

● Index가 왜 필요한가?

▶ Index는 데이터베이스에서 table의 검색 성능을 높여주는 대표적인 방법중 하나입니다. 일반적인 RDBMS(관계형 데이터 베이스)에서는 B+Tree구조로 된 index를 사용하여 검색 속도를 향상시킵니다. index는 책마다 마지막 페이지에 있는 색인(index)과 같은 역할을 하는 자료구조입니다. 책에서 어떤 용어나 단어를 찾기위해 첫 페이지부터 끝 페이지 까지 전체를 훑지 않아도(Full Table Scan) index를 찾아보면 몇 페이지에 적혀 있는지 바로 찾을 수 있는 것(Index Scan)과 비슷합니다. SELECT ~WHERE query를 통해 특정 조건을 만족하는 데이터를 찾을 때, full table scan할 필요 없이 정렬되어 있는 index에서 훨씬 빠른 속도로 검색을 할 수 있게 됩니다.

● Index의 구조

▶ Index는 Btree, B+tree, Hash, Bitmap로 구현될 수 있습니다. 많은 데이터베이스 시스템에서 index는 B+tree 구조를 가집니다. index를 생성하게 되면 특정 column(속성, attribute)의 값을 기준으로 정렬하여 데이터의 물리적 위치와 함께 별도 파일에 저장합니다. 이 때, Index에 저장되는 속성 값을 search-key값이라고 하고 실제 데이터의 물리적 위치를 저장한 값을 pointer라고 합니다. 즉, index는 순서대로 정렬된 search-key값과 pointer값만 저장하기 때문에 table보다 적은 공간을 차지합니다. 정리해보면 특정 column을 search-key 값으로 설정하여 index를 생성하면, 해당 search-key 값을 기준으로 정렬하여 (search-key, pointer)를 별도 파일에 저장합니다. 이를 index라고 합니다.

● Index 사용하는 이유

▶ Table에 데이터를 지속적으로 저장하게 되면 내부적으로 순서 없이 쌓이게 됩니다. 이 경우에 특정 조건을 만족하는 데이터를 찾고자 WHERE절을 사용한다면 Table의 row(record)를 처음부터 끝까지 모두 접근하여 검색조건과 일치하는지 비교하는 과정이 필요합니다. 이를 Full Table Scan이라고 합니다. 하지만 특정 column에 대한 Index를 생성해 놓은 경우 해당 속성에 대하여 search-key가 정렬되어 저장되어 있기 때문에 조건 검색(SELECT ~ WHERE) 속도가 굉장히 빠릅니다.

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

▶ clustering index: 특정 column을 기본키(primary key)로 지정하면 자동으로 클러스터형 인덱스가 생성되고, 해당 column 기준으로 정렬이 됩니다. Table 자체가 정렬된 하나의 index인 것입니다. 마치 영어사전처럼 책의 내용 자체가 정렬된 것을 떠올리면 쉽습니다.

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

학번	이름	학과	성별
202137	정재현	심리	남
202249	박현지	경영	여
202118	강상호	유아교육	남
202195	배준석	유아교육	남
202250	남영욱	유아교육	남
202073	박예지	디자인	여
202081	윤호정	화학	여
202299	박지은	영어	여
202163	노정호	전자	남
202011	박다은	전자	여



Primary key

학번	이름	학과	성별
202011	박다은	전자	여
202073	박예지	디자인	여
202081	윤호정	화학	여
202118	강상호	유아교육	남
202137	정재현	심리	남
202163	노정호	전자	남
202195	배준석	유아교육	남
202249	박현지	경영	여
202250	남영욱	유아교육	남
202299	박지은	영어	여

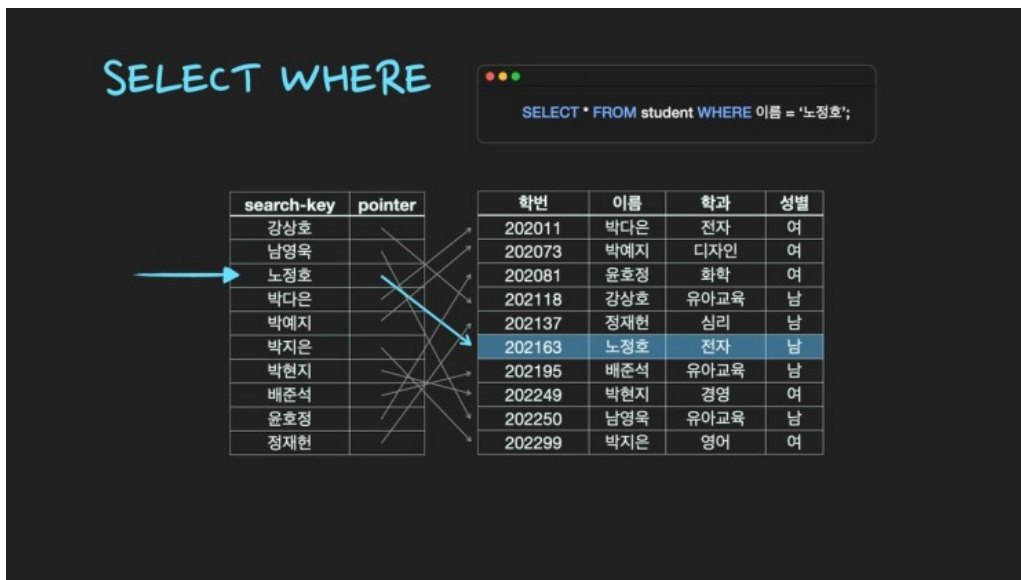
clustering index

search-key	pointer	학번	이름	학과	성별
202011	→	202011	박다은	전자	여
202073	→	202073	박예지	디자인	여
202081	→	202081	윤호정	화학	여
202118	→	202118	강상호	유아교육	남
202137	→	202137	정재현	심리	남
202163	→	202163	노정호	전자	남
202195	→	202195	배준석	유아교육	남
202249	→	202249	박현지	경영	여
202250	→	202250	남영욱	유아교육	남
202299	→	202299	박지은	영어	여

secondary index

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

search-key	pointer	학번	이름	학과	성별
강상호	→	202118	강상호	유아교육	남
남영욱	→	202250	남영욱	유아교육	남
노정호	→	202163	노정호	전자	남
박다은	→	202011	박다은	전자	여
박예지	→	202073	박예지	디자인	여
박지은	→	202299	박지은	영어	여
박현지	→	202249	박현지	경영	여
배준석	→	202195	배준석	유아교육	남
윤호정	→	202081	윤호정	화학	여
정재현	→	202137	정재현	심리	남



● 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 합니다. 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.

Index의 단점으로는 크게 두 가지가 있습니다.

1) 추가 저장공간 필요

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

2) 느린 데이터 변경 작업

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 인덱스의 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

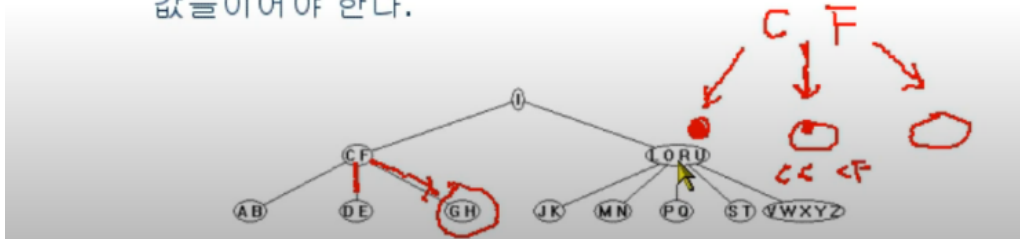
▶ index는 대부분 B+Tree 자료구조로 이루어져 있어, WHERE 조건과 일치하는 데이터들의 저장위치를 훨씬 빠르게 찾을 수 있습니다.

● 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-Tree 규칙 1

- 노드의 자료수가 N 이라면, 자식의 수는 $N+1$ 이어야 한다.
- 각 노드의 자료는 정렬된 상태여야 한다.
- 노드의 자료 D_k 의 왼쪽 서브트리는 D_k 보다 작은 값들이고, D_k 의 오른쪽 서브트리는 D_k 보다 큰 값들이어야 한다.



B-Tree는 탐색을 위해서 노드를 찾아서 이동해야 한다는 단점을 가지고 있다. 이러한 단점을 해소하고자 B+Tree는 같은 레벨의 모든 키값들이 정렬되어 있고, 같은 레벨의 Sibling node는 연결리스트 형태로 이어져 있다. (출처: ssocoittistory)