Indexing

Index란?

인덱스란 추가적인 쓰기 작업과 저장 공간을 활용하여 데이터베이스 테이블의 검색 속도를 향상시키기 위한 자료구조이다. 만약 우리가 책에서 원하는 내용을 찾는다고 하면, 책의 모든 페이지를 찾아 보는 것은 오랜 시간이 걸린다. 그렇기 때문에 책의 저자들은 책의 맨 앞 또는 맨 뒤에 색인을 추가하는데, 데이터베이스의 index는 책의 색인과 같다. 데이터베이스에서도 테이블의 모든 데이터를 검색하면 시간이 오래 걸리기 때문에 데이터와 데이터의 위치를 포함한 자료구조를 생성하여 빠르게 조회할 수 있도록 돕고 있다.

인덱스를 활용하면, 데이터를 조회하는 SELECT 외에도 UPDATE나 DELETE의 성능이 함께 향상된다. 그러한 이유는 해당 연산을 수행하려면 해당 대상을 조회해야만 작업을 할 수 있기 때문이다.

Index의 장점과 단점

- 장점
  - 테이블을 조회하는 속도와 그에 따른 성능을 향상시킬 수 있다.
  - 전반적인 시스템의 부하를 줄일 수 있다.
- 단점
  - 인덱스를 관리하기 위해 DB의 약 10%에 해당하는 저장공간이 필요하다.
  - 인덱스를 관리하기 위해 추가 작업이 필요하다.
  - 인덱스를 잘못 사용할 경우 오히려 성능이 저하되는 역효과가 발생할 수 있다.

만약 CREATE, DELETE, UPDATE가 빈번한 속성에 인덱스를 걸게 되면 인덱스의 크기가 비대해져서 성능이 오히려 저하되는 역효과가 발생할 수 있다. 그러한 이유 중 하나는 DELETE와 UPDATE 연산 때문이다. 앞에서 설명한대로, UPDATE와 DELETE는 기존의 인덱스를 삭제하지 않고, '사용하지 않음' 처리를 해준다. 만약 어떤 테이블에 UPDATE와 DELETE가 빈번하게 발생된다면 실제 데이터는 10만건이지만 인덱스는 훨씬 많이 존재하게 되어, SQL문 처리 시 비대해진 인덱스에 의해 오히려 성능이 떨어지게 될 것이다.

Index를 사용하면 좋은 경우

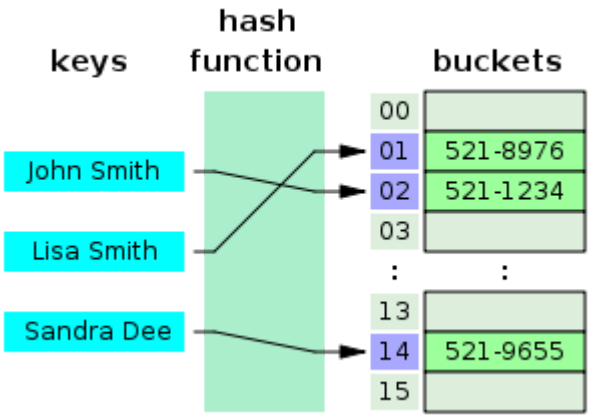
- 규모가 작지 않은 테이블
- INSERT, UPDATE, DELETE가 자주 발생하지 않는 컬럼
- JOIN이나 WHERE 또는 ORDER BY에 자주 사용되는 컬럼
- 데이터의 중복도가 낮은 컬럼

Index의 자료구조

인덱스를 구현하기 위해서는 다양한 자료구조를 사용할 수 있는데, 가장 대표적인 해시 테이블과 B+Tree 에 대해서 알아보도록 하자.

해시 테이블

해시 테이블은 (Key, Value)로 데이터를 저장하는 자료구조 중 하나로 빠른 데이터 검색이 필요할 때 유용하다. 해시 테이블은 Key값을 이용해 고유한 index를 생성하여 그 index에 저장된 값을 꺼내오는 구조이다.



해시 테이블 기반의 DB 인덱스는 데이터의 위치를 (Key, Value)로 사용하여 컬럼의 값으로 생성된 해시를 통해 인덱스를 구현하였다. 해시 테이블의 시간복잡도는 O(1)이며 매우 빠른 검색을 지원한다.

하지만 DB 인덱스에서 해시 테이블이 사용되는 경우는 제한적인데, 그러한 이유는 해시가 등호(=)연산에만 특화되었기 때문이다. 해시 함수는 값이 1이라도 달라지면 완전히 다른 해시 값을 생성하는데, 이러한 특성에 의해 부등호 연산(>, <)이 자주 사용되는 데이터베이스 검색을 위해서는 해시 테이블이 적합하지 않다.

즉, 예를 들면 “나는”으로 시작하는 모든 데이터를 검색하기 위한 쿼리문은 인덱스의 혜택을 전혀 받지 못하게 된다. 이러한 이유로 데이터베이스의 인덱스에서는 B+Tree가 일반적으로 사용된다.

B+Tree

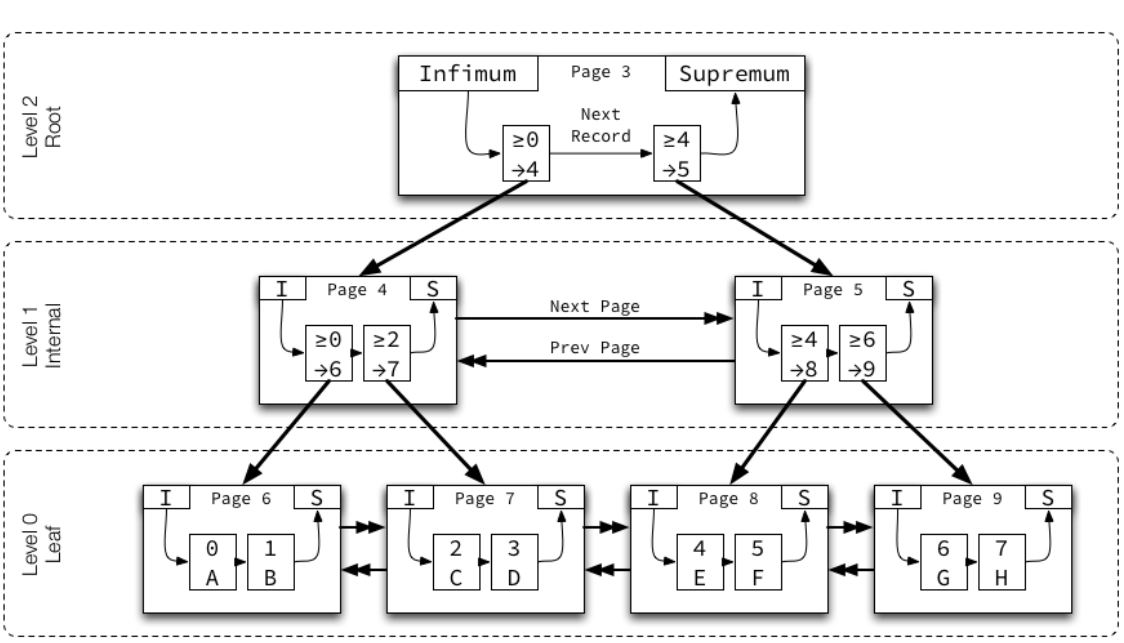
B+Tree는 DB의 인덱스를 위해 자식 노드가 2개 이상인 B-Tree를 개선시킨 자료구조이다. B+Tree는 모든 데이터를 저장했던 B Tree와 다른 특성을 가지고 있다.

- 리프 노드만 인덱스와 함께 데이터를 가지고 있고, 나머지 노드들은 데이터를 위한 인덱스만을 갖는다.
- 리프노드들은 LinkedList로 연결되어 있다.
- 데이터 노드 크기는 인덱스 노드 크기와 같지 않아도 된다.

데이터베이스의 인덱스 컬럼은 부등호를 이용한 순차 검색 연산이 자주 발생될 수 있다. 이러한 이유로 B Tree의 리프 노드들을 LinkedList로 연결하여 순차검색을 용이하게 하는 등 B Tree를 인덱스에 맞게 최적화하였다. (물론 Best Case에 대해 리프노드까지 가지 않아도 탐색할 수 있는 B Tree에 비해 무조건 리프노드까지 가야한다는 단점도 있다.

이러한 이유로 비록 B+Tree는 O(logN)의 시간복잡도를 갖지만 해시테이블보다 인덱싱에 더욱 적합한 자료구조가 되었다.

아래의 그림은 InnoDB에서 사용된 B+ Tree의 구조이다.



Levels are numbered starting from 0 at the leaf pages, incrementing up the tree.
Pages on each level are doubly-linked with previous and next pointers in ascending order by key.
Records within a page are singly-linked with a next pointer in ascending order by key.
Infimum represents a value lower than any key on the page, and is always the first record in the singly-linked list of records.
Supremum represents a value higher than any key on the page, and is always the last record in the singly-linked list of records.
Non-leaf pages contain the minimum key of the child page and the child page number, called a "node pointer".

InnoDB에서의 B+ Tree는 일반적인 구조보다 더욱 복잡하게 구현이 되었다. InnoDB에서는 같은 레벨의 노드들끼리는 LinkedList가 아닌 Double Linked List로 연결되었으며, 자식 노드들은 Single Linked List로 연결되어 있다.

https://mangkyu.tistory.com/96