다차원 데이타베이스에서 차원속성 레벨을 이용한 조인 테이블 추적의 정형화

윤 원 식　　신 동 천

ywonsik@cau.ac.kr　dcshin@cau.ac.kr

중앙대학교 정보시스템학과

Formalization of Tracing Join Table Using Dimension Attribute Level in Multidimensional Databases

Won-Sik Yoon Dong-Cheon Shin

Dept. of Information System, Chung-Ang University

요 약

다차원 데이타베이스에서 데이터 분석을 위한 OLAP질의에 대한 응답 시간을 줄이기 위해 실체 뷰를 고려할 수 있다. 다차원 데이타베이스에서의 실체 뷰는 차원 테이블과 사실 테이블의 조인으로 구성되어 있는 조인 뷰를 형성하며 적절한 개수의 실체 뷰를 선택하는 일은 중요하다. 조인비용은 다차원 데이타베이스의 실체 뷰 선택에 있어서 가장 중요한 요소이다. 본 논문에서는 조인 비용을 구하기 위해서 실체 뷰의 계층 정보를 이용하여 조인 테이블 추적하는 방법을 정형화하고 구현한다.

1. 서론

다차원 데이타베이스(multidimensonl database)는 거대한 용량의 이력적 데이터를 가지고 복잡한 집계(aggregation)를 요구하는 의사결정에 필요한 질의에 대하여 통합된 환경을 제공하는 데이터 저장소라고 할 수 있다. 즉 다차원 데이타베이스는 사용자의 다차원적인 관점을 지원해주기 위하여 정보가 스타 스키마로 구성되어 있는 관계형 데이터 웨어하우스이다[1]. 스타 스키마에서 각 차원 테이블은 특정 차원 자체에 대한 정보를 포함하고 있는 반면에 사실 테이블은 모든 차원 테이블과의 관계를 맺고 있으며 차원 키 속성에 대한 정보를 포함하고 있다.

다차원 데이타베이스의 사용자는 데이터 분석을 수행하기 위해 주로 복잡한 질의문을 사용한다. 일반적으로 데이터 분석을 수행하기 위한 질의문에는 하나 이상의 집계함수와 group-by 연산자가 포함되며 질의 대상인 다차원 데이타베이스의 레코드 개수는 수백만에서 수천만이기 때문에 이러한 레코드들에 대한 질의 처리시간이 수분 또는 수시간 정도 소요된다. 따라서 OLAP 응용 프로그램(예: 의사결정지원 애플리케이션)의 사용자가 요구하는 응답시간을 충족시키기 어렵다. 이를 위하여 응답시간을 최소화하는 사용자가 미리 요구하는 질의를 계산하여 결과를 실체화된 뷰(materialized view)에 저장해 두는 방법이 이용된다[4].

뷰를 실체화 할 때 고려해야 할 문제 중 하나가 실체 뷰 선택 문제이다. 실체 뷰는 실제의 저장공간에 결과를 저장해 두기 때문에 저장 할 수 있는 뷰의 개수는 제한되어 있다. 그러므로 사용자의 요구 즉 신속한 질의 처리시간을 잘 만족시킬 수 있도록 제한된 저장공간에 알맞은 뷰를 선택하여 저장하는 것이 중요한 문제이다.

기존의 실체 뷰 선택에 관한 여러 연구가 진행되어 왔다. 기존의 연구들은 다차원 데이타베이스의 사실 테이블만을 포함한 데이터 큐브에서 실체 뷰를 선택하는 연구이었다[2][3][5]. 이러한 환경에서 실체 뷰는 사실 테이블만으로 형성되는 뷰이며 따라서 실체 뷰 선택의 중요한 기준이 되는 질의 처리비용을 실체 뷰의 크기와 동일하다고 가정하고 있다. 하지만 현재 실세계에서 운영되는 다차원 데이타베이스들은 사용자의 다차원적 관점을 지원하기 위해 수십개의 차원과 차원계층을 포함하고 있으며 따라서 이러한 차원과 차원계층을 고려하여 실체 뷰를 선택하는 것이 필요하다.

다차원 데이타베이스는 여러 개의 차원 테이블과 사실 테이블을 포함하고 있으며 이러한 다차원 데이타베이스에서 생성되는 실체 뷰는 사실 테이블과 차원 테이블의 조인으로 형성된 뷰이다. 따라서 실체 뷰 선택시 중요한 기준이 되는 질의 처리비용에 많은 부분을 차지하는 조인비용을 고려하는 것은 타당하다.

본 논문은 다차원 데이타베이스에서 실체 뷰를 선택하는 데 중요한 기준이 되는 조인비용을 구하기 위하여 실체 뷰로 부터 조인 테이블을 추적하는 방법을 정형화하고 구현한다. 다차원 데이타베이스의 실체 뷰는 차원속성 레벨을 명시적으로 내재하고 있으며 이러한 차원속성 레벨을 이용하면 정형화된 방법으로 조인 테이블을 추적할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 사용하는 다차원 데이타베이스 환경에 대하여 기술한다. 3장에서는 조인 비용을 구하기 위하여 실체 뷰로부터 조인에 필요한 테이블을 추적하는 방법을 정형화하고 구현한 내용을 소개한다. 4장에서는 결론을 맺는다.

2. 다차원 데이타베이스 환경

2.1 스타 스키마와 스노우플레이크 스키마

다차원 데이타베이스의 설계에는 차원 모델링이 이용되며 차원 모델링으로 가장 대표적인 것이 스타 스키마와 스노우플레이크 스키마를 들 수 있다. 그림 1은 다차원 데이타베이스를 스타 스키마와 스노우플레이크 스키마로 나타낸 예이다. 그림 1의 다차원 데이타베이스는 데이터를 보는 방법을 나타내는 2개의 차원 테이블과 실제 분석하고자 하는 수치 데이터를 포함하는 1개의 사실 테이블로 구성되어 있다. 하나의 차원 테이블에는 여러 개의 차원 속성이 존재할 수 있다. Product라는 차

원 테이블은 product, type, category라는 차원 속성(dimension attribute)으로 구성되어 있고 각 차원 속성간에는 계층을 이루고 있으며 1:n의 관계를 가지고 있다. 이러한 계층 정보는 사용자가 데이터를 분석할 때 아주 효율적으로 쓰인다.

스타 스키마가 각 차원마다 하나씩 차원 테이블이 있는 것에 반하여 스노우플레이크 스키마는 그림 1에서 보듯이 차원 속성마다 1개의 차원 테이블을 가지고 있고 이 차원 테이블에는 차원 속성에 대한 정보와 상위 차원 테이블로 조인하기 위한 키가 들어 있다. 스노우플레이크 스키마의 차원 테이블은 정규화가 잘 되어 있어서 중복이 안된다. 따라서 차원 테이블의 크기도 작다. 즉 스타 스키마는 스노우플레이크 스키마를 비정규화(denomalization)한 것이라고 생각할 수 있으며 역으로 스노우플레이크 스키마는 스타 스키마의 정규화된 형태라고 생각할 수 있다.

본 논문에서 사용하는 다차원 데이타베이스는 중복을 제거한 스노우플레이크 스키마로 구성되어 있다고 가정하며 가정된 스키마는 다음과 같다.

sales (*product_id, *store_id, quantity)
products(product_id, product_name, *type_id)
stores(store_id, store_name, *city_id)
type(type_id, type_name, *category_id)
city(city_id, city_name, *state_id)
category(category_id, category_name)
state(state_id, state_name)

위의 스키마에서 sales 테이블은 다차원 데이타베이스의 사실 테이블을 나타내며 수치 값인 quantity를 포함하고 있다. sales 테이블의 product_id와 store_id는 복합 기본키(composite primary key)를 구성하며 각각은 stores, products차원 테이블과 관계를 맺어주는 외래키(foreign key)의 역할도 하고 있다. 또한 각각의 차원 테이블은 상위의 차원 테이블과 관계를 갖기 위해 외래키를 포함한다. 밑줄로 표시된 컬럼은 기본키를 나타내며 *표시된 컬럼은 외래키를 나타낸다.

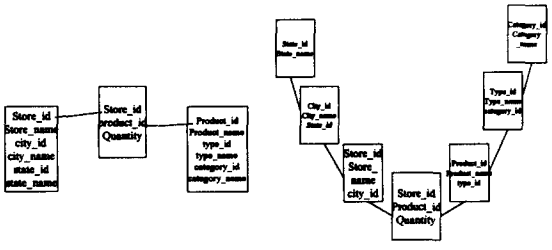

그림 1 Star_schema와 snowflake schema

2.2 복합격자(Composite Lattice)

다차원 데이타베이스에서 정의된 뷰는 또 다른 뷰로부터 계산될 수 있다. 이렇듯 다차원 데이타베이스를 구성하는 뷰들 사이에는 의존관계가 성립하며 격자구조는 이러한 관계를 가장 잘 설명해주는 구조이다.

그림 1과 같은 스카마를 가진 다차원 데이타베이스에서 단순격자는 사실 테이블에 포함되는 차원의 수만을 고려해서 생성된다. 즉 그림 1에서 고려되는 뷰는 product_id와 store_id에 따라 그룹핑 되는 뷰(product_store)와 store_id, product_id 각각에 대해서만 그룹핑 되는 뷰(store, product)그리고 그룹핑을 고려하지 않은 뷰(none)의 단순한 구조를 이룬다. 복합격자는 차원 테이블에 존재하는 계층구조인 차원속성을 고려해서 단순격자의 구조를 확장시킨 구조이다. 그림 2는 그림 1와 같은 다차원 데이타베이스에서 생성되는 단순격자와 복합격자를 나타낸다.

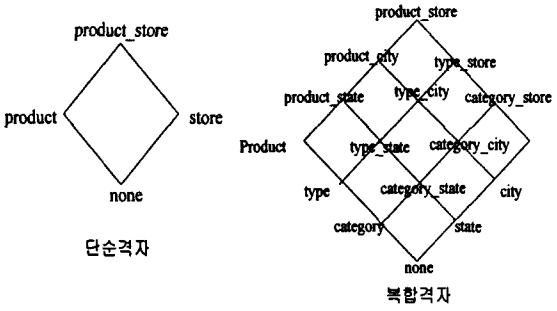

그림2 격자구조

단순격자에서 복합격자로 실체 뷰 선택을 확장한 경우 나타나는 특성은 다음과 같다.
• 격자의 수가 기하급수적으로 증가한다.
• 복합격자를 형성하는 뷰는 사실 테이블과 차원 테이블과의 조인이 필요한 조인 뷰로 형성된다. 예를 들면 데이타베이스에서 정의된 뷰 type_city정의는 다음과 같다.

```
create view  type_city  As
select type_id, city_id, sum(quantity)
from sales, product, store
where sales.store_id = store.store_id and
      sales.product_id = product.product_id
group by type_id, city_id
```

3. 조인 테이블 추적의 정형화

3.1 계층 정보를 이용한 조인 테이블 추적

스노우플레이크 스키마로 구성된 다차원 데이타베이스 내에서 차원 테이블의 계층 정보는 차원 테이블의 차원속성 레벨의 의하여 명시적으로 존재하며 이러한 다차원 데이타베이스로부터 도출된 실체 뷰들은 하나의 사실 테이블과 여러 개의 차원 테이블의 조인으로 형성된 복합 뷰(Complex view)로 정의되어 있다. 조인비용을 고려하여 실체 뷰를 선택을 하는데 있어서 먼저 각 실체 뷰가 생성하는데 소요되는 조인비용을 구하는 것이 앞서 진행되어야 하며 조인비용을 구하기 위해서는 조인 연산과 관련된 테이블을 추적하는 것이 필요로 한다. 그림 1의 다차원 데이타베이스로부터 정의되는 뷰들의 의존관계를 그림 2와 같은 복합격자로 표현할 수 있다. 위의 복합격자에서 형성된 type_store 뷰는 sales 테이블과 product 테이블의 조인으로 형성되며 반면에 격자의 최상위에 존재하는 product_store뷰는 격자 위에 존재하는 여러 뷰들 중 가장 상세한 정보를 포함하며 조인연산 없이 사실 테이블 sales로부터 바로 유도되어진다. 복합격자에서 존재하는 모든 뷰는 반드시 사실 테이블을 포함하고 있다. 복합격자에서 존재하는 뷰는 사실 테이블 요약하는 요약 정보를 가지고 있기 때문에 반드시 수치 데이터를 포함하는 사실 테이블을 포함하고 있어야 한다.

다차원 데이타베이스의 차원에 존재하는 계층구조에 대한 정보는 다차원 데이타베이스의 정보를 표현하는 스노우플레이크 스키마의 차원속성 레벨에 나타나며 또한 복합격자에 존재하는 실체 뷰에도 나타난다. 이와 같이 실체 뷰에 나타난 차원속성 레벨을 이용하여 쉽게 조인과 관련된 테이블을 추적할 수 있다.

본 논문에서 사용되는 다차원 데이타베이스의 차원 속성의 레벨을 정의하면 다음과 같다.

products차원의 차원속성 레벨
레벨1:product --> 레벨2:type --> 레벨3:category

stores차원의 차원속성 레벨
레벨1:store --> 레벨2:city --> 레벨3:state

표기 1 : $v_{d1d2...dn}$ 은 n개의 차원을 갖는 다차원 데이타베이스에서 임의의 실체 뷰이다. 여기서 d_i는 i번째 차원의 차원 속성의 레벨을 의미한다.

예를 들어, 그림 1에는 product 와 store의 2개 차원 중에서 product를 첫 번째 차원이라고 하면 product 차원에 있는 3개의 차원 속성 product, type, category의 레벨은 각각 1, 2, 3이 된다. 한편, 각 차원의 레벨 0는 해당 차원으로 실체 뷰를 만들지 않았음을 의미하며 사실 테이블의 레벨을 의미한다. 따라서, 실체 뷰 v_{13}은 product_state를, v_{20}은 type 뷰를 나타낸다.

표기 2: $t_{d1d2...dn}$ 은 각 차원에서 사실 테이블과 해당 차원 속성 레벨에 해당하는 차원 속성 테이블들의 집합을 의미한다. 여기서 d_i는 i번째 차원의 차원 속성의 레벨을 의미한다.

예를 들어, 2개의 차원이 있는 그림 1의 경우에 t_{22} 는 해당 차원 속성의 테이블인 city와 type 테이블이 포함되며 다차원 데이타베이스가 스노우플레이크 스키마로 구성되어 있기 때문에 city와type 테이블은 상위 테이블 즉, 하위 레벨의 store와 product 테이블과 연결되므로 이들도 포함된다. 또한, 사실 테이블인 sales는 다차원 데이타베이스의 전체 실체 뷰에 모두 포함되기 때문에 t_{22}= {sales, store, city, product, type} 이 된다.

한편, 임의의 실체 뷰 $v_{d1d2...dn}$ 와 $t_{d1d2...dn}$ 의 집합을 각각 V와 T라 할 때 모든 $v_{d1d2...dn} \in V$ 에 대해 $v_{d1d2...dn}$를 실체화하기 위해 조인에 관련되는 테이블의 집합은 $t_{d1-1d2-1...dn-1}$ 이다. 예를 들어, 그림 1의 경우 v_{22}인 뷰 type_city를 실체화하기 위해서는 product와 store 테이블, 그리고 사실 테이블인 slaes가 필요하며 t_{11} ={product, store, sales} 이므로 V_{22} = t_{11} 이 성립한다.

3.2 구현 개요

본 절에서는 임의의 실체 뷰로부터 조인과 관련된 테이블을 추적하는 방법을 구현한 내용을 간략히 소개한다.

조인 테이블 추적 프로그램은 c++언어로 구현하였으며 본 프로그램은 최대 4개의 차원과 10개의 차원속성 레벨을 가진 다차원 데이타베이스에서의 조인 테이블의 추적을 지원할 수 있다. 개략적인 프로그램 구조는 다음과 같으며 그림 3은 조인 테이블 추적 프로그램의 입력과 출력 양식이다.

단계1: Initialization step: 각 차원별 테이블 명을 파일을 통해 읽어들여 배열에 초기화한다.
단계2: Tracing step : 추적을 뷰 이름으로 할지 뷰의 계층으로 할지를 결정한다. 추적을 뷰 계층으로 하면 단계2.1로 뷰 이름으로 하면 단계2.2로 간다.
단계2.1: Tracing_hierarchies step : 뷰의 차원속성 레벨을 입력한다.
단계2.2: Tracing_name step : 뷰의 이름을 입력한다.
단계2.2.1: Tracing_conversion step : 뷰의 이름을 뷰의 차원속성 레벨로 전환시킨다.
단계3: Tracing_start_correct_exit step : 추적을 시작, 수정, 종료할지를 결정한다.
단계4: Tracing_Process step : 뷰의 이름이나 뷰의 차원속성 레벨의 입력 값에 대한 처리 및 출력한다. 입력된 뷰 차원 속성 레벨을 이용하여 조인과 관련된 테이블을 추적한다.
단계5: Tracing_Again step : 추적을 반복할지를 결정한다. 추적을 반복하려면 단계2로 간다.

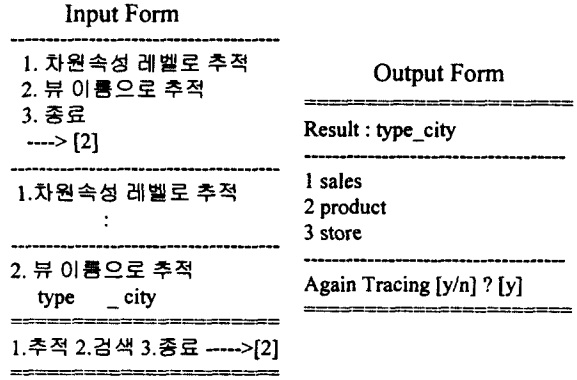

그림3 조인테이블 추적 프로그램

4. 결론 및 향후과제

다차원 데이타베이스에서의 실체 뷰는 사실 테이블과 차원 테이블의 조인으로 형성되며 조인 비용을 구하기 위해서는 어떤 뷰가 어떤 테이블의 조인으로 이루어져 있는지를 추적하는 것은 매우 중요하다. 일반적으로 스노우플레이크 스키마로 구성된 다차원 데이타베이스는 차원 테이블의 계층 정보가 차원 속성 레벨에 명시적으로 나타나며 또한 다차원 데이타베이스의 실체 뷰에도 계층 정보인 차원속성 레벨이 나타나기 때문에 이러한 차원속성 레벨을 이용하면 쉽게 조인 테이블을 찾을 수 있다. 본 논문에서는 다차원 데이타베이스의 실체 뷰에 나타난 차원속성 레벨을 이용하여 조인 관련 테이블을 추적하는 방법을 정형화하고 구현하였다.

향후에는 추적된 조인 테이블을 가지고 스타 조인을 기반으로 하는 조인 비용을 구하고 이를 이용하여 다차원 데이타베이스의 실체 뷰를 선택하는 기법을 연구할 계획이다.

참고 문헌

[1] E. Baraalis, S. Paraboschi, and E. Teniente "Materialized View Selection in a Multidimensional Database" In Procedings of the 23rd VLDB Conference, pp. 156-165, Athens, Greece, 1997
[2] V. Harinarayan, A. Rajaraman, and J. D Ullman, "Implementing Data Cube Efficiently" In Procedings of the ACM SIGMOD conference, pp. 205-216, Montreal, Quebec, canada June 1996
[3] H. Gupta, V. Harinarayan, A. Rajaraman "Index Selection for OLAP" Procedings of the International Conference on Data Engineering, Binghamtor, UK, April, 1997
[4] J. Hammer, H. Garcia-Molina, J, Widom, W. Labio, and Y. Zhuge "The Stanford Data warehousing Project" IEEE Data Enginerring Bulletin, Special Issue on Materialized View and Data Warehousing, 18(2):41-48, June 1995
[5] 서은주 "뷰 이용률을 고려한 효율적인 데이터 큐브 구현 기법" 포항공대 석사학위논문. 1997