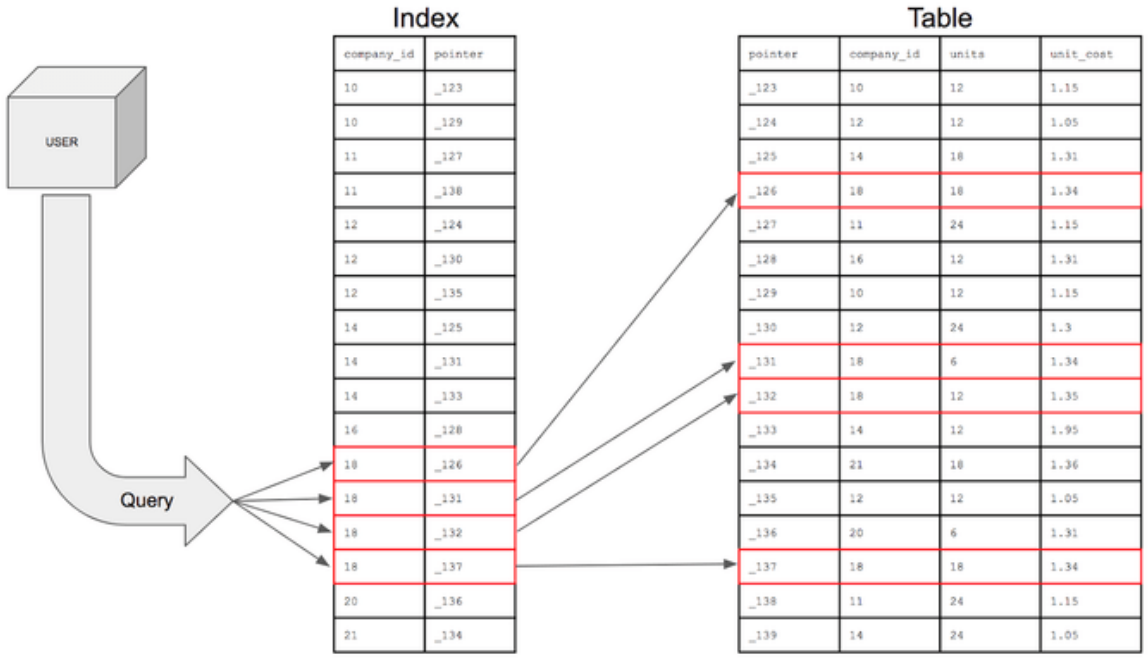
[DB] Index (1)

(1) 인덱스(Index)란?

> 💡 인덱스(index)란 자료를 쉽고 빠르게 찾을 수 있도록 만든 데이터 구조로 데이터베이스에서 인덱스란 원하는 **RDBMS 에서 검색 속도를 높이기 위한 기술**로 추가적인 쓰기 작업과 저장 공간을 활용하여 테이블 칼럼을 색인화 한 자료구조이다.

쉽게 말하자면, 책에 있는 목차를 생각하면 된다. 책에 있는 내용을 찾기 위해서, 먼저 목차에서 찾고 목차에 있는 페이지 번호를 보고 찾아가 원하는 내용을 찾듯이 인덱스에서 내가 원하는 데이터를 먼저 찾고 저장되어 있는 물리적 주소로 찾아가는 것이다.



예를 들어, 인덱스 기준이 하나도 잡혀 있지 않아 이름, 성별, 이메일 등이 담긴 회원 테이블이 입력 된 순서대로 존재한다고 가정해보자. 이때 이메일로 회원을 찾으려면 전체 데이터에서 순차적으로 찾는 이메일이 나올 때까지 검색을 하게 된다.

여기서 인덱스를 이메일로 정했다면, 이메일을 기준으로 정렬이 되게 됩니다. 또 다시 이메일로 회원을 찾으려고 한다면, 아까보다 속도가 훨씬 빨라질 것입니다.

즉, 인덱스의 특징으로는

(1) 인덱스는 항상 최신의 정렬 상태를 유지

(2) 인덱스도 하나의 데이터베이스 객체

(3) 데이터베이스 크기의 약 10% 정도의 저장 공간 필요

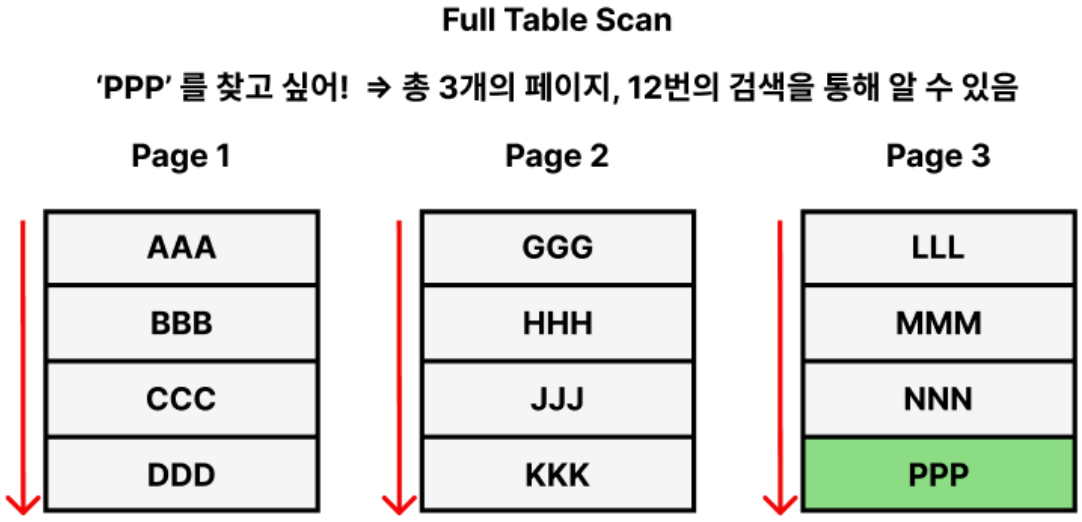
가 있다.

(2) 인덱스 알고리즘

1. Full Table Scan (인덱스 적용 X)

> 💡 Full Table Scan(순차 접근) 이란 테이블에 존재하는 모든 데이터를 읽어 가면서 조건에 맞으면 결과로서 추출하고 조건에 맞지 않으면 버리는 방식이다.

Full Table Scan 을 사용하는 경우는 (1) 적용 가능한 인덱스가 없는 경우 (2) 인덱스 처리 범위가 넓은 경우(적용 가능한 인덱스가 존재하더라도 처리 범위가 넓어서 Full Table Scan이 더 적은 비용이 든다면 ..) (3) 크기가 작은 테이블에 엑세스 하는 경우 등이 있다.

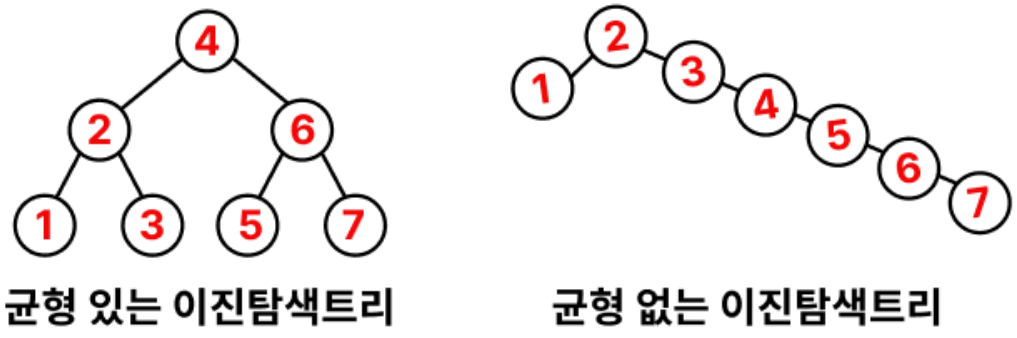


2. B-Tree(Balanced Tree)

Binary Search Tree

Binary Search Tree (이진 탐색 트리)의 단점을 극복하기 위해 나온 것 중 하나가 바로 B-Tree다.

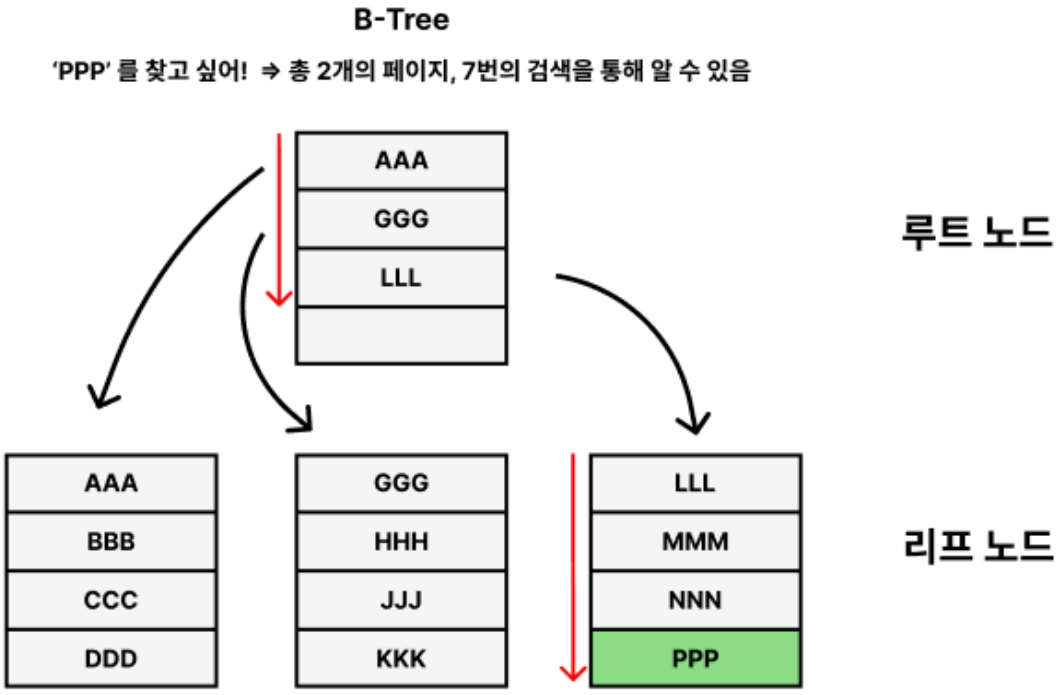
먼저, Binary Search Tree를 살펴보면, 이진 트리의 경우 각각의 노드가 최대 두 개의 자식 노드를 가지는 트리 자료구조이다. 균형 있는 이진 탐색 트리의 경우 검색 시간 복잡도가 O(logN) 이지만, 균형 없는 이진 탐색 트리의 경우 시간 복잡도는 O(n) 이다. 즉, '운이 나쁘면..' 전체를 다 탐색해야 할 수도 있다는 소리다.



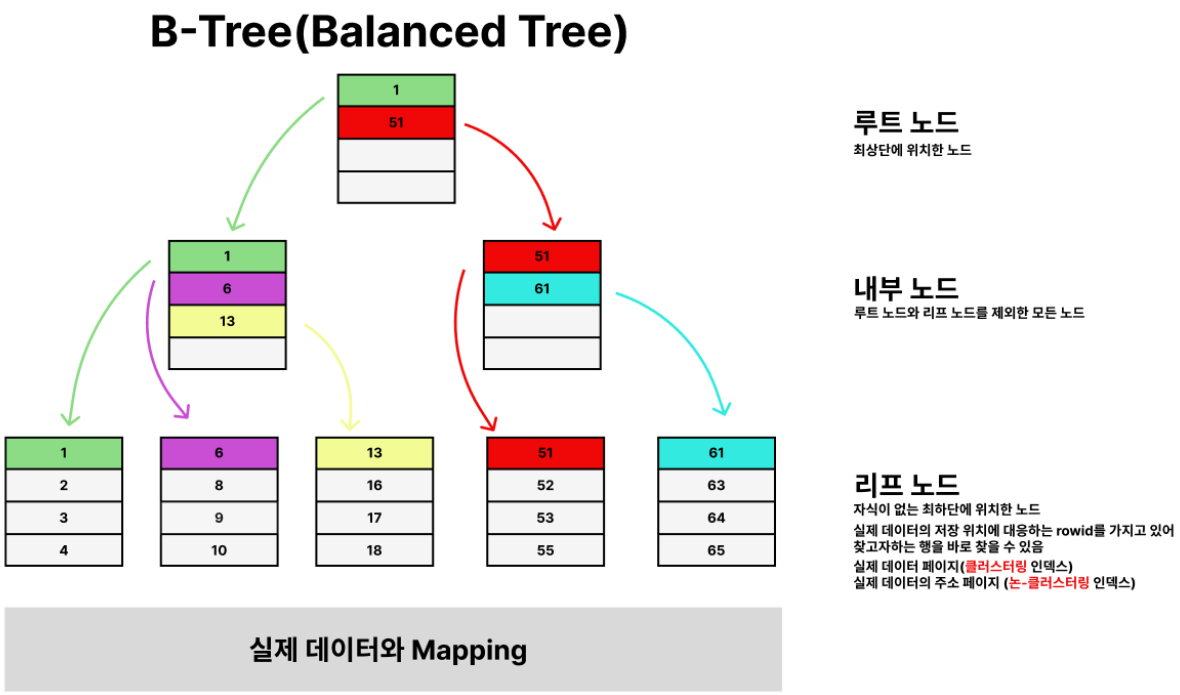
이러한 단점을 극복하기 위해 나온 것이 B-Tree 다.

B-Tree

아까 Full Table Scan으로 진행했던 것을 B-Tree로 진행을 한다면 아래와 같게 된다.



💡 B-Tree: 대부분의 RDBMS 는 B-Tree 구조로 되어 있다. 데이터의 검색 시간을 단축하기 위한 자료구조로, 루트 노드, 내부 노드, 리프 노드로 구성되며, 리프 노드가 모두 같은 레벨에 존재하는 균형(Balanced) 트리로 O(logN) 시간 복잡도를 가진다.



B-Tree 특징

(1) 모든 노드는 최대 m개의 자식들을 가진다. (2개 이상의 자식 노드를 가질 수 있다.)

(2) 키 값은 오름차순으로 저장

(3) 모든 리프노드들은 같은 높이에 있어야 한다.

B-Tree 의 장점

- 균형된 트리 구조를 이용하기 때문에 한 번 검색할 때마다 검색 대상이 줄어들어 접근 시간이 적게 걸린다.
- 노드의 삽입/삭제 후에도 균형 트리 유지로 균등한 응답 속도를 보장한다.

B-Tree 의 단점

- B-Tree 의 경우에는, 노드의 높이가 한정되어 있기 때문에 데이터를 삽입하려고 할 때 해당 노드가 꽉 차 있다면 동적으로 노드를 분할하게 된다. (비어있는 노드를 확보한 다음 문제가 되는 노드에 데이터를 공평하게 나누어 저장)
- 또한, 데이터를 삭제하려고 할 때도 리프 노드의 경우에는 삭제하고, 최소 키를 가지지 않았다면 다시 재조정하게 된다.

→ 즉, 삽입/삭제를 할 때 처리할 대상을 찾기 위한 조회 성능은 향상 하겠지만 B-tree의 모양을 유지하기 위해 노드의 분할 및 이동이 자주 발생해 성능이 떨어지게 된다.

> 💡 즉, SELECT 문에 대한 성능은 향상되지만 INSERT UPDATE DELETE 문에서는 성능이 저하된다. 이러한 단점에도 불구하고 일반적인 RDBMS 에서 B-Tree를 쓰는 이유는 일반적인 웹서비스에서 쓰기와 읽기의 비율이 2:8 ~ 1:9 정도 되기 때문이다.

다음주에는 인덱스의 종류인 클러스터링 인덱스, 논-클러스터링 인덱스에 대해 알아보고 인덱스 적용 시 주의사항에 대해 알아보자!!