객체 관계형 데이터베이스 환경에 적합한 공간 색인 통합 방법의 비교 연구

이민우, 박수홍

인하대학교 지리정보공학과
lmana4@hotmail.com, shpark@inha.ac.kr

A Comparative study on the integration method of spatial index in ORDBMS

Min Woo Lee, Soo Hong Park

요 약

최근 공간 데이터를 사용하는 응용 프로그램이 증가하면서 대용량의 공간 데이터를 효율적으로 저장하고 관리하기 위한 공간 데이터베이스가 요구되고 있다. 이러한 공간 데이터베이스는 객체 관계형 데이터베이스의 사용자 정의 타입과 사용자 정의 함수를 이용하여 기존의 데이터베이스를 확장하는 형태로 개발될 수 있다. 하지만, 대부분의 객체 관계형 데이터베이스는 공간 인덱스와 같은 사용자 정의 인덱스를 확장하는 일반적인 방법을 제공하고 있지 않기 때문에 객체 관계형 데이터베이스를 확장한 공간 데이터베이스는 공간 영역 질의의 성능이 떨어지는 문제점이 있다. 본 연구에서는 객체 관계형 데이터베이스를 확장한 공간 데이터베이스에서 공간 인덱스를 개발하고 객체 관계형 데이터베이스에 통합시킬 수 있는 방법인 GiST와 Relational Indexing을 비교/분석하고 향후 이들 방법을 이용하여 공간 인덱스를 구현하고 공간 영역 질의에 대한 성능을 비교하여 보다 적합한 방법을 제시하고자 한다.

1. 서 론

1.1 연구 배경과 목적

최근 GIS, LBS, ITS와 같이 공간 데이터를 사용하는 응용 시스템들이 급격하게 늘어남에 따라서 공간 데이터를 효율적으로 저장하고 관리할 수 있는 공간 데이터베이스가 요구되고 있다. 이러한 공간 데이터베이스는 공간 데이터만을 지원하기 위해서 새롭게 개발되는 경우는 극히 드물며, 대부분은 관계형 데이터베이스나 객체 관계형

데이터베이스, 객체 지향형 데이터베이스와 같은 기존의 데이터베이스를 확장하는 형태로 개발되고 있다.

특히, 객체 관계형 데이터베이스를 확장하는 방법은 객체 관계형 데이터베이스의 특징인 사용자 정의 타입과 사용자 정의 함수를 이용하여 쉽게 공간 데이터와 공간 연산자를 구현할 수 있고, 문자나 숫자와 같은 비공간적인 데이터를 데이터베이스의 기본 데이터 타입으로 저장할 수 있기 때문에 많이 사용되고 있다. 하지만, 대부분의 객체 관계형 데이터베이스는 사용자 정의 인덱스를 생성하는 방법을 제공하고 있지 않으며, 제공한다 하더라도 오라클의 Catridge나 DB2의 Data Blade와 같이 특정 업체에 종속적인 형태로 서로 다르게 제공하기 때문에 공간 인덱스와 같은 사용자 정의 인덱스를 구현하기가 매우 어려운 실정이다. 따라서 객체 관계형 데이터베이스에서 사용자 정의 인덱스를 개발할 수 있는 보다 일반적인 인덱스 확장 방법이 요구되고 있는데 현재 제안되고 있는 대표적인 인덱스 확장 방법으로는 GiST와 Relational Indexing이 있다. 하지만 아직까지 두 방법에 대한 비교/분석 및 성능 평가는 미흡한 실정이다.

본 연구의 목적은 객체 관계형 데이터베이스를 확장한 공간 데이터베이스에서 공간 인덱스를 개발하고 데이터베이스에 통합시킬 수 있는 방법인 GiST와 Relational Indexing을 비교/분석하고 향후 공간 인덱스 개발에 적합한 방법을 제시하는 것이다.

본 연구의 구성은 다음과 같다. 2장에서는 관련 연구에 대해서 살펴보고, 3장에서는 현재 제안되고 있는 사용자 정의 인덱스 확장 방법인 GiST와 Relational Indexing에 대해서 비교/분석을 하고, 4장에서는 이들 방법을 이용해서 공간 인덱스(R-Tree)를 구현하는 방법을 기술하고, 마지막으로 5장에서 향후 연구 방향에 대해 논의하고자 한다.

2. 관련 연구

현재 객체 관계형 데이터베이스에서 사용자 정의 인덱스를 데이터베이스 내부로 통합하려는 대표적인 해외 연구로는 "Generalized Search Tree for Database Systems"[1] (GiST)가 있다. 이 연구는 객체 관계형 데이터베이스에서 Tree 기반의 사용자 정의 인덱스를 데이터베이스 내부로 통합시키는 framework를 제안한 연구였다.

또 다른 해외 연구로는 "The Paradigm of Relational Indexing: A Survey"[2]가 있다. 이 연구는 객체 관계형 데이터베이스에 이미 존재하는 기법들을 이용하여 relation(테이블) 기반의 사용자 정의 인덱스를 개발하는 방법을 제안하였다.

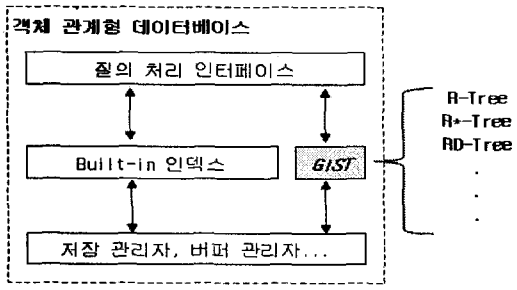
국내 관련 연구로는 "멀티미디어 데이터를 위한 확장형 인덱스 소개"[3]가 있다. 이 연구에서는 대표적인 상용 객체 관계형 데이터베이스인 오라클의 Catridge를 이용하여 사용자 정의 인덱스를 통합하는 방법에 대해서 소개하고 있다.

현재 특정 업체에 종속적이지 않으면서 객체 관계형 데이터베이스에 사용자 정의 인덱스를 개발하고 데이터베이스 내부로 통합시키는 일반적인 방법으로 GiST와 Relational Indexing이 제안되고 있는 실정이다.

3. 사용자 정의 인덱스의 통합 방법

3.1 GiST

GiST는 B-Tree 이후의 대부분의 Search Tree들이 B-Tree로부터 파생된 형태를 가지고 있다는 점에 착안하여 Search Tree로써 가지고 있어야 하는 기본적인 메소드와 사용자 정의 타입 및 사용자 정의 연산자를 수용할 수 있는 자료구조를 제공하여 Tree 기반의 사용자 정의 인덱스를 개발할 수 있게 해주는 Framework이다. 다음 [그림 1]은 객체 관계형 데이터베이스에서 GiST의 구조를 나타내고 있다.



[그림 1] ORDBMS에서 GiST

이러한 GiST는 3가지의 특징을 가진다.

첫 번째 특징은 질의의 확장성이다. 이는 인덱스를 사용하는 연산자를 사용자 임의대로 개발할 수 있다는 의미이다. 예를 들어, 데이터베이스에 이미 존재하는 B-Tree인 경우 인덱스를 실제 사용할 수 있는 연산자는 { >, =, <, !=, >=, <= }이 있으며 이들 연산자는 그 의미가 이미 정해져 있고 변경할 수 없는 연산자들이다. 하지만, GiST를 이용하여 B-Tree를 개발할 경우는 사용자 임의대로 연산자를 추가 및 변경할 수 있게 된다. 즉, "<" 연산자를 단순한 크기 비교가 아닌 전혀 다른 의미를 가지는 연산자로 변경할 수 있으며 "@"와 같은 새로운 연산자를 추가할 수도 있다는 것이다.

두 번째 특징은 탐색키의 일반화이다. 이는 GiST가 B-Tree와 같이 <key, pointer>를 쌍으로 가지는 자료구조를 가지지만 여기서 탐색키는 데이터베이스에 이미 존재하는 데이터 타입이 아닌 사용자 정의 타입을 가리킨다. 즉, 모든 사용자 정의 타입에 대해서 일반적이고 견고하게 인덱스를 확장할 수 있다는 것을 의미한다. 따라서 GiST를 이용하면 공간 인덱스뿐만 아니라 멀티미디어 데이터에 대한 인덱스도 가능할 수 있게 된다.

세 번째 특징은 일반화된 7개의 메소드이다. 이는 Tree를 구성하기 위해 기본적으로 필요한 메소드이며 데이터베이스 내부와 연결되어 있어 사용자에게는 데이터베이스 내부와 독립적으로 구현하고자 하는 인덱스 기법을 데이터베이스에 통합할 수 있게 한다. 다음 [표 1]은 7가지 메소드에 대한 간략한 설명을 나타내고 있다.

[표 1] GiST의 주요 메소드

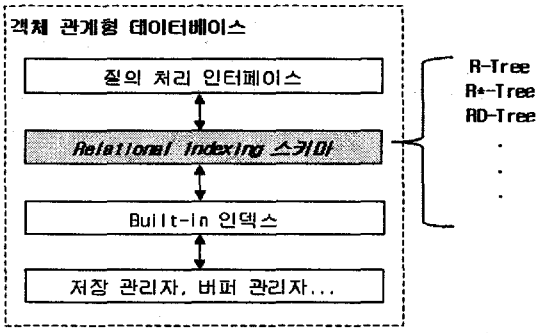
메소드	역 할
Consistent	인자로 들어온 연산자를 이용해서 현재 노드와의 연산수행의 참/거짓을 리턴
Union	인자로 들어온 엔트리들을 모두 통합하는 새로운 키를 리턴
Penalty	첫 번째 인자의 엔트리가 두 번째 인자의 엔트리로 삽입될 때 비용을 리턴
PickSplit	인자로 들어온 노드를 두 그룹으로 나눔
Compress	사용자 정의 타입으로부터 키를 추출
Decompress	키로부터 사용자 정의 타입을 복원
Equal	두 인자의 일치여부를 리턴

이러한 특징을 가지는 GiST는 현재 PostgreSQL 같은 공개형 객체 관계형 데이터베이스에 직접 내장된 형태로 사용되기도 하고 라이브러리 형태로 제공되어 오라클의 Catridge와 결합한 OraGiST와 같은 형태로 상용 객체 관계형 데이터베이스에 적용되고 있다.

3.2 Relational Indexing

Relational Indexing은 객체 관계형 데이터베이스에서 기본적으로 제공하는 기법들만을 이용하여 relation(테이블) 기반에서 사용자 정의 인덱스를 개발할 수 있는 일반적인 인덱스 스키마이다.

다음 [그림 2]는 객체 관계형 데이터베이스에서 Relation Indexing의 구조를 나타내고 있다.



[그림 2] ORDBMS에서 Relational Indexing

이러한 Relation Indexing이 제안하는 일반적인 인덱스 스키마는 3개의 테이블로 구성되며 사용자 테이블, 인덱스 테이블, 메타 테이블이 있다. 여기서 사용자 테이블은 튜플을 유일하게 식별할 수 있는 키와 사용자 정의 타입으로 구성되며 실제 데이터가 저장되는 테이블을 의미한다. 인덱스 테이블은 사용자 테이블의 튜플을 가리키는 고유한 키와 인덱스 되는 사용자 정의 타입에 대한 키로 구성되는 테이블이다. 메타 테이블은 인덱스 테이블 명과 인덱스 되는 사용자 테이블 명으로 구성되며 사용자 정의 인덱스에 대한 시스템 테이블과 같은 역할을 한다.

이와 같은 스키마를 이용하면 사용자 정의 인덱스를 데이터베이스의 내부 구조를 전혀 조작하지 않으면서 일반적인 절차적 SQL 언어를 사용하여 개발할 수 있게 된다.

3.3 비교 분석

이들 GiST와 Relational Indexing은 모두 특정 업체에 종속되지 않으며, 어떠한 객체 관계형 데이터베이스에도 적용될 수 있도록 제안된 사용자 정의 인덱스 확장 방법이다. 다음 [표 2]와 [표 3]은 GiST와 Relational Indexing을 비교하고 있다.

[표 2] GiST

GiST의 특징	
구 현	· DB 내부 구조와 연결하는 것이 어려움 · 사용자 정의 인덱스 구현은 쉬움
언 어	· C/C++만 가능
성 능	· Built-in 인덱스보다 떨어지지만 우수함
DB 수정	· GiST를 DB에 설치하기 위해서는 DB 내부를 수정 및 재 컴파일 필요 · 사용자 정의 인덱스 구현시에는 DB를 수정할 필요가 없음
인덱스기법	· 트리 기반의 모든 인덱스 가능
적 용 DB	· 객체 관계형 데이터베이스만 가능
활 용	· 공개형 데이터베이스만 적용 · 상업용 데이터베이스는 내부를 수정할 수 없기 때문에 오라클의 Catriadge와 같이 반드시 사용자 정의 인덱스를 제공하는 데이터베이스에만 적용이 가능함

[표 3] Relational Indexing

Relational Indexing의 특징	
구 현	· 구현이 쉬움
언 어	· 표준 SQL을 사용함
성 능	· 아직 검증되지 않음
DB 수정	· DB 내부 구조와 독립적이므로 수정할 필요가 전혀 없음
인덱스기법	· 트리, 비트리 기반의 모든 인덱스 가능
적 용 DB	· 관계형, 객체 관계형 데이터베이스에 가능
활 용	· 공개형, 상업용 모두 적용이 가능

두 방법을 비교하면 Relational Indexing이 GiST보다 구현 측면과 적용 가능한 데이터베이스 측면, 그리고 사용자 정의 인덱스 기법 측면에서 더 일반적이고 범용적인 것을 확인할 수 있다. 하지만, 성능 측면에서는 Relational Indexing의 경우 데이터베이스 내부의 운영 구조를 그대로 사용하면서 테이블을 사용하기 때문에 테이블에 대한 기본적인 트랜잭션과 locking에 대한 부하를 가질 수밖에 없으므로 이에 대한 성능 저하가 필연적이다. 반면, GiST는 이러한 사항을 데이터베이스의 내부 구조를 조작함으로써 데이터베이스의 Bulit-in 인덱스와 같은 구조를 형성하기 때문에 성능이 우수하다. 하지만 이 두 방법에 대한 실제적인 성능 평가는 이루어지지 미흡한 실정이다.

4. 공간 인덱스의 적용

GiST와 Relational Indexing은 사용자 정의 인덱스 개발을 위한 것으로써 객체 관계형 데이터베이스에서 확장된 공간 데이터베이스의 공간 인덱스(R-Tree)에도 적용이 가능하다.

4.1 GiST를 이용한 R-Tree

GiST를 이용한 R-Tree는 [표 1]의 7가지 메소드에 R-Tree 기법을 적용하여 구현이 가능하다. 공간 인덱스를 구현하기 위해 구현해야할 메소드는 다음과 같다.

공간 객체가 삽입/삭제/변경될 때 호출되는 Compress, Consistent, Union, PickSplit, Penalty, Equal 메소드가 그것이다.

Compress 메소드에서는 공간 객체를 인자로 받고 공간 객체로부터 MBR에 해당하는 키를 추출한 후 GiST 엔트리에 이 키를 저장하고 엔트리를 리턴한다.

Consistent 메소드는 첫 번째 인자로 들어온 GiST 엔트리로부터 MBR을 추출하고 또한 두 번째 인자로 들어오는 사용자 정의 함수를 이용해서 이 MBR이 현재 노드의 MBR과 사용자 정의 함수를 만족하는지 안하는지 여부를 리턴한다.

Union 메소드는 인자로 들어오는 엔트리들의 MBR을 모두 포함하는 MBR을 가지는 새로운 GiST 엔트리를 리턴한다.

PickSplit 메소드는 인자로 들어오는 GiST 엔트리를 R-Tree의 Split 알고리듬을 이용해 두 그룹으로 분할한다.

Penalty 메소드는 첫 번째 인자로 들어오는 엔트리가 두 번째 인자로 들어오는 엔트리에 더해질 때 MBR의 확장 영역의 값을 리턴한다.

Equal 메소드는 두 엔트리의 MBR이 같은지 확인한다.

탐색시 호출되는 메소드는 Consistent 메소드이며 앞에서 살펴본 것과 같이 사용자 정의 함수에 만족하는지 여부를 리턴한다.

이와 같이 기본적인 R-Tree 알고리듬을 GiST의 트리를 구성하기 위한 메소드에 적절히 구현을 해주면 데이터베이스 내부적인 운영체제와 상관없이 R-Tree를 공간 데이터베이스에 내장시킬 수 있다.

4.2 Relational Indexing을 이용한 R-Tree

Relational Indexing을 이용한 R-Tree는 사용자 테이블과 인덱스 테이블, 그리고 이들을 연결하는 메타 테이블을 생성해주면 된다.

사용자 테이블은 공간 객체를 유일하게 식별할 수 있는 식별자 ID와 실제 공간 객체의 필드로 구성된다.

인덱스 테이블은 노드 식별자 ID와 노드 레벨 ID, 그리고 노드의 자식 식별자 ID와 현재 노드의 MBR로 구성된다. 여기서, 인덱스 테이블과 사용자 테이블과의 연결은 인덱스 테이블의 노드 레벨 ID가 최하위 레벨인 노드의 자식 식별자 ID와 실제 사용자 테이블의 식별자 ID를 참조함으로써 가능하다.

메타 테이블은 사용자 테이블의 테이블 명과 인덱스 테이블의 테이블 명을 필드로 가지며 사용자 테이블의 인덱스 테이블에 대한 메타 정보를 제공하는 시스템 테이블과 같은 역할을 한다.

이러한 스키마에서 공간 객체가 삽입/삭제/변경될 때는 SQL의 트리거를 이용하여 트리를 구성하는 루틴을 작성하게 되는데, 이때 구현되는 모든 SQL 함수에 R-Tree 알고리듬을 적용하면 R-Tree를 SQL 수준에서 공간 데이터베이스에 내장시킬 수 있게 된다.

5. 결론 및 향후 연구

본 연구에서는 객체 관계형 데이터베이스를 확장한 공간 데이터베이스에서 공간 인덱스를 구현하기 위한 방법 중 특정 업체에 종속적이지 않는 일반적인 방법인 GiST와 Relational Indexing을 비교/분석해 보았다.

또한 GiST와 Relational Indexing을 이용해서 R-Tree를 구성하는 방법에 대해서 간략하게 살펴보았다. 현재 두 방법은 객체 관계형 데이터베이스를 확장하는 응용 데이터베이스에서 많이 사용되는 인덱스 확장 방법임에도 불구하고 각각의 비교 연구 및 성능 평가가 미흡한 실정이다. 따라서 향후에는 앞에서 살펴본 두 방법으로 R-Tree를 구현하고, 동일한 플랫폼에서 공간 영역 질의에 대한 성능을 비교하여 공간 인덱스 개발에 적합한 인덱스 개발 방법을 제시하고자 한다.

6. 참고 문헌

[1] Joseph M. Hellerstein, Jeffrey F. Naughton, Avi Pfeffer, "Generalized Search Trees for Database Systems", Proceedings of 12th VLDB Conference, 1995

[2] Hans-Peter Kriegel, Marco Potke, Thomas Seidl, "The Paradigm of Relational Indexing : A Survey", Database Systems for Business, Technology and Web, 2003

[3] 이상원, "멀티미디어 데이터를 위한 확장형 인덱스 소개", 데이터베이스연구 제 19권 제 3호 2003.9