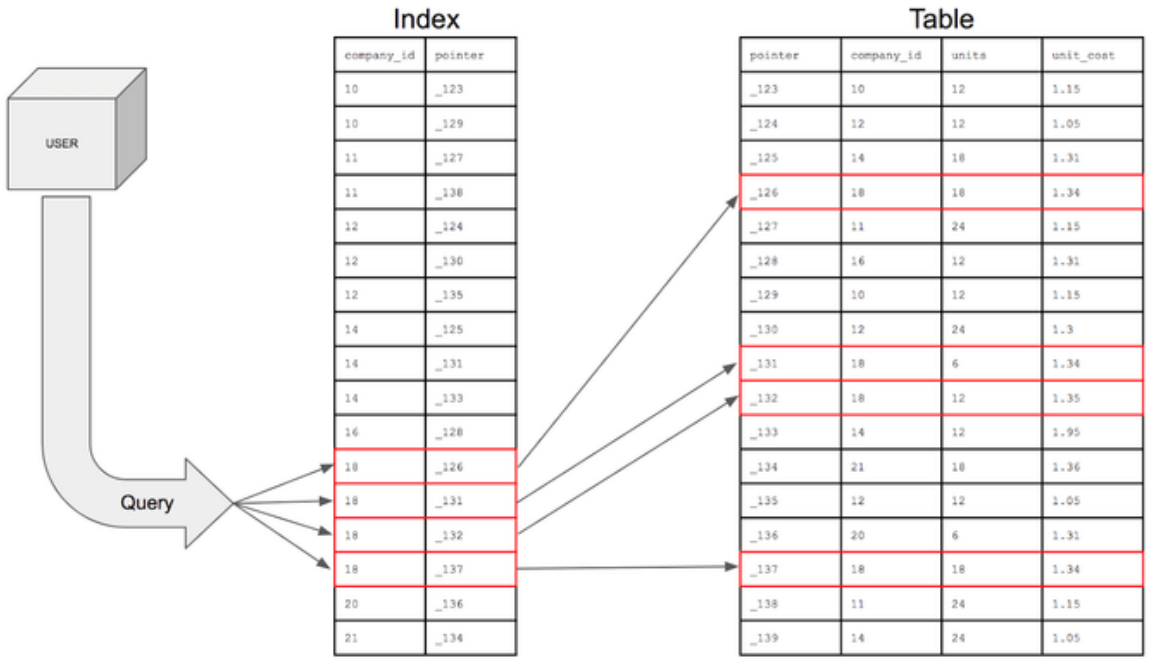
Indexing

태그	DB
주차	2주차

📌 목차

1. Index



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

1-1. Index의 관리

- DBMS는 index를 항상 최신의 정렬된 상태로 유지해야 함
- 따라서, `INSERT`, `DELETE`, `UPDATE`가 수행되는 경우에 추가 연산 필요(오버헤드 추가 발생)
  - `INSERT`: 새로운 데이터에 대한 인덱스 추가
  - `DELETE`: 삭제하는 데이터의 인덱스를 사용하지 않는다는 작업 진행
  - `UPDATE`: 기존의 인덱스를 사용하지 않음 처리하고 갱신된 데이터에 대한 인덱스 추가

> 💡 사용되지 않는 인덱스는 바로 제거해줘야 함!

1-2. 동작 원리

1. 해당 테이블을 생성 시 생성하고 싶은 인덱스 칼럼 지정
2. 생성 후 Index 조회 시, WHERE 절이 포함된 쿼리로 조회
3. Index로 저장된 Key-Value값을 참조해서 결과 출력

1-3. 장단점

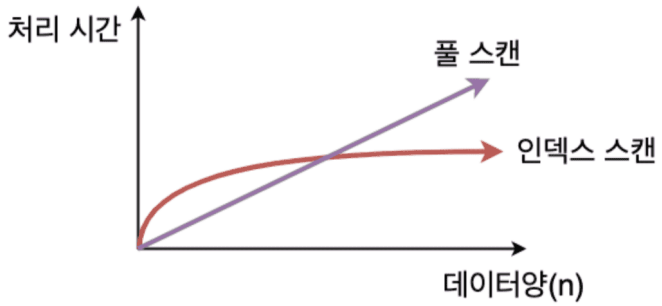
- 장점
  - 테이블을 조회하는 속도와 그에 따른 성능 향상
  - 전반적인 시스템의 부하 감소
- 단점
  - 인덱스를 관리하기 위해 DB의 약 10%에 해당하는 저장공간 필요
  - 인덱스를 관리하기 위해 추가 작업 필요
  - 인덱스를 잘못 사용하면 오히려 성능 저하

> ⚠ CREATE, DELETE, UPDATE가 빈번한 속성에 인덱스를 걸게 되면 인덱스 크기가 비대해져 성능이 오히려 저하되는 역효과 발생

1-4. Index를 사용하면 좋은 경우

- 규모가 작지 않은 테이블
- `INSERT`, `UPDATE`, `DELETE`가 자주 발생하지 않는 Column
- `JOIN`이나 `WHERE` 또는 `ORDER BY`에 자주 사용되는 Column
- 데이터 중복도가 낮은 Column
- 등등

1-5. Index Scan VS Full Scan



- 데이터 양이 많을 수록 Index Scan이 유리
- **Index Scan** 방식 B-Tree에서 leaf node까지 찾아 내려간 후 해당 데이터를 찾기 위해 disk로 접근
- 그러나 **Full Scan** 방식은 바로 disk로 가서 모든 데이터를 읽음
- 따라서 데이터 양이 많지 않거나, Index가 효율적으로 설정되지 않으면 Table Full Scan이 더 빠름!
- Index Scan으로 탐색하는 데이터가 테이블의 총 데이터 수와 비슷한 경우에도 Table Full Scan이 더 빠를 수도 있음

1-6. Index 사용시 주의사항

1) 칼럼의 가공

- 데이터를 가공하여 비교하는 경우 Index 검색이 불가능

```
WHERE price * 0.9 > 10000 --case1
WHERE price > 10000 / 0.9 --case2
```

2) 부정형 (!=)

- 부정형 `WHERE` 절 사용시 해당 데이터를 제외한 모든 데이터를 검색해야 함
- Full Scan으로 동작

3) like 앞 %

- B-Tree는 데이터의 첫 글자를 기준으로 정렬 후 저장

- 그러나 `like` 쿼리를 통해 앞에 %를 붙이는 경우 모든 데이터르 ㄹ찾아야 함
- Full Scan으로 동작

4) count(*)

- `count(*)`는 모든 데이터를 한바퀴 돌아야 알 수 있음
- Full Scan으로 동작

5) 멀티 칼럼에서 두 번째 칼럼을 조건으로 사용하는 경우

- 예를들어 age, name으로 멀티 칼럼 index가 설정된 경우, B-Tree는 age로 정렬한 후 name으로 정렬한 데이터를 저장함
- 그러나 name만을 조건으로 사용하면 Index 검색 불가능
- 반대로 age만 조건으로 사용하면 검색 가능

```
-- (age, name)으로 Indexing된 경우
WHERE age, name FROM ...  -- Index Scan
WHERE age FROM ...        -- Index Scan
WHERE name FROM ...       -- Full Scan
```

6) 멀티 칼럼 Index에서 순서를 바꾸는 경우

- 5)와 비슷한 이유로 Index 검색 불가능

```
-- (age, name)으로 Indexing된 경우
WHERE age, name FROM ...  -- Index Scan
WHERE name, age FROM ...  -- Full Scan
```

> 💡 DB의 Optimizer가 칼럼 순서를 바꿔서 조회하면 Index 검색이 되기는 한다.
> 그러나 Optimizer가 없다면 Full Scan!

---

2. Index의 종류

2-1. Clustered Index

- 개발자가 설정하는 Index가 아닌, MySQL이 자동으로 설정하는 Index
  1. 해당 테이블에 Auto increments 값으로 PK가 있다면, 해당 칼럼이 Clustered Index가 됨
  2. 그러한 PK가 없다면, 칼럼 중 Unique 칼럼을 Clustered Index로 선정됨
  3. Unique 칼럼도 없다면, MySQL 내부적으로 Hidden Clustered Index Key (row ID)를 만들어 사용

> 💡 **MySQL에서 Clustered Index Key로 사용하는 조건의 공통점**
>
> - 테이블의 `count(*)`와 `distinct key`한 값이 같음.
>   모든 row를 통틀어 중복이 적을 수록 Index 효율이 높기 때문
> - 때문에 MySQL에서 자동으로 설정되는 Clustered Index는 최대 효율을 위해 중복이 최대한 발생하지 않는 칼럼 사용

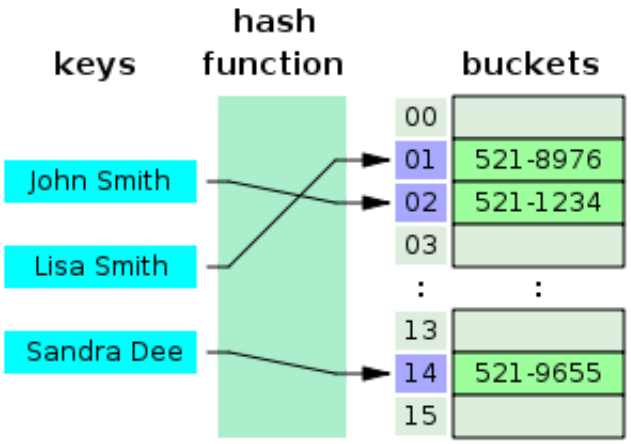
2-2. Non-Clustered Index

- 개발자 또는 DBA등이 설정하는 모든 Index
- 멀티 칼럼 Index의 경우 최대 16개의 칼럼 사용 가능
- 테이블당 최대 64개까지 지정 가능 (Clustered Index까지 총 65개)

---

3. Index의 자료구조

3-1. 해시 테이블(Hash Table)



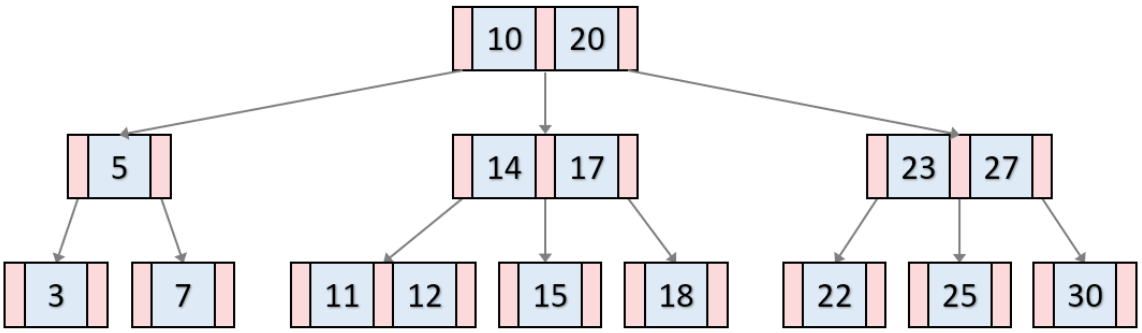
💡 **특징**

- (Key, Value)로 데이터를 저장하는 자료구조
- Key값을 이용해 고유한 index를 생성하여 해당 index에 저장된 값을 꺼내오는 방식

- 장점
  - 빠른 데이터 검색이 필요할 때 유용 (시간복잡도 O(1))
- 단점
  - 등호(=) 연산에만 특화되어 부등호 연산(>, <)이 자주 사용되는 DB 검색에는 적합하지 않음
    - 값이 조금만 달라져도 해시값이 크게 달라지기 때문

    ex) 특정 문자열로 시작하는 데이터를 검색하는 쿼리문은 인덱스 활용 불가

3-2. B-Tree

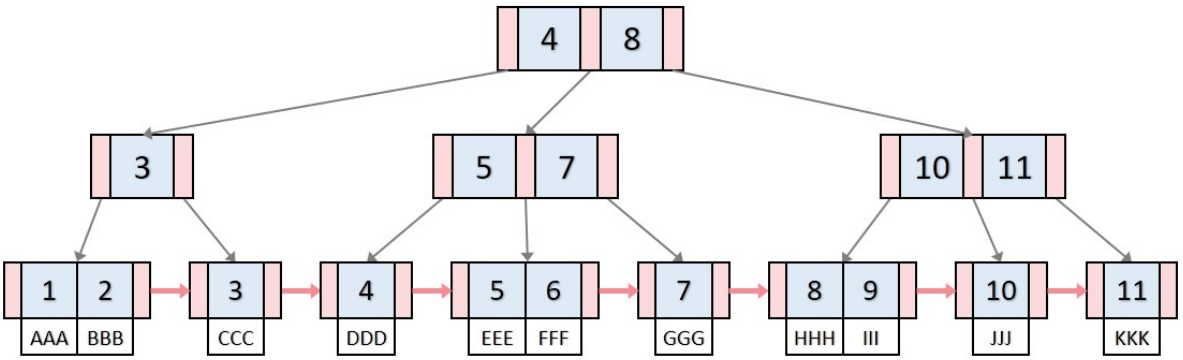


💡 **특징**

- Key값을 이용해 찾고자 하는 데이터를 트리구조를 이용해 탐색
- Binary Search와 유사하나, 한 노드당 자식 노드가 2개 이상 가능
  - 노드에는 최소 [M/2]개 ~ 최대 M개의 자식노드 가능
  - 노드에는 최소 [M/2]-1개 ~ 최대 M-1개의 KEY 가능
- 노드에 KEY는 항상 정렬된 상태로 유지됨
- ~~B-Tree의 'B'는 어떤 의미인지 알려진 바가 없음. 'Balanced'의 B로 추측만..~~

- 장점
  - 어떤 값에 대해서도 같은 시간에 결과를 얻을 수 있음 (균일성)
  - 모든 노드에 데이터 저장 가능
  - 데이터가 정렬된 상태로 유지
  - 키 중복이 없음
    - B+Tree는 리프 노드에 모든 데이터가 있기 때문에 중복 존재
- 단점
  - 한 노드에 한 데이터가 담겨 있어 트리의 높이가 높아짐
  - 트리의 노드가 수정되면 재정렬을 해야 함

3-3. B+Tree



💡 **특징**

- 자식 노드가 2개 이상인 ***B-Tree***를 개선시킨 자료구조
- 리프노드만 인덱스와 함께 데이터(Value)를 가지고 있음
  - 나머지 노드들은 인덱스만을 가짐
- 리프노드들은 LinkedList로 연결되어 있음
- 데이터 노드 크기는 인덱스 노드 크기와 같지 않아도 됨
- B-Tree와 다르게 Best/Worst Case가 없음

- 장점
  - 부등호를 이용한 순차 검색 연산에 유리함
    - 리프노드가 LinkedList 형식이기 때문
  - 한 노드에 여러 데이터를 담을 수 있어 트리의 높이가 낮음
  - 모든 노드 순회시 리프노드에서 선형탐색 가능
- 단점
  - B-Tree와 달리 무조건 리프노드까지 탐색해야 함
  - 해시테이블의 시간복잡도 O(1)에 비해 O(logN)이 걸림