Ranking-tree : N번째 레코드로부터의 순위 탐색

이태원[O] 송병호[+] 이석호
서울대학교 전기컴퓨터공학부
[+]상명대학교 소프트웨어학과
warrior@db.snu.ac.kr, bhsong@sangmyung.ac.kr, shlee@cse.snu.ac.kr

Ranking-tree: Select from the N[th] record

Taewon Lee[O] Byungho Song[+] Sukho Lee
School of Electric and Computer Engineering, Seoul National University
[+] Department of Software, Sangmyung University

요 약

데이타베이스에서 질의의 결과로 되돌려지는 레코드의 수가 많을 때, 이를 일정 개수의 레코드 단위로 구분하여 일부만 돌려주는 응용이 많다. 추가의 결과를 요청할 경우, 이를 수행하기 위해 기존에는 다시 동일한 질의를 수행한 후 필요한 레코드 위치까지 순차적으로 접근을 하는 방식을 썼다. 본 논문에서는 인덱스가 정의된 필드를 기준으로 정렬된 결과 집합에서 효율적으로 순위 탐색을 지원하기 위한 Ranking-tree 인덱스 구조를 제안한다. 순위 탐색은 어떤 기준에 의해 정렬된 결과 레코드에서 N번째 순서에 해당하는 레코드부터 탐색하는 것을 말한다. Ranking-tree를 통해 불필요한 질의 결과 탐색 시간을 줄여 성능 향상을 가져올 수 있다.

1. 서론

최근의 데이타베이스는 인터넷 상에서 웹서버와 연동되어 사용자가 원하는 결과를 동적으로 보여주는 응용에 많이 사용되고 있다. 인터넷 상에서 사용자가 특정 조건을 만족하는 결과를 웹서버에 요청하고, 요청을 받은 웹서버는 조건에 맞는 질의를 데이타베이스 서버에 요청하여 그 결과를 편집하여 보여준다. 결과 레코드가 많은 경우, 한번에 보여줄 수 없기 때문에 일정 개수(한 페이지)씩 나누어서 보여주게 된다. 이를 효율적으로 수행하기 위해서는 질의 수행 결과에서 n번째 레코드를 직접 접근할 수 있어야 한다.

지금까지의 상용 데이타베이스에서는 특정 레코드를 빠르게 접근하기 위해서 B^+-tree와 같은 인덱스를 사용하였다.[1] 이 경우, 특정 키 값을 갖는 레코드를 빠르게 접근하는 것이 용이하다. 하지만 순위 탐색을 위해서는 정렬되어 있는 레코드에서 맨 첫번째 레코드부터 순차적으로 접근하는 방법밖에 없다. 많은 응용들이 결과를 여러 페이지로 나누어 보여주는데, 데이타베이스가 순위 탐색을 효율적으로 지원하지 못하기 때문에 불필요한 질의 결과 탐색에 자원이 소비되고 있다.

본 논문에서는 데이타베이스에서 순위 탐색을 지원하기 위해 특정 순위에 해당하는 레코드를 빠르게 접근할 수 있는 인덱스 구조를 제안하고자 한다. 이런 인덱스 구조를 데이타베이스에 적용함으로써 대부분의 순위 탐색을 사용하는 응용에서 효율을 향상시킬 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구에 대해 살펴본다. 3장에서는 순위 탐색을 지원하기

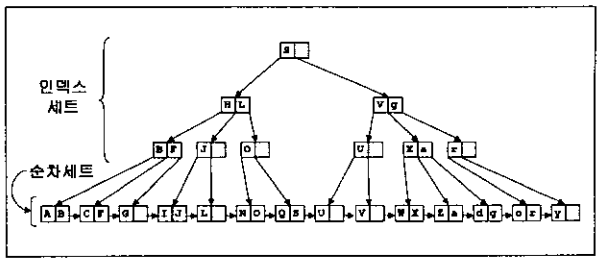
위한 인덱스 구조에 대해 살펴보고, 4장에서는 기존 인덱스와의 성능 비교를 해 본다. 마지막으로 5장에서 결론을 맺도록 한다.

2. 관련연구

상용 데이타베이스에서 사용하는 B^+-tree는 크게 두 부분으로 구성된다. 인덱스 세트(index set)와 순차 세트(sequence set)이다.[2] 인덱스 세트에 있는 키값은 리프 노드에 있는 키값을 찾아갈 수 있도록 경로를 제공하는 목적으로 사용된다. M원 트리인 경우, 인덱스 세트 내에 있는 중간 노드에는 m개의 키 값과, m+1개의 포인터가 존재한다. 포인터 P_i는 다시 중간 노드를 가리키거나 순차 세트에 존재하는 리프 노드를 가리킨다. P_i가 가리키는 서브 트리에 있는 모든 키 값들은 키 K_i보다 클 수 없다. P_{i+1}이 가리키는 서브 트리에 있는 키 값들은 모두 K_i보다 크다. 순차 세트에 있는 모든 노드는 순차적으로 연결되어 있고, 노드 내의 키 값은 오름차순으로 정렬되어 있다. 따라서 데이타를 순차적으로 접근하려면 인덱스 세트를 이용하여 첫번째 데이타를 찾고, 순차 세트를 이용하여 순차적으로 접근한다.

```
select * from T where a > 'K' order by a
```

위와 같은 질의가 들어왔을 때, 이 조건을 만족하는 레코드가 100개 정도 존재하고, 10개 단위로 그 결과를 표시하는 응용이 있다고 하자. 필드 a에 대해 B^+-tree 인덱스가 정의되어 있다고 하면, 키 값이 "K"인 레코드를 찾아 순차 세트에서 순차적으로 레코드를 얻어낼 수 있다. 하지만 10개 단위로 그 결과를 돌려주기 때문에, 첫 10개의 레코드는 가져올 수 있지만, 그 다음 10개의 레코드를 요청했을 때에는 다시 키 값이 "K"인 레코드를 찾아서 처음 10개의 레코드를 순차 세트에서 차례로 순회하고 그 다음 10개의 레코드를 결과로 돌려주어야 한다. 이것은 B^+-tree가 특정한 키 값을 갖는 레코드를 빠르게 찾기 위한 구조로 설계되었기 때문이다.



[그림 1] B^+-tree

3. Ranking-tree

본 논문에서 제안하는 Ranking-tree는 다음과 같은 특징을 가진다.

1) 인덱스 세트에 있는 중간 노드는 각 포인터 P_i마다 정수값 C_i를 추가로 갖는다. C_i는 P_i가 가리키는 서브 트리에 존재하는 키 값의 수를 나타낸다.
2) 한 노드에서 ΣC_i 는 해당 노드를 가리키는 포인터 P_j에 해당하는 C_j 값과 같다.

이외의 모든 구조는 B^+-tree와 동일하다.

이 인덱스 구조 하에서 탐색 / 삽입 / 삭제에 대해 살펴보면 다음과 같다.

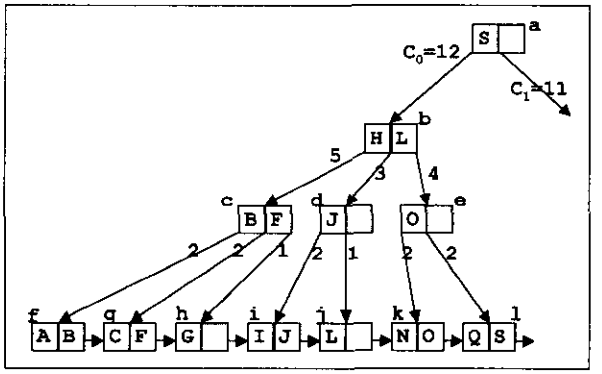
3.1 탐색

순위 탐색은 다음과 같이 수행한다. [그림 2]에서 11번째 순위에 해당하는 레코드를 찾기 위해서는 노드 a로부터 $11 \le \Sigma C_i$ (i=0..*j*)를 만족하는 *j*를 찾고, P_j를 따라 서브 트리로 이동한다. 여기서는 *j*가 0이 될 것이고 노드 b에서 $11 \le \Sigma C_i$ (i=0..*k*)를 만족하는 *k*를 찾는다. *k*는 2가 되므로 이번에는 노드 e로 포인터를 따라간다. 이 경우에 노드 e에 있는 C_i 값들은 해당 노드의 하위 서브 트리에 존재하는 키 값의 수만을 나타내기 때문에 해당 노드를 가리키는 상위 노드의 포인터가 P_j라 하면, 상위 노드에서의 ΣC_k (k=0..*j*-1) + 해당 노드에서의 ΣC_k (k=0..*p*)를 11과 비교해야 한다. 이 경우, 노드 b에서의 ΣC_k (k=0..1) = 8이었으므로 노

드 e에서는 $11 \le 8+\Sigma C_k$ (k=0..*j*) 에 해당하는 최소의 *j*를 찾아야 한다. 노드 l에 있는 것을 알 수 있으므로 최종적으로 노드 l에서 첫번째 키가 바로 찾으려는 11번째 순위에 해당하는 레코드가 된다. 이를 일반화한 공식은 자명하므로 생략하도록 하겠다.

인덱스를 이용하여 탐색하는 경우의 비용을 B$^+$-tree의 경우와 비교해 보면, 매 노드에서 차례로 어느 포인터를 따라 하위노드로 갈지를 결정해야 하기 때문에 같은 회수의 비교를 하게 된다. Ranking-tree에서는 ΣC_i를 계산해야 하는 추가비용이 들지만, 간단한 덧셈 연산이므로 무시할 수 있는 비용이다.

또한 Ranking-tree에서 특정 키 값을 갖는 데이타를 찾는 것은 B$^+$-tree에서와 동일하다.



[그림 2] Ranking-tree

3.2 삽입/삭제

새로운 레코드의 삽입시에는 레코드의 키가 위치할 리프 노드까지의 경로를 따라 경로상에 있는 포인터 P_i에 해당하는 C_i의 값을 1씩 증가시켜 준다. 리프 노드에서 오버플로가 발생하여 분열될 때에는 오버플로가 발생한 노드의 중간 키 값이 상위 노드로 올라가 저장되면서 중간 키보다 작거나 같은 키 값의 수가 오버플로가 발생한 노드를 가리키는 P_i의 C_i값이 되고, 추가된 노드에 들어있는 키 값의 수가 C_{i+1}이 된다.

삭제시에는 삽입과 반대과정이 일어나게 된다. 최상위 노드로부터 삭제될 키가 있는 노드까지 내려가면서 해당 포인터의 C_i 값을 1씩 감소시킨다.

4. 성능 평가

본 논문을 작성할 당시에는 실험이 완결되지 않아서 실험 결과를 싣지는 못했다. 기존 인덱스 구조에서는 순위 탐색이 불가능했기 때문에 순위 탐색의 성능을 비교하는 것은 어려움이 있다. 기존 데이타베이스에서는 순위 탐색을 위해 동일한 질의를 재실행한 후, 불필요한 레코드를 무시하는 방법을 사용하는데, 이 경우의 디스크 접근 회수를 Ranking-tree의 디스크 접근 회수와 비교해 봄으로써 성능 비교가 가능하다고 생각한다. 또한 B$^+$-tree에 비해 추가적인 정보를 저장할 공간이 요구되므로 이에 따른 fan-out의 감소와 이로 인한 탐색 시간 증가에 대한 실험도 진행 중이다.

5. 결론

본 논문에서 제안한 Ranking-tree는 기존의 인덱스 구조를 개선하여 데이타베이스 자체에서 순위 탐색을 효율적으로 지원하기 위한 구조이다. 이를 통해 특정 조건을 만족하는 많은 레코드를 돌려주는 다양한 응용들은 순위 탐색을 효율적으로 처리할 수 있다. 지금까지 데이타베이스에서 질의 결과 중 상위 N개만을 가져오는 방법에 대한 연구[3]는 있었지만, N번째부터의 결과를 가져오는 순위 탐색에 대한 연구가 필요하다는 것을 밝힌 점에서 본 논문의 가치가 충분하다.

많은 제약 조건 하에서의 성능 향상이지만, 이를 개선하기 위해 인덱스가 정의되어 있지 않은 필드가 포함된 조건이 있는 질의에서 순위 탐색을 지원하기 위한 개선 방안과 질의 최적화 단계에의 적용 등을 추후 연구할 예정이다.

6. 참고 문헌

[1]. 이석호, "데이타베이스론", 정익사, p.120-122

[2]. Don S. Batory, "*B+ Trees and Indexed Sequential Files: A Performance Comparison*", SIGMOD Conference 1981, p.30-39

[3]. Surajit Chaudhuri, Luis Gravano, "*Evaluating Top-k Selection Queries*", Proceedings of the 25th VLDB Conference, 1999, p.397-410