데이터베이스 스터디 - 1주차

상태	시작 전
날짜	@2024년 6월 20일

1주차 물리적 데이터베이스 설계

> 💡 물리적 데이터베이스 설계의 핵심
>
> 물리적 데이터베이스 설계의 핵심은 데이터베이스 시스템의 성능, 저장 효율성, 보안 등을 최적화하기 위해 데이터의 물리적 저장 구조를 결정하는 과정

- 논리적인 설계의 데이터 구조를 보조 기억 장치상의 파일로
- 예상 빈도를 포함하여 데이터베이스 쿼리와 트랜잭션 분석
- 데이터에 대한 효율적인 접근을 제공하기 위해 저장 구조와 접근 방법들을 다룸
- 특정 dbms의 특성을 고려하여 진행
- 인덱스 구조를 잘 활용하는 것이 효율적

데이터들은 보조 기억 장치에 저장되어 있음

보조 기억 장치

- 사용자가 원하는 데이터를 검색하기 위해서 DBMS는 디스크 상의 데이터베이스로 부터 사용자가 원하는 데이터를 포함하고 있는 블록을 읽어서 주기억 장치로 가져온다.
- 데이터가 변경된 경우에 블록들을 디스크에 다시 기록한다
- 블록 크기는 512바이트 ~ 수 킬로 바이트
- 전형적인블록 크기는 4096 바이트
- 디스크는 데이터베이스를 장기간 보관하는 주된 보조 기억 장치

💡 주기억 장치

주기억 장치는 컴퓨터 내부에서 현재 cpu가 처리하고 있는 내용을 저장하는 기억 장치
용량이 크고 처리 속도가 빠르다.

cpu 명령에 의해 기억된 장소에 직접 접근해서 읽고 쓸수 있음

대표적 주기억 장치는 ROM, RAM

ROM(Read Only Memory)는 전원이 끊어져도 기록된 데이터가 소멸되지 않는 비휘발성 메모리, 오직 기억된 데이터를 읽기만 가능한 장치

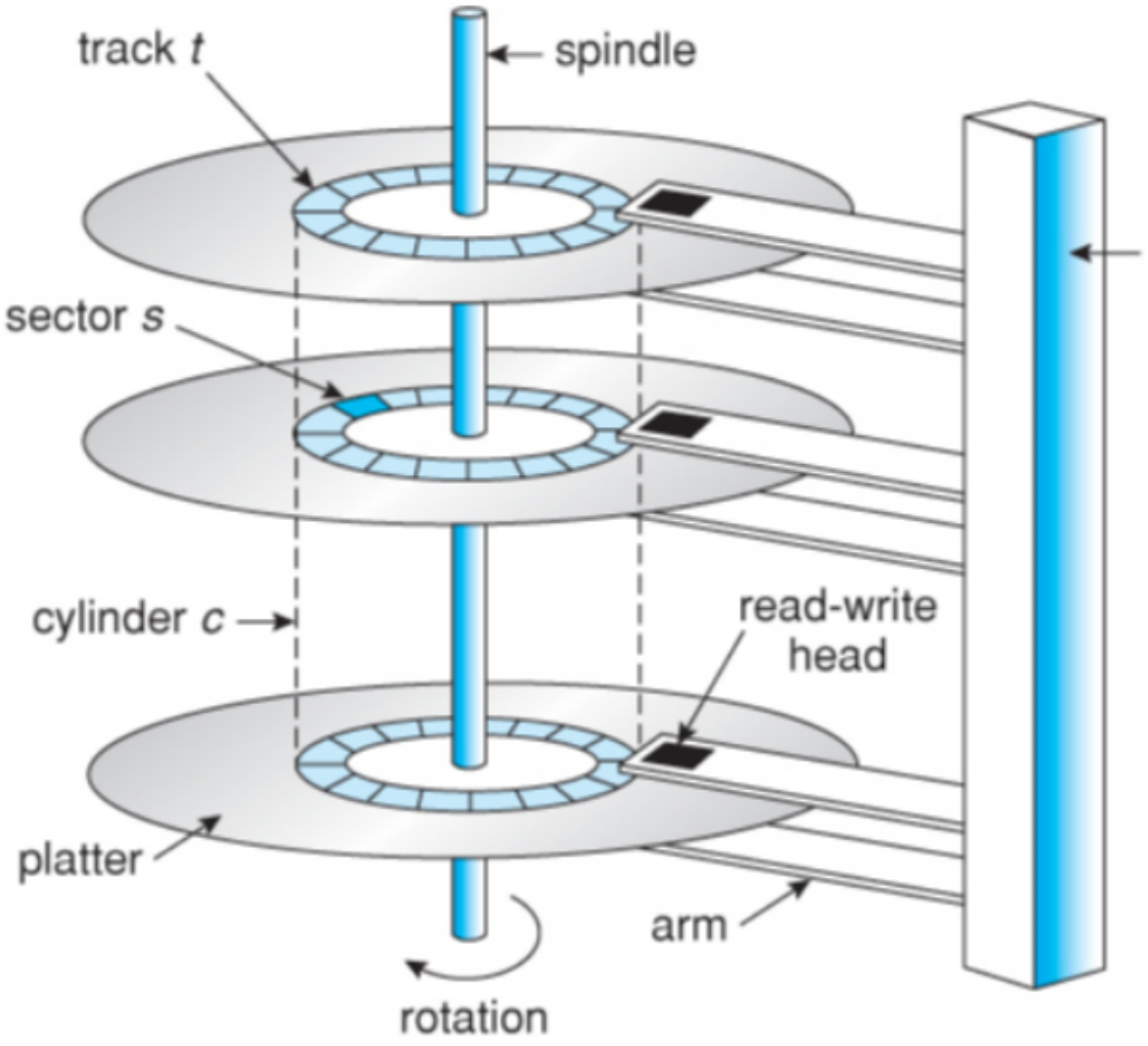
RAM(Random Access Memory)는 읽고 쓰기 가능하고 응용 프로그램, 운영 체제를 불러와 CPU가 작업할 수 있도록 하는 기억 장치, 전원이 끊어지면 데이터가 지워지기에 휘발성 메모리

💡 보조 기억장치

보조 기억 장치는 데이터를 영구적으로 저장하는 장치
하드 디스크 드라이브(HDD) : 기계적 회전 디스크를 사용하여 데이터 저장
솔리드 스테이트 드라이브(SDD) : 반도체 메모리를 사용하여 데이터를 저장, 빠른 속도 제공
USB 플래시 드라이브 : 이동성과 편리성을 제공하며, 소규모 데이터를 저장하는 데 유용
광학 디스크 : 레이저를 사용하여 데이터를 읽고 쓰며, 대용량 데이터를 저장하는 데 적합함

보조 기억장치는 대규모 데이터의 영구 저장 및 백업, 파일 시스템 유지, 데이터베이스 로그 저장에 사용

자기 디스크



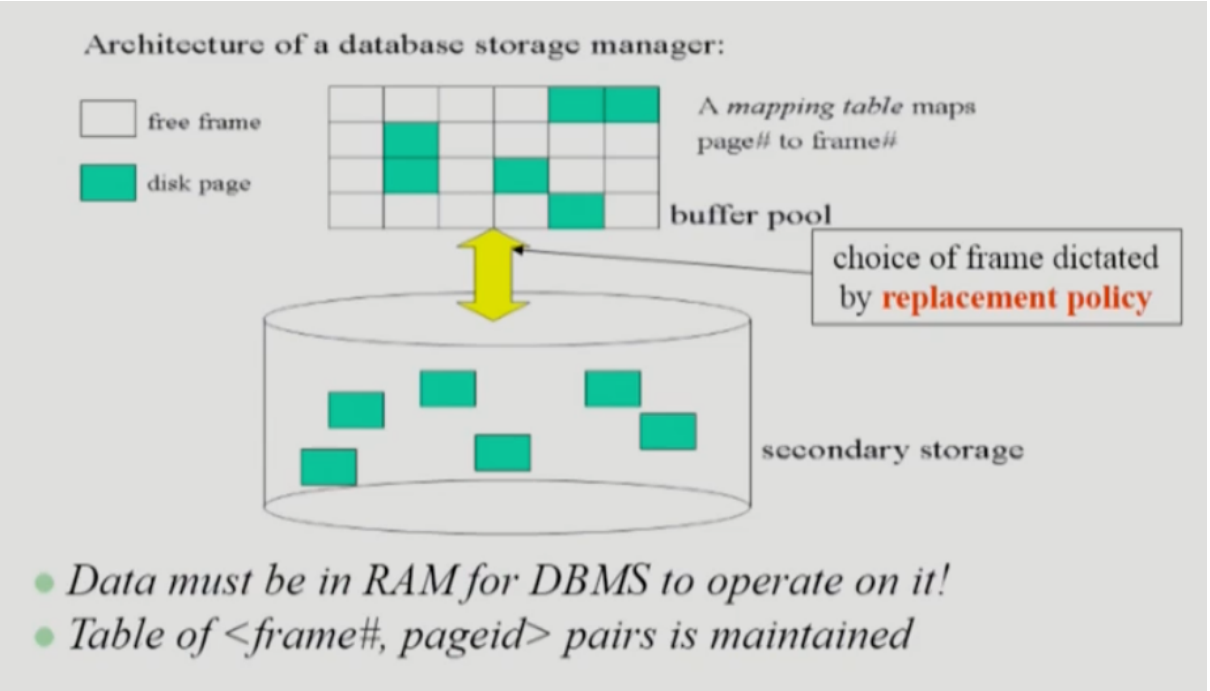
- 디스크는 자기 물질로 만들어진 여러 개의 판(플래터)로 이루어짐
- 각 면마다 디스크와 헤드가 있음
- 간 판은 트랙과 섹터로 구분
- 정보는 디스크 표면 상의 동심원(트랙)을 따라 저장된다.
- 여러 개의 디스크 면 중 같은 지름을 갖는 트랙들을 실린더라고 부른다.
- 블록은 한 개 이상의 섹터들로 이루어짐
- 디스크에서 임의의 블록을 읽어오거나 기록하는데 걸리는 시간은 탐구시간(seek time), 회전 지연시간(rotational delay), 전송 시간(transfer time)의 합
    - 탐구 시간 : 암이 플래터의 섹터를 찾아서 읽는 시간
    - 회전 지연시간 : 암이 전체적으로 회전할 순 없으니 플래터를 회전하는 시간
    - 전송 시간 : 암이 섹터의 데이터를 읽고나서 처리하는 시간

💡 탐구시간이 제일 오래 걸리므로 이 속도를 줄이는 것이 관건이다.

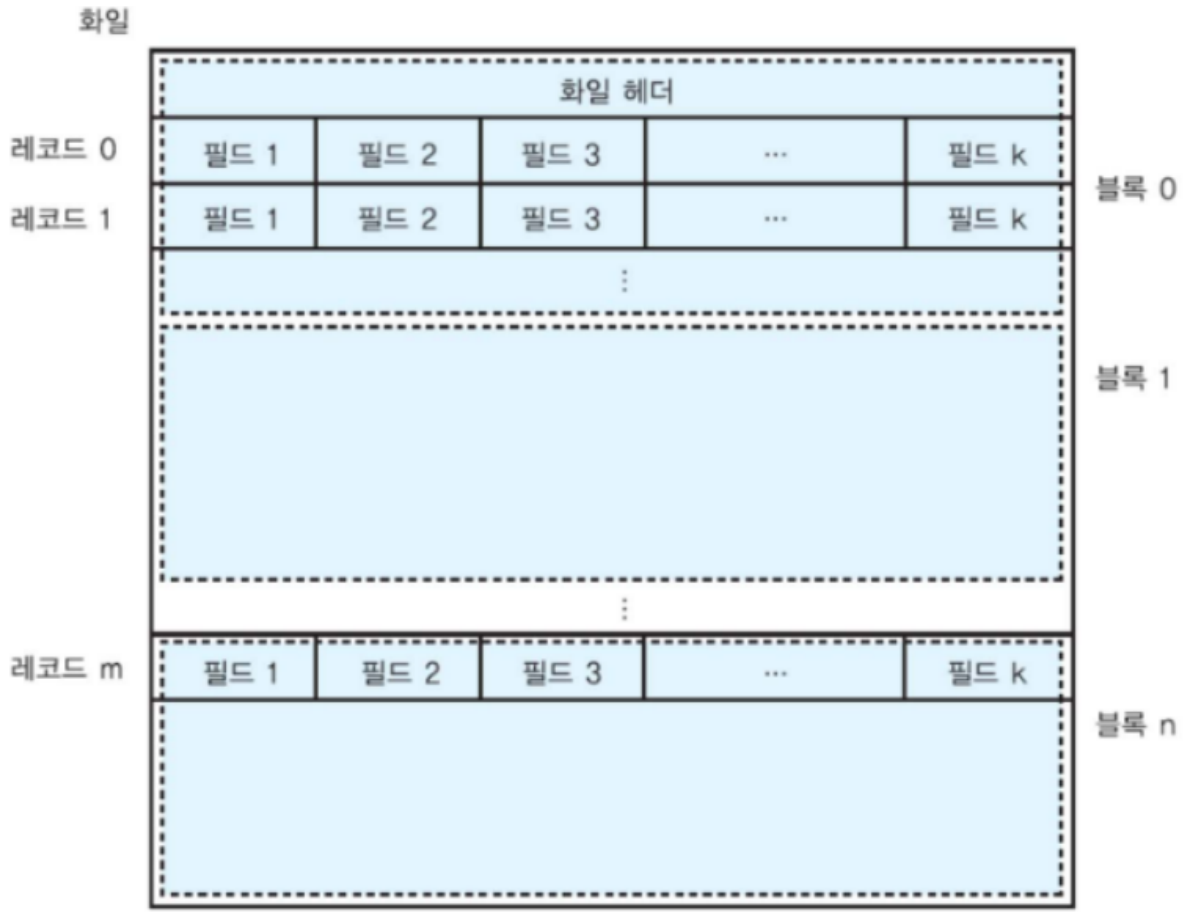
버퍼 관리와 운영체제

💡 버퍼 관리는 db 성능에 중요한 역할을 한다.
운영체제와 협력하여 메모리와 디스크 간의 데이터 전송을 최적화한다.

- 디스크 입출력은 컴퓨터 시스템에서 가장 속도가 느린 작업이므로 입출력 횟수를 줄이는 것이 DBMS 성능을 향상시키는데 중요하다.
- 가능하면 많은 블록을 주기억 장치에 유지하거나, 자주 참조되는 블록들을 주기억 장치에 유지하면 블록 전송 횟수를 줄일 수 있다.
- 버퍼는 디스크 블록을 저장하는데 사용되는 주기억 장치 공간이다.
- 버퍼 관리자는 운영체제의 구성요소로 주기억 장치 내에서 버퍼 공간을 할당하고 관리하는 일을 맡는다.
- 운영체제에서 버퍼관리를 위해 흔히 사용되는 LRU 알고리즘은 데이터베이스를 위해 항상 우수한 성능을 보이지는 않는다.



디스크 상에서 파일의 레코드 배치



파일 > 블록 > 레코드 > 필드 계층 순으로 구성

- 릴레이션의 애트리뷰트는 고정 길이 또는 가변 길이의 필드로 표현
- 연관된 필드들이 모여서 고정 길이 또는 가변 길이의 레코드가 된다.
- 한 릴레이션을 구성하는 레코드들의 모임은 파일이라고 부르는 블록들의 모임에 저장된다.
- 한 파일에 속하는 블록들의 위치가 반드시 인접해 있을 필요는 없다. (분산되어 저장될 수 있다.)
- 인접한 블록들을 읽는 경우에는 탐구 시간과 회전 지연 시간이 들지 않기 때문에 입출력 속도가 빠르므로 블록들이 인접하도록 한 파일의 블록들을 재조직할 수 있다.

Blob

- 이미지, 동영상 등 대규모 크기의 데이터를 저장하는데 사용

채우기 인수

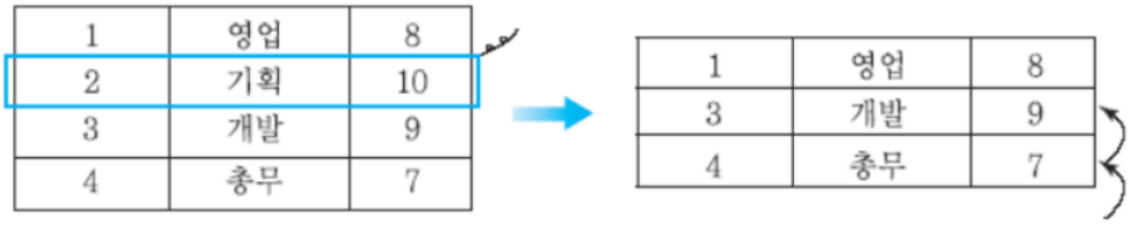
- 각 블록에 레코드를 채우는 공간의 비율
- 나중에 레코드가 삽입되는 상황에서 기존의 레코드들을 이동하는 가능성을 줄이기 위해서

> 💡 새로운 레코드는 새로운 블록에 채우는 것이 효율적이지 않는 상황?
> → 정렬하는 상황에서 레코드는 정해진 위치에 가야하므로 (정확히 이해가 안됨)

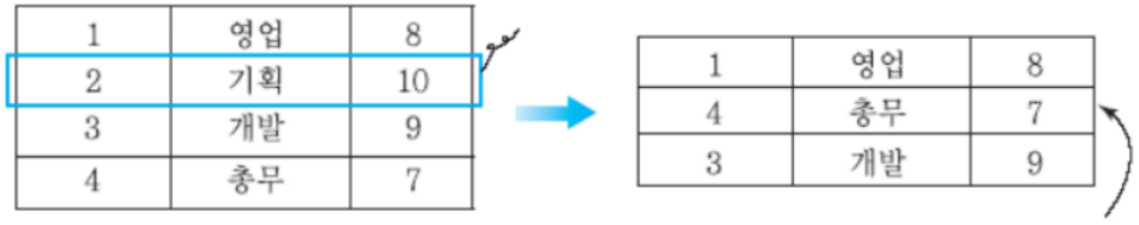
고정 길이 레코드

- 레코드 i를 접근하기 위해서는 n*(i-1) + 1의 위치에서 레코드를 읽음

레고드 삭제



[그림 6.8] 고정 길이 레코드 삭제시 여러 개의 레코드를 이동



[그림 6.9] 고정 길이 레코드 삭제시 한 개의 레코드를 이동

- 방법1: 하나를 지우고 뒤에 있던 레코드들을 모두 이동
- 방법2: 하나를 지우고 맨뒤의 데이터 하나만 삭제한 공간으로 이동
- 방법3(지연 관리 방법): 삭제한 공간을 free list에 등록해 삽입 시 재사용

파일 내의 클러스터링

- 파일 내에 함께 검색될 레코드들을 디스크 상에서 인접하게 위치 시킨다.

파일 간 클러스터링

- 논리적으로 연관되어 함께 검색될 레코드들을 디스크 상에서 인접하게 위치시킴

파일 조직

하프 파일

- 순차적이지 않은 단순한 파일 조직
- 삽입 : 파일 가장 끝에 레코드 삽입 (효율적)
- 검색 : 위에서 부터 아래로 읽어나가며 검색한다. (비효율적)
- 삭제 : 검색 후 레코드를 삭제하고 삭제된 레코드의 공간을 재 사용하지 않는다. (비효율적)
- 성능 : 모든 레코드들을 참조하거나 순서가 중요하지 않을 때 사용하고, 특정 레코드 탐색의 경우 하프 파일은 순차검색 알고리즘으로 n/2소요됨

순차 파일

- 탐색 키 값의 순서에 따라 저장
- 삽입 : 레코드 순서를 고려해야하므로 시간이 많이 걸릴 수 있다. (비효율)
- 검색 : 순차적으로 정의되어 있어 바로 검색 (효율적)
- 삭제 : 삭제된 레코드의 빈 공간을 주기적으로 재조직 (비효율적)
- 기본 인덱스가 없는 순차 파일은 거의 사용되지 않음
- 탐색 키가 아닌 필드로 검색하는 경우 파일 전체를 탐색하므로 비효율적이다.

단일 단계 인덱스

인덱스

- 순차 파일의 빠른 검색을 위해 <탐색 키, 레코드 포인터> 로 관리한다.
- 인덱스는 데이터 파일과는 별도의 파일에 저장
- 인덱스 파일은 데이터 파일의 크기에 비해 훨씬 작다.
- 하나의 데이터 파일에 여러 인덱스들을 정의 가능하다.

인덱스

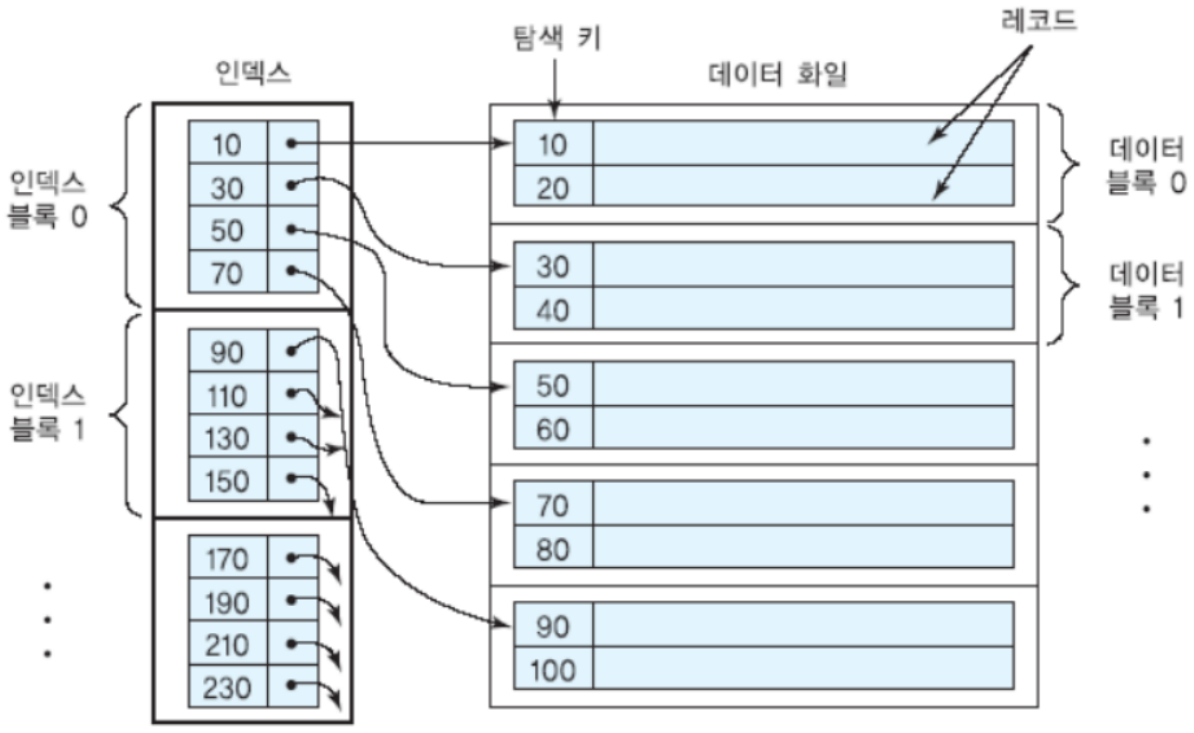
EmpnoIndex	Pointer
1003	
1365	
2106	
3011	
3426	
3427	
4377	

EMPLOYEE

EMPNO	EMPNAME	TITLE	MANAGER	SALARY	DNO
2106	김창섭	대리	1003	2500000	2
3426	박영권	과장	4377	3000000	1
3011	이수민	부장	4377	4000000	3
1003	조민희	과장	4377	3000000	2
3427	최종철	사원	3011	1500000	3
1365	김상원	사원	3426	1500000	1
4377	이성래	사장	∧	5000000	2

- 인덱스가 정의된 필드를 탐색 키라고 부른다.
- 탐색 키는 릴레이션의 기본 키와는 다른 개념이라 값이 중복되도 된다.

기본 인덱스

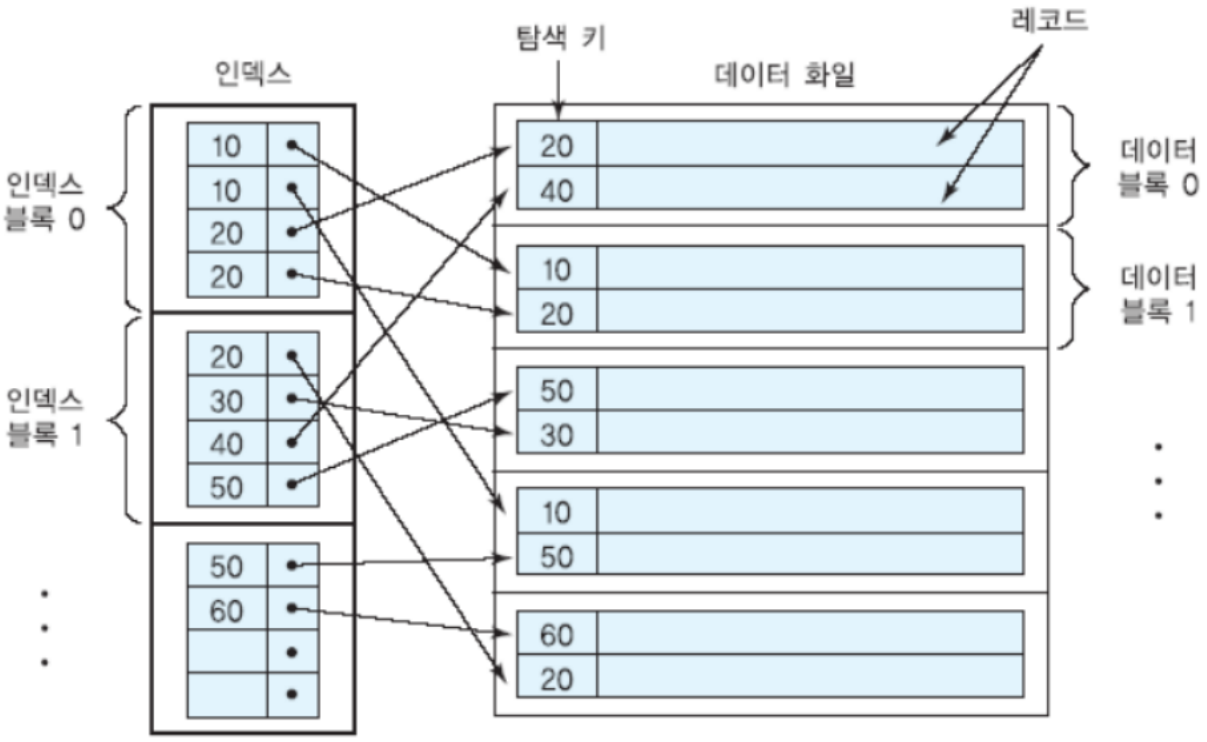


- 탐색 키가 릴레이션의 기본 키인 인덱스
- 인덱스 또한 순서대로 정의되며 중복이 없다.
- 희소 인덱스로 유지된다.
- 각 릴레이션 마다 최대한 한 개의 기본 인덱스를 가질 수 있다.

💡 기본 인덱스를 희소인덱스라고 하는 이유?

모든 레코드에 대해 인덱스 항목을 포함하지 않기 때문에
각 블록의 첫번째 레코드에 대해서만 인덱스 항목이 생성되어 인덱스의 크기를 줄이고, 공간 효율성을 높이며, 데이터베이스 검색 속도를 향상 시킨다.

보조 인덱스



- 탐색 키가 릴레이션의 기본 키가 아닌 인덱스
- 따라서 인덱스는 비순차적이고 중복이 발생할 수 있다.
- 밀집 인덱스로 유지된다.

💡 보조 인덱스를 밀집 인덱스라고 하는 이유?
데이터베이스 테이블의 모든 레코드에 대해 인덱스 항목을 포함하기 때문

밀집 인덱스는 인덱스 엔트리를 가지고 있어서 데이터 검색이 매우 빠르다. 즉, 인덱스의 크기는 커지지만 검색 속도는 극대화 할 수 있다.

클러스터링 인덱스

- 기본 인덱스처럼 탐색 키 값에 따라 정렬된 데이터 파일에 대해 정의한다.
- 기본 인덱스와는 다르게 탐색 키 값이 데이터 파일 내에 중복 존재 가능하다.
- 범위 쿼리에 유용하다(?)

희소 인덱스 vs 밀집 인덱스

희소 인덱스가 모든 갱신과 대부분 쿼리에 대해 더 효율적이다.

count 쿼리에서 같은 속성만 검색하는 쿼리 경우 데이터 파일을 접근할 필요 없이 인덱스만 접근하기에 밀집 인덱스가 더 효율적

한개의 파일은 한개의 희소 인덱스와 다수의 밀집 인덱스를 가질 수 있다.

클러스터링 인덱스 vs 보조 인덱스

클러스터링 인덱스는 희소 인덱스인 경우가 많고 범위 쿼리에 더 효율적임 - 정렬된 상태 이기에
보조 인덱스는 밀집 인덱스이므로 특정한 속성의 쿼리인 상황인 ex) count 경우에 보다 효율적