OLAP 환경의 수평적인 테이블에 대한 질의 최적화 방법*

신성현° 문양세 김진호
강원대학교 컴퓨터과학과
{shshin°, ysmoon, jhkim}@mail.kangwon.ac.kr

Query Optimization Techniques for Horizontal Tables in OLAP Environment

Sung-Hyun Shin°, Yang-Sae Moon, and Jinho Kim
Dept. of Computer Science, Kangwon National University

요 약

데이터 웨어하우스는 방대한 이력 데이터들을 저장하는 저장소이며, 이를 다양한 관점에서 분석하기 위해 OLAP (On-Line Analytical Processing) 연산을 이용한다. 일반적으로 이러한 저장소는 데이터를 저장할 때 많은 열(columns)을 기반으로 저장하는 와이드(wide) 형태의 테이블로 저장하게 된다. 하지만, 관계형 DBMS에서는 열 수의 제약(MS SQLServer, Oracle 등 열의 수는 1024개임)을 받게 되므로, 그 이상의 열들을 저장할 수 없다. 하지만, 열 기반(이하, 수평 테이블)으로 저장하는 것보다는 관계형 DBMS의 특징을 이용하여 행(row) 기반(이하, 수직 테이블)으로 저장하게 되면 많은 데이터를 효율적으로 저장할 수 있다. 이때, 저장 테이블의 스키마 구조가 변경되므로, 수평 테이블에 대한 질의도 저장된 수직 테이블에 적용 가능하도록 변화시켜야 한다. 또한, 사용자에게 빠른 질의 응답을 제공하기 위해 질의 최적화를 고려하여 실행전략을 세워야 한다. 따라서 본 연구에서는 경험(heuristic)을 근거로 각 연산(프로젝션, 실렉션, 조인 연산)을 위한 질의 트리를 생성하여 질의 최적화에 대한 여러 질의 경로를 고려하고, 다양한 실험을 통해 질의 최적화에 대한 접근 경로들을 분석한다. 이로써, 본 연구의 질의 경로 분석을 기반으로 최적화 실행 계획을 기대해 본다.

1. 서 론

데이터 웨어하우스는 효율적인 질의와 분석을 위해 통합된 소스 데이터로부터 가공하여 저장된 데이터 저장소[1]로서, 사용자의 요구에 적합하도록 만들어진 소스 데이터에 대한 실체뷰(Materialized View)의 집합으로 구성된다[2]. 데이터 웨어하우스에서 사용자 질의 요구에 빠른 응답을 제공하기 위해 전형적인 수평 테이블을 실체화하여 저장하는 방법이 필요하다. 이러한 수평 테이블을 그대로 저장하는 방법과 다른 스키마 형태로 변환하여 저장하는 방법 등이 있다. 일반적인 수평 테이블은 열 기반의 와이드(wide) 형태의 스키마 구조이므로 저장하고자 하는 스키마의 열의 수에 제약이 있다. 두 번째 방법에서는 열 기반보다는 행 기반의 저장 스킴인 수직 테이블 구조로 변환하여 저장하는 방법이다. 비록 스키마의 형태는 변경되더라도, 관계형 DBMS의 릴레이션 형태로 저장하기 때문에 본질의 특성을 활용할 수 있다. 또한, 스키마의 구조가 바뀌게 되므로 수평 테이블의 질의는 수직 테이블에 적합한 질의 형태로 변경해야 한다.

본 논문에서는 질의 변환에 대하여 행과 열을 변환하기 위해 상용 DBMS[4]에서 제공하는 PIVOT 연산을 이용하고, 빠른 질의 응답을 처리하기 위해 최적화 방법에 대해 고려한다. 여기서, Agrawal 등[3]이 제안한 PIVOT 연산을 실제 상용 DBMS에 사용되는 PIVOT 구문에 적합한 형태로 관계 대수를 표현하고, 경험적 규칙으로 질의의 내부 표현을 변형한다. 이에 따라, 실제 실험 결과를 기반으로 성능에 따른 최적화 방법을 논하고자 한다. 향후, 실험 결과를 기반으로 접근 경로를 기반으로 최적화된 실행 계획을 생성할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문의 이론적 배경이 되는 관련 연구에 대해 소개하고 3장에서는 효율적인 테이블을 실체화하여 저장하는 방법들을 소개한다. 저장된 방법을 기반으로 4장에서는 경험적 규칙들을 이용하여 질의 최적화 방법을 고려하고, 5장에서는 질의 최적화에 대한 여러 질의 트리의 접근 경로들을 실험을 통해 분석한다. 마지막으로 6장에서는 본 논문의 결론에 대해 논한다.

* 본 연구는 청단정보기술연구센터(AITrc)를 통하여 한국과학재단(KOSEF)의 지원을 받았음

2. 관련 연구

현재 데이터베이스에서는 일반적인 수평 테이블을 기반으로 방대한 양의 데이터를 저장하고 있다. 이러한 저장 방법에는 수평 테이블을 그대로 저장하는 방법과 이를 변환하여 저장하는 방법이 있다. 수평 테이블로 저장하는 방법은 실제 관계형 테이블에 저장할 수 있는 최대 열의 수가 1024개라는 제약을 가진다[4,5]. 그래서 튜플의 집합을 수직 테이블로 변환하여 저장하는 방법이 필요하다[3]. 또한, 저장 방법뿐만 아니라 수평 테이블의 질의는 수직 테이블로의 질의 형태로 적합한 변환이 필요하다.

많은 열의 효율적인 저장 및 질의 처리를 위한 기존의 연구로, [4]에서는 관계형 테이블의 행과 열을 변경하기 위해 PIVOT/UNPIVOT 연산에 대한 정의를 관계 대수로 제안하고 이러한 관계 대수를 이용하여 [6]에서는 질의 성능을 개선하기 위한 최적화 실행 전략들을 제안하였다. [7]에서는 다중 PIVOT 연산을 처리하기 위해 PIVOT 연산을 간결하고 명료하게 정의하고 수평 뷰 테이블을 유지 및 관리하는 방법을 제안하였다.

상용 데이터베이스 시스템[4,5]에서는 행과 열을 변경하기 위해 서브 쿼리(sub-query)들을 이용하거나 크로스탭 쿼리(crosstap query)들을 이용하여 PIVOT 연산을 대신한다. 하지만 행과 열을 변경하는 조작 연산으로 표현하기엔 너무 복잡해지는 경향이 있다. 이러한 문제점을 해결하기 위해 최근 상용 데이터베이스인 SQL Server 2005[4]에는 질의문의 FROM 절에서 사용되는 두 개의 관계형 연산자인 PIVOT 및 UNPIVOT을 제공한다. 시스템 내부의 PIVOT 연산자를 이용하여 행을 열로 회전시키는 과정에서 집계를 계산하고 열을 기준으로 출력 테이블을 유용한 형태로 표시한다. UNPVIOT 연산자는 반대 연산으로, 즉 열을 행으로 바꾸어 결과를 표시한다. 본 연구에서는 경험적 규칙을 기반으로 최적화 질의 처리에 대한 여러 접근 경로인 질의 트리를 생성하는 이에 대한 실험으로 상용 데이터베이스 시스템에서 제공하는 PIVOT 연산[4]을 이용한다.

3. 수직 테이블의 저장 방법

요구되는 정보 분석에 관계되는 뷰를 빠르게 제공되기 위해서, 데이터 웨어하우스는 뷰들을 실체화하여 빠르게 제공될 수 있어야 한다. 데이터를 저장하는 방법에는 소스 데이터베이스로부터 테이블을 그대로 저장하는 방법과 관계형 DBMS에 적합한 형태로 변환하여 저장하는 방법이 있다.

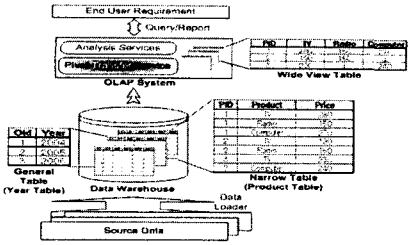

그림 1.데이터 웨어하우스 상의 OLAP 시스템.

첫 번째 방법에서는 소스 테이블 자체를 그대로 저장하는 방법(수평 테이블 형태)으로 복잡한 기법 없이 간단하게 실체뷰를 저장할 수 있다(그림 1에서의 Wide View Table). 하지만, 저장 가능한 블록 사이즈(block size)의 한계가 발생할 수 있고 여러 열들이 모여 하나의 큰 테이블을 저장할 경우, 열 중 다수가 널 값(null value)를 포함할 수 있다. 그림 2에서는 널 값으로 인한 데이터 희소성을 보여주고 있다.

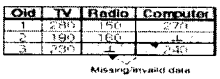

그림 2. 데이터 희소성.

두 번째 저장방법은 소스 테이블을 행 기반으로 변환하여 저장하는 방법(수직 테이블 형태)이다(그림 1에서의 Narrow Table). 저장 스키마에 대한 열의 수가 제약을 받는 첫 번째 방법과는 다르게 변환된 형태로 실체뷰들을 구성하기 때문에 행의 제약이 없는 DBMS의 특징을 그대로 활용할 수 있다. 따라서 본 연구에서는 [3]에서 제안한 PIVOT 연산을 이용하고 경험적 규칙을 기반으로 질의의 표현을 변경하여 성능을 향상시키는 최적화 방법을 고려하고자 한다.

4. PIVOT 연산의 소개 및 최적화 방법 고려

그림 1에서 수평 테이블의 저장 문제를 해결하기 위해, 수평 테이블을 수직 테이블로 변경하여 저장한다. 또한, 저장된 수직 테이블을 수평 테이블의 형태의 뷰로 변환하는 PIVOT 연산이 필요하다. 하지만, 전형적인 상용 데이터베이스 시스템에서는 행과 열을 변환하는 PIVOT 연산을 지원하지 않는다. 그래서, 사용자들은 서브 쿼리(Sub query)를 이용한 연산과 크로스탭 쿼리(CrossTab query)를 이용하여 PIVOT 연산을 대신하고 있다. 첫 번째 기존의 방법은 행과 열을 변환하는 구문으로 서브 쿼리를 사용한다(그림 3).

```
SELECT Oid
  (SELECT Product_value FROM ProductTable AS PT2
       WHERE Product_name = 'TV' AND NP1.Oid = NP2.Oid) AS TV,
  (SELECT Product_value FROM ProductTable AS PT2
       WHERE Product_name = 'Radio' AND NP1.Oid = NP2.Oid) AS Radio,
  (SELECT Product_value FROM ProductTable AS PT2
       WHERE Product_name = 'Computer' AND NP1.Oid = NP2.Oid) AS Computer
FROM ProductTable AS PT1
GROUP BY Oid
```

그림 3. 서브 쿼리를 이용한 PIVOT 기능의 구문.

다른 방법으로는 그림 4와 같이, 크로스탭 쿼리를 이용하여 행과 열을 변환하는 방법이 있다. 하지만 PIVOT 연산을 대신하는 기존의 구문들은 원하는 행의 수가 많을 경우 사용해야 할 구문이 복잡해지는 경향이 있다.

```
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. 크로스탭 쿼리를 이용한 PIVOT 기능의 구문.

행과 열을 변경하는 기존의 연산 대신에 Microsoft에서 최근 제공되는 SQL 서버 2005[4]에서는 PIVOT 연산을 이용하여 간결하게 구현될 수 있다. PIVOT 연산은 행을 열로 변경하는 과정에서 값을 집계하여 데이터를 유용한 형태로 표시한다. PIVOT 연산의 구문에 대한 예제는 그림 5에서 보이고 있다. PIVOT 연산의 반대 개념으로 UNPIVOT 연산은 열을 행으로 변경하기 위해 사용된다. UNPIVOT 연산의 자세한 내용은 [4]를 참조하면 된다.

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

그림 5. PIVOT 구문.

4.1 질의 처리 변환에 대한 대수적 표현

본 절에서는 상용 DBMS에서 제공하는 PIVOT 연산을 이용하고, 사용 가능한 여러 형태의 질의 경로를 경험(heuristics)적 규칙에 따라 질의의 내부 표현을 변형하여 처리하는 방법을 제시한다. 질의 최적화 방법을 논하기 위해 다음 표 1과 같이 주요 표기법을 이용한다.

표 1. 주요 표기법

Symbols	Definitions
viewWT	The wide format view table
NT	The narrow format stored relation
WT	The general format table
Oid	The tuple identifier
A, M	The column names of relation NT
Y	The column names of relation NT
A_i	The column values of column A of stored table $NT(1 \le i \le n)$ The column name of view table WT $(1 \le i \le n)$
M_i	The column values of column M of stored table NT $(1 \le i \le n)$
Y_i	The column values of column Y of stored table $NT(1 \le i \le n)$

열의 제약을 갖는 스킴은 효율적으로 변환하고 저장하는 방법뿐만 아니라 변경된 저장 테이블에 대한 질의 변환도 필요하다. 이러한 질의 변환은 기존 연구[7]에서 제안한 관계 대수 연산을 PIVOT 연산으로 일반화하여 소개하였다. 이를 기반으로 본 연구에서는 관계 대수 연산을 간결하게 표현하기 위한 연산으로 PIVOT 연산을 식(1)로 제안한다.

정의: 행과 열을 변경하는 조작 연산인 PIVOT 연산을 이용할 경우, 수직 테이블 NT의 기준이 되는 열 A의 값이 'in' 연산자를 이용하여 대상 테이블의 전체 Ai(1≤i≤n)인 열의 이름으로 추출하고, 열 Mi(1≤i≤k)은 집계함수)를 이용하여 대상 테이블의 열 값으로 변경하기 위해 사용하면 뷰 테이블 WT는 식(1)에 의해 충분조건이 성립한다. □

$$WT \xrightarrow[Transformation]{Query} PIVOT\left[\pi_{\frac{(A_1,A_2,\cdots,A_n)}{A\ in}} F_{Function\ M_1(NT)}\right] \quad (1)$$

4.2 프로젝션 연산의 최적화 고려

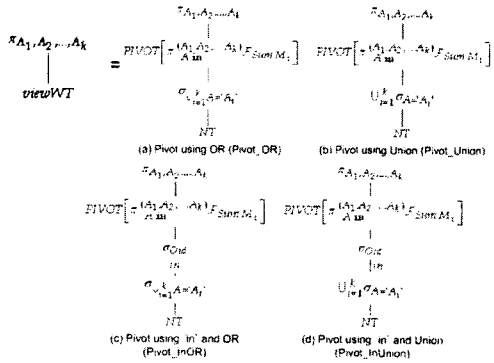

그림 6. 프로젝션을 이용한 최적화 고려.

프로젝션 연산은 대상 테이블로부터 관심있는 일부 열만을 선택하는 연산이다. 그림 6에서, 프로젝션 연산의 질의트리는 질의를 실행하기 위한 연산들의 최적화를 고려한 특정 순서를 나타낸다. 그림 6(a)와 (b)는 From 절에 명시한 테이블 NT에서 Where 절의 실렉션 연산(OR/Union 연산)을 적용한 다음에, PIVOT 연산을 적용한 과정을 보이고 있다. 그림 6(c)와 (d)에서는 테이블 NT에서 실렉션 연산(OR/Union 연산)을 적용하고

'in' 연산을 적용한 후, PIVOT 연산을 적용한 과정이다.

4.3 실렉션 연산의 최적화 고려

실렉션 연산은 논리식에서 선택 조건을 만족하는 튜플들의 집합을 선택하는 데 사용한다. 실렉션 연산은 그림 7에서와 같이 최적화를 고려한 접근 경로를 질의 트리로 나타낸다. 그림 7(a)와 (b)는 Where 절에 명시한 실렉션 연산의 조건(OR/Union 연산)을 명시하고, 조건에 해당하는 열을 기준으로 'in' 연산자를 사용하여 관련 튜플을 추출하기 위해서 Oid를 사용한다. 그런 다음에, PIVOT 연산을 적용한다.

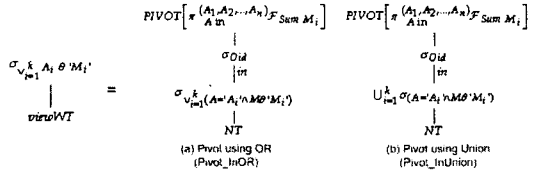

그림 7. 실렉션 연산을 이용한 최적화 고려.

4.4 조인 연산의 최적화 고려

조인 연산은 ⋈으로 표기하고, 두 테이블로부터 관련된 튜플들을 결합하여 하나의 튜플로 만든다. PIVOT 하고자 하는 대상 테이블에 대해서는 앞서 논했던 프로젝션 연산과 실렉션 연산에 대하여 최적화를 고려한 질의트리를 참고한다.

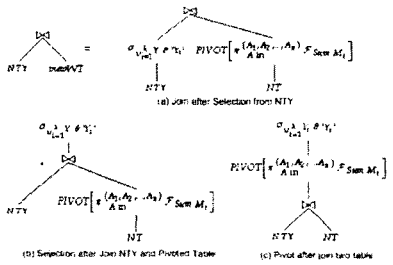

그림 8. Join 연산을 이용한 최적화 고려.

그림 8(a)에서는 From 절에 전형적인 테이블 NTY을 명시하고, Where 절의 실렉션 연산의 조건들이 적용되며, 그 다음에 PIVOT 연산이 적용된 수직 테이블 NT의 결과를 조인한 질의 트리를 나타낸다. 그림 8(b)는 테이블 NTY와 수직 테이블 NT의 PIVOT 연산에 의한 결과를 조인한 과정을 나타내며, 그림 8(c)는 테이블 NTY와 NT가 조인한 후에 PIVOT 연산을 적용한 질의 트리를 나타내고 있다.

5. 성능 분석 및 평가

5.1 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 실험은 [3]의 데이터 세트(dataset)와 같은 매개변수를 이용한다.

5.2 성능평가 결과

최적화를 이루기 위한 튜닝의 방법으로 특정 열에 인덱스를 어떻게 구성할 것인가는 정책적인 측면에서 고려될 사항이다. 따라서, 각 스키마 NT의 열 O, A에는 클러스터 인덱스를 각각 지정하였고, 열의 개수를 5, 10, 20를 선정하여 실험을 수행하였다. 실험 결과, 그림 12에서와 같이 클러스터 인덱스를 부여한 열 A는 열 O보다 계산 비용이 적게 소요된다.

본 실험에서는 실렉션 연산과 프로젝션 연산을 복합적으로 사용한 혼합 연산에 대해서 수행하였다. 실험 결과, 그림 10에서와 같이 Pivot_InUnion 연산의 경우는 계산 비용이 가장 적은 폭으로 나타났으며, 그 외 다른 연산들은 선택율에 대한 변화요인에 따라 계산비용이 큰 폭을 나타내고 있다.

조인 연산은 관계형 데이터베이스에서 가장 기본적이고 중요한 연산이다. 그림 11에서는 전형적인 테이블 Year(그림 1 참고)을 이용하여 수평 테이블 viewWT와 그림 11에서 계산비용이 가장 적은 Pivot_InUnion 연산으로 각각 조인하여 수행시간을 측정하였다. 실험 결과, 최적화된 Pivot_InUnion 연산은 선택율과 열의 개수에 따른 변환 요인에 따라 수평 테이블 viewWT에 대한 계산 비용보다 아주 작은 폭을 나타내고 있다.

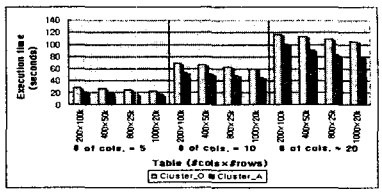

그림 9. 클러스터 인덱스 비교.

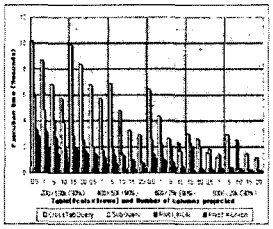

그림 10. Selectivity와 Projected columns에 대한 혼합 질의 성능 비교

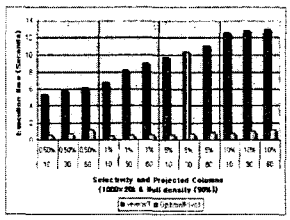

그림 11. Join 연산에 대한 성능 비교.

6. 결론

사용자들은 데이터 웨어하우스에 저장된 방대한 데이터를 OLAP 연산을 이용하여 다양한 관점으로 정보를 분석하기 원한다. 하지만 방대한 데이터를 저장할 경우에는 테이블 스키마의 열의 수에 제약이 있기 때문에 열 기반의 수평 테이블을 그대로 저장하는 것보다 행 기반의 수직 테이블로 저장한다면, 행의 수에 대한 제약이 없는 데이터베이스의 릴레이션 특성을 그대로 이용할 수 있다. 또한 저장 스키마의 구조가 변경되므로 수평 테이블에 대한 질의도 수직 테이블에 적합하도록 변환해야 한다. 이러한 질의 변환에는 행과 열을 변환하는 조작 연산으로 PIVOT 연산을 이용한다. 이러한 PIVOT 연산을 기반으로 본 논문에서는 빠른 질의 응답을 처리하기 위해서 경험적 규칙을 이용하여 여러 질의트리를 생성하여 접근 경로를 분석하고, 실험을 통해 효율적인 최적화 방법을 고려하였다. 향후, 본 연구의 최적화 방법을 기반으로 효율적인 실행전략을 세워 정책적인 방법을 기대해 본다.

참고문헌

[1] M. Mohania, S. Samtani, J. Roddick and Y. Kambayashi, Advances and Research Directions in Data Warehousing Technology, AJIS, Vol. 7, No. 1, pp. 41-59, 1999.

[2] D. Quess, A. Gupta, I. S. Mumick, and J. Widom. Making Views Self-Maintainable for Data Warehousing, Proc. of Conf. on Parallel and Database Information Systems, pp. 158-169, 1996.

[3] R. Agrawal, A. Somani and Y. Xu. Storage and Querying of E-Commerce Data. In Prodeedings of VLDB, page 149-158, 2001.

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

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

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

[7] Songting Chen and Elke A. Rundensteiner. GPIVOT: Efficient Incremental Maintenance of Complex ROLAP Views, Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, pp. 552-563, 2005.