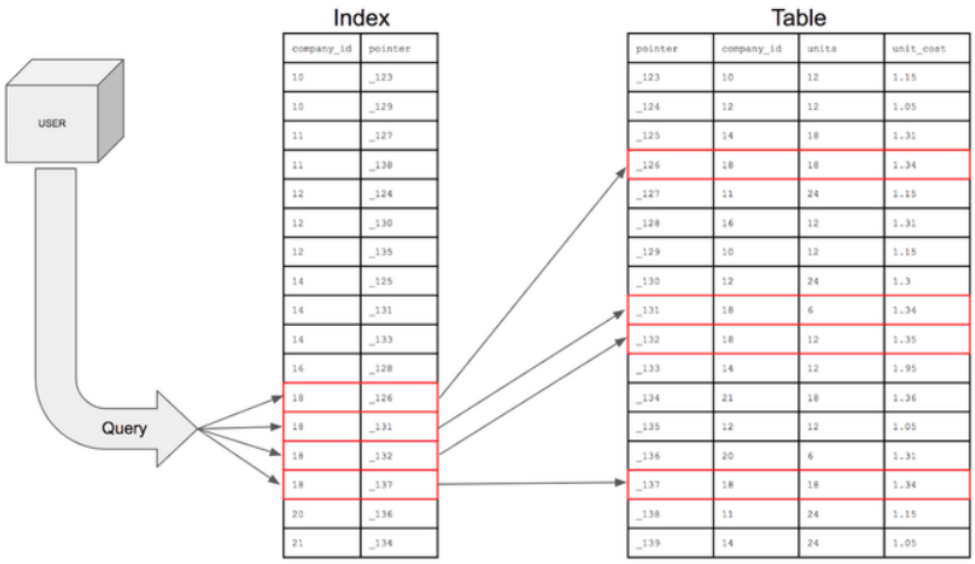
Index

DB Index?

1. DB 의 테이블에 대한 검색 속도를 향상시켜주는 자료구조
2. 테이블의 특정 컬럼에 인덱스를 생성하면, 해당 컬럼의 데이터를 정렬한 후 별도의 메모리 공간에 데이터의 물리적 주소와 함께 저장된다. 컬럼의 값과 물리적 주소를 (key, value)의 한 쌍으로 저장
3. 인덱스를 활용하면 SELECT 외, UPDATE, DELETE 성능이 함께 향상된다.
4. 해당 table 의 레코드를 full scan하지 않는다.

예) 책

데이터 - 책의 내용, 인덱스 - 책의 목차, 물리적 주소 - 책의 페이지 번호



Index의 관리

DBMS는 index를 항상 최신의 정렬된 상태로 유지해야 원하는 값을 빠르게 탐색할 수 있다.

인덱스가 적용된 칼럼에 하기와 같은 연산을 추가적으로 한다.

- INSERT : 새로운 데이터에 대한 인덱스 추가
- DELETE : 삭제하는 데이터의 인덱스를 사용하지 않는다는 작업 진행
- UPDATE : 기존의 인덱스를 사용하지 않음 처리, 갱신된 데이터에 대해 인덱스 추가

Index의 장점과 단점

- 장점

    - 테이블 조회 속도 및 그에 따른 성능향상
    - 전반적인 시스템 부하 줄임
- 단점
    - 인덱스를 관리하기 위해 db의 약 10%에 해당하는 저장공간이 필요
    - 인덱스를 관리하기 위한 추가 작업이 필요함
    - 인덱스를 잘 못 사용할 경우 오히려 성능이 저하되는 역효과가 발생

사용하면 좋은 경우

- 규모가 큰 테이블
- INSERT, UPDATE, DELETE가 자주 발생하지 않는 컬럼
- JOIN 이나 WHERE 또는 ORDER BY에 자주 사용되는 컬럼
- 데이터의 중복도가 낮은 컬럼

INDEX의 자료구조

해시테이블

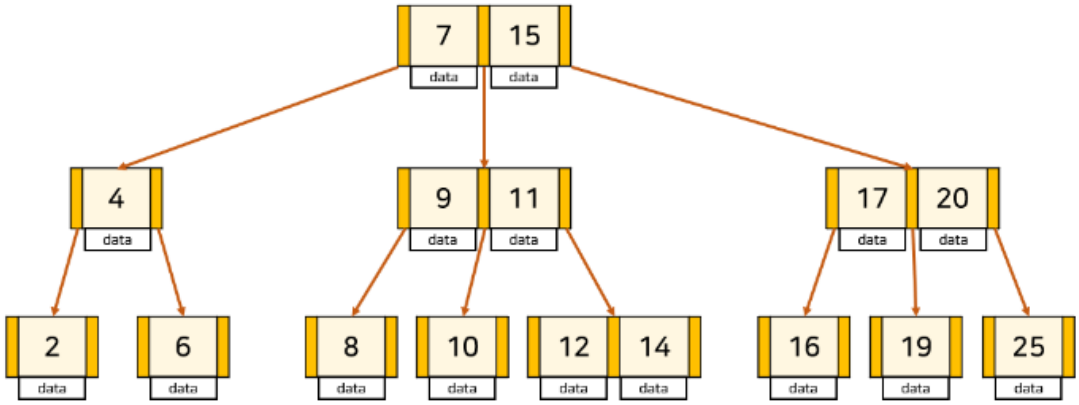
- (Key, Value)로 데이터를 저장하는 자료구조
- 시간복잡도 = O(1)로 빠른 데이터 검색 필요할때 유용
- Key값을 이용해 고유한 Index를 생성하여 그 index에 저장된 값을 꺼내오는 구조
    - (key,value) = (컬럼의 값, 데이터의 위치)

→ 하지만 실제로 인덱스에서 잘 사용하지 않음. 해시 테이블은 등호(=)연산에 최적화되어있기 때문. 데이터베이스에서는 부등호 연산이 자주 사용되는데, 해시 테이블 내의 데이터들은 정렬되어있지 않으므로 특정 기준보다 크거나 작은 값을 빠른 시간내에 찾을 수 없다.

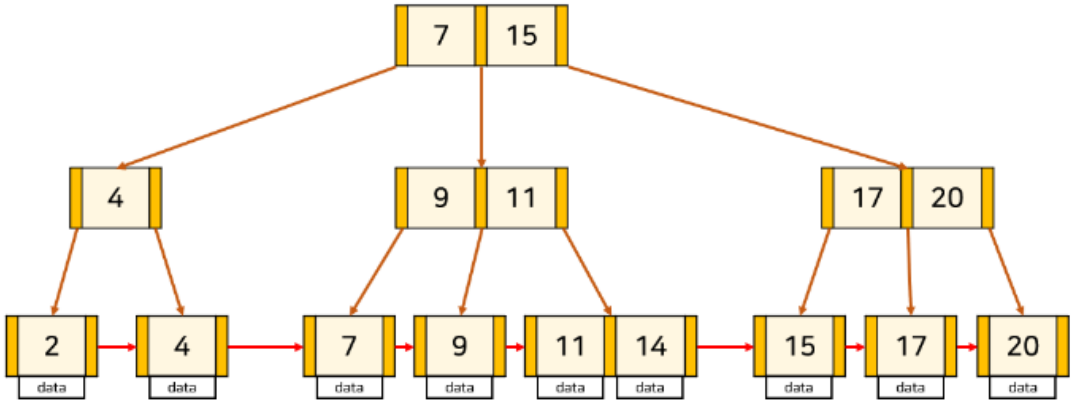
B+ Tree

기존의 B-Tree는 어느 한 데이터의 검색은 효율적이지만, 모든 데이터를 한번 순회하는 데에는 트리의 모든 노드를 방문해야하므로 비효율적. 이러한 B-Tree의 단점을 개선시킨 자료구조가 B+Tree

B+Tree는 오직 leaf node에만 데이터를 저장하고 leaf node가 아닌 node에서는 자식 포인터만 저장. 그리고 leaf node끼리는 linked list로 연결.



B-Tree 예시



B+Tree 예시

reference)

- https://rebro.kr/167
-