.B. Shi, E. Stokes, D. Byrne, C.F. Corn, D. Bachmann, and T. Jones, "An enterprise directory solution with DB2," IBM Syst. J., vol.39, no.2, pp.360–383, 2000.

[16] J.D. Ullmann, Principles of Database and Knowledge-Base Systems, vol.I, Computer Science Press, 1988.

[17] A. Witkowski, S. Bellamkonda, T. Bozkaya, G. Dorman, N. Folkert, A. Gupta, L. Sheng, and S. Subramanian, "Spreadsheets in RDBMS for OLAP," Proc. Int'l Conf. on Management of Data, ACM SIGMOD, pp.52–63, San Diego, California, June 2003.
[18] A. Witkowski, S. Bellamkonda, T. Bozkaya, N. Folkert, A. Gupta, L. Sheng, and S. Subramanian, "Business modeling using SQL spreadsheets," Proc. 29th Int'l Conf. on Very Large Data Bases, pp.1117–1120, Berlin, Germany, Sept. 2003.
[19] Y. Zhao, P. Deshpande, and J. Naughton, "An array-based algorithm for simultaneous multidimensional aggregates," Proc. Int'l Conf. on Management of Data, ACM SIGMOD, pp.159–170, Tucson, Arizona, June 1997.

Appendix A: Proof of Theorem 1

Using the mathematical induction, we first show the following Eq. (A· 1) holds for the number k of attributes projected:

$$\pi_{Oid,B_1,\ldots,B_k}(H) = \left[\rightthreetimes\!\!\!\!\!\!\subset_{j=1}^{k} \pi_{Oid,M}(\sigma_{A=`B_j'}(V))\right] \quad (A\cdot 1)$$

Induction basis: If the number of attributes projected is one (let the attribute be B_1), then Eq. (A· 2) trivially holds by the definitions of projection and selection. Thus, Eq. (A· 1) holds when k is one.

$$\pi_{Oid,B_1}(H) = [\pi_{Oid,M}(\sigma_{A=`B_1'}(V))] \quad (A\cdot 2)$$

Induction hypothesis: Assume that Eq. (A· 1) holds when the number of attributes projected is k (let the attributes be $B_1, \ldots, B_k$).

Induction step: Then, by Eqs. (A· 3)~(A· 6), Eq. (A· 1) also holds when the number of attributes projected is $k+1$ (let the attributes be $B_1, \ldots, B_k, B_{k+1}$). In the proving steps below, Eq. (A· 3) trivially holds by the definitions of the projection (π) and the full outer join ($\rightthreetimes\!\!\!\!\!\!\subset$). And, Eq. (A· 5) holds by Eq. (A· 2) in the induction basis and Eq. (A· 1) in the induction hypothesis. Also, Eq. (A· 6) holds by the definition of the full outer join.

$$\begin{aligned}
&\pi_{Oid,B_1,\ldots,B_k,B_{k+1}}(H) \\
&= \pi_{Oid,B_1,\ldots,B_k,B_k}(H) \rightthreetimes\!\!\!\!\!\!\subset \pi_{Oid,B_{k+1}}(H) && (A\cdot 3)\\
&= \left[\rightthreetimes\!\!\!\!\!\!\subset_{j=1}^{k} \pi_{Oid,M}(\sigma_{A=`B_j'}(V))\right] && (A\cdot 4)\\
&\quad \rightthreetimes\!\!\!\!\!\!\subset [\pi_{Oid,M}(\sigma_{A=`B_{k+1}'}(V))] && (A\cdot 5)\\
&= \left[\rightthreetimes\!\!\!\!\!\!\subset_{j=1}^{k+1} \pi_{Oid,M}(\sigma_{A=`B_j'}(V))\right] && (A\cdot 6)
\end{aligned}$$

As a result, we have shown that Eq. (A· 1) holds by the mathematical induction. Next, Eq. (A· 7) holds by the PIVOT definition (Eq. (2) in Sect. 3), and this completes the proof.

$$\left[\rightthreetimes\!\!\!\!\!\!\subset_{j=1}^{k} \pi_{Oid,M}(\sigma_{A=`B_j'}(V))\right] = \mathrm{PIVOT}_{A\ on\ M}^{[B_1,\ldots,B_k]}(V) \quad (A\cdot 7)$$

□

Appendix B: Proof of Theorem 2

Using the mathematical induction, we first show the following Eq. (A· 8) holds for the number k of selection predicates:

$$\begin{aligned}
&\sigma_{\wedge_{i=1}^{k}(A_i\theta M_i)}(H) \\
&= [\rightthreetimes\!\!\!\!\!\!\subset_{j=1}^{n} \pi_{Oid,M}(\sigma_{A=`A_j'}((\cap_{i=1}^{k}\pi_{Oid} \\
&\quad (\sigma_{A=`A_i'\wedge M\theta `M_i'}(V))) \rightthreetimes V))] && (A\cdot 8)
\end{aligned}$$

Induction basis: If the number of predicates is one (let the predicate be $A_1 = M_1$), then Eq. (A· 9) holds by the definition of storing tables in Eq. (1). That is, if a value of the attribute A_1 is M_1 in the horizontal table H, a tuple satisfying the predicates '$A = A_1 \wedge M = M_1$' should be contained in the vertical table V [1].

$$\begin{aligned}
&\sigma_{A_1\theta M_1}(H) \\
&= [\pi_{Oid}(\sigma_{A=`A_1'\wedge M\theta `M_1'}(V))] \\
&\quad \rightthreetimes \left[\rightthreetimes\!\!\!\!\!\!\subset_{j=1}^{n} \pi_{Oid,M}(\sigma_{A=`A_j'}(V))\right] && (A\cdot 9)
\end{aligned}$$

Here, the distributive law among the outer joins holds (i.e., $A \rightthreetimes (B \rightthreetimes\!\!\!\!\!\!\subset C) = (A \rightthreetimes B) \rightthreetimes\!\!\!\!\!\!\subset (A \rightthreetimes C)$) [16], and the order of projections and selections can be changed by the query optimization technique [7]. Thus, we can rewrite Eq. (A· 9) to Eq. (A· 10), and accordingly, Eq. (A· 8) holds when k is one.

$$\begin{aligned}
&\sigma_{A_1\theta M_1}(H) \\
&= [\rightthreetimes\!\!\!\!\!\!\subset_{j=1}^{n} \pi_{Oid,M}(\sigma_{A=`A_j'}(\\
&\quad (\pi_{Oid}(\sigma_{A=`A_1'\wedge M\theta `M_1'}(V))) \rightthreetimes V))] && (A\cdot 10)
\end{aligned}$$

Induction hypothesis: Assume that Eq. (A· 8) holds when the number of selection predicates is k (let the predicates be $\wedge_{i=1}^{k}(A_i = M_i)$).

Induction step: Then, by Eqs. (A· 11)~(A· 14), Eq. (A· 8) also holds when the number of predicates is $k + 1$ (let the predicates be $\wedge_{i=1}^{k+1}(A_i = M_i)$). In the proving steps below, Eqs. (A· 11) and (A· 12) hold by the definitions of the logical AND ($\wedge$) and the set union ($\cap$). And, Eq. (A· 13) holds by Eq. (A· 10) in the induction basis and Eq. (A· 8) in the induction hypothesis. Also, Eq. (A· 14) holds by the definition of the set union, the associative law between the set union and the outer join in relation algebra [16], and the query optimization technique [7] on selection, projection, and set union operations.

$$\begin{aligned}
&\sigma_{\wedge_{i=1}^{k+1}(A_i\theta M_i)}(H) \\
&= \sigma_{(\wedge_{i=1}^{k}(A_i\theta M_i))\wedge(A_{k+1}\theta M_{k+1})}(H) && (A\cdot 11)\\
&= \sigma_{\wedge_{i=1}^{k}(A_i\theta M_i)}(H) \cap \sigma_{(A_{k+1}\theta M_{k+1})}(H) && (A\cdot 12)\\
&= [\rightthreetimes\!\!\!\!\!\!\subset_{j=1}^{n} \pi_{Oid,M}(\sigma_{A=`A_j'}(\\
&\quad (\cap_{i=1}^{k}\pi_{Oid}(\sigma_{A=`A_i'\wedge M\theta `M_i'}(V))) \rightthreetimes V))] \\
&\quad \cap[\rightthreetimes\!\!\!\!\!\!\subset_{j=1}^{n} \pi_{Oid,M}(\sigma_{A=`A_j'}((\\
&\quad \pi_{Oid}(\sigma_{A=`A_{k+1}'\wedge M\theta `M_{k+1}'}(V))) \rightthreetimes V))] && (A\cdot 13)\\
&= [\rightthreetimes\!\!\!\!\!\!\subset_{j=1}^{n} \pi_{Oid,M}(\sigma_{A=`A_j'}((\\
&\quad \cap_{i=1}^{k+1}\pi_{Oid}(\sigma_{A=`A_i'\wedge M\theta `M_i'}(V))) \rightthreetimes V))] && (A\cdot 14)
\end{aligned}$$

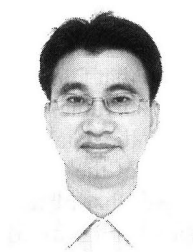
As a result, we have shown that Eq. (A· 8) holds by the mathematical induction. Next, Eq. (A· 15) holds by the PIVOT definition (Eq. (2) in Sect. 3), and this completes the proof.

$$[\rightthreetimes\!\!\!\!\!\!\subset_{j=1}^{n} \pi_{Oid,M}(\sigma_{A=`A_j'}($$

$$((\cap_{i=1}^{k} \pi_{Oid}(\sigma_{A='A_i' \wedge M\theta'M_i'}(V))) \rtimes V))]$$
$$= \text{PIVOT}_{A \text{ on } M}^{[A_1,\ldots,A_n]}$$
$$((\cap_{i=1}^{k} \pi_{Oid}(\sigma_{A='A_i' \wedge M\theta'M_i'}(V))) \rtimes V) \quad (A \cdot 15)$$

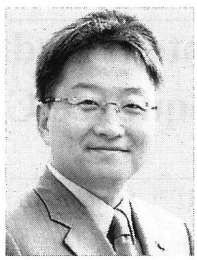
□

Sung-Hyun Shin received B.S. (2000) in Computer Science from Kwandong University and M.S. (2002) and Ph.D. (2007) degrees in Computer Science from Kangwon National University respectively. He is currently in a post-doctorial program at Hanyang University. His research interests include data warehousing and OLAP, XML database, Database application, and Moving object database.



Yang-Sae Moon received B.S. (1991), M.S. (1993), and Ph.D. (2001) degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST). From 1993 to 1997, he was a research engineer in Hyundai Syscomm, Inc., where he participated in developing 2G and 3G mobile communication systems. From 2002 to 2005, he was a technical director in Infravalley, Inc., where he participated in planning, designing, and developing CDMA and W-CDMA mobile network services and systems. He is currently an assistant professor at Kangwon National University. His research interests include data mining, knowledge discovery, storage systems, access methods, mobile/wireless communication systems, and network comunication systems. He is a member of the IEEE and a member of the ACM.

Jinho Kim received B.S. (1982) in Electrical Engineering from Kyungpook National University and M.S. (1985) and Ph.D. (1990) degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) respectively. From 1990, he joined at the Department of Computer Science, Kangwon National University. Now he is a professor at Kangwon National University. He was a visiting scholar at the University of Michgian in 1995 to 1996 and at the Drexel Univeristy in 2003 to 2004 respectively. His research interests include data warehousing and OLAP, data mining, main-memory database systems, XML databases, and information retrieval. He is a member of the IEEE and a member of the ACM.



Sang-Wook Kim received the B.S. degree in Computer Engineering from Seoul National University, Seoul, Korea at 1989, and earned the M.S. and Ph.D. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea at 1991 and 1994, respectively. From 1994 to 1995, he worked with the Information and Electronics Research Center in Korea, as a Senior Engineer. From 1995 to 2003, he served as an Associate Professor of the Division of Computer, Information, and Communications Engineering at Kangwon National University, Chunchoen, Kangwon, Korea. In 2003, he joined Hanyang University, Seoul, Korea, where he currently is a Professor at the School of Information and Communications. From 1999 to 2000, he worked with the IBM T. J. Watson Research Center, Yorktown Heights, New York, as a Post-Doc. He also visited the Computer Science Department of Stanford University as a Visiting Researcher in 1991. He is an author of over 80 papers in refereed international journals and conference proceedings. His research interests include storage systems, transaction management, main-memory DBMSs, embedded DBMSs, data mining, multimedia information retrieval, and geographic information systems. He is a member of the ACM and the IEEE.