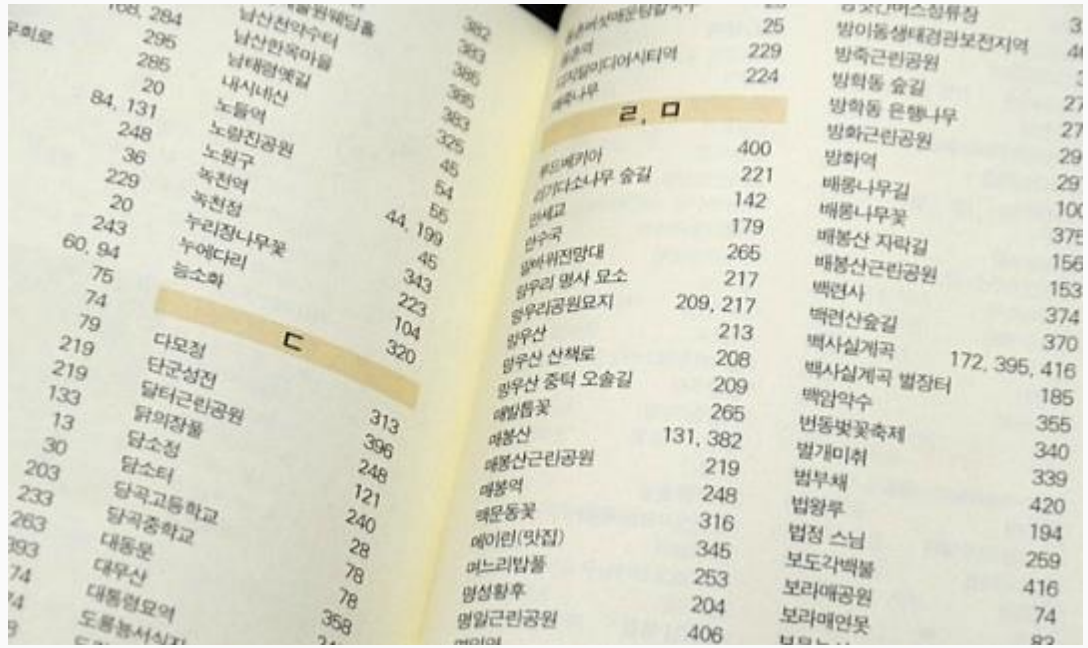
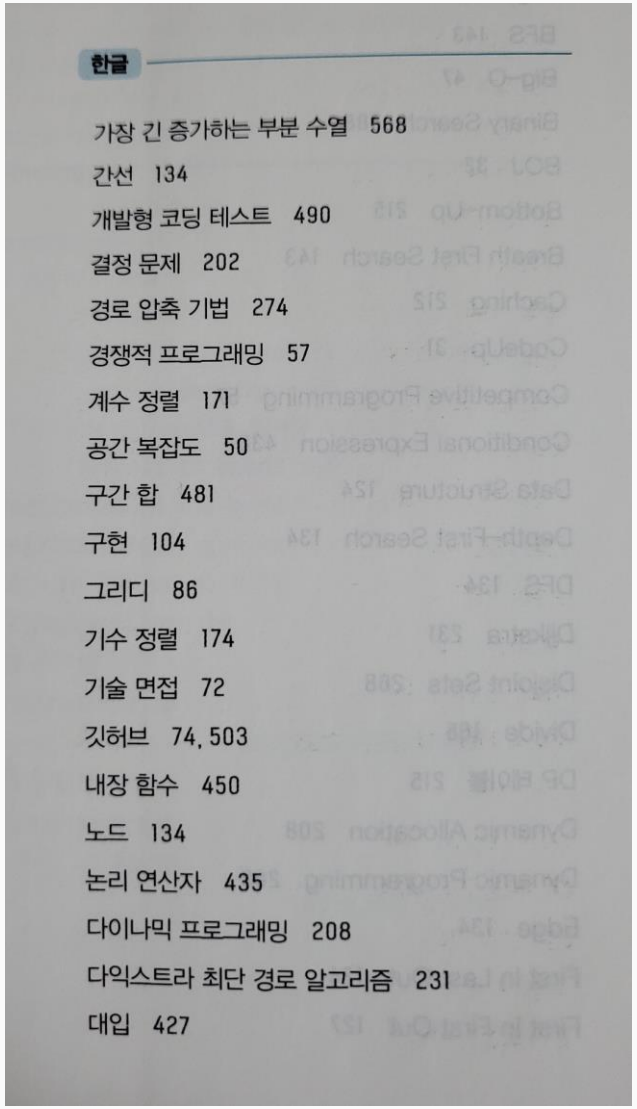
CS study 8주차
데이터베이스 Index 😎

인덱스란?

추가적인 쓰기 작업과 저장 공간을 활용하여 데이터베이스 테이블의 검색 속도를 향상시키기 위한 자료구조

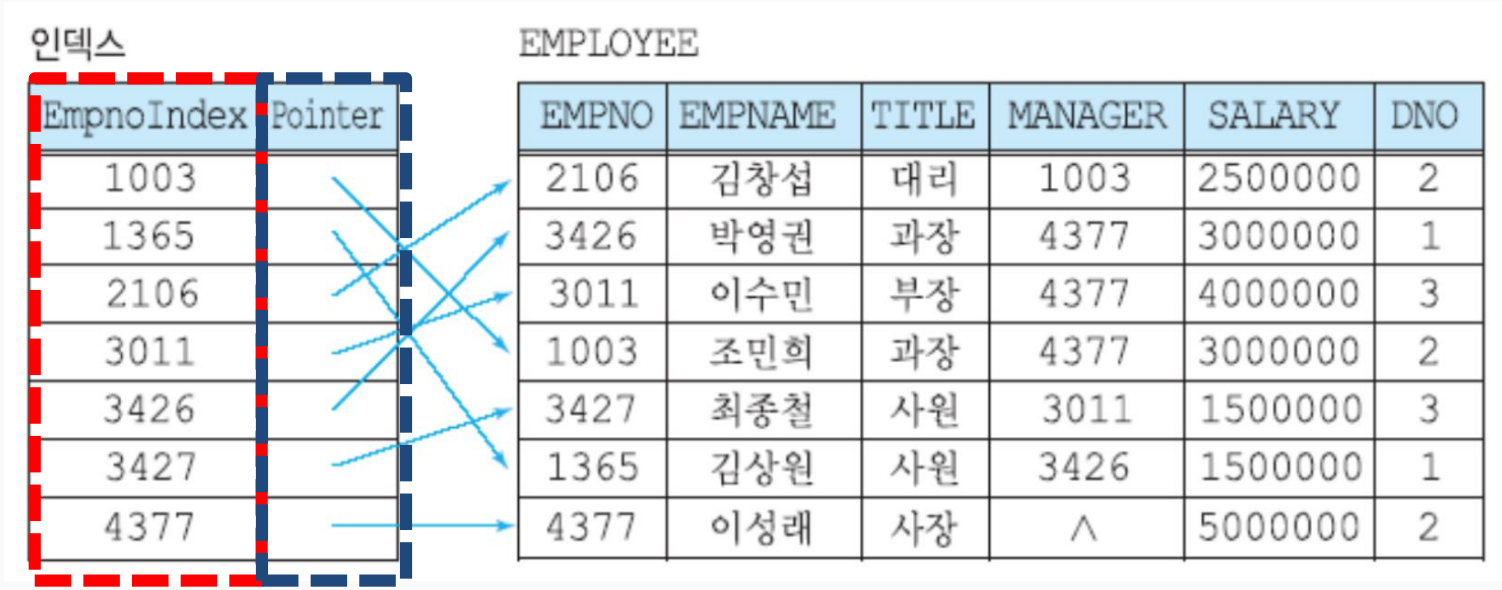


특정 컬럼에 인덱스를 생성하면, 해당 컬럼의 데이터들을 정렬하여 별도의 메모리 공간에 데이터의 물리적 주소와 함께 저장된다.



한글

책의 색인



데이터베이스 index

왜 사용할까?

테이블에 대한 검색 성능을 높이기 위해서!

데이터베이스에서 테이블의 모든 내용을 검색하면 시간이 오래 걸리기 때문
인덱스의 가장 큰 특징인 정렬된 데이터를 이용해 조건 검색을 할 때 빠르게
검색할 수 있다.

왜 사용할까?

1. 조건 검색 Where 절의 효율성
2. 정렬 Order by 절의 효율성
3. MIN, MAX의 효율적인 처리

1. 조건 검색 Where 절의 효율성

인덱스를 사용하면 데이터가 **정렬**되어 있기 때문에 테이블을 모두 스캔하지 않아도 해당 조건에 맞는 데이터를 빠르게 찾아낼 수 있다.

2. 정렬 order by 절의 효율성

order by에 의한 sort 과정을 피할 수 있다.
order by는 굉장한 부하가 걸리는 작업임.
하지만 인덱스를 사용하면 이미 정렬이 되어있기 때문에 데이터를 가져오기만 하면 된다.

3. MIN, MAX의 효율적인 처리

데이터가 정렬되어 있기 때문에 레코드의 시작값과 끝값만 가져오면 되기 때문에 효율적으로 찾을 수 있다.

그럼 **검색 기능을 향상시키기 위해 항상 Index를 사용해야 할까?** **놉!**

인덱스는 항상 정렬된 상태를 유지한다.

∴ 원하는 값을 탐색하는데는 빠르지만, 새로운 값을 추가/삭제/수정하는 경우 쿼리문의 실행 속도가 느려진다.

인덱스의 관리

인덱스가 적용된 컬럼에 INSERT, UPDATE, DELETE가 수행된다면 계속 정렬을 해주어야 하고, 그에 따른 부하가 발생한다.
이러한 부하를 최소화하기 위해 인덱스는 데이터 삭제를 인덱스를 사용하지 않는다라는 작업으로 대신한다.

인덱스의 관리 - DELETE & UPDATE

DELETE : 인덱스에 존재하는 값을 삭제하지 않고, 사용하지 않는다 라는 표시를 함

UPDATE : 인덱스에 존재하는 값을 수정하지 않고, 사용하지 않는다 라는 표시와 함께 갱신된 데이터에 대한 값을 인덱스에 새로 추가해준다.

인덱스 언제 사용?

삽입, 삭제, 업데이트보다 조회가 많이 일어나는 테이블
중복되는 데이터가 적은 테이블

인덱스 생성 전략

인덱스를 가장 효율적으로 사용하려면 데이터의 분포도는 최대한으로,
조건절에 자주 사용되는 컬럼을 인덱스로 생성하는 것이 좋다.

<u>가장 최선은 PK로 인덱스를 거는 것!</u>

* 데이터의 분포도
분포도가 좋다 == 해당 컬럼의 유니크한 데이터 종류가 많다.
분포도 값이 낮으면 분포도가 좋은 것 (https://jdm.kr/blog/169)

인덱스 생성 전략

1. 중복되는 데이터가 최소한인(분포도가 좋은) 컬럼
2. 조건절에 자주 등장하는 컬럼
3. LIKE 보다는 = 으로 비교되는 컬럼
4. ORDER BY 절에서 자주 사용되는 컬럼
5. JOIN 조건으로 자주 사용되는 컬럼

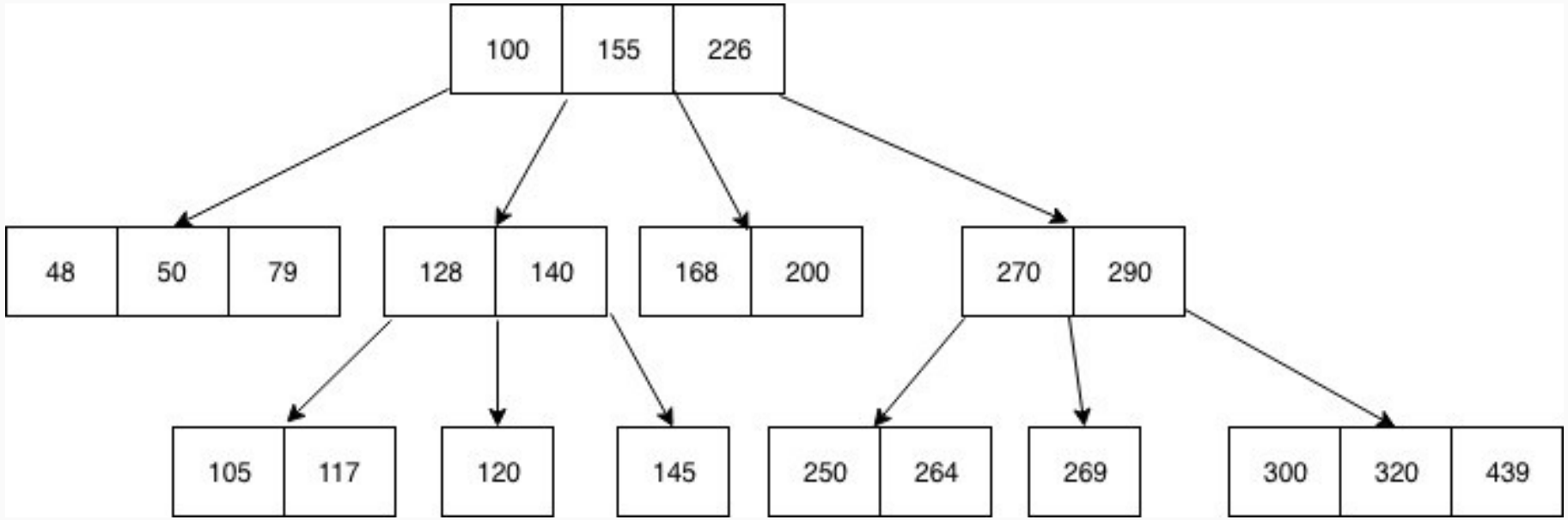
인덱스의 자료구조

B-Tree 인덱스

일반적으로 사용되는 인덱스 구조

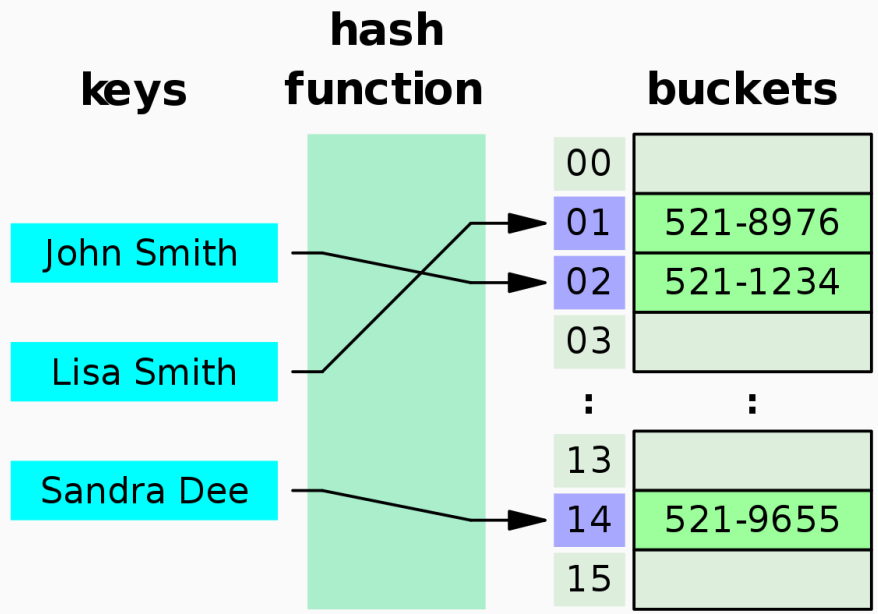
컬럼의 값을 변형하지 않고, 원래의 값을 이용해 인덱싱하는 구조이다.

일반적으로 O(logN)을 보장한다.



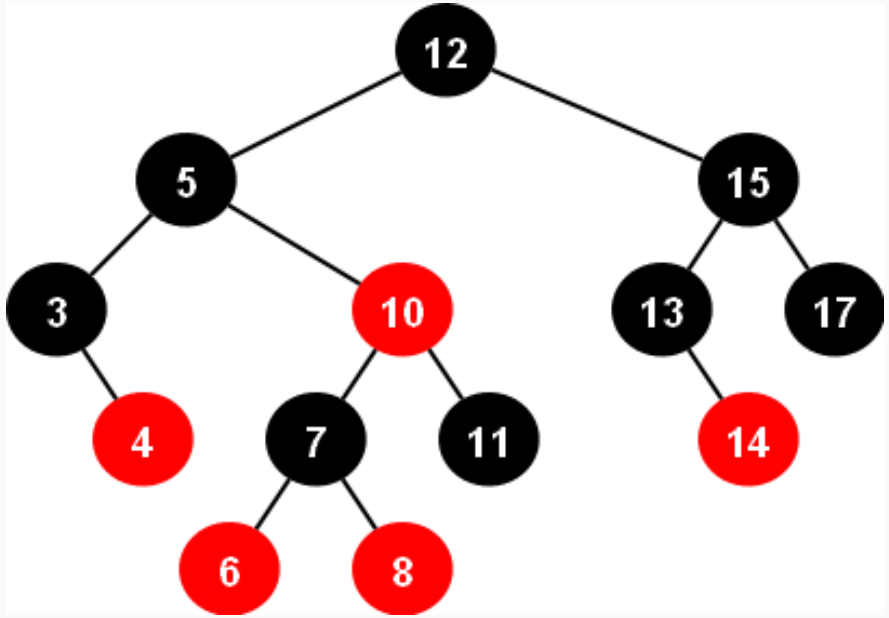
해시 테이블?

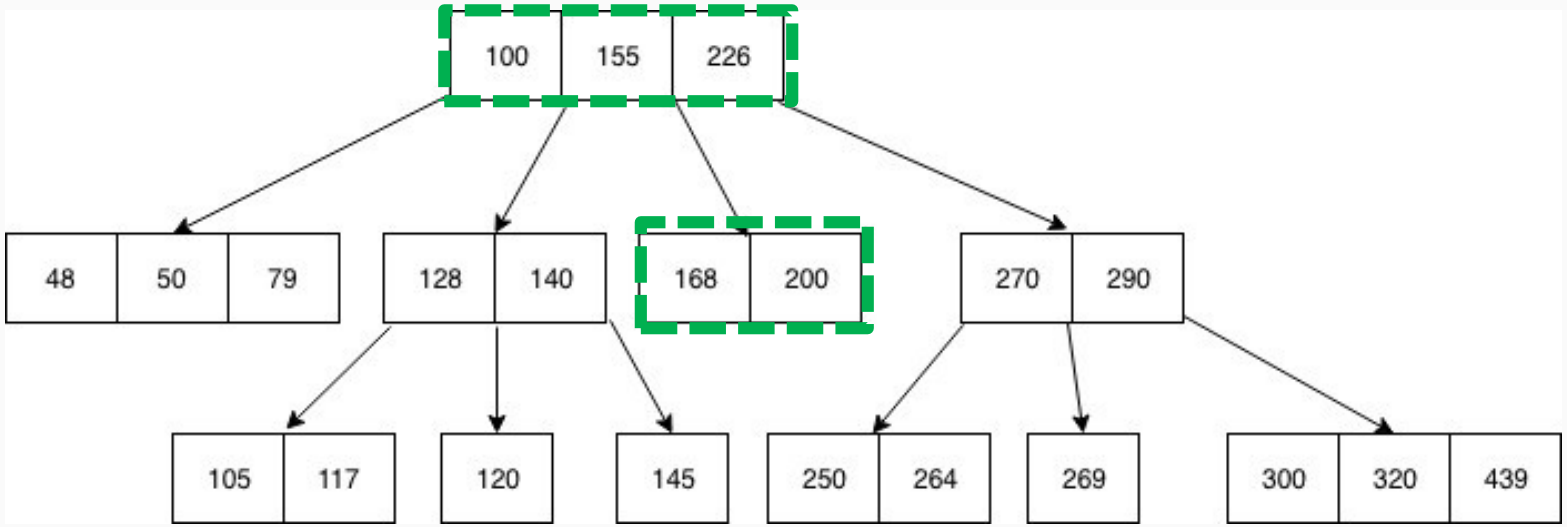
해시 함수를 통해 나온 해시 값을 이용하여 저장된 메모리 공간에 한번에 접근 한다. 따라서 탐색 시간은 O(1) 이다.
하지만, 모든 값이 정렬되어 있지 않기 때문에 특정 값보다 크거나 작은 값을 찾을 때는 시간 복잡도를 보장할 수 없다.

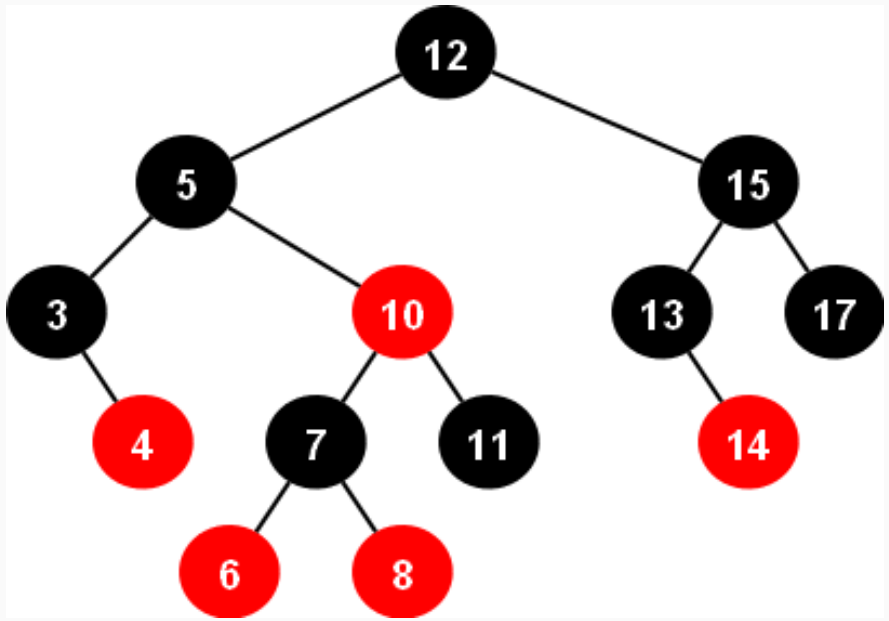


RedBlack-Tree

각 노드는 하나의 값만 가진 상태로 좌,우 자식 노드의 개수 밸런스를 맞춘 트리
빨간색,검은색 노드로 구분되어 있는 것은 삽입/삭제 작업을 할 때마다 규칙에 맞게 재정렬 하기 위한 수단이다.




100
155
226
48
50
79
128
140
168
200
270
290
105
117
120
145
250
264
269
300
320
439


12
5
15
3
10
13
17
4
7
11
14
6
8

참조 포인터 접근의 수

모든면에서 인덱스용으로 적합한 B-Tree

1. 항상 정렬된 상태로 특정 값보다 크고, 작은 부등호 연산에 문제가 없다. (해시 테이블 x)
2. 참조 포인터 개수가 적어 방대한 데이터 양에도 빠른 메모리 접근이 가능하다. (RedBlack-Tree x)
3. 데이터 탐색뿐만 아니라 저장,수정,삭제에도 항상 O(logN)의 시간 복잡도를 가진다. (배열 x)

인덱스란?

인덱스를 왜 사용할까?

언제 사용해야할까

인덱스의 자료구조