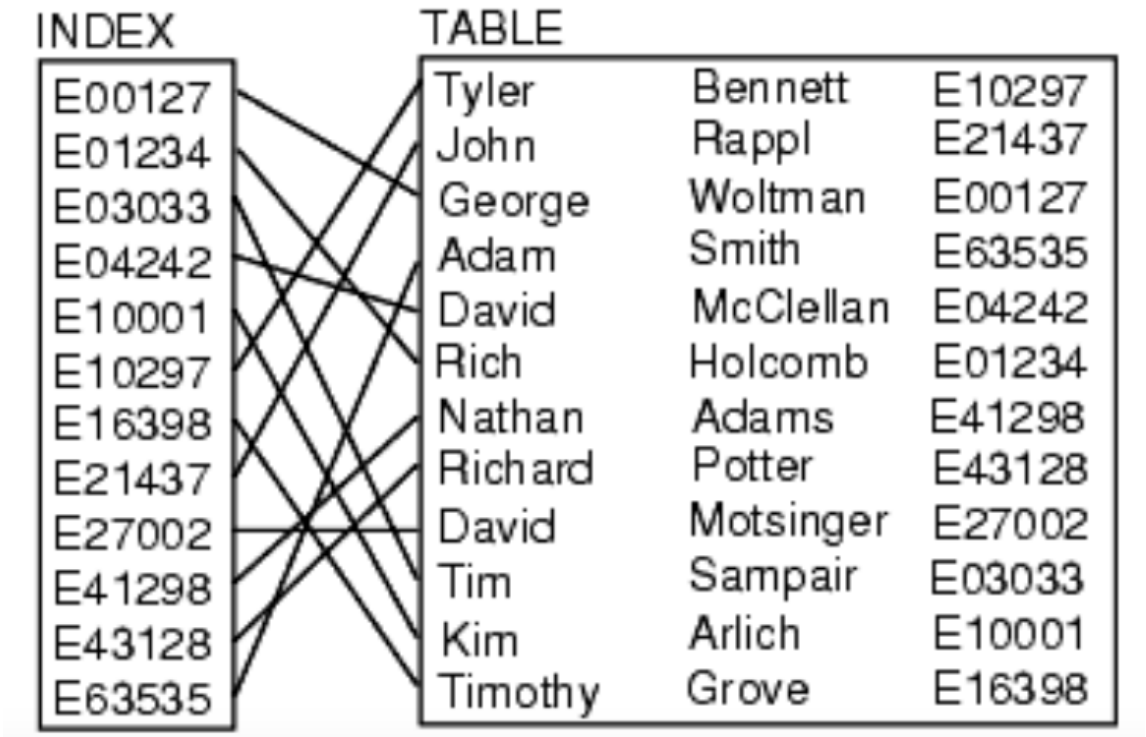
DB index_2022.04.03

▼ index란?

- 책 뒤쪽의 색인(index) 기능이라고 할 수 있음.
- 찾고자 하는 부분(페이지) 빠르게 찾아주는 기능 수행 = 정보 검색 최적화

▼ DB index란?

- DB에서 키값(key-value)으로 실제 데이터 위치 (C의 pointer와 비슷)를 식별하고 데이터 접근 속도를 높이고자 생성되는, 키 기준으로 정렬된 오브젝트 (보통 B-Tree 인덱스 사용). 그러한 자료 기술 테크닉.
- 인덱스가 생성되면 테이블과 매핑된 또 다른 테이블이 하나 생성된다고 생성된다고 생각하면 됨. 인덱스는 정렬되어 저장됨. (다른 테이블 데이터는 물리적으로 흩어져서 저장됨)
- DB에 왜 index 필요?
    - EMP_ID (https://www.progress.com/tutorials/odbc/using-indexes)



- **테이블 조회 속도 및 성능 향상**: 객체지향 포인터처럼 인덱스가 해당 테이블 블럭(블럭은 데이터가 저장되는 최소 단위)에 주소 가짐.
- 만약 테이블 컬럼 수가 많다면 하나의 블럭 에 저장되는 row 수가 적어지나 컬럼 수 적으면 하나의 블럭에 저장되는 row 수가 많아짐. (?)
- 특정 조건, **where**절에 자주 등장하는 컬럼을 인덱스로 설정해주면 효율적.
- **order by** 절에 자주 등장하는 컬럼을 인덱스로 지정해 놓으면 인덱스는 소팅이 되어서 저장되어 있다고 했기 때문에 order by 절에 자주 등장하는 컬럼을 인덱스로 지정해 놓으면 인덱스 바로 꺼내면 됨.
- MIN, MAX의 효율적인 처리가 가능: MIN값과 MAX값을 레코드의 시작값과 끝 값 한건씩만 가져오면 됨.

▼ 기본키 (PK, Primary Key) VS 인덱스

- 기본키(PK, Primary Key)는 index 역할도 수행하므로 기본 키 활용 인덱싱 가능
- MySQL/Maria DB PK는 클러스터형 인덱스 (clustered index)로 작동함.

- 클러스터형 인덱스란 기본키의 구성 열 순서를 기준으로 물리적인 스토리지에 데이터가 쌓인다는 뜻. 비슷한 PK 값들이 근거리에 적재되므로 PK를 활용하여 인덱스 스캔 수행하면 테이블 데이터에 더 빠르게 접근 가능.

▼ 인덱스 주의사항

- PK와 똑같은 인덱스 생성시 인덱스 저장되는 물리적 공간이 낭비되는 한편 데이터의 삽입(insert)/삭제(delete)/수정(update)에 따른 인덱스 정렬의 오버헤드 발생 (I_stu_no 인덱스는 불필요한 공간 낭비와 정렬의 오버헤드 발생하므로 삭제 해야 함)
- 생성된 인덱스 관리해주는 것 중요. 사용되지 않는 인덱스 바로 제거
  - INSERT: 새로운 데이터에 대한 인덱스 추가
  - DELETE: 삭제하는 데이터 인덱스 사용하지 않는다는 작업 진행
  - UPDATE: 기존 인덱스 사용하지 않음 처리, 갱신된 데이터 인덱스 추가

```
CREATE TABLE students (
  stu_no INT(11) NOT NULL,
    stu_name VARCHAR(14) NOT NULL,
    stu_dob DATE NOT NULL,
    stu_phone VARCHAR(16) NOT NULL,
    stu_major VARCHAR(3) NOT NULL,
    PRIMARY KEY (stu_no),
    INDEX stu_no_idx (stu_no)
 );
```

▼ 고유 인덱스(unique index) VS 비고유 인덱스(non-unique index)

- 고유 인덱스: 인덱스 구성하는 열들의 데이터가 유일하다는 의미 (서로 중복되지 않고 유일성 유지)
  - PK stu_no이나 연락처 자주 조회하는 경우

```
ALTER TABLE students
ADD UNIQUE INDEX stu_phone_idx(stu_phone);
```

- 기본키 VS 고유 인덱스
  - 기본 키와 고유 인덱스 모두 데이터의 유일성 보장하고 효율적인 데이터 접근 위한 인덱스 수단으로 사용됨.
  - **기본 키에는 NULL입력 할 수 없으나 고유 인덱스는 NULL 허용**

- 비고유 인덱스(non-unique index)
  - 고유 인덱스에서 데이터의 유일한 속성만 제외한 키. 흔히 사용됨. 데이터 신규 입력되어 인덱스 재 정렬되더라도 인덱스 열 중복 체크를 거치지 않고 단순한 정렬 작업 수행.
  - stu_name을 비고유 인덱스로 지정

```
ALTER TABLE students
ADD INDEX stu_name_idx(stu_name);
```

▼ 인덱스 장점

- 테이블 조회 속도 및 성능 향상, 전체적 시스템 부하 줄임
- **인덱스 손익 분기점**: 테이블 가지고 있는 **전체 데이터양의 10~15% 이내의 데이터가 출력**이 될 때만 인덱스를 타는게 효율적. 그 이상일 때 풀스캔이 빠름.

▼ 인덱스 단점

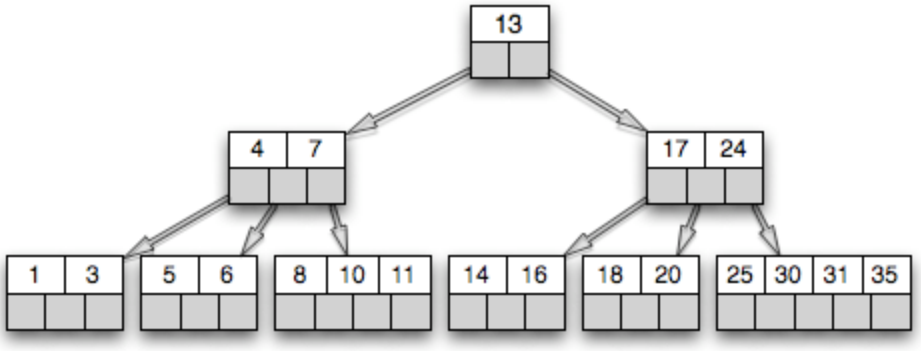
- **정렬된 상태를 계속 유지 시켜줘야 한다는 점이 가장 큰 단점**
- insert나 update, delete 경우 오히려 늦어질 수 있음. (인덱스 테이블에도 해야되므로..)
- 인덱스 설정시 신경써야함. (분별력 높은 id가 성별보다 앞에 오는게 나음)
- 관리 위해 DB의 약 10% 저장공간 필요
- 추가 작업 필요하며 잘못 사용시 (예) DELETE, UPDATE 의 경우 기존 인덱스 삭제하지 않고 처리하므로 인덱스가 실제 데이터 보다 커질 수 있음) 성능 저하

▼ 인덱스 사용이 용이한 경우

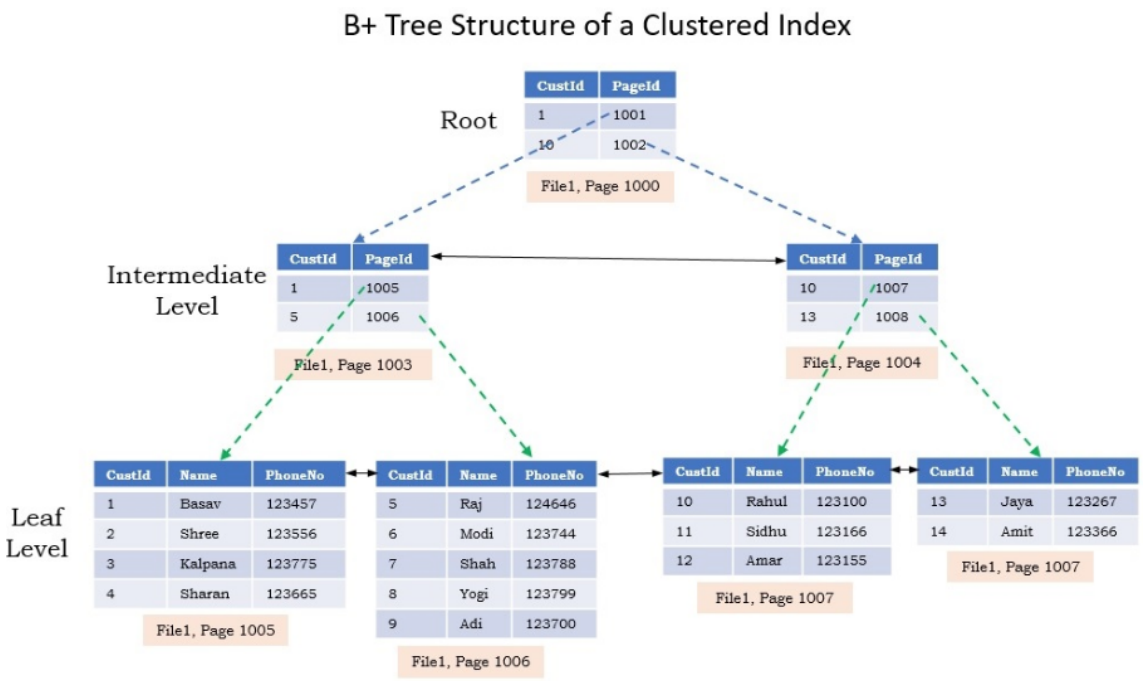
- 카디널리티(컬럼에 사용되는 값의 다양성 정도, 중복 수치) 높을수록(학번 > 이름),
- 선택도 낮을 수록(5~10% 적당)
- 데이터 중복도 중복도가 낮은 컬럼, 활용도가 높을수록 인덱스 설정에 좋은 컬럼
- 규모가 작지 않은 테이블
- INSERT, UPDATE, DELETE가 자주 발생하지 않는 컬럼
- 항상 = 으로 비교되는 컬럼
- JOIN이나 WHERE 또는 ORDER BY (굉장히 부하가 많이 걸리는 작업)에 자주 사용되는 컬럼

▼ 인덱스 자료구조

- 민규님 감사합니다! https://mangkyu.tistory.com/96
- Hash Table: 해시 인덱스의 경우 동등여부만 확인 가능하지 부등호(”20살 이하의 학생을 찾으세요.”)에 적합하지 않아 안쓰임.
- B * Tree 인덱스는 대부분의 DBMS 그리고 오라클에서 특히 중점적으로 사용하고 있는 가장 보편적인 인덱스



- B + Tree: DB 인덱스 위해 자식 노드가 2개 이상인 B-Tree 개선 시킨 자료구조. B-Tree Leaf 노드들 LinkedList로 연결 순차검색 용이하게 함.
- https://sqlhints.com/tag/clustered-index-b-tree-structure/



▼ References

- Blogs & Web pages
  - https://mangkyu.tistory.com/96
  - https://brunch.co.kr/@skeks463/25
  - https://coding-factory.tistory.com/746
  - https://www.progress.com/tutorials/odbc/using-indexes
  - https://www.geeksforgeeks.org/indexing-in-databases-set-1/
  - http://20bits.com/article/interview-questions-database-indexes
- Book: 업무에 바로 쓰는 SQL 튜닝, 한빛미디어
- Youtube: SQL전문가 정미나