

제 6 장



물리적 데이터베이스 설계

- 6.1 보조 기억 장치
- 6.2 버퍼 관리와 운영 체제
- 6.3 디스크 상에서 화일의 레코드 배치
- 6.4 화일 조직
- 6.5 단일 단계 인덱스
- 6.6 다단계 인덱스
- 6.7 인덱스 선정 지침과 데이터베이스 튜닝
 - 연습문제

6장. 물리적 데이터베이스 설계

□ 물리적 데이터베이스 설계

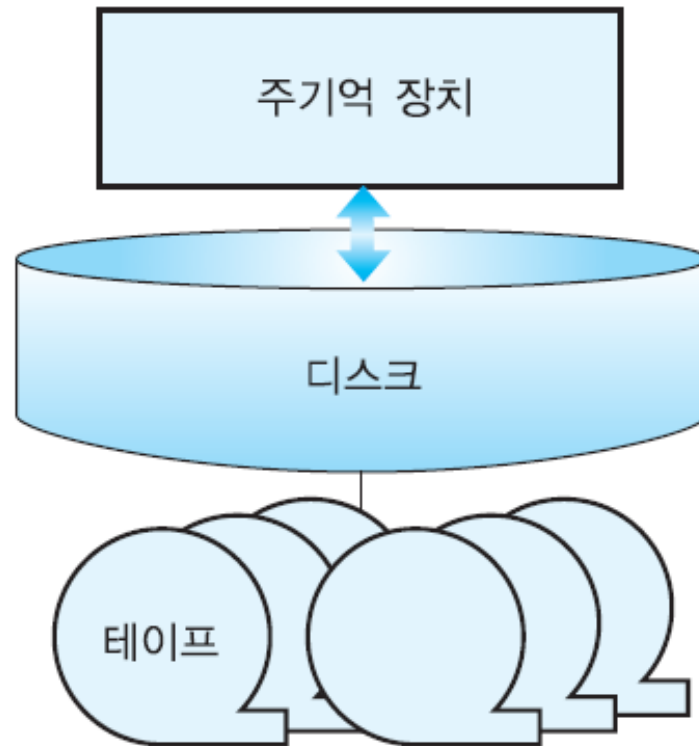
- ✓ 논리적인 설계의 데이터 구조를 보조 기억 장치상의 화일(물리적인 데이터 모델)로 사상함
- ✓ 예상 빈도를 포함하여 데이터베이스 질의와 트랜잭션들을 분석함
- ✓ 데이터에 대한 효율적인 접근을 제공하기 위하여 저장 구조와 접근 방법들을 다룸
- ✓ 특정 DBMS의 특성을 고려하여 진행됨
- ✓ 질의를 효율적으로 지원하기 위해서 인덱스 구조를 적절히 사용함

6.1 보조 기억 장치

□ 보조 기억 장치

- ✓ 사용자가 원하는 데이터를 검색하기 위해서 DBMS는 디스크 상의 데이터베이스로부터 사용자가 원하는 데이터를 포함하고 있는 블록을 읽어서 주기억 장치로 가져옴
- ✓ 데이터가 변경된 경우에는 블록들을 디스크에 다시 기록함
- ✓ 블록 크기는 512바이트부터 수 킬로바이트까지 다양함
- ✓ 전형적인 블록 크기는 4,096바이트
- ✓ 각 화일은 고정된 크기의 블록들로 나누어져서 저장됨
- ✓ 디스크는 데이터베이스를 장기간 보관하는 주된 보조 기억 장치

6.1 보조 기억 장치(계속)



[그림 6.1] 저장 장치의 계층 구조

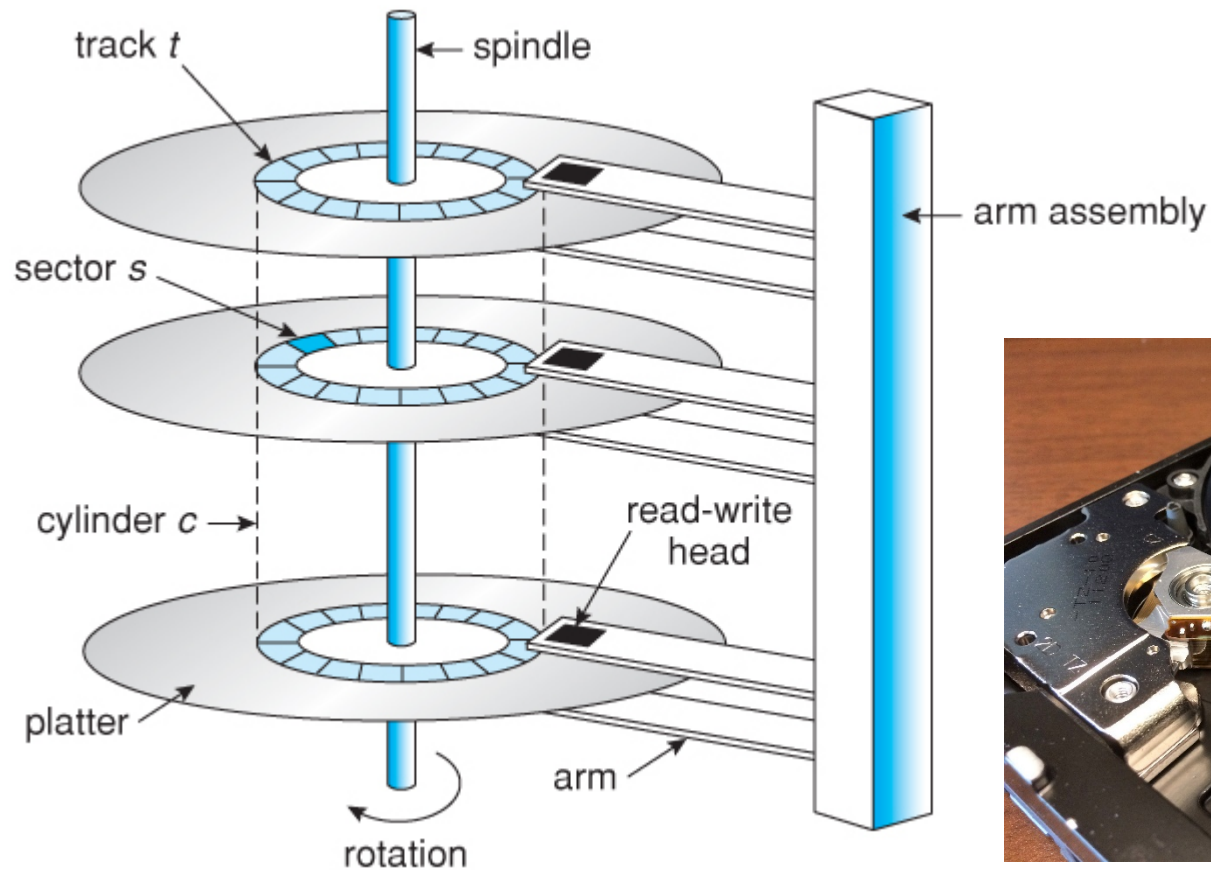
6.1 보조 기억 장치(계속)

□ 자기 디스크

- ✓ 디스크는 자기 물질로 만들어진 여러 개의 판으로 이루어짐
- ✓ 각 면마다 디스크 헤드가 있음
- ✓ 각 판은 **트랙**과 **섹터**로 구분됨
- ✓ 정보는 디스크 표면 상의 동심원(트랙)을 따라 저장됨
- ✓ 여러 개의 디스크 면 중에서 같은 지름을 갖는 트랙들을 **실린더**라고 부름
- ✓ 블록은 한 개 이상의 섹터들로 이루어짐
- ✓ 디스크에서 임의의 블록을 읽어오거나 기록하는데 걸리는 시간은 **탐구 시간**(seek time), **회전 지연 시간**(rotational delay), **전송 시간**(transfer time)의 합

6.1 보조 기억 장치(계속)

□ 자기 디스크



6.2 버퍼 관리와 운영 체제

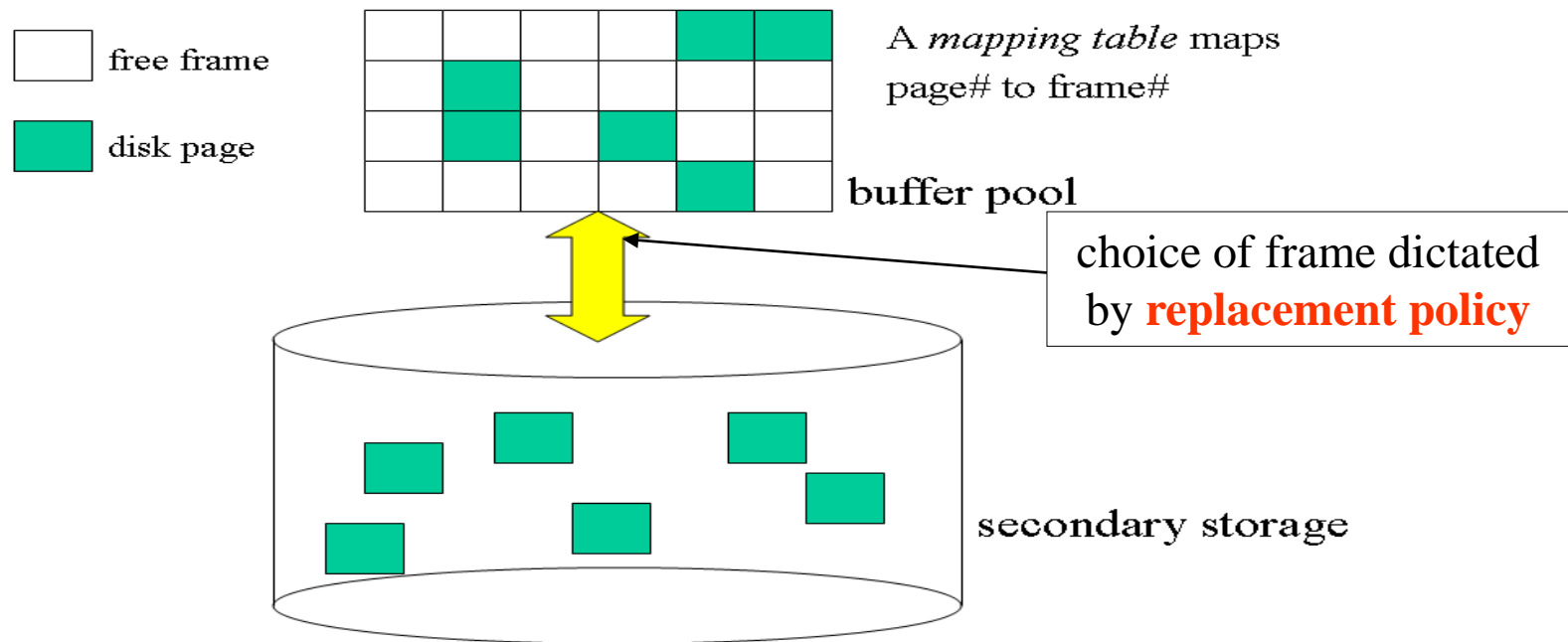
□ 버퍼 관리와 운영 체제

- ✓ 디스크 입출력은 컴퓨터 시스템에서 가장 속도가 느린 작업이므로 입출력 횟수를 줄이는 것이 DBMS의 성능을 향상하는데 매우 중요
- ✓ 가능하면 많은 블록들을 주기억 장치에 유지하거나, 자주 참조되는 블록들을 주기억 장치에 유지하면 블록 전송 횟수를 줄일 수 있음
- ✓ 버퍼는 디스크 블록들을 저장하는데 사용되는 주기억 장치 공간
- ✓ 버퍼 관리자는 운영 체제의 구성요소로서 주기억 장치 내에서 버퍼 공간을 할당하고 관리하는 일을 맡음
- ✓ 운영 체제에서 버퍼 관리를 위해 흔히 사용되는 LRU 알고리즘은 데이터베이스를 위해 항상 우수한 성능을 보이지는 않음

위험성 존재

6.2 버퍼 관리와 운영 체제(계속)

Architecture of a database storage manager:



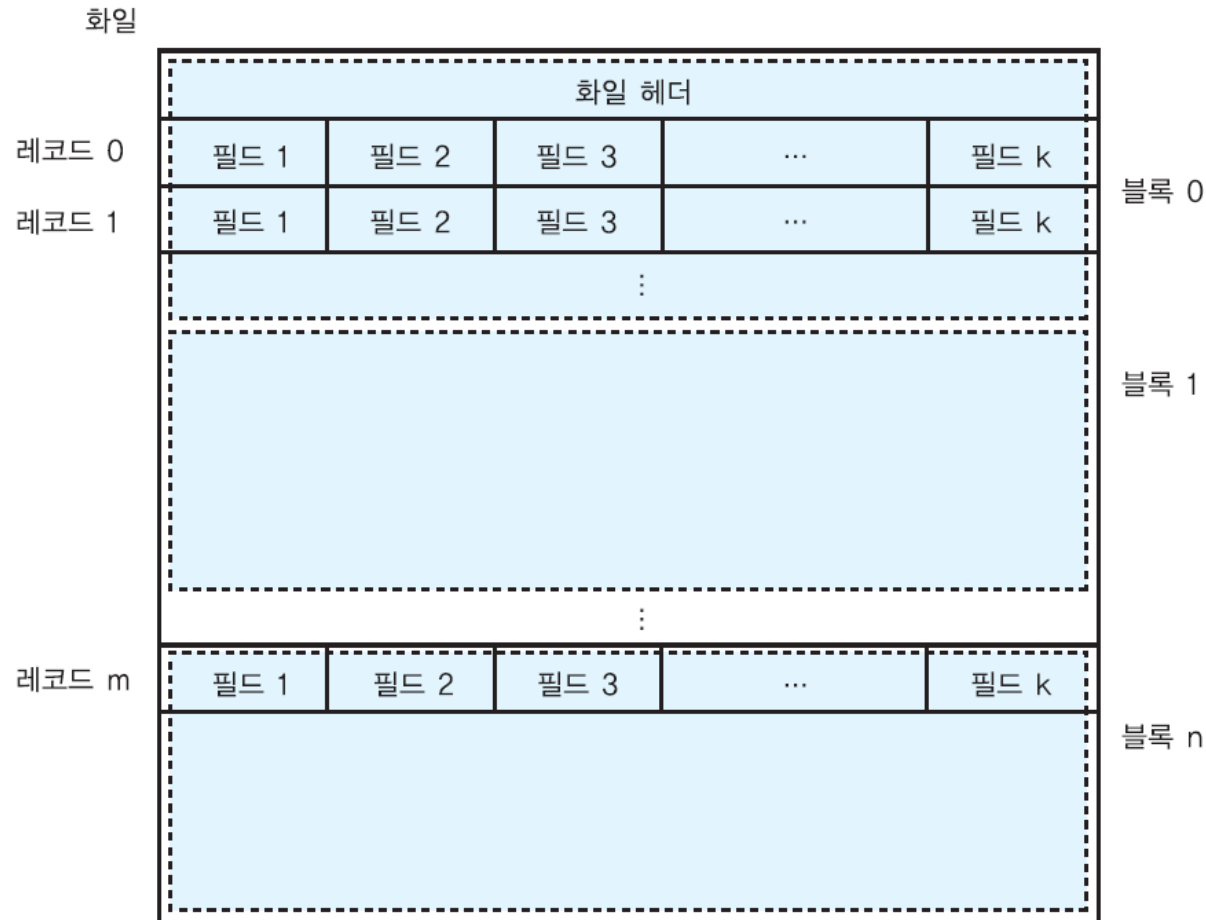
- *Data must be in RAM for DBMS to operate on it!*
- *Table of $\langle \text{frame\#}, \text{pageid} \rangle$ pairs is maintained*

6.3 디스크 상에서 화일의 레코드 배치

□ 디스크 상에서 화일의 레코드 배치

- ✓ 릴레이션의 애트리뷰트는 고정 길이 또는 가변 길이의 필드로 표현됨
- ✓ 연관된 필드들이 모여서 고정 길이 또는 가변 길이의 레코드가 됨
- ✓ 한 릴레이션을 구성하는 레코드들의 모임이 화일이라고 부르는 블록들의 모임에 저장됨

6.3 디스크 상에서 화일의 레코드 배치(계속)

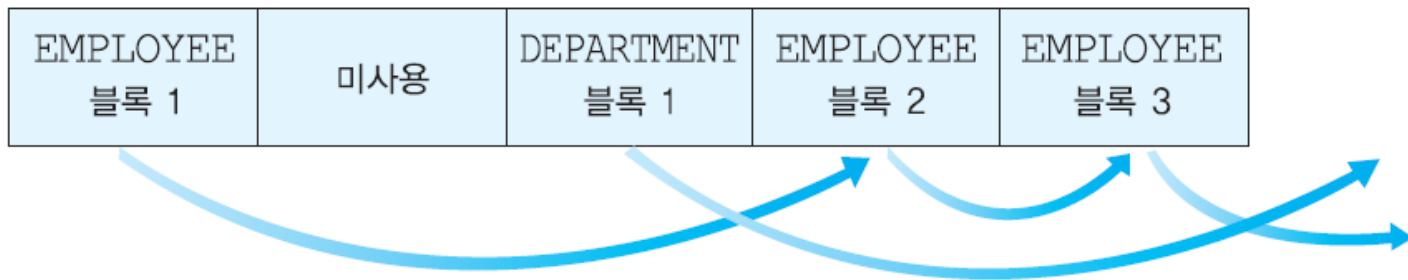


[그림 6.4] 파일과 블록과 레코드

6.3 디스크 상에서 화일의 레코드 배치(계속)

□ 디스크 상에서 화일의 레코드 배치(계속)

- ✓ 한 화일에 속하는 블록들이 반드시 인접해 있을 필요는 없음
- ✓ 인접한 블록들을 읽는 경우에는 탐구 시간과 회전 지연 시간이 들지 않기 때문에 입출력 속도가 빠르므로 블록들이 인접하도록 한 화일의 블록들을 재조직할 수 있음



[그림 6.5] 디스크에서 블록들의 연결

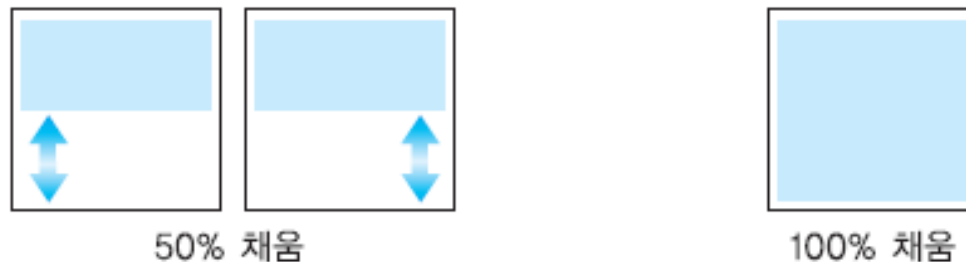
6.3 디스크 상에서 화일의 레코드 배치(계속)

❑ BLOB(Binary Large Object)

- ✓ 이미지 (GIF, JPG), 동영상 (MPEG, RM) 등 대규모 크기의 데이터를 저장하는데 사용됨
- ✓ BLOB의 최대 크기는 오라클에서 8TB~128TB

❑ 채우기 인수

- ✓ 각 블록에 레코드를 채우는 공간의 비율
- ✓ 나중에 레코드가 삽입될 때 기존의 레코드들을 이동하는 가능성을 줄이기 위해서



[그림 6.6] 채우기 인수

6.3 디스크 상에서 화일의 레코드 배치(계속)

□ 고정 길이 레코드

- ✓ 레코드 i 를 접근하기 위해서는 $n*(i-1)+1$ 의 위치에서 레코드를 읽음

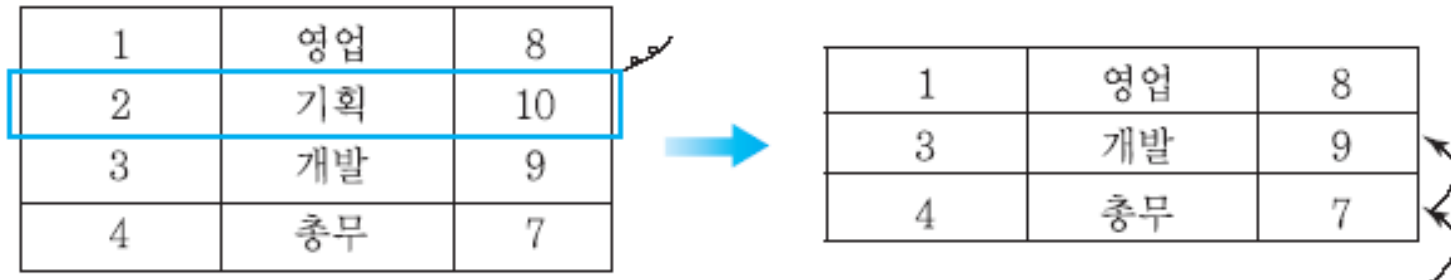
| | | |
|---|----|----|
| 1 | 영업 | 8 |
| 2 | 기획 | 10 |
| 3 | 개발 | 9 |
| 4 | 총무 | 7 |

← 18바이트 →

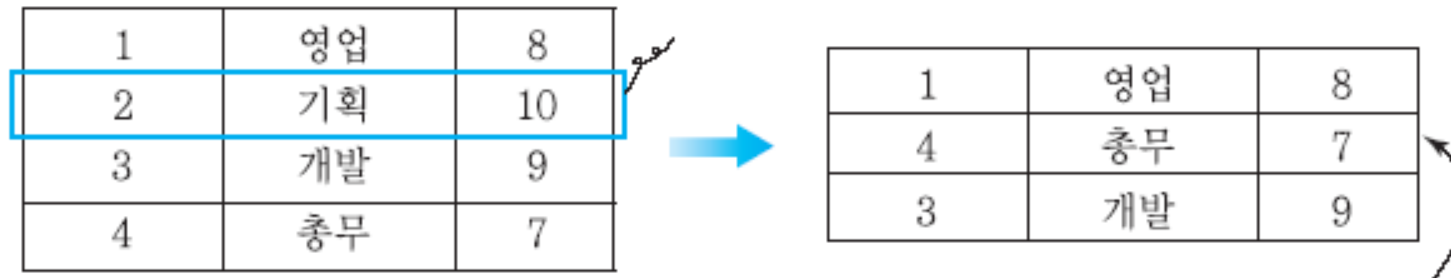
[그림 6.7] 고정 길이 레코드 읽기

6.3 디스크 상에서 화일의 레코드 배치(계속)

□ 고정 길이 레코드(계속)



[그림 6.8] 고정 길이 레코드 삭제시 여러 개의 레코드를 이동



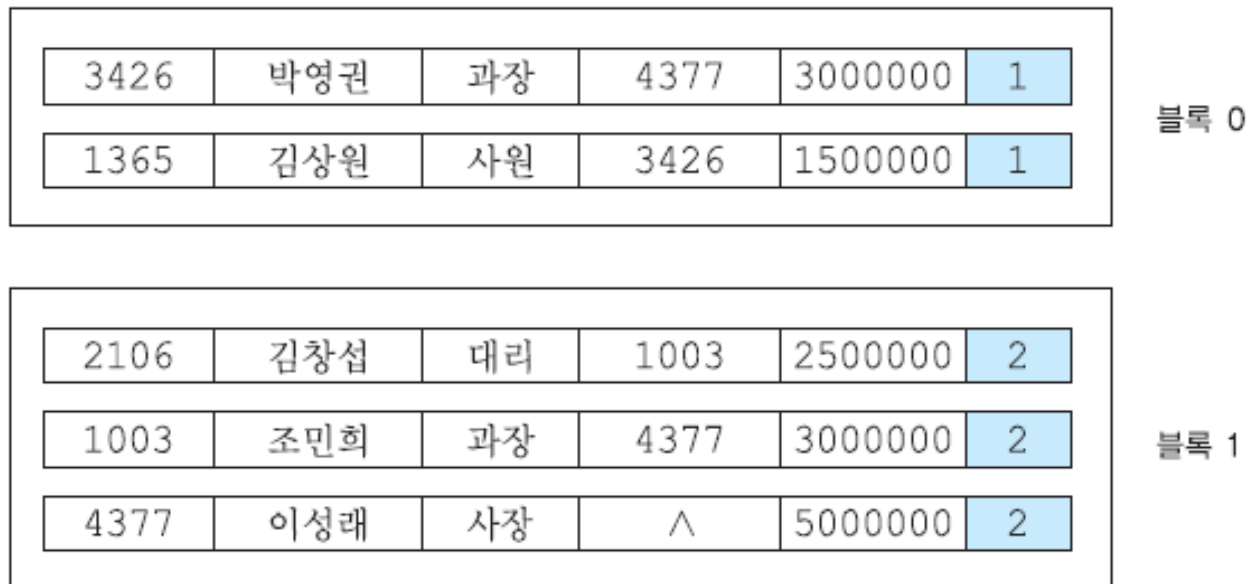
[그림 6.9] 고정 길이 레코드 삭제시 한 개의 레코드를 이동

* 자연 관리 방법 (이동 필요 없음): 삭제된 공간을 관리하기 위해 free list를 관리

6.3 디스크 상에서 화일의 레코드 배치(계속)

❑ 화일 내의 클러스터링(intra-file clustering)

- ✓ 한 화일 내에서 함께 검색될 가능성이 높은 레코드들을 디스크 상에서 물리적으로 가까운 곳에 모아두는 것

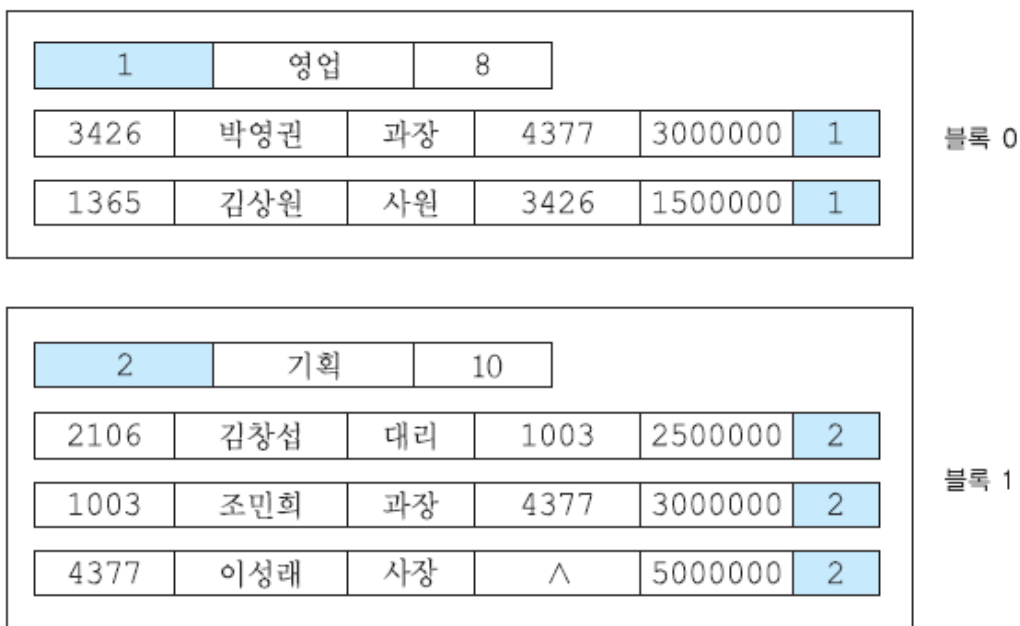


[그림 6.10] 화일 내의 클러스터링

6.3 디스크 상에서 화일의 레코드 배치(계속)

❑ 화일 간의 클러스터링(inter-file clustering)

- ✓ 논리적으로 연관되어 함께 검색될 가능성이 높은 두 개 이상의 화일에 속한 레코드들을 디스크 상에서 물리적으로 가까운 곳에 저장하는 것



[그림 6.11] 화일 간의 클러스터링

6.4 화일 조직

□ 화일 조직의 유형

- ✓ 히프 화일(heap file)
- ✓ 순차 화일(sequential file) → 연결이 되어있는 것에 대해 비바른 탐색 가능
- ✓ 인덱스된 순차 화일(indexed sequential file) (비진 탐색)
- ✓ 직접 화일(hash file) but, 한개밖에 안됨

↓
아는 여러개 됨
but, 인덱스에 대해
데이터 중복 발생
∴ 엔트리 만들 : 값 대신 포인터 가지고 있음

6.4 화일 조직(계속)

□ 히프 파일(비순서 화일)

- ✓ 가장 단순한 화일 조직
- ✓ 일반적으로 레코드들이 삽입된 순서대로 화일에 저장됨
- ✓ 삽입: 새로 삽입되는 레코드는 화일의 가장 끝에 첨부됨
- ✓ 검색: 원하는 레코드를 찾기 위해서는 모든 레코드들을 순차적으로 접근해야 함
- ✓ 삭제: 원하는 레코드를 찾은 후에 그 레코드를 삭제하고, 삭제된 레코드가 차지하던 공간을 재사용하지 않음
- ✓ 좋은 성능을 유지하기 위해서 히프 화일을 주기적으로 재조직할 필요가 있음

6.4 화일 조직(계속)

| | | | | | |
|------|-----|----|------|---------|---|
| 2106 | 김창섭 | 대리 | 1003 | 2500000 | 2 |
| 3426 | 박영권 | 과장 | 4377 | 3000000 | 1 |
| 3011 | 이수민 | 부장 | 4377 | 4000000 | 3 |

블록 0

| | | | | | |
|------|-----|----|------|---------|---|
| 1003 | 조민희 | 과장 | 4377 | 3000000 | 2 |
| 3427 | 최종철 | 사원 | 3011 | 1500000 | 3 |
| 1365 | 김상원 | 사원 | 3426 | 1500000 | 1 |

블록 1

| | | | | | |
|------|-----|----|---|---------|---|
| 4377 | 이성래 | 사장 | ∧ | 5000000 | 2 |
|------|-----|----|---|---------|---|

블록 2

[그림 6.12] EMPLOYEE 화일을 히프 화일로 저장

6.4 화일 조직(계속)

□ 히프 화일의 성능

- ✓ 히프 화일은 질의에서 모든 레코드들을 참조하고, 레코드들을 접근하는 순서는 중요하지 않을 때

```
SELECT      *  
FROM        EMPLOYEE;
```

- ✓ 특정 레코드를 검색하는 경우에는 히프 화일이 비효율적
 - 히프 화일에 b개의 블록이 있다고 가정하자. 원하는 블록을 찾기 위해서 평균적으로 $b/2$ 개의 블록을 읽어야 함

```
SELECT      TITLE  
FROM        EMPLOYEE  
WHERE       EMPNO = 1365;
```

6.4 화일 조직(계속)

예 :

화일에 레코드가 10,000,000개 있고, 각 레코드의 길이가 200바이트이고, 블록 크기가 4,096바이트이면 블록킹 인수는 $\lfloor 4,096/200 \rfloor = 20$ 인 화일을 가정하자. 블록킹 인수(blocking factor)는 한 블록에 포함되는 레코드 수를 의미한다. 블록킹 인수는 레코드의 크기에 따라 달라진다. 이 화일을 위해 필요한 총 블록 수는 $\lceil 10,000,000/20 \rceil = 500,000$ 이다.

특정 레코드를 찾기 위해서는 평균적으로 $500,000/2 = 250,000$ 개의 디스크 블록을 읽어야 한다. 한 블록을 읽는데 10ms가 걸린다고 가정하면 $250,000 \times 10\text{ms} = 2,500,000\text{ms} = 2,500\text{초} \approx 42\text{분}$ 이 소요된다.

6.4 화일 조직(계속)

❑ 히프 화일의 성능(계속)

- ✓ 몇 개의 레코드들을 검색하는 경우에도 비효율적
- ✓ 조건에 맞는 레코드를 이미 한 개 이상 검색했더라도 화일의 마지막 블록까지 읽어서 원하는 레코드가 존재하는가를 확인해야 하기 때문에 b개의 블록을 모두 읽어야 함

```
SELECT EMPNAME, TITLE  
FROM    EMPLOYEE  
WHERE   DNO = 2;
```

- ✓ 급여의 범위를 만족하는 레코드들을 모두 검색하는 아래의 질의도 EMPLOYEE 릴레이션의 모든 레코드들을 접근해야 함

```
SELECT EMPNAME, TITLE  
FROM    EMPLOYEE  
WHERE   SALARY >= 3000000  
        AND SALARY <= 4000000;
```

6.4 화일 조직(계속)

〈표 6.1〉 연산의 유형과 소요 시간

| 연산의 유형 | 시간 |
|-----------|-----------|
| 삽입 | 효율적 |
| 삭제 | 시간이 많이 소요 |
| 탐색 | 시간이 많이 소요 |
| 순서대로 검색 | 시간이 많이 소요 |
| 특정 레코드 검색 | 시간이 많이 소요 |

6.4 화일 조직(계속)

□ 순차 화일

- ✓ 레코드들이 하나 이상의 필드 값에 따라 순서대로 저장된 화일
- ✓ 레코드들이 일반적으로 레코드의 **탐색 키**(search key) 값의 순서에 따라 저장됨
- ✓ 탐색 키는 순차 화일을 정렬하는데 사용되는 필드를 의미
- ✓ 삽입 연산은 삽입하려는 레코드의 순서를 고려해야 하기 때문에 시간이 많이 걸릴 수 있음
- ✓ 삭제 연산은 삭제된 레코드가 사용하던 공간을 빈 공간으로 남기기 때문에 히프 화일의 경우와 마찬가지로 주기적으로 순차 화일을 재조직해야 함
- ✓ 기본 인덱스가 순차 화일에 정의되지 않는 한 순차 화일은 데이터베이스 응용을 위해 거의 사용되지 않음

6.4 화일 조직(계속)

| | | | | | |
|------|-----|----|------|---------|---|
| 1003 | 조민희 | 과장 | 4377 | 3000000 | 2 |
| 1365 | 김상원 | 사원 | 3426 | 1500000 | 1 |
| 2106 | 김창섭 | 대리 | 1003 | 2500000 | 2 |

블록 0

| | | | | | |
|------|-----|----|------|---------|---|
| 3011 | 이수민 | 부장 | 4377 | 4000000 | 3 |
| 3426 | 박영권 | 과장 | 4377 | 3000000 | 1 |
| 3427 | 최종철 | 사원 | 3011 | 1500000 | 3 |

블록 1

| | | | | | |
|------|-----|----|---|---------|---|
| 4377 | 이성래 | 사장 | ∧ | 5000000 | 2 |
|------|-----|----|---|---------|---|

블록 2

[그림 6.13] EMPLOYEE 화일을 순차 화일로 저장

6.4 화일 조직(계속)

□ 순차 화일의 성능

- ✓ EMPLOYEE 화일이 EMPNO의 순서대로 저장되어 있을 때 첫 번째 SELECT문은 이진 탐색을 이용할 수 있고, 두 번째 SELECT문의 WHERE절에 사용된 SALARY는 저장 순서와 무관하기 때문에 화일 전체를 탐색해야 함

```
SELECT  TITLE
FROM    EMPLOYEE
WHERE    EMPNO = 1365;
```

```
SELECT  EMPNAME, TITLE
FROM    EMPLOYEE
WHERE    SALARY >= 3000000 AND SALARY <= 4000000;
```

6.4 화일 조직(계속)

〈표 6.2〉 연산의 유형과 소요 시간

| 연산의 유형 | 시간 |
|----------------------|-----------|
| 삽입 | 시간이 많이 소요 |
| 삭제 | 시간이 많이 소요 |
| 탐색 키를 기반으로 탐색 | 효율적 |
| 탐색 키가 아닌 필드를 사용하여 탐색 | 시간이 많이 소요 |

6.4 화일 조직(계속)

예 : 순차 화일

6.4.1절의 히프 화일에서 예로 든 매개변수들의 값을 계속 사용하자. 이진 탐색을 한다면 「 $\log_2 500,000$ 」 = 19개의 블록을 읽어야 하고, 이에 필요한 시간은 $19 \times 10\text{ms} = 190\text{ms}$ 이다. 히프 화일에서 소요된 시간 2,500,000ms과 비교해서 1/13,158밖에 안 걸린다.

이 레코드 이후의 레코드들을 차례대로 검색하는 경우에는 연속된 레코드들이 동일한 블록이나 인접한 블록에 있을 가능성이 높다. 순서대로 레코드들이 저장된 블록들을 물리적으로 인접한 블록들에 저장함으로써 탐구 시간을 최소화할 수 있다.

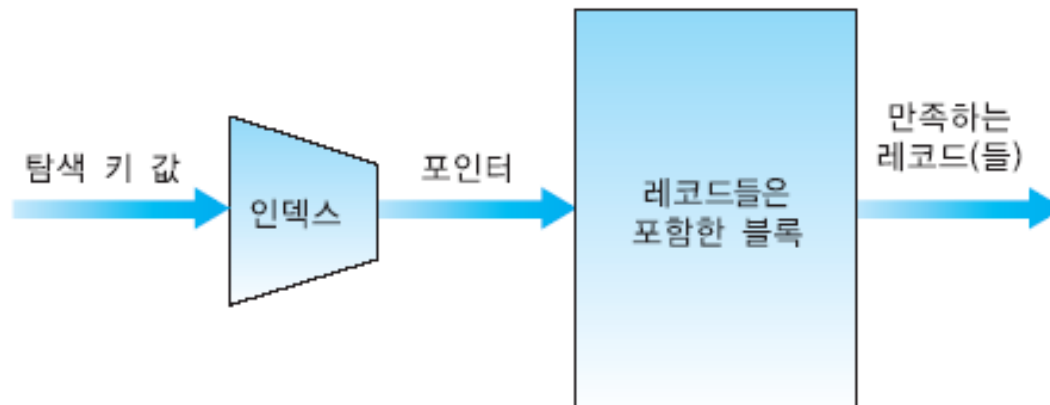
6.5 단일 단계 인덱스

□ 단일 단계 인덱스

- ✓ 인덱스된 순차 화일은 인덱스를 통해서 임의의 레코드를 접근할 수 있는 화일
- ✓ 단일 단계 인덱스의 각 엔트리는
 <탐색 키, 레코드에 대한 포인터>
- ✓ 엔트리들은 탐색 키 값의 오름차순으로 정렬됨

인덱스

탐색 키를 가진 레코드의 빠른 검색을 위해 <탐색 키, 레코드 포인터>의 쌍을 관리하는 구조



[그림 6.14] 인덱스를 통한 레코드 검색

6.5 단일 단계 인덱스(계속)

□ 단일 단계 인덱스(계속)

- ✓ 인덱스는 데이터 화일과는 별도의 화일에 저장됨
- ✓ 인덱스의 크기는 데이터 화일의 크기에 비해 훨씬 작음
- ✓ 하나의 화일에 여러 개의 인덱스들을 정의할 수 있음

6.5 단일 단계 인덱스(계속)

인덱스

| EmpnoIndex | Pointer |
|------------|---------|
| 1003 | |
| 1365 | |
| 2106 | |
| 3011 | |
| 3426 | |
| 3427 | |
| 4377 | |

EMPLOYEE

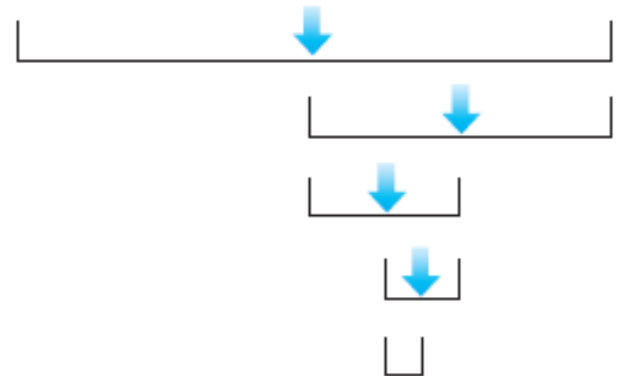
| EMPNO | EMPNAME | TITLE | MANAGER | SALARY | DNO |
|-------|---------|-------|---------|---------|-----|
| 2106 | 김창섭 | 대리 | 1003 | 2500000 | 2 |
| 3426 | 박영권 | 과장 | 4377 | 3000000 | 1 |
| 3011 | 이수민 | 부장 | 4377 | 4000000 | 3 |
| 1003 | 조민희 | 과장 | 4377 | 3000000 | 2 |
| 3427 | 최종철 | 사원 | 3011 | 1500000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

[그림 6.15] EMPLOYEE 화일의 EMPNO에 정의된 인덱스

6.5 단일 단계 인덱스(계속)

□ 단일 단계 인덱스(계속)

- ✓ 인덱스가 정의된 필드를 **탐색 키**라고 부름
- ✓ 탐색 키의 값들은 후보 키처럼 각 투플마다 반드시 고유하지는 않음
- ✓ 키를 구성하는 애트리뷰트뿐만 아니라 어떤 애트리뷰트도 탐색 키로 사용될 수 있음
- ✓ 인덱스의 엔트리들은 탐색 키 값의 오름차순으로 저장되어 있으므로 이진 탐색을 이용할 수도 있음



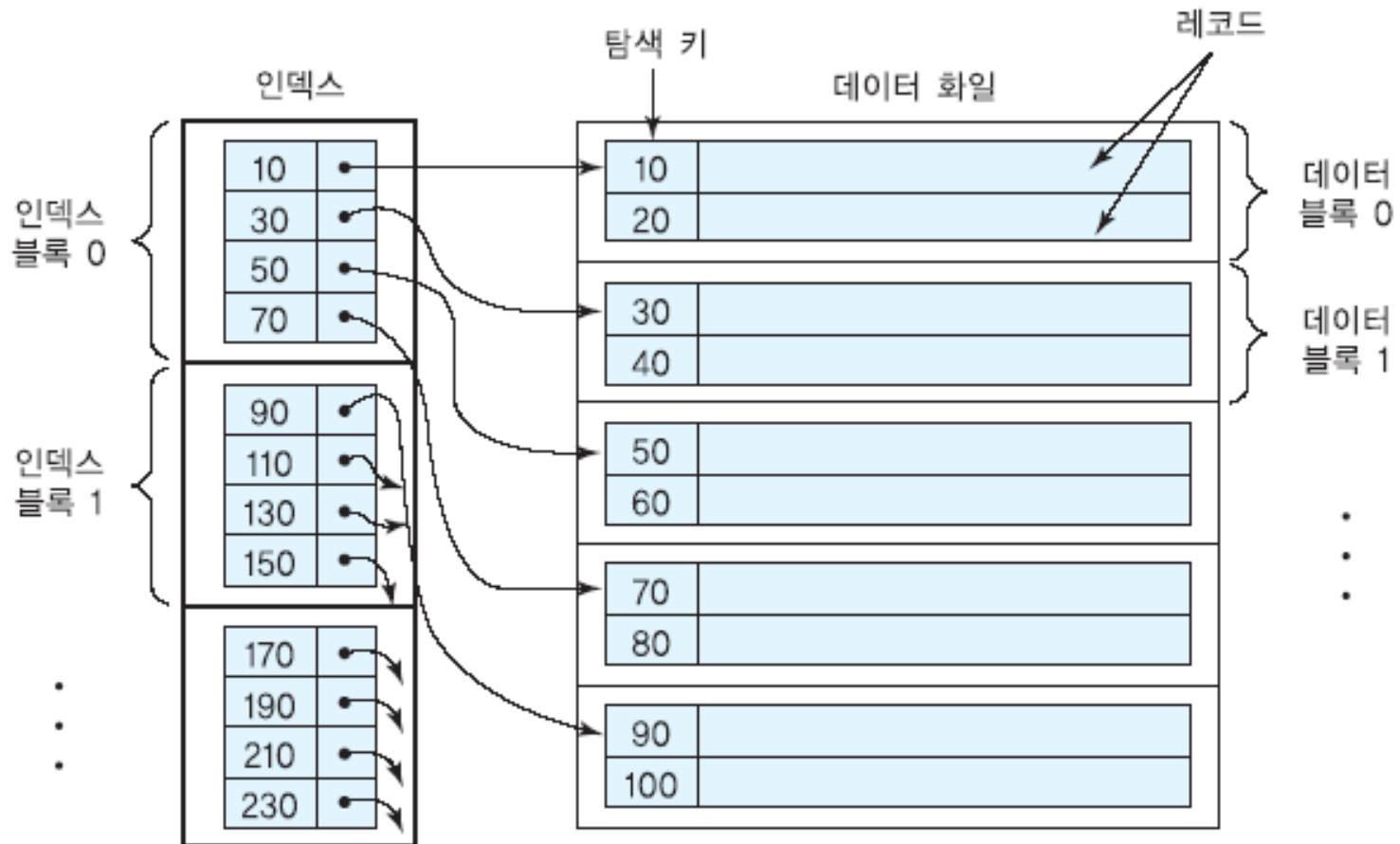
[그림 6.16] 이진 탐색

6.5 단일 단계 인덱스(계속)

□ 기본 인덱스(primary index)

- ✓ 탐색 키가 데이터 화일의 기본 키인 인덱스를 기본 인덱스라고 부름
- ✓ 기본 인덱스는 기본 키의 값에 따라 정렬된 데이터 화일에 대해 정의됨
- ✓ 기본 인덱스는 흔히 희소 인덱스로 유지할 수 있음
- ✓ 각 릴레이션마다 최대한 한 개의 기본 인덱스를 가질 수 있음

6.5 단일 단계 인덱스(계속)



[그림 6.17] 데이터 파일에 대한 기본(또는 희소) 인덱스

6.5 단일 단계 인덱스(계속)

예 : 기본 인덱스

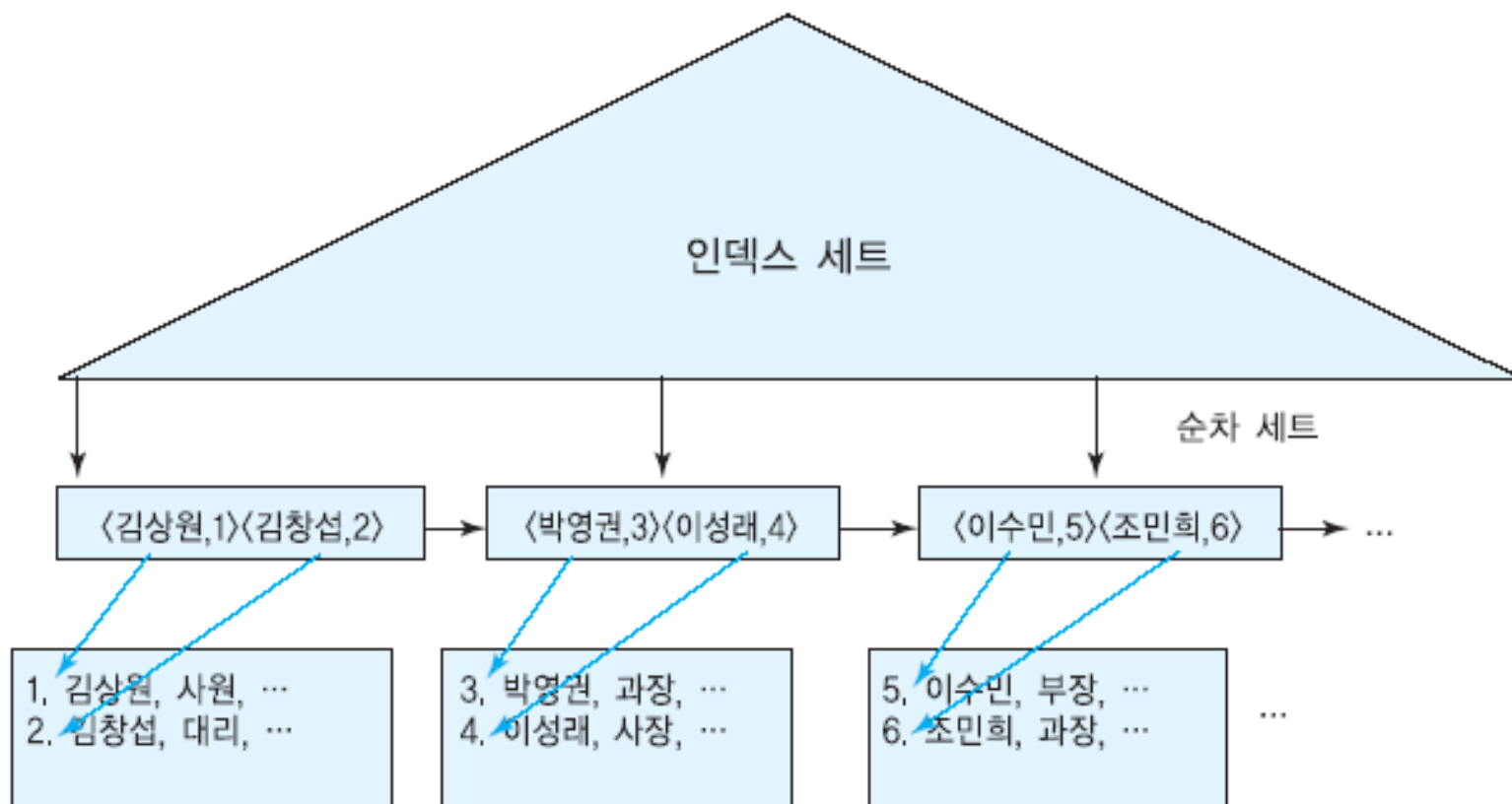
6.4.1절의 히프 화일에서 예로 든 매개변수들의 값을 계속 사용하자. 데이터 화일의 각 블록마다 하나의 인덱스 엔트리가 인덱스에 들어 있다. 블록 포인터는 4바이트이고 키 필드의 길이는 20바이트라고 가정하면, 한 인덱스 엔트리의 길이는 24바이트이다. 인덱스 블록킹 인수는 $\lfloor 4,096/24 \rfloor = 170$ 이므로 인덱스 블록당 170개의 인덱스 엔트리가 들어간다. 데이터 블록의 개수가 500,000이므로 인덱스의 크기는 $\lceil 500,000/170 \rceil = 2,942$ 블록이다. 인덱스에서 이진 탐색을 이용하여 하나의 레코드를 찾는데 필요한 블록 접근 횟수는 $\lceil \log_2 2942 \rceil + 1 = 13$ 이다. 여기서 1은 레코드가 들어 있는 데이터 블록을 접근하는 횟수이다. 따라서 레코드를 탐색하는데 걸리는 시간은 $13 \times 10\text{ms} = 130\text{ms}$ 이다. 순차 화일에서 소요된 시간 190ms와 비교하면 32%만큼 검색 속도가 향상되었다. 다단계 인덱스를 통해서 이 시간을 더 크게 줄일 수 있다.

6.5 단일 단계 인덱스(계속)

□ 클러스터링 인덱스(clustering index)

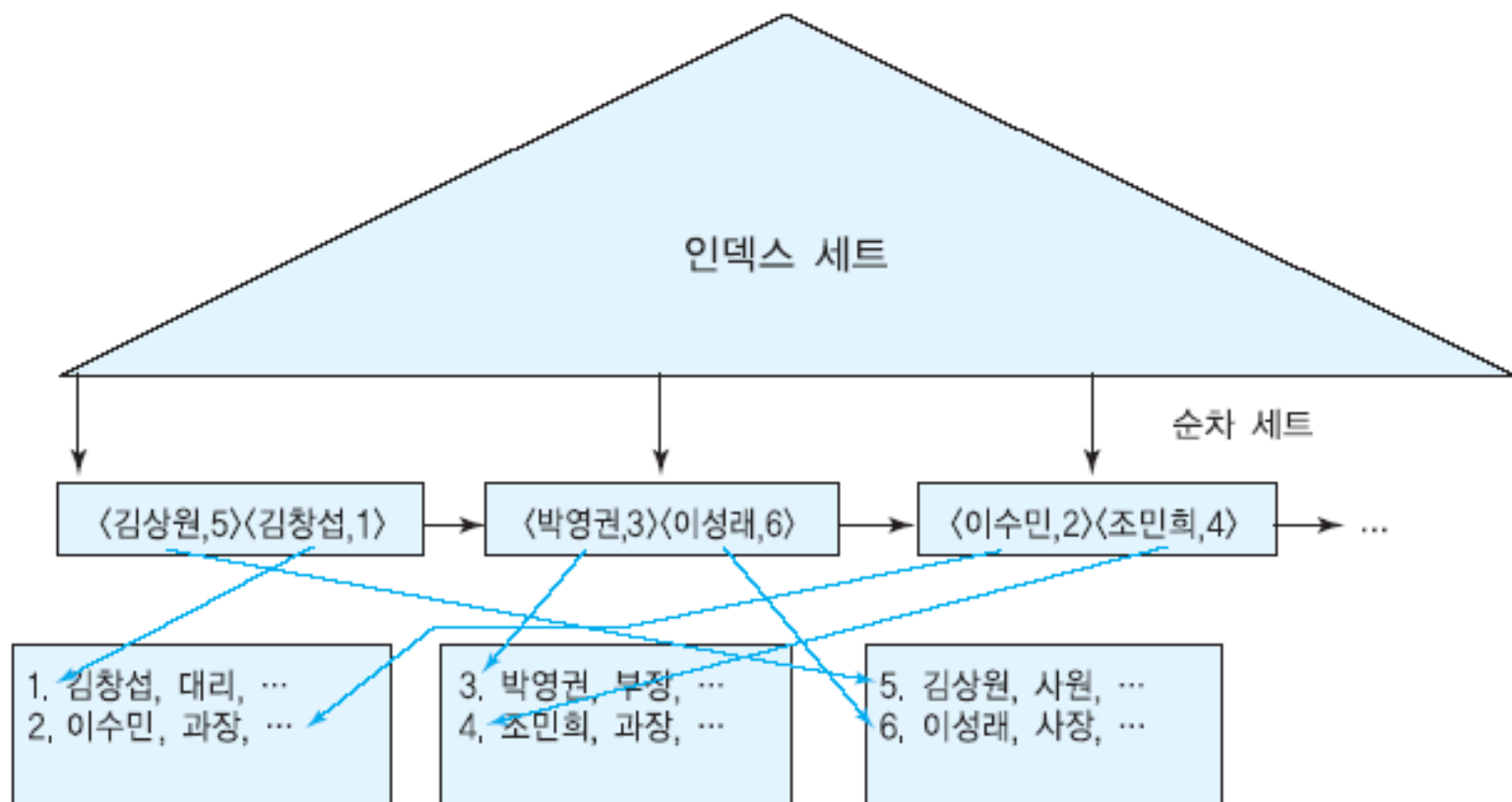
- ✓ 탐색 키 값에 따라 정렬된 데이터 화일에 대해 정의됨
- ✓ 각각의 상이한 키 값마다 하나의 인덱스 엔트리가 인덱스에 포함됨
- ✓ 범위 질의에 유용
- ✓ 범위의 시작 값에 해당하는 인덱스 엔트리를 먼저 찾고, 범위에 속하는 인덱스 엔트리들을 따라가면서 레코드들을 검색할 때 디스크에서 읽어오는 블록 수가 최소화됨
- ✓ 어떤 인덱스 엔트리에서 참조되는 데이터 블록을 읽어오면 그 데이터 블록에 들어 있는 대부분의 레코드들은 범위를 만족함

6.5 단일 단계 인덱스(계속)



[그림 6.18] 클러스터링 인덱스

6.5 단일 단계 인덱스(계속)



[그림 6.19] 비 클러스터링 인덱스

6.5 단일 단계 인덱스(계속)

예 : 클러스터링 인덱스

6.4.1절의 히프 화일에서 예로 든 매개변수들의 값을 계속 사용하자. 키가 아닌 탐색 키 필드에 인덱스가 정의된다. 총 10,000,000개의 레코드들에 대해 800,000개의 서로 상이한 탐색 키 값들이 있다고 가정하자. 또한 인덱스가 정의되는 필드의 길이가 20바이트라고 가정하자. 블록당 인덱스 엔트리 수는 $\lfloor 4,096/24 \rfloor = 170$ 이다. 인덱스의 크기는 $\lceil 800,000/170 \rceil = 4,706$ 블록이다. 인덱스에서 어떤 키값을 만족하는 첫 번째 레코드를 탐색하는데 필요한 블록 접근 횟수는 $\lceil \log_2 4706 \rceil + 1 = 14$ 이다. 여기서 1은 레코드가 들어 있는 데이터 블록을 접근하는 횟수이다. 따라서 레코드를 탐색하는데 걸리는 시간은 $14 \times 10\text{ms} = 140\text{ms}$ 이다.

6.5 단일 단계 인덱스(계속)

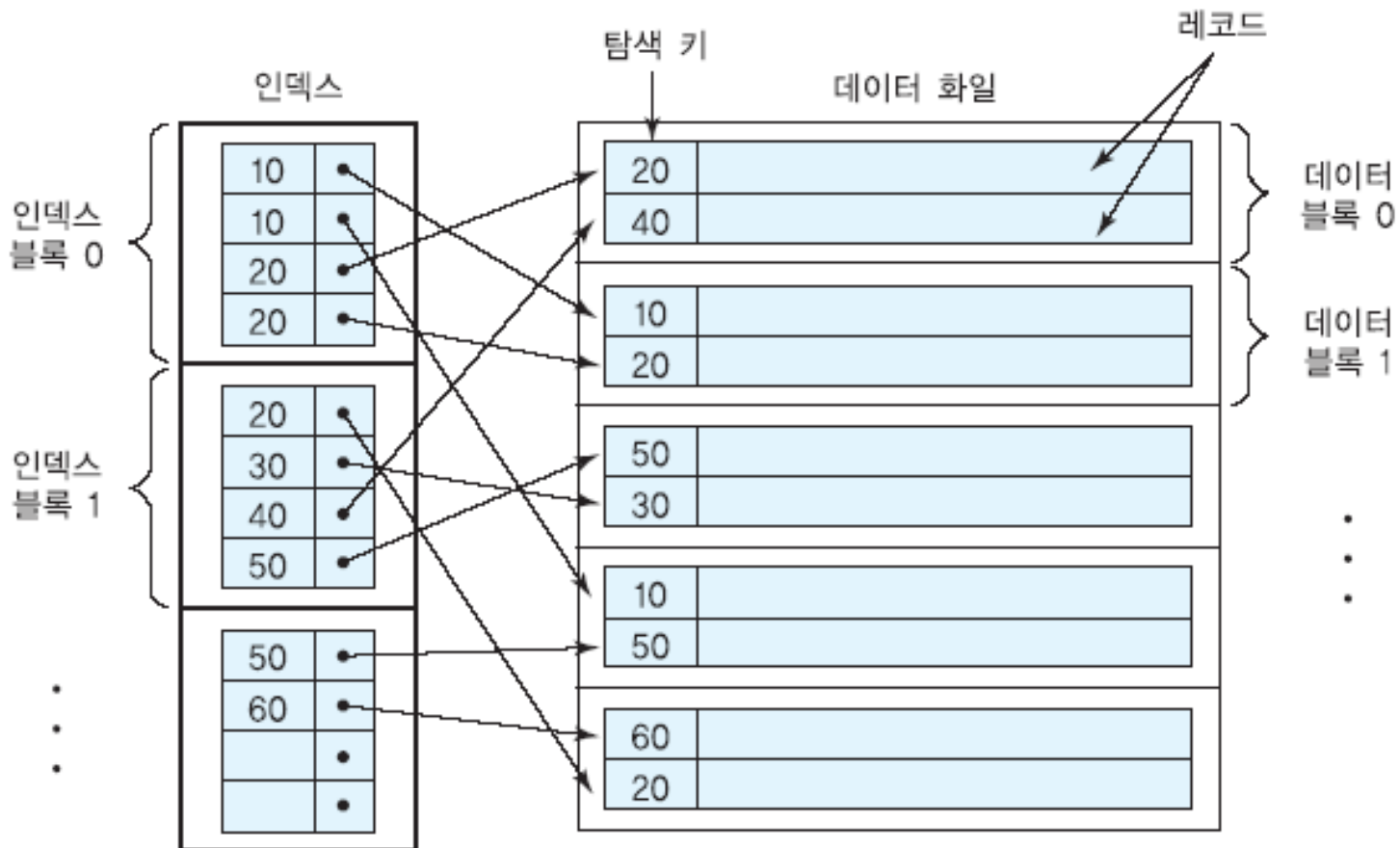
□ 보조 인덱스(secondary index)

- ✓ 한 화일은 기껏해야 한 가지 필드들의 조합에 대해서만 정렬될 수 있음
- ✓ 보조 인덱스는 탐색 키 값에 따라 정렬되지 않은 데이터 화일에 대해 정의됨
- ✓ 보조 인덱스는 일반적으로 밀집 인덱스이므로 같은 수의 레코드들을 접근할 때 보조 인덱스를 통하면 기본 인덱스를 통하는 경우보다 디스크 접근 횟수가 증가할 수 있음

↳ 모든 레코드에 대한 인덱스 거짐

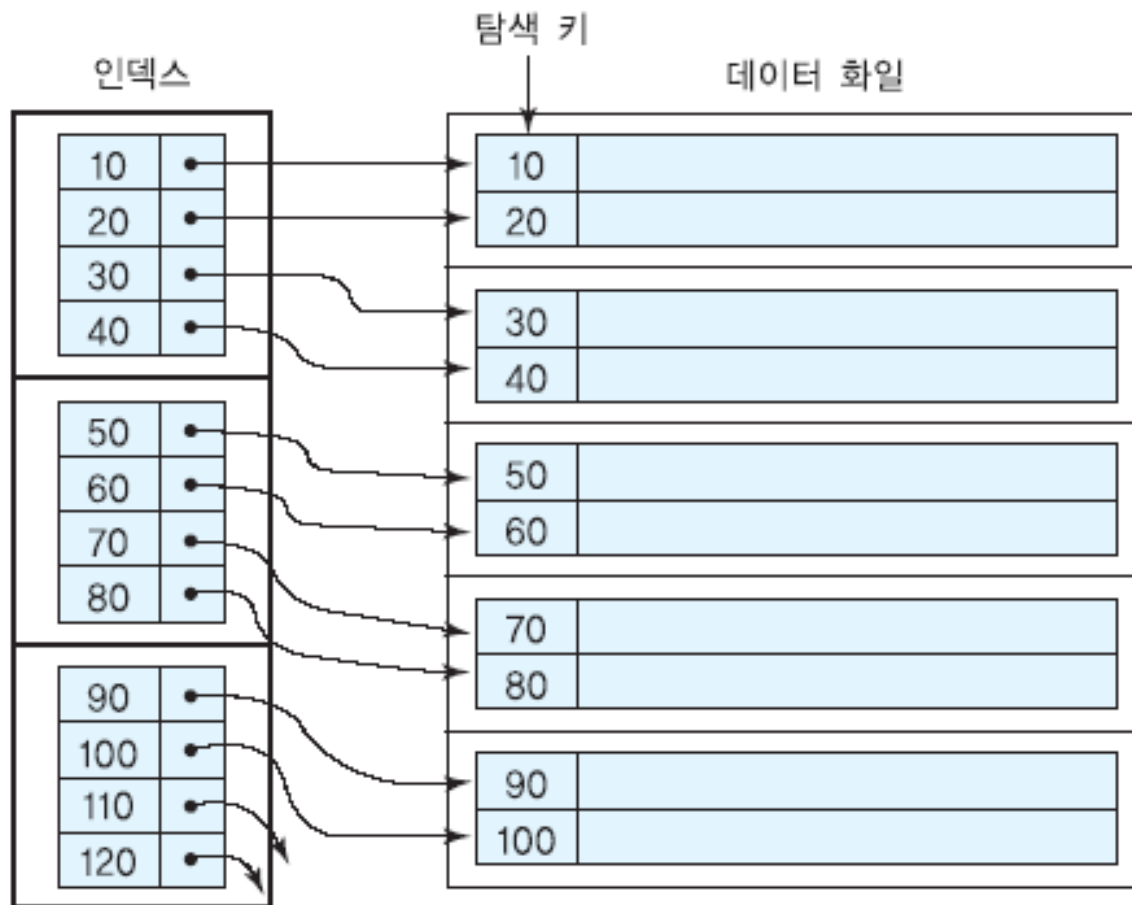
인덱스 블록마다 레코드마다.
크기 기본 < 보조

6.5 단일 단계 인덱스(계속)



[그림 6.20] 보조(밀집) 인덱스

6.5 단일 단계 인덱스(계속)



[그림 6.21] 데이터 화일에 대한 밀집 인덱스

6.5 단일 단계 인덱스(계속)

예 : 밀집 인덱스

6.4.1절의 히프 화일에서 예로 든 매개변수들의 값을 계속 사용하자. 데이터 화일의 각 레코드마다 하나의 인덱스 엔트리가 인덱스에 들어 있다. 블록 포인터는 4바이트이고 키 필드의 길이는 20바이트라고 가정한다. 한 인덱스 엔트리의 길이는 24바이트이다. 인덱스 블록킹 인수는 $\lfloor 4,096/24 \rfloor = 170$ 이므로 인덱스 블록당 170개의 인덱스 엔트리가 들어간다. 레코드의 개수가 10,000,000이므로 인덱스의 크기는 $\lceil 10,000,000/170 \rceil = 58,824$ 블록이다. 인덱스에서 이진 탐색을 이용하여 하나의 레코드를 찾는데 필요한 블록 접근 횟수는 $\lceil \log_2 58824 \rceil + 1 = 17$ 이다. 여기서 1은 레코드가 들어 있는 데이터 블록을 접근하는 횟수이다. 따라서 레코드를 탐색하는데 걸리는 시간은 $17 \times 10\text{ms} = 170\text{ms}$ 이다.

6.5 단일 단계 인덱스(계속)

□ 희소 인덱스와 밀집 인덱스의 비교

- ✓ 희소 인덱스는 각 데이터 블록마다 한 개의 엔트리를 갖고, 밀집 인덱스는 각 레코드마다 한 개의 엔트리를 가짐 ↳ 애널 노트 있음
- ✓ 레코드의 길이가 블록 크기보다 훨씬 작은 일반적인 경우에는 희소 인덱스의 엔트리 수가 밀집 인덱스의 엔트리 수보다 훨씬 적음
- ✓ 희소 인덱스는 일반적으로 밀집 인덱스에 비해 인덱스 단계 수가 1정도 적으므로 인덱스 탐색시 디스크 접근 수가 1만큼 적을 수 있음

6.5 단일 단계 인덱스(계속)

□ 희소 인덱스와 밀집 인덱스의 비교(계속)

- ✓ 희소 인덱스는 밀집 인덱스에 비해 모든 갱신과 대부분의 질의에 대해 더 효율적
- ✓ 그러나 질의에서 인덱스가 정의된 애트리뷰트만 검색(예를 들어, COUNT 질의)하는 경우에는 데이터 화일을 접근할 필요 없이 인덱스만 접근해서 질의를 수행할 수 있으므로 밀집 인덱스가 희소 인덱스보다 유리
- ✓ 한 화일은 한 개의 희소 인덱스와 다수의 밀집 인덱스를 가질 수 있음

6.5 단일 단계 인덱스(계속)

□ 클러스터링 인덱스와 보조 인덱스의 비교

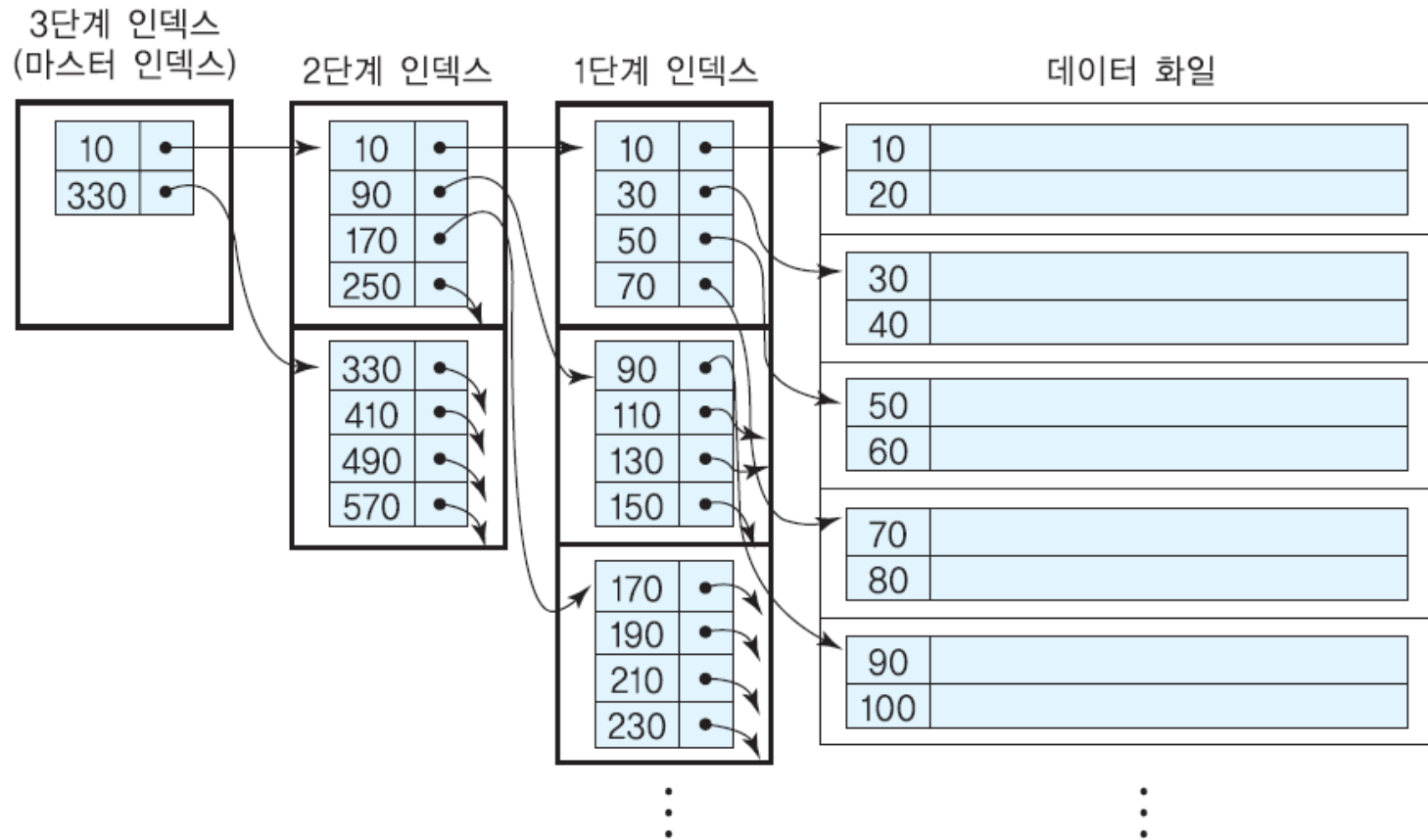
- ✓ 클러스터링 인덱스는 희소 인덱스일 경우가 많으며 범위 질의 등에 좋음
- ✓ 보조 인덱스는 밀집 인덱스이므로 일부 질의에 대해서는 화일을 접근할 필요 없이 처리할 수 있음

6.6 다단계 인덱스

□ 다단계 인덱스

- ✓ 인덱스 자체가 클 경우에는 인덱스를 탐색하는 시간도 오래 걸릴 수 있음
- ✓ 인덱스 엔트리를 탐색하는 시간을 줄이기 위해서 단일 단계 인덱스를 디스크 상의 하나의 순서 화일로 간주하고, 단일 단계 인덱스에 대해서 다시 인덱스를 정의할 수 있음
- ✓ 다단계 인덱스는 가장 상위 단계의 모든 인덱스 엔트리들이 한 블록에 들어갈 수 있을 때까지 이런 과정을 반복함
- ✓ 가장 상위 단계 인덱스를 **마스터 인덱스**(master index)라고 부름
- ✓ 마스터 인덱스는 **한 블록으로 이루어지기 때문에** 주기억 장치에 상주할 수 있음
↳ 버퍼에 저장이 됨
- ✓ 대부분의 다단계 인덱스는 **B⁺-트리**를 사용

6.6 다단계 인덱스(계속)



[그림 6.22] 다단계 인덱스

6.6 다단계 인덱스(계속)

예 : 기본 키에 대한 다단계 인덱스

6.4.1절의 히프 화일과 기본 인덱스에서 예로 든 매개변수들의 값을 계속 사용하자.

- 1단계 인덱스의 블록 수는 2,942
- 2단계 인덱스의 블록 수는 $\lceil 2,942/170 \rceil = 18$
- 3단계 인덱스의 블록 수는 $\lceil 18/170 \rceil = 1 \Rightarrow$ 주기억 장치에 상주할 수 있음

원하는 레코드를 탐색할 때, 3단계 인덱스는 주기억 장치에 상주하므로 디스크 접근이 필요 없고, 2단계 인덱스를 구성하는 18개의 블록 중에서 한 블록, 2,942개의 1단계 인덱스 블록 중에서 한 블록, 데이터 화일의 500,000개의 블록 중에서 한 블록 등 총 세 번의 디스크 접근만 하면 된다. 따라서 탐색 시간은 $3 \times 10\text{ms} = 30\text{ms}$ 이다. 기본 인덱스를 단일 단계 인덱스로 구성한 경우의 130ms와 비교하면 인덱스를 탐색하는 시간이 1/4.3 정도밖에 안 걸린다.

6.6 다단계 인덱스(계속)

□ SQL의 인덱스 정의문

- ✓ SQL의 CREATE TABLE문에서 **PRIMARY KEY**절로 명시한 애트리뷰트에 대해서는 DBMS가 자동적으로 기본 인덱스를 생성
- ✓ UNIQUE로 명시한 애트리뷰트에 대해서는 DBMS가 자동적으로 보조 인덱스를 생성
- ✓ SQL2는 인덱스 정의 및 제거에 관한 표준 SQL문을 제공하지 않음
- ✓ 다른 애트리뷰트에 추가로 인덱스를 정의하기 위해서는 DBMS마다 다소 구문이 다른 **CREATE INDEX**문을 사용해야 함

6.6 다단계 인덱스(계속)

□ 다수의 애트리뷰트를 사용한 인덱스 정의

- ✓ 한 릴레이션에 속하는 두 개 이상의 애트리뷰트들의 조합에 대하여 하나의 인덱스를 정의할 수 있음

- ✓ 예:

```
CREATE INDEX EmpIndex ON EMPLOYEE (DNO, SALARY);
```

- ✓ 이 인덱스는 아래의 질의에 활용될 수 있음

```
SELECT          *  
FROM EMPLOYEE  
WHERE          DNO=3 AND SALARY = 4000000;
```

6.6 다단계 인덱스(계속)

□ 다수의 애트리뷰트를 사용한 인덱스 정의(계속)

- ✓ 이 인덱스는 아래의 질의에도 활용될 수 있음

```
SELECT      *
FROM        EMPLOYEE
WHERE       DNO >= 2 AND DNO <= 3 AND SALARY >= 3000000
            AND SALARY <= 4000000;
```

- ✓ 이 인덱스는 아래의 질의에도 활용될 수 있음

```
SELECT      *
FROM        EMPLOYEE
WHERE       DNO = 2;           (또는 DNO에 대한 범위 질의)
```

- ✓ 이 인덱스는 아래의 질의에는 활용될 수 없음

```
SELECT      *
FROM        EMPLOYEE
WHERE       SALARY >= 3000000
            AND SALARY <= 4000000;

(또는 SALARY에 대한 동등 조건)
```

6.6 다단계 인덱스(계속)

□ 인덱스의 장점과 단점

- ✓ 인덱스는 검색 속도를 향상시키지만 인덱스를 저장하기 위한 공간이 추가로 필요하고 삽입, 삭제, 수정 연산의 속도는 저하시킴
- ✓ 소수의 레코드들을 수정하거나 삭제하는 연산의 속도는 향상됨
- ✓ 릴레이션이 매우 크고, 질의에서 릴레이션의 튜플들 중에 일부(예, 2%~4%)를 검색하고, WHERE절이 잘 표현되었을 때 특히 성능에 도움이 됨

6.7 인덱스 선정 지침과 데이터베이스 튜닝

□ 인덱스 선정 지침과 데이터베이스 튜닝

- ✓ 가장 중요한 질의들과 이들의 수행 빈도, 가장 중요한 갱신들과 이들의 수행 빈도, 이와 같은 질의와 갱신들에 대한 바람직한 성능들을 고려하여 인덱스를 선정함
- ✓ 워크로드 내의 각 질의에 대해 이 질의가 어떤 릴레이션들을 접근하는가, 어떤 애트리뷰트들을 검색하는가, WHERE절의 선택/조인 조건에 어떤 애트리뷰트들이 포함되는가, 이 조건들의 선별력은 얼마인가 등을 고려함
- ✓ 워크로드 내의 각 갱신에 대해 이 갱신이 어떤 릴레이션들을 접근하는가, WHERE절의 선택/조인 조건에 어떤 애트리뷰트들이 포함되는가, 이 조건들의 선별력은 얼마인가, 갱신의 유형(INSERT/DELETE/UPDATE), 갱신의 영향을 받는 애트리뷰트 등을 고려함

6.7 인덱스 선정 지침과 데이터베이스 튜닝(계속)

□ 인덱스 선정 지침과 데이터베이스 튜닝(계속)

- ✓ 어떤 릴레이션에 인덱스를 생성해야 하는가, 어떤 애트리뷰트를 탐색 키로 선정해야 하는가, 몇 개의 인덱스를 생성해야 하는가, 각 인덱스에 대해 클러스터링 인덱스, 밀집 인덱스/희소 인덱스 중 어느 유형을 선택할 것인가 등을 고려함
- ✓ 인덱스를 선정하는 한 가지 방법은 가장 중요한 질의들을 차례로 고려해보고, 현재의 인덱스가 최적의 계획에 적합한지 고려해보고, 인덱스를 추가하면 더 좋은 계획이 가능한지 알아봄
- ✓ 물리적 데이터베이스 설계는 끊임없이 이루어지는 작업

6.7 인덱스 선정 지침과 데이터베이스 튜닝(계속)

□ 인덱스를 결정하는데 도움이 되는 몇 가지 지침

- ✓ 지침 1: 기본 키는 클러스터링 인덱스를 정의할 훌륭한 후보
- ✓ 지침 2: 외래 키도 인덱스를 정의할 중요한 후보
- ✓ 지침 3: 한 애트리뷰트에 들어 있는 상이한 값들의 개수가 거의 전체 레코드 수와 비슷하고, 그 애트리뷰트가 동등 조건에 사용된다면 비 클러스터링 인덱스를 생성하는 것이 좋음
- ✓ 지침 4: 튜플이 많이 들어 있는 릴레이션에서 대부분의 질의가 검색하는 튜플이 2% ~ 4% 미만인 경우에는 인덱스를 생성
- ✓ 지침 5: 자주 갱신되는 애트리뷰트에는 인덱스를 정의하지 않는 것이 좋음
- ✓ 지침 6: 갱신이 빈번하게 이루어지는 릴레이션에는 인덱스를 많이 만드는 것을 피함

6.7 인덱스 선정 지침과 데이터베이스 튜닝(계속)

□ 인덱스를 결정하는데 도움이 되는 몇 가지 지침(계속)

- ✓ 지침 7: 후보 키는 기본 키가 갖는 모든 특성을 마찬가지로 갖기 때문에 인덱스를 생성할 후보
- ✓ 지침 8: 인덱스는 화일의 레코드들을 충분히 분할할 수 있어야 함
- ✓ 지침 9: 정수형 애트리뷰트에 인덱스를 생성
- ✓ 지침 10: VARCHAR 애트리뷰트에는 인덱스를 만들지 않음
- ✓ 지침 11: 작은 화일에는 인덱스를 만들 필요가 없음
- ✓ 지침 12: 대량의 데이터를 삽입할 때는 모든 인덱스를 제거하고, 데이터 삽입이 끝난 후에 인덱스들을 다시 생성하는 것이 좋음

6.7 인덱스 선정 지침과 데이터베이스 튜닝(계속)

□ 언제 인덱스가 사용되지 않는가?

- ✓ 시스템 카탈로그가 오래 전의 데이터베이스 상태를 나타냄
- ✓ DBMS의 질의 최적화 모듈이 릴레이션의 크기가 작아서 인덱스가 도움이 되지 않는다고 판단함
- ✓ 인덱스가 정의된 애트리뷰트에 산술 연산자가 사용됨

```
SELECT      *  
FROM        EMPLOYEE  
WHERE       SALARY * 12 > 40000000;
```

- ✓ DBMS가 제공하는 내장 함수가 집단 함수 대신에 사용됨

```
SELECT      *  
FROM        EMPLOYEE  
WHERE       SUBSTR(EMPNAME, 1, 1) = '김';
```

6.7 인덱스 선정 지침과 데이터베이스 튜닝(계속)

❑ 언제 인덱스가 사용되지 않는가?(계속)

- ✓ 널값에 대해서는 일반적으로 인덱스가 사용되지 않음

```
SELECT  *  
FROM    EMPLOYEE  
WHERE   MANAGER IS NULL;
```

6.7 인덱스 선정 지침과 데이터베이스 튜닝(계속)

❑ 질의 튜닝을 위한 추가 지침

- ✓ DISTINCT절의 사용을 최소화하라
- ✓ GROUP BY절과 HAVING절의 사용을 최소화하라
- ✓ 임시 릴레이션의 사용을 피하라
- ✓ SELECT * 대신에 SELECT절에 애트리뷰트 이름들을 구체적으로 명시하라