http://dx.doi.org/10.3745/KIPSTD.2012.19D.1.001

관계형 데이터 웨어하우스의 복잡한 질의의 처리 효율 향상을 위한 비트맵 조인 인덱스 선택에 관한 연구

안 형 근† · 고 재 진††

요 약

데이터 웨어하우스는 크기가 방대하기 때문에 인덱스의 선택은 질의어 처리의 효율성에 상대한 영향을 준다. 인덱스는 질의 처리 비용을 줄이지만, 그것이 차지하는 기억 영역과 데이터베이스의 변경에 따른 보수라는 비용이 수반된다. 데이터 웨어하우스에서 하나의 사실 테이블과 여러 개의 차원 테이블 사이의 조인을 행하는 스타 조인 질의어와 차원 테이블의 선택을 최적화하기 위해서 비트맵 조인 인덱스가 잘 적용된다. 비트맵 조인 인덱스는 이진수로 표현되기 때문에 저장 비용은 적게 들지만 인덱스 할 후보 속성들이 많이 생성되기 때문에 그 중에서 인덱스 할 속성들을 선택하는 일은 어려운 과제가 된다. 인덱스 선택은 일단 후보 속성들의 개수를 축소하고, 그 중에서 인덱스를 선택하게 된다.

본 논문에서는 데이터 마이닝 방법을 사용해서 비트 맵 조인 인덱스 선택 문제에서 후보 속성들의 개수를 축소하는 것을 해결한다. 질의어에 있는 속성들의 빈도에 기준해서 후보 속성들의 개수를 감소시키는 기존의 방법에 비해서 본 논문은 속성들의 빈도를 사용함과 동시에 차원 테이블의 크기, 차원 테이블의 튜플 크기, 디스크의 페이지 크기 등을 고려한다. 그리고 데이터마이닝 기법으로 빈발 항목집합을 마이닝하여 후보 속성들의 개수를 효과적으로 줄인다. 후보 속성집합들의 비트 맵 조인 인덱스에 비용함수를 적용해서 최소의 비용과 기억 영역 제한에 적합한 속성집합들의 비트 맵 조인 인덱스를 구한다. 본 논문의 방법의 효율성을 평가하기 위해서 기존의 방법들과 비교 분석을 한다.

키워드 : 비트맵 조인 인덱스, 빈발 항목집합, 데이터 마이닝, 관계형 데이터 웨어하우스

A Study on Selecting Bitmap Join Index to Speed up Complex Queries in Relational Data Warehouses

An Hyoung Geun† · Koh Jae Jin††

ABSTRACT

As the size of the data warehouse is large, the selection of indices on the data warehouse affects the efficiency of the query processing of the data warehouse. Indices induce the lower query processing cost, but they occupy the large storage areas and induce the index maintenance cost which are accompanied by database updates. The bitmap join indices are well applied when we optimize the star join queries which join a fact table and many dimension tables and the selection on dimension tables in data warehouses. Though the bitmap join indices with the binary representations induce the lower storage cost, the task to select the indexing attributes among the huge candidate attributes which are generated is difficult. The processes of index selection are to reduce the number of candidate attributes to be indexed and then select the indexing attributes.

In this paper on bitmap join index selection problem we reduce the number of candidate attributes by the data mining techniques. Compared to the existing techniques which reduce the number of candidate attributes by the frequencies of attributes we consider the frequencies of attributes and the size of dimension tables and the size of the tuples of the dimension tables and the page size of disk. We use the mining of the frequent itemsets as mining techniques and reduce the great number of candidate attributes. We make the bitmap join indices which have the least costs and the least storage area adapted to storage constraints by using the cost functions applied to the bitmap join indices of the candidate attributes. We compare the existing techniques and ours and analyze them in order to evaluate the efficiencies of ours.

Keywords : Bitmap Join Index, Frequent ItemSets, Data Mining, Relational Data Warehouse

※ 본 연구는 2009학년도 울산대학교 학술연구 지원으로 수행되었음.
† 정 회 원 : 울산대학교 전기공학부 외래강사
†† 정 회 원 : 울산대학교 전기공학부 교수(교신저자)

논문접수 : 2011년 8월 5일
수 정 일 : 1차 2011년 10월 17일, 2차 2011년 11월 10일
심사완료 : 2011년 11월 10일

1. 서 론

데이터 웨어하우스는 많은 기업의 결집된 데이터들을 저장하고, 기업 활동을 위해서 복잡한 질의를 통해서 접근된다. 데이터 웨어하우스는 스타 스키마(star schemas)나 스노우플레이크 스키마(snowflake schemas)와 같은 관계형 스키마를 사용하여 자주 모델링 한다. 이러한 스타 스키마는 하나의 큰 사실 테이블(fact table)과 여러 개의 차원 테이블(dimension table)들로 구성되고, 차원 테이블의 주 키(primary key)는 사실 테이블의 외래 키(foreign key)로 연결된다. 차원 테이블은 사실 테이블에 비해 상대적으로 작으며 거의 업데이트하지 않는다. 스타 스키마에서 질의어를 스타 조인 질의어(star join queries)라 하고, 하나의 사실 테이블과 여러 개의 차원 테이블들 사이의 다중 테이블 조인(multi-table join)이 필요하다. 조인에 참여하는 각 차원 테이블은 서술 속성(descriptive attribute)에 대한 다중 선택 술어(multiple selection predicate)를 갖고 있다.

데이터 웨어하우스에 대한 질의는 기업 활동에 있어 의사결정 지원을 위한 것이기 때문에 빠른 응답 시간을 필요로 한다. 만약 빠른 응답시간을 위한 효율적인 최적화 기법이 없다면 질의 처리가 장시간 걸릴 수 있기 때문에 최근에는 최적화 기술의 필요성이 대두되고 있으며, 복잡하고 시간이 많이 걸리는 의사 결정 지원 질의에 대처하기 위하여 효율적이고 정확한 물리적 설계(physical design) 기법이 필요하게 되었다[1]. 또한 데이터 웨어하우스에서 질의 실행의 가장 중요한 장애는 하나의 사실 테이블과 여러 개의 차원 테이블 사이의 조인이다. 조인 작업을 최적화기 위한 기술들은 크게 비중복 구조(non-redundant structure)와 중복 구조(redundant structure)로 나누고 있다. 비중복 구조 기법으로는 nested loop, sort merge join, hash join 기법 등이 있다. 비중복 구조는 조인된 테이블의 크기가 적당하고 두 개의 테이블 사이의 조인인 경우에 효율적이다. 하지만 데이터 웨어하우스에는 적합하지 않다는 단점이 있다[2]. 실체화 뷰(materialized view)나 조인 인덱스(join index) 같은 중복 구조는 여러 테이블들 사이의 조인 성능을 높이는데 효율적이고 데이터 웨어하우스의 인덱스를 최적화하는데 필요한 기법이다[3]. 하지만, 중복 구조는 저장 영역과 유지보수의 추가적인 비용이 필요하다는 단점이 있다. 따라서 최적화의 기술은 데이터 웨어하우스에서 피할 수 없으며 이를 위한 선택이 인덱스의 활용이다. 인덱스는 테이블 구조와 모든 경로에 대하여 정의 할 수 있기 때문에 다른 최적화의 기법들과 결합 할 수 있다.

따라서, 본 논문에서는 데이터 웨어하우스의 최적화 문제를 해결하기 위하여 인덱스에 대하여 기술하며, 데이터마이닝 기법을 이용한 비트맵 조인 인덱스의 선택에 중점을 두고 기술하고자 한다. 논문의 구성은 1장 서론에 이어, 2장에서 관련연구로 데이터 웨어하우스에서 사용되는 인덱스에 대하여 간략하게 설명하고, 비트맵 조인 인덱스와 데이터 마이닝의 닫힌 빈발 항목집합과 FP-성장 알고리즘을 이용하여 빈발 항목집합을 생성하는 방법에 대하여 기술, 비교 한 후 본 논문에서 제안하는 빈발 항목집합을 생성하기 위한 eFP-Tree를 소개한다. 3장에서는 예시 데이터 세트를 이용하여 조인 질의 실행시 인덱스 선택의 문제점에 대하여 설명하고, 4장에서는 본 논문에서 제안하는 인덱스 후보 빈발 항목집합 생성하는 알고리즘과 제약조건에 따른 비트맵 조인 인덱스의 선택 알고리즘을 소개한다. 이후 5장에서 제안하는 알고리즘과 이전 연구를 성능 평가하여 개선된 결과 부분을 설명한다. 마지막 6장에서 결론과 향후 연구를 기술한다.

2. 관련 연구

2.1 데이터 웨어하우스의 인덱스 기술

데이터 웨어하우스에 대한 인덱스로 단일 테이블 인덱스(single-table index)와 다중 테이블 인덱스(multiple-table index)로 나눌 수 있다. 단일 테이블 인덱스는 하나의 테이블에 한 개 또는 여러 속성들로 정의하는 인덱스이며 다중 테이블 인덱스는 두 개 또는 그 이상의 테이블에 정의한다. 위 두 분류의 인덱스에 속하는 전략으로 데이터 웨어하우스에는 밸류-리스트인덱스(value-list index), 비트맵 인덱스(bitmap index), 프로젝션 인덱스(projection index), 비트 슬라이스 인덱스(bit-sliced index), 조인 인덱스(join index), 비트맵 조인 인덱스(bitmap join index) 등을 제시하고 있다[4-6].

밸류-리스트 인덱스[4]는 균형 트리(valance tree) 구조와 맵핑 스킴(mapping scheme)과 같이 두 부분으로 구성된다. 맵핑 스킴은 트리구조의 잎 노드에 첨부되며 인덱스 된 테이블의 튜플들을 가리킨다. 맵핑 스킴에는 두 종류가 자주 사용되며, 하나는 각 유일한 탐색 키 값에 연관된 rid(row identifier)들로 구성되고, 두 번째는 비트맵 인덱스를 사용한다. 비트맵 인덱스는 각 속성 값(attribute value)에 대하여 하나씩 생성되며, 각 레코드마다 하나의 비트를 할당하여 해당 레코드가 그 속성 값을 갖고 있으면 1을 아니면 0의 비트 값을 저장한다. 비트맵 인덱스는 다른 컬럼(column)에 비해 낮은 카디낼리티(low cardinality)를 갖는 칼럼에 적용하며 Boolean 연산자 (and, or, not)나 count 연산자를 갖는 질의어에 효율적이다[6]. 아래 (그림 1)은 employee 테이블의 성별(gender) 속성에 대한 비트맵 인덱스의 예를 보여주고 있다.

프로젝션 인덱스[4]는 아래 (그림 2)와 같이 테이블의 컬럼과 인덱스된 컬럼과는 동일하다. 즉, 임의의 한 컬럼을 인덱스 한다면 해당 테이블에서 rid의 순서와 일치하는 컬럼(인덱스 컬럼)의 값을 순서대로 저장하여 구성하면 된다. 프로젝션 인덱스는 둘 이상의 컬럼 값 계산을 포함하는 실행 질의에 대해서는 다른 인덱스 구조보다 효과적이라 할 수 있고, 특히 GROUP BY 질의에서 더 잘 실행되는 특징을 가지고 있다.

rid	eid	name	gender	age	dept
1	101	kim	male	25	prod
2	102	lee	male	22	prod
3	103	sohn	female	32	sale
4	104	mun	female	35	sale
5	105	kang	male	41	research
6	106	noh	female	47	research
7	107	choi	male	33	account

employee table

rid	male	female
1	1	0
2	1	0
3	0	1
4	0	1
5	1	0
6	0	1
7	1	0

bitmap index on gender

(그림 1) gender 컬럼에 정의된 비트맵 인덱스

rid	eid	name	gender	age	dept
1	101	kim	male	25	prod
2	102	lee	male	22	prod
3	103	sohn	female	32	sale
4	104	mun	female	35	sale
5	105	kang	male	41	research
6	106	noh	female	47	research
7	107	choi	male	33	account

employee table

age
25
22
32
35
41
47
33

project index on age

(그림 2) age 컬럼에 정의된 프로젝션 인덱스

비트 슬라이스 인덱스[5]는 프로젝션 인덱스 된 컬럼의 키 값을 비트 값들로 변경하고 이러한 비트 값의 조각들로 결합하여 인덱스로 표현한다. 특히 합계나 평균 계산을 하는데 유용하게 활용되며 또한, 클러스터 된 데이터의 백분율 질의나 넓은 범위에 대한 범위 질의는 다른 인덱스 접근보다 더 효과적이다. 아래 (그림 3)은 프로젝션 인덱스 된 employee 테이블의 나이(age) 컬럼에 대한 비트 슬라이스 인덱스의 예를 보여주고 있다.

조인 인덱스[6]는 두 개의 릴레이션들 사이의 조인 연산을 미리 수행한 것이다. 조인 인덱스는 조인 속성들에 의해서 두 개의 테이블들을 조인하고, 그 두 개의 테이블들의 키들을 프로젝트 한 결과가 된다. 조인 인덱스는 조인된 두 개의 테이블들로부터 매칭되는 튜플들을 찾는데 사용된다. 조인 인덱스는 전통적인 데이터베이스에 많이 활용되었으며, 이후 관계형 데이터 웨어하우스의 다중 테이블 조인 인덱스로 확장되었다. 관계형 데이터 웨어하우스의 사실 테이블에 대한 비트맵 조인 인덱스는 사실 테이블과 조인할 차원 테이블의 임의 칼럼에 기초해서 만들어진다.

비트맵 조인 인덱스는 다중 테이블 조인 인덱스이며 데이터 웨어하우스에서 많이 사용되는 비트맵 인덱스는 조인 비용을 줄이기 위하여 응용한 인덱스이며, 데이터 웨어하우스와 같은 갱신이 거의 일어나지 않는 환경에 적합하다. 비트맵 조인 인덱스는 사실 테이블의 각 rid에 대해서 해당되는 차원 테이블의 기본키 값을 비트맵 인덱스 형태로 표현을 한다. 비트맵 조인 인덱스는 이차원 배열의 형태로서 표현이 되며, 이차원 배열의 행은 사실 테이블의 rid 리스트이고

rid	.....	age	dept
1		25	prod
2		22	prod
3		32	sale
4		35	sale
5		41	research
6		47	research
7		33	account

employee table

age
25
22
32
35
41
47
33

bn-1	...	b5	b4	b3	b2	b1	b0
0		0	1	1	0	0	0
0		0	1	0	1	1	0
0		1	0	0	0	0	0
0		1	0	0	0	1	1
0		1	0	1	0	0	1
0		1	0	1	1	1	1
0		1	0	0	0	0	1

bit-sliced index on age

(그림 3) age 컬럼에 정의된 비트 슬라이스 인덱스

orderers

rid	orid	name	gender	address
1	11	fff	male	seoul
2	12	ggg	male	seoul
3	13	iii	female	busan
4	14	jjj	female	busan
5	15	lll	male	daegu

stores

rid	stid	name	address	space
1	1001	aaa	seoul	large
2	1002	bbb	busan	small
3	1003	ccc	daegu	medium
4	1004	ddd	daejon	large
5	1005	eee	suwon	small

items

rid	itid	name	price	prodsite
1	101	mmm	1000	seoul
2	102	nnn	900	seoul
3	103	ooo	1200	busan
4	104	ppp	2300	busan
5	105	qqq	1500	daegu
6	106	sss	2600	daegu
7	107	ttt	3200	seoul

orders

rid	orid	itid	stid	amt
1	11	101	1001	10
2	11	102	1001	20
3	11	104	1002	30
4	12	105	1003	40
5	12	107	1003	50
6	12	103	1004	60
7	13	106	1005	70
8	13	101	1005	80
9	13	102	1002	90
10	14	103	1003	100
11	14	104	1003	110
12	14	105	1004	120
13	15	106	1004	130
14	15	107	1005	140
15	15	101	1001	150
16	11	102	1002	160
17	11	103	1002	170
18	12	104	1003	180
19	12	105	1003	190
20	13	106	1004	200
21	13	107	1004	210
22	14	101	1005	220
23	14	102	1005	230
24	15	103	1001	240
25	15	104	1001	250

Bitmap join index on prodsite

rid	seoul	busan	daegu
1	1	0	0
2	1	0	0
3	0	1	0
4	0	0	1
5	1	0	0
6	0	1	0
7	0	0	1
8	1	0	0
9	1	0	0
10	0	1	0
11	0	1	0
12	0	0	1
13	0	0	1
14	1	0	0
15	1	0	0
16	1	0	0
17	0	1	0
18	0	1	0
19	0	0	1
20	0	0	1
21	1	0	0
22	1	0	0
23	1	0	0
24	0	1	0
25	0	1	0

(그림 4) 비트맵 조인 인덱스

열은 차원 테이블의 기본 속성 값이 된다. 만약, 이차원 배열에서 행과 열이 교차하는 부분이 1로 저장되어 있으며 그 위치에 해당하는 사실 테이블의 한 레코드가 조인된다는 의미이며, 각 행과 열이 각각 사실 테이블과 차원 테이블의 어느 레코드와 대응되는지에 대한 정보이기도 하다. 아래 (그림 4)는 차원 테이블(orderers, stores, items)과 사실 테이블(orders) 간의 비트맵 조인 인덱스의 예를 보여준다.

2.2 비트맵 조인 인덱스의 선택

본 논문에서 관심을 가지는 비트맵 조인 인덱스의 사용의 예를 보이기 위하여 (그림 4)의 관계형 데이터 웨어하우스의 사실 테이블 orders, 차원 테이블 orderers, items, stores에 아래 (그림 5)와 같은 집계함수를 포함하는 OLAP 질의어를 적용하고자 한다.

```
SELECT Count(*)
FROM orders od, items it
WHERE it.prodsite = 'seoul' and it.itid = od.itid;
```

(그림 5) Count 함수를 포함하는 OLAP 질의

위 (그림 5)와 같이 데이터 웨어하우스 환경에서 널리 사용되는 OLAP 질의에서는 조인 연산이 빈번하게 사용된다. 이러한 조인은 차원 테이블의 기본 키와 사실 테이블의 외래 키를 주 대상으로 수행된다. 데이터 웨어하우스 환경에서 사실 테이블과 차원 테이블의 크기가 매우 크므로 조인 처리비용이 매우 크다. 또한, OLAP 질의는 분석적인 질의를 처리하기 위하여 집계함수를 빈번하게 사용된다. 이러한 집계함수는 사실 테이블과 차원 테이블 간의 조인 후에 처리되어지며, 집계 속성은 대부분의 경우 사실 테이블에 있는 속성 값이 된다. 따라서 위 (그림 5)의 질의어를 최적화하기 위해서 items의 prodsite 속성에 대한 사실 테이블 orders와 차원 테이블 items 사이의 oi_idx라는 비트맵 조인 인덱스를 아래 (그림 6)과 같이 생성한다.

```
CREATE BITMAP INDEX oi_idx
ON orders(items.prodsite)
FROM orders od, items it
WHERE od.itid = it.itid;
```

(그림 6) bitmap join index(oi_idx)의 생성

위의 생성 명령어의 실행에 의해서 (그림 5)의 items.prodsite에 대한 비트맵 조인 인덱스가 생성된다. 이 결과 (그림 5)의 질의어를 실행하면 질의어 최적기는 items와 orders를 조인하지 않고, (그림 6)의 비트맵 조인 인덱스를 사용하여 prodsite가 'seoul'을 나타내는 칼럼에 해당하는 비트맵을 접근한다. 따라서, (그림 5)와 같은 질의 실행시 비트맵 조인 인덱스를 사용할 경우 더 효율성이 있다는 것을 보여주고 있다.

상기 예시에서는 간단한 OLAP 질의를 보였다. 하지만, 데이터 웨어하우스에는 매우 많은 데이터가 저장되므로 의

사결정을 위한 복잡하고 분석적인 질의들의 처리를 위하여 많은 데이터에 접근하여야 한다. 따라서 데이터 웨어하우스에서 사용자의 요구를 빠르게 처리할 수 있어야 한다는 중요한 의미를 가지고 있다. 특히, 데이터 웨어하우스 환경에서 의사결정시 빈번하게 사용되는 집계함수 및 조인은 기본적인 관계 연산인 선택(select), 프로젝트(project) 연산과는 달리 질의 비용이 크므로 이를 위한 인덱스 선택의 연구는 매우 중요하며 데이터 웨어하우스와 자원 제약(유지보수, 용량) 환경 하에서 효과적인 질의 처리를 위하여 인덱스 집합을 자동적으로 선택하는 것 또한 중요한 이슈가 되고 있다[7]. 관계형 데이터 웨어하우스의 크기는 실 시스템에서 여러 개의 칼럼을 갖는 많은 테이블로 구성되어 아주 방대하며 인덱스는 서로 다른 테이블의 칼럼으로부터 정의된다. 기존 인덱스 연구는 단일 테이블 인덱스 선택에 대한 것이 대부분이며 다중 테이블 인덱스 선택 문제에 대한 몇 안 되는 연구가 있다[8].

본 연구에서는 다중 테이블 인덱스 선택에 관심을 가졌으며, 대부분 연구는 다음과 같은 핵심 두 단계를 이용하고 있다. 첫 번째 단계는 후보 속성들의 생성이고, 두 번째 단계에서 그리디 알고리즘 및 선형 프로그래밍 알고리즘과 같은 휴리스틱 알고리즘을 사용하여 최종 인덱스 구성을 선택하는 것이다. 최종 선택되어진 인덱스 집합의 품질은 인덱스 후보 속성들에서 관련이 적은 컬럼들을 제거하기 위한 가지치기(pruning) 단계에 의존하고 있기 때문에 인덱스 후보 탐색 공간의 가지치기를 위한 알고리즘으로 SQL Server에서는 주어진 인덱스 구성의 우수함을 결정하기 위하여 최적화 비용 추산기를 사용하는 그리디 알고리즘을 제공하고 있다[9]. 하지만, 상기 기존 연구들의 약점은 인덱스 선택 작업과정에서 후보 속성의 항목 수가 많이 생성된다는 것이다. 최근에는 이를 개선하기 위한 목적으로 닫힌 빈발항목집합을 이용하고 있지만 마이닝 과정에서의 가지치기(불필요 항목 제거작업)의 추가 작업과 과비용적인 문제점을 가지고 있다. 이를 개선하기 위하여 본 논문에서는 비트맵 조인 인덱스로 선택될 후보 빈발 항목집합의 생성과 후보 항목수를 줄이기 위한 FP-성장(FP-Growth) 알고리즘기반의 확장 빈발 패턴트리(extended Frequent Pattern Tree, eFP-Tree)를 이용하여 데이터마이닝 하는 방법을 제안한다.

2.3 비트맵 인덱스 선택을 위한 데이터마이닝 방법

집합 $I=\{i_1, i_2 \cdots i_m\}$는 서로 다른 m개 항목들의 집합(set)이다. 항목은 질의에 사용되는 테이블의 컬럼(column)이며, 속성(attribute)을 의미한다. 항목집합(itemset)인 X는 집합 I의 부분집합(subset)인 항목들의 모임이며, $X \subseteq I$를 의미한다. |X|의 크기(항목개수)를 K개라면 K-항목집합이라 한다. 예를 들어 $X=\{i_1, i_2, i5\}$라면 항목의 개수가 3이므로 3-항목집합이 된다. 트랜잭션 데이터베이스(Transaction Database) $TDB=\{T_1, T_2, \cdots T_n\}$는 트랜잭션들의 집합이며, 각 트랜잭션 Ti는 집합 I로부터 선택된 항목들의 부분집합을 포함하고 있다. 항목집합 X의 지지도(support) 또는 발생빈도(occurrences frequency)는 TDB에서 항목집합 X를 포함하고 있는 트랜잭션의 개수(supp(X))를 말한다. 빈발 패턴 마이닝은 TDB에 나타난 여러 항목집합 X의 지지도가 주어진 최소 지지도 임계값(m)보다 크거나 같은 패턴을 빈발($supp(X) \geq m$)이라 하며, 이러한 항목집합을 빈발 항목집합이라 한다. 데이터 마이닝에서 자주 사용하는 방법 중의 하나는 두 단계처리 과정을 가지고 있는 Apriori 알고리즘의 규칙 마이닝이다[10].

Apriori 알고리즘은 n번째 항목집합이 n+1번째 항목집합을 생성하기 위한 레벨단위로 진행되는 반복 접근법을 사용한다. Apriori 알고리즘은 k-항목집합이 없을 때까지 진행을 하게 되며 이러한 과정에서 불필요한 후보 패턴을 많이 생성할 뿐만 아니라 생성된 후보 항목집합들 중에서 빈발 항목집합을 찾아내기 위해 매번 계속적으로 대량의 트랜잭션을 스캔해야 하므로 속도가 느려 잘 활용되지 못하였으며 정확도 및 확장성에 대한 문제도 제기되었다. 이러한 후보 속성의 크기를 줄이기 위해서 닫힌 빈발 항목집합(Closed Frequent Itemset)과 최대 빈발 항목집합(Maximal Frequent Itemsets)을 이용한 가지치기와 FP-Tree를 이용한 FP-성장 연구가 진행되었다[11,12].

닫힌 빈발 항목집합은 항목집합들의 지지도 정보의 손실 없이 최소의 표현을 규정하고 있다. 닫힌 항목집합이란 항목집합 X의 포함집합들이 어느 것도 X와 정확히 같은 지지도 카운트를 가지지 않는 경우 항목집합 X는 닫혀있다고 정의하고 있다[13]. 만약 항목집합이 닫혀있고, 그 지지도가 최소지지도(m)보다 크거나 같으면 그 항목집합은 빈발 닫힌 항목집합이라 한다. 빈발 닫힌 항목집합들은 중복되는 연관규칙들의 일부를 제거하는 데 유익하다. 만약 연관규칙 $X \rightarrow Y$이 존재하고, X가 X′의 부분집합이고 Y가 Y′의 부분집합인 다른 연관 규칙 $X' \rightarrow Y'$이 존재하면서 양쪽 규칙들에 대한 지지도와 신뢰도가 동일하다면, 연관 규칙 $X \rightarrow Y$는 중복된다. 이러한 경우 만약 빈발 닫힌 항목집합들이 규칙 생성에 사용되면 그런 중복되는 규칙들은 생성되지 않는다. 더불어 최대 빈발 항목집합들 중 어떤 것도 그들의 인접한 포함집합과 같은 동일한 지지도 카운트를 가질 수 없기 때문에 모든 최대 빈발 항목집합들은 닫혀있다는 것이다. 이와 같은 방법들은 빈발 항목집합 전체를 마이닝한 후 빈발 항목집합과 동일한 지지도를 갖는 부분집합을 모두 찾아 제거해야 하는 작업과정의 과비용과 연관된 항목 손실의 단점을 보이고 있다.

반면에 트랜잭션 데이터베이스를 한 번만 스캔하여 빈발 패턴을 찾아내는 FP-성장 알고리즘은 특정 항목에 대하여 FP-Tree를 상향식 방법으로 탐색하여 빈발 항목집합들을 생성하는 방법이다. 본 논문에서는 FP-성장 기반의 확장 빈발 패턴 트리(extended Frequent Pattern Tree, eFP-Tree)를 이용하여 빈발 후보 항목집합을 생성하는 방법을 제안한다. eFP-Tree는 다음과 같이 기술되고 구성되어 진다. 아래 <표 1> TDB는 집합 I={a, b, c, d, e, f, g, h}와 같이 8개의 항목들과 총 9개의 트랜잭션(질의)들로 구성되어 있다. 빈발 항목은 최소 지지도 임계값(m)을 만족하는 항목($supp(X) \geq 3$)들이다. 정렬된 빈발 항목집합은 내림차순 정렬된 항목들이다.

〈표 1〉 예시 트랜잭션 데이터베이스(TDB)

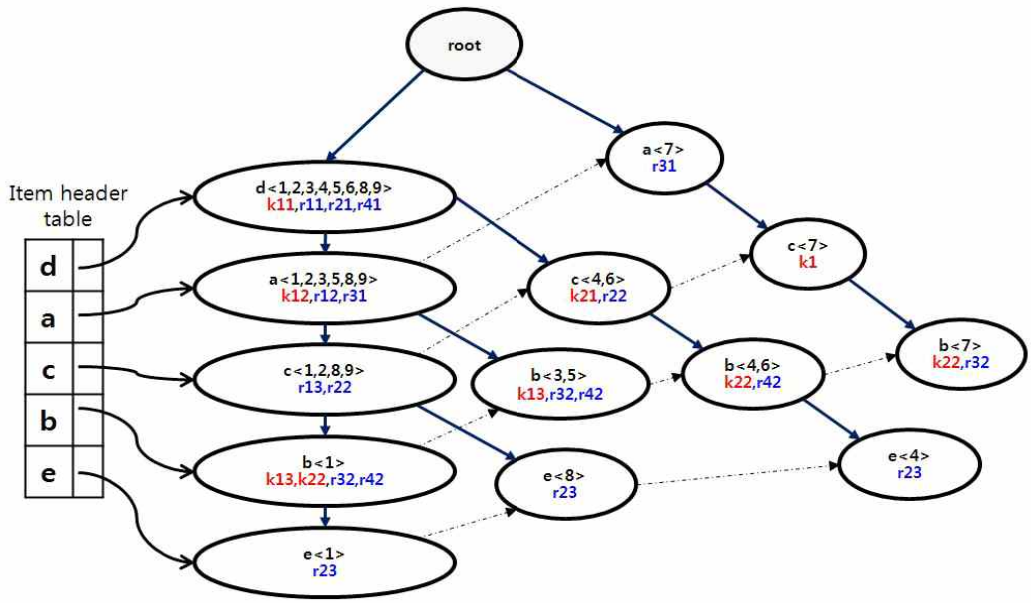
TID (query)	items (attribute)	frequent items	ordered frequent items
1	a b c d e	a b c d e	d a c b e
2	a c d	a c d	d a c
3	a b d f g	a b d	d a b
4	b c d e	b c d e	d c b e
5	a b d	a b d	d a b
6	b c d f h	b c d	d c b
7	a b c g	a b c	a c b
8	a c d e	a c d e	d a c e
9	a c d h	a c d	d a c

eFP-Tree는 FP-Tree를 생성하는 과정과 동일하다. 그러나 트리를 구성하고 있는 노드는 FP-Tree와 차이를 보인다. eFP-Tree의 노드 구성은 항목명과 <T_i, K_i, R_i>로 표현된다. 여기서 T_i는 해당 항목을 포함하고 있는 트랜잭션 리스트 즉, 질의 리스트(T_1, T_2,…T_n)를 의미하며 eFP-Tree 생성과정 중 빈발 항목 노드에 기록된다. K_i는 사실테이블과 차원테이블의 외래키와 주키 속성의 성질을 가지는 항목이며 비트맵 조인 인덱스 선택에서는 제거될 항목이 된다. R_i는 K_i와 연관 관계를 가지는 항목을 의미하며 제거될 항목으로 평가되어야 한다. eFP-Tree의 예시는 (그림 7)과 같으며, 지지도 3을 기준으로 <표 1>의 예시 TDB로부터 생성되었다. eFP-Tree 노드는 사용된 질의 정보, 키 속성 정보, 키와 연관된 속성 정보들로 구성하고 있기 때문에 비트맵 조인 인덱스의 후보 속성을 생성하는데 있어 트랜잭션 데이터베이스를 반복 스캔할 필요가 없다는 장점을 가지고 있다.

이전 연구로 CLOSE 알고리즘[13]은 닫힌 빈발 항목집합을 생성하는 방법 중의 하나이며, Apriori 알고리즘과 같이 항목집합 생성을 위하여 레벨 단위의 반복 접근법을 사용한다. CLOSET와 CLOSET+ 알고리즘[14]은 FP-Tree를 이용한 상향식 트리 검색기반의 FP-성장 알고리즘을 사용한다. 비록 이전 알고리즘이 데이터 마이닝 성능을 강화하기 위한 최적화된 기술이지만 빈약한 데이터 세트와 낮은 지지도 임계값일 때는 여전히 빈발 항목집합 생성작업에 어려움을 보였다.

3. 기존 연구의 비트맵 조인 인덱스 선택 문제점

비트맵 조인 인덱스는 데이터 웨어하우스에서 사실 테이블과 여러 개의 차원 테이블들 간의 조인을 위해서 낮은 카디널리티를 가지면서 하나 또는 여러 개의 키가 아닌 차원 테이블의 속성들에 대해서 정의된다. 차원 테이블 D의 비트맵 조인 인덱스 할 수 있는 속성 A라면 인덱스를 D.A로 나타낼 수 있으며, WHERE 조건절에 'D.A Θ 값'의 식이 있다. 여기서 Θ는 비교연산자({=, <, >, <=, >=})를 말한다. A={A_1, A_2 .... A_k}에서 A를 비트맵 조인 인덱스를 위한 인덱스된 후보 속성들의 집합이라면, 한 그룹의 속성들을 갖는 비트맵 조인 인덱스의 가능한 개수는 2^k-1이 된다. 예로 항목수 k=4일 경우 15개의 인덱스 가능수가 구해진다. 따라서 비트맵 조인 인덱스 가능한 수는 k에 의해 기하급수적으로 증가하는 것을 알 수가 있다. 후보 속성 집합에서 임의의 부분 항목집합들의 비트맵 조인 인덱스의 개수는 2^{2^k-1}이 된다. 예로 항목수 k=4일 경우 2^{15}=32768개가 된다. 이 많은 수의 결과 후보 속성 집합에서 비트맵 조인 인덱스 가능한 모든 항목들을 나열하여 평가할 수 없으며 각 후보의 비트맵 조인 인덱스 질의 비용을 전체적으로 계산한다는 것은 데이터 웨어하우스 환경에서는 불가능한 일이다. 따라서 저장 영역의 제약조건을 만족하면서 전체 질의 처리 비용을 최소화하는 인덱스 가능한 테이블의 속성을 찾아서 선택하는 것이 비트맵 조인 인덱스 집합을 찾는 문제이다.



(그림 7) 예시 TDB의 eFP-Tree

비트맵 조인 인덱스 선택 문제는 다음과 같이 정의할 수 있다. 데이터 웨어하우스에서 차원 테이블의 집합 D={D_1, D_2 .... D_n}과 사실 테이블 F, 그리고 질의어들의 집합 Q={Q_1, Q_2 .... Q_m}, 저장 용량 제약 사항이 S일 때 비트맵 조인 인덱스 선택 문제의 목적은 모든 가능한 후보 속성들의 미리 계산된 부분 집합 중에서 저장 용량 제약 사항을 만족하면서 질의어 처리 비용을 최소화하는 비트맵 조인 인덱스의 집합을 찾는 것이다. 이전 연구에서는 직관적으로 비트맵 조인 인덱스를 선택하는 것은 인덱스 가능한 속성들의 집합 A를 disjoint 그룹으로 분할하는 것이기 때문에 모든 가능한 속성들의 조합들을 계산하지 않고 빈발 속성 그룹들의 항목집합을 계산함으로써 데이터 마이닝에 기초한 방법을 사용하였다. 그러나 이러한 방법들도 다음과 같은 두 가지의 주요 제한점으로 정규화가 제안되지 않았고, 인덱스할 속성 집합을 나타내는 빈도수만 고려하였다. 추가적으로 본 논문에서는 좋은 인덱스 선택 결과를 얻기 위해서 매개변수 값으로 빈도수만 고려하지 않고 테이블 크기, 각 튜플의 크기, 디스크 페이지의 크기 등을 사용하였다.

비트맵 조인 인덱스의 탐색 공간을 축소하는 [8]의 연구와 비교한 후 문제점을 보이기 위하여 두 개의 차원 테이블 CHANNELS와 CUSTOMERS과 하나의 사실 테이블 SALES로 구성된 스타 스키마의 일부분을 예시로 한다. 각 테이블의 카디널리티의 개수는 |SALES|=16260336, |CHANNELS|=5, |CUSTOMERS|=50000이고 아주 빈번하게 실행되는 최종 다섯 개의 질의어는 아래 (그림 8)과 같다.

아래 (그림 8)의 질의어에서는 다음과 같은 핵심이 되는 중요한 두 개의 조인 연산을 확인 할 수 있으며 그 비용을 아래와 같이 구할 수 있다.

J_1 : SALES ⋈ CUSTOMERS cost of J_1 = 16260336 x 50000

J_2 : SALES ⋈ CHANNELS cost of J_2 = 16260336 x 5

단순하게 위 J_1, J_2에서 차원 테이블의 카디널리티 수(CUSTOMERS : 50000 instance, CHANNELS : 5 instance)만을 비교하더라도 J_1이 J_2보다 높다는 것을 알 수 있다. 다음으로 빈발 횟수의 최소지지도 임계치를 minsup라 하고, 해당 값으로 3이 주어졌을 때 빈발 횟수를 기반으로 하는 [8]의 연구 해결책으로는 channel_desc 속성을 사용하여 CHANNELS와 SALES에 비트맵 조인 인덱스를 정의하는 것으로 기술하였다. 여기서 minsup의 경우는 질의어에 사용된 속성의 빈발 기준을 의미한다. 또 다른 속성인 cust_gender의 경우 빈발 횟수가 2이다. 그러므로 minsup의 3의 값에 미치지 못하여 비 빈발하기 때문에 SALES와 CUSTOMERS 사이의 비트맵 조인 인덱스로는 제안되지 않으며 그 결과로 J_2만이 최적화 될 것이다. 위 방법과 같이 해당 질의에서 빈발 속성만을 대상으로 한다는 것은 시스템의 전반적인 제약 사항을 전혀 고려치 않아 성능 저하 원인이 되기도 한다. 문제점을 해결하기 위해서 본 논문에서는 최종 인덱스 선택에 있어 테이블의 크기, 튜플의 크기, 디스크 페이지 크기 등의 매개변수를 고려하며, 비트맵 조인 인덱스 선택에 있어 탐색 공간과 인덱스 선택을 위한 후보 속성의 수를 최소화 하는데 목적을 두고 있다.

4. eFP-Tree를 이용한 비트맵 조인 인덱스의 선택

본 논문에서는 전체적인 질의 처리 비용을 최소화하고 스토리지 제약조건을 만족하는 비트맵 조인 인덱스를 선택하기 위하여, 기존의 인덱스 선택 접근 방식[8]에서 일반적으로 기술하고 있는 방법을 참고하여 다음과 같은 단계를 고려한다. 첫 번째, eFP-Tree를 이용하여 인덱스 가능한 후보 속성을 생성하며, 두 번째, 인덱스의 최종 구성과 비트맵 조인 인덱스를 선택한다. 추가적으로 인덱스 선택에 있어 각 테이블의 크기, 튜플의 크기, 디스크 페이지의 크기 등을 매개변수로 고려한다. 따라서, 알고리즘의 접근을 돕기 위하여

```
(1) Select S.channel_id, sum(S.quantity_sold) from SALES S, CHANNELS Ch
    Where S.channel_id = Ch.channel_id and Ch.channel_desc = 'Internet'
    Group by S.channel_id
(2) Select S.channel_id, sum(S.quantity_sold) from SALES S, CHANNELS Ch
    Where S.channel_id = Ch.channel_id and Ch.channel_desc = 'Catalog'
    Group by S.channel_id
(3) Select S.channel_id, sum(S.quantity_sold) from SALES S, CHANNELS Ch
    Where S.channel_id = Ch.channel_id and Ch.channel_desc = 'Partners'
    Group by S.channel_id
(4) Select S.cust_id, avg(S.quantity_sold) from SALES S, CUSTOMER C
    Where S.cust_id = C.cust_id and C.cust_gender = 'M'
    Group by S.cust_id
(5) Select S.cust_id, avg(S.quantity_sold) from SALES S, CUSTOMER C
    Where S.cust_id = C.cust_id and C.cust_gender = 'F'
    Group by S.cust_id
```

(그림 8) 빈번하게 실행하는 선택된 최종 질의어

차원 테이블들의 집합을 {D_1, D_2 .... D_n}, 사실 테이블을 F, ||T||를 테이블 T의 카디널리티 개수(T : fact table, dimension table), LT를 테이블 T의 튜플 길이, |T|를 테이블 T가 필요로 하는 디스크 페이지 수라 한다면 |T|=(||T|| X LT) / P로 나타낼 수 있으며, 여기서 P는 디스크 페이지 크기를 말한다.

4.1 빈발한 후보 항목집합 생성

비트맵 조인 인덱스는 사실 테이블과 차원 테이블 사이에 키가 아닌 차원 테이블의 속성에 기초해서 만들어진다. 닫힌 빈발 항목집합이 차원 테이블의 키 속성 또는 사실 테이블의 외래키 만을 포함하고 있는 경우는 모든 항목들을 제거하고, 그 다음 키가 아닌 속성들 보다 키 속성들을 많이 갖고 있는 경우는 키가 아닌 속성들로만 후보 속성으로 구성한다. 이와 같이 닫힌 빈발 항목집합들은 잘못된 비트맵 조인 인덱스가 만들어지지 않도록 후보 생성 단계에서 인덱스 비후보 속성의 크기를 축소한다. 이러한 키 속성과 키가 아닌 속성을 구별하는 또 다른 제거 작업과정이 필요하며 선택된 빈발 항목집합들 중에서 키 속성을 가진 항목집합만을 제거한다는 것은 최종적인 인덱스 후보 항목집합의 결과라 볼 수가 없다. Apriori 원리[15]에 의하면 만약에 항목집합이 빈발하면 그것은 모든 부분집합들 역시 빈발해야만 한다. 즉, 항목집합이 키 속성을 가지고 있다면 그것을 포함하는 모든 항목집합들은 키 속성을 가진 항목집합과 밀접한 관계를 가지고 있기 때문에 후보 속성에서 제거되어야 한다. 따라서, 해당 절에서는 본 논문에서 제안하는 FP-성장 기반의 eFP-Tree를 이용하여 빈발 항목집합들을 생성하고 이중에서 키 속성과 연관된 항목집합을 제거하는 방법을 기술하고자 한다.

앞 <표 1> 예시 TDB에서 빈발 항목집합의 포함관계에 따른 격자(lattice) 구성은 아래 (그림 9)와 같으며, 굵은선 타원의 항목집합 {dab, dcb, cb}는 키 속성을 가진 항목집합(key attribute itemsets, KAi)이다.

위 (그림 9)에서 KAi와 완전 포함관계에 있는 항목집합은 {db, ab}이며 점선의 타원으로 표시되고 후보에서 제거되는 항목집합이다. 키가 아닌 항목집합(non key attribute itemsets, nKAi) {da, dc, ac, ce, de}는 키 속성과 완전 포함관계에 있지 않는 항목집합이다. nKAi는 KAi 항목집합과 연관 관계를 평가하여 후보 항목집합에서 제거할 대상이 된다. 항목집합들의 연관 관계를 평가하기 위하여 경계(border)를 이용하였다[15]. KAi에서 {dcb}는 <표 2>와 같이 {cb}와 서로 포함관계에 있기 때문에 평가 대상에서 제외되고 키 속성 중에 평가 대상이 되는 항목집합 KAi={dab, cb}가 된다.

〈표 2〉 키 속성 항목집합의 트랜잭션 리스트

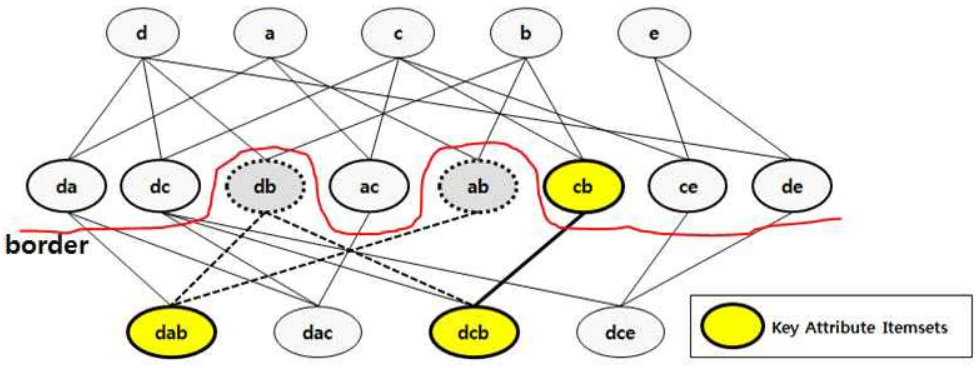
item	frequent	TIDS
dab	3	T_1, T_3, T_5
dcb	3	T_1, T_4, T_6
cb	4	T_1, T_4, T_6, T_7
⇓		
dab	3	T_1, T_3, T_5
cb	4	T_1, T_4, T_6, T_7

〈표 3〉 KAi와 nKAi의 트랜잭션 비트 패턴 테이블

k_{ij}	KAi	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	supp(k)
k_1	dab	1		1		1					3
k_2	cb	1			1		1	1			4

r_{ij}	nKAi	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	supp(r)
r_1	da	1	1	1		1			1	1	6
r_2	dc	1	1		1		1		1	1	6
r_3	ac	1	1					1	1	1	5
r_4	ce	1			1				1		3
r_5	de	1			1				1		3

상기에서 생성된 KAi와 nKAi 항목집합을 포함하고 있는 트랜잭션을 비트 패턴으로 표현한 테이블은 <표 3>과 같으며, k_{ij}는 KAi의 항목이며 키 속성을 가지는 항목집합을 의미한다. 여기서 i는 KAi의 요소인 k_i의 순번이며, j는 k_i가 포함하고 있는 항목 순번을 의미한다. 예를 들어 k_{12} 경우는 KAi={dab, cb}의 첫 번째 요소 k_1(abd)에서 두 번째 항목을



(그림 9) 빈발 항목집합 격자

의미하고 결과 k_{12}={b}가 된다. r_{ij}는 nKAi의 항목집합이며, i는 nKAi의 r_i의 순번이며, j는 해당 r_i가 포함하고 있는 항목 순번을 의미한다. 예를 들어 r_{22}의 경우는 nKAi={da, dc, ac, ce, de}의 두 번째 요소 k_2(dc)에서 두 번째 항목을 의미하고 결과 k_{22}={c}가 된다.

키 속성 항목과 연관 관계에 있는 항목집합의 제거를 위하여 KAi={dab, cb}에 대한 평가 impact 값을 구하고 그 결과 중에서 최소 impact 값을 찾아서 해당 항목을 제거한다[16]. 여기서 impact 값은 KAi를 포함하고 있는 nKAi의 지지도를 적용한 가중치(weight)를 구한 후 항목들의 가중치 전체 합으로 구할 수 있으며, 가중치는 KAi와 nKAi의 상대적 지지도를 유지하기 위함이다. KAi의 제거 후보항목을 k_i라 하고, $w(r_i)$는 각 $r_i \in nKAi|_k$의 가중치라 한다면 impact는 $I(T_i, k_{ij}) = \sum w(r_i)$와 같이 계산할 수 있다. 여기서 T_i는 키 속성 항목 k_{ij}를 포함하고 있는 트랜잭션을 말한다. 가중치는 제거 작업동안 변경되는 지지도에 의해서 다시 계산하여 반영된다. 가중치 합에 의한 impact 값이 작을수록 nKAi 손실이 최소화되고 제거 될 확률이 높은 KAi가 되는 것이다. [16]의 연구에서 impact 값을 구하기 위한 nKAi 항목집합 r의 가중치(w(r))는 아래 조건에 의해서 구해진다.

```
if Psupp(r)≥(m+1) then  // m(최소 지지도 임계값)
   w(r)=supp(r)-Psupp(r)+1 / (supp(r)-m);
if 0≤Psupp(r)≤m then
   w(r)=n+m-Psupp(r);  // 0≤n≤∞(n : 임의의 큰 정수)
```

여기서 제시된 TDB에서 nKAi의 항목집합 r의 지지도는 supp(r)이고, 수정 처리 중인 TDB를 PTDB이라하고, Psupp(r)는 PTDB의 지지도이다. impact 값에 의한 후보 속성 항목집합 생성을 위한 알고리즘은 아래 (그림 10)과 같다.

위 <표 3>의 KAi, nKAi 비트 패턴 테이블과 (그림 10)의 알고리즘을 이용하여 제거 항목으로 KAi의 첫 번째 경계 항목집합인 {dab}를 실례로 적용고자 한다.

① 키 속성 k_i의 항목 k_i(dab)을 포함하는 트랜잭션 리스트를 구한다. 결과는 T_i, T_3, T_5 이다.

② 각 T_i별 r_i 리스트를 구한다.

T_1 : r_1, r_2, r_3, r_4, r_5, r_6 T_3 : r_1 T_5 : r_1

```
Algorithm for Candidate Attribute Itemsets(eFP-Tree)
Input :
   TDB : Transaction Database;
   m : Minimal Support Threshold;
   KAi : Key Attribute Itemsets;
Output :
   CA : Candidate Attributes;
Method :
   Compute FIS and FIS transaction list;
   Compute KAi, nKAi transaction list;
   Compute border itemset of KAi, nKAi;
   for each Xi∈KAi do
      Compute nKAi|k  and w(rj) where rj∈nKAi|k;
      Initialize C(C is the set of removing candidate of X);
      until(supp(r)<m) do
         Find ui=(Ti, xi) such that I(ui)=Min {I(u)|u∈C};
         Update C=C- {(T,x)|T=Ti};
         Update w(rj) where rj∈nKAi|k;
      end
   end
   Update database TDB;
   Output CA;
```

(그림 10) 후보 속성 항목집합 생성 알고리즘

③ 각 r_i들의 가중치 $w(r_i)$를 계산한 후, s_i의 각 항목별 impact $I(T_i, k_{ij})$를 계산한다.

i) 가중치 $w(r_i)$를 구한다.

- W(r1::(da:6))=(6-6+1)/(6-3)=1/3
- W(r2::(dc:6))=(6-6+1)/(6-3)=1/3
- W(r3::(ac:5))=(5-5+1)/(5-3)=1/2
- W(r4::(ce:3))=n+3-3=n
- W(r5::(de:3))=n+3-3=n

ii) 가중치 $w(r_i)$ 결과를 참고하여 impact $I(T_i, k_{ij})$를 구한다.

- I(T1, d)==w(da)+w(dc)+w(de)=1/3+1/3+n
- I(T1, a)==w(da)+w(ac)=1/3+1/2
- I(T1, b)== null
- I(T3, d)==w(da)=1/3
- I(T3, a)==w(da)=1/3
- I(T3, b)== null
- I(T5, d)==w(da)=1/3
- I(T5, a)==w(da)=1/3
- I(T5, b)== null

제거 항목

kij	KAi	T1	T2	T3	T4	T5	T6	T7	T8	T9	supp(k)
k1	dab	1		1		1					3
k2	cb	1			1		1	1			4

rij	nKAi	T1	T2	T3	T4	T5	T6	T7	T8	T9	supp(r)	Psupp(r)
r1	da	1	1	1		1			1	1	6	5
r2	dc	1	1		1		1		1	1	6	5
r3	ac	1	1					1	1	1	5	4
r4	ce	1			1				1		3	
r5	de	1			1				1		3	

제거 평가 항목

(그림 11) 항목들의 연관 관계 평가 후 반영된 결과

④ ③의 impact I(T_i, k_{ij}) 결과 중에서 가장 작은 I(T_3, d), I(T_3, a), I(T_5, d), I(T_5, a)중 하나를 선택한다.
⑤ Impact 값이 동일한 경우는 우선 트랜잭션 크기가 큰 것을 선택하여 삭제한다. 트랜잭 션 크기까지 같을 경우 해당 k_{ij}의 지지도가 큰 것을, 둘 다 같은 경우는 트랜잭션 순서대로 선택한다. 현재 결과에서는 트랜잭션의 크기가 크면서 민감한 항목 k_{ij}의 지지도가 큰 I(T_3, d)를 삭제를 수행한다.
⑥ T_3에서 d를 삭제한 후 가중치 w(r_i)를 수정하고, 트랜잭션 데이터베이스를 수정한다.
⑦ KAi의 지지도가 임계값 아래가 될 때까지 ①부터 반복 수행한다.

4.2 처리비용 제약조건에 따른 인덱스 후보 선택

최종 비트맵 조인 인덱스를 선택하기 위한 생성된 후보 속성(CA)들은 상황 행렬(context matrix)을 사용한다. 상황 행렬은 질의어들의 집합 Q={Q_1, Q_2 ....Q_m}과 인덱스 가능한 속성들의 집합 A={A_1, A_2 .... A_n}으로부터 구성한다. 상황 행렬에서 행은 질의어들을 나타내고 칼럼은 속성들을 나타낸다. 질의에 사용된 속성에 비트값 1을 표현하고 그 외는 0의 값으로 표현된다. 앞 절에서 제시한 데이터 세트를 이용하며 각각의 테이블 카디널리티로 ||CUSTOMERS|| = 500000, ||SALES|| = 1626033, ||CHANNELS|| = 5, 각 테이블 튜플의 크기로 LCUSTOMERS = 24, LSALES = 36, LCHANNELS = 24, 디스크의 페이지 수로 P = 65536이다. 아래 <표 4>의 상황 행렬 구성을 쉽게 하기 위하여 (그림 8)의 질의어에서 사용한 속성들을 다음과 같이 재명명한다. SALES.cust_id:A_1, CUSTOMERS.cust_id:A_2, CUSTOMERS.cust_gender:A_3, CHANNELS.channel_id:A_4, SALES.channel_id:A_5, CHANNELS.channel_desc:A_6. A_1, A_2, A_4, A_5에서 정의된 속성은 해당 테이블의 키 속성들이며 A_3, A_6는 각 테이블에서 키가 아닌 속성들이다.

〈표 4〉 질의에 따른 상황 행렬

속성 / 질의	A1	A2	A3	A4	A5	A6
Q1	0	0	0	1	1	1
Q2	0	0	0	1	1	1
Q3	0	0	0	1	1	1
Q4	1	1	1	0	0	0
Q5	1	1	1	0	0	0
support	2	2	2	3	3	3

본 논문에서는 비트맵 조인 인덱스의 선택 문제인 탐색 공간을 줄이기 위하여 [8,13,17]의 CLOSE 알고리즘을 개선하여 sBJI(selecting Bitmap Join Index) 알고리즘을 제안한다. 비트맵 조인 인덱스의 선택 문제의 탐색 공간을 줄이기 위해서 측정 함수인 nfit(X)를 아래와 같이 정의한다.

$$nfit(X)=(supp_1X||D_1|| + supp_2X||D_2|| ..... + supp_mX||D_m||) / m$$

여기서, X는 후보 속성인 빈발 항목집합이며, m은 X에서 키가 아닌 속성들의 개수, $||D_i||$는 키가 아닌 속성 A_i를 갖는 차원 테이블의 튜플의 개수, suppi는 속성 A_i의 지지도이다. minsup가 주어졌을 때, nminfit(minimal fitness number)는 아래와 같이 정의 계산한다.

$$nminfit = minsup\ X\ (||D_i||+||D_2|| .... ||D_m||) / m$$

앞의 <표 4>의 상황 행렬로부터 두 개의 C_1={A_1, A_2, A_3}, C_2={A_4, A_5, A_6}와 같은 disjoint 항목집합이 생성된다. minsup가 3인 경우 단순히 [8]의 논문 방법으로는 질의에 사용하는 속성의 빈발 정도를 매개변수로 하기 때문에 C_2가 선택이 되어서 A_6(channel_desc)에 대한 비트맵 조인 인덱스가 생성되며, CHANNELS 테이블의 channel_desc 값에 따라서 CHANNELS 테이블의 레코드들과 SALES 테이블의 레코드들이 조인되는 것이다.

본 논문에서는 disjoint 항목집합 중에서 키가 아닌 속성에 대해서 테이블의 크기, 튜플의 크기, 디스크 페이지 크기 등의 매개변수를 고려하여 적합성 값 nfit(X)를 계산하고, 주어진 minsup에 대한 값을 계산한 후 빈발 및 기타 제약사항의 적합성을 판별한다. 위 빈발 항목집합(C_1, C_2)에서 적합성 값 nfit(X)의 값을 아래와 같이 구할 수 있다.

C_1에서 키가 아닌 속성은 A_3 하나이기 때문에
$nfit(C_1) = (supp(A_3)\ X\ ||D_3||) / 1 = 2\ X\ 50000 = 100000$
C_2에서 키가 아닌 속성은 A_6 하나이기 때문에
$nfit(C_2) = (supp(A_6)\ X\ ||D_6||) / 1 = 3\ X\ 5 = 15$

그리고, 최소 지지도 minsup가 3일 경우 nminfit는
$nminfit = minsup\ X\ (||D_1|| + ||D_2|| .... + ||D_m||) / m$
$= 3\ X\ (||CUSTOMERS|| + ||CHANNELS||) / 2$
$= 3\ X\ (50000 + 5) / 2 = 75007.5$

본 논문에서는 빈발 항목 및 기타 제약 사항의 매개변수를 적용한 결과로 nminfit보다 상회하는 결과를 얻었기 때문에 닫힌 빈발 항목집합 C1의 A3을 비트맵 조인 인덱스 속성으로 선택하여 최적화를 한다.

4.3 제약조건에 따른 비트맵 조인 인덱스 선택 알고리즘(sBJI)

비트맵 조인 인덱스의 최종 구성을 선택하기 위한 절차는 다음과 같다. 4.1절에서 비트맵 조인 인덱스의 최종 구성에 필요한 후보 속성 CA를 만들었다. 최종 구성을 위한 sBJI 알고리즘에 대한 입력은 (1) 차원 테이블 집합과 사실 테이블로 구성된 스타 스키마, (2) 질의어들의 집합, Q={Q_1, Q_2 .... Q_n}, (3) CA : 후보 속성들의 집합, (4) S : 저장 용량 제약 사항 등이다. sBJI 알고리즘은 CA의 한 속성에 대해서 정의된 비트맵 인덱스를 가지고 시작한다. CA에 속한 속성들의 카디널리티 수가 많기 때문에 가장 적은 카디널리티

수를 가지는 속성을 첫 번째 속성으로 선택하고 그것을 A_{min}라 한다. CA의 속성들을 그들의 각 속성 값들의 개수에 의해서 오름차순으로 정렬한다. 그리고 알고리즘은 저장 용량 제약 사항을 만족하고 질의어 처리 비용의 감소가 가능한 동안 CA의 다른 속성을 선택해서 초기의 속성 구성 사항을 반복적으로 진행시킨다. 속성 구성 사항의 효율성을 측정하기 위하여 일련의 질의들을 실행할 때 디스크 페이지 접근 횟수(즉, 입출력 비용)을 계산하는 새로운 비용 모델(new COST, nCOST)을 사용했다. 또한 비용 모델은 비트맵 조인 인덱스에 필요한 저장 용량도 계산한다. 따라서 nCOST 함수의 값은 질의 비용(query cost)와 저장 용량 비용(storage cost) 값들의 결합이다. 아래 (그림 12)은 비트맵 조인 인덱스 선택을 위한 sBJI 알고리즘의 주요 부분을 기술하고 있다.

Algorithm for Bitmap Join Index Selection(**sBJI**)

Input :
SCA : Sort Candidate Attribute /* ascending sort */
Q : Query(TDB)
S : Storage
Output :
outBJI : selected Bitmap Join Index
Remark :
BJI_n : a bitmap join index defined on attribute A_n
Size(BJI_n) : storage cost required for BJI_n
begin
SCA = SORT(call eFP-Tree);
outConfig = BJI_{min};
S := S - Size(BJI_{min});
SCA : = SCA - A_{min};
WHILE (Size(outBJI) ≤ S) DO
FOR each next element A_n in SCA Do
IF(nCOST[Q,(outBJI∪BJI_n)] < nCOST[Q,outBJI])
AND (Size(outBJI∪BJI_n) ≤ S) THEN
outBJI := outBJI∪BJI_n;
Size(outBJI) := Size(outBJI) + Size(BJI_n);
SCA := SCA - A_n; /* remove A_n from SCA */
end

(그림 12) 비트맵 조인 인덱스 선택 알고리즘

5. 실험 평가

본 연구의 알고리즘을 평가하기 위해서 Windows2008 Server 운영체제, 2GB 메모리, MS-SQL 데이터베이스를 이용하고 기존의 알고리즘[8]과 비교 실험하였다. 또한, [8]의 연구에서와 같은 데이터 세트와 40개의 OLAP 질의를 사용하여 실험의 데이터로 이용하였다. 본 실험의 데이터 웨어하우스는 하나의 사실 테이블 SALES와 다섯 개의 차원 테이블 TIME, CUSTOMERS, PRODUCTS, PROMOTIONS, CHANNELS 등으로 구성되었다. 데이터 웨어하우스 나머지에 대한 정보는 아래 <표 5>와 같다.

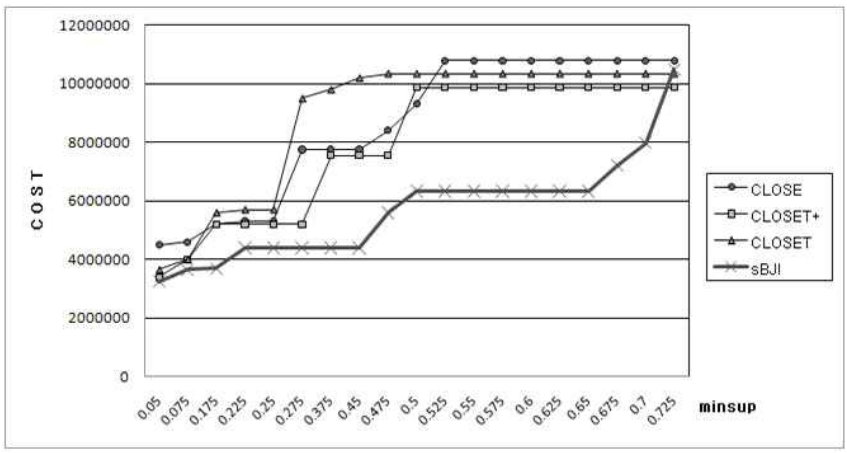
실험을 통한 비교 평가는 다음을 두 가지를 대상으로 진행하였다. 첫 번째, 지지도에 따른 인덱스를 선택하는데 있어 필요한 비용의 결과와 두 번째, 고정적인 지지도에 따른 기억용량의 효율성, 다양한 지지도에 따른 저장 용량의 변화 등이다.

〈표 5〉 실험 평가에 사용한 카디널리티

Table	number of rows
SALES	16,260,336
CUSTOMERS	50,000
PRODUCTS	10,000
TIME	1,461
PROMOTIONS	501
CHANNELS	5

5.1 지지도에 따른 비용 평가

기존 알고리즘(CLOSE, CLOSE+, CLOSET)과 제안하는 eFP-Tree를 이용하는 알고리즘(sBJI)에 대한 실험의 데이터로 주어진 <표 5>의 데이터 세트, minsup 변화 값, 40개의 OLAP 질의를 적용 실행하여 발생되는 비용을 비교하였다. 비교 결과는 (그림 13)과 같으며 minsup의 값이 0.175까지는 기존의 알고리즘과 비슷한 비용을 발생하여 성능의 개선



(그림 13) 최소지지도에 따른 비용(cost) 비교

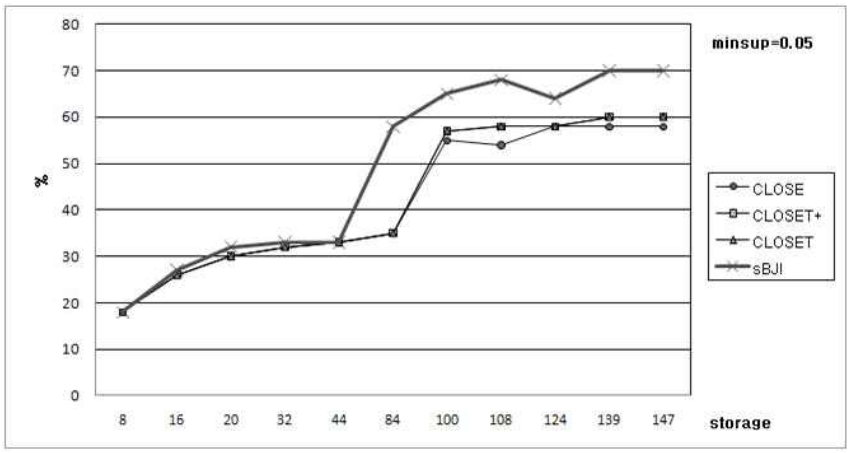
을 확인하지 못하였지만, minsup의 값이 0.225에서 0.7까지의 범위 내에서는 본 논문에서 제안하는 sBJI 알고리즘이 이전 알고리즘보다 대부분 모든 지지도 값에서 개선되었다는 것을 알 수 있었다. 그러나 minsup 값 0.5와 0.7에서 sBJI 알고리즘은 낮은 minsup 값에서의 비용보다 조금 높은 차이의 비용 발생으로 성능에 문제를 조금 보여 주었으나, 이전 알고리즘의 비용과 차이를 보여 개선된 것을 확인할 수가 있었다. 이후 minsup 값에서는 대부분 안정적인 비용을 발생함을 알 수가 있었다.

5.2 지지도에 따른 용량 효율성 평가

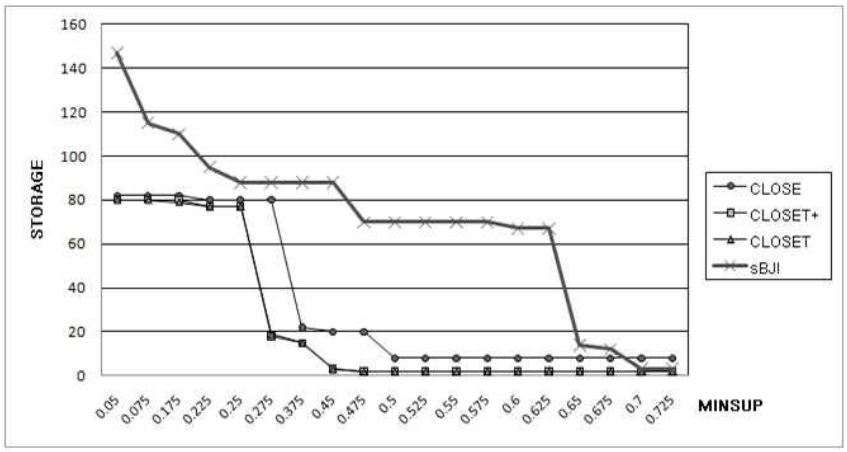
본 논문의 실험에서 제시한 데이터 세트의 사실 테이블 용량은 372.17MB이다. 첫 번째 minsup=0.05 기준에 대한 sBJI 알고리즘으로 생성된 비트맵 조인 인덱스 집합은 약 146.88MB 저장 용량을 필요로 하고 있다. 이 용량은 사실 테이블과 비교했을 때 크기가 좀 높음을 알 수가 있다. sBJI 알고리즘이 생성하는 용량이 높은 이유는 기존의 알고리즘과는 적용하는 매개변수 값이 다양하기 때문에 낮은 minsup에서는 높은 용량을 보이지만, 해당 용량에 대한 활용적인 측면에서의 효율성은 다른 결과를 보였다. 이를 평가하기 위하여 minsup의 값 0.05에 다양한 용량 값을 적용하여 이전 알고리즘과의 비교하였으며 결과는 아래 (그림 14)와 같다.

위 결과 낮은 용량에서는 거의 비슷한 효율을 가져 이전 알고리즘과는 비교, 평가하기 힘든 부분도 있었으나, 84MB 되는 지점부터 이전 알고리즘과는 조금의 변화된 효율을 보여 sBJI 알고리즘 성능이 개선된 것으로 볼 수가 있었다. 이러한 용량의 효율성은 전체 디스크의 손실 용량을 줄일 수 있다는 효과를 가지는 것이다. minsup=0.05의 용량 효율성 결과를 기반으로 다양한 minsup 값에 따른 비트맵 조인 인덱스 집합의 생성된 용량 결과는 아래 (그림 15)와 같다.

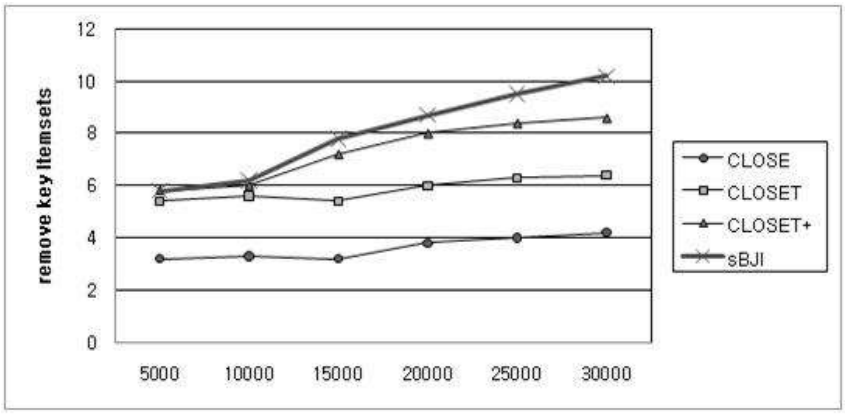
minsup에 따른 생성 용량은 이전 알고리즘보다 높으나 해당 용량에 따른 효율성 측면에서는 좋음을 앞 결과에서 살펴보았다. sBJI 알고리즘은 minsup=0.7이 되는 지점부터는 생성 용량이 많이 개선되고 있음을 알 수가 있다. 이 결과 sBJI 알고리즘은 minsup가 증가할수록 생성 용량은 줄어들었으며 효율성 측면에서는 다른 이전의 알고리즘보다 개선되었다고 할 수 있다.



(그림 14) minsup=0.5에 따른 용량 활용의 효율성 평가



(그림 15) 지지도에 따른 생성 용량 비교



(그림 16) |KAi|=5에 대한 실험 적용결과

5.3 키 속성과 연관된 항목집합 제거 효율성 평가

Apriori 원리에 의한 키 속성(KAi)과 연관된 항목집합의 제거 효율성을 평가하기 위하여 IBM data generator[18]을 이용하여 실험을 위한 TDB를 혼합 생성하였다. TDB는 다양한 질의를 의미하고 TDB에 포함된 항목은 질의에 사용된 속성을 의미한다. 데이터베이스 크기는 연속적인 5K, 10K, 15K, 20K, 25K, 30K로 생성하여 비교하였다. 생성된 TDB에서 각 트랜잭션의 평균 크기는 10개에서 30개의 항목들로 구성되었으며, 최소 지지도 임계값을 30%로 설정하였다. 본 논문에서 비트맵 인덱스 속성을 선택하기 위하여 키 속성을 가지 항목들은 후보 속성에서 제거하였고, 키 속성과 관련된 항목들도 연관 관계를 평가하여 비트맵 인덱스 후보 속성에서 제거를 고려하였다. 따라서 본 실험에서는 KAi와 연관된 항목의 제거 효율성을 이전 연구와 비교하기 위하여 KAi를 5개로 설정하여 실험한 결과는 위 (그림 16) 같이 나타낼 수 있다.

닫힌 빈발 항목집합 기반의 이전 연구인 CLOSE 알고리즘의 경우 키 속성과 관련된 항목 수는 변화 없음을 알 수 있다. 이 결과 후보 속성의 수는 많아지기 때문에 비트맵 인덱스의 선택에 있어 비효율적이라 볼 수 있다. 반면에 FP-Tree를 이용한 FP-성장 기반의 CLOSET와 CLOSET+ 알고리즘의 경우 키 속성과 연관된 항목의 제거 효율성은 닫힌 빈발 항목집합의 알고리즘보다 효율적이다. 하지만 본 논문에서 제안하는 eFP-Tree를 이용한 sBJI 알고리즘의 경우 트리의 노드에 키 속성 정보와 키와 연관된 속성 정보를 저장하고 각 항목들 간에 연관 관계를 평가하여 제거하기 때문에 질의와 항목 수의 크기가 클수록 제거 효율성이 이전 연구보다 우수한 성능을 보였다.

6. 결 론

본 논문에서 조인 연산을 필요로 하는 복잡한 질의어들의 처리 능률을 높이기 위해서 관계형 데이터 웨어하우스의 다중 테이블 인덱스의 사용에 대해서 연구했다. 특히 다중 테이블 조인 인덱스의 하나인 비트맵 조인 인덱스에 중점을 두었다. 저장 영역 제한 크기에 맞추는 비트맵 조인 인덱스들을 선택하는 문제를 정립했고, 빈발 항목집합의 생성을 이용한 데이터 마이닝 기법(FP-성장기반의 eFP-Tree 알고리즘)에 기초한 방법을 제시했다. 인덱스 가능한 후보 속성들을 대표하는 빈발 항목집합을 확인함으로써 비트맵 조인 인덱스 선택 문제의 탐색 공간을 축소하는 기법을 제시했다. 그리고 비트맵 조인 인덱스의 최종 구성 사항을 선택하기 위한 최적화 알고리즘(sBJI)을 제시했다. 최적의 비트맵 조인 인덱스를 선택하기 위해서 인덱스 선택을 위한 탐색 공간 축소를 위해서, 차원 테이블의 속성들의 빈도수만 고려하지 않고, 테이블 크기나 레코드 크기, 페이지 크기 등을 고려했다. 그리고 최적의 비트맵 조인 인덱스들을 선택하기 위해서 비용 함수를 사용한 최적화 알고리즘을 활용했다.

본 논문의 알고리즘을 평가하기 위해서 기존 연구의 데이터 세트를 기반으로 실험 연구를 하였다. 지지도에 따른 알고리즘의 비용을 평가한 결과 많은 개선이 된 것으로 보였으며, 인덱스 생성에 필요한 용량 평가에서는 초기 많은 용량의 생성으로 인하여 성능에 문제가 있는 것으로 보였으나 해당 용량의 사용 효율성 측면에서는 기존 알고리즘보다 개선된 것 알 수가 있었다. 하지만 minsup 값이 증가할수록 생성 용량이 크게 감소하여 기존 알고리즘과 비교했을 때 차이 없음을 보였다.

향후의 연구 과제로는 현재 본 논문의 알고리즘이 다양한 매개변수를 사용하고 있어 초기 생성되는 용량이 크기 때문에 낮은 minsup 값에서 용량의 개선 문제와 데이터 웨어하우스 환경에 변화가 있을 때 비트맵 조인 인덱스를 동적으로 수정하여 적용할 예정이다.

참 고 문 헌

[1] S. Chaudhuri, and V. Narasayya, “Self-tuning database systems: A decade of progress,” Proc. of the Intl. Conf. on VLDB, pp.3-14, 2007.

[2] M. Golfarelli, S. Rizzi, and E. Saltarelli, “Index Selection for data warehousing,” Proc. 4th Intl. Workshopon Design and Management of DataWarehouse, pp.33-42, 2002.

[3] P. O'Neil, and G. Graefe, "Multi-table joins through bitmapped join indices," SIGMOD Record 24, No.3, pp.8-11, 1995.
[4] C. Y. Chan, and Y. E. Ioannidis, "Bitmap index design and evaluation," Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, pp.355-366, 1998.
[5] C. Chee-Yong, "Indexing techniques in decision support systems," Phd. Thesis, University of Wisconsin-Madison, 1999.
[6] P. Valduriez, "Join Indices," ACM Trans. On Database Systems 12, 2, pp.218-246, June, 1987.
[7] S. Chaudhuri, "Index selection for databases: A hardness study and a principled heuristic solution," IEEE Trans. On Knowledge and Data Eng., pp.1313-1323, 2004.
[8] K. Aouiche, O. Boussaid, and F. Bentayeb, "Automatic selection of bitmap join indices in data warehouse," 7th Intl. Conf. on DataWarehouse and Knowledge Didcovery, pp.64-73, 2005.
[9] S. Chaudhuri, and V. Narasayya, "An efficient cost-driven index selection tool for Microsoft SQL server," Proc. of the Intl. Conf. on VLDB, pp.146-155, 1997.
[10] R. Agrawal and R. Srikant, "Mining Sequential Patterns," Proc. of the 11th International Conference on Data Engineering(ICDE'95), pp.3-14, 1995.
[11] D. Burdick, M. Calimlim, and J. Gehrke, "Mafia: a maximal frequent itemset algorithm for transaction databases," ICDB01, pp.443-452, 2001.
[12] J. Han, J. Pei, Y. Yin, R. Mao, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach," Data Mining and Knowledge Discovery, Vol.8, pp.53-87, 2004.
[13] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal, "Discoverying frequent closed itemsets," ICDT, pp.398-416, 1999.
[14] J. Han, J. Pei. and Y. Yin, "Mining Frequent Patterns without Candidate Generation," In Proceedings of the ACM-SIGMOD 2000 Conference, pp.1-12, 2000.
[15] Yi-Hung Wu, Chai-Ming Chiang, and Arbee L. P. Chen, "Hiding Sensitive Association Rules with Limited Side Effects," IEEE Transactions on Knowledge and Data Engineering, Vol.19, Issue 1, pp.29-42, 2007.
[16] H. Mannila and H. Toivonen, "Levelwise search and borders of theories in knowledge discovery," Data Mining and Knowledge Discovery, Vol.1, No.3, pp.241-258, 1997.
[17] 3. Ladjel Bellatreche, "A Data Mining Approach for Selecting Bitmap Join Indices," Journal of Computing Science and Engineering, Vol.1, No.2, December, 2007.
[18] http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/mining.shtml

안 형 근

e-mail : hkahn@ulsan.ac.kr
2003년 울산대학교 정보통신대학원 정보통신공학과(공학석사)
2008년 울산대학교 컴퓨터정보통신공학부(공학박사)
1997년~2004년 현대오토시스템 기술지원부
2004년~2006년 (주)CFIC 기업부설연구소 연구소장
2008년~2010년 울산대학교 컴퓨터정보통신공학부 객원교수
2011년~현 재 울산대학교 전기공학부 외래강사
관심분야 : 멀티미디어DB, DB설계/분석, ERP, BPM, Workflow

고 재 진

e-mail : jjkoh@ulsan.ac.kr
1972년 서울대학교 응용수학과(공학사)
1981년 서울대학교 계산통계학과(이학석사)
1990년 서울대학교 컴퓨터공학과(공학박사)
1975년~1979년 한국후지쯔(주) 기술개발부 사원
1979년~2010년 울산대학교 컴퓨터정보통신공학부 교수
2011년~현 재 울산대학교 전기공학부 교수
관심분야 : DB시스템, 전문가 시스템, DB설계, ERP