5.2 인덱스란?

생성일	@2021년 6월 13일 오후 5:23
태그	

DBMS가 데이터를 검색해서 원하는 결과를 가지고 올때 걸리는 시간을 줄이기 위해서 `칼럼의 값`과 해당 레코드가 `저장된 주소`를 키와 값의 `pair`로 인덱스를 만들어둔다.

⇒ 이 인덱스는 칼럼의 값을 주어진 순서로 미리 정렬해서 보관한다

자료구조의 관점에서 보는 인덱스

- SortedList : DBMS의 인덱스와 같은 자료구조
  - 저장된 값을 항상 정렬된 상태로 유지
  - 장점: 이미 정렬이 되어있어서 원하는 값을 빨리 찾을 수 있음
  - 단점: 데이터가 저장될 때마다 항상 값을 정렬해야 해서 저장하는 과정이 복잡하고 느림. (DML 구문의 처리가 늦어짐, BUT SELECT는 인덱스로 인해 속도가 빠름)
- ArrayList : 데이터 파일과 같은 자료구조
  - 값이 저장되는 순서대로 그대로 유지

인덱스의 성능

- 데이터의 저장(DML) 성능을 희생하고 READ 성능을 높이는 기능
- 인덱스 추가 → 데이터 저장 속도를 어디까지 희생할 수 있는지, 읽기 속도를 얼마나 더 빠르게 만들어야 하는지 여부에 따라 결정
- 인덱스를 많이 생성하면 데이터 저장성능이 떨어지고, 인덱스 크기가 비대해져서 역효과가 날 수 있음

인덱스의 역할별 구분

1) 프라이머리 키

: 그 레코드를 대표하는 칼럼의 값으로 만들어진 인덱스를 의미 (=식별자)

NULL값을 허용하지 않고, 중복을 허용하지 않음

2) 보조 키 (Secondary Key)

프라이머리 키를 제외한 나머지 모든 인덱스는 보조 인덱스로 분류

유니크 인덱스 → 프라이머리 키와 성격이 비슷 (대체해서 사용할 수 있어서 대체 키라고도 함)

데이터 저장방식(알고리즘) 별 구분

1) B-Tree 인덱스 : 가장 일반적으로 사용되는 인덱스 알고리즘

- 칼럼의 값을 변형하지 않고 원래의 값을 이용해 인덱싱하는 알고리즘

2) Hash 인덱스 : 칼럼의 값으로 해시 값을 계산해서 인덱싱하는 알고리즘

- 속도가 매우 빠름
- 값을 변형해서 인덱싱하므로 Prefix일치와 같이 값의 일부만 검색하고자 할때는 Hash 인덱스 사용불가
- 주로 메모리 기반의 데이터베이스에서 많이 사용

3) Fractal-Tree 인덱스 : B-Tree 의 단점을 보완해서 데이터 저장/삭제 될때 처리 비용을 상당히 줄일 수 있다

- 값을 변형하지 않고 인덱싱 (B tree와 거의 비슷)

유니크 인덱스

인덱스가 유니크 = 같은 값이 1개만 존재

→ 유니크 인덱스에 대해 = 이퀄 조건으로 검색한다는 것은 1개의 레코드만 찾으면 더 찾지 않아도 된다 - 라는 뜻을 옵티마이저에게 알려주는 효과