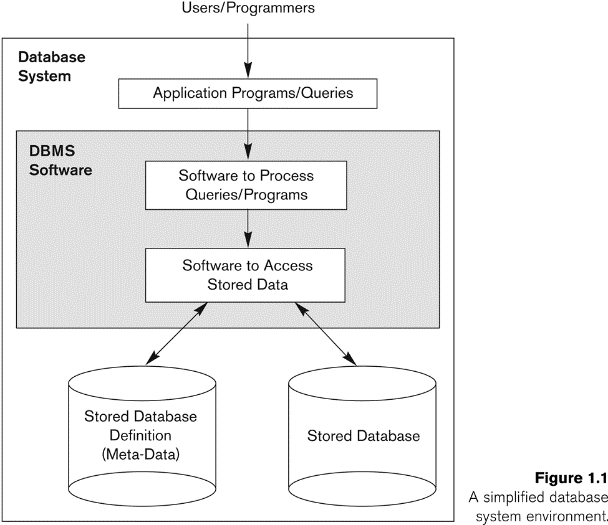
1. **데이터베이스와 데이터베이스 사용자**

용어

* 데이터: 의미를 가지면서 기록될 수 있는 알려진 사실
* 데이터베이스: 관련 있는 데이터의 모임
* 데이터베이스 관리시스템: 데이터베이스 생성과 관리를 담당하는 소프트웨어 패키지
* 데이터베이스 시스템: 데이터베이스와 그를 관리하는 소프트웨어 모두를 칭하는 용어
* 작은 세계: 데이터베이스 구축의 대상이 되는 실세계의 일부분

A Simplified database system environment



DB 응용분야

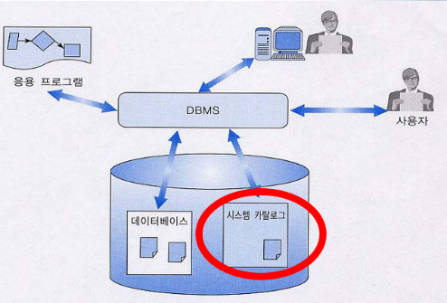
* 전통 분야: 문서 DB
* 최신 분야

1. Multimidia DB
2. GIS(지리정보시스템): 네비, LBS(Location Based System)
3. Data Warehouse (Data mining)
4. 실시간 DB
5. 빅데이터

데이터베이스 사용사례

* 대학에서 학생들의 신상 정보, 수강 과목, 성적 등을 기록
* 항공기 예약 시스템에서 여행사를 통해 항공기 좌석 예약 시 모든 예약 정보가 데이터베이스에 기록

데이터 베이스 시스템의 구성 요소

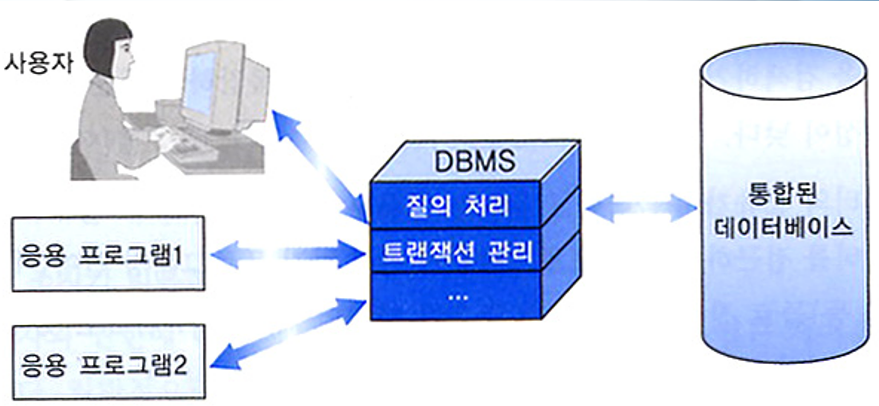


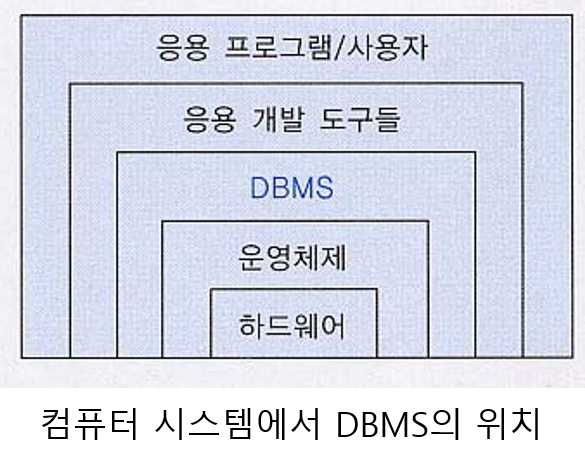
화일 시스템의 특징

* 데이터에 대한 프로그램의 의존도가 높다.
* 데이터의 중복성 -> 데이터의 불일치 야기
* 각종 기능 부족
  1. 데이터 모델링 개념
  2. 질의어
  3. 동시성제어
  4. 파손, 회복
  5. 보안

DBMS를 사용한 데이터베이스 관리

* 여러 사용자/응용 프로그램 데이터베이스 공유
* 사용자의 질의를 빠르게 수행할 수 있는 방법을 자동적으로 선택하여 수행
* 다양한 인터페이스 제공
* 데이터 간의 복잡한 관계를 표현 (모델링)
* 무결성 제약조건 자동적으로 유지
* 시스템 고장 나면 데이터베이스를 고장 전의 일관된 상태로 회복시킴
* 프로그램에 영향주지 않으면서 데이터베이스 구조 변경 가능 (의존성 낮음)





데이터베이스의 특징

1. 데이터베이스 시스템의 자기기술성: 메타데이터를 통해 데이터의 구조를 직접 알고 있지 않아도 데이터를 액세스할 수 있음
2. 프로그램과 데이터의 분리: 데이터베이스 내의 데이터 저장구조가 변경되어도 데이터베이스 응용 프로그램은 영향을 받지 않음(변경될 필요 없음) 사용자 편의성 높음
3. 데이터 추상화: 데이터 모델을 사용하여 저장구조와는 별도로 데이터의 의미를 표현하는 방법 제공
4. 데이터에 대한 다양한 뷰: 데이터베이스의 일부를 뷰로 정의할 수 있음(학생, 교수가 바라보는 데이터 다름)
5. 여러 사용자의 동시 사용

데이터베이스 스키마와 상태

스키마: 전체적인 구조. 자주 변경되지 않음

상태: 특정 시점의 데이터베이스의 내용. 시간이 지남에 따라 계속 바뀜

DBMS의 기능

1. 데이터 중복의 최소화
2. 동시성 제어
3. 보안기능
4. 다양한 사용자 인터페이스
5. 사이에 존재하는 복잡한 관련성을 표현
6. 데이터베이스의 무결성을 보장
7. 백업과 복구 가능

DBMS의 사용 효과

* 표준화된 데이터 관리: 업무 효율 증대
* 데이터 구조 변경에 융통성 부여: 사용자 영향 거의 없음
* 응용 프로그램의 개발 시간 단축
* 항상 최신의 정보 제공
* 규모의 경제성: 통합 관리

DBMS의 단점

* 비용 증가
* 성능 감소
* 고장의 영향 확대

데이터베이스 응용의 역사

1. 계층모델, 네트워크 모델
2. 관계형 모델
3. 객체 지향(복잡한 데이터 모델링)
4. XML 이용 데이터 교환(인터넷 등장)
5. 최신 DB 기능 확장(이미지, 오디오 등)
6. **순서화일의 구조, 검색 삽입 삭제**

레코드의 특징

1. 필드의 개수는 동일
2. 각 필드의 크기는 고정
3. 모든 레코드의 크기는 동일

자료구조 생각해보기

* 가장 시간이 오래 걸리는 것? 하드디스크 액세스
* 배열과 같은 구조로 데이터 화일을 만들기? 문제점은?
  + - 크기 고정, 물리적으로 연속된 공간 할당 받아야함
* 연결리스트 구조로 데이터 화일 만들기? 문제점은?
  + - 레코드 액세스할 때마다 하드디스크 액세스해야 함

**∴** 배열, 연결리스트 둘을 합친 데이터 화일을 사용

데이터 화일의 구조

테이블이(가) 표시된 사진

자동 생성된 설명

* 레코드 추가 삽입, 하드디스크 액세스 횟수 감소
* 블록의 크기는 하드디스크에서 읽을 수 있는 최대 크기로

순서화일에서의 검색

* 순서화일: 특정 필드 값의 크기 순으로 레코드를 저장한 파일
* 순차 검색 (이진검색은 정렬, 임의의 data 바로 access할 수 있어야 하므로 사용 불가능. 더 효율적이긴 하지만)

순서화일에서의 삽입

1. 삽입할 자리 찾기
2. 찾은 블록(A)에 공간이 있으면 삽입
3. 공간이 없으면
   1. 새로운 블록 할당하여 A뒤에 연결
   2. A에 저장되어 있던 레코드와 삽입할 레코드를 A와 B에 절반씩 나누어 저장하고 종료

순서화일에서의 삭제

1. 삭제할 레코드 찾기
2. 삭제한다
3. 블록 사용률 50%이하면
   1. 다음 블록의 블록 사용률 계산
   2. 두 블록 합친게 <= 100%이면 합친다
   3. 100%보다 크면 다음블록에서 일부를 이동하여 두 블록 모두 블록 사용률 50%이상 되도록 한다.

* 저장공간 사용량의 관점에서?

: 2배이하. 최소 절반은 채워져 있으니

* 하드디스크 액세스 횟수의 관점에서?

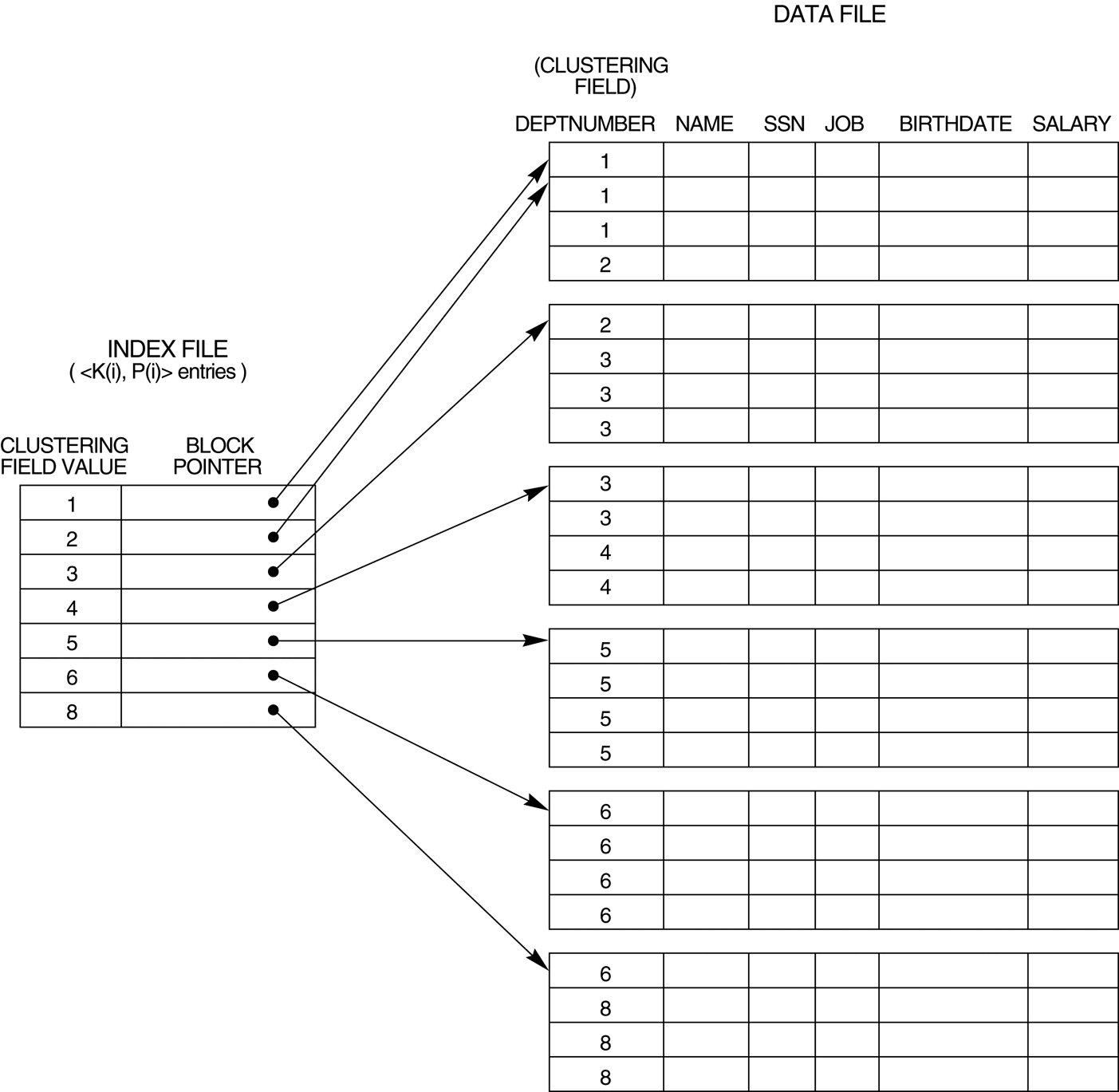
:O(N) -> 블록의 개수가 미니멈일때. 최대2배. (2N)

1. **인덱스**

인덱스란?

* 레코드를 빠르게 찾을 수 있도록 도와주는 보조 파일
* 파일에 대한 또 다른 접근 경로 (순차검색 이외에)

일반적인 인덱스 구조 (중복 있는)



시간: 검색 80%, 삽입/삭제 20%

인덱스 엔트리

* 인덱스 파일의 레코드
* 인덱스 엔트리의 구조

검색 알고리즘

1. 인덱스에서 해당 엔트리 찾기
2. 블록포인터가 가리키는 블록부터 원하는 레코드 찾기

삽입 알고리즘

1. 삽입될 자리 찾기
2. 공간이 없으면 공간을 만든다.
3. 삽입한다.
4. 필요하면 인덱스에 반영

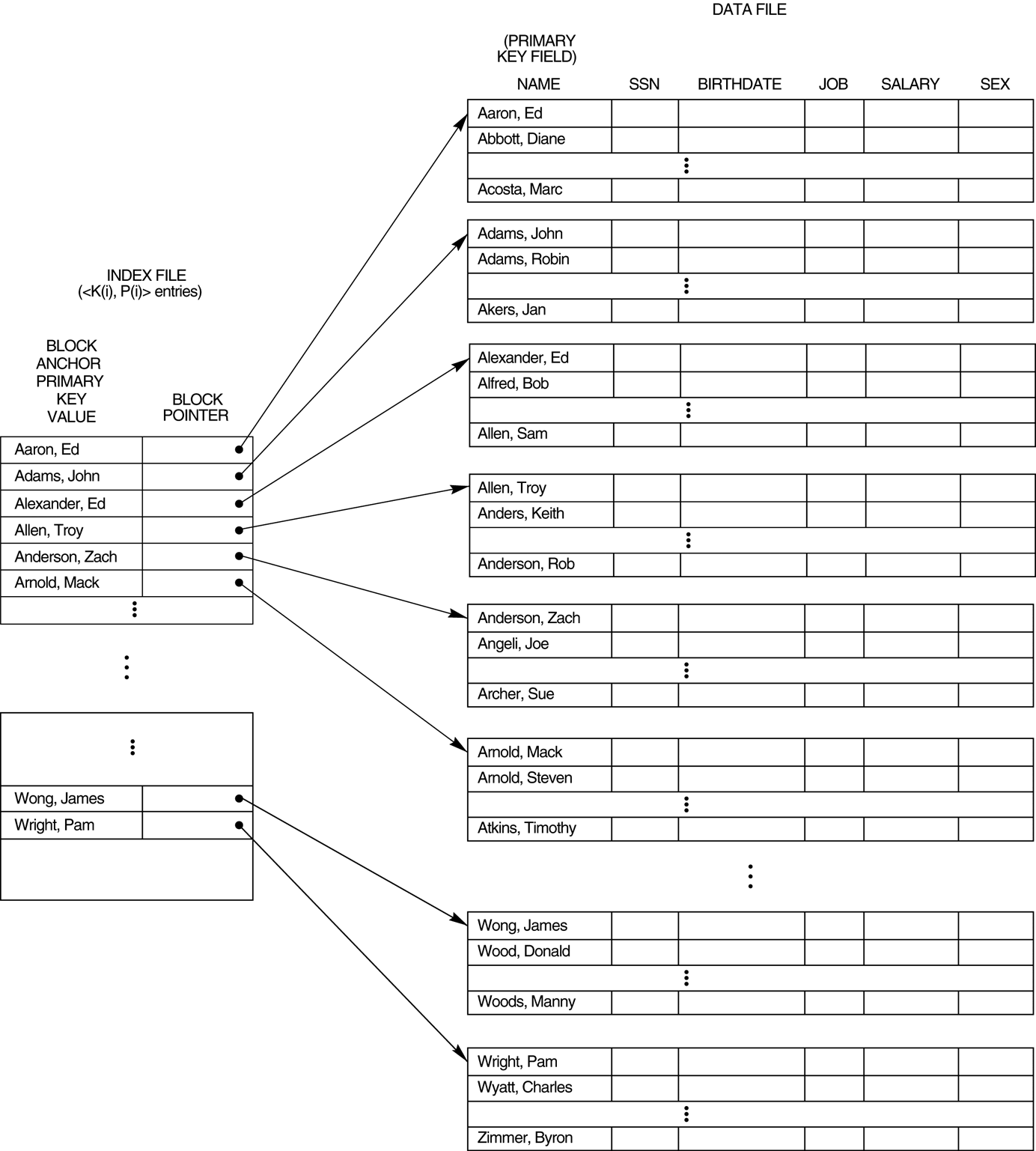
삭제 알고리즘

1. 삭제할 레코드 찾기
2. 삭제한다.
3. 필요하면 삭제한 공간 처리
4. 필요하면 인덱스에 반영 (병합/재분배)

기본 인덱스 (중복 있음)

ex> A, B, C, …으로 인덱스 사용할 경우

* 불필요한 인덱스 엔트리
* 데이터 파일에서의 블록 액세스 증가
  + - 순서 필드 값의 분포 불균형 때문
    - 전체 키 범위가 균등하게 분할된 것이 문제
    - 인덱스 엔트리가 표현하는 범위에 블록 하나만!
* 키 필드가 특정 값이 아닌 범위임



검색, 삽입, 삭제 알고리즘 일반적인 인덱스와 같음.

* 삭제의 경우, 블록의 첫번째 레코드가 삭제되었다고 인덱스에서도 삭제할 필요는 없음. 각 인덱스 엔트리가 표현하고 있는 범위 내에 레코드의 개수가 한 블록의 개수~ 한 블록의 개수 \* 1/2 내에만 있으면 된다.

인덱스의 종류

* 밀집 인덱스: 모든 탐색 키 값에 대한 인덱스 엔트리 정의
* 희소 인덱스: 탐색 값의 일부만 인덱스 엔트리 정의

**∴** 기본 인덱스는 희소 인덱스이다.

화일의 크기와 성능

화일의 크기가 클 수록?

* 저장공간 증가
* 순차검색 시 디스크 액세스 증가
* 희소 인덱스가 밀집 인덱스보다 더 좋음

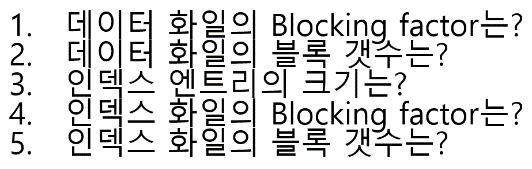
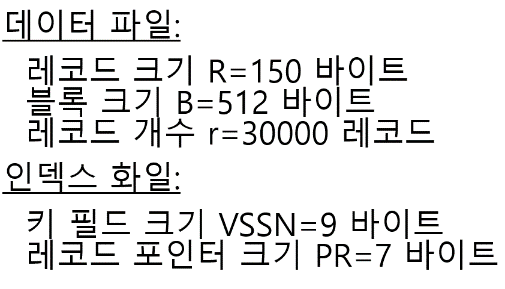
★화일의 크기 계산

* 가정: 각 블록은 레코드로 가득 차 있다.
* 화일의 크기 = 블록의 개수 \* 블록의 크기
* 블록의 개수 =┏ 레코드의 개수/bf ┓(소수점 올림)
* bf = ┗블록의 크기/레코드의 크기┛(소수점이하 버림)

(블록의 개수는 모든 레코드를 포함하여야 하므로 소수점이하는 버린다. -> bf 작아지면 블록의 개수 증가)

※ bf(blocking factor): 하나의 블록에 들어갈 수 있는 레코드의 최대 개수

★밀집인덱스를 사용한 접근 비용의 예



단일 단계 인덱스의 유형

* 기본 인덱스
  + 순서화일의 키 필드에 대하여 정의함
  + 블록 당 하나의 인덱스 엔트리
* 클러스터링 인덱스
  + 순서화일의 non-key 필드에 정의함
  + 키 값당 하나의 인덱스 엔트리
* 보조 인덱스
  + 순서화일의 순서 키가 아닌 키 필드에 대한 인덱스

|  |  |  |
| --- | --- | --- |
| 인덱스 종류 | 중복 | 정렬 |
| 기본 인덱스 | X | O |
| 클러스터링 인덱스 | O | O |
| 첫 번째 보조 인덱스 | X | X |
| 두 번째 보조 인덱스 | O | X |

* + - 2단계 인덱스는 모두 기본 인덱스

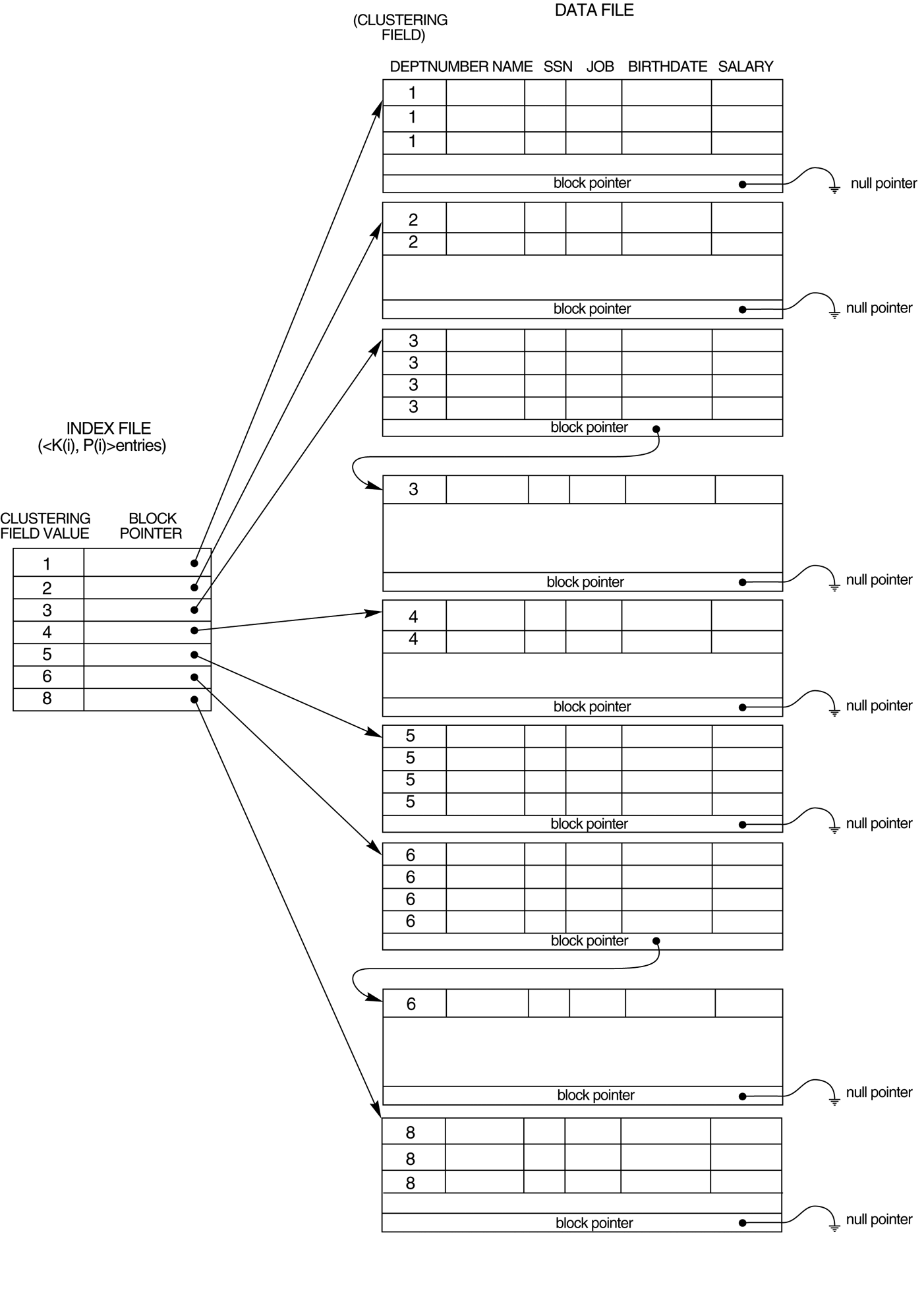
순서화일의 non-key 필드에 대한 인덱스

문제점

* 중복이 많은 경우 데이터 화일 순차 검색 성능 저하
* 중복이 별로 없는데 디스크 액세스 여러 번 함
* 이를 해결하기 위한 클러스터링 인덱스

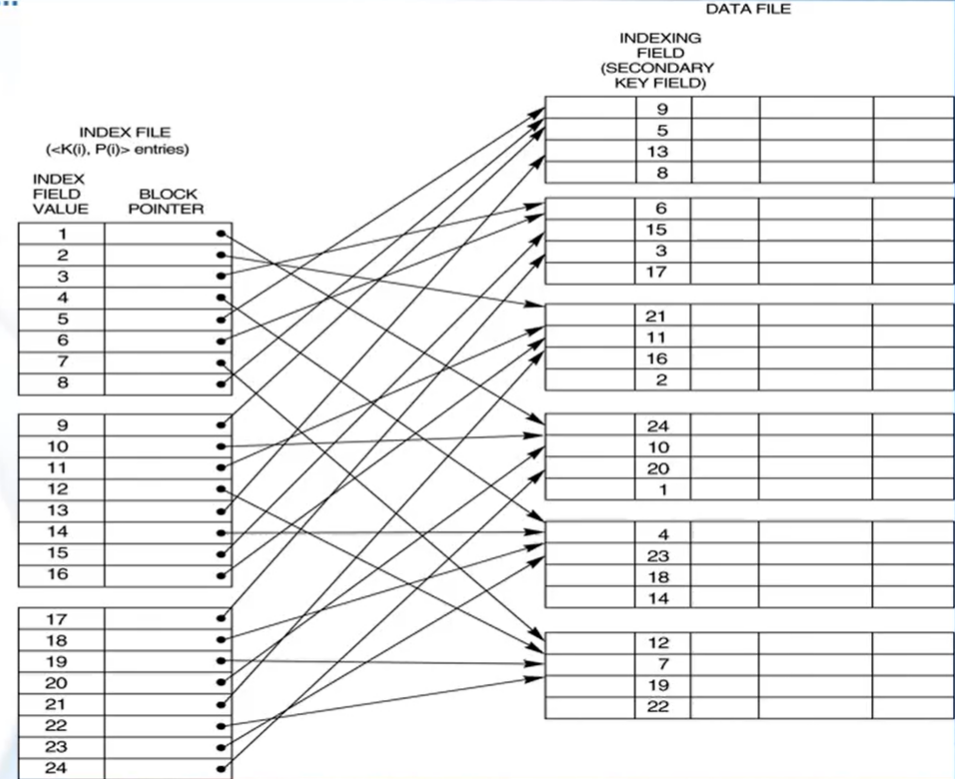
클러스터링 인덱스

* 중복이 없으면 비효율적이다.



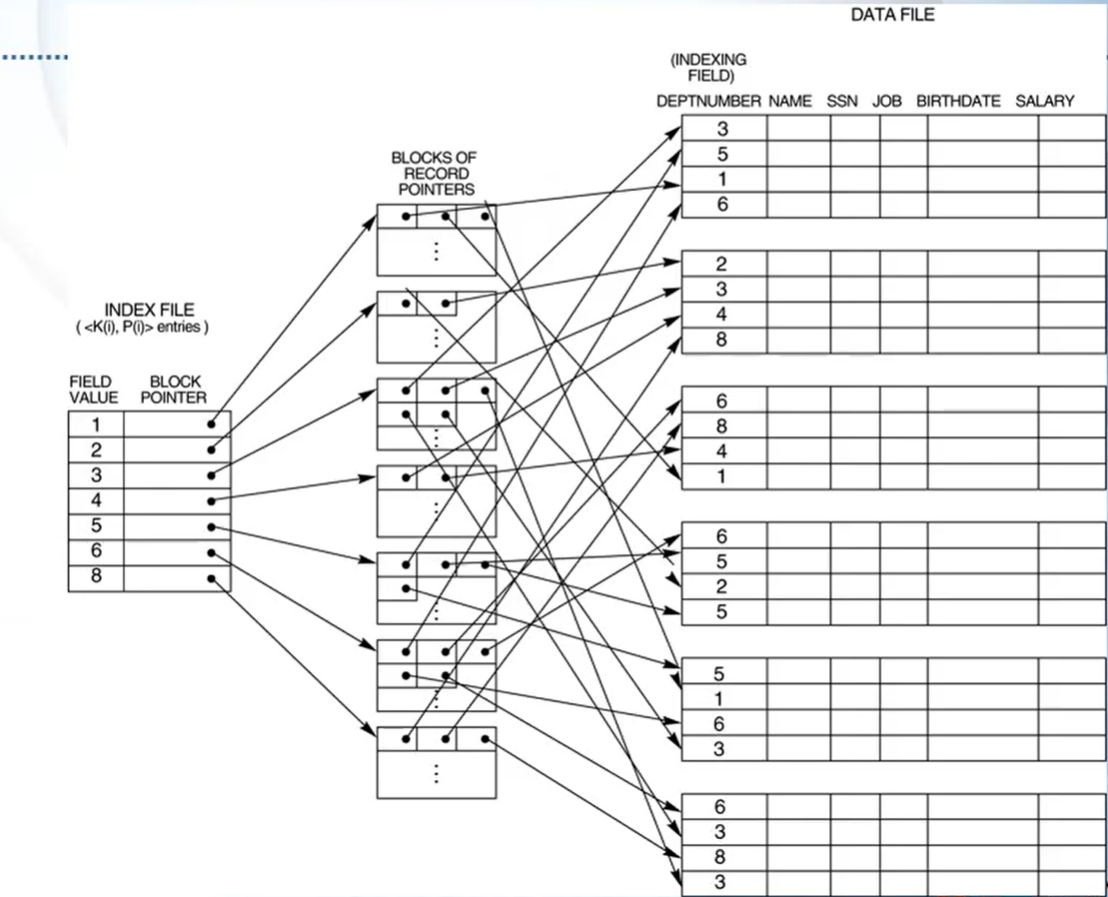
첫 번째 보조 인덱스

* 인덱스 화일의 엔트리의 크기가 더 작으므로, 인덱스 화일의 한 블록 내 엔트리의 개수 > 데이터 화일의 한 블록 내 레코드의 개수. 따라서 인덱스 화일 순차검색이 더 유리하다.



두 번째 보조 인덱스

* 중복이 많을 수록 효율적. 클러스터링 인덱스와 비슷하다.

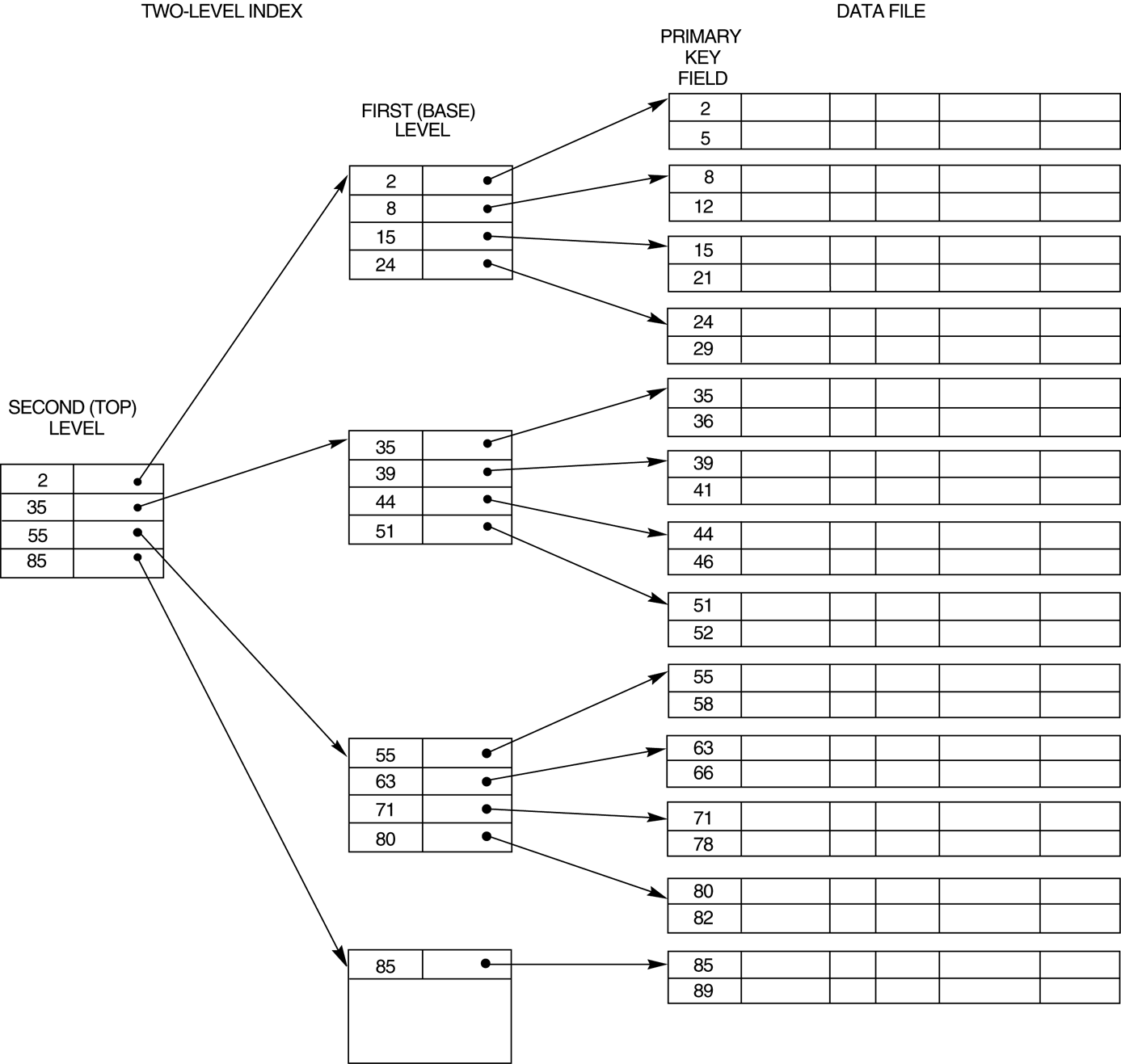


**∴** 인덱스 엔트리의 검색이 최적화되어야 한다

* + - 인덱스의 인덱스 만들기

다단계 인덱스

* 탐색 트리 형태
* 트리의 높이가 디스크 액세스 횟수 (n+1회)
* 2단계 기본 인덱스의 경우 3번의 디스크 액세스
* bf가 클수록 단계가 팍팍 준다.



★ Q. 다단계 인덱스 화일의 치명적 오류?

* 37 삽입 그림 그리기
* 1 삽입 -> 처음 만들어진 값보다 작은 값 삽입 불가
  + - 맨 앞의 앞에도 포인터를 만든다! (N진 트리 형태)

1. **B, B+ 트리**

★★★

OX

* 복합 에트리뷰트는 키가 될 수 있냐 O
* 우리가 배운 인덱스는 다 기본 인덱스인가? X
* 클러스터링 인덱스는 밀집으로 표현할 수 있냐 O
* 비순서 중복에는 보조 인덱스밖에 쓸 수 없다 X
* 비트리, 비플러스 트리 중에 어느 것이 높이가 더 높나

: 비플러스가 더 낮음(검색 시간과 엑세스 횟수가 적음)

- 다단계 인덱스 관련 문제 많음

단답형

* 데이터 베이스 관리 시스템과 데이터 베이스를 합치면?

: 데이터 베이스 시스템

* 데이터 베이스 특징 단어
* 데이타베이스 시스템의 자기 기술성: 데이터 베이스의 구조를 직접 알고 있지 않아도 데이터를 사용할 수 있음
* 프로그램과 데이터의 분리: 구조가 변경되어도 응용프로그램이 영향을 받지 않음 --> 이로 인해 데이터 무결성이 성립
* 데이터 추상화: 저장구조와는 별도로 실제 저장하는 구조와 보여주는 모습이 다를 수 있음
* 데이터에 대한 다양한 뷰: 사용자는 관심이 있는 데이터베이스의 일부를 뷰로 정의할 수 있음
* 여러 사용자의 동시 사용
* 데이터 무결성(데이터의 정확성와 일관성을 유지하고 보증하는 것)
* 데이터에 대해 프로그램이 독립적일 때의 단점 -> DBMS의 단점????
* B+ tree 3개 인서트 딜리트
* B+ tree 위에 이어서 인서트 딜리트
* 같은 엔티티 타입일 때 관계는?

: 순환적 관계 (상사와 부하 직원 사이의 관계 타입 --> 직원과 직원 사이의 순환적 관계 타입)

* 학생과 과목 사이에 수강이라는 관계가 있을 때 그건 몇대몇?

: n:n

* 학생과 과목 사이의 관계가 각각 필수인가 선택인가?

: 둘 다 선택 아닐까?

* 학생과 과목 사이에 애트리뷰트들 그려라(복합, 다치 등등 여러개 그리라고함)
* 데이터 베이스에 비한 파일 단점 (여러개 나열하고 맞는 것들 고르기)
* 프로그램에 대한 파일 의존도 높다
* 데이터의 중복성 --> 불일치가 생기고 데이터 낭비
* 데이터 모델링 개념
* 질의어
* 동시성 제어 --> 여러사람이 한 파일에 접근할 수 없지만 데이터 베이스는 가능함
* 파손, 회복: 데이터 베이스는 백업을 해두지만 파일은 놉
* 보안
* 레코드 크기 64, 인덱스 레코드 크기 32, 200000개 레코드 기본 인덱스로 만들 때 계산하는 문제

:필드 마다의 크기를 알려주고 내가 다 더해서 레코드의 크기를 알아내야함

* 데이터 화일의 Blocking factor는?

: └데이터 블록의 크기 / 데이터 레코드의 크기┘(버림)

* 데이터 화일의 블록 갯수는?

:┌레코드 개수 / 데이터 blocking factor ┐(올림)

* 인덱스 엔트리의 크기는?

: 인덱스 키 필드 크기 + 인덱스 레코드 포인터 크기

* 인덱스 화일의 Blocking factor는?

: 인덱스 블록 크기 / 인덱스 엔트리 크기

* 인덱스 화일의 블록 갯수는?

밀집: 총 레코드 개수 / 인덱스 blocking factor

기본: 데이터 블록 개수 / 인덱스 blocking factor

1. **ER 모델**

데이터베이스 설계

* 조직체의 운영과 목적을 지원하기 위해 데이터베이스를 생성하는 과정
* 모든 주요 응용과 사용자들이 요구하는 데이터, 데이터 간의 관계를 표현하는 것

데이터베이스 설계의 요구사항

* 데이터의 모든 측면을 표현
* 중복 최소화
* 효율적 접근
* 데이터 무결성
* 일관성 및 이해 용이

데이터베이스 설계의 주요단계

ER 모델

ER 모델의 요소

엔티티와 엔티티 타입

애트리뷰트

애트리뷰트 종류

1. **관계형 데이터 모델**