Section5. 인덱스

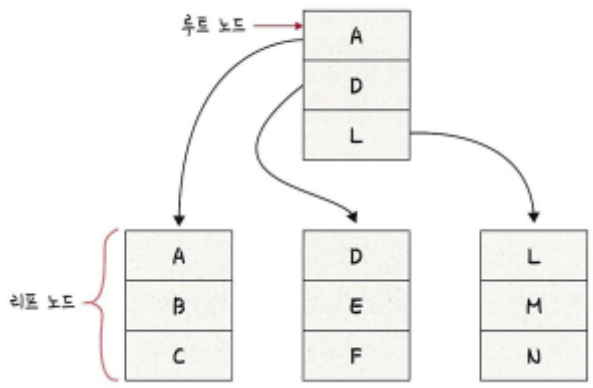
4.5.1 인덱스의 필요성

- 원하는 데이터를 빠르게 찾기위한 장치.
- DB의 테이블 내에서 내가 찾고자하는 데이터를 빠르게 찾을 수 있게 해준다.

4.5.2 B-트리

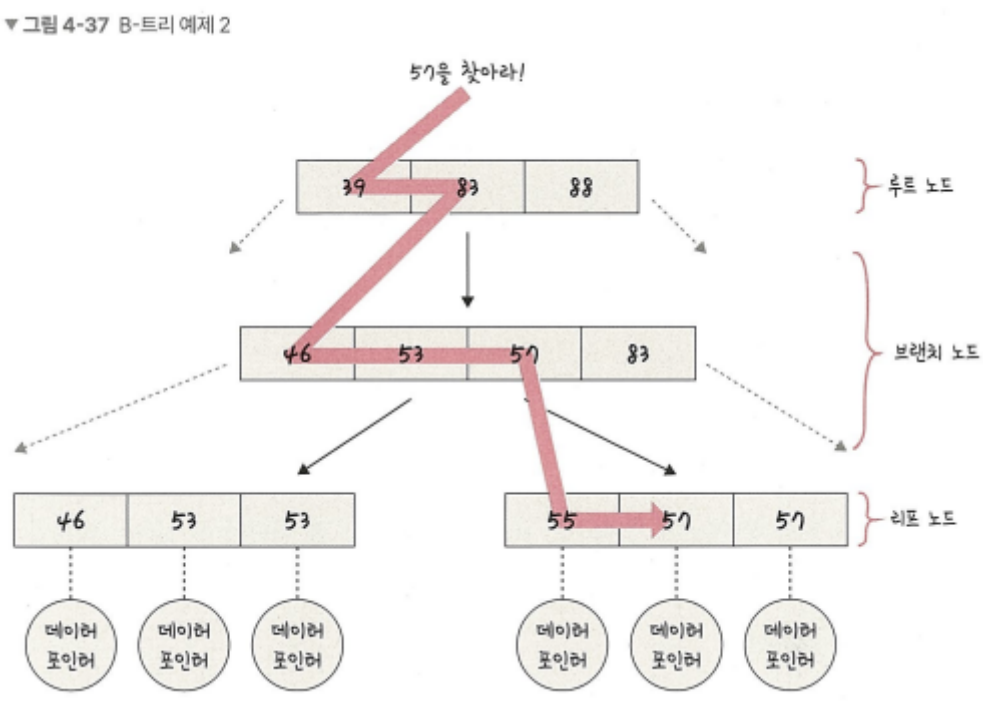
- B트리는 인덱스를 구성하는 자료구조
- 루트 노드, 리프 노드 ,브랜치노드(루트노드와 리프노드 사이의 노드)로 나뉨

▼그림 4-36 B-트리 예제 1



- 위 예제에서, E를 찾으려면 2번만 탐색하면된다.
  - 일반적인 경우라면 A,B,C,D,E로 5번을 탐색해야 한다.

▼그림 4-37 B-트리 예제 2



- 인덱스가 효율적인 이유 !
  - 효율적인 단계를 거쳐,
    - 모든 요소에 접근할 수 있는 균형잡힌 트리 구조
    - 트리 깊이의 대수확장성
      - 이때, 대수확장성이란, 트리 깊이가 리프 노드 수에 비해 매우 느리게 성장하는 것을 의미.
      - 기본적으로 한 depth 씩 증가할 때마다, 최대 인덱스 항목의 수는 4배씩 증가

▼ 표 4-3 트리의 대수확장성

트리 깊이	인덱스 항목의 수
3	64
4	256
5	1,024
6	4,096
7	16,384
8	65,536
9	262,144
10	1,048,576

- 즉, 10개 깊이로도 100만개의 레코드를 검색할 수 있음.

4.5.3 인덱스 만드는 방법

- **MySQL**
  - 인덱스 종류
    - 클러스터형 인덱스만드는 방법
      - primary key 옵션으로 기본키 생성 시
      - 기본키로 만들지 않고, unique not null 옵션을 붙여서 생성
    - 세컨더리 인덱스
      - 보조인덱스로 여러개의 필드 값을 기반으로 쿼리를 많이 보낼 때 생성해야 함.
- **MongoDB**
  - 도큐먼트 만들면 자동으로 ObjectID가 생성됨.
    - 해당키가 기본키로 설정됨
  - 세컨더리키도 부가적으로 설정해서, 기본키와 세컨더리키를 같이 사용하는 복합 인덱스를 설정할 수 있음.

4.5.4 인덱스 최적화 기법

- 인덱스 최적화
  - DB마다 조금씩 다르지만, 기본적인 골조는 같다.

1. 인덱스는 비용이다.
   a. 인덱스는 두번 탐색하도록 강요한다.
   b. 인덱스 리스트, 컬렉션 순으로 탐색하기 때문에, 관련 읽기 비용이 든다.
   c. 컬렉션 수정 시, 인덱스도 수정되어야 한다.
   d. 수정과정에서는 B-tree의 높이를 균형있게 조절하는 비용, 데이터를 효율적으로 조회하도록 분산시키는 비용도 소모된다.
   e. 따라서 !
      - 쿼리에 있는 필드에 인덱스를 무작정 다 설정하는 것은 답이아니다.
      - **컬렉션에서 가져와야 하는 양이 많을 수록 인덱스를 사용하는 것은 비효율적이다.**
2. 항상 테스팅하라
   a. 인덱스 최적화 기법은 서비스 특징에 따라 달라짐.
   b. 서비스에서 사용하는 객체의 깊이, 테이블 양이 다르기 때문.
   c. **explain() 함수**
      - 인덱스를 만들고 쿼리를 보낸 이후에 테스팅하며 걸리는 시간을 최소화 해야 함.

```
EXPLAIN
SELECT * FROM t1
JOIN t2 ON t1.c1 = t2.c1
```

3. **복합 인덱스는 같음, 정렬, 다중 값, 카디널리티 순이다.**
    - 보통 여러 필드를 기반으로 조회 시, 복합 인덱스를 생성.
    - 이때, 인덱스 생성 순서가 있고, 생성 순서에 따라 성능이 달라짐.
    - 같음, 정렬, 다중값, 카디널리티 순으로 생성해야 한다.
        1. ==이나 equal
        2. 정렬에 사용됨
        3. 다중값을 출력해야 하는 필드, 즉 쿼리 자체에 > < 등 많은 값을 출력해야하는 쿼리에 쓰이는 필드는 나중에 인덱스를 설정해야함.
        4. 유니크한 값의 정도를 카디널리티라고함.
            a. 카디널리티가 높은 순서를 기반으로 인덱스를 생성해야함.