Journal of Korea Multimedia Society Vol. 9, No. 11, November 2006(pp. 1381-1394)

공간 데이터 웨어하우스에서 해쉬 테이블을 이용한 데이터큐브의 생성 기법

이 연[†], 김형선[††], 유병섭[†††], 이재동[††††], 배해영[†††††]

요 약

축적된 데이터를 기반으로 의사결정을 지원하는 데이터 웨어하우스에서 빠른 응답을 제공하기 위하여 데이터큐브 생성기법에 대한 많은 연구가 진행되었다. 대표적으로 다차원 배열을 사용한 기법과 hyper-tree를 기반으로 하는 H-cubing 기법이 연구되었다. 하지만 전자는 다차원 집계 연산에 필요한 모든 데이터를 배열로 저장하여 데이터의 양이 많아질수록 메모리 사용이 증가하였으며 후자는 hyper-tree를 기반으로 모든 튜플을 트리로 구축하여 트리 구축비용이 증가하였다. 본 논문에서는 데이터 웨어하우스에서 해쉬 테이블을 이용한 효율적인 데이터큐브 생성 기법을 제안한다. 제안 기법은 데이터큐브 생성 시 가중치 맵핑 테이블과 레코드 해쉬 테이블을 사용하여 다차원 데이터의 저장될 레코드 순서를 빠르게 찾아 저장한다. 따라서 데이터큐브의 생성속도가 향상되며 해쉬 테이블 만을 유지하여 메모리 사용량이 감소한다. 이는 성능평가를 통해 기존 기법보다 데이터의 빠른 검색과 데이터큐브 생성 요청에 빠른 응답을 보였다.

Data Cube Generation Method Using Hash Table in Spatial Data Warehouse

Li Yan[†], Kim Hyung Sun[††], Byeong-Seob You[†††], Jae-Dong Lee[††††], Hae-Young Bae[†††††]

ABSTRACT

Generation methods of data cube have been studied for many years in data warehouse which supports decision making using stored data. There are two previous studies, one is multi-way array algorithm and the other is H-cubing algorithm which is based on the hyper-tree. The multi-way array algorithm stores all aggregation data in arrays, so if the base data is increased, the size of memory is also grow. The H-cubing algorithm which is based on the hyper-tree stores all tuples in one tree so the construction cost is increased. In this paper, we present an efficient data cube generation method based on hash table using weight mapping table and record hash table. Because the proposed method uses a hash table, the generation cost of data cube is decreased and the memory usage is also decreased. In the performance study, we shows that the proposed method provides faster search operation time and make data cube generation operate more efficiently.

Key words: Data Warehouse(공간데이터웨어하우스), Data Cube(데이터큐브), Hash Table(해쉬테이블)

※ 교신저자(Corresponding Author) : 이 연, 주소 : 인천광역시 남구 용현동 인하대학교 하이테크센터 1008호(402-751), 전화 : 032)860-8712, FAX : 032)862-9845, E-mail : leeyeon@dblab.inha.ac.kr
접수일 : 2006년 6월 1일, 완료일 : 2006년 10월 19일
[†] 준회원, 인하대학교 컴퓨터공학부
[††] (주)카이네스 GIS공학연구소
(E-mail : sunnymsg@kainess.com)
[†††] 인하대학교 컴퓨터공학부
(E-mail : subi@dblab.inha.ac.kr)
[††††] 정회원, 단국대학교 정보·컴퓨터 학부
(E-mail :letsdoit@dku.edu)
[†††††] 정회원, 인하대학교 컴퓨터공학부
(E-mail : hybae@inha.ac.kr)
※본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성.지원사업의 연구결과로 수행되었음.

1. 서 론

최근 물류 관리, 입지 선정 등을 위주로 하는 기업에서 공간 정보에 기반하여 시장 변화를 빠르고 정확하게 분석하고 신속한 의사 결정 지원을 요구하고 있다. 따라서 과거부터 현재까지의 공간 및 비공간 데이터를 이용한 의사 결정 지원 시스템인 공간 데이터 웨어하우스의 중요성이 증가하였다[1-3]. 공간 데이터 웨어하우스는 공간 정보를 주제 중심적이고 통합적이며 시간성을 가지는 비휘발성 자료로 저장하여 효율적인 의사 결정을 지원하는 시스템으로 공간 데이터 웨어하우스 구축기(Builder)와 공간 데이터 웨어하우스 서버로 구성된다[4-6]. 공간 데이터 웨어하우스 구축기는 이질적인 운영 데이터베이스에서 데이터를 추출, 변환하여 공간 데이터 웨어하우스 서버로 변환된 데이터를 적재하며 공간 데이터 웨어하우스 서버는 사용자의 OLAP(On-Line Analytical Processing) 질의에 따라 데이터의 신속한 분석 결과를 제공한다[7-10]. 특히 공간 데이터 웨어하우스 서버는 적재된 데이터의 다차원 분석 결과를 빠르게 제공하기 위하여 기반 데이터를 미리 다차원 분석하고 그 계산된 결과 데이터로 큐브를 생성한다[11,12]. 따라서 데이터큐브 생성 성능은 의사 결정 지원에 대한 시스템 전체 성능에 중요한 영향을 미친다.

기존 데이터큐브의 생성 기법으로는 다중배열을 사용한 기법과 H-cubing을 사용한 기법 등이 제안되었다. 다중배열를 사용한 기법은 다차원 집계 연산에 필요한 기반 데이터를 메모리 크기에 맞춰 일정한 청크(Chunk)로 나누어 배열에 순차적으로 저장하고 저장된 데이터를 이용하여 다차원 집계 연산을 수행함으로써 데이터큐브를 생성한다[13,14]. 하지만 다차원 집계 연산을 위한 기반 데이터의 양이 증가하는 경우, 기반 데이터를 일정한 청크로 나누는 부하와 다차원 집계 연산 결과를 데이터큐브에 저장하는 비용이 증가하여 데이터큐브 생성이 느려지는 단점이 있다. H-cubing을 사용한 기법은 hyper-tree구조를 기반으로 하여, 데이터큐브 생성에 필요한 기반 데이터를 저장 한다[15]. 트리의 레벨은 같은 차원을 의미하며, 루트 노드부터 단말 노드까지의 간선에는 레코드를 저장하고, 각 단말 노드에는 해당 레코드의 정보를 갖는다. 따라서 데이터의 빠른 검색과 빠른 다차원 집계 연산이 가능하다. 하지만 기반 데이터의 레코드 수가 증가함에 따라 트리 구축비용이 증가하고, 트리 갱신 시 트리 재구성 비용이 증가하는 단점이 있다[16,17].

본 논문은 공간 데이터 웨어하우스에서 해쉬 테이블을 이용한 데이터큐브 생성 기법을 제안한다. 제안 기법은 가중치 맵핑 테이블과 레코드 해쉬 테이블을 사용하여 데이터큐브를 생성한다. 가중치 맵핑 테이블은 데이터큐브 생성에 필요한 비 집계 필드에 대해서만 생성하며 각 필드의 어트리뷰트 및 해당 필드의 어트리뷰트 모두를 포함하는 All을 추가하여 0부터 순차적인 값을 가중치 맵핑 테이블에 저장될 가중치 값으로 입력한다. 레코드 해쉬 테이블은 데이터큐브를 생성하는 각 레코드들에 대하여 가중치 매핑 값을 이용하여 계산된 해쉬 값과 레코드의 저장 위치를 관리한다. 레코드 해쉬 테이블에 저장된 해쉬 값은 데이터큐브에 저장될 레코드의 논리적 위치(순서) 값을 의미하며, 저장 위치는 해당 레코드가 실제로 저장된 주소를 의미한다. 데이터큐브에 저장될 레코드의 순서 값은 해당 레코드의 각 필드의 가중치 맵핑 테이블의 해쉬 값으로 계산한다. 해당 레코드에 대한 순서 계산이 끝나면, 레코드 해쉬 테이블은 해당 레코드가 저장된 주소 값을 저장 위치에 입력한다.

제안 기법은 데이터큐브 생성을 위해 해쉬 테이블을 이용하여 빠른 데이터 집계 연산으로 데이터큐브 생성 속도를 증가시켰으며 레코드 해쉬 테이블을 유지하여 저장 데이터의 빠른 검색 및 큐브 결과의 순서를 보장한다.

본 논문의 구성은 다음과 같다. 2장에서 기존의 데이터큐브 생성기법으로 다중배열과 H-cubing을 사용한 데이터큐브 생성기법에 대하여 설명하고, 3장에서는 제안 기법인 데이터큐브의 생성기법에 대해 설명한다. 4장에서는 해쉬 테이블을 이용한 데이터큐브 생성과정을 예를 들어 전체적으로 설명하고, 5장에서는 해쉬 테이블을 이용한 데이터큐브 생성기법의 성능평가를 한다. 마지막으로 6장에서는 결론을 맺은 뒤, 향후 연구에 대하여 기술한다.

2. 관련 연구

2.1 다중배열을 사용한 데이터큐브 생성 기법

다중배열기법은 데이터큐브를 생성하기 위해 기

반 데이터를 배열에 저장하여 다차원 집계 연산을 수행하는 기법으로 하향식(Top-down)으로 데이터 큐브를 계산한다[14]. 이는 데이터큐브생성 SQL에 포함된 Group-by절에 따라 데이터큐브 생성에 필요한 기반 데이터를 배열로 생성하게 된다. 배열로 구성된 기반데이터에 다차원 집계 연산을 수행하며, 수행이 완료된 기반데이터는 이후에 하나의 데이터큐브로 구성한다.

데이터큐브 생성 시 n개의 차원을 요구하면 SQL 절에 포함된 Group-by절에 따라 필드의 데이터를 나누고 배열에 적재하지만, 메모리의 한계로 인하여 배열로 생성하기 위한 필드 데이터를 청크(Chunk) 단위로 나누어 배열로 저장하고 다차원 집계 연산을 수행하게 된다.

[그림 1]은 다중배열을 사용한 데이터큐브 생성을 나타내고 있다.

[그림 1]에서는 3개의 비 집계 필드 A, B, C와 3개의 기반레코드를 가지는 테이블로 64개의 청크로 구성된 데이터큐브를 생성하였다. 다중배열기법은 A, B, C 차원 테이블을 각각 배열에 저장하기 위해 총 64개의 청크로 나누어 계산을 한다. 이렇게 하나의 청크는 배열에 저장되어 데이터큐브 생성을 위한 다차원집계 연산을 수행한다. 모든 청크의 다차원 집계 연산이 완료된 뒤, 각 청크에서 수행한 다차원 집계 연산의 결과를 데이터큐브로 생성하기 위해 순서를 계산하여 하나의 데이터큐브로 생성한다.

다중배열기법은 데이터큐브의 기반 데이터가 증가함에 따라 메모리에 모두 적재가 불가능하므로, 데이터를 일정한 청크로 나누고 배열에 저장하는 계산 비용이 증가한다. 따라서 모든 다차원 집계 연산을 수행하기 위해서는 각 청크의 모든 집계 연산을 수행하기 위해 메모리를 점유하고 있어야하는 문제점과 다차원집계 연산이 끝난 데이터를 하나의 데이터큐브에 저장하기 위한 계산 비용이 증가하는 문제점이 있다.

2.2 H-cubing을 사용한 데이터큐브 생성 기법

H-cubing은 hyper-tree를 기반으로 데이터큐브를 생성하는 기법으로 상향식(Bottom-up)으로 데이터큐브를 생성한다[15]. 데이터큐브 생성에 필요한 모든 데이터를 hyper-tree에 저장함으로써, 중복된 데이터에 대한 계산과 다차원집계 연산이 가능하다.

[그림 2]는 H-cubing기법을 사용한 데이터큐브 생성에 대하여 나타내고 있다.

[그림 2]와 같이 hyper-tree를 구성하기 위하여 큐브를 생성하는 기반 테이블인 Base Table로 비 집계 필드의 모든 어트리뷰트에 대한 집계연산 결과를 저장하는 Header Table을 구성한 뒤, Header Table로 hyper-tree를 구성한다. Header Table에서 Attr.Val는 Base Table에서의 비 집계 필드에 있는 모든 어트리뷰트를 입력하고, Quant-Info에는 해당 어트리뷰트에 대한 집계 값을 입력하게 되며, Side-Link에는 hyper-tree에서의 해당 위치를 가리키게 된다. hyper-tree의 루트 노드에는 Null값을 입력하고, 하나의 하위레벨에는 같은 필드의 데이터를 입력한 뒤, 마지막 리프노드에서는 해당 간선에 존재하는 값을 포함한 레코드의 집계 연산을 수행한다. 또한 Header Table에는 각 레벨마다 생성된 값을 Attr.Val에 저장하고 해당 간선에 존재하는 리프노드 값에 대한 집계 값을 계산하여 저장한다.

다차원 집계 연산을 위한 기반 데이터 레코드의 수와 필드 수가 증가하고 또한 필드 종류가 증가함에 따라, hyper-tree를 생성하는 계산 비용이 증가하고, 두 차원 이상 되는 범위에 대한 집계 값을 계산하기 위해서 hyper-tree를 검색하는 오버헤드가 발생하는

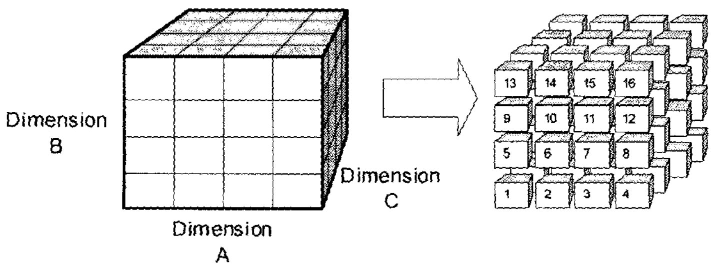
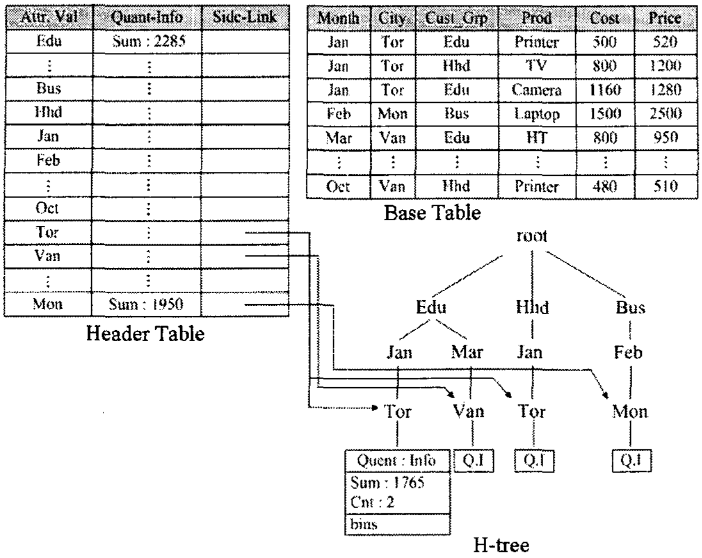


그림 1. 다중배열을 사용한 데이터큐브 생성



Attr. Val	Quant-Info	Side-Link
Edu	Sum : 2285	
⋮	⋮	
Bus	⋮	
Hhd	⋮	
Jan	⋮	
Feb	⋮	
⋮	⋮	
Oct	⋮	
Tor	⋮	
Van	⋮	
⋮	⋮	
Mon	Sum : 1950	

Month	City	Cust_Grp	Prod	Cost	Price
Jan	Tor	Edu	Printer	500	520
Jan	Tor	Hhd	TV	800	1200
Jan	Tor	Edu	Camera	1160	1280
Feb	Mon	Bus	Laptop	1500	2500
Mar	Van	Edu	HT	800	950
⋮	⋮	⋮	⋮	⋮	⋮
Oct	Van	Hhd	Printer	480	510

그림 2. H-cubing 기법을 사용한 데이터큐브 생성

단점이 있다.

3. 해쉬 테이블을 이용한 데이터큐브 생성 기법

해쉬 테이블은 해쉬 테이블에 기반 레코드의 저장소와 해쉬 키 값을 연결시켜 관리하여 별도의 트리 또는 정크의 생성과정이 필요 없게 된다. 또한 해쉬 테이블의 생성 속도는 정크 생성 또는 트리생성보다 훨씬 빠르다. 큐브 생성 후 레코드의 갱신, 검색 등 연산을 할 때 빠른 처리성능을 보이게 된다. 제안 기법에서는 레코드 해쉬 테이블에 기반하여 데이터큐브를 생성함으로 hyper-tree기반 기법 및 다중배열 기반 기법보다 더 빠른 큐브 생성 성능을 보여준다.

데이터 웨어하우스에서 데이터큐브는 분석하려는 데이터 및 분석하려는 데이터의 일반화 값까지 저장 하고 있으며, 그 개념적인 구성도는 [그림 3]과 같다.

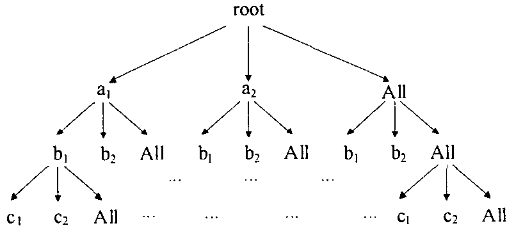


그림 3. 데이터큐브의 개념적인 구성도

[그림 3]은 각각 2개의 필드 값을 갖는 A, B, C 차원 테이블의 데이터큐브 결과에 대한 개념적인 구성도이다. 각 차원 테이블의 필드 값은 일반화 값인 'All' 필드 값을 가지며, 최종 데이터큐브 결과는 Tree의 후위순서(post-fix traverse)에 따라 (a1, b1, c1), (a1, b1, c2), … , (All, All, c2), (All, All, All) 순으로 구성하게 된다. 따라서 데이터큐브를 위한 다차원 집계 연산이 가능하다.

본 장 3.1에서는 제안기법의 전체적 과정을 살펴보도록 하고, 3.2에서는 레코드 해쉬 테이블의 생성을 위한 가중치 맵핑 테이블의 생성과정과 가중치 맵핑 테이블을 이용한 레코드 해쉬 테이블의 생성과정을 설명하고 3.3에서는 가중치 맵핑 테이블을 이용하는 레코드 해쉬 테이블의 생성과정을 설명한다. 3.4에서는 레코드 해쉬 테이블을 사용하는 데이터큐브 결과를 생성하는 알고리즘을 설명한다.

3.1 전체 구성

의사결정을 지원하는 데이터큐브를 생성하기 위하여, 제안 기법에서는 [그림 4]에서와 같은 구성을 사용한다.

[그림 4]에서는 우선 데이터큐브를 생성하기 위한 기반 테이블 Base Table의 데이터를 읽고 비 집계필드에 대한 가중치 맵핑 테이블을 만든다. 가중치 맵핑 테이블은 나중에 레코드의 집계정보를 저장할 수

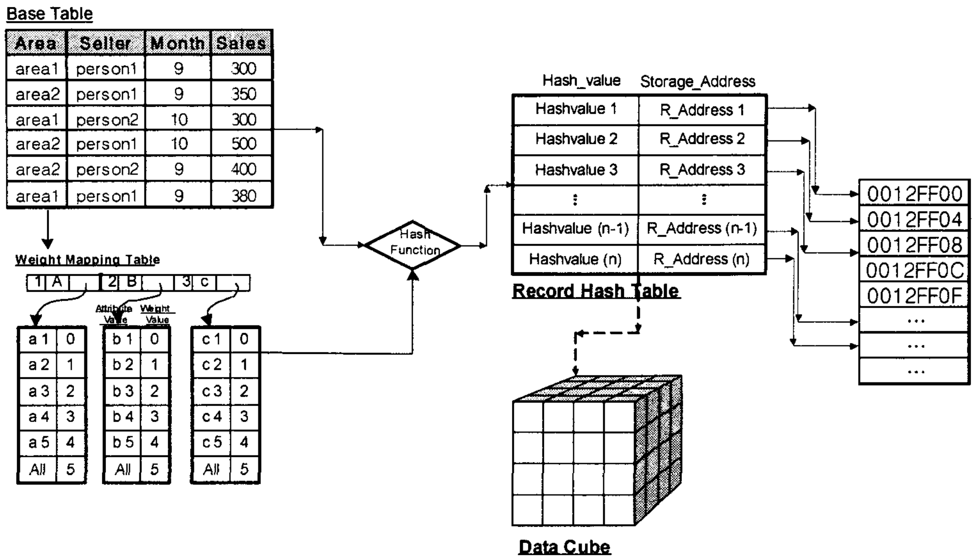


그림 4. 데이터큐브 생성 과정

있는 해쉬 테이블을 만들기 위한 기반 정보를 제공하며 또한 생성된 데이터큐브에 대한 검색 연산을 수행할 때 사용됩니다. [그림 4]에서의 레코드 해쉬 테이블은 집계연산 결과를 포함하는 데이터큐브를 전체적으로 관리하는 구조로서 해쉬 값과 데이터큐브를 구성하는 레코드들의 저장주소로 구성되었다. 이런 레코드 해쉬 테이블은 가중치 맵핑 테이블의 데이터로 해당 레코드의 해쉬 값을 계산하여 해쉬 값을 구성하고, 기반 테이블 데이터 및 해당 데이터에 대한 집계연산 결과를 저장하는 주소를 해당 해쉬 값에 대응되는 위치에 저장함으로써 생성된다.

제안기법에서 사용하는 해쉬 알고리즘을 사용함으로써 레코드 해쉬 값이 데이터큐브에서의 위치와 서로 매칭 된다. 따라서 레코드 해쉬 테이블을 통한 데이터큐브에 대한 연산이 보다 빠르게 진행될 수 있고, 의사결정을 위한 데이터를 빨리 제공할 수 있다.

3.2 가중치 맵핑 테이블의 생성

제안 기법에서는 하나의 데이터큐브에 대하여 하나의 가중치 맵핑 테이블을 생성하는데, 이런 가중치 맵핑 테이블은 기반 데이터 테이블의 집계 필드를 제외한 필드에 대해서만 생성한다. 비 집계 필드에 대하여 각 필드마다 하나의 가중치 테이블을 생성하고 전체 맵핑 테이블을 가중치 테이블에 연결하여 하나로 묶는다.

필드에 대한 가중치 테이블 구조는 [그림 5]에서처럼 Attr_Value와 Weight_Value 로 이루어진다. Attr_Value는 해당 필드에 저장된 어트리뷰트의 값을 저장하며, Weight_Value 는 해당 어트리뷰트의 가중치 값을 저장한다.

가중치 테이블의 구조는 [그림 5]와 같다.

가중치 테이블을 실제로 구성할 때, 하나의 비 집계 필드에 대하여 하나의 가중치 테이블을 생성하고 중복되지 않는 어트리뷰트값을 순차적으로 가중치 테이블의 Attr_Value에 저장하고, 필드 테이블의 모든 필드 값을 중복 없이 입력을 모두 마치면, 마지막 필드 값으로 All을 저장한다. 완성된 가중치 테이블의 필드 값에 0부터 순차적인 가중치 값을 입력한다. [그림 6]은 비 집계 필드 A에 대한 가중치 테이블을

Attr_Value	Weight_Value
Attr_Value_1	Weight_Value_1
Attr_Value_2	Weight_Value_2
Attr_Value_3	Weight_Value_3
⋮	⋮
Attr_Value_n	Weight_Value_n
All	Weight_Value_n+1

그림 5. 가중치 테이블의 구조

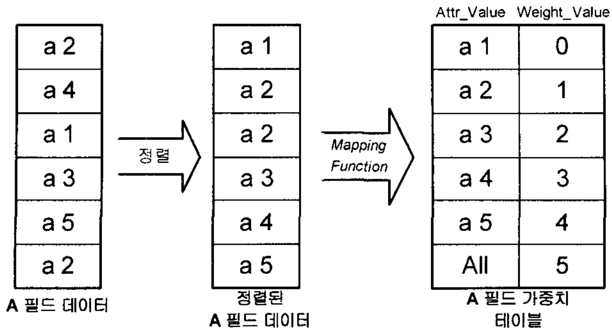


그림 6. 가중치 테이블의 구성 과정

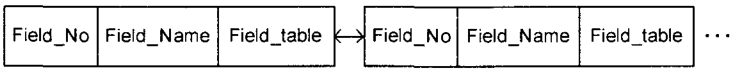


그림 7. 가중치 맵핑 테이블 정보리스트 구조체

생성한 예를 보여준다.

[그림 6]과 같이 A필드에 대한 가중치 맵핑 테이블 생성하기 위해서는 우선 A필드에 대한 정렬을 수행한다. 이는 중복되지 않는 필드 값을 검색하기 위한 계산 비용을 줄인다. 다음으로 A필드에 대한 중복되지 않는 필드 값의 총 개수를 계산하여 5라는 결과를 받는다. A필드에 대한 중복 없는 필드 값을 가중치 테이블에 저장하고, All을 마지막 Attr_Value값으로 저장한다. 마지막으로 A필드의 가중치 테이블의 Attr_Value에 저장된 a1, a2,..a5, All의 필드 값에 대응되는 Weight_Value 값을 0부터 5까지 순차적으로 입력하면 A필드에 대한 가중치 테이블이 완성된다.

가중치 테이블은 필드 값들과 가중치 값을 저장하고 관리하는 구조이며 이렇게 생성된 가중치 테이블은 [그림 7]와 같이 가중치 맵핑 테이블 정보리스트를 통해 관리된다.

[그림 7]에서 가중치 맵핑 테이블 정보 리스트는 해당 필드가 몇 번째 필드인지를 나타내는 Field_No와 필드 이름을 나타내는 Field_Name, 및 해당 가중치 테이블을 가리키는 포인터 Field_table로 구성되었다. 가중치 맵핑 테이블 정보리스트는 맵핑 테이블을 갖는 모든 필드의 정보를 담고 있어, 이런 정보는 큐브 생성시 레코드 해쉬테이블에 정보를 제공해 주며 또한 레코드 검색 시 신속한 위치정보를 제공하게 된다. 가중치 맵핑 테이블정보리스트를 이용하여 각각의 가중치 맵핑 테이블을 관리하는 전체 구조는 [그림 8]에서와 같다.

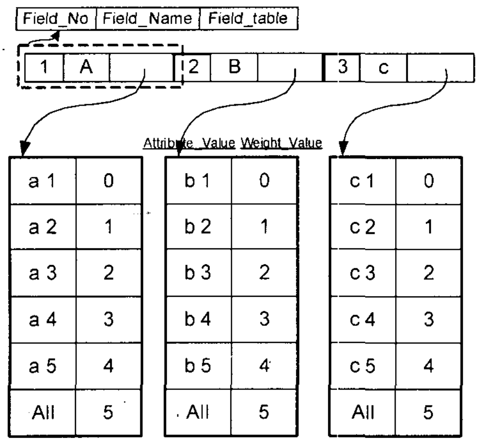


그림 8. 큐브의 가중치 맵핑 테이블

[그림 8]에서는 큐브를 생성하는 비 집계테이블인 A, B, C 필드에 대해서 각각 가중치 테이블을 생성하고 해당 정보를 가중치 맵핑 테이블정보리스트에 저장한다.

3.3 레코드 해쉬 테이블의 생성

레코드 해쉬 테이블은 레코드들의 각 가중치 테이블의 해쉬 값 계산 결과를 이용하여 데이터큐브 결과 순서와 저장 위치를 관리하는데 사용된다. 이런 레코드 해쉬 테이블을 생성하기 위해서는 다음과 같은 순서를 거친다. 우선 레코드 해쉬 테이블을 생성하기 위해 데이터큐브에 저장될 최대 레코드 개수 MaxRecordCnt을 (식 1)에 따라 계산한다.

$$MaxRecordCnt = \prod_{i=1}^{n} Weight\_ValueCnt_i \cdots\cdots\cdots\cdots\cdots (1)$$

식 (1)에서 n은 기반 테이블에서 비 집계 필드의 개수이고, i 는 가중치 맵핑 테이블정보 리스트에 저장되어 있는 해당 필드의 Field_No값이고, Weight_ValueCnt는 해당 필드에 있는 어트리뷰트의 개수이다. 이렇게 계산된 결과에 따라 해당 가중치 맵핑 테이블과 MaxRecordCnt개의 해쉬 값을 갖는 레코드 해쉬 테이블을 생성하고, 기반 레코드의 해쉬 값을 계산하고 저장소 값을 초기화 하며 할당된 저장 공간에 데이터큐브 레코드를 저장한 뒤, 레코드 해쉬 테이블의 Storage_Address에 저장 공간 주소를 저장한다.

레코드 해쉬 테이블은 Hash_value와 Storage_Address로 이루어진다. Hash_value는 데이터큐브의 레코드 순서를 의미하고, Storage_Address는 데이터큐브의 레코드가 저장된 저장소의 위치를 의미한다. 레코드 해쉬 테이블의 구조는 [그림 9]과 같다.

[그림 9]과 같이 레코드 해쉬 테이블은 데이터큐브에 저장될 레코드 순서와 데이터큐브 레코드와 집계값을 저장한 저장 공간의 주소를 관리한다. 이런 레코드 해쉬 테이블의 생성 과정은 해쉬 값의 생성과정과 데이터큐브 레코드 저장소 저장 두 개 과정을 거친다. 그 중에서 해쉬 값 Hash_value는 레코드를 구성하는 각 필드들의 어트리뷰트를 3.2절에서 만들어진 가중치 맵핑 테이블과 매칭시켜 해당 가중치 값을 사용하여 다음 식 (2)에 따라 계산한다.

$$Hash\_Value = \sum_{i=1}^{n} (Weight\_Value_i \times \prod_{i+1}^{n} Weight\_ValueCnt_i) \cdots\cdots\cdots\cdots (2)$$

식 (2)에서 n은 레코드를 구성하는 필드의 개수이고, i는 레코드를 구성하는 i번째 필드임을 의미하는 값으로서 가중치 맵핑 테이블정보 리스트에 저장되어 있는 해당 필드의 Field_No값이다. 그러므로 $Weight_Value_i$는 i번째 필드에서 해당 어트리뷰트의 가중치 값이고 Weight_ValueCnt는 i번째 필드가 갖고 있는 전부의 어트리뷰트 개수 즉 Weight_Value의 개수이다. 이렇게 계산된 해쉬 값은 레코드 해쉬 테이블에서 유일한 값을 가지며 데이터큐브 데이터와 매칭하여 볼 때 순차적으로 배치된다.

Hash_value	Storage_Address
Hashvalue 1	R_Address 1
Hashvalue 2	R_Address 2
Hashvalue 3	R_Address 3
⋮	⋮
Hashvalue (n-1)	R_Address (n-1)
Hashvalue (n)	R_Address (n)

그림 9. 레코드 해쉬 테이블의 구조

[그림 10]에서처럼 데이터큐브에서의 하나의 셀은 A,B,C 세 필드로 구성된 레코드의 집계연산 값을 저장하고 있으며 해당 해쉬 값은 가중치 맵핑 테이블로 식 (1)에 따라 계산한다. 이렇게 계산 되었을 때 해쉬 값은 데이터큐브에 순차적으로 배치되며 유일한 값을 가지게 된다.

레코드 해쉬 테이블을 구성하는 다른 한 요소인 Storage_Address는 해당 데이터큐브 레코드의 저장순서에 맞는 레코드와 해당 레코드의 집계 값을 저장하기 위해 할당된 저장 공간의 주소를 저장한다. 레코드 해쉬 테이블을 생성 할 때 Storage_Address의

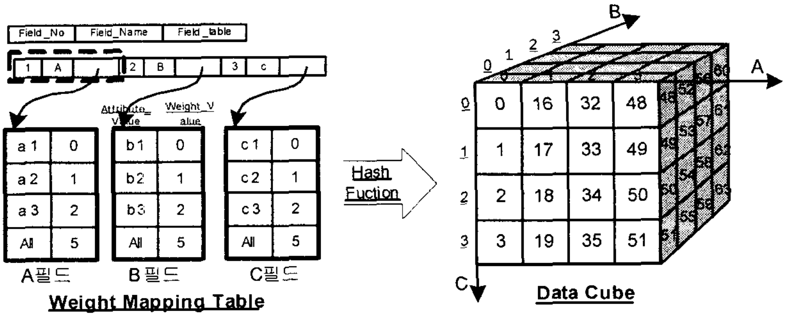


그림 10. 해쉬 값 계산 결과

초기값은 NO_ADDRESS로 입력된다. 그리고 집계 연산의 결과 값을 포함하는 데이터큐브가 생성된 후 해당 레코드 및 집계연산 결과 값을 Storage_Address에 입력한다. Storage_Address의 초기값을 NO_ADDRESS로 입력하면 데이터큐브의 구축 완료 후 검색 연산을 진행 할 때 Storage_Address에 NO_ADDRESS값이 입력되어 있는 지를 이용해서, 해당 해쉬 값에 할당된 저장소의 저장공간이 있는지를 구분한다. 따라서 데이터큐브 결과 생성 시, 해당 순서에 값이 있는지 검색하는 검색 비용이 감소한다. 상세한 데이터큐브 생성과정은 3.4에서 설명한다.

3.4 데이터큐브 생성

데이터큐브 생성 순서는 기반 테이블 레코드를 순차적으로 읽어, 레코드에 대하여 생성될 수 있는 모든 경우의 일반화 레코드를 구성하여 저장한다. 다음 식 (1), 식 (2)를 이용하여 각 일반화 레코드의 해쉬값을 계산하여 레코드 해쉬 테이블의 Storage_Address에 일반화 레코드의 저장소 위치를 기록 한다. 이렇게 모든 레코드에 대하여 일반화 레코드를 구성하고 레코드 해쉬 테이블의 주소록에 기록하는 과정에서 만약 해당 해쉬 값에 기존에 저장된 일반화 레코드가 있다면, 집계 값에 입력된 레코드의 집계값을 더하여 수정된 집계값을 얻고 그 값을 저장한다. 기존에 저장된 일반화 레코드가 아니라면, 해당 일반화 레코드를 저장소에 저장하고, 저장소 주소를 레코드 해쉬 테이블에 기록하는 식으로 기반 데이터 레코드와 이에 따른 일반화 레코드를 계산하여 데이터큐브 생성 순서를 거친다. 이 과정을 flow-chart로 표시하면 다음 [그림 11]과 같다.

[그림 11]에서는 데이터큐브를 생성하기 위하여 우선 기반 데이터 레코드를 하나씩 읽어 모든 일반화 레코드를 생성하고 저장공간을 할당하여 저장한다. 여기에서 기반 데이터 레코드의 일반화는 레코드 집계 필드를 제외한 나머지 필드의 개수를 넣어 생성한다. 따라서, 하나의 레코드가 데이터큐브에 생성하는 일반화 레코드 생성 개수는 식 (3)와 같이 계산된다.

$$All\,Record = 2^n - 1 \cdots\cdots\cdots\cdots (3)$$

식 (3)에서 AllRecord은 하나의 데이터큐브 레코드에 생성되는 일반화 레코드 총 개수이고 n은 기반 테이블 레코드의 비 집계 필드의 개수이다.

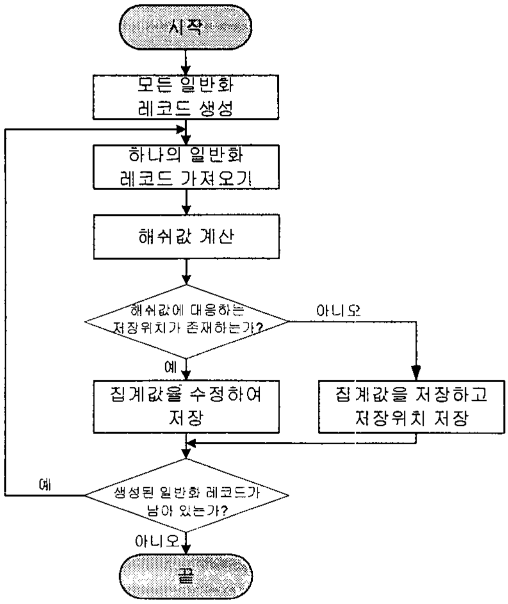


그림 11. 데이터큐브 생성 순서도

다음 생성된 일반화 레코드의 저장주소를 레코드 해쉬 테이블에 저장하게 되는데, 이 과정에서 레코드의 해쉬값을 계산하여 해당 레코드의 Storage_Address 찾아 저장소를 저장하게 된다. 만약 찾은 Storage_Address 값이 NO_ADDRESS(Null)이면 레코드의 저장주소를 저장하고, 그렇지 않을 경우 저장된 집계값에 기존 레코드의 집계값을 더하여 새로운 집계값을 얻어 일반화 레코드의 집계값을 갱신한다. 이렇게 모든 레코드를 읽어 데이터큐브를 생성한다.

4. 데이터큐브 생성과정

본 장에서는 해쉬 테이블을 이용한 데이터큐브 생성과정을 예제 테이블을 이용하여 설명한다. [그림 12]는 본 절에서 예제로 사용하기 위한 도서 판매 테이블이다.

Area	Seller	Month	Sales
area1	person1	9	300
area2	person1	10	350
area1	person2	9	300
area1	person1	10	500
area2	person2	10	400

그림 12. 도서 판매 테이블

[그림 12]는 area1과 area2 지역에서 9월과 10월에 도서를 판매한 person1과 person2를 나타내는 도서 판매 테이블이다. 도서 판매 테이블을 기반 차원 테이블로 다차원 분석한 뒤, Area필드의 area1과 area2를 갖는 공간 데이터로 구성된 차원 테이블을 갖는 데이터큐브를 생성한다. 따라서 [그림 12]는 공간 데이터를 기반으로 한 의사 결정 지원이 가능하다.

3.2, 3.3, 3.4에서 설명한 순서에 따라 데이터큐브를 생성한다.

4.1 가중치 맵핑 테이블의 생성

[그림 12]에서의 도서 판매 테이블에서 비 집계 필드로는 지역 필드인 Area, 판매원 필드인 Seller, 월 필드인 Month이고, 집계 필드로는 판매액 필드인 Sales이다. 도서 판매 테이블의 판매액 필드는 집계 값을 계산하기위한 집계 필드(Aggregation Field)로서, 가중치 맵핑 테이블이 생성하지 않는다. 따라서 지역 필드와 판매자 필드 그리고 날짜 필드에 대하여 [그림 13]과 같이 가중치 맵핑 테이블을 생성한다.

[그림 13]에서 보면 각 필드에 대한 중복되지 않는 필드 값을 가중치 맵핑 테이블의 가중치 값으로 하며 가중치 맵핑 테이블로 생성하였다. 또한 각 가중치 맵핑 테이블에 All값을 추가하였으며 각 가중치 맵핑 테이블의 해쉬 값에는 0부터 순차적인 값을 저장한 것을 볼 수 있다.

4.2 가중치 맵핑 테이블을 이용한 레코드 해쉬 테이블의 해쉬 값 계산

레코드 해쉬 테이블은 각 가중치 맵핑 테이블의

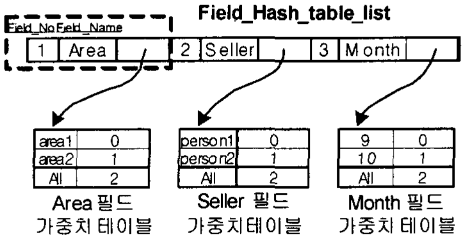


그림 13. 도서 판매 테이블 가중치 매핑 테이블

가중치 값을 이용하여 데이터큐브 레코드에 저장하기 위한 해쉬 값을 계산한다. [그림 14]는 도서 판매 테이블의 다섯 번째 레코드 (area2, persion2, 10, 400)에 대하여 집계 값을 갖는 Sales필드를 제외한 필드에 대하여 해쉬 값을 계산하고 있다.

[그림 14]에서 레코드의 해쉬 값인 Hash_Value를 계산하기 위해 필드 가중치 맵핑 테이블을 사용하여 식 (2)를 적용시킨다. 5번째 레코드에 대응되는 비 집계 필드의 개수는 3 이므로 위 수식에 적용될 n값은 3이다.

n= 1일 때 대응되는 가중치 값인 $Weight_Value_1=1$이고, 2번째 필드Seller와 3번째 필드 Month의 Weight_Value의 개수는

$Weight\_ValueCnt_2= 3$; $Weight_valueCnt_3= 3$이다.

그러므로 i=1일 때 $Hash_Value_1= 1\times3\times3=9$이다.

이와 같은 방법으로 Seller필드와 Month필드에 해당되는 부분레코드 해쉬 값을 계산한다.

i = 2일 때: $Hash\_Value_2= 1\times3=3$

i = 3일 때: $Hash\_Value_3= 1\times1=1$

이렇게 나온 각 필드의 부분레코드 해쉬 값을 더

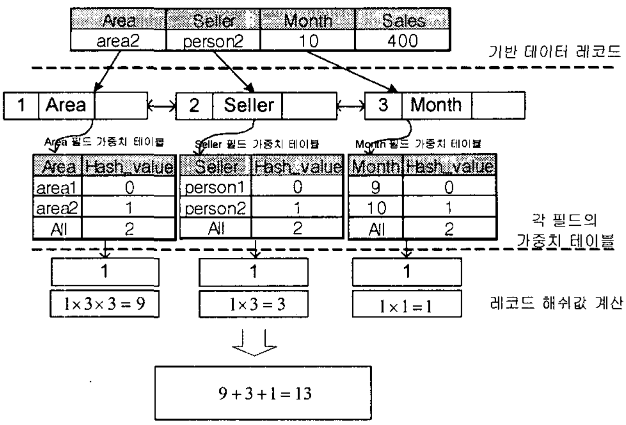


그림 14. 가중치 맵핑 테이블의 해쉬 값 계산

하게 되면, 도서 판매 테이블의 다섯 번째 레코드 해쉬 값은 13이다. 같은 방법으로 기타 레코드 및 집계 연산 결과 값의 해쉬 값을 계산할 수 있다.

4.3 데이터큐브 생성을 위한 집계연산 레코드 생성

의사결정을 지원하는 데이터큐브를 생성하기 위해서는 모든 기반 데이터의 집계연산을 하여 일반화 레코드를 만들고 레코드 해쉬 테이블에서 관리하게 된다. 기반 테이블의 5번째 레코드에 대한 집계연산 레코드를 만드는 과정을 보도록 한다. 식 (3)을 이용하여 도서 판매 테이블의 5번째 레코드 (area2, persion2, 10, 400)이 데이터큐브 레코드에 생성되는 레코드의 총 개수를 계산하면 집계 값을 제외한 필드의 총 개수가 3개이므로, 총 7개의 데이터큐브 레코드를 생성하고 저장 공간을 할당한다.

[그림 15]는 5번째 레코드에 일반화를 포함하여 데이터큐브에 생성되는 레코드를 나타낸다. [그림 15]를 보면, 마지막에 생성된 일반화 레코드는 (All, All, All)임을 확인할 수 있다. (All, All, All)레코드는 모든 레코드를 포함하는 일반화 레코드로 다른 레코드 계산 시, (All, All, All)레코드는 집계 값에 대한 추가만 이루어진다. 같은 방법으로 기반 테이블의 모든 레코드에 대한 집계 연산을 한다. 레코드에 대한 집계 연산 과정은 레코드 해쉬 테이블의 생성과정과 같이 진행하게 된다.

4.4 레코드 해쉬 테이블 및 데이터큐브 생성

레코드 해쉬 테이블은 데이터큐브를 구성하는 레코드의 순서를 저장하고, 집계 연산 결과를 저장하므로, 우선 레코드 해쉬 테이블에 데이터큐브 테이블에 저장될 레코드의 개수만큼의 순차적 해쉬 값을 갖는 레코드 해쉬 테이블을 생성한다. 데이터큐브에 저장될 레코드 수의 최대값은 식 (1)에서와 같이 계산한다.

Area	Seller	Month	Sales
area2	person2	All	400
area2	All	10	400
All	person2	10	400
area2	All	All	400
All	All	10	400
All	person2	All	400

그림 15. 데이터큐브에 생성되는 레코드

식 (1)을 적용하여 도서 판매 테이블에 대한 데이터큐브 레코드의 총 개수를 계산 하면

Weight_ValueCnt1=3,

Weight_ValueCnt2=3,

Weight_ValueCnt3=3 이므로, MaxRecordCnt값은 27 이다. 즉 데이터큐브에 저장될 레코드 수의 최대 값은 27개 이다. 따라서 레코드 해쉬 테이블에서 필요한 해쉬 값도 27개가 된다. 레코드 해쉬 테이블에 생성된 27개의 순차적인 해쉬 값의 초기 저장 공간 주소는 NO_ADDRESS로 설정한다. 따라서 4.2에서 계산한 도서 판매 테이블의 다섯 번째 레코드에 대한 계산 결과는 13번째 해쉬 값에 저장하기 위해 저장 공간을 할당하여, 할당된 저장 공간 주소에 다섯 번째 레코드 (area2, person2, 10, 400)를 저장한다. 레코드 해쉬 테이블의 해쉬 값이 13인 레코드의 Storage_Address에는 NO_ADDRESS를 대신하여 다섯 번째 레코드를 저장한 저장 공간의 주소를 입력한다. 같은 방법으로 기타 레코드 및 해당 집계 연산 결과의 주소를 입력하고 레코드 해쉬 테이블의 생성 과정을 완성한다. 이렇게 생성된 레코드 해쉬 테이블로 3.4절에서 설명한 방법을 사용하여 데이터큐브를 생성한다.

5. 성능평가

본 장에서는 제안기법의 데이터큐브 생성을 위한 저장소와 수행 소요시간 및 검색질의를 수행할 때의 수행시간을 다중배열기법과 H-cubing기법과 비교하여 평가하였다.

5.1 데이터큐브 생성성능 비교

공간데이터웨어하우스의 데이터큐브 생성기법을 성능평가하기 위한 평가환경은 [표 1]과 같다.

표 1. 성능평가에 사용된 시스템 환경

기종	IBM PC Compatible
CPU	Pentium IV 3.0GHz (LGA 775) L2 Cache : 1MB
Memory	1GB (DDR 400MHz)
HDD	160GB, 7200RPM, Buffer 8MB, S-ATA
OS	Windows XP Professional

[표 1]과 동일한 사양의 시스템에 운영 데이터베이스 2대와 구축기와 서버가 각 1대로 총 4대의 시스템으로 구축되어 있으며, 구축기는 데이터큐브의 기반 데이터가 추출되어 데이터큐브 생성 요청을 대기한다.

본 성능평가는 서울시 건축물 정보를 모델링하여 테스트 데이터로 하며 랜덤으로 생성하였다. 공간 좌표점 정보를 포함하는 테스트데이터는 데이터 자체가 비공간 데이터보다 용량이 크다. 이러한 대용량 공간 데이터(1 만개 ~ 9 만개)를 기반으로 기존 기법과 제안 기법의 데이터큐브 생성과정에서 사용하는 저장소의 크기와 생성시간을 비교 평가한다. [그림 16]은 데이터큐브 생성과정에서 사용되는 저장소의 크기를 비교 및 평가한 결과이다.

[그림 16]에서 보면, 다중배열을 사용한 기법은 모든 생성 가능한 차원테이블데이터를 배열에 적재하기 위하여, 배열을 생성하므로 보다 많은 메모리를 요구하게 되고, 제안 기법은 가중치 맵핑 테이블과 레코드 해쉬 테이블을 사용하여 데이터큐브를 생성하기 때문에 이런 테이블을 저장하기 위한 별도의 저장소가 필요하므로 H-cubing기법 보다 많은 메모리 공간을 필요로 한다. 다중배열기법과 비교하여 보면 소요 저장 공간은 최소 30% 적은 것을 볼 수 있다. 하지만 메모리와 디스크의 가격 하락으로 대용량 저장 공간을 이용할 때 저장 공간의 차이가 크게 문제되지 않는다.

[그림 17]은 차원 테이블과 저장된 레코드의 개수가 증가됨에 따라 소요 시간을 비교하여 평가하였다. [그림 17]에서의 성능평가는 기반이 되는 차원 테이블의 개수를 1개부터 10개까지 증가시키면서 5000개의 공간데이터를 사용하여 데이터큐브를 생성하였다.

실험 결과는 데이터큐브 생성을 요청한 시점부터, 데이터큐브 생성이 완료되는 시점까지의 시간을 측

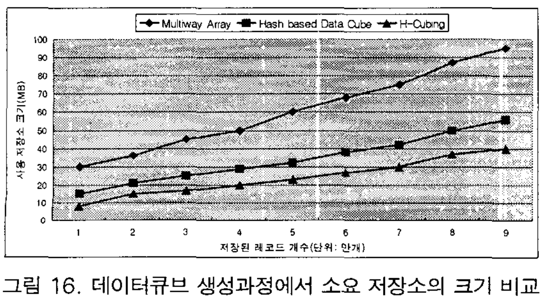


그림 16. 데이터큐브 생성과정에서 소요 저장소의 크기 비교

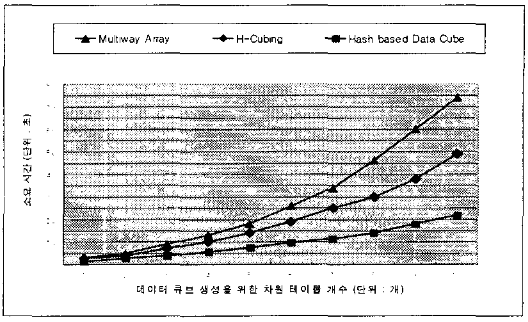


그림 17. 차원 테이블에 따른 데이터큐브 생성응답시간 비교

정 하였다.

[그림 17]에서 다중배열을 사용한 기법은 데이터큐브 생성 시 차원 테이블 데이터를 배열에 적재하여 데이터큐브를 생성한다. 따라서 데이터큐브에서 필요로 하는 차원 테이블의 증가에 따라 이를 계산하기 위한 많은 처리 비용을 요구하였다. H-cubing을 사용한 기법의 경우, 데이터큐브 생성 시 모든 차원 테이블 데이터를 트리로 구성하여 데이터큐브 생성을 수행함으로 데이터큐브 생성에 필요로 하는 차원 테이블의 개수가 증가하면서 트리 구축비용과 다차원 집계 연산을 위한 트리 검색 비용이 증가하였다. 제안 기법은 데이터큐브 생성 시, 각 필드마다 가중치 맵핑 테이블의 유지비용을 요구하지만, 데이터큐브 생성 시 레코드 해쉬 테이블과 가중치 맵핑 테이블의 사용으로 레코드 필드에 대한 빠른 저장 위치 계산과 다차원 집계 연산을 수행하게 된다. 또한 레코드 해쉬 테이블을 유지하여 데이터큐브 결과의 순서 계산과 저장된 레코드의 빠른 검색을 가능하게 하였다.

따라서 데이터큐브 생성과정에서의 응답 시간은 제안기법이 다중배열기법보다 최대 3.5배, H-cubing 기법보다 최대 2배 성능이 향상되었다.

5.2 검색 연산에 대한 평가

[그림 18]은 검색 연산에 대하여 다중배열을 사용한 기법과 H-cubing을 사용한 기법 및 제안기법을 비교하여 평가하였다. 평가 환경은 5.1에서 소개된 환경이며, 기반 레코드를 5000개로 하였고, 한 번에 검색 요청한 공간데이터가 하나의 레코드, 전체 레코드의 1/3, 전체 레코드일 때 검색 질의의 수행시간의 변화를 비교하고 평가하였다.

[그림 18]에서 보면 다중배열을 사용한 기법은 배

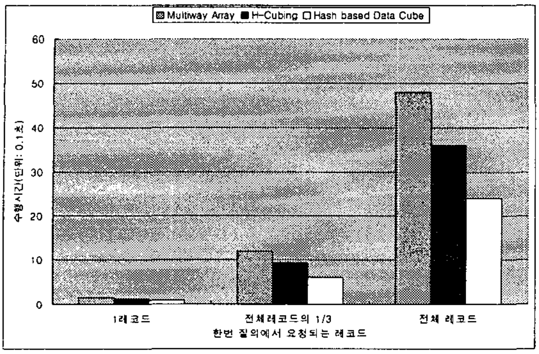


그림 18. 검색질의에 대한 수행 시간 비교

열에서 레코드를 검색해야 되므로 H-cubing 기법이나 제안기법보다 더 많은 수행 시간을 소요한다. H-cubing을 사용한 기법은 hyper-tree에서 검색하므로 수행 속도가 해쉬기법을 사용한 제안기법보다 많은 수행 시간을 소요한다. 따라서 세 가지 방법으로 생성한 데이터큐브에서 검색 질의를 수행할 때 제안 기법은 다중배열기법보다 최대 2배, H-cubing 기법보다 최대 1.5배의 성능이 향상되었다.

5. 결론 및 향후연구

본 논문은 공간데이터 웨어하우스에서 해쉬 테이블을 이용한 데이터큐브의 생성기법에 대하여 제안하였다. 제안기법은 데이터큐브의 빠른 생성과 갱신 및 검색을 고려하여 가중치 맵핑 테이블과 레코드 해쉬 테이블을 사용하여 중복되지 않은 필드 값을 저장하고 해쉬 함수를 통해 데이터큐브에 해쉬값의 오름차순으로 레코드를 저장한다. 따라서 효율적으로 다차원 집계 연산을 하여 데이터큐브를 생성하고, 데이터큐브의 생성과정에서 가중치 맵핑 테이블과 레코드 해쉬 테이블만 유지하기 때문에 적은 메모리 공간을 사용하며, 해쉬 함수를 통한 빠른 검색 성능을 제공하게 된다. 성능평가에서는 데이터큐브의 생성을 위해 필요한 저장 공간과 데이터큐브의 생성속도를 기존 기법과 비교하여 평가하였다. 성능평가 결과에서 큐브생성 시 필요한 저장 공간은 다중배열기법의 평균 40% 적었고, 데이터큐브의 생성과정에서 응답 시간은 제안기법이 다중배열기법 보다 최대 3.5배, H-cubing 기법보다 최대 2배 성능이 향상되었다. 또한 제안기법은 검색연산에서 다중배열보다 최대 2배, H-cubing기법보다 최대 1.5배의 성능향상이 향상되었다. 제안 기법은 데이터큐브의 생성속도를 향상시키고 공간사용량을 줄였으며 보다 빠른 검색 성능을 제공하므로 제안 기법을 사용하여 기존 기법보다 빠르고 효과적인 의사결정을 지원할 수 있다.

향후연구로는 기존 데이터큐브의 필드에 새로운 레코드 값을 추가하는 데이터큐브 갱신 기법 연구가 필요하다.

참 고 문 헌

[1] Lafond, "Designing and Building the Distributed Geospatial Data Warehouse Architecture," *In Proc. Twelfth Annual Symposium*, Toronto, 1998.

[2] N. Roussopoulos and D. Leifker, "Direct Spatial Search on Pictorial Databases Using Packed R-trees," *ACM SIGMOD Record archive*, Vol. 14, Issue 4 table of contents, pp. 17-31, 1985.

[3] E. Sperley, *The Enterprise Data Warehouse: Planning, Building and Implementation*, Prentice Hall PTR, Indiana. USA, pp. 88-115, 1999.

[4] S. Chaudhuri and U. Dayal, "An Overview of Data Warehousing and OLAP Technology," *ACM SIGMOD Record*, Vol. 26, No. 1, pp. 65-74, 1997.

[5] W. H. Inmon, "What is a Data Warehouse?," *Prism Solutions Tech Topics*, Vol. 1, No. 1, pp. 32-37, 1995.

[6] W. H. Inmon, *Building the Data Warehouse*, 2nd Ed. John Wiley & Sons; Bk&CD Rom edition, Canada, pp. 66-67, 1996.

[7] R. Kimball, *Data Warehouse ToolKit*, John Wiley & Sons, New York, 1996.

[8] R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite, *The Data Warehouse Lifecycle Toolkit*, Robert Ipsen, U.S.A, pp. 18-21, 1998.

[9] N. Stefanovic, J. Han, and K. Koperski, "Object-Based Selective Materialization for Efficient Implementation of Spatial Data Cubes," *IEEE Transaction on Knowledge

and Data Engineering, pp. 938-958, 2000.

[10] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos, “Conceptual Modeling for ETL Processes,” *In Proc. of Data Warehousing and OLAP(DOLAP)*, pp. 14-21, 2002.

[11] G. Graefe, “Partitioned B-trees - a User's Guide,” *Datenbanksysteme für Business, Technologie und Web (BTW) 2003*, pp. 668-671, 2003.

[12] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M.Venkatrao, F. Pellow, and H. Pirahesh, “Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-tab, and Sub-total,” *Data-Mining and knowledge Discovery*, Vol. 1, No. 1, pp. 29-53, 1997.

[13] N. Roussopoulos, Y. Kotidis, and M. Roussopoulos, “Cubetree: Organization of and Bulk Incremental Updates on The Data Cube,” *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, Vol. 26, Issue 2 table of contents pp. 89-99, 1997.

[14] Y. Zhao, P. Deshpande, J. F. Naughton, “An Array Based Algorithm for Simultaneous Multidimensional Aggregates,” *ACM SIGMOD Record archive*, Volume 26, Issue 2 table of contents, pp. 159-170, 1997.

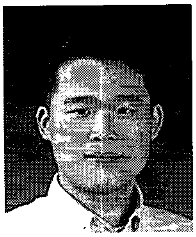
[15] J. Han, J. Pei, G. Dong, and K. Wang. “Efficient Computation of Iceberg Cubes with Complex Measures,” *In Proc. (2001) ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'01)*, Santa Barbara, CA, 2001.

[16] 김형선, 유병섭, 박순영, 이재동, 배해영, “공간 데이터 웨어하우스 구축기에서 추출된 데이터의 효율적인 적재를 위한 테이블 단위의 데이터 관리 기법,” 한국정보과학회:학술대회지, 2005년도 한국컴퓨터종합학술대회 논문집(B), pp. 79-81, 2005.

[17] 김형선, 유병섭, 이재동, 배해영, “데이터 웨어하우스에서 해쉬 테이블을 이용한 효율적인 데이터큐브 생성 기법,” 한국정보과학회:학술대회지 , 한국정보과학회 05 추계 학술발표논문집(2), pp. 211-213, 2005.

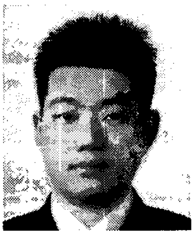
이 연

2006년 중국 중경우전대학교 지리정보공학과(학사)
2006년~현재 인하대학교 컴퓨터정보공학과(석사)
관심분야 : 공간데이터베이스, 공간 데이터웨어하우스, 지리정보시스템



김 형 선

2004년 청주대학교 컴퓨터정보공학과(학사)
2006년 인하대학교 컴퓨터정보공학과(석사)
2006년~현재 (주)카이네스 GIS 공학연구소 연구원
관심분야 : 공간 데이터웨어하우스, 데이터 마이닝



유 병 섭

2002년 인하대학교 컴퓨터공학부(공학사)
2004년 인하대학교 컴퓨터공학부(공학석사)
2004년~현재 인하대학교 대학원 컴퓨터정보공학과(박사과정)
관심분야 : 공간데이터베이스, 공간 데이터 웨어하우스, Data Stream, 유비쿼터스 컴퓨팅

이 재 동

1985년 인하대학교 전자계산학과 학사
1991년 미국 Cleveland State Univ., Dept. of Computer & Information Science(M.S.)
1996년 미국 Kent State Univ., Dept. of Computer Science (Ph.D.)
1996년~1997년 (주)두루넷 기술기획팀 팀장
1997년~현재 단국대학교 정보컴퓨터학부 컴퓨터과학 교수

2004년~2006년 단국대학교 정보통신원장 (CIO)
2002년~현재 농협중앙회 전산고문
2004년~2006년 전국대학정보화협의회 이사
2006년~현재 민관확대 콘텐츠 정책협의회 위원
관심분야 : Contents Technology, High Performance Computing(Clustering systems etc.), GIS Technologies and Applications, Many aspects of parallel/distributed processing

배 해 영

1974년 인하대학교 공학사(응용물리학)
1978년 연세대학교 공학석사(전자계산학)
1990년 숭실대학교 공학박사 (전자계산학)
1992~1994년 인하대학교 전자계산소 소장.
1982년~현재 인하대학교 컴퓨터공학부 교수.
1999년~현재 지능형GIS연구센터 센터장.
2000년~현재 중국 중경우전대학교 대학원 명예교수.
2004년~2006년 인하대학교 정보통신대학원 원장.
2006년~현재 인하대학교 대학원 원장.
관심분야 : 분산 데이타베이스, 공간 데이타베이스, 지리정보 시스템, 멀티미디어 데이타베이스 등