● Index가 왜 필요한가?

▶ Index는 데이터베이스에서 table의 검색 성능을 높여주는 대표적인 방법중에 하나입니다. 일반적인 RDBMS(관계형 데이터 베이스)에서는 B+Tree구조로 된 index를 사용하여 검색 속도를 향상시킵니다. Table에 데이터를 지속적으로 저장하게 되면 내부적으로 순서 없이 쌓이게 되는데, 이 경우 특정 조건을 만족하는 데이터를 찾고자 WHERE절을 사용한다면 Table의 row(record)를 처음부터 끝까지 모두 접근하여 검색조건과 일치하는지 비교하는 과정이 필요하고, 이를 Full Table Scan이라고 합니다. 하지만 특정 coloumn에 대한 Index를 생성해 놓은 경우 해당 속성에 대하여 search-key가 정렬되어 저장되어 있기 때문에 조건 검색(SELECT ~ WHERE) 속도가 굉장히 빠든 것입니다. 즉, index는 마치 책마다 마지막 페이지에 있는 색인(index)과 같은 역할을 하는 자료구조로, SELECT ~WHERE query를 통해 특정 조건을 만족하는 데이터를 찾을 때 full table scan을 할 필요 없이 정렬되어 있는 index에서 index scan을 통해 훨씬 빠든 속도로 검색을 할 수 있게 되는 것입니다.

● Index의 구조

▶ Index는 Btree, B+tree, Hash, Bitmap로 구현될 수 있지만, 많은 데이터베이스 시스템에서 index는 B+tree구조를 가집니다. 특정 column을 search-key값으로 설정하여 index를 생성하면, 해당 column(즉 속성 attribute) 값을 기준으로 정렬한 search-key값을 실제 데이터의 물리적 위치 pointer와 함께 별도 파일에 저장을 하고, 이를 index라고 합니다. 즉, index는 순서대로 정렬된 search-key값과 pointer값만 저장하기 때문에, table보다 적은 공간을 차지합니다. 보통 table의 크기의 10%정도의 공간을 차지합니다.

● 클러스터형 인덱스와 보조 인덱스(clustering index & secondary index)

▶ clustering index이란 특정 column을 기본키(primary key)로 지정하면 자동으로 클러스터형 인덱스가 생성되고, 해당 column 기준으로 테이블 전체의 데이터가 정렬이 됩니다. Table 자체가 정렬된 하나의 index인 것입니다. 마치 영어사전처럼 책의 내용 자체가 정렬된 것과 같습니다.

▶ secondary index(보조인덱스)이란 일반 책의 찾아보기와 같이 별도의 공간에 인덱스가 생성됩니다. create index와 같이 index 생성하기를 하거나 고유키(unique key)로 지정하면 보조 인덱스가 생성됩니다.

```
CREATE INDEX idx_name ON Student (이름);

SELECT * FROM Student WHERE 이름 = '노정호';
```

● Index의 장단점

▶ Index의 최대 장점은 검색 속도 향상(SELECT~WHERE~)입니다. 테이블을 만들고 안에 데이터가 쌓이게 되면 테이블의 record는 내부적으로 순서가 없이 뒤죽박죽으로 저장됩니다. 이렇게 되면 WHERE절을 통해 특정 조건에 맞는 데이터들을 찾아낼 때에도 record의 처음부터 끝까지 다 읽어서 검색 조건과 맞는지 비교해야 하고, 이것을 Full Table Scan이라고 합니다. 반면에 index를 생성하면 index에는 데이터들이 정렬되어 저장되어 있기 때문에 검색 조건에 일치하는 데이터들을 빠르게 찾아낼 수 있습니다. 이것이 Index를 사용하는 가장 큰 이유입니다.

반면, Index의 단점으로는 크게 두 가지가 있습니다.

1) 첫째, 추가 저장공간 필요합니다.

index를 생성하면 index 자료구조를 위한 저장 공간이 추가적으로 필요합니다. 보통 table의 크기의 10%정도의 공간을 차지합니다.

2) 그리고 둔째, 데이터 변경 작업이 느립니다.

검색이 아닌 데이터 변경을 할 때, 즉 INSERT, UPDATE, DELETE가 자주 발생하면 성능이 나빠질 수 있습니다. 그 이유는 보통 B+tree구조의 index는 데이터가 추가 삭제 될 때마다 tree의 구조가 변경될 수 있기 때문입니다. 즉 인

덱스의 재구성이 필요하기 때문에 추가적인 자원이 소모됩니다.

● SELECT의 성능을 높일 수 있는 방법은 뭐가 있을까?
▶ SELECT query의 성능을 높이는 가장 대표적인 방법중에 하나는 Index를 사용하는 것입니다. index를 사용하게 되면 SELECT WHERE절처럼 특정 조건을 만족하는 데이터를 검색할 때 table의 모든 데이터를 살펴볼 필요 없이 index에서 빠르게 해당 데이터에 접근할 수 있게 됩니다.

● index의 내부동작은 어떻게 동작하나?
▶ index는 대부분 B+Tree 자료구조로 이루어져 있어, WHERE 조건과 일치하는 데이터들의 저장위치를 훨씬 빠르게 찾을 수 있습니다. B+Tree는 리프 노드에 모든 데이터가 있어 한 번의 선형 탐색으로 원하는 자료를 찾을 수 있기 때문입니다.

● index를 많이 생성하면 안되나?
▶ index를 생성하면 조건 검색 성능이 향상될 수 있습니다. 하지만 추가 저장공간이 필요하고, 데이터를 추가/수정/삭제를 할 때마다 관련 index를 모두 수정해줘야 되기 때문에 시간이 추가적으로 소요됩니다. 따라서 추가 저장공간과, 데이터 업데이트시 소요되는 추가 시간등을 복합적으로 고려하여 조건 검색의 성능향상이 더 큰 이득이 된다고 판단되는 column에만 index를 생성하는 것이 좋습니다.

B+tree는 이진 트리를 확장해 하나의 노드가 가질 수 있는 자식 노드의 최대 숫자가 2보다 큰 트리 구조로 자료를 정렬된 상태로 보관하는데 브랜치 노드에 key만 담아두고, data는 담지 않는다. 오직 리프 노드에만 key와 data를 저장하고, 리프 노드끼리 Linked list로 연결되어 있다. B+tree의 장점은 풀 스캔 시, B+tree는 리프 노드에 데이터가 모두 있기 때문에 한 번의 선형탐색(일렬로 된 자료를 왼쪽부터 오른쪽으로 차례대로 탐색)만 하면 되기 때문에 모든 노드를 확인해야 하는 B-tree에 비해 빠르다(zorba91tistory).

Resources: inflearn 개발남 노씨, zorba91tistory