ORDMS 기반 데이터 웨어하우스에서 효율적인 질의 처리를 위한 AH 인덱스†

장혜경[1] 이정남[2] 조완섭[3] 이충세[1] 김홍기[1]
[1]충북대학교 전자계산학과, [2]한국컴퓨터통신, [3]충북대학교 경영정보학과

The AH Index for Efficient Query Processing in ORDBMS-based Data Warehouses

Hye-Kyoung Jahng[1], Jeong-Nam Lee[2], Wan-Sup Cho[3], Chung-Sei Rhee[1], Hong-Ki Kim[1]
[1]Dept. of Computer Science, ChungBuk Nat'l Univ., [2]KCOM, [3]Dept. of MIS, ChungBuk Nat'l Univ.

요 약

본 논문에서는 차세대 DBMS로 각광을 받고 있는 객체-관계형 DBMS(Object-Relational DBMS : ORDBMS) 기반의 데이터 웨어하우스(data warehouse)에서 질의 처리의 성능을 향상시키는 AH(Attribute Hierarchy) 인덱스와 이를 이용한 질의 처리 기법을 제안한다. 지금까지 관계 DBMS를 이용한 데이터 웨어하우스의 성능 향상에 관한 연구는 활발히 이루어져 왔으나, ORDBMS에 기반한 데이터 웨어하우스의 구축 및 질의 처리 성능에 관한 연구는 거의 이루어지지 않고 있다. 데이터 웨어하우스는 기존의 데이터베이스와는 비교할 수 없을 만큼의 대용량 데이터를 가정하므로 ORDBMS를 이용하여 데이터 웨어하우스를 구축하는 경우에서도 적절한 성능의 보장이 필수적으로 요구된다. 이 논문에서 제안된 AH 인덱스를 사용함으로써 데이터 웨어하우스 분석용 질의에서 자주 사용되는 조인과 그루핑 연산은 비용이 저렴한 인덱스 액세스 연산으로 대치되며, 데이터의 량과 무관하게 질의 처리비용이 거의 고정되는 효과를 얻을 수 있다.

1. 서론

데이터 웨어하우스(Data Warehouse)는 기업의 의사 결정을 지원하기 위한 주제 중심적이고, 통합적이며, 시간성을 가지는 비휘발성 자료의 집합이다[1,3,5,10,12]. 최근들어 데이터베이스 구축이 보편화되고, 데이터베이스에 수집된 자료가 급증함에 따라서 수집된 자료를 가공하고 분석하여 조직의 의사 결정에 활용할 수 있도록 지원하는 데이터 웨어하우스의 구축이 활발하다.

데이터 웨어하우스는 사용자 질의에 대하여 최적의 성능을 발휘할 수 있도록 관계 혹은 다차원 DBMS를 사용하여 스타 스키마나 스노우 플레이크 스키마 형태로 구축하며, 비트맵 인덱스(bit-map index)를 주로 사용한다[1,3]. 지금까지의 데이터 웨어하우스 구축은 관계 DBMS나 다차원 DBMS를 사용하여 이루어져 왔으나 최근들어 객체-관계 DBMS를 사용하는 연구가 진행되고 있다[8]. 그러나, 상용 ORDBMS를 이용하여 데이터 웨어하우스를 구축한 결과 ORDBMS는 데이터 모델링의 측면에서는 기존의 관계 DBMS보다 우수하지만, 성능의 측면에서는 여러 가지 문제점이 존재한다[13].

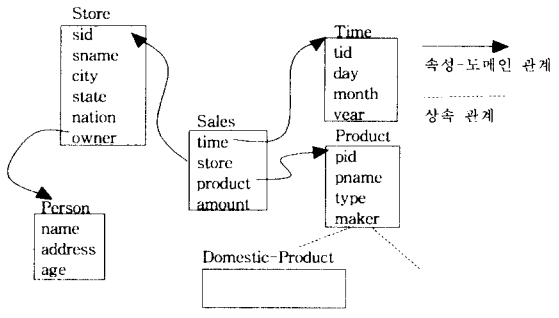
본 논문에서는 객체-관계 DBMS를 이용한 데이터 웨어하우스의 구축에서 분석용 질의의 처리 성능을 증대시키는 새로운 인덱스인 AH (Attribute Hierarchy) 인덱스를 제안하고, 이를 이용하여 OLAP 질의를 처리하는 방법을 제시한다. 제안된 AH 인덱스는 차원 테이블에서 속성들 사이에 계층 관계가 존재하는 경우에 생성하며, 공간 데이터베이스에서 널리 사용되는 R-Tree 인덱스와 객체-관계 데이터베이스에서 제안된 경로 인덱스의 장점을 결합한 구조이다. AH 인덱스는 경로식에 대한 조건들을 가진 Where 절의 처리뿐 아니라 그루핑이나 roll-up 및 drill-down 연산자의 처리에 효율적으로 이용될 수 있다.

이 논문의 구성은 다음과 같다. 제 2장에서는 관련 연구로써 객체-관계 모델을 기반으로 하는 데이터 웨어하우스 구축과 OLAP 질의어를 소개한다. 제 3 장에서는 데이터 웨어하우스에서 질의 처리의 성능 향상을 위한 새로운 인덱스를 제안하고, 제 4장에서 AH 인덱스를 이용한 질의 처리 기법을 설명한다. 제 5 장에서는 결론 및 향후 연구 과제를 기술한다.

2. 관련 연구

이 장에서는 데이터 웨어하우스의 구조를 객체-관계 스타 스키마 형태로 모델링하고, 객체-관계형 OLAP 질의를 기술한다.

그림 1에서 sales는 사실 클래스이고, time, store, product는 각각 차원 클래스이다. 사실 클래스와 차원 클래스간의 연결선은 애트리뷰트-도메인 관계를 나타낸다. 객체-관계형 스타 스키마에서는 외래키-주키가 아니라 경로식(path expression)을 사용하여 암시적 조인이 이루어지고 다차원적 질의가 가능하게 된다. 또한, 객체-관계형 모델링이므로 각 클래스는 서브 클래스(subclass)를 가질 수 있고, 각 차원 클래스의 애트리뷰트도 사용자 정의 클래스를 도메인으로 가질 수 있다[14]. 그림 1에서 Product 클래스와 Store 클래스는 서브 클래스를 포함하고 있고, Store 클래스의 owner 속성은 도메인으로써 Person 클래스가 지정된 경우를 보여준다.



<그림 1> 객체-관계형 스타 스키마

그림 1의 객체-관계형 스타 스키마에 대하여 객체-관계 질의어를 사용하여 다양한 분석용 질의를 할 수 있다.

(질의 1) 5 월에 청주에서 판매된 상품을 제조회사별로, 상품이름 별로 총액을 구하라.

```
SELECT s.product.maker, s.product.pname, sum(amount)
 FROM  Sales s
WHERE  s.time.month=5 AND s.store.city='청주'
 GROUP BY  s.product.maker, s.product.pname;
```

반면, 질의 1을 관계형 OLAP 질의로 작성하면 다음과 같이 표현된다.

(질의 2)

```
 SELECT  maker, name, sum(amount)
 FROM    Sales, Product, Store, Time
 WHERE   Sales.pid=Product.pid AND Sales.sid=Store.sid
 AND     Sales.tid =Time.tid AND Time.month=5 and Store.city='청주'
 GROUP BY  Product.maker, Product.name;
```

† 본 연구는 첨단정보기술연구센터를 통하여 과학재단의 지원을 받았음.

질의 1과 질의 2는 동일한 의미를 갖지만 질의 1은 ORDB에서의 경로식을 이용하기 때문에 명시적 조인조건이 필요하지 않다. 따라서 질의 2에 비하여 간단하고 직관적이다. 질의 1의 FROM 절에는 사실 클래스만 명시하며, WHERE 절에서 애트리뷰트-도메인 관계의 경로식(path expression)을 사용하여 각 차원 클래스와 조인을 수행한다. GROUP BY 절과 SELECT 절에도 길이가 2 이상인 경로식을 사용하여 그루핑과 프로젝션을 간단하게 명시할 수 있다.

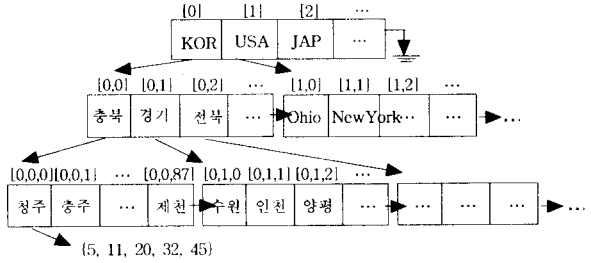
3. AH 인덱스

본 장에서는 ORDBMS를 이용한 데이터 웨어하우스 구축에서 유용한 AH (Attribute-Hierarchy) 인덱스를 제안하고, 탐색과 삽입 방법을 설명한다.

3.1 AH 인덱스의 구조

AH 인덱스는 데이터 웨어하우스에서 자주 사용되는 집계 연산 및 roll-up/drill-down 연산이 포함되어 있는 질의처리의 비용을 효과적으로 감소시켜주는 인덱스로써, 클래스 내에 존재하는 애트리뷰트 간의 계층 관계를 이용하여 구축한다. 예를들면, 그림 1의 스타 스키마에서 store 클래스의 속성인 nation, state, city는 계층을 이루고 있다. AH 인덱스는 각 레벨마다 다른 키 값을 갖으며, B^+-tree처럼 키 값을 비교해서 인덱스를 탐색하는 것이 아니라 인덱스의 각 노드에 논리적인 ID를 부여하여, 이를 비교함으로써 인덱스를 검색하게 된다. 트리의 레벨 수는 계층을 이루는 애트리뷰트들의 개수와 동일하다. 각 레벨에서 노드들은 클러스터링 되어 있고, fan-out의 수는 상위 레벨의 value 개수와 같다.

그림 2는 store 클래스의 애트리뷰트인 nation, state, city의 계층관계를 이용한 AH 인덱스의 한 예이다. 인덱스의 첫 번째 레벨인 nation의 노드의 개수는 국가의 수 즉, 애트리뷰트 nation의 value의 개수(cardinality)와 같다. 이때 state 레벨의 노드들은 부모 노드인 nation의 노드의 Key_ID를 노드 자신의 Key_ID에 포함시킨다(예: 충북의 Key_ID=[0,0]에서 앞에 있는 '0'은 'KOR'의 Key_ID이고, 뒤의 '0'은 '충북' 노드 자신의 Key_ID이다). 다음으로 city 레벨은 단말노드이므로 실제 데이터베이스에서 사실테이블에 대한 객체 ID를 가리키고 있다(예: {5, 11, 20, 32, 45}는 청주 노드가 가리키는 사실 클래스에 대한 객체 ID의 집합이다).



<그림 2> AH 인덱스의 구조

단말 노드(leaf_node)는 Key_ID, Key, pointer를 포함하고 있다. Key_ID는 각 노드에 대한 논리적 identifier이고, Key는 각 노드에 대한 서술 정보로써 '충북' 또는 '청주'와 같은 실제 값을 말하는 것이고, pointer는 실제 데이터베이스의 사실 클래스에 있는 객체의 주소를 저장하고 있는 기억 장소이다. 단말노드의 Key_ID는 애트리뷰트 계층에서 자신의 모든 부모노드의 Key_ID를 순차적으로 저장한 배열의 구조-배열의 차원은 레벨의 수에 따라서 정해진다-를 띠고 있다. 인덱스의 각 레벨을 h라고 하고, 루트 레벨을 0이라고 한다면 레벨마다 갖는 Key-ID는 h+1 차원의 배열로 표현될 수 있다. 예를 들어, 그림 3을 보면 루트 노드인 KOR 의 Key-ID는 [0]으로써 1차원 배열로, KOR의 자식노드인 '충북'은 두 개의 정수값을 갖는 2차원 배열 [0,0]으로, '청주'의 Key_ID는 {0,0,0}으로 표현되었다. 단말노드인 '청주'의 Key_ID에서 첫 번째 '0'은 루트노드 KOR 필드의 Key_ID [0]을, 두 번째 '0'은 두 번째 레벨에서 '충북'의 Key_ID [0,0] 중 뒷자리 '0'으로 두 번째 레벨의 첫 번째 노드를 나타낸다. 마지막 '0'은 '청주'가 세 번째 레벨의 첫 번째 항목이라는 의미이다.

단말노드를 제외한 나머지 노드들(non leaf_node)은 Key_ID, Key, childptr 인덱스 레코드 엔트리를 포함하고 있다. Key_ID는 각 노드의 논리적 ID를, childptr는 하위 노드의 첫 번째 노드에 대한 포인터를 가리킨다.

3.2 탐색과 갱신

데이터 웨어하우스에서 튜플의 삭제는 자주 일어나지 않는다. 또, 데이터 웨어하우스에 있는 데이터들은 사용자들의 예측 분석을 돕기 위하여 과거의 기록까지 모두 보존을 하는 역사성을 갖기 때문에 튜플의 갱신도 거의 일어나지 않는다. 따라서 이 절에서는 AH 인덱스에서 탐색과 삽입이 어떻게 진행되는가에 대해서만 설명하며, 상세한 내용은 참고문헌[14]에 있다.

(가) 탐색

AH 인덱스는 'read-oriented' 특징을 갖는 데이터 웨어하우스를 위해 제안된 인덱스이다. 특히, group by절의 조건을 처리하기 위한 인덱스로서 애트리뷰트의 계층 관계를 이용해서 집계(aggregate) 연산을 빠르게 처리한다. AH 인덱스의 탐색 알고리즘은 B^+-tree와 달리 탐색키를 비교해서 트리를 탐색하는 것이 아니라, KeyID를 비교해서 원하는 객체 ID를 찾는 것이다. where 절의 선택 조건을 처리할 경우에는, 탐색 알고리즘을 실행하기 전에 B^+-tree를 이용해서 키값에 해당하는 튜플에서 KeyID를 찾는다. 찾은 KeyID를 이용해서 AH 인덱스의 탐색 알고리즘을 실행하게 된다. 예를 들면, 그림 2에서 키값 '충북'에 해당하는 사실 클래스의 객체 ID를 찾기 위해서는 먼저 차원 테이블에서 키값 '충북'의 KeyID를 찾아서 인덱스를 이용한다. 인덱스가 group by 절의 경로식을 처리하는데 이용될 경우에는 모든 단말 노드를 찾아가는 모든 탐색 경로를 찾아서 사실 클래스의 객체 ID를 읽으면 된다. 탐색 알고리즘을 실행하는데 있어서 다음과 같은 테이블이 필요하다. 다음의 각 테이블들은 AH 인덱스를 탐색하는데 필요한 각 레벨의 Key에 대한 Key_ID를 저장한 것이다.

KeyID_3[ ] : 세 번째 city Level에 Key 대한 KeyID

Key	KeyID
청주	[0,0,0]
충주	[0,0,1]
...	...
수원	[0,1,0]
인천	[0,1,1]
...	...

KeyID_2[ ]: 두 번째 state Level 의 Key에 대한 KeyID

Key	KeyID
충북	[0,0]
경기	[0,1]
전북	[0,2]
...	...
Ohio	[1,0]
...	...

KeyID_1[ ] : 첫 번째 nation Level의 Key에 대한 KeyID

Key	KeyID
KOR	[0]
USA	[1]
JPN	[2]
...	...

<그림 3> 그림 2의 AH 인덱스를 위한 KeyID 구성

(나) 삽입

데이터 웨어하우스에서 삽입이 빈번하게 일어나는 클래스는 사실 클래스이다. 사실 클래스에 새로운 객체의 생성이 발생하면, 우선 각 레벨의 Key에 대한 KeyID를 저장한 테이블(그림 3)에서 속성 값의 KeyID를 검색하고, 인덱스의 계층의 경로(path)를 따라서 사실 클래스에 삽입된 객체의 OID를 포인터 필드에 삽입한다.

4. AH 인덱스를 이용한 DW의 질의 처리

여기서는 3 장에서 제안된 AH 인덱스를 사용하여 OLAP 질의를 처리하는 방법을 기술한다.

4.1 질의 처리기법

AH 인덱스는 클래스 내에 존재하는 애트리뷰트 간의 계층 관계를 이용한 인덱스이다. 이 인덱스는 집계 연산을 빠르게 수행하는 것이 목적이지만, roll-up과 drill-down 연산의 수행에도 이용 가능하다. 애트리뷰트간의 계층 관계가 roll-up이나 drill-down 연산의 기준이 되기 때문이다. 이런 연산들은 SQL의 group by 절에 의해 수행된다. Q1은 집계 연산의 일반적 형태를 나타내는 질의문이다. 질의문에서 group by 절의 경로는 차원 클래스의 계층을 이루는 애트리뷰트가 올 경우에 하나의 AH 인덱스를 이용해서 처리 가능하다.

(Q1) SELECT $spath_1, spath_2, \ldots, spath_{n2}$
FROM C
WHERE $pred_1 \wedge pred_2 \wedge, \ldots, pred_n$
GROUP BY $gpath_1, gpath_2, \ldots, gpath_{n1}$

※ 질의 처리 알고리즘

단계1. WHERE 절에서 AH 인덱스와 일치하는 경로들의 조건들을 처리해서 조건을 만족하는 객체들의 OID 집합을 구한다.

단계2. 일치하는 술어의 개수가 2개 이상인 경우는 술어들을 연결한 부울 연산자가 AND 이면 OID intersection을, OR이면 OID union을 시행하여 인덱스로 처리 가능한 술어들을 만족하는 OID 집합을 구한다 : OID-IQ

단계3. OID-IQ 집합에 속하는 OID들을 데이터베이스로부터 하나씩 메모리로 가져와서 인덱스로 처리할 수 없는 질의 조건을 처리한다 : Object-Q

단계4. group by절의 경로와 일치하는 AH 인덱스를 이용해서 grouping 연산을 실행한다.

- group by 절의 모든 경로들의 마지막 속성값이 AH 인덱스가 구축된 계층을 이루는 애트리뷰트일 경우에 단계2의 OID_IQ집합과 AH

인덱스만 이용해서 처리한다.
- 인덱스가 없는 나머지 경로들은 단계2에서 생성된 OID_IQ 집합의 OID를 사용해서 데이터베이스에 직접 접근하여 grouping 연산을 처리한다.

단계5. 각 그룹별로 select_list에 나타난 집계연산을 실행한다.

여기에서는 AH 인덱스가 유용한 질의 형태를 제시하고, 이 경우에 질의 처리 과정을 설명한다.

(가) where절이 없고 group by절만 있을 경우

(Q2) SELECT spath$_1$, spath$_2$,...,spath$_{n2}$,
FROM C
GROUP BY gah_path$_1$, gah_path$_2$, ..., gah_path$_n$

Q2와 같이 where 절의 조건이 없는 경우에는 AH 인덱스와 일치하는 group by절의 경로를 처리한다. 이때 group by절의 경로가 AH 인덱스의 각 레벨순서와 일치할 경우에 하나의 AH 인덱스만을 이용해서 group by 절을 처리가 가능하므로 질의 처리비용이 매우 적게 드는 경우에 해당한다.

(나) where절만 있을 경우

(Q3) SELECT spath$_1$, spath$_2$,...,spath$_{n2}$,
FROM C
WHERE pred$_1$∧pred$_2$∧...pred$_n$

Q3와 같이 group by 절이 없을 경우에는 where 절의 predicate을 처리하는데 (1)AH 인덱스를 이용하는 경우와 (2)다른 인덱스가 존재할 경우에 해당 인덱스를 사용하는 경우를 고려해야 한다. 이때는 질의 최적화 과정에서 두 가지 경우의 질의 처리비용을 예측한 후 더 적은 것을 선택해서 처리한다.

(다) where 절과 group by 절이 모두 있을 경우

(Q4) SELECT spath$_1$, spath$_2$,...,spath$_{n2}$,
FROM C
WHERE pred$_1$∧pred$_2$∧...pred$_n$
GROUP BY gpath$_1$, gpath$_2$, ..., gpath$_k$

Q4과 같은 경우는 where 절의 predicate와 group by절의 경로에 같은 AH 인덱스의 적용가능 여부를 결정해야 한다. where 절의 predicate과 group by 절의 경로에 적용 가능한 AH 인덱스가 존재할 경우에 하나의 AH 인덱스만을 이용해서 where 절과 group by 절을 모두 처리 가능하다.

(다-1) group by 절에 계층을 갖는 애트리뷰트에 대한 경로가 두 개 이상 있을 경우

(Q5) SELECT spath$_1$, spath$_2$,...,spath$_{n2}$,
FROM C
WHERE pred$_1$∧pred$_2$∧...pred$_n$
GROUP BY gpath$_1$, gpath$_2$, ..., gpath$_k$
(Q5') SELECT spath$_1$, spath$_2$,...,spath$_{n2}$,
FROM C
WHERE pred$_1$∧pred$_2$∧...pred$_n$
GROUP BY gpath$_1$, gpath$_2$, ..., gpath$_k$, gpath$_{k+1}$,...,gpath$_{n1}$

Q5 질의 후 drill-down 해서 Q5'을 질의했다고 가정하자. 이 경우에도 AH 인덱스는 유용하게 사용된다. Q5'의 group by 절의 gpath$_1$~gpath$_{n1}$에 AH 인덱스가 구축되어 있다고 가정하자. gpath$_{k+1}$~gpath$_{n1}$으로 drill-down 하기 위해서 AH 인덱스의 level$_{k+1}$에서 level$_{n1}$을 읽어서 OID별로 그루핑한 후 각 그룹별로 집계함수를 실행하면 되므로 AH 인덱스의 일부(level$_{k+1}$~level$_{n1}$)를 읽는 비용이 소요된다. AH 인덱스가 없으면 전체 질의를 다시 실행해야 하므로 drill-down 비용이 절감됨을 알 수 있다. 유사하게 roll-up의 경우(Q5' 질의 후 Q5의 질의)도 drill-down과 반대의 경우로 생각해서 AH 인덱스로 처리할 수 있다.

Q5의 group by 절의 gpath$_1$~gpath$_n$ 과 gpath$_{n1}$~gpath$_{n2}$ 는 각각 계층을 이루는 애트리뷰트의 경로에 해당한다. 각각의 경로와 일치하는 AH 인덱스가 존재하면 인덱스를 이용해서 처리한다. where 절의 predicate은 일치하는 인덱스가 있으면 처리한다.

(다-2) group by 절에 계층을 갖는 애트리뷰트들과 계층을 이루지 않는 애트리뷰트가 함께 있을 경우

(Q6) SELECT spath$_1$, spath$_2$,...,spath$_{n2}$,
FROM C
WHERE pred$_1$∧pred$_2$∧...pred$_n$
GROUP BY gah_path$_1$, gah_path$_2$, ..., gah_path$_n$,...,
gpath$_{n1}$,...,gpath$_{n2}$

위와 같은 질의 형태는 group by 절에 계층을 이루는 애트리뷰트에 대한 경로식과 계층을 이루지 않는 경로식이 혼합된 경우의 예이다. gah_path$_1$~gah_path$_n$ 는 계층을 애트리뷰트들에 대한 경로식이고, gpath$_{n1}$~gpath$_{n2}$ 는 계층을 이루지 않는 애트리뷰트들에 대한 경로식이다. 먼저 group by 절의 경로식에 이용 가능한 AH 인덱스가 있는지를 조사한다. AH 인덱스가 존재할 경우에 해당 인덱스가 where 절의 조건에도 이용가능한지 검사한다. where 절과 group by 절에 동시에 적용 가능한 AH 인덱스가 존재하면, 선택된 AH 인덱스를 이용해서 where 절의 조건과 group by 절의 질의를 처리한다. 다음에 계층을 이루지 않는 애트리뷰트에 대한 경로식을 그루핑한다.

4.2 AH 인덱스의 특성분석

4.1 절의 질의 처리 기법 논의를 바탕으로 AH 인덱스의 특징을 살펴보면 다음과 같다.

1. where 절과 group by 절에서 모두 이용 가능하다.
-AH 인덱스를 where 절에서만 사용하거나 group by절 절에서만 사용할 수도 있으며,
-하나의 AH 인덱스가 where 절과 group by 절에 동시에 사용될 수 있다.
2. 데이터 웨어하우스가 읽기 위주이므로 AH 인덱스 자체도 갱신이 거의 발생하지 않으며, 경로에 대한 인덱스 유지비용이 적렴하다.
3. 하나의 AH 인덱스(level=k)가 기존의 k개의 인덱스의 역할을 한다.
5. OODB에서 Nested 인덱스의 성질을 그대로 가지므로 경로 탐색 비용(조인 비용)을 줄여준다.

5. 결론 및 향후 연구

본 연구에서는 객체-관계 DBMS기반의 데이터 웨어하우스의 질의 처리 성능을 향상시키기 위한 새로운 인덱스를 제안하고, 제안된 인덱스를 사용하여 질의를 처리하는 기법을 설명했다. 제안된 인덱스는 공간 데이터베이스에서 널리 사용되는 R-Tree 인덱스와 객체-관계 데이터베이스에서 제안된 경로 인덱스(path index)의 장점을 모두 가진다. 제안된 인덱스를 사용함으로써 고비용으로 알려진 where 절에 포함된 조인 조건과 group by 절에 포함된 경로들을 저비용인 인덱스 액세스로 처리할 수 있으므로 질의 처리의 성능이 개선된다. 특히, 데이터 웨어하우스는 사실 클래스나 차원 클래스에 대한 삭제나 변경 연산이 거의 발생하지 하지 않고, 사실 클래스에 대한 삽입 연산만 주로 발생하므로, AH 인덱스가 객체-관계 데이터베이스에서의 경로 인덱스 성질을 그대로 가짐에도 불구하고 인덱스 유지 비용이 저렴하다.

향후 연구로써 제안된 인덱싱 기법으로 인한 질의 처리 성능의 향상 정도를 수학적인 모델링으로 정확히 분석하는 작업과 기존에 주로 이용되고 있는 인덱스들과 AH 인덱스의 성능을 비교 분석하는 작업이 필요하다.

[참고 문헌]

[1] Lyman Do et al., "Issues in Developing Very Large Data Warehouses," In Proc. Int'l Conf. on VLDB, 1998.

[2] Wilburt Labio et al., Jennifer Widom, "The WHIPS Prototype for Data Warehouse Creation and Maintenance," In Proc. Int'l Conf. ACM SIGMOD, 1997.

[3] Wu, M. C. and Buchmann, A.P., "Research Issues in Data Warehousing," Submitted for publication.

[4] Wu, M. C. and Buchmann, Alejandro P., "Encoded Bitmap Indexing for Data Warehouses," In Proc. Int'l Conf. ICDE, 1998.

[5] Inmon, W.H., Building the Data Warehouse. John Wiley, 1992.

[6] Surajit Chaudhuri and Umeshwar Dayal, "Data Warehousing and OLAP for Decision Support," In Proc. Intl. Conf. DOOD, 1997.

[7] George Colliat, "OLAP, Relational, and Multidimensional Database System," In ACM SIGMOD Record, 25(3): 1996.

[8] Yihong Zhao et al., "Array-Based Evaluation of Multi-Dimensional Queries in Object-Relational Databases System," In Proc. Int'l Conf ICDE, 1998.

[9] Prasad Deshpande et al., "Cubing Algorithms, Storage Estimation, and Storage and Processing Alternatives for OLAP," IEEE Data Engineering Bulletin 20(1): 3-11, 1997.

[10] Kimball, R. The Data Warehouse Toolkit. John Wiley, 1996.

[11] OLAP Council White Paper, http://www.olapcouncil.org

[12] Stanford Technology Group, "Designing the Data Warehouse On Relational Database," White Paper.

[13] Bertino, E. and Kim, W. "An indexing technique for query on nested object," IEEE Trans. on Knowledge and Data Engineering, Vol, No2(1989), 196-214.

[14] 이정남, ORDBMS를 이용한 데이터 웨어하우스에서 효율적인 질의 처리를 위한 인덱싱 기법, 석사학위논문, 충북대학교, 2000.