관계형 데이타베이스에서 다차원 데이타의 뷰를 위한 효율적인 질의 변환

(An Efficient Query Transformation for Multidimensional Data Views on Relational Databases)

신 성 현 † 김 진 호 †† 문 양 세 †††
(Sunghyun Shin) (Jinho Kim) (Yangsae Moon)

요 약 온라인 분석 처리(OLAP, On-Line Analytical Processing)에서는 다양한 분석을 효과적으로 처리하기 위해, 다차원 구조의 데이타를 열에 차원 애트리뷰트의 값이 표시되는 넓은 형태의 수평 테이블로 표현한다. 관계형 테이블들은 보통 애트리뷰트의 개수에 제한이 있으므로 이러한 수평 테이블을 직접 저장하기 어렵고, 저장하더라도 많은 수의 널(null) 값을 갖는 희박(sparse) 테이블이 되기 쉽다. 따라서 관계 데이타베이스에서는 이러한 수평 테이블을 차원의 이름을 열로 갖는 수직 테이블로 바꾸어 저장할 수 있다. 이렇게 할 경우, 수평 뷰 테이블에 대한 OLAP 질의를 저장된 수직 테이블에 대한 질의로 변환하는 작업이 필요하다. 본 논문에서는 전통적인 관계 대수 연산자들 외에 최근 DBMS 버전들이 제공하는 PIVOT 연산자를 사용하여 수평적인 뷰의 질의를 효율적으로 변환하는 방법을 제안하였다. 이를 위해 PIVOT 연산과 동등한 관계 대수식을 만들고 이를 증명하였으며, 이 PIVOT 연산을 사용하여 수평적인 테이블에 대한 질의를 수직적인 테이블에 대한 질의로 변환하는 방법을 제시하였다. 또한 실험을 통해 제안한 변환 방법이 기존 방법에 비해 성능이 더욱 우수함을 보였다.

키워드 : OLAP; 다차원 데이타; PIVOT; 수평 테이블; 질의 변환

Abstract In order to provide various business analysis methods, OLAP(On-Line Analytical Processing) systems represent their data with multidimensional structures. These multidimensional data are often delivered to users in the horizontal format of tables whose columns are corresponding to values of dimension attributes. Since the horizontal tables may have a large number of columns, they cannot be stored directly in relational database systems. Furthermore, the tables are likely to have many null values (i.e., sparse tables). In order to manage the horizontal tables efficiently, we can store them as the vertical format of tables which has dimension attribute names as their columns thus transforms the columns of horizontal tables into rows. In this way, every queries for horizontal tables have to be transformed into those for vertical tables. This paper proposed a technique for transforming horizontal table queries into vertical table ones by utilizing not only traditional relational algebraic operators but also the PIVOT operator which recent DBMS versions are providing. For achieving this goal, we designed a relational algebraic expression equivalent to the PIVOT operator and we formally proved their equivalence. Then, we developed a transformation technique for horizontal table queries using the PIVOT operator. We also performed experiments to analyze the performance of the proposed method. From the experimental results, we revealed that the proposed method has better performance than existing methods.

Key words : OLAP, Multidimensional data, PIVOT, horizontal tables, query transformations

· 본 연구는 첨단정보기술연구센터를 통하여 과학기술부/한국과학재단의 지원과 2003년도 강원대학교 연구년 교수 연구비의 지원을 받았음

† 학생회원 : 강원대학교 컴퓨터학부
shshin@kangwon.ac.kr
†† 종신회원 : 강원대학교 컴퓨터학부 교수
jhkim@kangwon.ac.kr
††† 정 회 원 : 강원대학교 컴퓨터학부 교수
ysmoon@kangwon.ac.kr
(Corresponding author임)

논문접수 : 2006년 4월 12일
심사완료 : 2006년 10월 12일

1. 서 론

온라인 분석 처리(OLAP, On-Line Analytical Processing)은 데이타 웨어하우스에 저장된 대용량의 데이타로부터 다양한 분석 정보를 추출하기 위한 다차원 데이타 분석을 지원하는 기법이다. 이러한 OLAP은 다차원 데이타를 저장하는 방법에 따라, ROLAP(Relational

OLAP), MOLAP(Multidimensional OLAP)방식으로 나누어진다[1,2]. 여기서, ROLAP은 관계형 데이타베이스를 기반으로 다차원 데이타를 저장하는 방식이고, MOLAP은 다차원 데이타 구조를 직접 지원하는 다차원 시스템을 이용하는 방식이다. 이에 따라, OLAP에서는 이러한 다차원 데이타 분석을 편리하고 효율적으로 분석할 수 있도록 다양한 차원에 대해 측정 값을 유지할 수 있도록 데이타를 여러 형태로 표현하고 저장한다[3]. 이러한 데이타를 사용자들에게 편리하게 보여주는 저장 및 표현 방법 중의 하나로 분석 기분이 되는 차원 애트리뷰트들의 값을 행과 열의 색인 값으로 하는 다차원 데이타 구조 형태로 이차원 배열(또는 테이블) 형태를 많이 사용한다[3,4]. 즉, 분석의 기준이 되는 여러 차원(dimension)들과 분석할 내용에 해당하는 측정(measure)값들로 구성되며, 다차원으로 표현될 수 있다. 예를 들어, 측정 값 sales에 대한 차원 애트리뷰트는 product, location, time 차원들이다. 이는 Microsoft Excel의 피벗 테이블이나 통계 데이타 표현에 많이 사용되는 교차 테이블(cross table)과 비슷한 형태인데, 테이블의 열 이름으로 (차원 애트리뷰트 이름이 아닌) 차원 애트리뷰트 값이 표시되며 따라서 많은 수의 열을 갖는 수평적인 스키마 구조를 갖는다[4-7].

현재 널리 사용되고 있는 관계 데이타베이스 시스템에서는 데이타를 행과 열의 간단한 테이블로 표현하는데, 이들 테이블들은 대개 제한된 개수의 열로 구성된 스키마를 가지며 많은 수의 행(즉, 튜플)을 저장하는 수직적인 테이블을 주로 사용하고 있다. 예를 들면, 현재 대표적인 상용 관계 DBMS인 Microsoft SQL Server 2005와 Oracle 9i 등은 한 테이블이 최대 1,024개의 열(즉, 애트리뷰트)을 갖도록 제한돼 있다[8,9]. 따라서 이러한 관계 데이타베이스 시스템에서는 OLAP에서 널리 사용하는 수평적인 형태의 테이블을 직접적으로 정의하고 이를 사용자에게 직접 제공하기 어렵다[4,7]. 또한 이러한 수평적인 테이블은 데이타가 널(null) 값을 많이 포함하는 희박(sparse) 테이블이 될 수 있어 저장 공간을 낭비하는 문제가 있다[4,10].

이러한 수평적인 테이블 구조의 문제점을 해결하기 위한 효과적인 방법 중의 하나는, 테이블의 스키마로 (차원 애트리뷰트 값이 아니라) 차원 애트리뷰트들의 이름을 갖고 차원 값과 측정 값을 행(튜플)으로 표현하는 수직적인(vertical) 테이블 형태로 표현하는 것이다[7,11]. 예를 들면 다음 그림 1에서 왼쪽에 있는 테이블은 각 판매 매장(*PID*)에서 상품(*product*)별로 판매한 금액을 보여주는 수평적인 형태의 테이블(WT of Product)을 보여주고 있으며, 그 오른쪽 테이블은 이를 수직적인 테이블(NT of Product) 형태로 표현한 것이다. 또한 아래에 있는 테이블들은 각 판매 매장(*PID*)별로 판매한 상품들의 제조 국가별 판매액을 표현한 수평적인 테이블(WT of Manufacturer)과 수직적인 테이블(NT of Manufacturer)을 보여주고 있다.

위 그림에서 보는 바와 같이, 수평적인 테이블 NT of Product는 매장과 상품별 판매량을 서로 잘 비교할 수 있는 편리한 구조이며, 다차원 분석을 위한 편리한 사용자 뷰를 제공할 수 있다. 하지만 상품 종류가 많을 경우 수천개 또는 수만개 이상의 많은 열을 가질 수 있으며, 이는 관계 데이타베이스에서 직접적으로 저장하는 것이 불가능하다. 하지만 수직적인 테이블은 이를 (*PID, product, price*)의 세 애트리뷰트로 표현하기 때문에 제한된 수의 애트리뷰트를 갖는 테이블 구조로 표현이 가능하다. 또한 수평적인 테이블에서 실제로 값이 존재하는 셀에 대해 *PID, product, price* 값을 갖는 행만을 수직적인 테이블에서 표현한다. 따라서 수평적인 테이블

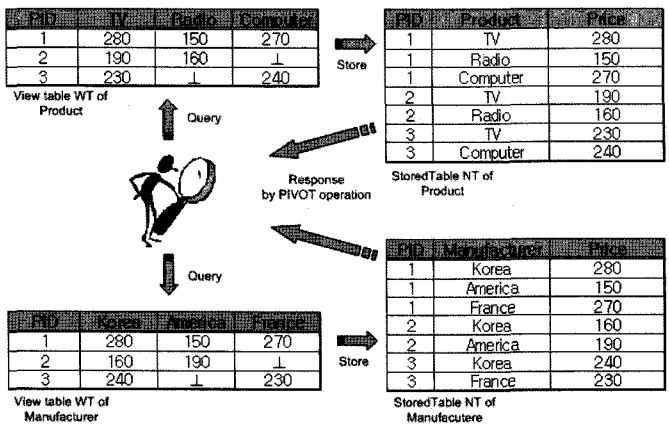


그림 1 뷰 테이블 *WT*와 PIVOT 연산에 의한 저장 테이블 *NT*

에 많은 널 값이 존재할 경우, 수직적인 테이블 형태로 표현할 경우 저장 공간을 더 줄일 수도 있다.

따라서 사용자들에게는 편리한 수평적인 테이블을 가상적인 뷰로 제공하며, 실제로는 수직적인 테이블 형태로 저장하는 것이 관계 데이타베이스 시스템에서 더욱 효율적이고 실현 가능한 방법이다. 이렇게 할 경우, 사용자는 수평적인 테이블을 기반으로 질의를 하게 되므로 해당 질의를 수직적인 저장 테이블에 대한 질의로 변환하여 처리하는 방법이 필요하다. 이러한 방법의 일환으로, [7]에서는 수평적인 뷰 스킴을 수직적인 테이블로 변환한 후, 수평적인 테이블에 대한 관계 대수 연산을 수직적인 테이블에 대한 관계 대수 연산으로 변환하는 방법을 제안하였다. 또한 [12]에서는 테이블의 행과 열을 서로 바꾸는 *PIVOT/UNPIVOT* 연산을 제안하고 이를 다른 연산들과 함께 사용할 때 질의를 최적화하는 방법을 소개하였다. [13]에서는 다중 *PIVOT* 연산의 처리를 위한 *GPIVOT* 연산에 대해 소개하였다. 또한 최근 상용 데이타베이스 시스템들은 피벗 테이블과 같은 수평적인 형태의 테이블을 생성할 수 있도록 *PIVOT/UNPIVOT* 연산을 질의문에 제공하거나 이와 유사한 기능을 추가하고 있다[8,9].

본 연구에서는 다차원 테이블을 관계 데이타베이스에 효율적으로 저장하고 관리하기 위해 수직적인 테이블로 저장하고, 수평적인 테이블에 대한 사용자들의 질의를 수직적인 테이블로 효율적으로 변환하여 처리하는 방법을 제시한다. 이렇게 함으로써 사용자들에게는 다차원 분석에 편리한 뷰를 제공하면서, 관계 데이타베이스 기능을 그대로 활용하여 다차원 데이타를 제공할 수 있다. 수평적인 사용자 뷰 테이블에 대한 관계 대수 연산들(즉, 프로젝트, 셀렉트, 합집합, 차집합, 카티션 곱)을 최근 데이타베이스 시스템에서 제공하는 *PIVOT* 연산을 사용하여 수직적인 테이블에 대한 관계 대수 연산으로 변환하는 방법을 제공하였으며, 이 변환 방법이 정확함을 증명하였다. 또한 실험을 통하여, 제시한 변환 방법과 기존의 관계 대수 연산만을 활용하는 변환 방법과 서로 비교.평가하였으며, 제안된 방법이 성능을 더욱 향상시킬 수 있음을 보였다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구와 관련된 기존 연구들에 대해 소개하고, 3장에서는 사용자 뷰 테이블과 저장 테이블을 소개하고 질의 처리에 대한 관계 대수 표현과 *PIVOT* 연산을 정의한다. 4장에서는 사용자 뷰의 관계 대수 연산들에 대한 질의 변환 방법을 소개한다. 5장에서는 실험을 통해 제안한 방법과 기존의 방법을 비교한 성능 평가를 설명하고, 마지막으로 6장에서는 결론을 맺는다.

2. 관련 연구

OLAP에서 사용하는 다차원 데이타는 여러 차원 애트리뷰트들과 이에 대응하는 측정값들로 구성된다. 이 다차원 데이타를 사용자들에게 자연스럽게 표현하는 방법은 Excel의 피벗 테이블 또는 통계 분야의 교차 테이블 형태로 표현하는 것이다[4,5]. 이런 표현 방법은 테이블의 스키마(즉, 열의 이름)에 차원 애트리뷰트의 값이 오게 되므로 많은 수의 열을 갖는 수평적인 형태의 테이블이 된다. 그러나 일반적인 관계 데이타베이스는 테이블 스키마로 특정한 개수의 애트리뷰트 이름들을 가지며, 이 애트리뷰트들의 값들로 구성된 행을 많이 갖는 일종적의 수직적인 형태의 테이블을 사용한다. 따라서 관계 데이타베이스를 이용하여 다차원 데이타를 저장하기 위해서는, 다차원 데이타를 수직적인 형태로 변환하여 관계 데이타베이스 테이블에 저장해야 한다. 스타 스키마 등에 의해 다차원 데이타를 표현하는 많은 ROLAP (Relational OLAP) 시스템들은 이런 수직적인 테이블을 사용하는데, 이 구조는 사용자들에게 친숙한 뷰를 제공할 수 없다.

사용자들에게 다차원 데이타에 대한 편리한 뷰를 제공하기 위해, 다차원 데이타를 스프레드시트 또는 수평적인 테이블 형태로 제공하는 기법에 대해 여러 연구가 진행되었다. [5,6]에서는 SQL 문을 확장하여 관계 데이타베이스에서 스프레드시트 형태의 테이블을 정의하고 조작할 수 있도록 하였다. 이렇게 함으로써 사용자들이 OLAP 분석 기능을 좀 더 편리하게 할 수 있도록 하였다. 또한 [12]에서는 행과 열을 역할을 서로 바꾸어 수직적인 테이블에서 수평적인 테이블을, 또는 수평적인 테이블에서 수직적인 테이블을 생성하는 *PIVOT/UNPIVOT* 새로운 연산자를 추가하고, 이를 다른 대수 연산자들과 함께 사용할 때 최적화하는 방법에 대해 연구하였다. 이 연산을 확장하여, [13]에서는 좀 더 일반적인 피벗 기능을 갖는 *GPIVOT* 연산자를 정의하였다. 이 *PIVOT* 연산은 수직적인 테이블에서 사용자들에게 편리한 수평적인 뷰를 유도할 수 있는 장점이 있어서 MS SQL Server2005 등과 같은 실제 상용 DBMS에서 실제로 구현되었다. 하지만 이렇게 생성된 수평적인 테이블은 많은 수(수천 또는 수만 개)의 열을 가질 수 있어서, 테이블의 열의 개수에 제한이 있는 관계 데이타베이스에 저장하기 어려울 수도 있다. (대부분의 상용 DBMS에서는 각 테이블이 최대 1,024개 이하의 열을 가질 수 있다[8,9].) 또한 수평적인 테이블의 각 행은 모든 차원 애트리뷰트 값들마다 하나의 값을 갖는 길이가 긴 구조를 갖는다. 따라서 실제로 값이 존재하지 않는 경우가 많기 때문에 많은 수의 널 값을 가질 수 있어 저장 공간을 낭비할 수 있다[4,10].

이러한 희박 수평적 테이블을 효율적으로 저장하는 방법은 수직적인 테이블을 사용하여 실제로 존재하는 값에 대해서 하나의 행을 갖도록 표현하는 방법이다 [7,11]. 또한 사용자들에게 편리한 수평적인 뷰를 제공하고, 이 수평적인 테이블에 대한 질의들을 수직적인 테이블에 대한 질의로 변환하여 처리한다. 이렇게 하면 사용자들에게 저장 테이블에 대한 구조를 숨기고 편리한 수평적인 뷰를 사용하게 할 수 있다. 많은 열을 갖는 테이블을 효율적으로 저장하고 처리하기 위해, [11]에서는 네 가지 재구성 연산들(*unfold, fold, split, unite*)을 제안하고 이를 관계 대수로 대응시키는 방법을 제시하였다. 이 연산들 중에서 *unfold*와 *fold* 연산은 수평적인 테이블을 수직적인 테이블로 행과 열을 변경하는 연산이다. 이러한 연구를 기반으로, [7]에서는 관계 대수를 이용하여 *unfold*와 *fold* 연산을 정의하고, 수평적인 테이블에 대한 관계 대수 연산들을 수직적인 테이블에 대한 대수 연산으로 변환하는 방법을 제안하였다. 하지만 이 변환 방법은 전통적인 관계 대수 연산만을 사용하여 변환하기 때문에 변환이 상당히 복잡하다. 현재 상용 DBMS에서는 OLAP 응용을 효율적으로 처리하기 위한 여러 고급 기능을 가지고 있으며, *PIVOT*과 같은 연산을 기본 연산으로 제공하고 있다[8,9]. 따라서 최신 DBMS의 이러한 기능을 이용하여 좀 더 효율적이고 간결하게 변환하는 방법에 대한 연구가 필요하다.

3. 뷰 테이블과 저장 테이블

현재 관계 데이타베이스 시스템에서는 테이블의 최대 컬럼 수(MSSQL 또는 Oracle의 경우 1024개)에 제한이 있다. 따라서 이 논문에서는 다차원 데이타를 효과적으로 관리하기 위해 사용자들에게 수평적인 뷰를 제공하고, 실제의 저장은 수직적인 테이블로 저장한다. 이때 수평적인 뷰에 대한 질의를 수직적인 테이블에 대한 질의로 변환한다. 이 장에서는 이러한 수평적인 사용자 뷰와 수직적인 저장 테이블에 대한 구조와 이들 간의 상호 관계에 대해 설명한다.

3.1 뷰 테이블과 저장 테이블의 스키마

본 논문에서 수평적인 뷰 테이블(*WT*, wide table)과 수직적인 저장 테이블(*NT*, narrow table)에서 질의 처리 변환 방법을 논하기 위해 다음 표 1과 같이 주요 표기법을 사용한다.

각 테이블의 스키마는 표 1을 이용하여 다음과 같이 정의한다. 수평적인 뷰 테이블 WT는 스키마 $(Oid, A_1, A_2, ..., A_n)$을 나타내며, 여기서 Oid는 식별자이고 $A_1, A_2, ..., A_n$는 열의 이름으로 표현한다. 그리고 수직적인 저장 테이블 NT는 스키마(Oid, A, M)로 나타내며, 여기서 Oid는 식별자이고 A,M은 각각 열의 이름으로 구

표 1 주요 표기법

Symbols	Definitions
WT	수평 형태의 뷰 테이블
NT	수직 형태의 저장 테이블
Oid	튜플 식별자
A, M	저장 테이블 NT 내의 애트리뷰트들
A_i	저장 테이블 NT의 A 애트리뷰트에 올 수 있는 값들 $(1 \le i \le n)$ 뷰 테이블 WT의 애트리뷰트 이름들 $(1 \le i \le n)$
M_i	저장 테이블 NT의 M 애트리뷰트에 올 수 있는 값들 $(1 \le i \le n)$

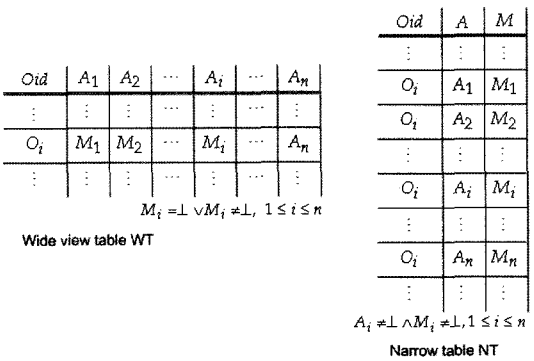


그림 2 테이블 WT와 테이블 NT의 일반화 표기

성된다. 그림 2에서는 이러한 뷰 테이블 WT와 저장 테이블 NT의 구조를 표현한다.

뷰 테이블 WT와 저장 테이블 NT는 다른 형태의 테이블 구조로 구성되었지만, 각 테이블 내에 저장되어 있는 데이타 값은 동일하다. 뷰 테이블 WT에서 저장 테이블 NT로 변환하여 저장할 때, 뷰 테이블 WT의 열의 이름 A_i와 데이타 값 M_i는 저장 테이블 NT에서 열 A의 데이타 값 A_i와 열 M의 데이타 값 M_i로 각각 변환하여 표현할 수 있다. 뷰 테이블 WT의 특정한 데이타 값 M_i은 널 값이거나(=⊥), 널 값이 아닐($\neq \perp$)수 있으나, 저장 테이블 NT에서는 데이타 값 M_i가 널 값일 경우에는 저장하지 않는다. 따라서, 그림 2와 같이 뷰 테이블 WT의 저장공간은 n개의 행과 m개의 열을 포함하므로 $n \times m$에 비례하고, 저장 테이블 NT의 저장공간은 $((n \times m)-(n \times m \times \alpha))$에 비례한다. 여기서 $\alpha(1 \le \alpha \le 1)$는 널 값의 비율이다.

3.2 질의 처리 변환에 대한 대수적 표현

사용자들에게 수평적인 뷰를 제공하기 위해서는 이 뷰에 대한 질의를 실제로 저장된 수직적인 테이블에 대해 수행할 수 있도록 변환하는 것이 필요하다. [13]에서는 기본적인 관계 대수 연산을 사용하여 *PIVOT* 연산을 정의하였다. 이들 연구를 기반으로 본 연구에서는 수직적인 저장 테이블에서 기준이 되는 행과 열을 추출하

고 수평적인 뷰 테이블로 좀 더 효율적으로 변환할 수 있도록 완전 외부 조인(⟗)을 사용하여 변환하는 대수식을 정의한다. 또한 본 연구에서는 이 관계 대수 연산을 좀 더 간결하게 표현할 수 있도록 *PIVOT* 연산을 사용한다.

정의 1[13]: 수평적인 뷰 테이블 *WT*와 이에 대한 저장 테이블 *NT*가 있다고 하자. 저장 테이블 *NT*에서 뷰 테이블 *WT*로 변환하기 위해 저장 테이블 *NT*에서 우선 기준이 되는 조건의 열 A의 값이 A_i인 데이타 값을 추출하고 열들의 목록 *Oid,M*을 지정한다. 다음으로 수평적인 뷰 테이블로 표현하기 위해 완전 외부 조인(⟗) 연산을 적용하여 아래 식 (1)과 같이 정의할 수 있다.

$$\mathrm{WT} = \left[\text{⟗}_{i=1}^{n} \pi_{Oid,M}(\sigma_{A='A_i'}(NT)) \right] \quad (1)$$

다음으로, 이 수평적인 뷰(*WT*)는 행과 열을 변경하는 연산인 *PIVOT* 연산을 이용할 경우 좀더 간결한 방식으로 저장 테이블(*NT*)로부터 유도할 수 있다. 정의 2는 정의 1에서 정의한 대수 연산을 *PIVOT* 연산을 사용하여 다시 정의한다.

정의 2: 행과 열을 변경하는 조작 연산인 *PIVOT* 연산을 이용한다고 하자. 정의 1의 식 (1)에서 뷰 테이블로 표현하기 위한 완전 외부 조인은 *PIVOT* 연산으로 대신할 수 있다. 식 (1)에서의 완전 외부 조인은 식 (2)에서와 같이 *PIVOT* 연산으로 대신 사용할 경우, 기준이 되는 조건의 열 A의 값이 전체 $A_i(1 \le i \le n)$인 데이타 값을 추출하고 열들의 목록 *Oid,M*을 지정한 후, 완전 외부 조인 연산 대신에 행과 열을 변경하는 *PIVOT* 연산을 사용하여 아래 식 (2)와 같이 정의할 수 있다.

$$WT \xrightarrow{transformation} \mathrm{PIVOT}\left[\pi_{Oid,M}(\sigma_{\vee_{i=1}^{n} A='A_i'}(NT)) \right] \quad (2)$$

예제 1: 그림 3은 저장 테이블 *NT*에서 뷰 테이블 *WT*로 변환하기 위해 완전 외부 조인 연산의 이용(정의 1)하는 것과 *PIVOT* 연산의 이용(정의 2)하는 것에 대한 예를 보인다. 그림 3의 (a)는 저장 테이블 *NT*의 기준이 되는 열 A에서 $1 \le i \le n$인 데이타 값 A_i를 추출하고 완전 외부 조인 연산을 이용하여 뷰 테이블 *WT*로 처리 변환한다. 여기에서 음영으로 처리된 부분은 완전 외부 조인 연산에 의해 뷰 테이블 *WT*에서 각 열의 데이타 값으로 처리된다. 마찬가지로 그림 3의 (b)에서도 *PIVOT* 연산을 적용하여 뷰 테이블 *WT*로 처리하여 변환한 과정을 나타낸다. 우선 해당 질의에 대한 열의 이름 A의 데이타 값 A_i을 $1 \le i \le n$순으로 추출하고 후에 행과 열을 변경하기 위해 *PIVOT* 연산을 적용하여 뷰 테이블 *WT*로 처리하는 과정을 예로 보여준다.

4. 질의 변환 처리

이 장에서는 수평적인 뷰에 대한 질의가 들어올 경우 이를 수직적인 저장 테이블에 대한 질의로 바꿔 수행한 후 수평적인 뷰에 대한 결과를 생성하도록 하는 질의 변환 방법에 대해 설명한다. 사용자의 질의는 관계 대수

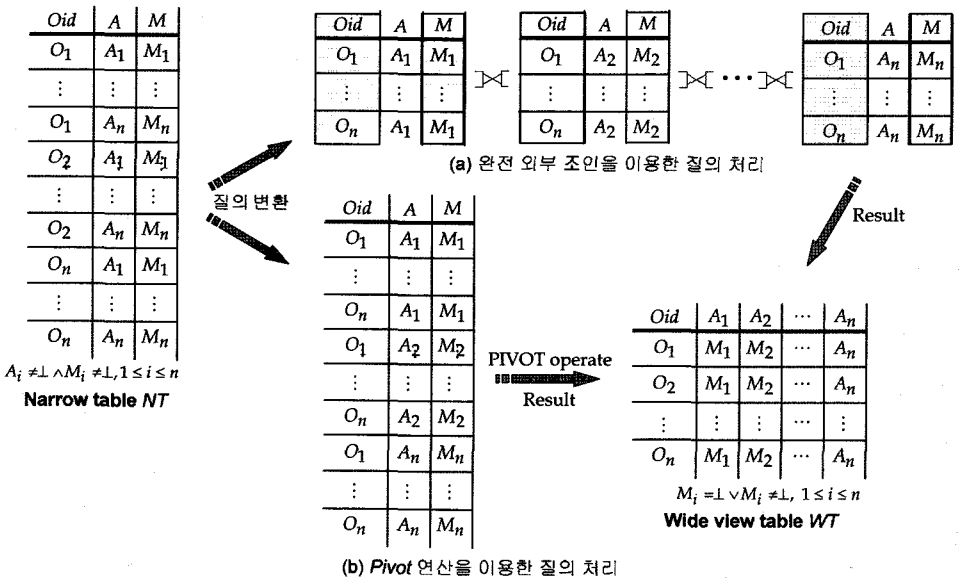


그림 3 완전 외부 조인과 *PIVOT* 연산을 이용한 질의 변환

연산으로 표현한다고 가정하였으며, 가장 기본적인 관계 대수 연산인 셀렉트, 프로젝트, 합집합, 차집합, 카티션 곱의 다섯 가지 연산에 대해 질의를 변환하는 방법을 제안한다. 수평적인 뷰에 대한 이들 다섯 가지 연산 각각에 대해 수직적인 저장 테이블을 이용하여 처리하는 변환 규칙과 정리를 제시하고 각 변환이 서로 동일함을 증명한다.

4.1 프로젝트 연산

프로젝트 연산은 테이블의 특정 열들의 부분집합으로 구성된 새로운 테이블을 구하기 위한 연산이며, π기호로 표기한다. 연산 $\pi_{A_1,A_2,...,A_k}(WT)$의 값은 뷰 테이블 WT의 속성 중에 열의 목록 $A_1, A_2, ..., A_k$만을 추출한다. 연산에 대한 결과 테이블의 스키마는 속성의 집합 $A_1, A_2, ..., A_k$이 된다. 이 프로젝트 연산은 아래의 식 (4)와 같이 저장 테이블 NT에 대한 연산으로 변환될 수 있다.

$$\pi_{Oid,\vee_{i=1}^{k}A_i}(WT) = \left[\bowtie_{i=1}^{n}\pi_{Oid,M}(\sigma_{A='A_i'}(NT))\right] \quad (3)$$

$$\xrightarrow{Transformation} \text{PIVOT}\left[\pi_{Oid,M}(\sigma_{\vee_{i=1}^{k}A='A_i'}(NT))\right] \quad (4)$$

정리 1: 질의에 포함되는 열의 목록 $A_1, A_2, ..., A_k$는 뷰 테이블 WT의 열에 속하는 애트리뷰트들이고 $k \geq 1$일 때, 다음 식의 조건이 성립한다.

$$\pi_{Oid,A_1,A_2,...,A_k}(WT) = \left[\bowtie_{i=1}^{k}\pi_{Oid,M}(\sigma_{A='A_i'}(NT))\right] \quad (5)$$

증명: 수학적 귀납법을 이용하여 열의 수가 k개일 때 식 (5)가 성립함을 증명한다.

기본 단계: 우선 열의 수가 1개일 경우, i=1,2,...,k에 대해 임의의 열을 A_i라 하자. $P(1)$은 다음과 같이 정의할 수 있다.

$$P(1): \pi_{Oid,A_i}(WT) = [\pi_{Oid,M}(\sigma_{A='A_i'}(NT))]$$

$P(1)$의 결과는 $P(1)$에서 좌변의 식과 우변의 식의 결과는 동일하다.

귀납가정: $k \geq 1$인 모든 k에 대하여, 열의 수가 k개일 경우 $P(k)$를 참이라고 가정하자.

귀납단계: 열 수가 k개의 부분 집합에 대해 등식이 성립하면 열의 수가 k+1개의 집합에 대해서도 등식이 성립함을 보여야 한다. $P(k)$가 정의된다고 가정하면 $P(k+1)$에 대해서도 정의할 수 있다.

$$\text{P(k)}: \pi_{Oid,A_1,A_2,...,A_k}(WT) = \left[\bowtie_{i=1}^{k}\pi_{Oid,M}(\sigma_{A='A_i'}(NT))\right]$$

여기서, 귀납적 가설 $P(k)$가 성립한다고 가정하면 다음과 같다.

$$P(k+1): \pi_{Oid,A_1,A_2,...,A_k,A_{k+1}}(WT)$$

$$= [\pi_{Oid,A_1,A_2,...,A_k}(WT)] \bowtie [\pi_{Oid,A_{k+1}}(WT)]$$

$$= \left[\bowtie_{i=1}^{k}\pi_{Oid,A_i}(WT)\right] \bowtie [\pi_{Oid,A_{k+1}}(WT)]$$

$$= [\pi_{Oid,A_1}(WT)] \bowtie [\pi_{Oid,A_2}(WT)]$$

$$\bowtie \cdots \bowtie [\pi_{Oid,A_k}(WT)]] \bowtie [\pi_{Oid,A_{k+1}}(WT)]$$

$$= [\pi_{Oid,M}(\sigma_{A='A_1'}(NT))] \bowtie [\pi_{Oid,M}(\sigma_{A='A_2'}(NT))]$$

$$\bowtie \ldots \bowtie [\pi_{Oid,M}(\sigma_{A='A_k'}(NT))] \bowtie [\pi_{Oid,M}(\sigma_{A='A_{k+1}'}(NT))]$$

$$= \left[\bowtie_{i=1}^{k+1}\pi_{Oid,M}(\sigma_{A='A_i'}(NT))\right]$$

$P(k+1)$도 참이 성립한다. 따라서 모든 애트리뷰트들에 대해, 수평적인 뷰 테이블 WT에 적용한 프로젝트 연산은 수직적인 테이블 NT에 대해 변환한 연산과 동일함을 증명하였다. □

예제 2: 프로젝트 연산에 대해 예제를 나타내면 다음 식과 같다.

$$\pi_{Oid,A_1,A_2,...,A_k}(WT) = \left[\bowtie_{i=1}^{k}\pi_{Oid,M}(\sigma_{A='A_i'}(NT))\right]$$

프로젝트 연산에 대하여 수평적인 뷰 테이블 WT와 수직적인 테이블 NT에서 적용한 결과를 그림 4에서 보인다. 그림 4는 저장 테이블 NT에서 완전 외부 조인 연산을 이용한 프로젝트 연산인 정리 1에 대한 예를 보이고 있다.

뷰 테이블 WT에 대한 질의를 저장 테이블 NT에 대한 질의로 변환하여 처리하는 과정을 정의 1에서 제시하였다. 정의 1은 저장 테이블 NT에서 해당 튜플을 추출하고 열 O,M을 기반으로 완전 외부 조인 연산을 적용한 것이며, 이는 $PIVOT$ 연산으로 해결할 수 있다. 따라서 정리 1과 $PIVOT$ 연산의 정의 2에 의해서 다음의 변환 규칙 1이 성립한다.

변환 규칙 1: 뷰 테이블 WT에 대한 프로젝트 연산에서는 명시된 열들의 목록이 열거된 순서에 의해 열들의 집합 $Oid,\vee_{i=1}^{k}A_i$로 나타낼 수 있으며, 정리 1에서 저장 테이블 NT 내의 각 열 목록들은 완전 외부 조인 연산을 이용하는 대신에 정리 2를 이용하여 $PIVOT$ 연산을 변환하여 표현할 수 있다. 즉, 다음과 같이 식 (6)이 성립한다.

$$\pi_{Oid,\vee_{i=1}^{k}A_i}(WT)$$

$$\xrightarrow{Transformation} \text{PIVOT}\left[\pi_{Oid,M}(\sigma_{\vee_{i=1}^{k}A='A_i'}(NT))\right] \quad (6)$$

4.2 셀렉트 연산

셀렉트 연산은 테이블 내의 특정 튜플을 조회하는 명령문으로, σ기호로 표현한다. 뷰 테이블 WT에 적용된 셀렉트 연산은 뷰 테이블 WT에 포함된 튜플의 부분 집합에 대해 새로운 테이블을 생성한다. 이 연산에 대하여 데이타를 추출하기 위해 사용되는 셀렉트 연산의 조건은 술어(predicate)라고 하며, 술어 $A_i\theta'M_i'$로 표기한다.

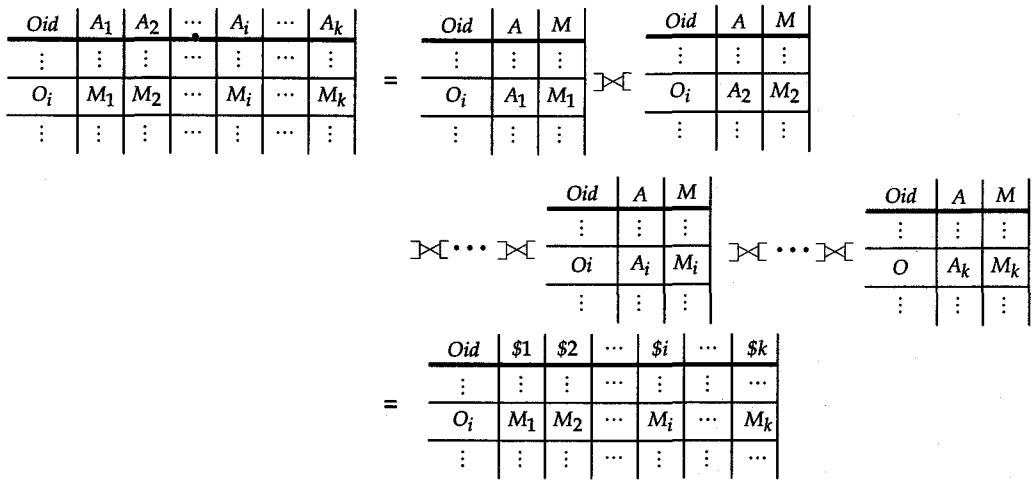


그림 4 프로젝트 연산에 대한 질의 변환 예제

여기서 A_i는 열 이름을 나타내며 M_i는 상수를 나타낸다. 그리고 θ는 비교 연산자들 $\{=,\neq,\leq,<,\geq,>\}$ 중의 하나의 의미를 갖는다. 이 셀렉트 연산은 다음과 같이 저장 테이블에 대한 연산으로 표현될 수 있다.

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

$$= [\cap_{i=1}^{k}\pi_{Oid}(\sigma_{A='A_i'\wedge M\theta'M_i'}(NT))] \bowtie [\bowtie_{j=1}^{n}\pi_{Oid,M}(\sigma_{A='A_j'}(NT))] \quad (7)$$

$$\xrightarrow{Transformation} \text{PIVOT}\left[\sigma_{Oid\ \text{in}\ (\cup_{i=1}^{k}\pi_{Oid}(\sigma_{A='A_i'\wedge M\theta'M_i'})(NT))}\right] \quad (8)$$

정리 2: 셀렉트 연산의 술어 $A_i\theta'M_i'$는 특정 튜플에 대한 검색 조건이며, 술어의 수에 대하여 $k \geq 1$일 때, 다음 식의 조건이 성립한다.

$$\sigma_{\wedge_{i=1}^{k}(A_i\,\theta\,'M_i')}(WT) = [\cap_{i=1}^{k}\pi_{Oid}(\sigma_{A='A_i'\wedge M\theta'M_i'}(NT))] \bowtie [\bowtie_{j=1}^{n}\pi_{Oid,M}(\sigma_{A='A_j'}(NT))] \quad (9)$$

증명: 수학적 귀납법을 이용하여 술어의 수가 k개일 때 식 (9)가 성립함을 증명한다.

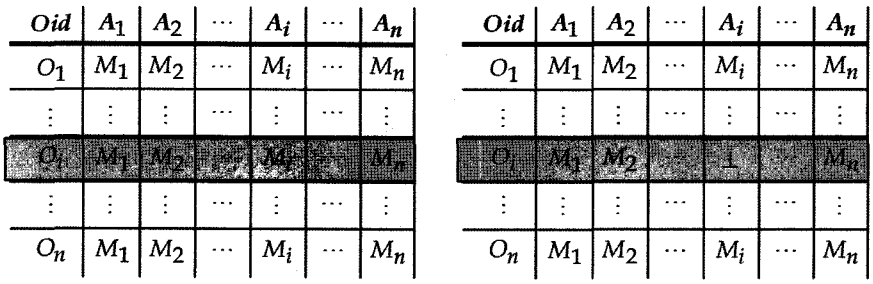
기본 단계: 셀렉트 연산의 조건에 대해 술어의 수가 1개일 경우, 먼저 $i = 1,2,...,k$에 대해 임의의 술어 $A_i\theta'M_i'$라 하자. $P(1)$은 다음과 같이 정의할 수 있다.

P(1): $\sigma_{A_i\theta'M_i'}(WT)$

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

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

그림 5에서는 뷰 테이블 WT에서 셀렉트 연산에 대해 결과인 특정한 O_i에 대한 튜플을 나타내고 있다. 여기서, 그림 5(a)는 특정한 O_i가 해당하는 튜플 내에 열 A_i의 값 M_i이 널 값을 포함하지 않는 경우와 그림



(a) 애트리뷰트 M_i가 널 값이 아닐 경우 (b) 애트리뷰트 M_i가 널 값일 경우

그림 5 뷰 테이블 WT에서의 셀렉트 연산의 결과

5(b)는 열 A_i의 값 M_i이 널 값을 포함할 경우를 나타내고 있다.

그림 5(a)의 경우 저장 테이블 NT 내에서는 열 M의 값 M_i는 널 값이 아닌 실제 데이타를 튜플 (Oid, A, M)로 저장을 하지만, 그림 5(b)의 경우 열 M의 값 M_i는 널 값을 포함하므로 저장 테이블 NT에서 튜플은 저장하지 않는다. 따라서 저장 테이블 NT에서 열 M의 데이타 값 M_i이 널 값을 포함할 경우에, 완전 외부 조인(full outer join) 연산(⟗로 표기)은 $NT_i ⟗ NT_j$에 대하여 상대 릴레이션에 관련된 튜플이 없더라도 왼쪽 테이블 NT_i과 오른쪽 테이블 NT_j의 모든 튜플을 결과에 포함시키고, 필요한 경우 널 값으로 채워 WT의 튜플과 같은 결과를 생성한다.

귀납가정: $k \geq 1$인 모든 k에 대하여, 술어의 수가 k개일 경우 $P(k)$를 참이라고 가정하자.

귀납단계: 술어의 수가 k개에 대해 등식이 성립하면 술어의 수가 $k+1$개에 대해서도 등식이 성립함을 보인다. 즉, $P(k)$가 정의된다고 가정하면 $P(k+1)$에 대해서도 정의할 수 있다.

$$P(k): \ {}^{\sigma}{}_{\wedge_{i=1}^{k}(A_i\theta' M_i')}(WT)$$

$$= [\cap_{i=1}^{k} {}^{\pi}{}_{Oid}({}^{\sigma}{}_{A='A_i' \wedge M\theta' M_i'}(NT))] ⟗$$

$$[⟗_{j=1}^{n} {}^{\pi}{}_{Oid,M}({}^{\sigma}{}_{A='A_j'}(NT))]$$

귀납적 가설 $P(k)$가 성립한다고 가정하면 다음과 같다.

$$P(k+1): \ {}^{\sigma}{}_{\wedge_{i=1}^{k+1}(A_i\theta' M_i')}(WT)$$

$$= {}^{\sigma}{}_{(\wedge_{i=1}^{k} A_i\theta' M_i') \wedge (A_{k+1}\theta' M_{k+1}')}(WT)$$

$$= {}^{\sigma}{}_{\wedge_{i=1}^{k}(A_i\theta' M_i')}(WT) \cap {}^{\sigma}{}_{(A_{k+1}\theta' M_{k+1}')}(WT)$$

$$= [\cap_{i=1}^{k} {}^{\pi}{}_{Oid}({}^{\sigma}{}_{A='A_i' \wedge M\theta' M_i'}(NT))] ⟗$$

$$[⟗_{j=1}^{n} {}^{\pi}{}_{Oid,M}({}^{\sigma}{}_{A='A_j'}(NT))]$$

$$\cap [{}^{\pi}{}_{Oid}({}^{\sigma}{}_{A='A_{k+1}' \wedge M\theta' M_{k+1}'}(NT))] ⟗$$

$$[⟗_{j=1}^{n} {}^{\pi}{}_{Oid,M}({}^{\sigma}{}_{A='A_j'}(NT))]$$

$$= [\cap_{i=1}^{k+1} {}^{\pi}{}_{Oid}({}^{\sigma}{}_{A='A_i' \wedge M\theta' M_i'}(NT))] ⟗$$

$$[⟗_{j=1}^{n} {}^{\pi}{}_{Oid,M}({}^{\sigma}{}_{A='A_j'}(NT))] \quad (10)$$

저장 테이블 NT에서 적용된 식은 보조정리 1에 의하여 분배 법칙이 성립되므로 아래와 같이 표현될 수 있다. 이는 귀납법에 의한 증명으로 $P(k+1)$이 참임을 보였다. □

수평적인 뷰 테이블 WT와 수직적인 저장 테이블 NT에서 적용한 셀렉트 연산이 동일함을 정리 2에서 제시하였다. 셀렉트 연산의 술어에 명시된 조건들은 여러 개의 절로 구성될 수 있으며 두 개 이상의 조건이 사용될 경우 부울 연산자를 이용하여 조건은 임의로 연결하여 나타낼 수 있다. 예를 들어, ${}^{\sigma}{}_{\vee_{i=1}^{k}(A_i\theta' M_i')}(WT)$인 경우, 정리 2에서 $\cap_{i=1}^{k}$을 $\cup_{i=1}^{k}$로 변경될 수 있다.

표 2 보조정리를 위한 주요 표기

Symbols	Definitions
R, S, T	스키마 R, S, T는 동일한 구조
$R(o,r)$	릴레이션 R의 튜플 (o,r)
$S(o,s)$	릴레이션 S의 튜플 (o,s)
$T(o,t)$	릴레이션 T의 튜플 (o,t)

보조정리 1: 표 2에 따라, 릴레이션 R, S, T는 $(R \cap S) ⟗ T$이라 가정하면 다음과 같이 성립한다.

$$(R ⟗ T) \cap (S ⟗ T) = (R \cap S) ⟗ T$$
$$where \ R \subseteq T, \ S \subseteq T \quad (11)$$

증명: 우선 식 (11)에서 $(o,r) \in R$이면 $\exists t(o,t) \in T$라 하면, $R ⟗ T$는 다음과 같이 표기할 수 있다.

$$(R ⟗ T) = \{(o, r, t) | (o,r) \in R \wedge (o,t) \in T\}$$
$$+\{(o,r,\perp) | (o,r) \in R \wedge (\perp = null)\}$$
$$= \{(o, r, t) | (o,r) \in R \wedge ((o,t) \in T \vee t = \perp)\} \quad (12)$$

식 (12)는 $R ⟗ T$의 경우, $(o,r) \in R$, $(o,t) \in T$이며 튜플 (o,r)와 (o,t)에서 대응되는 열 o와 결합한 튜플들의 결과는 (o,r,t)이다. 기호 $\perp$는 널 값을 표시하며, 만약 릴레이션 T에 관련된 튜플이 없으면 결과 릴레이션의 T 튜플들은 $(o,r,\perp)$와 같이 널 값으로 채워진다.

다음으로, 왼쪽 외부 조인 연산의 결과가 서로 동일함을 보이기 위해 $(o,q,t) \in (R ⟗ T) \cap (S ⟗ T)$일 때 $(o,q,t) \in (R \cap S) ⟗ T$ 임을 보인다.

$$(R ⟗ T) \cap (S ⟗ T)$$
$$= \{(o, q, t) | (o,q) \in R \wedge ((o,t) \in T \vee t = \perp)\} \cap$$
$$\{(o, q, t) | (o,q) \in S \wedge ((o,t) \in T \vee t = \perp)\}$$
(식 (12)의 정의에 의함)
$$= \{(o, q, t) | ((o,q) \in R \wedge (o,q) \in S) \wedge ((o,t) \in T \vee t = \perp)\}$$
(분배법칙과 멱등법칙 정의에 의함)
$$= \{(o, q, t) | ((o,q) \in (R \cap S)) \wedge ((o,t) \in T \vee t = \perp)\}$$
(교집합의 정의에 의함)
$$= (R \cap S) ⟗ T$$
(식 (12)의 정의에 의함)

여기서, $(o,q,t) \in (R ⟗ T) \cap (S ⟗ T)$ 라고 하면 교집합의 정의에 의해 $(o,q,t) \in (R ⟗ T) \wedge (o,q,t) \in (S ⟗ T)$이다. 이것은 식 (12)에 의해 $(o,q) \in R \wedge (o,t) \in T \wedge (o,s) \in S \wedge (o,t) \in T$가 되며 이로부터 분배법칙와 멱등법칙에 의해 $(o,q) \in R \wedge (o,s) \in S \wedge (o,t) \in T$가 되며 교집합의 정

의에 의해 $(o,q)\in(R\cap S)\wedge(o,t)\in T$가 된다. 여기서 교집합 $R\cap S$와 T는 (o,q)와 (o,t)에서 대응되는 튜플과 결합한 결과 (o,q,t)와, $R\cap S$와 관련된 튜플이 없는 결과 $(o,q,\perp)$에 의해 $(R\cap S)\rtimes T$임을 알 수 있다. 따라서 $(R\rtimes T)\cap(S\rtimes T)=(R\cap S)\rtimes T$이다.

예제 3: 그림 6에서는 수평적인 뷰 테이블 WT와 수직적인 저장 테이블 NT에서 적용한 셀렉트 연산의 결과가 동일함을 예제로 나타내고 있다. 수평적인 뷰 테이블 WT에서 특정 튜플을 검색하기 위한 술어 $\sigma_{A_i='M_i'\wedge A_j='M_j'}(WT)$에 대해 수직적인 저장 테이블 WT에서 적용한 동일한 연산은 식 (13)에서 예로 보인다.

$$\begin{aligned}&\sigma_{A_i='M_i'\wedge A_j='M_j'}(WT)\\ &= [[\pi_{Oid}(\sigma_{A='A_i'\wedge M='M_i'}(NT))]\\ &\quad\cap[\pi_{Oid}(\sigma_{A='A_j'\wedge M='M_j'}(NT))]]\\ &\quad\rtimes[\rtimes_{j=1}^{n}\pi_{Oid,M}(\sigma_{A='A_j'}(NT))]\end{aligned}\tag{13}$$

뷰 테이블 WT의 부분 집합의 행을 출력하기 위해 셀렉트 연산을 사용함으로써 특정 조건을 만족하는 데이타에 대한 검색이 저장 테이블 NT에서도 적용할 수 있도록 질의 변환에 대한 결과가 동일함을 정리 2에서 증명하였다. 이는 뷰 테이블 WT에서 셀렉트 연산의 조건은 $\sigma_{\wedge_{i=1}^{k}(A_i\theta'M_i')}(WT)$로 이를 저장 테이블 NT로 적용하기 위한 조건은 $[\cap_{i=1}^{k}\pi_{Oid}(\sigma_{A='A_i'\wedge M\theta'M_i'}(NT))]$로 변환할 수 있다. 그리고 나서, 완전 외부 조인 연산을 이용하여 모든 행의 데이타를 표기할 수 있다. 정의 2에서 사용한 완전 외부 조인 연산에 대한 부분은 $PIVOT$ 연산으로 대신할 수 있다.

변환 규칙 2: 검색될 데이타를 포함하는 뷰 테이블 WT에서 모든 데이타의 행을 검색하는 셀렉트 연산의 조건 $\wedge_{i=1}^{k}(A_i\theta'M_i')$로 나타낼 수 있으며 정리 2에서 사용되는 외부 조인 연산들에 대해 $PIVOT$ 연산으로 변환하여 표현할 수 있다. 즉, 다음 식 (14)가 성립한다.

$$\sigma_{\wedge_{i=1}^{k}(A_i\,\theta\,'M_i')}(WT)\xrightarrow{Transformation}PIVOT\left[\sigma_{Oid\ \text{in}\ (\cap_{i=1}^{k}\pi_{Oid}(\sigma_{A='A_i'\wedge M\theta'M_i'})(NT))}\right]\tag{14}$$

4.3 집합 연산

집합 연산자는 피연산자로서 두 개 이상의 테이블을 필요로 한다. 관계 데이타베이스의 테이블은 튜플들의 집합이기 때문에 기존의 집합 연산이 테이블에 적용된다. 이러한 연산자들은 합집합(union), 차집합(set difference), 카티션 곱(cartesian product)이 있다. 여기서

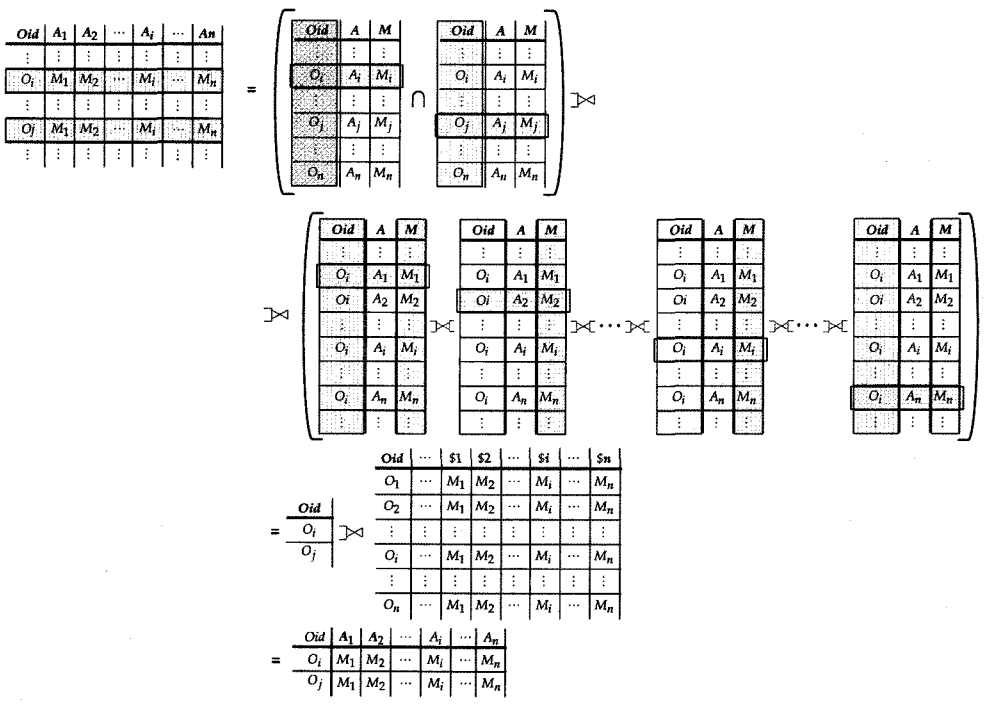


그림 6 셀렉트 연산에 대한 예제

카티션 곱 연산을 제외하고는 집합 연산을 하기 위한 조건은 각 테이블이 갖는 속성들의 개수가 같아야 하며 각 속성별로 도메인이 같아야 한다는 것이다. 예를 들어, 두 테이블 WT와 WT'가 집합 연산을 하면 테이블 WT의 i번째 속성의 도메인이 테이블 WT'의 i번째 속성의 도메인과 서로 같아야 한다.

4.3.1 합집합 연산

뷰 테이블들 $WT_i\,(1 \le i \le n)$의 합집합 $WT_1 \cup WT_2 \cup ... \cup WT_n$은 모든 테이블 WT_i에 속하는 모든 튜플로 포함하는 새로운 결과 테이블을 만드는 연산이다. 합집합 연산 결과 테이블은 $|WT_1 \cup WT_2 \cup ... \cup WT_n| \le |WT_1|+|WT_2|+...+|WT_n|$의 관계가 성립한다. 모든 뷰 테이블 $WT_i\,(1 \le i \le n)$의 합집합 연산 $\cup_{i=1}^{k} WT_i$는 정리 3에 의해 식 (15)가 성립하고 변환 규칙 3에 의해 식 (16)이 성립한다.

$$\bigcup_{i=1}^{n} WT_i = \left[\pi_{Oid}\left(\bigcup_{i=1}^{n} NT_i\right)\right] \bowtie \left[\bowtie_{j=1}^{k+l} \pi_{Oid,M}\left(\sigma_{A=' A_j'}\left(\bigcup_{i=1}^{n} NT_i\right)\right)\right] \quad (15)$$

$$\xrightarrow{Transformation} PIVOT\left[\pi_{Oid,M}\left(\sigma_{\vee_{i=1}^{k} A=' A_i'}\left(\bigcup_{i=1}^{n} NT_i\right)\right)\right] \quad (16)$$

4.3.2 차집합 연산

여러 테이블들 WT_i의 차집합 연산 $WT_1 - WT_2 - ... - WT_n$는 결과적으로 테이블 WT_1에서만 존재하는 튜플로 구성된 테이블을 만드는 연산이다. 따라서, 차집합 연산의 결과 테이블은 $|WT_1 - WT_2 - ... - WT_n| \le |WT_i|$의 관계가 성립한다. 모든 뷰 테이블 $WT_i\ (1 \le i \le n)$의 차집합 연산 $\overset{n}{i=1} WT_i$는 정리 3에 의해 식 (17)이 성립하고 변환 규칙 3에 의해 식 (18)이 성립한다.

$$\overset{n}{i=1} WT_i = \left[\pi_{Oid}\left(\overset{n}{i=1} NT_i\right)\right] \bowtie \left[\bowtie_{j=1}^{k} \pi_{Oid,M}\left(\sigma_{A=' A_j'}\left(\overset{n}{i=1} NT_i\right)\right)\right] \quad (17)$$

$$\xrightarrow{Transformation} PIVOT\left[\pi_{Oid,M}\left(\sigma_{\vee_{j=1}^{k} A=' A_j'}\left(\overset{n}{i=1} NT_i\right)\right)\right] \quad (18)$$

정리 3: 합집합 연산과 차집합 연산에 대해 기호 Θ는 합집합($\cup$)과 차집합($-$)들 중에 하나인 집합 연산자를 나타낸다. 따라서 $\Theta_{i=1}^{n} WT_i$일 때 다음 식 (19)이 성립한다.

$$\Theta_{i=1}^{n} WT_i = \left[\pi_{Oid}\left(\Theta_{i=1}^{k} NT_i\right)\right] \bowtie \left[\bowtie_{j=1}^{n} \pi_{Oid,M}\left(\sigma_{A=' A_j'}\left(\Theta_{i=1}^{k} NT_i\right)\right)\right] \quad (19)$$

증명: 합집합 연산과 차집합 연산에 대한 수평 뷰 테이블에서의 $WT_1 \Theta WT_2$ 연산은 수직형태의 저장 테이블에서 동일한 연산이 수행됨을 증명한다.

$$\begin{aligned}
&WT_1 \Theta WT_2 \\
&= \left[\left[\pi_{Oid}(NT_1)\right] \bowtie \left[\bowtie_{j=1}^{n} \pi_{Oid,M}\left(\sigma_{A=' A_j'}(NT_1)\right)\right]\right] \\
&\quad \Theta \left[\left[\pi_{Oid}(NT_2)\right] \bowtie \left[\bowtie_{j=1}^{n} \pi_{Oid,M}\left(\sigma_{A=' A_j'}(NT_2)\right)\right]\right] \\
&\qquad \text{(정의 1에 의함)} \\
&= \left[\pi_{Oid}(NT_1)\ \Theta \pi_{Oid}(NT_2)\right] \\
&\quad \bowtie \left[\bowtie_{i=1}^{k} \pi_{Oid,M}\left(\sigma_{A=' A_i'}(NT_1)\right)\right] \Theta \\
&\quad \left[\bowtie_{j=1}^{l} \pi_{Oid,M}\left(\sigma_{A=' A_i'}(NT_2)\right)\right] \\
&\qquad \text{(분배법칙의 정의에 의함)} \\
&= \left[\pi_{Oid}(NT_1 \Theta NT_2)\right] \bowtie \left[\bowtie_{i=1}^{k+l} \pi_{Oid,M}\left(\sigma_{A=' A_i'}(NT_1 \Theta NT_2)\right)\right] \\
&\qquad \text{(보조정리 2에 의함)}
\end{aligned}$$

수평 뷰 테이블 $WT_1 \Theta WT_2$에서 적용한 집합 연산이 수직적인 저장 테이블 NT_1, NT_2에서 적용한 연산은 보조정리 2에 의해 성립된다.

보조정리 2: 릴레이션 R,T,R',T'에 대하여 외부 조인 연산 중에 왼쪽 외부 조인 연산($\bowtie$)과 집합 연산(Θ)을 이용한 분배 법칙은 다음 조건을 만족한다.

$$(R \bowtie T) \Theta (R' \bowtie T') = (R \Theta R') \bowtie (T \Theta T') \quad (20)$$

증명: 우선 식 (20)에서 $(o,r) \in R$이면 $\exists t(o,t) \in T$라 하면, $(R \bowtie T)$는 보조정리 1을 이용하여 $(R \bowtie T)$와 $(R' \bowtie T')$라 하자. 여기서, 조건 릴레이션 R의 스키마는 릴레이션 R'의 스키마와 동일하고, 릴레이션 T의 스키마는 릴레이션 T'의 스키마와 동일한 성질이 만족한다. (20)의 조건을 증명하기 위해, 각 릴레이션 R,T,R',T'에 대한 튜플 $R(o,r)$, $T(o,t)$, $R'(o,r')$ 그리고 $T'(o,t')$를 이용하여 다음과 같이 성립함을 보인다.

$$\begin{aligned}
&(R \bowtie T) \Theta (R' \bowtie T') = \{(o,r,t) \mid (o,r) \in R \wedge ((o,t) \in T \vee t = \perp)\} \\
&\quad \Theta \{(o,r',t') \mid (o,r') \in R' \wedge ((o,t') \in T' \vee t' = \perp)\} \\
&\qquad \text{(보조정리 1에 의함)} \\
&= \{(o,r,t) \mid (o,r) \in R \cap ((o,t) \in T \vee t = \perp)\} \\
&\quad \Theta \{(o,r',t') \mid (o,r') \in R' \cap ((o,t') \in T' \vee t' = \perp)\} \\
&\qquad \text{(교집합 정의에 의함)} \\
&= \left[\{(o,r) \mid (o,r) \in R\} \bowtie \{(o,t) \mid (o,t) \in T \vee t = \perp\}\right] \\
&\quad \Theta \left[\{(o,r') \mid (o,r') \in R'\} \bowtie \{(o,t') \mid (o,t') \in T' \vee t' = \perp\}\right] \\
&\qquad \text{(보조정리 1에 의함)} \\
&= \{(o,r) \mid (o,r) \in R\} \Theta \{(o,r') \mid (o,r') \in R'\} \\
&\quad \bowtie \{(o,t) \mid (o,t) \in T \vee t = \perp\} \Theta \{(o,t') \mid (o,t') \in T' \vee t' = \perp\} \\
&\qquad \text{(분배법칙의 정의에 의함)} \\
&= (R \Theta R') \bowtie (T \Theta T') \qquad \text{(보조정리 1에 의함)}
\end{aligned}$$

그러므로 $(R \bowtie T) \Theta (R' \bowtie T') = (R \Theta R') \bowtie (T \Theta T')$ 이 성립한다. □

변환 규칙 3: 합집합 연산과 차집합 연산에 대해 수직적인 저장 테이블에서 PIVOT 연산의 조건으로 다음 식 (21)이 성립한다.

$$\Theta_{i=1}^{n} WT_i \xrightarrow{Transformation} PIVOT\left[\pi_{Oid,M}(\sigma_{\vee_{j=1}^{k} A=' A_j '}(\Theta_{i=1}^{n} NT_i))\right] \quad (21)$$

4.3.3 카티션 곱 연산

카티션 곱 연산은 여러 테이블들의 튜플을 결합하여 조합 가능한 테이블로 생성한다. 예를 들어, $WT(A_1,A_2,...,A_n) \times WT(A_1',A_2',...,A_n')$의 결과는 $(n+m)$개의 열들을 가진 결과 테이블 $Q(A_1,A_2,...,A_n,A_1',A_2',...,A_n')$이다. 결과 테이블 Q는 모든 조합에 대한 튜플을 가진다. 따라서, 이 연산에 대한 결과 테이블은 $|WT_1 \times WT_2 \times ... \times WT_n| = |WT_1| \times |WT_2| \times ... \times |WT_n|$의 관계가 성립한다. 따라서 카티션 곱 연산 $\times_{i=1}^{n} WT_i$은 정리 4에 의해 식 (22)이 성립하고 변환규칙 3에 의해 식 (23)이 성립한다.

$$\times_{i=1}^{n} WT_i = [\times_{i=1}^{n} \pi_{Oid}(NT_i)] \bowtie [\bowtie_{j=1}^{k} \pi_{Oid,M}(\sigma_{A=' A_j '}(\cup_{i=1}^{n} NT_i))] \quad (22)$$

$$\xrightarrow{Transformation} PIVOT\left[[\times_{i=1}^{n} \pi_{Oid}(NT_i)] \bowtie [\cup_{i=1}^{n} \pi_{Oid,M}(NT_i)]\right] \quad (23)$$

정리 4: 뷰 테이블들 $WT_1,WT_2,...,WT_n$에 대해 카티션 곱 연산을 실행할 때, 저장 테이블들 $NT_1,NT_2,...,NT_k$에 대해서는 우선, 저장 테이블 $NT_1 \times NT_2 \times ... \times NT_n$을 연관시켜 열 Oid를 추출하고, 해당 튜플의 조건 $A = 'A_j'$에 대한 모든 $NT_1 \cup NT_2 \cup ... \cup NT_n$의 열 Oid, M을 추출하고 이를 완전 외부 조인 연산으로 실행한다. 그런 후에, 앞서 추출한 Oid로 왼쪽 외부 조인을 이용하면 다음 식 (24)이 성립한다.

$$\times_{i=1}^{n} WT_i = [\times_{i=1}^{n} \pi_{Oid}(NT_i)] \bowtie [\bowtie_{j=1}^{k} \pi_{Oid,M}(\sigma_{A=' A_j '}(\cup_{i=1}^{n} NT_i))] \quad (24)$$

증명: 카티션 곱 연산에 대해 두 개의 수평적인 뷰 테이블 WT_1,WT_2에서 적용한 $WT_1 \times WT_2$ 연산은 수직적인 저장 테이블에서 동일한 연산이 수행됨을 증명한다.

$WT_1 \times WT_2$

$$= \left[[\pi_{Oid}(NT_1)] \bowtie [\bowtie_{j=1}^{k} \pi_{Oid,M}(\sigma_{A=' A_j '}(NT_1))]\right] \times \left[[\pi_{Oid}(NT_2)] \bowtie [\bowtie_{j=1}^{k} \pi_{Oid,M}(\sigma_{A=' A_j '}(NT_2))]\right]$$

(정리 1에 의함)

$$= \left[[\pi_{Oid}(NT_1) \times \pi_{Oid}(NT_2)]\right] \bowtie \left[[\bowtie_{j=1}^{k} \pi_{Oid,M}(\sigma_{A=' A_j '}(NT_1))] \times [\bowtie_{j=1}^{k} \pi_{Oid,M}(\sigma_{A=' A_j '}(NT_2))]\right]$$

(분배법칙의 정의에 의함)

$$= [\pi_{Oid}(NT_1) \times \pi_{Oid}(NT_2)] \bowtie [\bowtie_{j=1}^{k} \pi_{Oid,M}(\sigma_{A=' A_j '}(NT_1 \cup NT_2))]$$

(보조정리 3 에 의함)

수평적인 뷰 테이블 WT_1,WT_2에서 적용한 카티션 곱 연산이 수직적인 저장 테이블 NT_1,NT_2에서 적용한 연산은 보조정리 3에 의해 성립되고, 여러 테이블에 대한 카티션 곱 연산이 적용이 되므로, 다음 식 (25)으로 나타낼 수 있다.

$$[\times_{i=1}^{n} \pi_{Oid}(NT_i)] \bowtie [\bowtie_{j=1}^{k} \pi_{Oid,M}(\sigma_{A=' A_j '}(\cup_{i=1}^{n} NT_i))] \quad (25)$$

보조정리 3: 릴레이션 R,T,R',T'에 대하여 카티션 곱 연산을 이용한 분배 법칙은 다음 조건에 대한 식 (26)을 만족시킨다.

$$(R \bowtie T) \times (R' \bowtie T') = (R \times R') \bowtie (T \cup T') \quad (26)$$

증명: 우선 $(o,r) \in R$이면 $\exists t(o,t) \in T$라 하면, $(R \bowtie T)$는 보조정리 2를 이용하여 $(R \bowtie T)$와 $(R' \bowtie T')$라 하자. 식 (19)의 조건을 증명하기 위해 이용될 각 릴레이션 $R,T,R'T'$에 대하여 임의의 튜플 $R(o,r)$, $T(o,t)$, $R'(o,r')$ 그리고 $T'(o,t')$에 대해 다음과 같이 성립함을 보인다.

$$(R \bowtie T) \times (R' \bowtie T') = \{(o,r,t) \mid ((o,r) \in R \wedge ((o,t) \in T \vee t = \perp)\} \times \{(o,r',t') \mid ((o,r') \in R \wedge ((o,t') \in T \vee t = \perp)\}$$

(보조정리 2의 식 (12)에 의함)

$$= \{(o,r) \mid (o,r) \in R\} \times \{(o,r') \mid (o,r') \in R'\} \bowtie \{(o,t) \mid (o,t) \in T \vee t = \perp\} \times \{(o,t') \mid (o,t') \in T' \vee t' = \perp\}$$

(보조정리 2에 의함)

여기서, $\{(o,t) \mid (o,t) \in T \vee t = \perp\} \times \{(o,t') \mid (o,t') \in T' \vee t' = \perp\}$에서 ×연산은 튜플 $T(o,t)$과 $T'(o,t')$을 하나의 릴레이션으로 통합하기 위해 ×연산에서 ∪ 연산으로 변경한다.

$= \{(o,r) \mid (o,r) \in R\} \times \{(o,r') \mid (o,r') \in R'\}$

$\rtimes \{(o,t) \mid (o,t) \in T \vee t = \perp\} \cup \{(o,t') \mid (o,t') \in T' \vee t' = \perp\}$

(×→∪ 연산의 변경)

$= (R \times R') \rtimes (T \cup T')$ (보조정리 2에 의함)

그러므로 $(R \rtimes T) \times (R' \rtimes T') = (R \times R') \rtimes (T \cup T')$이 성립한다. □

정리 3에 의하여 카티션 곱 연산은 완전 외부 조인 연산 대신에 *PIVOT* 연산을 이용하여 간결하게 표현될 수 있다. 따라서 *PIVOT* 연산의 이용은 정의 2와 정리 3을 기반으로 카티션 곱 연산에 대한 변환규칙 4가 성립한다.

변환 규칙 4: 한 테이블에서 다른 테이블들의 튜플들의 모든 가능한 조합으로 테이블을 생성시키는 카티션 곱 연산에 대한 *PIVOT* 연산은 다음 식 (27)이 성립한다.

$$\times_{i=1}^{n} WT_i \xrightarrow{Transformation} \text{PIVOT}\left[\left[\times_{i=1}^{n} \pi_{Oid \text{ on } NT_i}(NT_i)\right] \rtimes \left[\bigcup_{i=1}^{n} \pi_{Oid,M}(NT_i)\right]\right] \quad (27)$$

지금까지 논했던 셀렉트 연산, 프로젝트 연산, 합집합 연산, 차집합 연산, 카티션 곱 연산은 관계 대수의 필수적인 연산자이며 조인 연산, 디비젼 연산들은 필수적인 관계 연산자를 두 개 이상 조합하여 표현이 가능하다.

5. 성능 분석 및 평가

이 장에서는 제안한 질의 변환 방법의 성능을 비교.분석하기 위해, 수평적인 뷰 테이블을 저장하고 이에 대해 연산을 직접 수행한 것과 이를 수직적인 저장 테이블에 대해 변환한 질의를 처리하여 성능을 평가한다. 또한 질의 변환 방법으로 기존에 제시되었던 [7]의 방법과 본 논문의 PIVOT 연산을 사용하여 변환하는 방법에 대한 성능을 서로 비교하였다. .

5.1 실험 환경

모든 실험은 물리적 메모리 1GB와 1.70GHz의 Intel Pentium 4 CPU를 사용하는 윈도우 환경에서 Microsoft SQL Server 2005 Beta 2의 데이타베이스 시스템에서 수행하였다. SQL Server의 구성으로 쿼리당 최소 메모리의 크기는 1024KB이다. 그리고 본 실험은 [7]의 데이타 세트와 같이 다음과 같은 매개변수를 이용한다: 즉, 수평적인 뷰 테이블의 열과 행의 수, 널 값의 비율, 각 열에 대한 술어의 선택율, 그리고 검색될 열의 수 등이다. 매개 변수로는 e-비즈니스에서 사용되는 수평 테이블을 언급한 Agrawal 등[7]의 실험에서와 같이, 데이타 집합, 널 값의 비율, 선택률(selectivity) 등을 달리하면서 실험하였다. 데이타 집합은 수평 테이블을 기준으로 열의 수와 행의 수(#cols×#rows)을 달리하여, 200×100k, 400×50k, 800×25k, 1000×20k의 네 개를 구성하였다. 널 값 비율은 Agrawal 등[7]의 실험에서와 같이 90%와 95%를 사용하였고, 선택률은 질의 종류에 따라 값을 달리하면서 실험하였다.

수평적인 뷰 테이블 *WT*과 수직적인 저장 테이블 *WT*의 스키마는 다음과 같이 생성하였다.

WT(*Oid* integer, *A*1 float, *A*2 float,..., *An* float)

NT(*Oid* integer, *A* char(4), *M* float)

이러한 스키마에 대해 저장 테이블 *NT*의 데이타 타입을 float형으로 설정하였다. 이런 이유는 *PIVOT* 연산시 행을 열로 변경할 때 집계로 계산하기 때문에 수치 데이타 형으로 변수를 선언한 것이다. 또한, 뷰 테이블 *WT*에서도 데이타 타입을 float형으로 선언하였다. 그리고 처리 효율을 높이기 위해 열 *Oid*와 *Key*를 기준으로 각각 클러스터링 하였다. 수평적인 뷰 테이블의 경우 질의에 포함하는 모든 열들에 대해 인덱스를 생성하였고, 수직적인 저장 테이블의 경우 세 개의 열들에 대해 인덱스를 생성하였다. 또한 뷰 테이블 *WT*에 대한 질의들에 대해, 수평적인 테이블에서 직접 실행하는 방법(이하 *WT* 연산), 수직적인 저장 테이블 *NT*의 연산으로 변환하는 기존 방법(이하 *ANT* 연산), 및 이 논문의 변환 방법(이하 *PIVOT* 연산)의 세가지 방법을 서로 비교한다.

실험에 사용한 질의는 프로젝션, 셀렉션, 그리고 프로젝션과 셀렉션을 섞은 혼합 질의의 세 가지이다. 다음은 이중 프로젝션($\pi_{A10,A100}(WT)$)에 대한 세 가지 방법의 SQL 구문 예를 나타내며, 셀렉션과 혼합 질의도 유사한 방법으로 SQL 구문을 작성하였다.

- WT 연산:

```
select A10, A100
from WT
```

- ANT 연산:

```
select nt10.A10, nt100.A100
from (select distinct O from NT) as nt
left outer join (select O, M as A10 from ONT
where A='A10') as nt10 on nt.O = nt10.O
left outer join (select O, M as A50 from ONT
where A='A100') as nt100 on nt.O = nt100.O
```

- Pivot 연산:

```
select A10, A100
from NT
pivot (sum(m) for A in (A10, A100)) as pvt
```

5.2 성능평가

이 절에서는 실험 결과를 제시한다. 수평적인 뷰 테이

블에서는 원래 주어진 질의를 사용하고, 수직적인 저장 테이블에서는 그 질의를 변환하여 실행한다. 질의 변환은 [7]에서 제안한 방법과 본 연구에서 제안한 방법을 통해 실험을 수행하였다. 또한 전형적인 데이타베이스의 질의는 하나의 연산만을 사용하는 것이 아니라 여러 연산을 이용한 복합적인 연산을 이용하므로 본 실험에서도 복합적인 연산을 통해 실험하였다. 이 복합적인 연산에 대해서 선택율과 질의 결과에 포함하려는 열 목록의 수를 매개 변수를 통해 임의적으로 변경하여 실험하였다.

5.2.1 공간 활용도 실험

저장공간의 비교는 수평적인 뷰 테이블 WT와 수직적인 저장 테이블 NT에 대해서 수행하였다. 그림 7은 수평 테이블 WT과 수직 테이블 NT의 저장 공간을 비교한 결과로, 널 값이 90%와 95%인 경우이다[7]. 실험 결과를 보면, 수직 테이블 NT은 수평 테이블 WT에 비해 저장공간을 크게, 90%인 경우 평균 1/5로, 95%인 경우 평균 1/10로 줄였음을 알 수 있다. 하지만, 데이타 세트 1200×10k인 뷰 테이블 WT는 열의 수 1024로 제한되므로 생성되지 않았다. 이와 같이 저장공간을 줄일 수 있는 이유는 제3장에서 설명한 바와 같이, 수평 테이블 WT에서는 널 값들이 저장되어야 하는 반면에, 수직 테이블 NT에서는 널 값을 저장하지 않아도 되기 때문이다[7]. 즉, 수평 테이블 WT에서는 널 값을 포함한 튜플들이 저장되어 널 값 비율이 저장공간에 영향을 주지 않는 반면에, 수직 테이블 NT에서는 널 값을 갖는 튜플은 저장하지 않기 때문에 널 값 비율이 높을수록 저장공간이 줄어듦을 알 수 있다. 그리고, 데이타 집합에 따라 저장공간의 변화가 없는데, 이는 저장공간 자체는 행과 열의 비율이 아닌 널 값의 비율에 따라 결정되기 때문이다.

5.2.2 프로젝트 연산의 실험

그림 8과 그림 9의 실험 결과는 데이타 세트 200×100k, 400×50k, 800×25k, 1000×20k에서 프로젝트 연산의 성

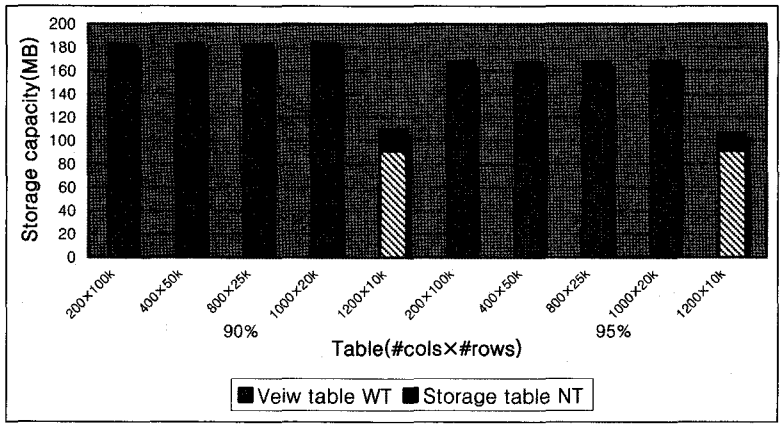


그림 7 테이블 WT와 테이블 NT에 대한 저장 공간 비교

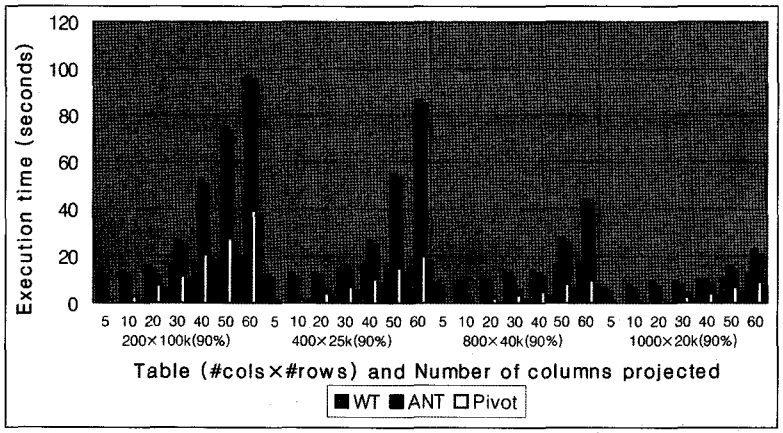


그림 8 널 비율 90%인 프로젝트 연산의 성능 비교

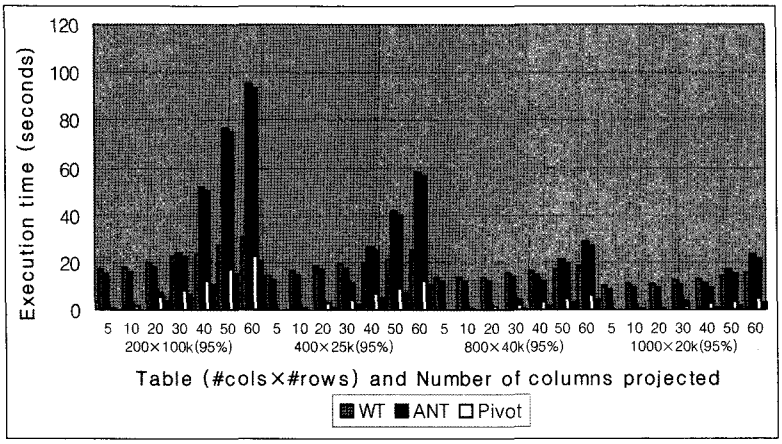


그림 9 널 비율 95%인 프로젝트 연산의 성능 비교

능 비교를 보이고 있다. 그림 8에서는 널의 값 비율이 90%이고, 그림 9에서는 널 값의 비율이 95%이다. 또한 실험의 변경요인은 질의에 포함될 열의 수를 5개, 10개, 20개, …, 60개로 변경하여 실험을 수행하였다.

그림 8과 그림 9에서는 뷰 테이블 *WT*에서의 프로젝트 연산의 결과가 비교적 변화의 폭이 적음을 나타내고 있으며, *ANT* 연산과 *PIVOT* 연산은 질의에 포함될 열의 수에 따라 계산비용이 점진적으로 증가되었다. 특히, 저장 테이블 *NT*에서의 *ANT*연산은 질의에 포함될 열의 수의 증가에 따라 조인에 참가하는 튜플 수가 증가하게 되므로 이에 대한 계산비용의 차이가 크게 된다. 또한, 뷰 테이블 *WT*에서의 프로젝트 연산이 *PIVOT* 연산보다 계산비용이 높은 것은 튜플의 수에 따라 영향을 받기 때문이다.

5.2.3 셀렉트 연산의 실험

그림 10과 그림 11에서는 각각의 널 값의 비율 90%, 95%에 대해 셀렉트 연산의 조건 선택율 0.5%, 1%, 5%, 10%으로 변경하고 데이타 세트 200×100*k*, 400×50*k*, 800×25*k*, 1000×20*k*에 적용한 셀렉트 연산의 실험 결과를 보이고 있다. 셀렉트 연산에 대한 질의는 다음과 같다.

Select *A*10, *A*100 From *WT*
Where *A*10 = 1.0 or *A*100 = 10.0

그림 10과 그림 11에서와 같이, 셀렉트 연산에 대해 테이블 *WT*에서 전형적인 질의에 대한 결과의 성능이 좋지 않는 이유는 검색하기 위한 해당 튜플을 검색하기 전에 전체 튜플을 스캔하기 때문이다. 반면에, 저장 테이블 *NT*에서의 *PIVOT* 연산은 질의의 선택율이 낮은 경우 *ANT*연산과 근소한 성능 차이를 보이고 있으며, 이 연산들은 선택율이 높아질수록 뷰 테이블 *WT*에서의 계산비용보다는 더 적게 소요된다.

5.2.4 혼합 연산의 실험

사용자의 질의를 처리하기 위해서는 일반적으로 셀렉트 연산과 프로젝트 연산을 복합적으로 사용한다. 따라서 본 실험에서도 셀렉트 연산과 프로젝트 연산을 복합

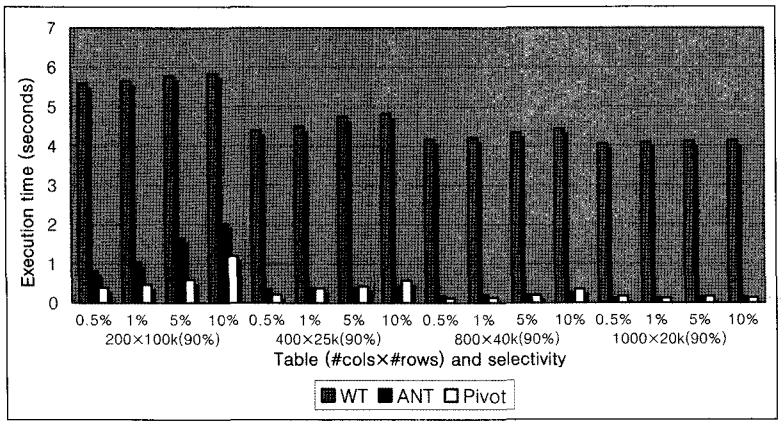


그림 10 널 비율 90%인 셀렉트 연산의 성능 비교

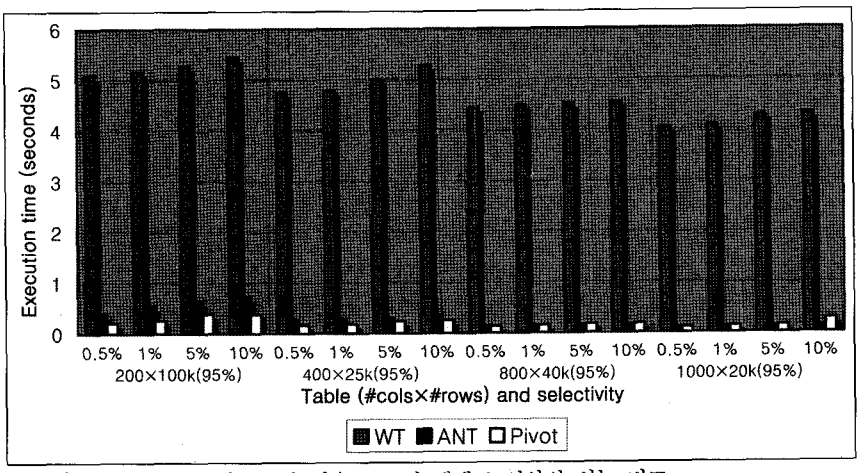


그림 11 널 비율 95%인 셀렉트 연산의 성능 비교

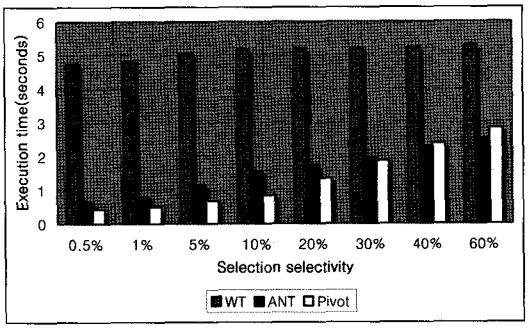


그림 12 선택율에 대한 혼합 질의 성능 비교

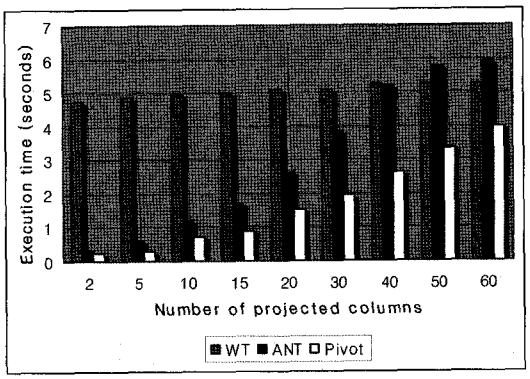


그림 13 컬럼의 수에 대한 혼합 질의 성능 비교

적으로 사용한 혼합 연산에 대해서 수행하였다. 그림 12에서는 각 테이블의 데이타 세트에 대한 널 값의 비율 90%인 데이타 세트에 대해 실험한 결과를 보인다. 실험은 질의에 포함하는 열의 개수를 10개로 하고 셀렉트 연산에 대한 술어의 선택율은 각각 0.5%, 1%, 5%, 10%, 20%, 30%, 40%, 60%으로 변경하여 수행하였다.

마찬가지로 그림 12에서는 널 값의 비율 95%인 각 테이블의 데이타 세트에 대해 셀렉트 연산의 선택율은 5%으로 고정하고 질의에 포함될 열의 개수를 2, 5, 10, 15, 20, 30, 40, 50, 60개로 변경하여 나타낸 실험결과를 보이고 있다.

그림 12와 그림 13에서 보여지는 것처럼, 뷰 테이블 *WT*에서 적용한 연산의 결과 실행시간은 큰 변화의 폭이 없다. 따라서, 셀렉트 연산의 선택율과 질의 결과에 포함될 열의 개수의 변화요인으로는 작용하지 않는다. 그리고 그림 12에서 *ANT*연산과 *PIVOT* 연산의 경우 선택율에 대한 변화요인에 대한 영향은 다소 적게 나타났으며, 그림 13에서의 *ANT* 연산은 결과에 포함될 열의 개수에 따른 변환요인에 따라 *PIVOT* 연산의 계산 비용보다는 큰 폭을 나타내고 있다.

그림 14에서는 검색될 열의 개수와 선택율의 점증적으로 변화시켜 실험한 결과를 보이고 있다. 실험은 널 값의 비율 90%이고 질의 결과에 포함될 열의 개수는 5개, 30개, 60개이며, 선택율은 0.5%, 1%, 5%, 10%으로 다양한 관점으로 수행하였다.

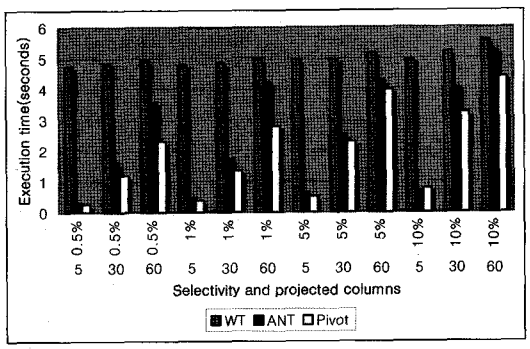


그림 14 선택율과 컬럼의 수에 대한 혼합 질의 성능 비교

실험의 결과로 뷰 테이블 *WT*에 대한 계산비용은 비교적 동일한 성능을 나타내고 있다. 그리고 저장 테이블 *NT*에서의 *ANT* 연산과 *PIVOT* 연산은 열의 개수와 선택율이 아주 낮을 경우 근소한 성능 차이를 나타내고 있으나, 열의 개수와 선택율이 점증적으로 높아질수록 PIVOT 연산이 보다 좋은 성능을 보이고 있다.

6. 결 론

OLAP에서 사용하는 다차원 데이타는 여러 차원에 대해, 각 차원의 값에 대응하는 측정값을 유지하는 다차원 배열 구조를 갖는다. 이 다차원 데이타를 사용자들이 편리하게 사용할 수 있도록 피벗 테이블이나 교차 테이블과 같이 많은 수의 열을 가진 수평적인 테이블을 널리 사용하고 있다. 관계 데이타베이스에서 다차원 데이타를 제공하기 위해 이러한 수평적인 테이블을 다룰 수 있는 기능을 추가하려는 노력이 최근에 많이 진행되고 있다[5,7-9].

하지만 관계 데이타베이스에서는 제한된 수의 애트리뷰트들을 가지며, 애트리뷰트들의 값의 모임인 행을 많이 갖는 수직적인 테이블을 제공하기 때문에 이 수평적인 테이블 형태의 다차원 데이타를 직접적으로 제공하는데 여러 가지 어려움이 있다. 이런 문제를 해결하는 방법의 하나는 사용자들에게는 수평적인 테이블을 뷰로 제공하고 실제로는 관계 데이타베이스의 수직적인 테이블로 저장하며, 수평적인 뷰에 대한 질의를 저장된 수직적인 테이블에 대한 질의로 바꿔 실행하는 방법을 사용할 수 있다[7,11]. 이렇게 할 경우, 수평적인 뷰 테이블을 관계 데이타베이스에서 구현 가능하다는 것 외에도, 수평적인 다차원 데이타에 많이 발생하는 희박 데이타도 유지할 필요가 없어 저장 공간을 효율적으로 사용할 수 있다는 장점이 있다. 따라서 이 연구에서는 수평적인 뷰에 대한 관계 대수 연산들(즉, 프로젝트, 셀렉트, 합집합, 차집합, 카티션 곱)을 수직적인 저장 테이블에 대한 대수 연산으로 변환하는 규칙을 정의하고 이를 증명하였다. 이 변환을 효율적으로 처리할 수 있도록 최근 DBMS들이 OLAP 응용을 위해 제공하는 *PIVOT* 연산을 사용함으로써[8,9], 기본 대수 연산만을 사용하는 기존 방법[7]에 비해 더욱 효율적으로 변환할 수 있도록 하였다. 또한 실험을 통하여 제안된 방법이 다른 방법들에 비해 저장 효율과 질의 처리 성능이 우수함을 보였다.

따라서 이 논문의 방법을 사용할 경우, 사용자들에게 다차원 데이타에 대한 편리한 사용자 뷰를 제공하면서 이를 기존의 관계 DBMS에서 효과적으로 저장하고 관리할 수 있을 것이다. 이를 토대로 향후에는 좀 더 복합적인 질의를 처리하고 최적화하는 방법에 대해 연구할 예정이다. 또한, 3차원 이상의 수평 형태의 데이타를 2차원 수직 테이블로 저장하고 질의하는 방안에 대해 향후 연구를 진행할 계획이다.

참 고 문 헌

[1] Chaudhuri, S. and Dayal, U., "An Overview of Data Warehousing and OLAP Technology," *SIGMOD Record*, 26(1): 65-74, 1997.

[2] Pilot Software, "An Introduction to OLAP: Multidimensional Terminology and Technology," http://www.pilotsw.com/olap/olap.htm.

[3] 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, 1999.

[4] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, H. Pirahesh, 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, 1997.

[5] A. Witkowski, S. Bellamkonda, T. Bokaya, G. Dorman, N. Folkert, A. Gupta, L. Shen, and S. Subramanian. "Spreadsheets in RDBMS for OLAP," *In Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, San Diego, CAA, pp. 52-63, 2003.

[6] A. Witkowski, S. Bellamkonda, T. Bozkaya, N. Folkert, A. Gupta, L. Sheng, and S. Subramanian, "Business Modeling Using SQL Spreadsheets," *"In Proc. of the 29th Int'l Conf. on Very Large Data Bases*(VLDB), Berlin, Germany, pp. 1117-1120, 2003.

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

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

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

[10] Y. Zhao, P. Deshpande, and J. Naughton, "An Array-Based Algorithm for Simultaneous Multidimensional Aggregates," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, Tucson, Arizona, United States, pp. 159-170, 1997.

[11] L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. "On efficiently implementing Schema-SQL on an SQL database system," In *Proc, of the 25th Int'l Conf. on Very Large Data Bases* (VLDB), Edinburgh, Scotland, pp. 471-482. September 7-10, 1999.

[12] C. Cunningham, G. Graefe, and C. A. Galindo-legaria. "PIVOT and UNPIVOT: Optimization and

Execution Strategies in an RDBMS," In *Proc. of the 30th Int'l Conf. on Very Large Data Bases* (VLDB), Toronto, Canada, pp. 998-1009, 2004.

[13] Songting Chen and Elke A. Rundensteiner. "GPIVOT: Efficient Incremental Maintenance of Complex ROLAP Views," In *Proc. of the 21st Int'l Conf. on Data Engineering*(ICDE), pp. 552-563, 2005.

신 성 현

2000년 2월 관동대학교 컴퓨터공학과 학사. 2002년 8월 강원대학교 컴퓨터과학과 석사. 2005년 8월 강원대학교 컴퓨터과학과 박사 수료. 2000년 3월~현재 한국과학기술원 첨단정보기술연구센터 연구보조원. 관심분야는 Data warehouse, OLAP, Database Applications, XML data processing

김 진 호

1982년 2월 경북대학교 전자공학과 학사 1985년 2월 한국과학기술원 전산학과 석사. 1990년 2월 한국과학기술원 전산학과 박사. 1995년 8월~1996년 7월 미국 미시간 대학교 객원 교수. 2003년 2월~2004년 2월 미국 Drexel University 객원 교수. 1999년 3월~현재 한국과학기술원 첨단정보기술연구센터 연구원. 1990년 8월~현재 강원대학교 컴퓨터과학과 교수. 관심분야는 Data warehouse, OLAP, Data Mining, Real-time/Embedded Database, Main-memory database, Data Modeling, Web Database Technology

문 양 세

1991년 2월 한국과학기술원 과학기술대학 전산학과 학사. 1993년 2월 한국과학기술원 전산학과 석사. 2001년 8월 한국과학기술원 전자전산학과 전산학전공 박사. 1993년 2월~1997년 2월 현대전자산업(주) 통신사업본부 주임연구원. 2001년 9월~2002월 2월 (주)현대시스콤 호처리개발실 선임연구원 2002년 2월~2005년 2월 (주)인프라밸리 기술연구소 기술위원(이사). 2005년 3월~현재 한국과학기술원 첨단정보기술연구센터 연구원. 2005년 3월~현재 강원대학교 컴퓨터과학과 조교수. 관심분야는 Data Mining, Knowledge Discovery, Stream Data, Storage System, Database Applications, Mobile/Wireless Communication Services & Systems