데이터 웨어하우스에서 다차원 데이터를 위한 피벗 테이블의 효율적인 처리를 위한 관계 대수 변환*

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

Relational Algebra Query Transformation for Processing Efficiently Pivot Tables for Multi-dimensional Data in Data Warehouses

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

요 약

데이터 웨어하우스에서는 데이터를 다양한 관점으로 분석하기 위해 데이터를 다차원 형태로 유지한다. 이 다차원 데이터를 간단하고 편리한 형태로 사용자에게 표현하기 위해 피벗 테이블이 이용된다. 피벗 테이블은 데이터에 대한 요약된 정보를 제공하는데 널리 사용되는 편리한 표현 방법이지만, 실제 값이 열의 제목으로 나오기 때문에 많은 개수의 열을 가질 수 있다. 이러한 피벗 테이블을 그대로 저장할 경우 관계 DBMS의 테이블 컬럼 수에 제약을 받게 되며, 데이터 저장 및 질의 처리에 성능이 떨어질 수 있다. 이 논문은 관계 데이터베이스의 테이블을 이용하여 피벗 테이블을 효율적으로 저장하는 방법을 제안한다. 이때, 피벗 테이블에 대한 질의를 저장된 형태의 테이블에 적용 가능하도록 질의를 변환시켜야 한다. 따라서 이 연구에서는 피벗 테이블에 대한 관계 연산자들(실렉션, 프로젝션, 합집합, 차집합, 카디션 곱)을 효율적으로 변환하는 질의 변환 방법을 제안한다.

1. 서 론

대용량의 소스 데이터로부터 사용자의 여러 관점에서 다양한 분석 정보를 추출하기 위해 온라인 처리 분석(OLAP)을 지원하는 시스템에 대한 관심이 높아지고 있다. 정보 분석에 대한 다양한 사용자들의 요구를 만족시키기 위해 과거로부터 현재까지 누적된 방대한 양의 데이터를 통합적으로 관리하고, 의사 결정에 필요한 다양한 OLAP 분석을 지원하는 효과적인 접근법으로 데이터 웨어하우스를 사용한다[1].

데이터 웨어하우스는 효율적인 질의와 분석을 위해 통합된 소스 데이터로부터 가공하여 저장된 데이터 저장소로서, 다양한 관점의 데이터 분석을 효율적으로 지원하기 위해 다차원 형태로 데이터를 표현한다. 이 다차원 데이터를 편리한 형태로 사용자에게 표현하기 위해 피벗 테이블이 널리 이용된다. 피벗 테이블은 관계 데이터 모델의 테이블(즉, 릴레이션)에서 행에 나타난 값(즉, 어떤 애트리뷰트의 값)들이 열에 나타나도록 변환한 것이다. 이는 차원 애트리뷰트 값을 행과 열로 하여 다차원 데이터에 대한 집계 값을 편리하게 표현할 수 있는 방법으로, MS Excel과 같은 스프레드시트에서 널리 사용하는 방법이다[2,3,4].

피벗 테이블은 소스 데이터로부터 집계 데이터를 리포팅하거나 분석하는데 편리하게 사용될 수 있는 도구이지만, 차원 애트리뷰트의 실제 값이 열의 제목으로 나타나기 때문에 많은 개수의 열을 가질 수 있다. 관계 DBMS를 이용하여 이러한 피벗 테이블을 저장할 경우, 관계 DBMS의 테이블 열의 개수에 제약이 따른다. 또한 피벗 테이블은 행(즉 투플)의 길이가 긴 넓은 테이블(wide table) 형태를 가진다. 설사 피벗 테이블을 관계 DBMS의 테이블에 저장할 수 있다고 하더라도, 투플의 길이가 길어 프로젝트 연산 등의 질의에 성능이 현저히 떨어질 수 있다는 문제점이 있다. 따라서 이러한 피벗 테이블을 효과적으로 저장하고, 질의를 효율적으로 처리할 수 있는 방법에 대한 연구가 필요하다[4,5].

이 논문에서는 관계 DBMS의 테이블을 이용하여 피벗 테이블을 효과적으로 저장하는 방법을 이용한다. 이 방법은 피벗 테이블의 열을 행의 값으로 저장하여 보통의 테이블 형태로 표현한다. 이렇게 할 경우 피벗 테이블이 갖는 열의 개수에 제한을 받지 않으며, 관계 데이터베이스의 인덱스를 이용하여 프로젝트 연산을 처리할 수 있기 때문에 피벗 질의를 좀 더 효율적으로 처리할 수 있을 것으로 기대된다. 이런 방법으로 피벗 테이블을 저장할 경우, 피벗 테이블에 대한 사용자 질의를 저장된 테이블 형태에 대한 질의로 변환하는 것이 필요하다. 이 논문에서는 피벗 테이블에 대한 질의는 관계 데이터 모델의 기본 연산인 관계 대수를 사용하여 작성된다고 가정하였으며, 피벗 테이블에 대한 관계 대수 질의를 저장된 테이블에 대한 관계 대수 질의로 변환하는 방법을 개발하였다. 이렇게 함으로써 사용자들은 편리한 형태의 피벗 테이블을 자신의 뷰로 질의를 작성하며, 내부적으로는 효과적으로 데이터를 저장하고 질의를 효율적으로 처리하고자 한다.

이 논문의 구성은 다음과 같다. 2장에서는 논문의 이론적 배경이 되는 관련 연구에 대해 소개하고, 3장에서는 피벗 테이블을 실체화하여 저장하는 방법을 소개한다. 이 저장 방법을 기반으로 4장에서는 피벗 테이블을 위한 관계 대수를 저장 테이블에 대한 질의로 변환하는 방법을 설명하고, 마지막으로 5장에서는 결론에 대해 언급한다.

2. 관련 연구

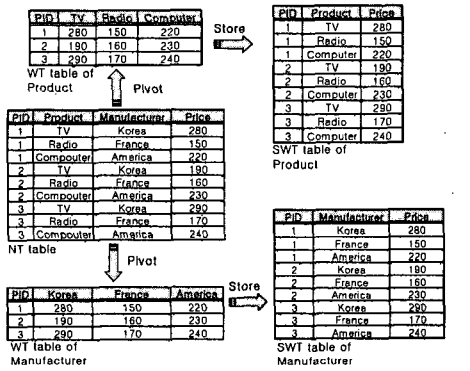
SQL 언어는 다양한 정보 분석을 목적으로 만들어지지 않았기 때문에 다차원 데이터 분석에 활용하기 어렵거나 질의가 복잡해진다. 이러한 SQL 언어나 관계 데이터 모델을 확장하여 다차원 데이터를 다루는 방법에 대해 최근에 많이 연구되고 있다[2,3,4]. [2]에서는 관계형 DBMS인 MS SQL Server2000에서 다차원 정보 분석을 제공하기 위해 CUBE와 ROLLUP 연산자들을 추가하여 기존 SQL 언어를 확장하였다. [3]은 스프레드시트(spreadsheets)의 2차원 배열 기반 분석 및 계산 기능을 SQL에 추가하였다. 그리고 [4]는 릴레이션 테이블에서 열과 행을 변경하는 데이터 조작 연산인 피벗(pivot) 연산을 기존 관계 대수에 추가하였으며, 이 확장된 관계 대수 질의를 최적화는 방법에 대해 소개하였다. 이들 연구들은 관계 데이터 모델과 SQL을 활용하여 다차원 데이터와 피벗 테이블을 다루는 방법을 제안하였으나, 이러한 피벗 테이블을 효과적으로 저장하고 그 질의를 효율적으로 처리하는 방법에 대한 연구는 부족하였다. 이 연구에서는 이러한 피벗 테이블의 저장과 그 대수 연산

* 이 논문은 2003년도 강원대학교 연구년 교수 연구비에 의하여 연구되었음

을 효율적으로 처리하는 방법에 대해 연구하고자 한다.

3. 피벗 테이블의 저장 방법

다차원 데이터는 OLAP 도구에서 스타 스키마와 같은 다차원 데이터 모델로 많이 표현된다. OLAP 도구들은 데이터 분석 이 기능이 강력하지만 사람들에게 친숙하지 않다. 데이터 분석에 가장 널리 친숙하게 사용되는 방법은 MS Excel과 같은 스프레드시트이다. 스프레드시트는 집계 값을 행과 열의 차원 값을 기준으로 이차원 형태의 표로 표현하는 방법으로, 사용자들에게 의해 널리 사용되는 친숙한 방법이다. 따라서 데이터베이스에 저장된 데이터를 이러한 피벗 테이블 형태로 표현한다면 사용자들이 쉽게 데이터 분석에 활용할 수 있을 것이다.



[그림 1] 피벗 연산과 저장된 테이블

[그림 1]은 소스 테이블 NT에서 피벗 연산을 통해 얻은 피벗 테이블 WT의 두가지 예를 보여준다. 이 피벗 테이블에서는 차원 애트리뷰트(즉, product 또는 manufacturer)의 값 (즉, TV, Radio, Computer)들이 열로 나타난다. 이 피벗 테이블을 관계 데이터베이스에 저장하는 방법에는 소스 테이블로부터 피벗 테이블을 그대로 저장하는 방법과 이를 관계형 DBMS에 적합한 릴레이션으로 변환하여 저장하는 방법이 있다. 첫 번째 방법에서는 피벗 테이블 자체를 그대로 저장하는 방법으로 복잡한 저장 기법 없이 간단하게 피벗 테이블을 저장할 수 있다. 하지만 이렇게 저장하는 방법에서 발생하는 문제점 중의 하나는 피벗 테이블의 열의 개수가 매우 방대할 수 있다는 것이다. 열의 개수가 많을 경우, 저장 가능한 블록 사이즈(block size)의 한계가 발생할 수 있으므로 저장이 불가능한 경우가 있을 수 있다.

두 번째 방법으로는 피벗 테이블을 보통의 릴레이션 형태로 저장하는 방법이다. 위 [그림 1]에는 각 피벗 테이블에 대해 보통의 릴레이션 형태로 저장한 예가 나타나 있다. 이러한 방법의 경우 피벗 테이블의 열의 개수에 제한을 받지 않으며, 관계 데이터베이스 시스템의 인덱스나 질의 처리 등의 기능을 그대로 이용할 수 있다는 장점이 있다. 하지만 이렇게 저장할 경우, 사용자에게 제공되는 피벗 테이블 형태의 뷰와 실제 저장 구조가 다르기 때문에 피벗 테이블의 뷰에 가해진 사용자들의 질의를 실제 저장된 테이블 형태의 질의로 변환해 주는 것이 필요하다. 즉, 사용자 관점에서 정보를 분석하기 위해 소스 테이블로부터 피벗 테이블에 대한 질의를 하게 된다. 이 질의를 처리하기 위해서는 해당 질의를 저장 테이블에 대한 질의로 변환하는 과정이 필요하다. 따라서 이 논문에서는 피벗 테이블에 대한 관계 대수 질의를 실제 저장 테이블에 적용할 수 있는 관계 대수 연산으로 효율적으로 변환하는 방법을 개발하였으며, 이에 대해서는 다음 장에서 설명한다.

4. 피벗 테이블에 대한 질의 처리 방법

본 논문에서 이용하고자 하는 관계 대수는 관계 데이터 모델에서 중요한 언어이며 DBMS에 흔히 사용되는 SQL의 이론적인 기초이다. 따라서 본 논문에서는 관계 대수식의 다섯 가지 연산들인 실렉트, 프로젝션, 합집합, 차집합, 카티션 곱에 대한 질의 변환 방법을 제안한다. 제안하고자 하는 질의에 대해 피벗 테이블에서 실제 저장 테이블로 변환할 때 사용하고자 하는 메타 정보를 정의하기 위하여 [그림 2]의 메타 테이블의 표기법을 사용한다.

Rname	Type	GroupCol	Colname	ColValue	MeasureCol
NT	N	K	A	a_i	M
WT	W	K	a_j	m_i	⊥
SWT	S	K	A	a_i	M

[그림 2] NT, WT, SWT table의 메타 정보 테이블

실렉션(selection) 연산 실렉션 연산은 피벗 테이블에서 사용자가 요구하는 조건에 만족하는 튜플들의 부분 집합을 생성한다. 따라서 실렉션 조건을 프레디키드(predicate)라고도 하며 $P(a_i, x)$ 로 표기한다. 여기서 a_i 는 소스 테이블, 피벗 테이블, 실제 저장 테이블의 메타 정보를 나타내는 [그림 2]에서와 같이 피벗 테이블의 컬럼명을 의미하며, x 는 상수를 나타낸다. 이 프레디키드는 저장 테이블의 컬럼명 A 와 피벗 테이블의 컬럼명 a_i 은 $P'(a_i, x) \equiv (A = a_i) \wedge (M \theta x)$ 와 같이 매핑될 수 있다. M 은 소스 테이블과 저장 테이블의 컬럼명을 나타내고, 이에 대한 m_i 은 컬럼 값인 측정값으로 나타낸다. 그리고 θ 는 비교 연산자들 $\{=, \neq, \leq, \geq, <, >\}$ 중의 하나이다.

프레디키드에 명시된 조건들은 여러 개의 절(clause)로 구성될 수 있다. 즉, 프레디키드는 하나인 경우와 두 개 이상인 경우로 나누어 변환한다. 먼저, 하나인 경우는 피벗 테이블 WT에서 사용자가 실렉트 연산을 이용하여 $\sigma_{TV>280}(WT)$ 와 같이 하나의 프레디키드를 기술할 수 있다. 피벗 테이블을 대상으로 질의가 요구되면 실제 저장 테이블에서는 이 질의를 변환하여 제공해야 한다. 피벗 테이블의 대한 질의에서 실제 저장 테이블에 대한 질의로 변환하는 과정은 아래의 규칙 1로 정의한다.

규칙 1.

$$\sigma_{P(a_i,x)}(WT) \xrightarrow{transform} \text{PIVOT}\left[\sigma_{Key \text{ in } \pi_{Key'}(\sigma_{P'(a_i,x)}(SWT'))}(SWT)\right]$$

규칙 1에서 피벗 테이블에서 프레디키드에 연관된 SWT' 는 참조 무결성 규칙이 존재하는 Key 를 가진 SWT 의 사본(copy)으로 피벗 테이블에서의 프레디키드 조건을 만족하도록 해당 조건 절을 명시한다.

규칙1에 대한 예를 들어 $\sigma_{TV>280}(WT)$ 은 예제 (1)로 나타낼 수 있다.

$$\text{PIVOT}\left[\sigma_{PID \text{ in } \pi_{PID'}(\sigma_{product='TV' \wedge price>280}(SWT'))}(SWT)\right] \quad (1)$$

다음으로, 두 개 이상의 프레디키드가 사용될 경우 AND, OR, NOT과 같은 부울 연산자를 이용하여 프레디키드를 임의로 연결하여 나타낼 수 있다. 우선 AND 연산은 규칙 2로 정의한다.

규칙 2.

$$\sigma_{\bigwedge_{i=1}^{n} P_i(a_i,x_i)}(WT) \xrightarrow{transform} \text{PIVOT}\left[\bigwedge_{i=1}^{n} \sigma_{Key \text{ in } \pi_{Key'}(\sigma_{P'(a_i,x_i)}(SWT'))}(SWT)\right]$$

피벗 테이블에서 AND 연산을 사용하여 나타낸

$\sigma_{\bigwedge_{i=1}^{n} P_i(a_i,x_i)}(WT)$ 은 $\sigma_{P_1(a_1,x_1)\wedge P_2(a_2,x_2)\wedge\cdots\wedge P_n(a_n,x_n)}(WT)$ 로 나타낼 수 있으며 이를 다시 $\sigma_{P_n(a_n,x_n)}(\cdots(\sigma_{P_2(a_2,x_2)}(\sigma_{P_1(a_1,x_1)}(WT)))\cdots)$ 으로 나타낼 수 있다[6]. 마지막으로 프레디키드 조건을 만족하도록 이러한 연산에 규칙 1을 적용하게 되면 결과적으로 규칙 2를 정의할 수 있다. 규칙 2에 대하여 $\sigma_{TV>180\wedge TV<290}(WT)$ 는 $\sigma_{TV<290}(\sigma_{TV>180}(WT))$ 으로 표현되며 실제 저장 테이블에 대한 질의를 변환은 예제 (2)로 나타낼 수 있다.

$$\text{PIVOT}\begin{bmatrix}\sigma_{PID \text{ in } \pi_{PID'}(\sigma_{product='TV'\wedge price>180}(SWT'))}(SWT)\\ 1\ \sigma_{PID \text{ in } \pi_{PID'}(\sigma_{product='TV'\wedge price<290}(SWT'))}(SWT)\end{bmatrix} \quad (2)$$

다음으로, OR 연산은 규칙 2와 유사한 방법으로 규칙 3으로 정의할 수 있다.

규칙 3.

$$\sigma_{\bigvee_{i=1}^{n} P_i(a_i,x_i)}(WT) \xrightarrow{transform} PIVOT\left[\bigvee_{i=1}^{n}\sigma_{Key \text{ in } \pi_{Key'}(\sigma_{P'(a_i,x_i)}(SWT'))}(SWT)\right]$$

NOT 연산은 드모르간 법칙(DeMorgan's laws)에 의해 AND 연산에서 OR 연산으로, 또는 OR 연산에서 AND 연산으로 표현이 가능[6]하므로 NOT 연산에 대한 규칙 2는 규칙 3으로 또는 규칙3에서 규칙 2로 적용 가능하다.

프로젝션(Projection) 연산 프로젝션 연산자는 실렉트 연산과 달리 컬럼들의 부분 집합을 구하는 연산자이다. 프로젝션 연산자는 규칙 4로 정의할 수 있다.

규칙 4.

$$\pi_{a_1,a_2,\ldots,a_n}(WT) \xrightarrow{transform} \text{PIVOT}\left[\pi_M(\sigma_{\bigvee_{i=1}^{n} A=ai}(SWT))\right]$$

규칙 4에 대하여 $\pi_{TV}(WT)$ 는 예제 (3)으로 변환이 가능한다.

$$\pi_{TV}(WT) \xrightarrow{transfer} \text{PIVOT}\left[\pi_{price}(\sigma_{product='TV'}(SWT))\right] \quad (3)$$

합집합(Union) 연산 합집합 연산은 두 개의 릴레이션에서 존재하는 모든 튜플들을 포함하도록 연산이다. 합집합 연산은 규칙 5에서 정의할 수 있다.

규칙 5.

$$\bigcup_{i=1}^{n} WT_i \xrightarrow{transform} \text{PIVOT}\left[\bigcup_{i=1}^{n} SWT_i\right]$$

차집합(Set difference) 연산 두 릴레이션 WT_1과 WT_2의 차집합 $WT_1 - WT_2$는 WT_1에 존재하지만 WT_2에는 존재하지 않는 릴레이션을 구성하는 연산이다. 이러한 차집합 연산은 규칙 6에서 정의할 수 있다.

규칙 6.

$$WT_1 - WT_2 = \pi_{a_1^1}(WT_1) - \pi_{a_i^2}(WT_2) \xrightarrow{transform} \text{PIVOT}\left[SWT_1 - \sigma_{m \text{ in } \pi_{m'}(\sigma_{A=a_i}(SWT_2))}(SWT_1)\right]$$

카티션 곱(Cartesian product) 연산 실렉션이나 프로젝션을 이용하여 피벗 테이블에 대한 질의는 실제 저장 테이블에 적용하기 위해 변환하여 데이터를 검색할 수 있지만, 이러한 테이블을 다른 테이블과 연관시키기 위해서 두 테이블들을 결합할 필요가 있다. 카티션 곱 연산에 대한 질의는 규칙 7에 의해 정의될 수 있다.

규칙 7.

$$WT_1 \times WT_2 \xrightarrow{transform} \text{PIVOT}[storedWT_1] \times \text{PIVOT}[storedWT_2]$$

카티션 곱 연산은 우선 실제 저장 테이블들을 각각 피벗한 후에 적용해야 하는 데, 피벗하기 이전에 카티션 곱 연산을 적용하게 되면 질의에 요구에 부합될 수 있는 가능성의 문제를 발생시킬 수 있다. 따라서 이 연산은 각각의 저장 테이블들을 피벗한 후에 두 테이블들을 결합해야만 사용자가 실제 원하는 질의 응답을 제공할 수가 있다.

지금까지 논했던 실렉션, 프로젝션, 합집합, 차집합, 카티션 곱은 관계 대수의 필수적인 연산자이며, 조인(join) 연산자, 디비젼(division) 연산자들은 필수적인 관계 연산자를 두 개 이상 조합하여 표현이 가능하다.

5. 결론

데이터 웨어하우스에서 다양한 정보를 분석하기 위해 사용자들은 온라인 처리 분석(OLAP)을 이용하여 소스 테이블에서 피벗 형태의 테이블을 통해 다양한 분석 질의를 처리한다. 소스 테이블에서 추출한 피벗 테이블을 효율적으로 저장하는 방법이 필요하다. 하지만 관계 데이터베이스의 테이블은 컬럼 수에 제한이 있기 때문에 피벗 테이블을 그대로 저장하는 것은 여러 문제점이 발생한다. 피벗 테이블을 편리하게 저장하는 방법 중의 하나는 기존의 관계형 테이블을 이용하는 것이다. 이 논문에서는 피벗 테이블을 기존의 관계형 테이블로 저장하는 방법을 제안하고, 피벗 테이블에 대한 질의를 저장 테이블에 대한 질의로 변환하는 방법을 제안하였다. 이 방법은 관계 데이터베이스의 기능을 그대로 이용하여 피벗 테이블의 연산을 효율적으로 처리할 수 있다는 장점이 있다.

참고문헌

[1] 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.

[2] J. Gray, A. Bosworth, A. Layman, 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, 1997.

[3] A. Witkowski, S. Bellamkonda, T. Bokaya, G. Dorman, N. Folkert, A. Gupta, L. Shen, S. Subramanian. Spreadsheets in RDBMS for OLAP, 2003 Proceedings of the ACM SIGMOD international Conference on Management of Data, San Diego, CAA, pages 52-63.

[4] 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.

[5] 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.

[6] Hector Garcia-Molina and Jeffery D. Ullman, Database System: The Complete Book, Stanford University, 2001.