관계 데이터베이스에서 수평 뷰 테이블에 대한 PIVOT 기반의 질의 최적화 방법

신 성 현[†] · 문 양 세[††] · 김 진 호[†††] · 강 공 미[††††]

요 약

온라인 분석 처리(On-Line Analytical Processing)에서는 다양한 분석을 효과적으로 처리하기 위해, 다차원 구조의 데이터를 열(column)에 애트리뷰트 값이 표시되는 넓은 형태의 수평 테이블로 표현한다. 관계형 테이블들은 보통 애트리뷰트의 개수에 제한이 있으므로(MS SQLServer와 Oracle은 1,024개 컬럼으로 제한), 이러한 수평 테이블을 직접 저장하기 어렵다. 본 연구에서는 상용 RDBMS에서 제공하는 PIVOT 연산을 이용하여 수평 테이블에 대한 질의를 저장된 수직 테이블에 대한 질의로 변환하는 효율적인 최적화 실행 전략을 제시한다. 우선, 관계 데이터베이스에서 수평 테이블을 차원의 이름을 열로 갖는 수직 테이블로 저장하고, 수평 테이블에 대한 질의를 수직 테이블에 대한 질의로 변환하는 다양한 최적화 전략을 제시한다. 특히, 관계 연산(셀렉션, 프로젝션, 조인 등)에 대한 여러 변환 방법을 제안한다. 이때, 변환된 질의는 여러 가지 방법으로 수행할 수 있으며, 각 방법에 따라 수행 시간이 서로 다르다. 그러므로 PIVOT 연산을 사용하여 변환된 질의를 수행하는 최적화 전략을 제시한다. 마지막으로, 다양한 실험을 통해 여러 질의 트리의 방법에 따라 수행 시간을 측정하여 비교 평가한다.

키워드 : 다차원 데이터, 데이터 웨어하우스, *PIVOT* 연산, 질의 최적화

A PIVOT based Query Optimization Technique for Horizontal View Tables in Relational Databases

Sung-Hyun Shin[†] · Yang-Sae Moon[††] · Jinho Kim[†††] · Gong-Mi Kang[††††]

ABSTRACT

For effective analyses in various business applications, OLAP(On-Line Analytical Processing) systems represent the multidimensional data as the horizontal format of tables whose columns are corresponding to values of dimension attributes. Because the traditional RDBMSs have the limitation on the maximum number of attributes in table columns(MS SQLServer and Oracle permit each table to have up to 1,024 columns), horizontal tables cannot be directly stored into relational database systems. In this paper, we propose various efficient optimization strategies in transforming horizontal queries to equivalent vertical queries. To achieve this goral, we first store a horizontal table using an equivalent vertical table, and then develop various query transformation rules for horizontal table queries using the PIVOT operator. In particular, we propose various alternative query transformation rules for the basic relational operators, selection, projection, and join. Here, we note that the transformed queries can be executed in several ways, and their execution times will differ from each other. Thus, we propose various optimization strategies that transform the horizontal queries to the equivalent vertical queries when using the PIVOT operator. Finally, we evaluate these methods through extensive experiments and identify the optimal transformation strategy when using the PIVOT operator.

Key Words : Multidimensional Data, Data Warehouse, PIVOT operation, Query Optimization

1. 서 론

온라인 분석 처리(On-Line Analytical Processing: OLAP)는 데이터 웨어하우스에 저장된 대용량의 데이터에서 여러 관점의 다양한 분석 정보를 추출하기 위한 다차원 데이터 분석을 지원하는 기법이다[1]. OLAP에서는 이러한 다차원 데이터 분석을 편리하고 효율적으로 분석할 수 있도록 다양한 차원에 대해 측정 값을 유지할 수 있도록 데이터를 여러 형태로 표현하고 저장한다[2]. 이러한 데이터 저장 및 표현 방법 중의 하나가 수평 테이블(horizontal table)[3]이다. 수평 테이블은 다차원 데이터를 사용자에게 편리하게 보여주는 방법으로, 분석의 기준이 되는 애트리뷰트들의 값을 테이블의 행과 열의 색인 값으로 하는 이차원 테이블을 의미

※ 첨단정보기술연구센터(AITrc)를 통하여한국과학재단(KOSEF)의 지원을 받았음.
† 정 회 원 : 한양대학교 BK21사업단 정보기술분야 박사후연구원
†† 정 회 원 : 강원대학교 IT특성화학부(대학) 컴퓨터학부 조교수 (교신저자)
††† 정 회 원 : 강원대학교 IT특성화학부(대학) 컴퓨터학부 교수
†††† 준 회 원 : 강원대학교 컴퓨터과학과 박사수료
논문접수 : 2006년 8월 22일, 심사완료 : 2007년 3월 9일

Year	TV	Radio	…	DVD
2004	160	150	…	⊥
2005	190	⊥	…	230
2006	⊥	220	…	240

(a) 수평 테이블

Year	Product	Sales
2004	TV	160
2004	Radio	150
:	:	:
2005	TV	190
:	:	:
2005	DVD	230
2006	Radio	220
:	:	:
2006	DVD	240

(b) 수직 테이블

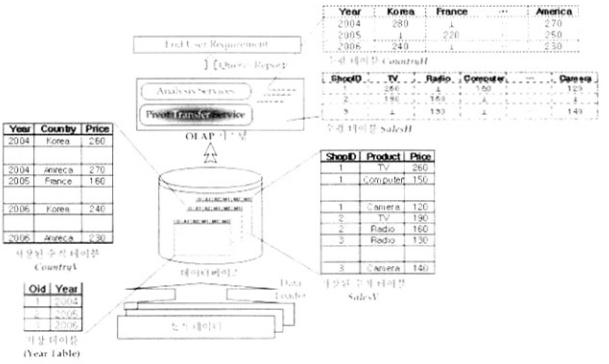
(그림 1) 수평 뷰 테이블과 수직 테이블의 예

한다[2, 4]. 즉, 분석의 기준이 되는 여러 차원(dimension)들과 분석할 내용에 해당하는 측정(measure)값들로 구성된다.

그림 1(a)는 상품 판매량을 나타낸 수평 테이블의 예를 나타낸다. 그림을 보면, 수평 테이블은 차원 *Year*, 차원 *Product*의 열 이름(*TV, Radio, …, Computer*), 그리고 측정 값 *Sales*로 구성된다. 이러한 수평 테이블을 기반으로 각 기간별과 상품별 차원에 대한 상품 판매량을 쉽게 파악할 수 있다. 예를 들어, 그림 1(a)의 수평 테이블을 보면 2005년도에는 *Radio*의 판매량이 가장 저조하며, *DVD*의 판매량은 가장 높음을 쉽게 파악할 수 있다. 이러한 수평 테이블은 Excel의 피벗 테이블이나 통계 데이터 표현에 많이 사용되는 교차 테이블(cross table)과 유사한 형태로써, 테이블의 열 이름으로 (차원 애트리뷰트 이름이 아닌) 차원 애트리뷰트 값이 표시되며 많은 수의 열을 갖는 수평적인 스키마 구조를 갖는다[3-6].

그런데, 기존의 사용되는 관계형 DBMS(Relational DBMS: RDBMS)에서는 이러한 수평 테이블을 효과적으로 지원하지 못한다. 그 이유는 그림 1(b)와 같이, 기존 RDBMS가 소수의 애트리뷰트(열)와 다수의 튜플(행)으로 구성되는 수직 테이블(vertical table)을 주로 다루고 있으며, 이에 따라 대부분의 RDBMS는 테이블이 가질 수 있는 애트리뷰트 개수를 제한하기 때문이다. 예를 들어, 상용 RDBMS인 마이크로소프트의 SQLServer 2005[7]와 오라클의 Oracle 9i[8] 등은 하나의 테이블이 가질 수 있는 열의 개수를 최대 1,024개로 제한하고 있다. 반면에, e-비즈니스 응용이나 OLAP 분석에서는 수 천 혹은 수 만개의 애트리뷰트로 구성된 수평 테이블의 지원을 요구하고 있다[3]. 하지만, 여러 애트리뷰트들이 모여 하나의 큰 수평 테이블로 저장할 경우, 애트리뷰트 중 다수가 널 값(null value)를 포함할 수 있으며 그 결과 테이블 저장 단계에서 공간을 낭비하게 되는 희박(sparse) 테이블이 될 수 있다. 이에 따라 다차원 분석이 용이한 수평 테이블을 상용 RDBMS의 수직 테이블을 사용하여 저장할 수 있는 방법이 필요하게 되었다.

수평 테이블 구조의 문제점을 해결하기 위한 효과적인 방



(그림 2) 수평 뷰 테이블과 수직 테이블의 구성

법 중의 하나는, 많은 수의 열을 갖는 수평적인 스키마 구조에서 소수의 열과 다수의 행으로 구성되는 수직적인 스키마 구조로 표현하는 것이다[3, 9, 10]. 예를 들어, 그림 2에서는 소스 데이터로부터 유도된 다차원 데이터를 관계 DBMS에서 수직 테이블로 저장하고 있으며, 저장된 데이터를 편리하게 분석할 수 있도록 OLAP 시스템에서는 수평 테이블의 뷰를 사용하고 있다. 이러한 시스템에서 다수의 행과 소수의 열로 구성하는 수직 테이블은 관계형 DBMS에 적합한 형태로 저장하고, OLAP 시스템에서는 수평 테이블을 뷰로서 제공한다. 예를 들어, 그림 2에서는 수평 뷰 테이블 *CountryH*와 수직 테이블 *CountryV*를 나타낸다. 사용자에게 뷰로 제공하는 수평 테이블 *CountryH*는 열 *ShopID, Korea, France, …, America*로 구성되며, 실제 저장되는 수직 테이블 *CountryV*는 열 *ShopID, Product, Price*로 구성된다. 여기서, 수평 테이블의 행과 열은 수직 테이블의 데이터 값(튜플)로 변환하여 저장한다. 이렇게 할 경우, 수평 뷰 테이블을 수직 테이블로 효율적으로 저장할 수 있다. 하지만 사용자 관점에서 정보를 분석하기 위해 수평 뷰 테이블에 대한 질의를 수행하며, 이 질의는 실제 저장된 수직 테이블에 대한 질의로 변환하여 처리할 수 있다.

따라서, 사용자들에게는 편리한 수평 테이블을 가상적인 뷰로 제공하며, 실제로는 수직 테이블 형태로 저장하는 것이 기존 RDBMS에서 더욱 효율적이고 실현 가능한 방법이 될 수 있다. 이에 따라, 상용 RDBMS의 수직 테이블을 사용하여 저장할 수 있는 방법이 필요하게 되었다. Agrawal 등[3]은 수평 테이블을 수직 테이블로 변환한 후, 수평 테이블에 대한 질의를 관계 대수 연산으로 변환하여 실행하는 방법을 제안하였다. 또한, Cunningham 등[9]에서는 테이블의 행과 열을 변경하는 *PIVOT/UNPIVOT* 연산을 정형적으로 관계 대수로 제안하고 다른 연산들과 함께 사용될 때 질의를 최적화하는 방법을 소개하였다. Chen 등[10]에서는 다중 *PIVOT* 연산의 처리를 위한 *GPIVOT* 연산에 대해 소개하였다. 또한, 최근 상용 RDBMS들은 피벗 테이블과 같은 수평 테이블을 생성할 수 있도록 서브 쿼리(Sub Query)나 *CASE*문을 제공하고 있거나 유사한 기능을 추가하고 있다[7, 8].

본 논문에서는 수직 테이블을 디스크 상에 저장하고 상용 RDBMS에서 제공하는 *PIVOT* 연산을 이용하여 가상의 수

평 뷰 테이블에 대한 질의를 변환할 때 이를 최적화하는 방법을 연구하였다. *PIVOT* 연산은 수직 테이블을 수평 테이블로 변환하기 위해 개발된 연산으로, 수평 뷰 테이블에 대한 질의를 간단하게 수직 테이블에 대한 질의로 변환하는데 사용할 수 있다. 하지만 *PIVOT* 연산을 사용한 변환 질의는 여러 가지 방법으로 처리할 수 있으며, 이들 각 방법들마다 수행하는 시간이 서로 다르다. 따라서 이 논문에서는 수평 뷰 테이블에 대한 사용자 질의를 수직 테이블에 대한 질의로 변환할 때, 이를 처리할 수 있는 여러 가지 방법들을 제시하였으며, 이들 각 방법들에 대한 수행 시간을 비교 및 평가하였다. 이를 토대로 수평 테이블에 대한 질의를 수직 테이블에 대한 질의로 변환할 때 이를 최적화하는 방법을 제시하였다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문의 이론적 배경이 되는 관련 연구에 대해 소개한다. 3장에서는 행과 열을 변경하는 여러 구문과 *PIVOT* 연산을 소개하고 *PIVOT* 연산을 이용하여 프로젝션, 셀렉션, 조인 연산을 처리하는 여러 방법들을 고려하며, 4장에서는 이들 방법들을 다양한 실험을 통해 분석한다. 마지막으로, 5장에서는 본 연구의 결론을 맺는다.

2. 관련 연구

OLAP 시스템에서 사용되는 다차원 데이터는 여러 차원 애트리뷰트들과 이에 대응하는 측정 값들로 구성된다. 참고문헌 [4-6]에서는 이 다차원 데이터를 사용자에게 편리하게 보여주는 방법으로 마이크로소프트 엑셀의 피벗 테이블이나 통계 분야의 교차 테이블 형태로 표현하였다. 이러한 표현 방법은 테이블 열의 이름에 측정 값이 오게 되므로 많은 수의 열을 포함하는 수평 형태의 테이블이 된다. 이러한 수평 테이블을 기반으로 이들 연구에서는 SQL 문을 확장하여 관계 데이터베이스에서 스프레드시트 형태의 테이블을 정의하고 조작할 수 있도록 하였다. 그래서 사용자들이 OLAP 분석 기능을 좀 더 편리하게 조작할 수가 있다. 하지만, 이들 연구에서는 수평 테이블을 직접 저장하고 효율적인 질의하는 데 초점을 두었을 뿐 저장 방법에 대해서는 소개하지 않았다.

Agrawal 등[3]은 e-비즈니스 분야에서 수평 테이블을 소개하였으며, 수평 테이블을 전형적인 테이블 구조로 처리할 때의 문제점을 소개하였다. 이러한 문제점은 수평 뷰 테이블을 저장할 경우, 상용 DBMS은 테이블에 저장할 수 있는 최대 열의 개수를 1,024개로 제한하고 있다는 것이다[7, 8]. 따라서 관계형 데이터베이스를 이용하여 다차원 데이터를 저장하기 위해서는 많은 수의 열을 포함하는 수평 테이블 구조가 아닌 다른 형태의 스키마 구조로 저장하는 방법이 필요하다. 따라서, 수평 테이블을 효율적으로 저장하는 방법은 수직 테이블을 사용하여 실제로 존재하는 측정값에 대하여 하나의 행을 갖도록 표현하는 방법이 있다[3, 11]. 이들 연구에서는 사용자에게 편리한 수평 뷰 테이블을 제공하고, 수평 뷰 테이블에 대한 질의들을 수직 테이블에 대한 질의로 변환하여 처리한다. 그러므로 사용자들에게 저장 테이블의 구조를 숨기고 편리한 수평 뷰 테이블을 제공할 수가 있는 것이다. 수평 뷰 테이블을 효율적으로 저장하고 처리하기 위해, Lakshmanan 등[11]은 네 가지의 재구성 연산(*unfold, fold, split, unite*)을 제안하고 이를 관계 대수로 대응시키는 방법을 제시하였다. 이들 연산 중에서, *unfold*와 *fold* 연산은 수평 뷰 테이블을 수직 테이블로, 혹은 반대로 변환하는 연산이다. 이러한 연구를 기반으로, Agrawal 등[3]은 관계 대수를 이용하여 *unfold*와 *fold* 연산을 정의하고, 수평 테이블을 수직 테이블로 변환하여 저장하고 질의하는 체계적인 방법을 제안하였다. 이러한 연구에서는 수직 테이블을 저장 방법으로 사용함으로써, 수평 테이블이 갖는 널 값(null value)을 저장하지 않아 저장 효율을 높임과 동시에 성능을 향상시킴을 보였다. 하지만, 이러한 방법은 전형적인 관계 대수 연산을 사용하여 변환하기 때문에 변환 과정이 상당히 복잡하며, 최근 상용 DBMS에서 제공하는 최적화된 *PIVOT* 연산의 장점을 활용하지 못하는 문제점이 있다.

다음으로는, *PIVOT* 연산에 대한 여러 연구들이 있다[9, 10]. Cunningham 등[9]에서는 행과 열의 역할을 변경하여 수직 테이블에서 수평 테이블로, 혹은 수평 테이블에서 수직 테이블로 변경하는 *PIVOT/UNPIVOT* 연산을 새롭게 정의하였고, 질의 성능을 개선하기 위해서 이를 다른 대수 연산자들과 함께 사용할 때 최적화하는 방법에 대해 연구하였다. 이 연산을 확장하여, Chen 등[10]에서는 *PIVOT* 연산을 일반화하여 다중 *PIVOT* 연산의 처리를 위한 *GPIVOT* 연산에 대해 소개하였고, 이를 사용하여 생성한 수평 뷰 테이블에 대한 구체화된 뷰(materialized view)를 점진적으로 갱신하는 방법을 제시하였다. 하지만, 이들 연구에서 정의한 PIVOT 연산은 최근 상용 DBMS에서 사용하는 것과는 다른 방식으로 정의하고 있어 실제 DBMS가 제공하는 *PIVOT* 연산을 활용하기 곤란하며, *PIVOT* 연산을 수행하는 방법에 대한 최적화는 고려하지 않았다.

기존의 상용 RDBMS은 기본적으로 수직 테이블을 다루기 위해 만들어졌으며, 이를 행과 열의 역할을 변경한 수평 테이블 형태의 결과를 제공하기 위해서는 여러 서브 쿼리(sub-query)나 *CASE* 문을 사용하여 복잡한 SQL문으로 표현할 수 있다[7,8]. 이렇게 할 경우 질의문이 매우 복잡해지는 문제점이 있다. 이러한 문제점을 해결하기 위해 최근의 상용 RDBMS인 SQL Server 2005[7]에는 질의문의 FROM 절에서 사용되는 두 개의 연산자인 *PIVOT* 및 *UNPIVOT*을 최근 개발되어 제공하고 있다. 이는 시스템 내부의 *PIVOT* 연산자를 이용하여 행을 열로 회전시키는 과정에서 집계를 계산하고 열을 기준으로 출력 테이블을 유용한 형태로 표시한다. *UNPIVOT* 연산자는 반대 연산으로, 즉 열을 행으로 바꾸어 결과를 표시한다.

본 연구에서는 수평 뷰 테이블에 대한 질의를 *PIVOT* 연산을 사용하여 수직 테이블에 대한 질의로 변환하며, 이 변환된 질의를 처리할 수 있는 다양한 처리 방법을 제시하고 이를 서로 비교 및 평가하며, 이를 기반으로 상용 데이터베이스 시스템에서 질의 변환을 최적화하는 방법을 연구한다.

3. PIVOT 연산의 소개 및 최적화 방법 고려

전형적인 관계형 데이터베이스에서 수직 테이블의 행별로 저장된 값을 수평 뷰 테이블로 보여 주기 위해 여러 가지 방법을 사용할 수 있다. 그 대표적인 방법으로 서브 쿼리(Sub query)를 이용하는 방법과 *CASE* 문을 이용하는 방법이 있다. 우선, 서브 쿼리의 방법은 아래 그림 3과 같이 동일한 테이블 간에 조인하는 여러 자체 조인을 갖는 서브 쿼리를 이용하여 수평 형태의 결과를 생성하도록 하는 방법이다. 그림에서 보는 바와 같이 이 방식은 질의가 매우 복잡해진다는 문제점이 있다.

다른 방법으로는 *CASE* 문을 사용하는 방법이 있는데, 그림 4는 일반적인 SQL문에서 *CASE* 문을 사용하여 수평 테이블을 생성하는 질의문의 예를 나타낸다. 이 질의에서 사용되는 *CASE* 문은 여러 조건을 평가하고 각 조건에 대해 단일 값으로 반환하는 데 사용된다.

이러한 기존의 변환 방법인 서브 쿼리와 *CASE* 등의 구문들은 원하는 행의 수가 많을 경우 사용해야 할 구문이 복잡해지는 경향이 있다. 하지만 마이크로소프트에서 최근 제공되는 SQL Server 2005[7]의 *PIVOT* 연산을 이용할 경우 간결하게 이를 구현할 수 있다. 다음 그림 5는 이 *PIVOT* 연산을 사용하여 간결하게 변환한 질의의 예를 보여주고 있다. 이 예제 구문에서 *FROM* 절의 *ProductTable*은 피벗하고자 하는 소스 테이블을 의미하고, *PIVOT* 절의 *IN* 연산은 소스 테이블의 열 *Product_name*에 속해 있는 값들(*TV, Radio, Computer*)을 평가하여 수평 뷰 테이블의 열 이름으로, *Product_value*는 수평 뷰 테이블의 열의 측정값으로 변경하기 위해 사용된다. 이렇게 *PIVOT*을 사용하면 수직 테이블에서 수평 테이블 뷰를 간편하게 제공할 수 있다.

```
SELECT Oid,
    (SELECT Product_value FROM ProductTable AS PT1
      WHERE Product_name = 'TV' AND PT.Oid = PT1.Oid) AS TV,
    (SELECT Product_value FROM ProductTable AS PT2
      WHERE Product_name = 'Radio' AND PT.Oid = PT2.Oid) AS Radio,
  (SELECT Product_value FROM ProductTable AS PT3
      WHERE Product_name = 'Computer' AND PT.Oid = PT3.Oid) AS Computer
FROM ProductTable AS PT
GROUP BY Oid
```

(그림 3) 서브 쿼리를 이용한 수평 테이블 생성 질의문

```
SELECT Oid,
  SUM(CASE WHEN Product_name= 'TV' THEN Product_value ELSE null END) AS TV,
  SUM(CASE WHEN Product_name= 'Radio' THEN Product_value ELSE null END) AS Radio,
  SUM(CASE WHEN Product_name= 'Computer' THEN Product_value ELSE null END) AS Computer
FROM  ProductTable
GROUP BY Oid
```

(그림 4) CASE 문을 사용한 수평 테이블 생성 질의문

```
SELECT *
FROM ProductTable
PIVOT (SUM(Product_value) FOR Product_name IN (TV, Radio, Computer)) AS PVT
```

(그림 5) PVIOT을 사용한 수평 테이블 생성 질의문

3.1 질의 처리 변환에 대한 대수적 표현

본 절에서는 질의의 내부 표현(질의 트리 이용)을 사용하여 변환 질의에 대한 여러가지 수행 방법을 제시하고 실행 시 성능을 향상시킬 수 있는 여러 최적화 방법에 대해 논의한다. 우선, 질의 최적화 방법을 논하기 위해 다음 표 1과 같이 주요 표기법을 이용한다. 수평 뷰 테이블 *WT*는 스키마 (*Oid*, A_1, A_2,…, A_n)을 나타내며 *Oid*는 식별자이고 A_1, A_2,…, A_n는 열의 이름으로 표현한다. 그리고 수직 테이블 *NT*는 스키마 (*Oid, A, M*)로 나타내며 *Oid*는 식별자이고 *A, M* 은 각각 열의 이름으로 구성된다.

수평 테이블은 OLAP 사용자 뷰로 사용되며, 수직 테이블은 수평 테이블을 효과적으로 저장하는데 사용된다. 사용자들의 질의는 수평 테이블에 대해 주어지며, 이 수평 뷰 테이블의 질의를 실제로 처리하기 위해서는 수직 테이블에 대한 질의로 변환하는 과정이 필요하다.

수평 뷰 테이블은 앞에서 설명한 바와 같이 *PIVOT* 연산을 사용하여 간단하게 수직 테이블에 대한 질의로 변환할 수 있다. 이에 대해서는 Lakshmanan 등 [10]에서 명확하게 제시하였는데, 수평 테이블 ***WT***와 수직 테이블 ***NT***사이에는 다음 식(1)이 성립한다. 즉, 수직 테이블 ***NT***에서 기준이 되는 조건의 열 ***A***의 값이 $\boldsymbol{A_1, A_2, \ldots, A_n}$인 데이터 값을 추출하고 열들의 목록 ***Oid***, ***M***을 지정한 후, 수평 뷰 테이블로 표현하기 위해 완전 외부 조인(⟗) 연산을 이용하면 수평 뷰 테이블 ***WT***이 생성된다.

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

위 식(1)의 우변 연산에 있는 완전 외부 조인(⟗)식을 간략하게 표현하기 위해 도입된 연산이 *PIVOT* 연산이다. 따라서 수평 테이블에 대한 질의는 아래와 같이 *PIVOT* 연산을 사용하여 관계 대수식으로 변환될 수 있다. 다음 절에서는 이 *PIVOT* 연산을 사용한 변환 질의를 효과적으로 처리할 수 있는 여러 방법들을 제시한다.

$$WT \xrightarrow[Transformation]{Query} PIVOT\left[\begin{matrix} A \text{ in} \\ (A_1, A_2, \ldots, A_n) \end{matrix} \mathcal{F}_{Function\ \exists M}(NT) \right] \quad (2)$$

〈표 1〉 주요 표기법

기 호	정 의
WT	수평 형태의 뷰 테이블
NT	수직 형태의 저장 테이블
Oid	튜플 식별자
A, M	저장 테이블 *NT* 내의 애트리뷰트들
A_i	저장 테이블 *NT*의 *A*애트리뷰트에 올 수 있는 값들 ($1 \le i \le n$) 뷰 테이블 *WT*의 애트리뷰트 이름들 ($1 \le i \le n$)
M_i	저장 테이블 *NT*의 *M*애트리뷰트에 올 수 있는 값들 ($1 \le i \le n$)

3.2 프로젝션 연산의 처리 방법

프로젝션 연산은 수평 뷰 테이블로부터 관심있는 일부 열만을 선택하는 연산이다. 그림 6에서는 수평 뷰 테이블 WT와 수직 테이블 NT에서 프로젝션 연산을 나타내고 있다. 그림 6의 좌변 연산은 수평 뷰 테이블 WT에서 적용한 프로젝션 연산의 일반적인 형태를 표현하며, 수직 테이블 NT에서 $PIVOT$ 연산을 이용한 프로젝션 연산은 우변 연산으로 명시할 수 있다. 우변 연산에서는 수직 테이블 NT에서 열 A의 전체 값 (A_1, A_2,…, A_n)을 추출하고 계산 가능한 특정 열 M을 지정한 후, 집계 함수를 적용한다. 그 다음에, $PIVOT$ 연산을 적용하면 수평 뷰 테이블 형태로 되며, 여기에 프로젝션 연산을 적용하면 좌변 연산의 결과와 동일하게 된다.

이렇게 변환된 수직 테이블에 대한 연산을 처리하는 데는 여러가지 방법이 있을 수 있는데, 다음 그림 7은 이 $PIVOT$ 연산을 처리하는 여러 방법들을 질의 트리로 나타낸 것이다. 우선, 그림 7(a)는 $FROM$ 절에 명시한 수직 테이블 NT를 나타내고 $WHERE$ 절의 조건이 적용되며, 그 다음에 $PIVOT$ 연산을 적용한 과정을 보이고 있다. 그림 7(b)는 그림 7(a)의 유사한 과정으로 수직 테이블 NT에서 합집합 연산을 적용한 후에 $PIVOT$ 연산을 적용한 과정을 보이고 있다. 그림 7(c)에서는 테이블 NT에서 셀렉션 연산을 적용하고 in 연산을 적용한 후, $PIVOT$ 연산을 적용한 과정이다. in 연산자는 열 A의 목록 안의 값들 중에 하나 이상과 일치하는 지 여부를 파악하는 데 사용되는 연산자이다. 그림 7(d)도 그림 7(c)에서와 유사한 과정으로 합집합 연산을 적용하여 $PIVOT$ 연산을 적용한 과정을 보이고 있다. 그림 7(c)와 7(d)에서 프로젝션 연산을 수행 중에 in 연산자를 사용한 것은 후에 논하는 셀렉션 연산에 필요하다. 이는 연산 결과에 포함하는 하나의 열로 이루어진 다수의 튜플이 반환하려면 Oid가 필요하기 때문이다.

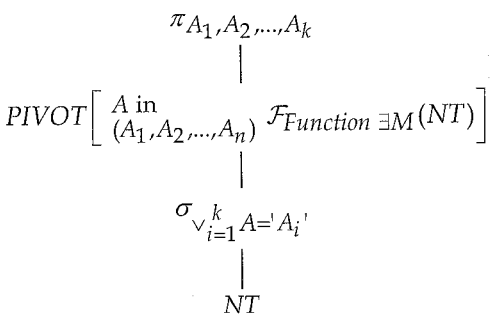
$$\pi_{A_1,A_2,\ldots,A_k}(WT) = \pi_{A_1,A_2,\ldots,A_k}(PIVOT_{A \text{ in } (A_1,A_2,\ldots,A_n)}\,\mathcal{F}_{Function}\,\exists M(NT))$$

(그림 6) 프로젝션을 위한 $PIVOT$ 연산 (Pivot_All)

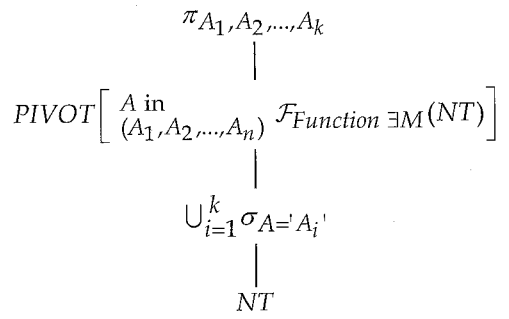
3.3 셀렉션 연산의 처리 방법

다음으로, 셀렉션 연산은 논리식에서 선택 조건을 만족하는 튜플들의 집합을 선택하는 데 사용한다. 따라서, 일반적인 수평 뷰 테이블 WT에서 특정 조건에 만족하는 튜플들을 선택하려면 그림 8의 좌변 연산과 같이 셀렉션 연산을 이용하여 각 조건을 기술할 수 있다. 또한, 수직 테이블 NT에서 질의를 변환하여 처리할 경우 우변 연산과 같이, $PIVOT$ 연산을 적용하고 그 다음에, 셀렉션 연산을 명시하면 좌변 연산과 동일한 결과값으로 표현할 수 있다.

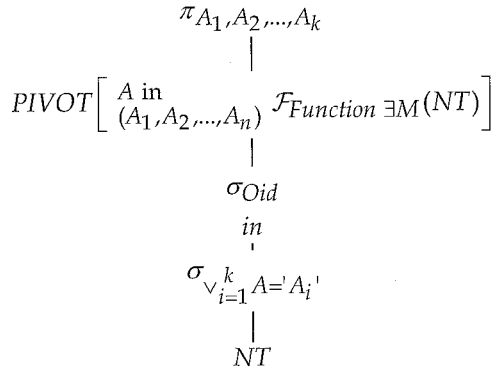
이 셀렉션 연산에 대한 $PIVOT$ 연산도 여러 가지 방법으로 처리할 수 있는데, 그림 9는 이들 방법들을 질의 트리로 나타낸 것이다. 그림 9(a)는 $WHERE$ 절에 명시한 셀렉션 연산의 조건을 명시하고, 조건에 해당하는 열을 기준으로 in 연산자를 사용하여 관련 튜플을 추출하기 위해서 Oid를 사



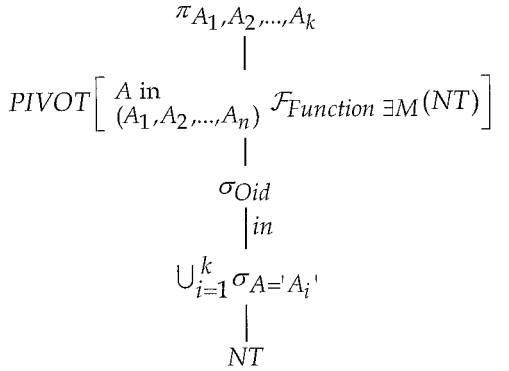
(a) 'OR'을 이용한 $PIVOT$연산 (Pivot_OR)



(b) 'Union'을 이용한 $PIVOT$연산 (Pivot_Union)

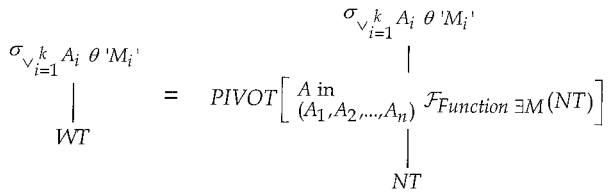


(c) 'in'과 'OR'을 이용한 $PIVOT$연산 (Pivot_InOR)

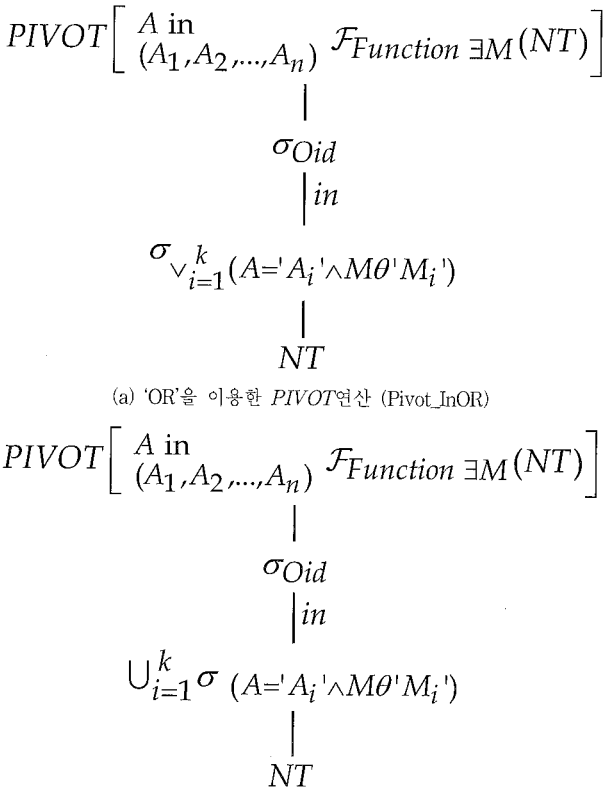


(d) 'in'과 'Union'을 이용한 $PIVOT$연산 (Pivot_InUnion)

(그림 7) 프로젝션을 위한 $PIVOT$ 연산의 처리 방법들



(그림 8) 셀렉션 연산을 위한 *PIVOT* 연산 (Pivot_All)



(a) 'OR'을 이용한 *PIVOT*연산 (Pivot_InOR)

(b) 'Union'을 이용한 *PIVOT*연산 (Pivot_InUnion)

(그림 9) 셀렉션 연산을 위한 *PIVOT* 연산의 처리 방법들

용한다. 그 다음에, *PIVOT* 연산을 적용한다. 그림 9(b)는 그림 9(a)와 동일한 과정이며, 합집합 연산을 이용하여 *PIVOT* 연산을 적용한 질의 트리를 나타낸다.

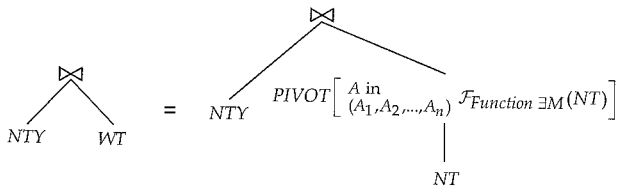
3.4 조인 연산의 처리 방법

조인 연산은 ⋈으로 표기하고, 두 테이블로부터 관련된 튜플들을 결합하여 하나의 튜플로 생성할 때 이용되는 연산이다. 이러한 조인 연산을 논의하기 위해 표2의 보조 표기법을 이용한다. 여기서, 테이블 *NTY*은 전형적인 형태의 테이블을 명시하며, 테이블 *NTY*의 스키마는 (*Oid*, Y_1, Y_2,…, Y_n)을 나타낸다. 여기서, *Oid*는 식별자이고, *Y*는 열 이름으로 값 Y_1, Y_2,…, Y_n으로 구성한다.

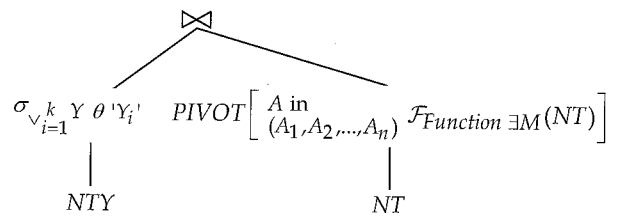
조인 연산의 일반적인 표현은 그림 10의 좌변 연산으로 나타낼 수 있다. 이는 뷰 테이블 *WT*과 전형적인 테이블 *NTY*와의 조인 연산을 나타내며, 테이블 *NTY*와 수직 테이블 *NT*에서 *PIVOT* 연산을 적용한 결과의 조인 연산은 좌변 연산으로 표현할 수 있다.

〈표 2〉 보조 표기법

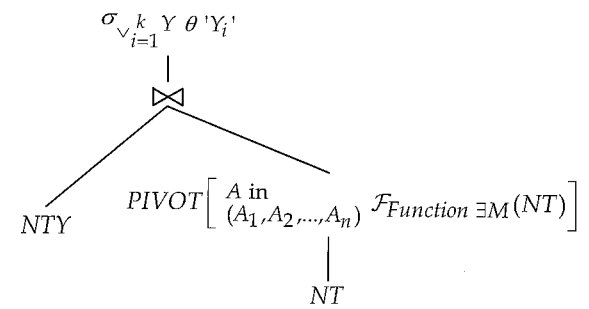
기 호	정 의
NTY	전형적인 저장 테이블
Oid	튜플 식별자
Y	저장 테이블 *NTY*내의 애트리뷰트들
Y_i	저장 테이블 *NTY*의 *Y* 애트리뷰트에 올 수 있는 값들 ($1 \le i \le n$)



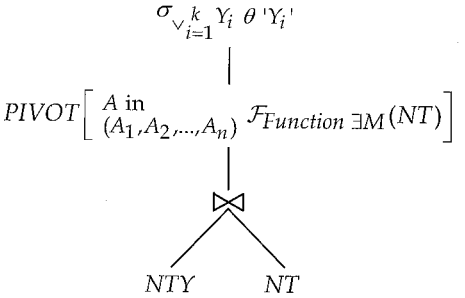
(그림 10) 두 테이블의 조인 연산을 위한 PIVOT 연산



(a) *NTY* 테이블에서 실렉션 연산 후 조인 연산 적용 (Join_SelectNTY_PivotT)



(b) *NTY* 테이블과 *PIVOT* 연산을 적용한 테이블과의 조인 연산 적용 (Join_NTYPivotT)



(b) 두개의 테이블을 조인 후 *PIVOT* 연산 적용 (Select_JoinAll)

(그림 11) 조인 연산을 위한 PIVOT 연산의 처리 방법들

조인 연산은 질의 처리에서 가장 실행시간이 많이 소요되는 연산 중의 하나이다. 여기서, 조인 연산은 자연 조인이나 동등 조인을 의미한다. 이와 같은 조인 연산은 다른 연산과 달리 많은 실행시간이 소요되므로, 조인 연산에 대한 최적화를 다른 연산들보다 신중히 고려해야 한다. 그림 11에서

는 일반적인 테이블 *NTY*와 조인하는 과정을 고려한 여러 질의 트리를 나타낸다. *PIVOT* 연산을 적용하는 수평 뷰 테이블에 대해서는 앞서 논했던 프로젝션 연산과 셀렉션 연산에 대하여 최적화를 고려한 질의 트리를 참고한다. 여기에서는 전형적인 테이블 *NTY*에 셀렉션 연산의 조건을 명시하는 것으로 고려한다.

그림 11에서는 3가지 최적화를 고려한 접근 경로를 질의 트리로 보이고 있다. 그림 11(a)에서는 *FROM* 절에 전형적인 테이블 *NTY*을 명시하고, WHERE 절의 셀렉션 연산의 조건들이 적용되며, 그 다음에 *PIVOT* 연산이 적용된 수직 테이블 *NT*의 결과를 조인한 질의 트리를 나타낸다. 그림 11(b)는 테이블 *NTY*와 수직 테이블 *NT*에서 적용한 *PIVOT* 연산의 결과를 조인한 과정을 나타내며, 그림 11(c)는 테이블 *NTY*와 *NT*가 조인한 후에 *PIVOT* 연산을 적용한 질의 트리를 나타내고 있다. 그림 11(b)와 11(c)의 질의 트리는 경험적 규칙 상 전체 테이블들이 조인 연산을 적용한 후에 *PIVOT* 연산과 셀렉션 연산이 이루어지기 때문에 실행시간이 많이 소요될 것이다.

4. 성능 분석 및 평가

본 장에서는 *PIVOT* 연산을 적용한 셀렉션, 프로젝션, 조인 연산에 대하여 앞서 논의한 여러 수행 방법들에 대한 성능을 비교 평가한다. 질의 최적화에 대한 여러 질의 트리의 접근 경로를 분석하기 위해서 실험은 다양한 관점으로 여러 매개변수 통해 수행하였다. 또한, 실험에 대한 분석 내용으로는 최적화 실행을 고려하기 위해 중요한 요소로 작용할 수 있는 인덱스 사용에 대해 분석하고, 기본적인 연산(프로젝션, 셀렉션)과 조인 연산의 효율을 향상시킬 수 있는 방법을 실험을 통해 분석한다.

4.1 실험 환경

모든 실험은 물리적 메모리 1GB와 1.70GHz의 CPU를 사용하는 Intel Pentium 4이며, 윈도우 환경 Windows XP에서 상용 데이터베이스 시스템인 Microsoft SQL Server 2005 Beta 2 상에 수행하였다. 또한, 본 실험의 요소는 Agrawal 등[3]의 데이터 세트(dataset)와 같은 매개변수를 이용한다. 우선 수평 형태의 뷰 테이블의 스킴을 생성하고 널 값의 비율인 90%, 95%에 따라 랜덤하게 데이터 값을 입력한 후, 수직 저장 테이블의 스킴으로 동일한 데이터 값을 변환하여 저장하였다. 테이블의 생성은 수평 뷰 테이블을 기준으로 열의 수와 행의 수(*#col*×*#rows*)인 데이터 세트 200×100K, 400×50K, 800×25K, 1000×25K으로 구성하였다. 다음으로, 수평 뷰 테이블 *WT*과 수직 저장 테이블 *NT*에 대한 스키마는 다음과 같이 생성하였다.

$$WT(Oid \text{ integer}, A1 \text{ float}, A2 \text{ float}, ..., An \text{ float})$$

$$NT(Oid \text{ integer}, A \text{ char}(4), M \text{ float})$$

이러한 스키마에 대해 수직 테이블 *NT*의 열 *M*에 대한 데이터 타입을 *float*형으로 설정하였다. 이런 이유는 *PIVOT* 연산시 행을 열로 변경할 때 집계로 계산하기 때문에 수치 데이터 형으로 변수를 선언한 것이다. 또한, 수평 뷰 테이블 *WT*에서도 데이터 타입을 *float*형으로 변수를 선언하였다.

4.2 성능평가 결과

최적화를 이루기 위한 튜닝의 방법으로 특정 열에 인덱스를 어떻게 구성할 것인가는 정책적인 측면에서 고려될 사항이다. 따라서, 데이터 세트 200×100K, 400×50K, 800×25K, 1000×25K에 대한 수직 테이블 *NT*의 스키마에서는 열 *O*, *A*에 클러스터 인덱스를 각각 지정하였고, 열의 개수는 5, 10, 20개를 임의로 선정하여 실험을 수행하였다.

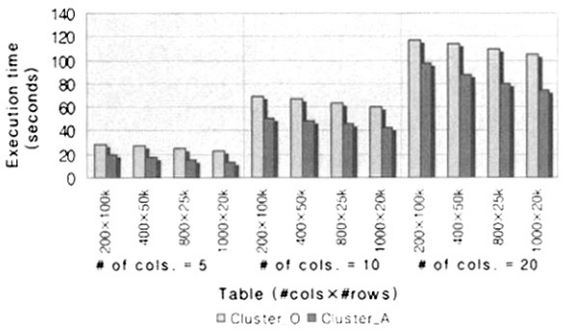
그림 12에서는 인덱스에 대한 실험 결과를 보이고 있다. 수직 테이블에서 수평 뷰 테이블로 표현하기 위해 *PIVOT* 연산을 이용하였고 실험 결과, 클러스터 인덱스를 부여한 열 *A*는 열 *O*보다 계산 비용이 적게 소요된다. 이는 수평 뷰 테이블의 해당 열을 추출하기 위해 *PIVOT* 연산 내의 조건절에 열 *A*를 사용하게 되므로 인덱스의 직접적인 영향이 미치게 되며, 계산 비용의 차이가 나게 되는 것이다.

다음 실험에서는 프로젝션, 셀렉션, 조인 연산 등에 대하여 성능을 비교한다. 이러한 연산들의 질의 처리 효율을 높이기 위해 열 *Oid*와 *Key*를 기준으로 각각 넌클러스터링과 클러스터링 인덱스를 설정하였다. 또한 수직 테이블에서 수평 뷰 테이블 형태의 질의로 변환하기 위해 앞서 논했던 최적화 방법을 고려한 여러 질의 트리들을 실험하였다.

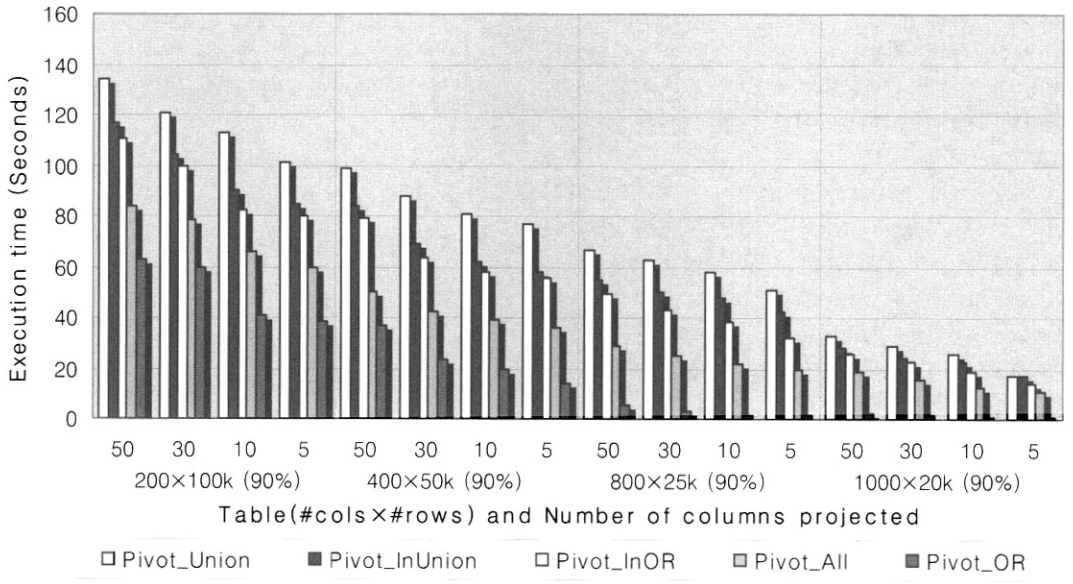
4.2.1 프로젝션 연산의 성능 분석

프로젝션 연산의 실험은 앞서 논의했던 처리 방법들의 질의 트리 *Pivot_All*, *Pivot_Union*, *Pivot_OR*, *Pivot_InUnion*, *Pivot_InOR*를 이용하였으며, 데이터 세트 200×100K, 400×50K, 800×25K, 1000×25K에서 널 비율이 각각 90%, 95%로 설정하여 수행하였다. 실험의 변경요인은 열의 수를 5, 10, 30, 50개로 변경하여 실험을 수행하였다.

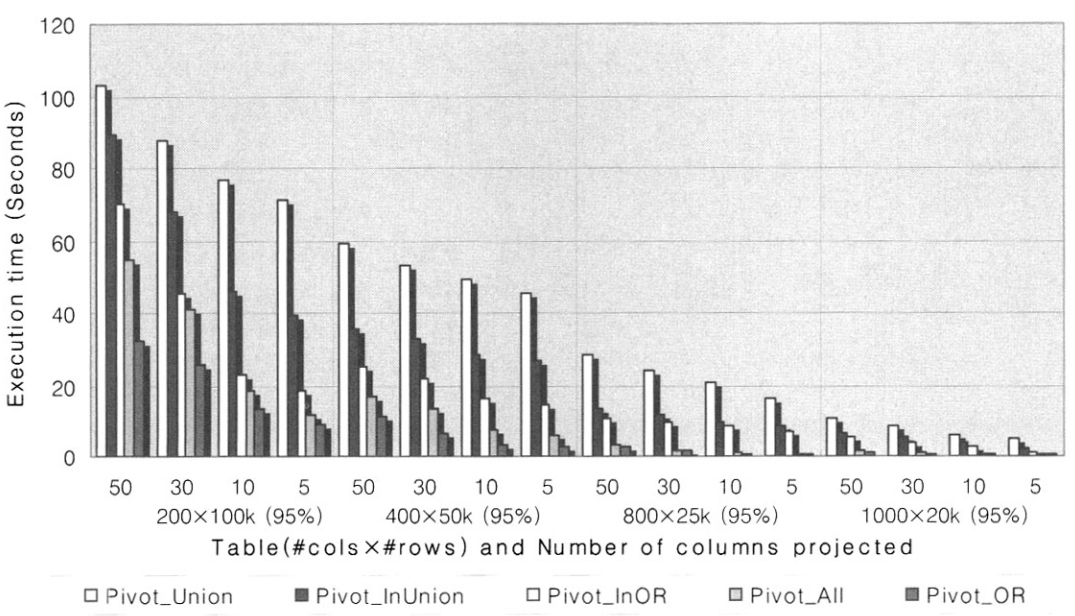
그림 13과 14에서는 프로젝션 연산에 대한 여러 질의 트리의 실험을 결과를 보이고 있다. 여러 질의 트리 중 *Pivot_OR*은 계산 비용이 가장 낮은 것으로 나타낸다. 다른 질의 트



(그림 12) 클러스터 인덱스 비교 (# of columns)



(그림 13) 널 비율 90%인 프로젝션 연산의 성능 비교

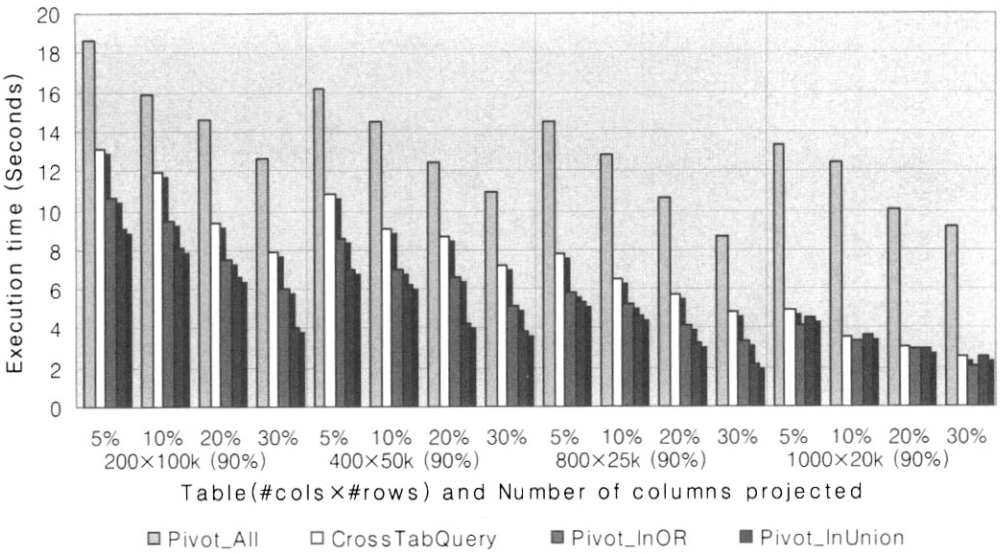


(그림 14) 널 비율이 95%인 프로젝션 연산의 성능 비교

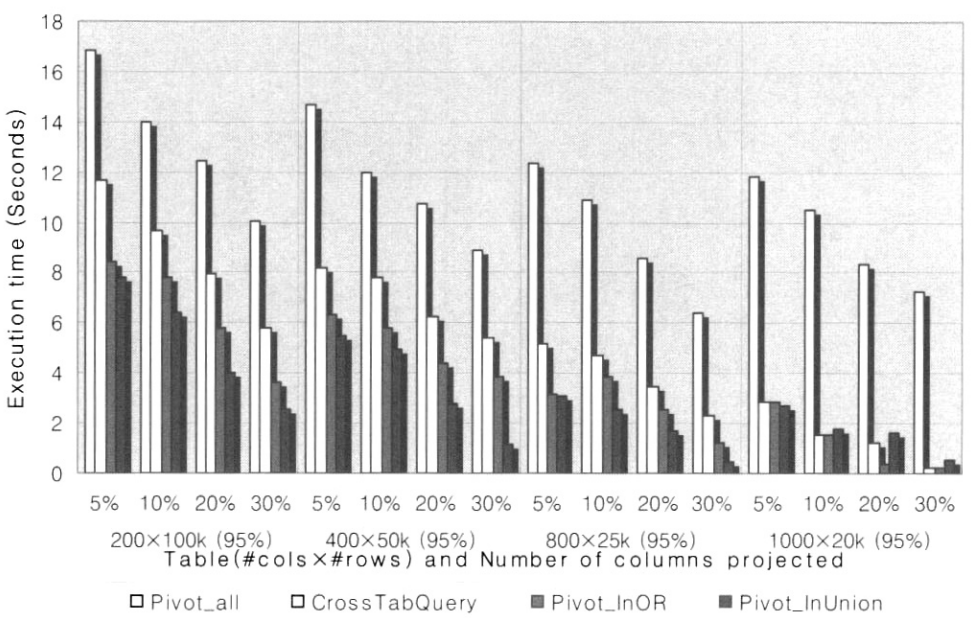
리들 (*Pivot_Union*, *Pivot_InUnion*, *Pivot_InOR*)은 해당하는 열의 개수가 증가할수록 전체 테이블의 스캔 횟수가 증가하기 때문에 비교적 높은 계산 비용이 소요되며, 테이블의 스캔 횟수가 적은 질의 트리 *Pivot_All*이 *Pivot_OR* 보다 계산 비용이 높은 원인은 인덱스의 영향을 미치는 조건절에 해당하는 열이 포함되지 않고, 전체 테이블에 대하여 *PIVOT* 연산을 적용한 후에 프로젝션 연산이 이루어지므로 계산 비용이 높다. 따라서 프로젝트 연산을 처리하는 데는 *Pivot_OR* 방식이 가장 효과적인 수행 방법으로 평가된다.

4.2.2 셀렉션 연산의 성능 분석

셀렉션 연산에 대한 실험의 요소는 프로젝션 연산에 대한 실험의 요소와 마찬가지로 동일한 데이터 세트와 널 비율을 이용하였다. 실험에 포함되는 연산으로는 기존에 행과 열을 변경하는 데 사용한 *CASE* 문과 서브 쿼리를 이용하였고, 질의 트리 *Pivot_InUnion*, *Pivot_InOR*를 이용하여 비교 실험하였다. 또한, 실험은 질의에 포함하는 열의 개수를 10개로 설정하고 조건의 선택률은 각각 5%, 10%, 20%, 30% 순으로 변경하여 수행하였다. 셀렉션 연산에 대한 질의는 수



(그림 15) 널 비율이 90%인 선택률과 컬럼 수에 대한 셀렉션 연산의 성능 비교



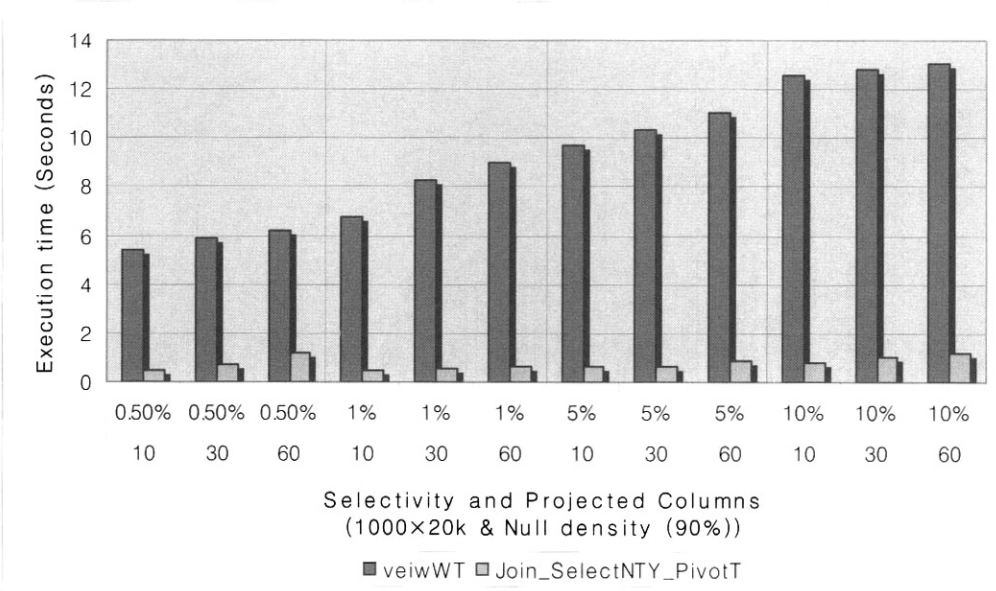
(그림 16) 널 비율이 95%인 선택률과 컬럼 수에 대한 셀렉션 연산의 성능 비교

평 뷰 테이블에 적용되는 다음과 같은 구문을 이용하였다.

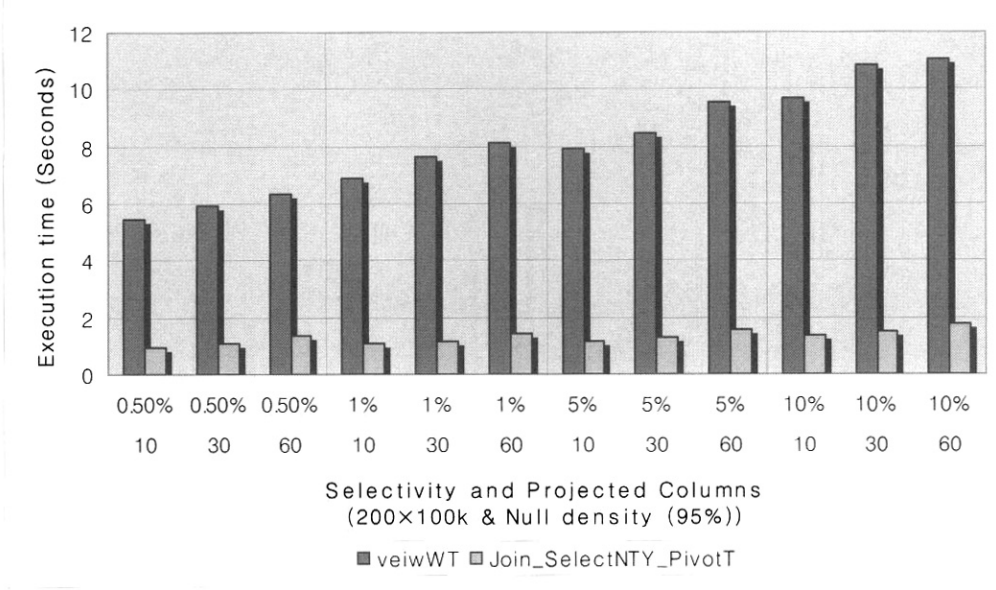
SELECT *A10, A100* FROM *WT*

WHERE *A20 = 1.0* OR *A200 = 10.0*

그림 15과 16에서는 셀렉션 연산에 대한 여러 구문들에 대한 실험 결과를 보이고 있다. 여기서, 기존 연산인 서브 쿼리를 이용한 구문은 계산 비용이 다른 구문과는 다르게 상당히 높기 때문에 본 실험에서 제외시켰다. 또한, 실험에서 서브 쿼리와 *CASE* 구문을 사용할 때 문제점이 발생하게 되는 데, 이는 셀렉션 조건에 대한 결과로 어떤 튜플이 전체 널 값을 포함할 경우 그 튜플은 출력되지 않는 문제점을 갖는다. 실험결과, 전체 테이블을 스캔 후 *PIVOT* 연산을 적용하여 셀렉션 조건을 이용한 질의 트리 *Pivot_All*는 *CASE* 문을 이용한 쿼리 구문보다 높은 계산 비용을 나타내고 있다. 그리고 질의 트리 *Pivot_InUnion*의 경우는 계산 비용이 가장 적은 폭으로 나타났다. 그외 다른 구문 및 질의 트리들은 선택률에 대한 변화요인에 따라 계산비용이 큰 폭을 나타내고 있다. 그리고 질의 트리 *Pivot_InOR*이 *Pivot_InUnion*의 경우보다 계산 비용이 높은 원인으로는 셀렉션 연산의



(그림 17) 널 비율 90%인 조인 연산의 성능 비교



(그림 18) 널 비율 95%인 조인 연산의 성능 비교

조건 절에 많은 열의 수를 명시할 때마다 그 조건에 포함되어 있는 해당 열을 접근해야 하므로 많은 계산 비용이 소요된다. *Pivot_InUnion* 방식이 셀렉션 연산을 실행하는 가장 효과적인 방법으로 평가된다.

4.2.3 조인 연산의 성능 분석

조인 연산은 관계 데이터베이스에서 가장 기본적이고 중요한 연산이다. 조인 연산은 논리적인 관계만으로도 원하는 정보를 서로 연결하여 참조할 수 있다. 본 실험을 위하여, 전형적인 테이블 *Year* (그림 2 참고)을 이용하여 데이터 세트 200×1000K인 수평 뷰 테이블 *WT*와 계산비용이 가장 적은 질의 트리 *Pivot_InUnion*을 이용한 최적화된 *Join_SelectNTY_PivotT* 연산으로 각각 조인하여 수행시간을 측정하였다. 실험 요소는 널 값의 비율은 각각 90%, 95%이고 선택률은 0.5%, 1%, 5%, 10%이며, 열의 개수를 10, 30, 60 순으로 변경하면서 다양한 관점으로 수행하였다.

그림 17과 18에서는 전형적인 테이블과 각각 수평 뷰 테이블과 수직 테이블의 조인 연산에 대한 실험 결과를 보이고 있다. 실험 결과, 최적화 질의 트리 *Join_SelectNTY_PivotT*는 선택률과 열의 개수에 따른 변환 요인에 따라 수평 뷰 테이블 *WT*에 대한 계산 비용보다 아주 작은 폭을 나타내고 있다. 수평 뷰 테이블 *WT*에 대한 질의에 대하여 계산 비용이 현저히

큰 폭으로 나타난 이유는 조건에 해당하는 해당 튜플을 검색하기 전에 전체 테이블을 스캔하는 계산 비용이 많이 소요되기 때문이다. 따라서, 본 실험 결과로 열의 개수와 선택률이 점증적으로 높아질수록 최적화 질의 트리 *Join_SelectNTY_PivotT*의 성능이 더욱 향상됨을 알 수 있다.

5. 결 론

사용자들은 데이터 웨어하우스에 저장된 방대한 데이터를 OLAP 연산을 이용하여 다양한 관점으로 정보를 분석하기 원한다. 하지만 방대한 데이터를 전형적인 테이블 구조에 저장할 경우에는 테이블 열의 수에 제약이 있기 때문에 열 기반의 수평 뷰 테이블을 그대로 저장하는 것은 많은 문제점들을 발생시킬 수 있다. 하지만 효율적인 저장 방법으로 행 기반의 수직 테이블로 저장한다면, 행의 수에 대한 제약이 없는 데이터베이스의 릴레이션 특성을 그대로 이용할 수 있다. 또한 저장 스키마의 구조가 변경되므로 저장뿐만 아니라 수평 뷰 테이블에 대한 질의도 수직 테이블에 적합하도록 변환해야 한다. 이러한 질의 변환에는 행과 열을 변환하는 조작 연산으로 상용 RDBMS에서 제공되는 *PIVOT* 연산을 이용할 경우 간단히 표현할 수 있다. 본 연구에서는 *PIVOT* 연산을 기반으로 질의를 변환할 경우, 이 변환 질의를 수행하는 여러 방법들에 대한 질의 트리를 생성하고 그 실행 시간을 평가하였으며, 이를 통하여 변환 질의를 최적화하는 방법을 연구하였다. 이를 통해 상용 DBMS 상에서 다차원 데이터베이스에 대한 분석을 효율적으로 저장하고 처리할 수 있도록 하였다.

본 논문에서는 단일 행과 열을 구성하는 이차원 테이블인 수평 테이블을 이용하여 질의 변환에 대한 최적화에 초점을 두어 연구를 진행하였다. 앞으로 연구해야 할 과제로는 데이터 분석을 위한 다중 행과 열을 포함하는 이차원 이상의 수평 테이블을 이용하여 질의를 처리하는 연구이다.

참 고 문 헌

[1] S. Chaudhuri and Dayal. U., "An Overview of Data Warehousing and Technology," *ACM SIGMOD Record*, Vol.26, No.1, pp. 65-74, Mar. 1997.

[2] 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.

[3] 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, Sept. 2001.

[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] Microsoft SQLServer 2005. http://www.microsoft.com/sql.

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

[9] C. Cunningham, G. Graefe, and C. A. Galindo-legaria. "PIVOT and UNPIVOT: Optimization and Execution Strategies in an RDBMS," In *Proceedings of VLDB*, pp. 998-1009, 2004.

[10] S. Chen and E. A. Rundensteiner. "GPIVOT: Efficient Incremental Maintenance of Complex ROLAP Views," In *Proc. of the 21st International Conference on Data Engineering (ICDE)*, pp. 552-563, 2005.

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

신 성 현

e-mail : shshin@kangwon.ac.kr

2000. 2. 관동대학교 컴퓨터공학과(학사)

2002. 8. 강원대학교 컴퓨터과학과(석사)

2007. 2. 강원대학교 컴퓨터과학과(박사)

2000. 3. ~ 2007. 2. 한국과학기술원 첨단정보기술연구센터 연구보조원

2007. 3 ~ 현재 한양대학교 BK21사업단 정보기술분야 박사후연구원

관심분야: OLAP, Data warehouse, Data Applications, XML data processing

문 양 세

e-mail : ysmoon@kangwon.ac.kr

1991. 2. 한국과학기술원 과학기술대학 전산학과(학사)

1993. 2. 한국과학기술원 전산학과(석사)

2001. 8. 한국과학기술원 전자전산학과 전산학전공(박사)

1993. 2. ~ 1997. 2. 현대전자산업(주) 통신사업본부 주임연구원

2001. 9. ~ 2002. 2. ㈜현대시스콤 호처리개발실 선임연구원

2002. 2. ~ 2005. 2. ㈜인프라밸리 기술연구소 기술위원(이사)

2005. 3. ~ 현재 한국과학기술원 첨단정보기술연구센터 연구원

2005. 3. ~ 현재 강원대학교 IT특성화학부(대학) 컴퓨터학부 조교수

관심분야: Data Mining, Knowledge Discovery, Stream Data, Storage System, Database Applications, Mobile/ Wireless Communication Services & Systems

김 진 호

e-mail : jhkim@kangwon.ac.kr

1982. 2. 경북대학교 전자공학과(학사)

1985. 2. 한국과학기술원 전산학과 (석사)

1990. 2. 한국과학기술원 전산학과 (박사)

1995. 8. ~ 1996.7. 미국 미시간 대학교 객원 교수

2003. 2. ~ 2004. 2. 미국 Drexel University 객원 교수

1999. 3. ~ 현재 한국과학기술원 첨단정보기술연구센터 연구원

1990. 8. ~ 현재 강원대학교 IT특성화학부(대학) 컴퓨터학부 교수

관심분야: Data warehouse, OLAP, Data Mining, Real-time/Embedded Database, Main-memory database, Data Modeling, Web Database Technology

강 공 미

e-mail : gmkang@hanmail.net

1986. 2. 한양대학교 가정학과(학사)

1999. 2. 한국방송통신대학교 전자계산학과(학사)

2002. 2. 강원대학교 교육대학원 컴퓨터교육과(석사)

2006. 2 ~ 현재 강원대학교 IT특성화대학 컴퓨터학부 박사과정

2006. 2 ~ 현재 남원주 중학교 교사

관심분야: Data Mining, Quantitative Association Rules, Data Warehouse, OLAP, Web Database Technology