Indexing

인덱스란

- 추가적인 쓰기 작업과 저장 공간을 활용하여 데이터베이스 테이블의 검색 속도를 향상 시키기 위한 자료구조
- 인덱스 사용 시 SELECT, UPDATE, DELETE 성능도 함께 향상

인덱스의 관리

- DBMS는 인덱스를 항상 정렬된 상태로 유지해야 원하는 값을 빠르게 탐색 가능
- 때문에 INSERT, DELETE, UPDATE의 명령문이 들어오게 되면 index 관련 연산을 추가로 진행
    - INSERT : 새로운 데이터에 대한 인덱스 추가
    - DELETE : 삭제하는 데이터의 인덱스를 사용하지 않는다는 작업을 진행
    - UPDATE : 기존의 인덱스를 사용하지 않음 처리하고, 갱신된 데이터에 대한 인덱스 추가

인덱스 장/단점

- 장점
    - 테이블을 조회하는 속도와 그에 따른 성능 향상이 가능
    - 전반적인 시스템의 부하를 줄일 수 있음
- 단점
    - 인덱스를 관리하기 위해 DB의 약 10%에 해당하는 저장공간이 필요
    - 인덱스를 관리하기 위해 추가 작업 필요
    - 잘못 사용하는 경우(중복 값이 많거나, 삽입, 삭제가 빈번한 테이블) 오히려 성능이 저하된다

인덱스를 사용하면 좋은 경우

- 규모가 작지 않은 테이블

    - 작은 경우엔 Full Scan이나 인덱스 탐색이나 시간이 크게 차이 나지 않음
    - 즉 규모가 작을 경우 비슷한 시간에 메모리를 덜 사용하는 Full Scan이 효율적
- JOIN, WHERE, ORDER BY에 자주 사용되는 컬럼
- 데이터의 중복도가 낮은 컬럼

인덱스의 자료구조

- 해시 테이블
    - (Key, Value) 쌍으로 데이터를 저장하는 자료구조
    - 빠른 데이터 검색이 필요할 때 유용
    - Key 값을 이용해 고유한 인덱스 생성 후 해당 인덱스에 저장된 값을 꺼내오는 구조
    - 등호 = 연산에만 특화됨
- B+ Tree
    - DB 인덱스를 위해 자식 노드가 2개 이상인 B-Tree를 개선시킨 자료구조
        - 리프노드만 인덱스와 함께 데이터를 가지고 있음
        - 나머지 노드는 데이터를 위한 인덱스만 가지고 이음
        - 리프 노드들은 LinkedList로 연결
        - 데이터 노드 크기는 인덱스 노드 크기와 같지 않아도 됨
    - 부등호를 이용한 연산 시 Tree 탐색을 통해 효율적으로 연산 가능