論文2002-39CI-3-5-1

관계형 OLAP 데이터 웨어하우징 환경에서 조인과 집계함수를 포함하는 질의의 효율적인 처리

(Efficient Processing of Queries with Joins and Aggregate Functions in ROLAP Data Warehousing Environment)

金 鎭 皓 *, 金 潤 浩 **, 金 尙 煜 **

(Jin-Ho Kim, Yun-Ho Kim, and Sang-Wook Kim)

요 약

대용량의 데이터가 저장되는 데이터 웨어하우징 환경에서 조인이나 집계 함수와 같은 고비용의 연산의 효율적인 처리는 매우 중요하다. 본 논문에서는 집계 함수(aggregate function)와 조인(join)이 모두 포함된 질의를 처리하는 새로운 기법을 제안한다. 제안하는 기법은 먼저 차원 테이블(dimension table)을 미리 그루핑한 후, 비트맵 조인 인덱스(bitmap join index)를 이용하여 조인을 처리하는 방식을 사용한다. 이 결과, 사실 테이블(fact table)만을 접근하여 집계 함수를 처리함으로써 기존 기법이 가지는 성능 저하의 문제점을 해결할 수 있다. 기존 기법과 제안하는 기법에 대한 비용 모델(cost model)을 정립하고, TPC-H 벤치마크를 기반으로 하는 다양한 시뮬레이션을 수행함으로써 제안된 기법의 우수성을 규명한다.

Abstract

Efficient processing of expensive queries that include joins and/or aggregate functions is crucial in data warehousing environment since there reside enormous volume of data. In this paper, we propose a new method for processing of queries that have both of joins and aggregate functions. The proposed method first performs grouping of the dimension table and then processes join by using the bitmap join index. This makes only the fact table accessed for processing aggregate functions, and thus resolves the serious performance degradation of the existing method. For showing the superiority of the proposed method, we suggest the cost models for the proposed and existing ones, and perform extensive simulations based on the TPC-H benchmark.

* 正會員, 江原大學校 電子計算學科

(Department of Computer Science Kangwon National University)

** 正會員, 江原大學校 컴퓨터 情報通信工學部

(Division of Computer, Information, and Communications Engineering, Kangwon National University)

※ 이 연구는 첨단정보기술연구센터(AITrc)를 통하여 한국과학재단의 지원을 받았으며, 과학재단 기초 연구 (과제번호 R05-2002-000-01085-0)의 지원을 받았습니다.

接受日字:2002年2月6日, 수정완료일:2002年8月27日

Ⅰ. 서 론

의사결정 과정은 축적된 수많은 데이타로부터 통계, 추이, 경향 분석 작업을 통하여 추출된 유용한 정보를 기반으로 수행된다. 이러한 정보를 추출하기 위해서는 그 동안 축적되어온 많은 정보를 검색해야 하고 복잡한 계산을 처리해야 한다. 이러한 고비용의 작업을 운용 데이터베이스(operational database)의 일반 연산들과 병행하는 것은 심각한 응답시간의 지연을 초래한다. 따라서 운용 데이터베이스와는 별도로 과거로부터 누적된 방대한 양의 데이터를 통합 관리할 필요성이 증

가하게 되었으며, 이를 위한 대량의 정보 저장소로서 데이터 웨어하우스(data warehouse)라는 새로운 개념이 등장하게 되었다.[IK93][IM01]

데이터 웨어하우스에는 매우 많은 데이터가 저장되므로 의사결정을 위한 복잡하고 분석적인 질의들의 처리를 위하여 많은 데이터에 접근하여야 한다.[Inm96] 따라서 데이터 웨어하우스에서 사용자의 요구를 빠르게 처리할 수 있는 기법의 개발은 중요한 의미를 갖는다.[GHQ95][YL95][CS96] 특히, 데이터 웨어하우스 환경에서 의사결정시 빈번하게 사용되는 집계 함수(aggregate function) 및 조인은 기본적인 관계 연산인 실렉트, 프로젝트 연산과는 달리 질의처리 비용이 크므로 이에 대한 연구는 매우 중요하다.

본 논문에서는 비트맵 조인 인덱스(bitmap join index)[OQ97][CI98][WB98][WU99]를 이용하여 집계 함수와 조인을 포함하는 질의를 효과적으로 처리하기 위한 새로운 기법을 제안하고자 한다. 제안하는 기법은 조인을 처리하기 전에 차원 테이블을 미리 그룹핑한 후 비트맵 조인 인덱스를 이용하여 조인을 처리함으로써 사실 테이블만을 검색하여 집계함수를 처리할 수 있는 장점을 갖는다. 정량적인 성능분석을 위하여 기존의 기법과 제안하는 기법에 대한 비용모델을 정립하고 시뮬레이션을 수행함으로써 기존의 방법보다 적은 수의 디스크 액세스만으로 질의를 처리할 수 있음을 보인다.

본 논문의 구성은 다음과 같다. 제 2장에서는 데이터 웨어하우스의 스타 스키마(star schema)와 OLAP 질의에 대해서 설명하고, 제 3장에서는 비트맵 조인 인덱스의 특징과 이를 이용한 기존의 질의 처리 기법에 대해서 설명한다. 제 4장에서는 비트맵 조인 인덱스를 이용하여 조인과 집계함수를 포함하는 질의의 효율적인 처리 기법을 제안한다. 제 5장에서는 성능 평가를 통하여 제안한 기법의 우수성을 규명한다. 끝으로 제 6장에서는 본 논문의 공헌을 요약하고, 결론을 내린다.

Ⅱ. 데이터 웨어하우징 환경의 연구 배경

본 장에서는 논문의 연구 배경인 데이터 웨어하우스의 스키마와 OLAP(online analytic processing) 질의에 대해서 설명한다. 제 2.1절에서는 데이터 웨어하우스에 적합한 모델링 방법인 스타 스키마에 대해서 설명한다. 제 2.2절에서는 데이터 웨어하우징 환경에서 빈번하게 사용되는 OLAP 질의에 대해서 설명한다.

1. 스타 스키마

다차원 데이터 모델(multidimensional data model)은 데이터 웨어하우스 환경에 적합한 모델링 기법으로 알려져 있다.[Kim96] 다차원 모델은 사용자의 관점에서 중요한 데이터를 포함하는 테이블을 스키마의 중심에 놓고, 이를 설명하기 위한 테이블을 주변에 배치한다. 중심에 있는 테이블을 사실 테이블(fact table)이라고 하고, 주변에 있는 테이블을 차원 테이블(dimension table)이라고 한다. 이러한 형태의 스키마는 별 모양을 가지므로 스타 스키마(star schema)라고 부른다. 일반적으로 사실 테이블은 차원 테이블에 비해 그 크기가 매우 크다. 그리고 사실 테이블의 기본 키(primary key)는 모든 차원 테이블의 기본 키들의 결합된 형태로 구성된다. 이러한 결합 키로 인하여 사실 테이블과 차원 테이블간의 조인은 질의 처리시 매우 빈번하게 발생된다.

<그림 1>은 판매 업체의 판매실적을 나타내기 위한 스타 스키마의 예이다. Sales 테이블은 사실 테이블이며, Time, Product, Store, Customer 테이블은 차원 테이블이다. 볼드는 각 테이블의 이름, 언더라인은 그 테이블의 기본 키(primary key), 이탤릭은 외래 키(foreign key)를 나타낸다.

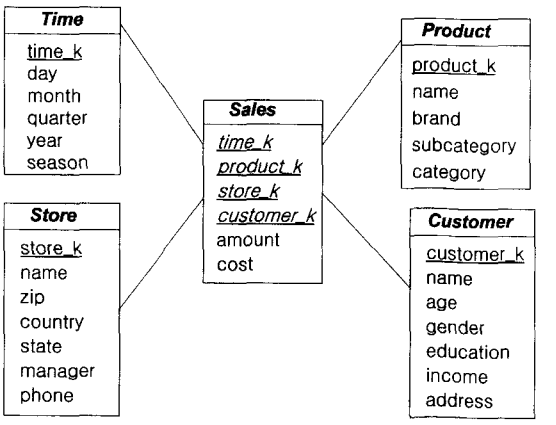


그림 1. 스타 스키마의 예
Fig. 1. An Example Star schema.

2. OLAP 질의

데이터 웨어하우징 환경에서 널리 사용되는 OLAP 질의에서는 조인 연산이 빈번하게 사용된다. 이러한 조

인은 차원 테이블의 기본 키와 사실 테이블의 외래 키를 주 대상으로 수행된다.[OQ97] 데이터 웨어하우징 환경에서는 사실 테이블과 차원 테이블의 크기가 매우 크므로 조인 처리비용이 매우 크다. 또한, OLAP 질의는 분석적인 질의를 처리하기 위하여 집계함수를 빈번하게 사용한다. 이러한 집계함수는 사실 테이블과 차원 테이블간의 조인 후에 처리되어지며, 집계 속성은 대부분의 경우 사실 테이블에 있는 속성 값이 된다.

```
SELECT      P.brand, SUM(S.cost)
FROM        Product P, Sales S
WHERE       P.product_k = S.product_k
            AND     P.category = 'Food'
GROUP BY    P.brand;
```

그림 2. 집계함수를 포함하는 OLAP 질의의 예
Fig. 2. An Example of OLAP Query with Aggregate Function.

<그림 2>는 <그림 1>의 스타 스키마 환경에서 category가 'Food'인 상품에 대해서, 각 brand별로 총 판매액이 얼마인지 검색하고자 하는 OLAP 질의를 SQL의 형태로 표현한 것이다. 이 질의에서 그룹핑 속성(grouping attribute)은 차원 테이블의 한 속성인 brand이고 집계 속성(aggregate attribute)은 사실 테이블의 한 속성인 cost이다. 따라서 사실 테이블인 Sales 테이블과 차원 테이블인 Product 테이블을 조인하여 조인 결과에 포함되어 있는 그룹핑 속성으로 조인 결과를 그룹핑하고 집계 속성에 대해서 집계함수 연산을 처리한다. 즉, 조인 결과에 대해서 그룹핑 속성을 brand로 집계 속성을 cost로 하여 SUM을 계산한다.

Ⅲ. 기존의 접근 방법

본 장에서는 비트맵 조인 인덱스를 이용하여 집계함수와 조인을 포함하는 질의를 처리하는 기존의 방법에 대해서 설명한다. 먼저, 제 3.1절에서는 비트맵 조인 인덱스에 대해 설명하고, 제 3.2절에서는 기존의 방법이 비트맵 조인 인덱스를 이용하여 조인과 집계함수를 포함하는 질의를 어떻게 처리하는가에 대해서 설명한다.

1. 비트맵 조인 인덱스

비트맵 조인 인덱스(bitmap join index)는 데이터 웨어하우스에서 많이 사용되는 비트맵 인덱스(bitmap index)를 조인 비용을 줄이기 위하여 응용한 인덱스이며, 데이터 웨어하우스와 같은 갱신이 거의 일어나지 않는 환경에 적합하다.[OQ96] 우선, 비트맵 인덱스는 각 속성 값에 대하여 하나씩 생성되며, 각 레코드마다 하나의 비트를 할당하여 해당 레코드가 그 값을 갖고 있으면 1을 저장하고 아니면 0을 저장한다. 특히, 스타 스키마의 경우 차원 테이블의 크기는 사실 테이블에 비해 매우 작으므로 사실 테이블에 존재하는 각 차원의 조인 속성에 대해서 비트맵 인덱스를 사용하면 일반적인 RID(record identifier)를 이용한 인덱스보다 훨씬 적은 공간을 차지하게 된다. 또한, AND, OR, NOT과 같은 논리 연산에 대해서도 각 비트맵 인덱스에 대해 비트 연산을 수행함으로써 그 결과를 빠르게 얻을 수 있다. 따라서 비트맵 조인 인덱스는 비트맵 인덱스의 장점을 이용하여 스타 스키마에서 조인을 효율적으로 처리한다.

비트맵 조인 인덱스는 사실 테이블의 각 RID에 대해서 해당되는 차원 테이블의 기본키 값을 비트맵 인덱스의 형태로 표현을 한다. 비트맵 조인 인덱스는 이차원 배열의 형태로서 표현이 되며, 이차원 배열의 행은 사실 테이블의 RID 리스트이고 열은 차원 테이블의 기본 속성 값이 된다. 만약, 이차원 배열에서 행과 열이 교차하는 부분이 1로 저장되어 있으면 그 위치에 해당하는 사실 테이블의 한 레코드와 차원 테이블의 그 기본 속성 값을 갖는 레코드가 조인된다는 의미이다. 각 행과 열이 각각 사실 테이블과 차원 테이블의 어느 레코드와 대응되는 지에 대한 정보는 시스템 카탈로그에 별도로 저장된다.

<그림 3>은 차원 테이블인 Product 테이블과 사실 테이블인 Sales 테이블간의 비트맵 조인 인덱스의 예를 보여준다. <그림 3(a)>는 차원 테이블인 Product 테이블이고, <그림 3(b)>는 사실 테이블인 Sales 테이블이다. 그리고 <그림 3(c)>는 Product 테이블과 Sales 테이블간의 비트맵 조인 인덱스이다. <그림 3(c)>의 비트맵 조인 인덱스의 열은 Product 테이블의 product_k에 대응되고, 행은 사실 테이블의 RID에 대응된다. 만약, Product 테이블의 product_k가 1인 레코드는 <그림 3(b)>인 Sales 테이블에서 RID2와 RID10인 레코드와 조인을 하게 되므로 <그림 3(c)>의 비트맵 조인 인덱스에서 RID2, RID10인 위치가 1로 저장되고 다른 레코드들은 0으로 저장된다.

product_k	name	brand	subcategory	category
1	Lasagna	Cold Gourmet	Frozen Foods	Food
2	Beef Stew	Cold Gourmet	Frozen Foods	Food
3	Turkey Dinner	Frozen Bird	Frozen Foods	Food
4	Chicken Dinner	Frozen Bird	Frozen Foods	Food
5	Extra Nougat	Chewy Industries	Candy	Food
6	Lots of Nuts	Chewy Industries	Candy	Food
7	Sweet Tooth	Chewy Industries	Candy	Food
8	Fizzy Light	Big Can	Soft Drinks	Drinks
9	Fizzy Classic	Big Can	Soft Drinks	Drinks
10	Athletic Drink	Big Can	Soft Drinks	Drinks

(a) Product 테이블

	time_k	product_k	store_k	customer_k	amount
RID1	1	3	1	15	58
RID2	2	1	1	16	76
RID3	3	2	1	14	97
RID4	6	5	1	9	64
RID5	7	8	1	14	31
RID6	22	9	11	10	83
RID7	5	4	1	19	72
RID8	11	7	11	5	18
RID9	12	9	11	14	34
RID10	13	1	11	20	20
RID11	124	6	11	2	11
RID12	125	3	11	4	22
RID13	181	2	11	17	48
RID14	20	7	11	19	60
RID15	21	9	11	11	86
RID16	46	8	1	18	74
RID17	34	9	11	5	18
RID18	35	3	11	16	35
RID19	139	6	8	7	75
RID20	140	3	8	10	51

(b) Sales 테이블

	1	2	3	4	5	6	7	8	9	10
RID1	0	0	1	0	0	0	0	0	0	0
RID2	1	0	0	0	0	0	0	0	0	0
RID3	0	1	0	0	0	0	0	0	0	0
RID4	0	0	0	0	1	0	0	0	0	0
RID5	0	0	0	0	0	0	0	1	0	0
RID6	0	0	0	0	0	0	0	0	1	0
RID7	0	0	0	1	0	0	0	0	0	0
RID8	0	0	0	0	0	0	1	0	0	0
RID9	0	0	0	0	0	0	0	0	1	0
RID10	1	0	0	0	0	0	0	0	0	0
RID11	0	0	0	0	0	1	0	0	0	0
RID12	0	0	1	0	0	0	0	0	0	0
RID13	0	1	0	0	0	0	0	0	0	0
RID14	0	0	0	0	0	0	1	0	0	0
RID15	0	0	0	0	0	0	0	0	1	0
RID16	0	0	0	0	0	0	0	1	0	0
RID17	0	0	0	0	0	0	0	0	1	0
RID18	0	0	1	0	0	0	0	0	0	0
RID19	0	0	0	0	0	1	0	0	0	0
RID20	0	0	1	0	0	0	0	0	0	0

(c) Product-Sales 비트맵 조인 인덱스

그림 3. 비트맵 조인 인덱스의 예.
Fig. 3. An Example of Bitmap Join Index.

2. 비트맵 조인 인덱스를 이용한 기존의 처리 기법

비트맵 조인 인덱스는 조인의 처리 성능을 효과적으로 하기 위한 인덱스이므로 이 인덱스를 이용하여 집계함수를 처리하는 방법에 대한 연구 결과는 별도로 제안된 바 없다. 그러나 일반적인 데이터베이스 환경에서 사용하는 집계함수 처리 방법을 적용하여 사용자의 질의에 포함되어 있는 집계함수를 처리할 수 있다.[Gra93][Mur92]

사용자의 질의에 차원 테이블과 사실 테이블간의 조인과 SUM, MAX, MIN, COUNT, AVG와 같은 집계함수가 포함되어 있는 경우의 처리는 다음과 같이 수행된다. 먼저, 비트맵 조인 인덱스를 이용하여 조인 연산을 처리하여 그 결과를 임시 테이블에 저장한다. 또한, 집계함수를 처리하기 위해서 먼저 SQL의 GROUP BY절에 포함되어 있는 속성 값에 의해 그룹핑을 수행한다. 그룹핑을 하기 위해서는 먼저 임시 테이블을 해당되는 그룹핑 속성 값으로 정렬하고 그 속성 값에 따라 그룹핑을 수행한다. 끝으로, 집계함수를 처리하기 위해서 그룹핑되어 있는 임시 테이블의 레코드를 검색한다.

<그림 2>의 OLAP 질의를 예로 사용하여 비트맵 조인 인덱스를 이용하여 조인과 집계함수를 포함하는 질의를 처리하는 방법에 대해서 설명한다. 우선, OLAP 질의에서 category가 'Food'인 선택조건을 만족하는 레코드를 먼저 차원 테이블에서 선택한다. 따라서 차원 테이블의 레코드 중에서 product_k가 1, 2, 3, 4, 5, 6, 7인 레코드가 선택된다. 이때, <그림 3(c)>의 비트맵 조인 인덱스에서 선택된 product_k에 대응되는 비트맵을 찾아서 모두 OR 연산을 수행한다. 이 결과, OR 연산을 수행한 결과 값은 11110011011111000111이 된다. 이 비트맵에서 1로 저장되어 있는 비트들의 위치가 사실 테이블의 RID를 나타낸다. 따라서 조인 후의 임시 테이블은 비트맵에서 1로 저장되어 있는 사실 테이블의 레코드와 그 레코드와 조인하는 차원 테이블의 레코드를 포함한다. 이러한 임시 테이블을 그룹핑 함으로써 실제 집계함수를 계산한다.

Ⅳ. 제안하는 방법

본 장에서는 비트맵 조인 인덱스를 이용하여 집계함수를 효과적으로 처리하는 새로운 기법을 제안한다. 먼저, 제 4.1절에서는 제안하는 기법의 기본 전략에 대해서 설명하고, 제 4.2절에서는 제안하는 기법에 대해서 자세히 설명한다.

1. 기본 전략

집계함수와 조인이 포함된 질의를 제 3장에서 설명

한 기존의 방법을 이용하여 처리하는 경우 성능저하가 심각하다. 그 이유는 스타 스키마 환경에서 사실 테이블의 크기는 차원 테이블에 비해 상당히 크므로 집계함수를 처리하기 위해서 조인 후의 임시 테이블을 정렬하는 과정은 큰 처리비용을 요구하기 때문이다. 이러한 정렬은 외부정렬에 해당되므로 디스크 액세스 수는 임시 테이블의 크기에 비례하게 되고, 조인 후의 임시 테이블은 사용자의 질의에서 차원 테이블에 대한 질의 조건이 존재하지 않는 경우에는 사실 테이블의 크기보다도 커지게 된다.

또한, 임시 테이블을 그룹핑 속성으로 정렬 한 후에 사용자 질의에 포함된 집계 함수를 처리하기 위해서는 임시 테이블을 한 번 검색해야 한다. 이 비용 또한 사실 테이블보다 크기가 커진 임시 테이블을 검색함으로써 사실 테이블을 검색하는 디스크 액세스 수보다 더 많은 비용을 요구하게 된다.

기존의 방법의 근본적인 문제점은 조인의 결과로서 큰 임시 테이블을 만들고 그 테이블을 그룹핑하기 위해서 외부정렬을 수행함으로써 많은 처리비용을 유발한다는 것이다. 이러한 문제점을 해결하기 위하여 본 연구에서는 임시 테이블보다 상대적으로 크기가 작은 차원 테이블을 정렬을 통하여 미리 그룹핑하고, 이를 기반으로 조인을 수행함으로써 디스크 액세스 수를 줄일 수 있는 방법을 제안하고자 한다.

OLAP 질의에서 집계 속성은 사실 테이블의 한 속성이 되며, 그룹핑 속성은 차원 테이블의 한 속성이 된다. 따라서 기존의 방법에서는 조인 결과를 저장하는 임시 테이블에서 차원 테이블의 그룹핑 속성에 대해서 그룹핑을 하고, 그룹핑 결과로부터 다시 사실 테이블의 집계 속성에 대해서 집계함수 연산을 처리한다. 그러나 제안하는 방법에서는 차원 테이블에 대해서 그룹핑 속성이 존재하므로 조인을 처리하기 전에 크기가 작은 차원 테이블을 미리 그룹핑하여 처리함으로써 작은 처리비용을 가지고 임시 테이블을 그룹핑하는 것과 동일한 결과를 얻을 수 있다.

2. 알고리즘

<그림 4>는 사용자의 OLAP 질의가 사실 테이블과 차원 테이블의 조인뿐만 아니라 사실 테이블의 속성 값에 대하여 집계함수 연산을 포함한 경우의 처리 알고리즘을 개략적으로 나타낸 것이다.

<그림 4>의 단계 1과 2에서는 사용자의 질의에서 선택조건을 만족하는 레코드들을 임시 테이블에 저장하고 임시 테이블을 그룹핑 속성으로 정렬한다. 선택조건을 만족하는 레코드들을 선택함으로써 임시 테이블에 저장될 레코드의 수가 줄어들게 된다. 따라서 이 임시 테이블을 외부 정렬을 통하여 정렬을 하는 비용도 줄어들게 된다.

1. 차원 테이블을 탐색하여 선택조건을 만족하는 레코드들을 임시 테이블에 저장한다.
2. 임시 테이블을 그룹핑 속성으로 정렬하여 그룹핑한다.
3. 비트맵 조인 인덱스를 검색하여 각 그룹마다 하나의 비트맵을 얻는다.
4. 각 그룹에 대해서, 비트맵에서 1로 저장된 위치에 있는 사실 테이블의 레코드들을 검색하여 집계함수를 계산한다.

그림 4. 제안하는 방법의 처리 알고리즘
Fig. 4. Proposed Algorithm.

단계 3에서는 각 그룹마다 하나의 비트맵을 얻는다. 만약, 한 그룹에 레코드 R1, R2, R3이 속한다면 비트맵 조인 인덱스에서 레코드 R1, R2, R3에 해당하는 비트맵을 검색하여 각 레코드에 대해서 하나씩의 비트맵을 얻는다. 그리고 이 비트맵에 대해서 OR 연산을 수행함으로써 한 그룹에 대한 비트맵을 얻을 수가 있다. 단계 4에서, 각 그룹마다 얻어진 비트맵에서 어떤 한 비트가 1로 저장되어 있다는 것은 그것에 대응되는 사실 테이블의 한 레코드가 그 그룹에 속한다는 것을 의미한다. 따라서 실제적인 집계함수의 계산은 각 그룹마다 얻은 비트맵을 조사하여 1로 저장되어 있는 것과 대응되는 사실 테이블의 레코드를 검색하여 처리를 한다. 그러나 COUNT 집계함수인 경우에는 각 그룹마다 얻은 비트맵에서 1로 저장되어 있는 비트 수를 계산함으로써 사실 테이블을 검색하는 처리비용을 제거 할 수 있다.

<그림 5>는 제안하는 알고리즘을 설명하기 위한 예이다. <그림 2>의 OLAP 질의를 처리하기 위해서 먼저 차원 테이블인 Product 테이블에서 선택조건을 만족하는 레코드를 검색하여 임시 테이블에 저장한다. 그리고 임시 테이블을 GROUP BY절의 차원 테이블의 속성인 brand에 대해서 정렬을 하여 그룹핑한다. 각 그룹에 속한 레코드에 해당하는 비트맵을 조사하여 각 비트맵에 대해서 OR 연산을 수행하여 그룹의 비트맵을

얻는다. <그림 5(a)>가 정렬 후의 임시 테이블을 나타내고 <그림 5(b)>가 각 그룹에 해당하는 비트맵을 나타낸다.

예를 들어, <그림 5(a)>의 정렬된 임시 테이블에서 brand가 'Chewy Industries'인 그룹에는 차원 테이블의 product_k가 5, 6, 7인 레코드가 속한다. 따라서 <그림 3(c)>의 비트맵 조인 인덱스에서 5, 6, 7인 비트맵을 찾아 OR 연산을 수행하면 이 그룹의 비트맵은 00010001001001000010이 된다. 만약, OLAP 질의의 집계함수가 COUNT라면 사실 테이블의 액세스 없이 비트맵의 1로 저장되어 있는 각 비트의 수를 계산하여 5라는 결과 값을 얻을 수 있다. <그림 2>와 같이 집계함수가 COUNT가 아닌 경우에는 사실 테이블을 액세스하여 집계함수를 처리한다. 따라서 1로 저장되어 있는 비트에 해당되는 사실 테이블의 RID4, RID8, RID11, RID14, RID19를 검색하여 집계함수를 계산한다.

brand	비트맵
Chewy Industries	00010001001001000010
Cold Gourmet	01100000010010000000
Frozen Bird	10000010000100000101

(a) 정렬 후의 임시 테이블

product_k	name	brand	subcategory	category
5	Extra Nougat	Chewy Industries	Candy	Food
6	Lots of Nuts	Chewy Industries	Candy	Food
7	Sweet Tooth	Chewy Industries	Candy	Food
1	Lasagna	Cold Gourmet	Frozen Foods	Food
2	Beef Stew	Cold Gourmet	Frozen Foods	Food
3	Turkey Dinner	Frozen Bird	Frozen Foods	Food
4	Chicken Dinner	Frozen Bird	Frozen Foods	Food

(b) 각 그룹에 해당하는 비트맵

그림 5. 제안하는 기법의 처리 예.
Fig. 5. A Processing Example by the Proposed Method.

V. 성능 평가

본 장에서는 비용모델을 이용한 성능 분석을 통하여 제안하는 기법의 성능을 평가한다. 제 1절에서는 기존의 방법과 제안하는 방법의 비용모델을 정립하고, 제 2절에서는 다양한 시뮬레이션을 통하여 성능을 분석한다.

1. 비용모델

데이터 웨어하우스에서 질의 처리의 주요 비용은 디스크 액세스 시에 발생하므로 본 논문에서는 디스크 액세스 수를 기반으로 성능 평가를 수행하고자 한다. 먼저, 디스크 액세스 수를 분석하기 위하여 필요한 인자를 정의하고, 기존의 방법과 제안하는 방법으로 질의를 처리하는 경우의 디스크 액세스 수를 추정한다.

데이터 웨어하우스는 매우 방대한 양의 데이터를 저장하므로 비트맵 조인 인덱스는 디스크에 존재한다. 이 차원 배열의 형태를 가지는 비트맵 조인 인덱스는 효과적인 질의 처리를 위하여 열 단위로 디스크에 저장됨을 가정한다. 이 결과, 사용자 질의에서 선택 조건을 만족하는 차원 테이블의 레코드와 대응되는 (비트맵 벡터의 크기)/(디스크 블록의 크기)의 수만큼의 디스크만 액세스하면 된다. 또한, 제안하는 방법에서는 차원 테이블을 그룹핑하고 한 그룹에 대해서 하나의 비트맵을 만듦으로 각 그룹에 존재하는 비트맵은 주기억장치에 존재할 수 있다.

다음은 기존의 방법과 제안하는 방법의 디스크 액세스 수를 추정하기 위한 비용모델에 대해서 설명한다. 우선, 사용자의 질의에 선택조건이 포함되어 있는 경우에는 차원 테이블에 선택률(selectivity)을 적용하여야 한다. 그러나 기존의 방법이나 제안하는 방법에서 첫 단계로서 공통적으로 필요한 부분이므로 본 연구에서는 질의의 선택조건을 만족하는 레코드를 찾기 위한 처리비용은 고려하지 않는다. <표 1>은 디스크 액세스 수를 분석하기 위하여 필요한 인자들을 정리한 것이다.

비트맵 조인 인덱스를 이용하여 조인과 집계함수를 포함하는 질의를 처리하는 기존의 방법에 대한 처리비용은 다음과 같다. 첫 번째, 기존의 방법에서는 선택조건을 만족하는 레코드에 대해서 비트맵 조인 인덱스를

표 1. 비용모델에서 사용되는 인자들
Table 1. Factors Used in the Cost Model.

인 자	인자의 의미
n_d	차원 테이블의 레코드 수
n_f	사실 테이블의 레코드 수
s_d	차원 테이블의 레코드 크기
s_f	사실 테이블의 레코드 크기
b	디스크 블록의 크기
rsel	질의에 대한 선택률
rparti	차원 테이블의 조인 참여 비율
k	외부 정렬 시 사용될 디스크 블록 수

액세스하여 사실 테이블의 RID에 대해서 1로 저장되어 있는 레코드가 있는지를 조사하여야 한다. 이 비용은 (1-1)과 같다. 이 식에서 비트 맵 조인 인덱스가 비트 단위의 크기를 가지므로 블록의 크기를 비트 단위로 맞추기 위해 8을 곱하였다. 두 번째, 조인에 참여하는 비트맵을 얻으면 사실 테이블의 해당하는 레코드를 읽음으로써 그 레코드와 조인되어야 하는 차원 테이블의 해당 레코드를 얻을 수 있다. 그리고 두 테이블을 조인한 결과를 임시 테이블에 저장한다. 이 비용은 (1-2)와 같다. 세 번째, 집계함수 연산을 처리하기 위하여 조인 결과가 저장되어 있는 임시 테이블을 K원 병합(K-way merge)[FW72]를 사용하여 외부 정렬하여야 한다. 이 비용은 (1-3)과 같다. 마지막으로, 사용자의 질의에 포함된 집계함수를 처리하기 위해서 임시테이블을 한 번 읽는다. 이 비용은 (1-4)와 같다.

따라서, 비트맵 조인 인덱스를 이용하여 기존의 방법으로 질의를 처리할 때 발생하는 총 디스크 액세스 수는 다음과 같다.

Total Cost = Bitmap Join Index Read Cost
+ Table Read-Write Cost
+ Temp_Table Sort Cost
+ Temp_Table Read Cost

$$\text{Total Cost} = \lceil (r_{sel} \times r_{parti} \times n_d) \rceil \times \lceil \frac{n_f}{8 \times b} \rceil \quad (1\text{-}1)$$

$$+ (n_f \times r_{sel}) + \lceil \frac{(s_d + s_f) \times (n_f \times r_{sel})}{b} \rceil \quad (1\text{-}2)$$

$$+ 2 \times \lceil \frac{(s_d + s_f) \times (n_f \times r_{sel})}{b} \rceil \times \log_k \lceil \frac{(s_d + s_f) \times (n_f \times r_{sel})}{b} \rceil \quad (1\text{-}3)$$

$$+ \lceil \frac{(s_d + s_f) \times (n_f \times r_{sel})}{b} \rceil \quad (1\text{-}4)$$

다음은 제안하는 방법의 처리비용이다. 첫 번째, 차원 테이블에서 사용자 질의에 포함된 선택조건을 만족하는 레코드를 검색하여 임시 테이블에 저장한다. 이 비용은 (2-1)과 같다. 두 번째, 이 임시 테이블을 먼저 차원 테이블에 존재하는 그룹핑 속성으로 K원 병합 외부 정렬을 사용하여 그룹핑한다. 이 비용은 (2-2)와 같다. 세 번째, 각 그룹의 해당되는 레코드에 대해서 비트맵 조인 인덱스에서 해당되는 비트맵을 얻기 위하여 비트맵 조인 인덱스를 액세스 한다. 이 비용은 (2-3)과 같다. 마지막으로, 집계함수의 실제 처리 연산은 각 그룹마다 존재하는 비트맵에서 1로 저장되어 있는 사실 테이블의 레코드를 액세스함으로서 처리된다. 이 비용은 (2-4)와 같다.

따라서 비트맵 조인 인덱스를 이용하여 제안하는 방법으로 질의를 처리할 때 발생하는 총 디스크 액세스 수는 다음과 같다.

Total Cost = Temp Table Write Cost
+ Temp Table Sort Cost
+ Bitmap Join Index Read Cost
+ Fact Table Read Cost

$$\text{Total Cost} = \lceil \frac{s_d \times (n_d \times r_{sel})}{b} \rceil \quad (2\text{-}1)$$

$$NK + 2 \times \lceil \frac{s_d \times (n_d \times r_{sel})}{b} \rceil \times \log_k \lceil \frac{s_d \times (n_d \times r_{sel})}{b} \rceil \quad (2\text{-}2)$$

$$+ \lceil (r_{sel} \times r_{parti} \times n_d) \rceil \times \lceil \frac{n_f}{8 \times b} \rceil \quad (2\text{-}3)$$

$$+ (n_f \times r_{sel}) \quad (2\text{-}4)$$

2. 시뮬레이션 결과

본 절에서는 시뮬레이션을 통하여 제 5.1절에서 설명한 비용모델의 인자 변화에 따른 두 가지 처리 방법들의 성능상의 경향에 대하여 논의한다. 먼저, 시뮬레이션에 사용된 인자들의 기본 값에 대하여 설명하고 시뮬레이션 결과를 제시한다.

본 시뮬레이션은 TPC-H 벤치마크[TPC99]에서 사용하는 스타 스키마 환경을 참조하여 수행하였다. TPC-H 벤치마크는 대량의 데이터들에 대한 복잡하고 긴 질의들을 필요로 하는 데이터 웨어하우스 시스템과 의사 결정 지원 시스템의 처리 능력을 테스트하기 위한 벤치마크이다.

시뮬레이션을 위하여 차원 테이블의 레코드 크기와 사실 테이블의 레코드 크기는 각각 112와 144로 지정하였다. 그리고 차원 테이블의 레코드 수와 사실 테이블의 레코드 수는 각각 800,000과 6,000,000으로 지정하였다. 그리고 선택률과 참여비율은 0.05와 0.8로 지정하였으며 시스템 버퍼의 수와 디스크 블록의 크기는 100과 4K로 지정하였다.

첫 번째 시뮬레이션에서는 차원 테이블의 레코드 수를 100,000부터 800,000까지 변화시킴으로써 사실 테이블에 비해 차원 테이블의 레코드 수가 상대적으로 작

아지는 경우의 성능상 경향을 관찰하였다. <표 2>는 각각 집계함수가 COUNT인 경우와 COUNT가 아닌 경우에 대해서 타원 테이블의 레코드 수 변화에 따른 디스크 액세스 수를 나타낸다.

표 2. 차원 테이블의 레코드수 변화에 따른 디스크 액세스 수

Table 2. The Number of Disk Accesses Over the Number of Tuples In Dimension Table.

차원 테이블의 레코드 수	기존의 방법	집계함수가 COUNT인 경우 (제안하는 방법)	집계함수가 COUNT가 아닌 경우 (제안하는 방법)
100,000	1,905,982	738,266	1,036,614
200,000	2,641,982	1,476,896	1,773,126
300,000	3,377,982	2,215,732	2,509,669
400,000	4,113,982	2,954,681	3,246,229
500,000	4,849,982	3,693,752	3,982,807
600,000	5,585,982	4,432,902	4,719,398
700,000	6,321,982	5,172,118	5,455,998
800,000	7,057,982	5,911,367	6,192,603

<그림 6>은 이 시뮬레이션 결과를 그래프화한 것이다. 가로축은 차원 테이블의 레코드 수를 나타내며, 세로축은 디스크 액세스 수를 나타낸다. 결과에 의하면, 제안된 기법은 기존 기법과 비교하여 모든 경우에 나은 성능을 보였으며, 특히 집계함수가 COUNT인 경우에는 기존 방법에 비해 디스크 액세스 수가 약 31%정도 감소된 것으로 나타났다.

두 번째 시뮬레이션에서는 사실 테이블의 레코드 수를 6,000,000부터 20,000,000까지 변화시킴으로써 사실 테이블의 레코드 수가 차원 테이블의 레코드 수에 비해 상대적으로 커지는 경우의 성능 변화를 관찰하였다. <그림 7>은 이 시뮬레이션 결과를 나타낸 것이다. 가로축은 사실 테이블의 레코드 수를 나타내며, 세로축은 디스크 액세스 수를 나타낸다. 결과에 의하면, 사실 테이블의 레코드 수가 증가하더라도 제안하는 기법은 모든 경우에 대하여 기존의 기법보다 나은 성능을 보였으며, 집계함수가 COUNT인 경우에는 기존 방법에 비해 17% 정도 감소된 것으로 나타났다.

세 번째 시뮬레이션에서는 차원 테이블의 레코드가 사실 테이블에 모두 참여하지 않는 경우를 위하여 참여비율을 0.1부터 1.0까지 0.1씩 증가시키면서 성능상의 변화를 관찰하였다. <그림 8>은 이 시뮬레이션 결과를 나타낸 것이다. 시뮬레이션에서 사용되는 참여 비율은 비트맵 조인 인덱스의 크기에 영향을 미친다. 가로축은 차원 테이블에 대한 참여비율을 나타내며 세로축은 디스크 액세스 수를 나타낸다. 결과에 의하면, 참여비율이 0.1부터 1.0으로 변하더라도 모든 경우에 나은 성능을 보였으며 집계함수가 COUNT인 경우에는 기존 방법에

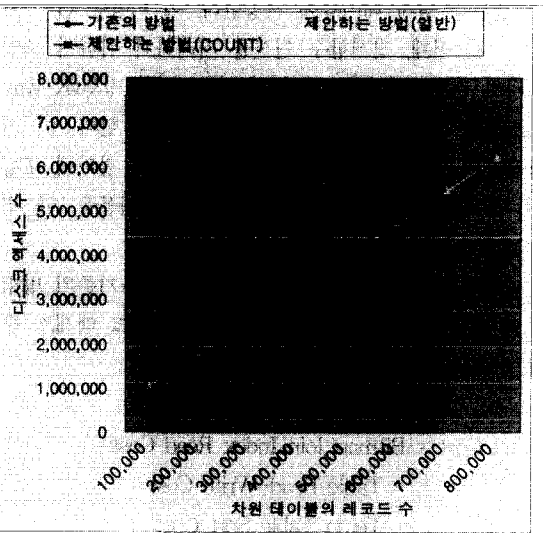


그림 6. 차원 테이블의 레코드 수 변화에 따른 성능변화

Fig. 6. Performance Over the Number of Tuples in Dimension Table.

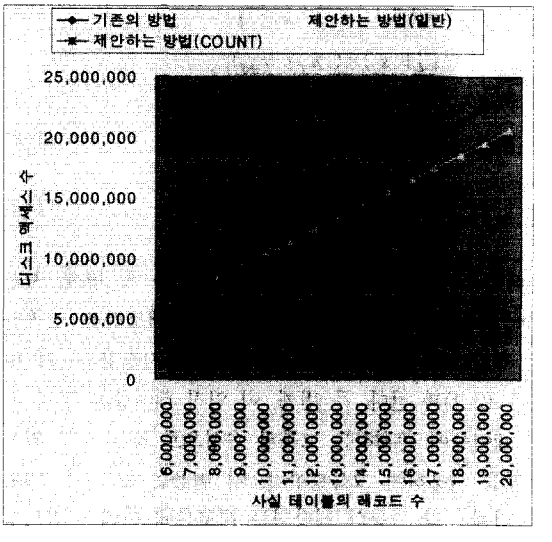


그림 7. 사실 테이블의 레코드 수 변화에 따른 성능변화

Fig. 7. Performance Over the Number of Tuples in Fact Table.

비해 디스크 액세스수가 약 20%정도 감소된 것으로 나타났다.

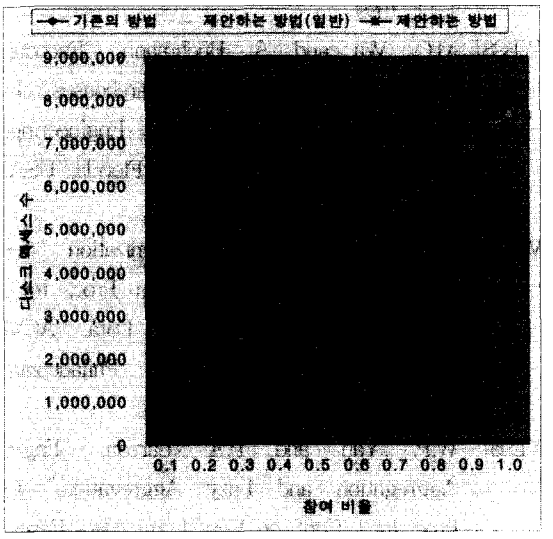


그림 8. 참여 비율 변화에 따른 성능변화
Fig. 8. Performance Over Participating Ratio.

VI. 결 론

데이터 웨어하우스는 대용량의 데이터를 저장하므로 데이터에 대한 OLAP 질의와 같은 복잡하고 분석적인 질의들을 처리하기 위해서 많은 처리비용이 사용된다. 또한, 스타 스키마 환경에서 OLAP 질의는 차원 테이블과 사실 테이블간의 조인을 요구하며 사실 테이블의 속성 값에 대한 요약된 정보를 요구한다. 본 논문에서는 집계함수와 조인이 포함된 질의를 효과적으로 처리하는 방안에 관하여 논의하였다.

상용 데이터 웨어하우스에서는 조인 연산을 효과적으로 처리하기 위하여 비트맵 조인 인덱스를 사용한다.[Inf98] 그러나 비트맵 조인 인덱스를 사용하더라도 집계함수를 처리하기 위해서는 기존의 데이터베이스에서 사용하였던 정렬-합병 방법을 사용한다. 이러한 방법은 사실 테이블의 레코드 수가 차원 테이블의 레코드 수보다 상대적으로 많은 스타 스키마 환경에서는 효과적이지 않다. 그 이유는 사실 테이블과 차원 테이블을 조인한 후에 집계함수를 계산하기 위해서 조인 결과에 대하여 그룹핑을 수행하므로 정렬을 하기 위한 디스크 액세스 수가 증가하게 되기 때문이다.

따라서 본 논문에서는 차원 테이블에 대하여 먼저 그룹핑을 적용함으로써 조인 후의 결과 테이블을 정렬하기 위한 디스크 액세스 수를 줄이는 방법을 제안하였다. 제안된 기법은 스타 스키마 환경에서 집계함수에 대한 그룹핑 속성이 차원 테이블에 존재하고 집계 속성이 사실 테이블에 존재한다는 특성을 이용하여 테이블의 레코드 수가 작은 차원 테이블을 먼저 그룹핑한다. 이러한 기법은 조인을 효과적으로 처리하기 위한 인덱스를 사용하여 집계함수도 효과적으로 처리가 가능하다.

제안된 기법의 성능을 평가하기 위하여 대표적인 데이터웨어하우스 벤치마크인 TPC-H의 스타 스키마를 기반으로 기존의 기법과 시뮬레이션을 통한 성능 비교를 수행하였다. 시뮬레이션 결과에 의하면, 제안된 기법은 기존의 기법과 비교하여 모든 경우에서 나은 성능을 보였다. 본 연구 결과는 데이터 웨어하우징 환경에서 OLAP 질의의 처리 과정에 효과적으로 적용될 수 있다.

참 고 문 헌

[CI98] C.Y. Chan and Y.E. Ioannidis, "Bitmap Index Design and Evaluation," In Proc. Int'l. Conf. on Management of Data, ACM SIGMOD, pp. 355~366, Seattle, Washington, USA, June 1998.

[CS96] S. Chaudhuri and K. Shim, "Optimizing Queries with Aggregate Views," In Proc. Int'l. Conf. on Extending Database Technology, pp. 167~182, Avignon, France, March 1996.

[FW72] W.D. Frazer and C.K. Wong, "Sorting by Natural Selection," Communications of the ACM, Vol. 15, No. 10, pp. 910~913, October 1972.

[GHQ95] A. Gupta, V. Harinarayan, and D. Quass, "Aggregate-Query Processing in Data Warehousing Environments," In Proc. Int'l. Conf. on Very Large Data Bases, pp. 358~369, Zurich, Switzerland, September 1995.

[Gra93] Goetz Graefe, "Query Evaluation Techniques for Large Databases," ACM Computing Surveys, Vol. 25, No. 2, pp. 73~170, 1993.

[IH94] W.H. Inmon and R.D. Hackathorn, Using the Data Warehouse, John Wiley & Sons, 1994.

[Inf98] Informix White Paper, Informix Decision Support Indexing for the Enterprise Data Warehouse, 1998.

[Inm96] W.H. Inmon, Building the Data Warehouse, John Wiley & Sons, March 1996.

[Kim96] R. Kimball, The Data Warehouse Toolkit, John Wiley & Sons, 1996.

[Mur92] M. Muralikrishna, "Improved Unnesting Algorithms for Join Aggregate SQL Queries," In Proc. Int'l. Conf. on Very Large Data Bases, pp. 91~102, Vancouver, Canada, August 1992,

[OQ97] P. O'Neil and D. Quass, "Improved Query Performance with Variant Indexes," In Proc. Int'l. Conf. on Management of Data, ACM SIGMOD, pp. 38~49, Tucson, Arizona, USA, May 1997.

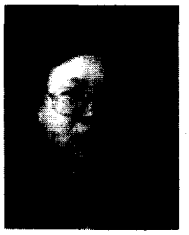
[TPC99] Transaction Processing Performance Council (TPC), TPC Benchmark H (Decision Support), Standard Specification Revision 1.2.1, 1999.

[WB98] M.C. Wu and A. Buchmann, "Encoded Bitmap Indexing for Data Warehouses," In Proc. Int'l. Conf. on Data Engineering, IEEE, pp. 220~230, Orlando, Florida, USA, February 1998.

[Wu99] M.C. Wu, "Query Optimization for Selections using Bitmaps," In Proc. Int'l. Conf. on Management of Data, ACM SIGMOD, pp. 227~238, Philadephia, Pennsylvania, USA, June 1999.

[YL95] W.P. Yan and P.A. Larson, "Eager Aggregation and Lazy Aggregation," In Proc. Int'l. Conf. on Very Large Data Bases, pp. 345~357, Zurich, Switzerland, September 1995.

저 자 소 개

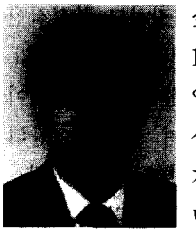


金 鎭 皓(正會員)

1982년 경북대학교 전자공학과 졸업(학사). 1985년 한국과학기술원 전산학과 졸업(석사). 1990년 한국과학기술원 전산학과 졸업(박사). 1995년~1996년 미국 미시간 대학교 객원 교수. 1990년~현재 강원대학교 전자계산학과 교수. <주관심분야 : OLAP, 데이터웨어하우징, 주기억장치 실시간 데이터베이스, 내장형 시스템, 웹 데이터베이스 등임>

金 潤 浩(正會員)

1999년 강원대학교 정보통신공학과 졸업(학사). 2001년 강원대학교 컴퓨터정보통신공학부 졸업(석사). 2001년~현재 쌍용정보통신(주) 근무. <주관심분야 : 데이터 웨어하우징, 실체 뷰 관리 기법, 동시성 제어 기법 등임>



金 尙 煜(正會員)

1989년 서울대학교 컴퓨터공학과 졸업(학사). 1991년 한국과학기술원 전산학과 졸업(석사). 1994년 한국과학기술원 전산학과 졸업(박사). 1991년 미국 스텐포드 대학 방문 연구원. 1994년~1995년 KAIST 정보정자연구소 연구원. 1999년~2000년 미국 IBM T. J. Watson 연구소 Post-Doc. 1995년~현재 강원대학교 컴퓨터정보통신공학부 부교수. <주관심분야 : 데이터베이스, 데이터 마이닝, 멀티미디어 내용 기반 검색, 주기억 DBMS, 공간 DBMS/GIS, 인터넷 데이터베이스 등임>