데이터베이스

- 순서 : Ch6(물리적 데이터베이스 설계)
- 학기 : 2018학년도 2학기
- 학과 : 가천대학교 컴퓨터공학과 2학년
- 교수 : 박양재

데이터베이스

- 목차

6장. 물리적 데이터베이스 설계

❑ 물리적 데이터베이스 설계

- ✓ 논리적인 설계의 데이터 구조를 보조 기억 장치상의 파일(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

  -너무 많이 인덱스 정의하면 데이터베이스 갱신 시 시간이 많이 소요

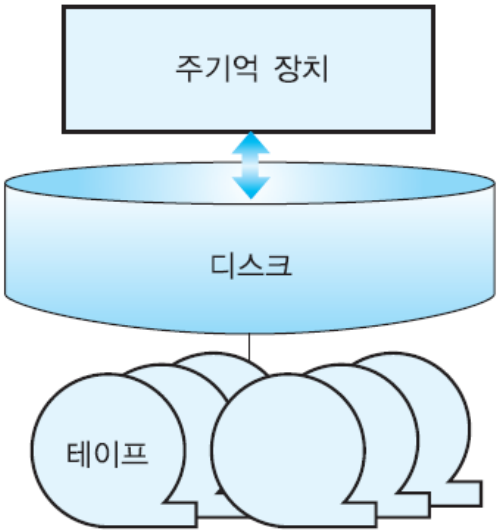
  -너무 적은 인덱스를 정의하면 검색연산이 효율적으로 지원되지 않는다.
- ✓ 대부분 논리적인 데이터베이스 설계는 바로 물리적 데이터베이스 설계로 사상되나 데이터베이스의 사용 패턴에 관한 정보를 추가하여 성능 향상가능.

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
  - ✓ 보조기억장치->주기억장치로 이동하는 데이터 단위는 블록,주기억장치는 페이지 단위
  - ✓ 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)

- 주기억 장치, 디스크 –임의접근장치(Random Access)
- 자기 테이프 –순차접근장치(Sequential Access)



[그림 6.1] 저장 장치의 계층 구조

6.1 보조 기억 장치(계속)

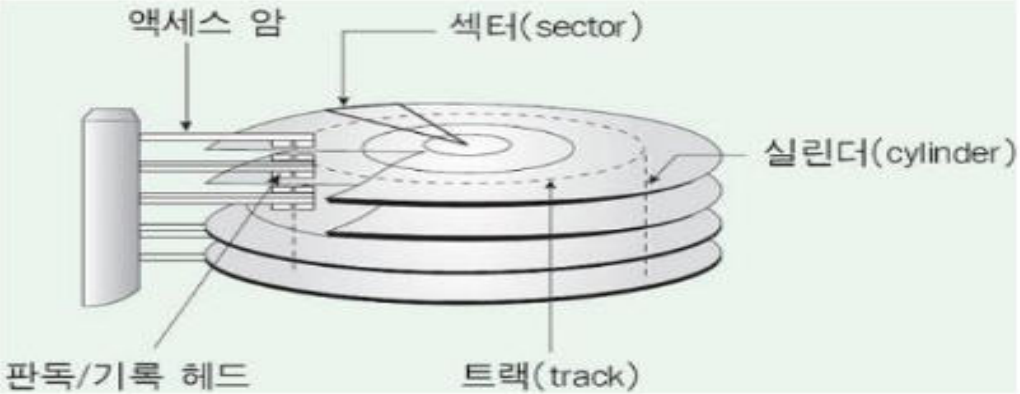
❑ 자기 디스크

- ✓ 디스크는 자기 물질로 만들어진 여러 개의 판으로 이루어짐
- ✓ 각 면마다 디스크 헤드가 있음
- ✓ 각 판은 트랙과 섹터로 구분됨
- ✓ 정보는 디스크 표면 상의 동심원(트랙)을 따라 저장됨
- ✓ 여러 개의 디스크 면 중에서 같은 지름을 갖는 트랙들을 실린더라고 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
- ✓ DBMS는 가능한 한 순차접근을 이용하려고 한다.Why?
- ✓ 병목지점-디스크와 주기억 장치 간의 데이터 전송

6.1 보조 기억 장치(계속)

자기디스크(Magnetic Disk)

- 여러 장의 디스크 원판을 압축하여 자성체를 입힌 기록 매체
- 순차 접근(SASD)과 직접 접근(DASD)이 모두 가능
- 자기디스크 구성



자기디스크 구성

트랙(Track)	정보가 저장되는 일정한 간격의 동심원
섹터(Sector)	트랙을 나눈 작은 조각으로 512Byte의 정보를 저장할 수 있음
실린더(Cylinder)	디스크에서 중심축으로부터 같은 위치에 있는 트랙의 모임
클러스터(Cluster)	• 여러 개의 섹터를 묶어 놓은 것 • 실제 데이터를 읽고 쓰는 단위
헤드(Head)	디스크 내의 자료에 대한 읽기/쓰기를 수행하는 장치
탐색 시간(Seek Time)	디스크에서 읽기/쓰기 헤드가 원하는 트랙을 찾는데 걸리는 시간
전송 시간(Transfer Time)	읽은 데이터를 주기억장치로 보내는 시간

6.2 버퍼 관리와 운영 체제

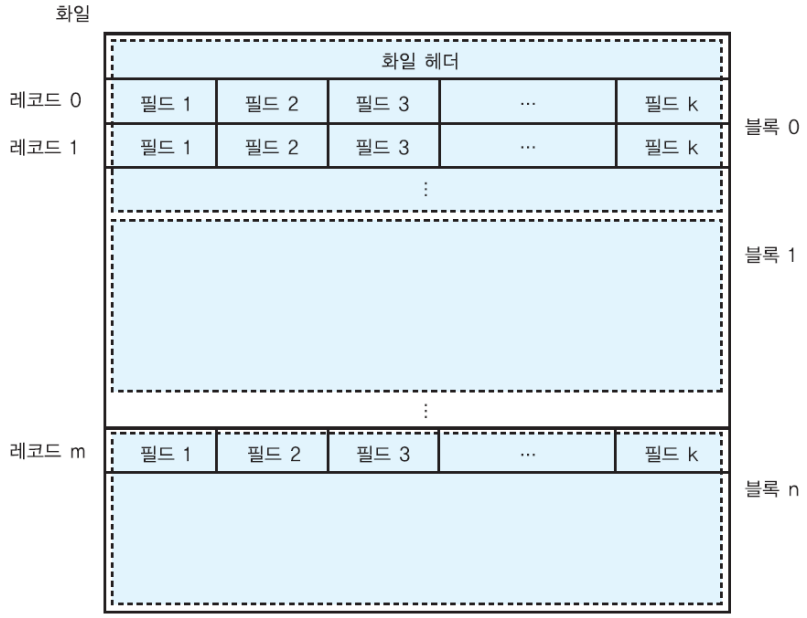
❑ 버퍼 관리와 운영 체제

- ✓ 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
- ✓ 한정된 버퍼공간에 어느 블록들을 주기억장치에 유지할 것인가?-버퍼가 꽉 찾을 때 새로운 블록을 디스크에서 읽어 오려면 가장 오래 전에 참조된 블록(LRU:Least Recently Used)을 먼저 디스크로 보내고, 그 자리에 읽어온다.LRU 알고리즘은 블록전송 횟수를 줄이기 위해서 주기억 장치 내의 버퍼에 최근에 접근된 블록을 유지한다. 어떤 블록에 대한 요청은 디스크에 접근할 필요없이 주기억 장치 내의 버퍼에서 처리하여 블록전송 횟수를 줄여 성능 향상

6.3 디스크 상에서 화일의 레코드 배치

❑ 디스크 상에서 화일의 레코드 배치

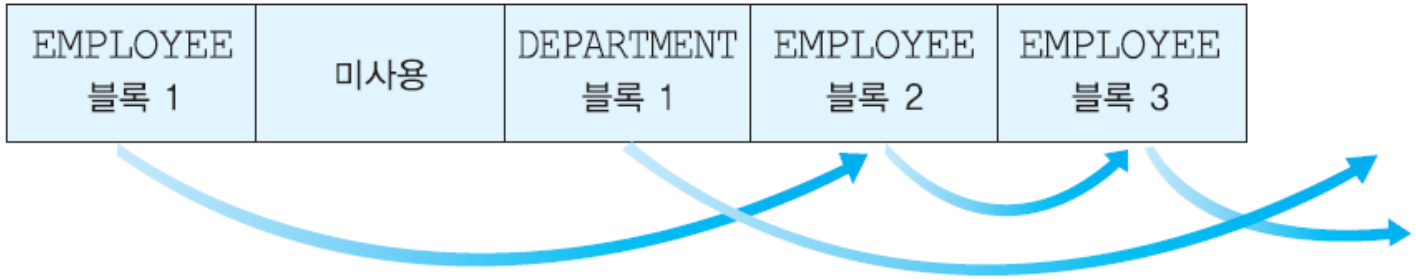
- ✓ 릴레이션의 애트리뷰트는 고정 길이 또는 가변 길이의 필드로 표현됨
- ✓ 연관된 필드들이 모여서 고정 길이 또는 가변 길이의 레코드가 됨
- ✓ 한 릴레이션을 구성하는 레코드들의 모임이 화일이라고 부르는 블록들의 모임에 저장됨



[그림 6.4] 화일과 블록과 레코드

6.3 디스크 상에서 화일의 레코드 배치(계속)

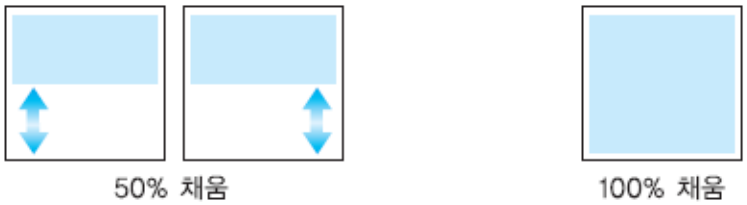
- 디스크 상에서 화일의 레코드 배치(계속)
  - ✓ 한 화일에 속하는 블록들이 반드시 인접해 있을 필요는 없음
  - ✓ 인접한 블록들을 읽는 경우에는 탐구 시간과 회전 지연 시간이 들지 않기 때문에 입출력 속도가 빠르므로 블록들이 인접하도록 한 화일의 블록들을 재조직할 수 있음



[그림 6.5] 디스크에서 블록들의 연결

6.3 디스크 상에서 화일의 레코드 배치(계속)

- 한 블록에는 많은 레코드가 들어가는데 레코드 길이가 블록 크기를 초과하면 한 레코드를 두개의 블록에 저장한다.신장 레코드(spanned record)
- BLOB(Binary Large Object) 선언된 애트리뷰트
  - ✓ 이미지(GIF, JPG), 동영상(MPEG, RM) 등 대규모 크기의 데이터를 저장하는데 사용됨
  - ✓ BLOB의 최대 크기는 오라클에서 8TB~128TB
- 채우기 인수(Fill factor)
  - ✓ 각 블록에 레코드를 채우는 공간의 비율
  - ✓ 빈공간 남겨두는 이유-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(투플)
  - ✓ 레코드 길이가 n바이트인 고정 길이 레코드에서 레코드 i를 접근하기 위해서는 n*(i-1)+1의 위치에서 레코드를 읽음
  - ✓ 예: DEPARTMENT 릴레이션에 해당하는 파일에서 각 레코드 길이가 18바이트(DEPTNO:4바이트, DEPTNAME:10바이트, FLOOR:4바이트)
  - ✓ 3번째 레코드를 읽어오려면 18*(3-1)+1=37번째 바이트부터 18바이트 읽어오면 된다.

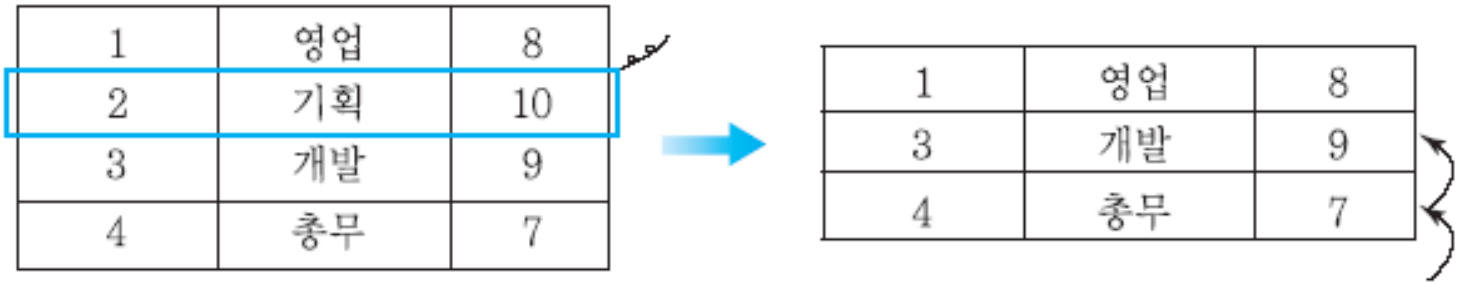
1	영업	8
2	기획	10
3	개발	9
4	총무	7

← 18바이트 →

[그림 6.7] 고정 길이 레코드 읽기

6.3 디스크 상에서 화일의 레코드 배치(계속)

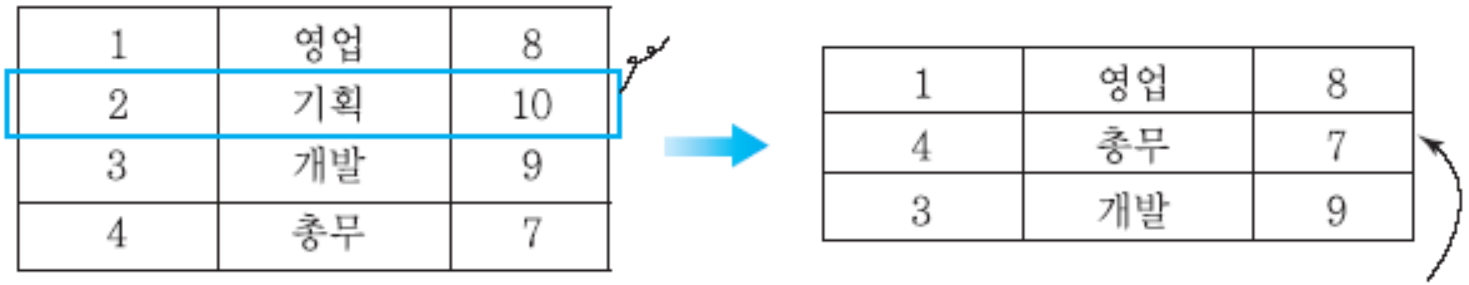
❑ 고정 길이 레코드(계속)



1	영업	8
2	기획	10
3	개발	9
4	총무	7

1	영업	8
3	개발	9
4	총무	7

[그림 6.8] 고정 길이 레코드 삭제시 여러 개의 레코드를 이동



1	영업	8
2	기획	10
3	개발	9
4	총무	7

1	영업	8
4	총무	7
3	개발	9

[그림 6.9] 고정 길이 레코드 삭제시 한 개의 레코드를 이동

6.3 디스크 상에서 화일의 레코드 배치(계속)

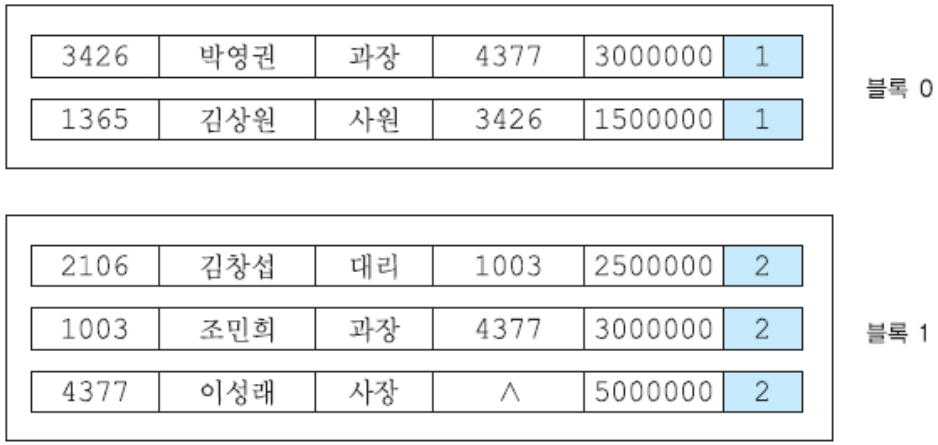
❑ 클러스터링(clustering)

- ✓ 레코드들을 블록에 넣을 때 질의 처리 시 입출력을 최소화하기 위해서 하나의 질의에서 함께 요구될 정보를 동일한 블록에 저장하는 것.
- ✓ 화일 내 클러스터링 (intra-file clustering): 한 파일 내에서 함께 검색될 가능성이 높은 레코드들을 디스크 상에서 물리적으로 가까운 곳에 모아 저장하는 것
- ✓ 파일 간의 클러스터링(inter-file clustering):논리적으로 연관되어 함께 검색될 가능성이 높은 두 개 이상의 화일에 속하는 레코드들을 디스크 상에서 물리적으로 가까운 곳에 모아 저장하는 것

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
- ✓ EMPLOYEE 릴레이션에 해당하는 파일에서 같은 부서에 속하는 사원 레코드이 함께 검색될 가능성이 높으므로 같은 블록에 저장

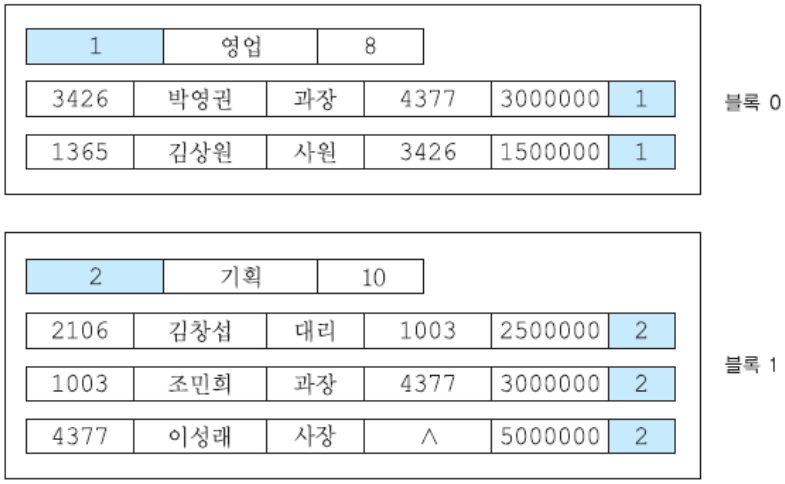


[그림 6.10] 화일 내의 클러스터링

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
- ✓ DEPARTMENT 파일에서 1번 부서번호에 해당하는 레코드와 EMPLOYEE 파일에서 1번 부서에 소속된 레코드를 동일블록에 저장
- ✓ 어떤 부서의 이름등과 함께 그 부서에 소속된 사원에 관한 정보를 함께 검색하는 경우가 많기 때문



[그림 6.11] 화일 간의 클러스터링

6.4 화일 조직

❑ 화일 조직

- ✓ 화일 내의 데이터를 보조 기억 장치에서 블록과 레코드로 배치하는 것.

❑ 화일 조직의 유형

- ✓ 히프 화일(heap file)
- ✓ 순차 화일(sequential file)
- ✓ 인덱스된 순차 화일(indexed sequential file)
- ✓ 직접 화일(hash file)

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
- ✓ 시간이 오래되면 삭제된 레코드 빈공간이 증가되어 화일크기가 증가, 검색시에도 빈공간을 검색하므로 시간 오래 소요.
- ✓ 좋은 성능을 유지하기 위해서 히프 화일을 주기적으로 재조직할 필요가 있음(빈공간 회수->자유공간)

6.4 화일 조직(계속)

❑ 히프 파일(비순서 화일)

- ✓ 릴레이션에 데이터를 한꺼번에 적재할 때(bulk loading), 릴레이션에 몇 개의 블록들만 있을 때, 모든 투플들이 검색 위주로 사용될 때 주로 사용
- ✓ EMPLOYEE 파일이 히프파일로 저장(레코들이 임의순서로 저장)

블록 0

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

블록 1

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

블록 2

4377	이성래	사장	∧	5000000	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 효율적
- ✓ (EMPLOYEE 릴레이션에서 모든 레코드들을 임의의 순서로 검색)

```
SELECT *
FROM EMPLOYEE;
```

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

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

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
  - ✓ EMLPOYEE 릴레이션에서 부서번호가 2번인 사원을 검색하는데 사원이 여러 명 있을 수 있으므로 모든 레코드들을 접근해야만 한다.
  - ✓

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

6.4 화일 조직(계속)

- 히프 화일의 성능(계속)
  - ✓ 몇 개의 레코드들을 검색하는 경우에도 비효율적
  - ✓ 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;
```

〈표 6.1〉 연산의 유형과 소요 시간

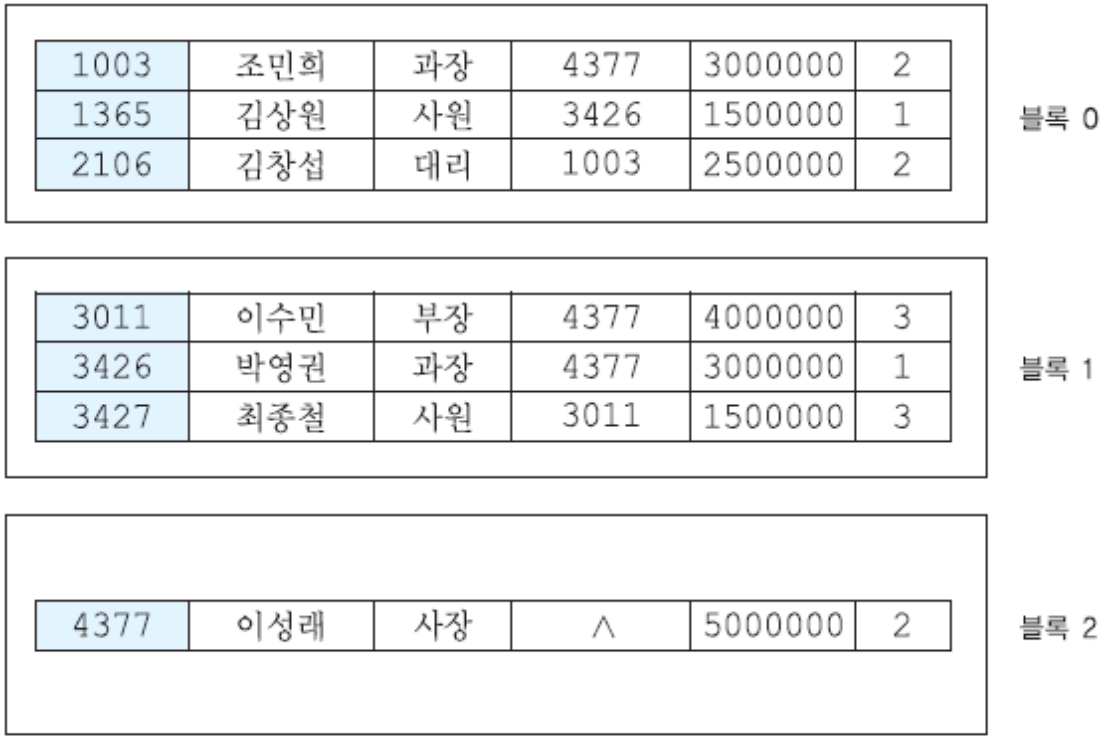
연산의 유형	시간
삽입	효율적
삭제	시간이 많이 소요
탐색	시간이 많이 소요
순서대로 검색	시간이 많이 소요
특정 레코드 검색	시간이 많이 소요

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(빈공간-삽입, 없으면-오버플로블록에 저장 또는 이전 이후 레코드들을 하나씩 이동 후 삽입)
- ✓ 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

- EMPNO를 키로 순서대로 저장되어 있는 순차화일

6.4 화일 조직(계속)

❑ 순차 화일의 성능

✓ EMPLOYEE 화일이 EMPNO의 순서대로 저장되어 있을 때

✓ SELECT문은 EMPNO 키로 이진 탐색을 이용하여 효율적인 탐색 가능,

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

✓ SELECT문의 WHERE절에 사용된 SALARY는 저장 순서와 무관하기 때문에 화일 전체를 탐색해야 함

```
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)

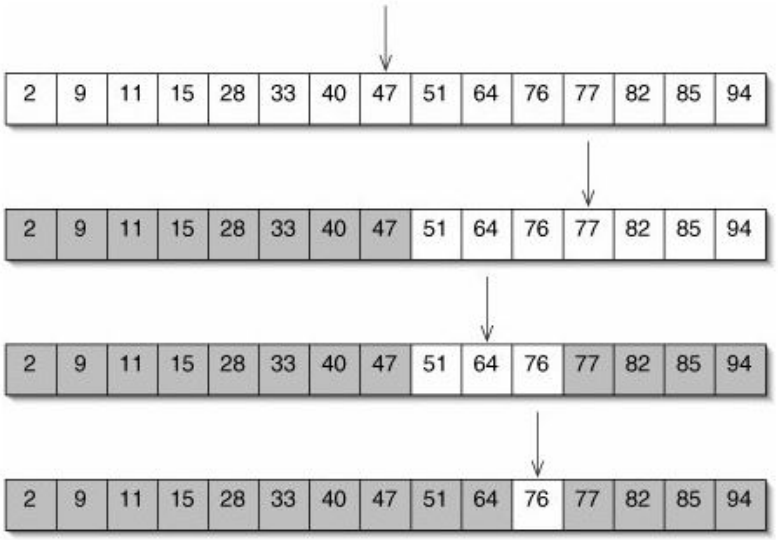
❑ 이진탐색 알고리즘(binary search algorithm)

✓ 정렬된 리스트에서 특정 값의 위치를 찾는 알고리즘

✓76찾기(처음 선택값과 목표값을 비교하여)

✓처음값 > 목표값 왼쪽, 처음값 < 목표값 오른쪽 검색

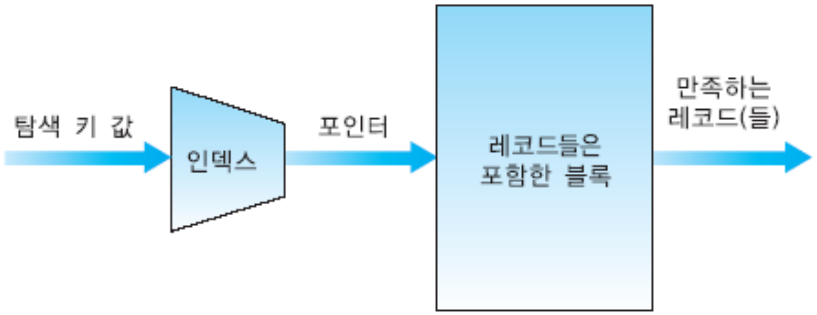
✓선형탐색보다 속도가 빠르다. 반드시 정렬되어 있어야 한다.



6.5 단일 단계 인덱스

❑ 단일 단계 인덱스

- ✓ 인덱스된 순차 화일은 인덱스를 통해서 임의의 레코드를 접근할 수 있는 파일 (인덱스 자체가 파일)
- ✓ 단일 단계 인덱스의 각 엔트리는
  <탐색 키, 레코드에 대한 포인터>
- ✓ 엔트리들은 탐색 키 값의 오름차순으로 정렬됨
- ✓ 인덱스는 DBMS가 화일내의 특정 레코드를 빠르게 찾을 수 있도록하는 데이터 구조로 인덱스를 통하여 질의하면 응답시간이 향상 된다.



[그림 6.14] 인덱스를 통한 레코드 검색

6.5 단일 단계 인덱스

❑ 단일 단계 인덱스

- ✓ 디스크 접근시간이 주기억 장치 접근 시간에 비해 매우 크고, 대부분의 데이터베이스 응용에서 디스크 접근을 많이 요구하므로, 인덱스를 통해 디스크 접근 횟수를 줄이면 데이터베이스 성능을 크게 향상 시킬 수 있다.

- ✓ 인덱스는 임의접근을 필요로하는 응용에 적합
- ✓ 인덱스는 데이터 파일과 별도의 파일에 저장
- ✓ 인덱스 파일의 크기는 데이터 파일의 **10~20%**정도**(**데이터 파일에 들어 있는 여러 애트리뷰트들 중에서 탐색키에 해당하는 일부 애트리뷰트만 인덱스에 포함**)**
- ✓ 하나의 인덱스 파일에 여러 개의 인덱스를 정의 할 수 있다.

6.5 단일 단계 인덱스(계속)

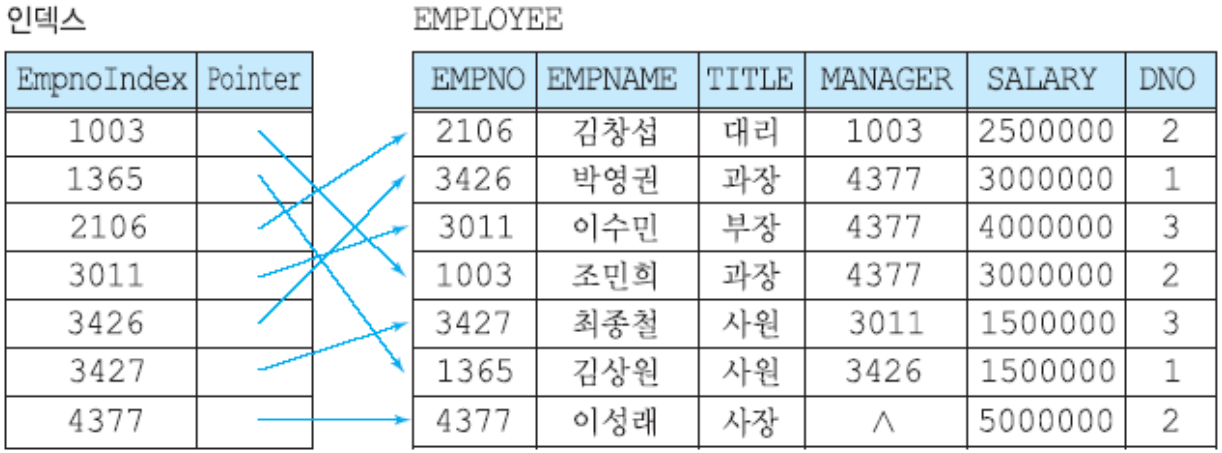
❑단일 단계 인덱스

✓EMPLOYEE 파일의 EMPNO에 정의된 인덱스

✓인덱스 엔트리 <EmpnoIndex, Pointer> EmpnoIndex값이 증가하는 순으로 엔트리 정렬

✓ EMPOLYEE 릴레이션의 한 레코드 길이는 36바이트

✓ 인덱스 파일의 레코드는 EmpnoIndex 4바이트, Pointer 4바이트 총 8바이트로 작다.



EmpnoIndex	Pointer
1003	
1365	
2106	
3011	
3426	
3427	
4377	

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 키라고 부름(엔트리=<탐색키, 레코드에 대한 포인터>
- ✓ 탐색 키의 값들은 후보 키처럼 각 투플마다 반드시 고유하지는 않음
  즉, 후보 키와 달리 두 개 이상의 투플들이 동일한 탐색 키값을 가질 수 있다.
- ✓ 기존의 키 개념과 다름.
- ✓ 키를 구성하는 애트리뷰트뿐만 아니라 어떤 애트리뷰트도 탐색 키로 사용될 수 있음
- ✓ 인덱스가 데이터 파일보다 크기가 작으므로 인덱스를 순차적으로 찾는 시간은 데이터 파일을 순차적으로 탐색하는 시간보다 적게 소요.
- ✓ 인덱스의 엔트리들은 탐색 키 값의 오름차순으로 저장되어 있으므로 이진 탐색을 이용할 수도 있음
- ✓ 인덱스를 사용하면 한 파일에서 특정 레코드를 찾기 위해서 모든 레코드를 탐색할 필요가 없으므로 매우 큰 파일의 경우 유용
- ✓ 인덱스 전체를 주기억 장치에 유지 할 수 있을 때, 어떤 레코드를 찾는 질의도 한번의 디스크 접근만으로 처리되어 성능에 많은 도움이 된다.

6.5 단일 단계 인덱스(계속)

❑ 기본 인덱스(primary index)

- ✓ 탐색 키가 데이터 화일의 기본 키인 인덱스를 기본 인덱스라고 부름
  레코드들은 기본 키값에 따라 클러스터링
- ✓ 기본 인덱스는 기본 키의 값에 따라 정렬된 데이터 화일에 대해 정의됨
- ✓ 기본 인덱스는 흔히 희소 인덱스로 유지할 수 있음

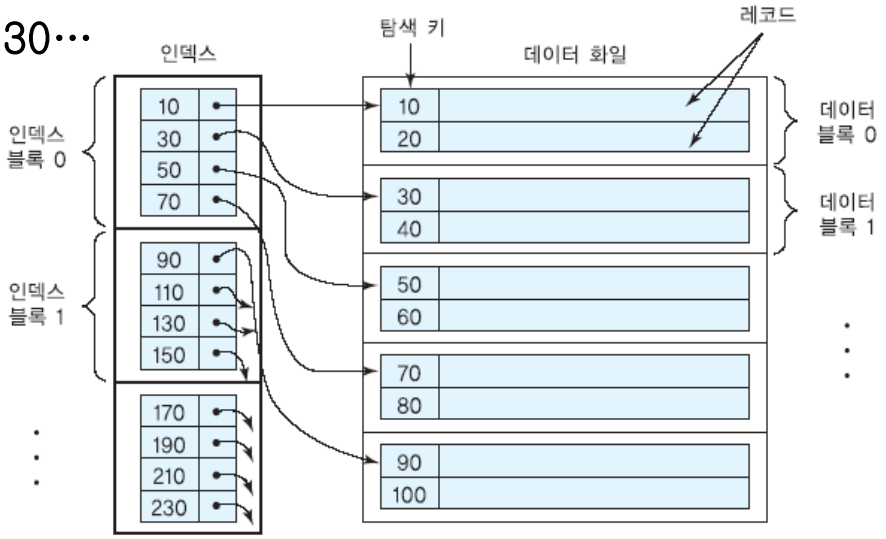
(희소 인덱스) : 데이터 파일을 구성하는 각 블록마다 하나의 탐색 키값이 인덱스 엔트리에 포함된다. 각 인덱스 엔트리는 블록 내의 첫번째 레코드의 키값(블록앵커:block anchor)을 갖는다.

- ✓ 각 릴레이션마다 최대한 한 개의 기본 인덱스를 가질 수 있음

6.5 단일 단계 인덱스(계속)

❑기본 인덱스(primary index)

- ✓데이터 파일의 레코드들은 저장하는 블록 마다 두 개의 레코드가 들어 있다.
- ✓인덱스 엔트리들은 저장하고 있는 블록 마다 네 개의 엔트리가 포함되어 있다.
- ✓데이터 블록 당 레코드 수 와 인덱스 블록 당 엔트리 수는 더 많은 차이가 난다.
- ✓첫 번째 데이터블록의 기본 키는 10,20인 레코드 중 기본 키가 10이 인덱스에 포함
- ✓두 번째 데이터블록의 기본 키는 30,40인 레코드 중 기본 키가 30이 인덱스에 포함
- ✓블록앵커 10, 30…



[그림 6.17] 데이터 화일에 대한 기본(또는 희소) 인덱스

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)

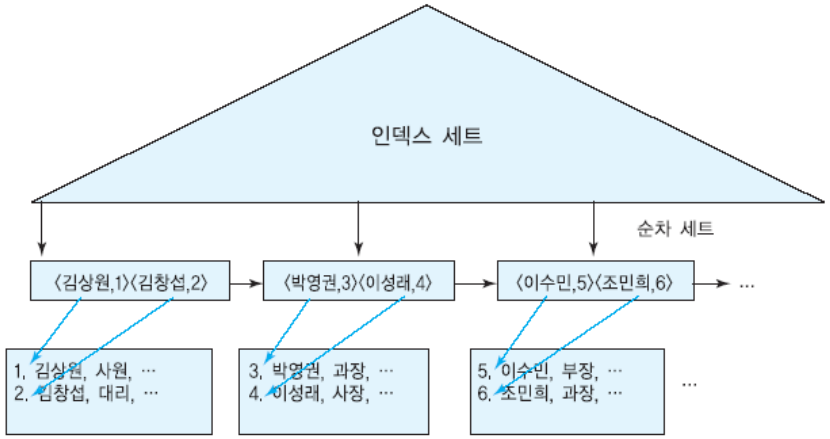
❑클러스터링 인덱스(clustering index)

✓EMPLOYEE 파일의 EMPNAME 애트리뷰트에 대한 클러스터링 인덱스

✓데이터 파일의 레코드들은 EMPNAME 애트리뷰트의 값이 증가하는 순으로 정렬

✓인덱스 엔트리들도 EMPNAME의 값이 증가하는 순으로 정렬

✓결론 : 인덱스 엔트리의 정렬 순서와 데이터 파일의 레코드 정렬순서가 일치하므로 EMPNAME에 대한 어떤 범위를 만족하는 레코드를 검색 시 범위에 속하는 첫 레코드를 찾은 후 인덱스 엔트리의 포인터를 따라가면서 주어진 범위에 속하는 레코드들을 인접한 데이터 블록에서 빠르게 찾을 수 있다.



[그림 6.18] 클러스터링 인덱스

6.5 단일 단계 인덱스(계속)

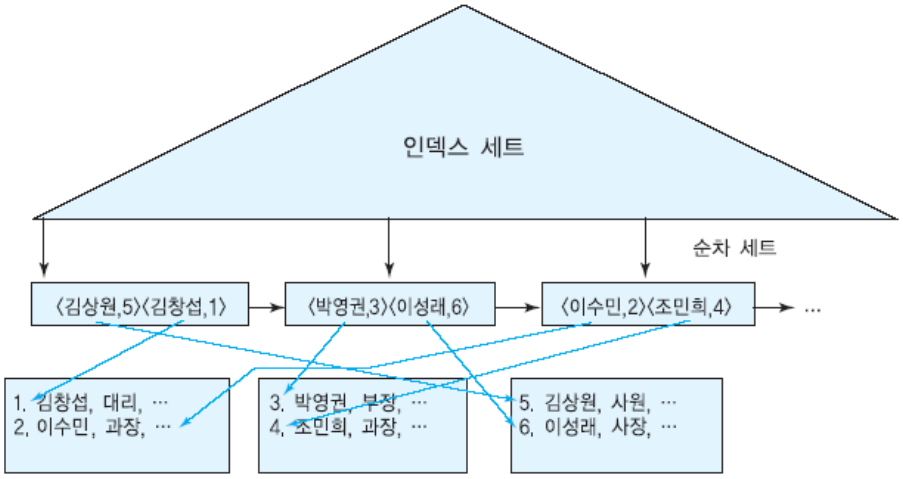
❑비클러스터링 인덱스(non -clustering index)

✓EMPLOYEE 파일의 EMPNAME 애트리뷰트에 대한 클러스터링 인덱스

✓데이터 파일의 레코드들은 EMPNAME 애트리뷰트의 값의 순서와 관계없이 저장

✓인덱스 엔트리들이 인접해 있어도 레코드들은 대부분 멀리 떨어져 있다.

✓결론 : 어떤 범위를 만족하는 레코드를 검색 시 범위에 속하는 첫 레코드를 인덱스 엔트리에서 찾은 후, 범위 내의 인덱스 엔트리를 차례대로 읽으면서 데이터 레코드를 검색할 때마다 디스크 블록을 접근해야하는 경우가 많다.



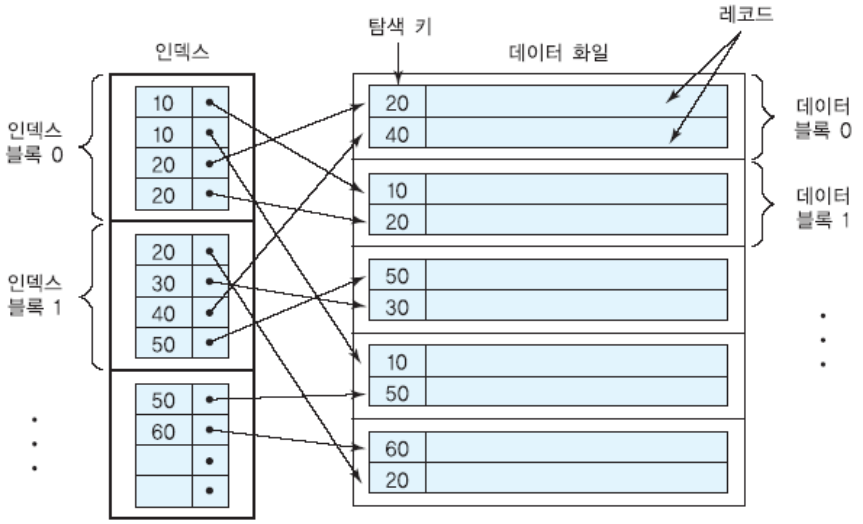
[그림 6.19] 비 클러스터링 인덱스

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
- ✓ 기본 인덱스를 사용한 순차접근 효율적, 보조 인덱스는 디스크 접근 시간을 증가시켜 비효율적
- ✓ 신용카드회사의 카드번호(기본 인덱스) 주민등록번호(보조 인덱스)

데이터 파일의 레코드들이 인덱스가 정의된 필드 값과 무관하게 저장되어 있다.

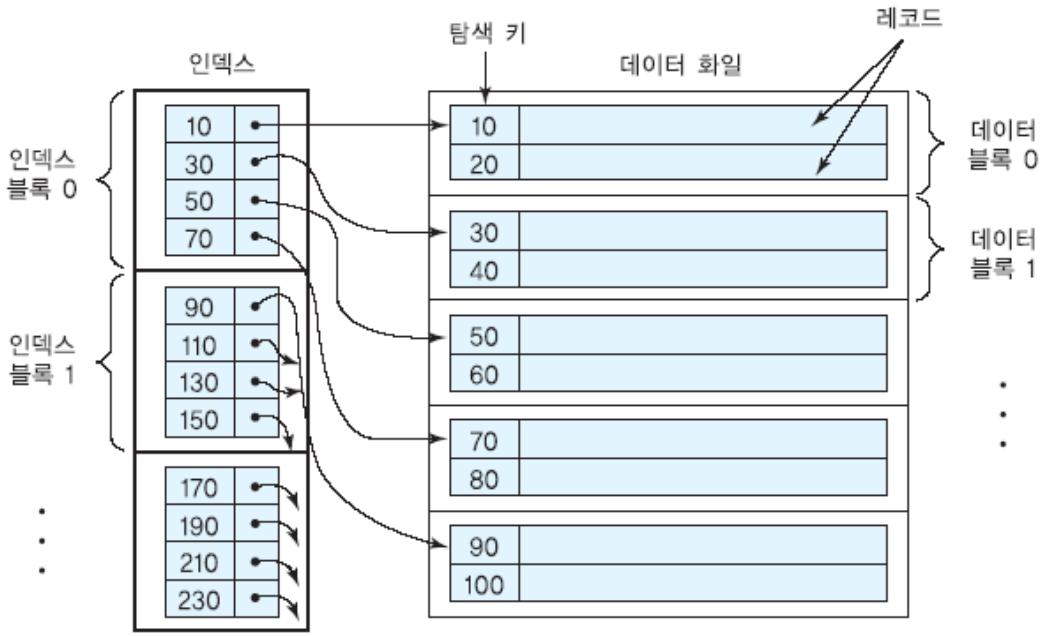


[그림 6.20] 보조(밀집) 인덱스

6.5 단일 단계 인덱스(계속)

❑ 희소 인덱스(sparse index)

- ✓ 일부 키 값에 대해서만 인덱스에 엔트리를 유지하는 인덱스
- ✓ 각 블록마다 한 개의 탐색 키 값이 인덱스 엔트리에 포함된다.
- ✓ 기본 인덱스와 동일

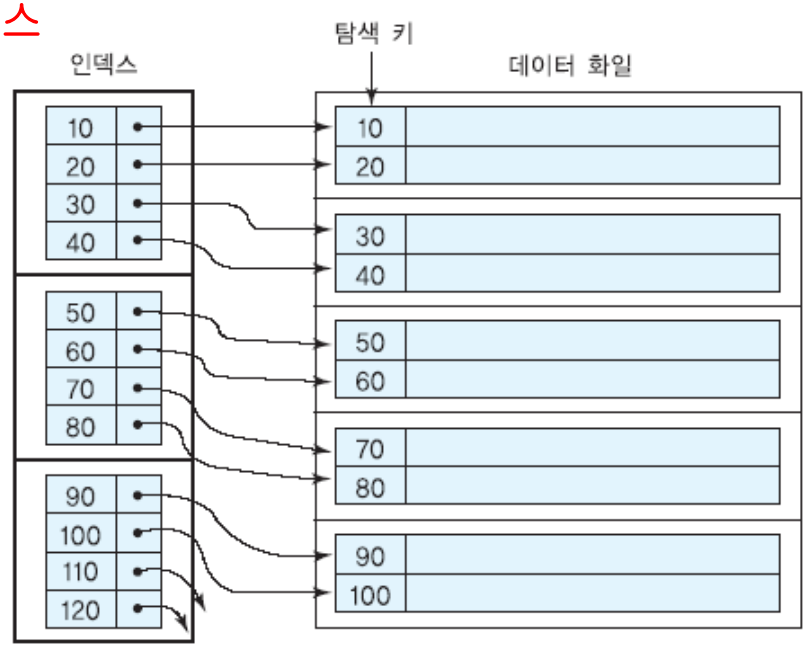


[그림 6.17] 데이터 화일에 대한 기본(또는 희소) 인덱스

6.5 단일 단계 인덱스(계속)

❑ 밀집 인덱스(dense index)

- ✓ 각 레코드의 키 값에 대하여 인덱스에 엔트리를 유지하는 인덱스
- ✓ 데이터 파일의 레코드 탐색 키 값이 인덱스 엔트리에 포함된다.
- ✓ 데이터 파일의 레코드 탐색 키 값이 증가하는 순으로 정렬되어 있으므로 클러스터링 인덱스



[그림 6.21] 데이터 화일에 대한 밀집 인덱스

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

✓ 밀집 인덱스 내의 엔트리 수= 희소 인덱스 내의 엔트리 수* 블록 당 평균 레코드 수 곱
–레코드 길이가 블록 크기에 가까울 수록 밀집 인덱스와 희소 인덱스의 크기가 비슷

✓ 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

✓ 희소 인덱스는 밀집 인덱스에 비해 모든 갱신과 대부분의 질의에 대해 더 효율적
✓ 만일 질의에서 인덱스가 정의된 애트리뷰트 만 검색(예:COUNT 질의)경우에는 데이터 파일에 접근할 필요없이 인덱스만 접근해서 질의 수행하므로 밀집 인덱스가 유리

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

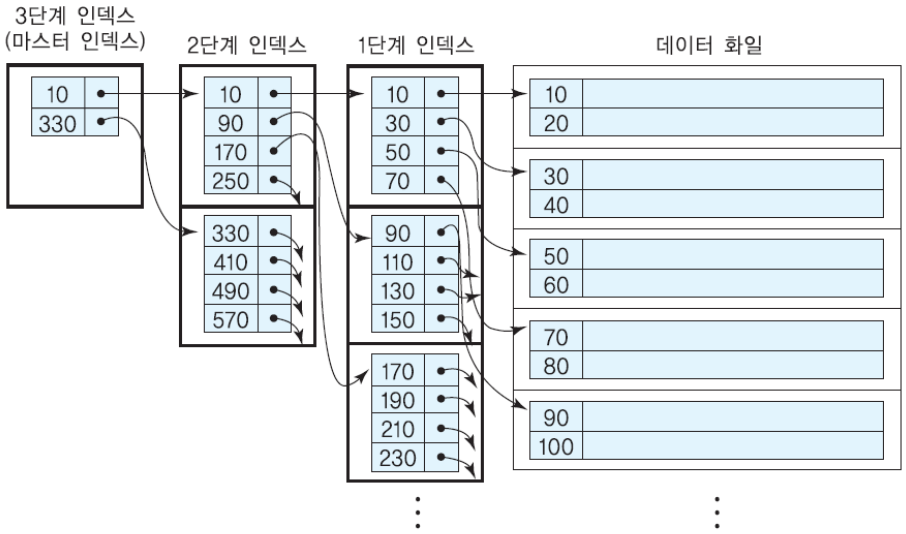
❑ 다단계 인덱스

- ✓ 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
- ✓ 다단계 인덱스의 각 단계는 하나의 순서 파일이다.
- ✓ 가장 상위 단계 인덱스를 마스터 인덱스(master index)라고 부름
- ✓ 마스터 인덱스는 한 블록으로 이루어지기 때문에 주기억 장치에 상주할 수 있음
- ✓ 대부분의 다단계 인덱스는 B^+-트리를 사용

6.6 다단계 인덱스(계속)

❑ 다단계 인덱스

- ✓ 3단계 인덱스
- ✓ 마스터 인덱스의 엔트리는 2단계 인덱스 블록을, 다시 1단계 인덱스 블록을 가르키고, 마지막으로 1단계 인덱스 엔트리들은 데이터 파일의 블록을 가르킨다.
- ✓ 다단계 인덱스는 B+ 트리(이진균형 트리)사용하며, B+ 트리는 4단계 이상 필요한 경우가 없고, 추가될 인덱스 엔트리를 위해 각 인덱스 블록에 예비공간을 둔다.



[그림 6.22] 다단계 인덱스

6.6 다단계 인덱스(계속)

❑ SQL의 인덱스 정의문

- ✓ SQL의 CREATE TABLE문에서 PRIMARY KEY절로 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 INDEX문을 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
  - ✓ 추천 : 대부분의 데이터베이스 전문가들은 복합 애트리뷰트에 인덱스를 정의 할 때, 3개 이하의 애트리뷰트를 사용(복합 애트리뷰트에 포함된 애트리뷰트의 개수가 늘어날수록 탐색조건이 복잡해지고, 인덱스 엔트리의 길이도 늘어나므로 탐색성능이 저하)
  - ✓ 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–두 애트리뷰트를 참조 질의 OK

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

  - ✓ 이 인덱스는 아래의 질의에도 활용될 수 있음–첫번째 애트리뷰트 참조 질의 OK

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

  - ✓ 이 인덱스는 아래의 질의에는 활용될 수 없음–인덱스가 정의된 두번째 이후 애트리뷰트만 참조하는 질의 NO
  - ✓

```
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. 먼저 찾는 속도우수
- ✓ 데이터 파일이 갱신되는 경우에는 그 파일에 정의된 모든 인덱스들에도 갱신 사항이 반영되어야 하므로 인덱스의 갱신은 데이터베이스의 성능을 저하 시킨다. 가능하면 릴레이션 당 인덱스의 개수를 3개 이내로 유지
- ✓ 릴레이션이 매우 크고, 질의에서 릴레이션의 투플들 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.

- ✓ 어느 애트리뷰트에 인덱스를 정의할 것인가? 어렵고, 경험이 필요하며 물리적 데이터베이스 설계자에게 가장 중요한 업무 중 하나

- ✓ 워크로드(work load:작업량) 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(자동으로 생성)
- ✓ 지침 2: 외래 키도 인덱스를 정의할 중요한 후보(외래키 인덱스 : 조인 연산 도움)
- ✓ 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: 투플이 많이 들어 있는 릴레이션에서 대부분의 질의가 검색하는 투플이 2% ~ 4% 미만인 경우에는 인덱스를 생성(한 릴레이션에 세 개 이하 인덱스)
- ✓ 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)

- 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;
```

  - ✓ SALARY가 인덱스로 정의되어 있으면 아래 질의에서 인덱스로 사용

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

6.7 인덱스 선정 지침과 데이터베이스 튜닝(계속)

❑ 언제 인덱스가 사용되지 않는가?(계속)

- ✓ DBMS가 제공하는 내장 함수가 집단 함수 대신에 사용됨
- ✓ EMPNAME 애트리뷰트에 인덱스가 정의되어 있어도 사용되지 않는다.
- ✓ SUBSTR( )은 주어진 문자열에서 일부를 반환하는 오라클 내장함수로 EMPNAME 애트리뷰트에서 첫번째 바이트부터 두 바이트가 '김'인 사원을 검색하라
- ✓ 이런 내장함수는 SQL의 표준 기능이 아니다.

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

- ✓ EMPNAME 애트리뷰트가 인덱스 정의되어 있으면 아래 질의에서 인덱스로 사용

```
SELECT      *
FROM        EMPLOYEE
WHERE       EMPNAME LIKE '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
  - ▪ 중복된 투플을 허용 할 수 있는 경우 또는 SELECT절에 키가 포함되어 있을 경우 DISTINCT절이 불필요
- ✓ GROUP BY절과 HAVING절의 사용을 최소화하라
- ✓ 임시 릴레이션의 사용을 피하라
- ✓ SELECT * 대신에 SELECT절에 애트리뷰트 이름들을 구체적으로 명시하라

❑ 결론

- ✓ 불필요한 애트리뷰트가 전송되지 않으므로, 네트워크 트래픽이 감소되고, SELECT문을 이해하기 쉽고, 릴레이션의 추가, 삭제 시 SELECT문 변경이 감소한다.
- ✓ 릴레이션에 수 많은 애트리뷰트들이 포함되어 있거나 긴 애트리뷰트(BLOB)가 포함되어 있을 때 SELECT절에서 필요한 애트리뷰트들만 구체적으로 명시하는 것이 네트워크 트래픽에 큰 영향을 미칠 수 있다.

데이터베이스

- 순서 : Ch6-1(인덱스 실습)
- 학기 : 2015학년도 2학기
- 학과 : 가천대학교 컴퓨터공학과 2학년
- 교수 : 박양재

데이터베이스

- 목차

6-1장. 인덱스 실습

❑ 인덱스의 개요

01.인덱스란?

SQL 명령문의 처리 속도를 향상 시키기 위해서 컬럼에 생성하는 오라클 객체이다.

(인덱스는 조회 성능을 향상 시키는 객체)

인덱스의 내부구조는 B-트리 형식으로 구성 되어 있다. 컬럼에 인덱스를 설정하면 B-트리도 생성 되어야 하기 때문에 인덱스를 생성하기 위한 시간도 필요하고, 인덱스를 위한 추가공간도 요구된다.

인덱스가 생성된 후 새로운 행을 추가하거나 삭제할 경우 인덱스로 사용된 컬럼값도 함께 변경되는 경우가 발생된다. 인덱스로 사용된 컬럼값이 변경될 때는 이를 위한 내부구조(B-트리)역시 함께 수정되어야 한다.

이 작업은 오라클 서버에 의해 자동적으로 일어나는데, 인덱스가 없는 경우보다 있는 경우에 DML작업이 훨씬 무거워지게 된다.

6-1장. 인덱스 실습

❑ 인덱스의 개요

01.인덱스란?

인덱스의 장.단점

장점 : 검색 속도가 빨라진다.

시스템에 부하를 줄여서 시스템의 전체 성능을 향상 시킨다.

단점 : 인덱스를 위한 추가 공간이 필요하다.

인덱스를 생성하는데 시간이 소요된다.

데이터 변경 작업(INSERT/UPDATE/DELETE)이 자주 일어날 때 오히려 성능이 저하된다.

6-1장. 인덱스 실습

❑ 인덱스의 개요

02.인덱스 정보 조회

오라클에서 인덱스 객체에 대한 정보를 알아보기 위해서는 USER_COLUMNS와 USER_IND_COLUMNS 데이터 딕셔너리 뷰를 살펴보아야 한다.

책의 색인란과 동일한 역할을 하는 쿼리를 빠르게 수행하기 위한 용도로 사용되는 인덱스는 기본 키나 유일한 키와 같은 제약조건을 지정하면 별도로 생성하지 않아도 자동으로 인덱스를 생성한다.(DEPT 테이블의 인덱스 키 : DEPTNO, EMP 테이블: EMPNO)

```
SQL> COLUMN TABLE_NAME FORMAT A15
SQL> COLUMN COLUMNE_NAME FORMAT A15
SQL>
SQL> SELECT INDEX_NAME, TABLE_NAME, COLUMN_NAME
  2  FROM USER_IND_COLUMNS
  3  WHERE TABLE_NAME IN('EMP', 'DEPT') ;

INDEX_NAME                     TABLE_NAME
------------------------------ ---------------
COLUMN_NAME
-----------------------------------------------
PK_DEPT                        DEPT
DEPTNO

PK_EMP                         EMP
EMPNO
```

6-1장. 인덱스 실습

❑ 인덱스의 개요

03.조회 속도 비교하기

인덱스로 지정된 컬럼이 그렇지 않은 컬럼보다 검색 속도가 빠른지 확인하기

방법1:기본 키나 유일 키로 지정하지 않은 컬럼인 사원명으로 검색 했을 때 검색 시간 비교

방법2:검색을 위해 WHERE절에 사용되는 컬럼인 사원명을 인덱스로 생성 후 사원명으로 검색 시간 비교

실습1. 사원 테이블 복사하기

```
SQL> CREATE TABLE EMP_IDX
  2  AS
  3  SELECT * FROM EMP ;

테이블이 생성되었습니다.
```

EMP와 EMP_IDX 테이블에 인덱스 설정확인

```
SQL> SELECT TABLE_NAME, INDEX_NAME, COLUMN_NAME
  2  FROM USER_IND_COLUMNS
  3  WHERE TABLE_NAME IN('EMP', 'EMP_IDX') ;

TABLE_NAME      INDEX_NAME
--------------- ------------------------------
COLUMN_NAME
--------------------------------------------------

EMP             PK_EMP
EMPNO
```

중요 :결과화면에 EMP 테이블에는 EMPNO 컬럼에 인덱스가 존재하지만 서브쿼리로 만든 EMP_IDX에는 인덱스가 존재하지 않는다. 서브쿼리로 복사한 테이블은 구조와 내용만 복사하고 제약조건(CONSTRAINT)은 복사되지 않는다.

6-1장. 인덱스 실습

❑ 인덱스의 개요

03.조회 속도 비교하기

인덱스로 지정된 컬럼이 그렇지 않은 컬럼보다 검색 속도가 빠른지 확인하기

① EMP_IDX 테이블에 같은 행을 반복적으로 여러 번 복사하기

```
SQL> INSERT INTO EMP_IDX SELECT * FROM EMP_IDX ;
28 개의 행이 만들어졌습니다.
SQL> INSERT INTO EMP_IDX SELECT * FROM EMP_IDX ;
56 개의 행이 만들어졌습니다.
SQL> INSERT INTO EMP_IDX SELECT * FROM EMP_IDX ;
112 개의 행이 만들어졌습니다.
   . . . . .
   . . . . .
917504 개의 행이 만들어졌습니다.
SQL> INSERT INTO EMP_IDX SELECT * FROM EMP_IDX ;
1835008 개의 행이 만들어졌습니다.
```

6-1장. 인덱스 실습

❑ 인덱스의 개요

03.조회 속도 비교하기

인덱스로 지정된 컬럼이 그렇지 않은 컬럼보다 검색 속도가 빠른지 확인하기

② 검색에 사용할 새로운 행을 EMP_IDX 테이블에 추가하기

```
SQL> INSERT INTO EMP_IDX (EMPNO, ENAME) VALUES(9999, 'GACHON') ;

1 개의 행이 만들어졌습니다.
SQL> SELECT * FROM EMP_IDX
  2  WHERE ENAME ='GACHON' ;

     EMPNO ENAME      JOB              MGR HIREDATE        SAL       COMM     DEPTNO
---------- ---------- --------- ---------- -------- ---------- ---------- ----------
      9999 GACHON
```

③ 명령 실행 시간 측정 준비

```
SQL> SET TIMING ON
SQL> SELECT DISTINCT EMPNO, ENAME
  2  FROM EMP_IDX
  3  WHERE ENAME='GACHON' ;

     EMPNO ENAME
---------- ----------
      9999 GACHON

경   과: 00:00:00.05
```

6-1장. 인덱스 실습

❑ 인덱스의 개요

04.인덱스 생성 및 제거하기

기본 키나 유일 키가 아닌 컬럼에 대해서 인덱스를 지정하려면 CREATE INDEX명령어를 사용하고, 삭제하기 위한 명령어는 DROP INDEX명령어를 사용한다.

[형식] CREATE INDEX index_name
ON table_name(column_name) ;

[형식] DROP INDEX index_name ;

실습 : 인덱스 설정하기-인덱스를 설정하지 않은 컬럼인 ENAME으로 조회하여 검색시간을 확인하였다. 이번에는 인덱스를 설벙 후 조회시간이 얼마나 단축되는지 확인하자.

① 테이블 EMP_IDX의 ENAME 컬럼으로 인덱스 생성

```
SQL> CREATE INDEX IDX_EMP_IDX_ENAME
  2  ON EMP_IDX(ENAME) ;

인덱스가 생성되었습니다.

경    과: 00:00:00.12
```

6-1장. 인덱스 실습

❑ 인덱스의 개요

04.인덱스 생성 및 제거하기

실습 : 인덱스 설정하기-인덱스를 설정하지 않은 컬럼인 ENAME으로 조회하여 검색시간을 확인하였다. 이번에는 인덱스를 설정 후 조회시간이 얼마나 단축되는지 확인하자.

② 인덱스 생성유무 확인하기

```
SQL> SELECT INDEX_NAME, TABLE_NAME
  2  FROM USER_INDEXES
  3  WHERE TABLE_NAME IN('EMP_IDX') ;

INDEX_NAME                     TABLE_NAME
------------------------------ --------------
IDX_EMP_IDX_ENAME              EMP_IDX

경    과: 00:00:00.05
```

③ 사원명(ENAME) GACHON인 행을 검색 수행 시간 비교(단축 확인 필요)

```
SQL> SELECT DISTINCT EMPNO,ENAME
  2  FROM EMP_IDX
  3  WHERE ENAME='GACHON' ;

     EMPNO ENAME
---------- ----------
      9999 GACHON

경    과: 00:00:00.03
```

6-1장. 인덱스 실습

❑ 인덱스의 개요

04.인덱스 생성 및 제거하기

실습 : 인덱스 제거하기

```
SQL> DROP INDEX IDX_EMP_IDX_ENAME ;

인덱스가 삭제되었습니다.

경    과: 00:00:00.23
SQL> SELECT INDEX_NAME, TABLE_NAME
  2  FROM USER_INDEXES
  3  WHERE TABLE_NAME IN ('EMP_IDX') ;

선택된 레코드가 없습니다.

경    과: 00:00:00.03
```

실습2:EMP_IDX 테이블에 직급 컬럼을 인덱스로 설정하되 인덱스 이름을 IDX_EMP_IDX_JOB으로 설정하고 확인 하세요.

```
SQL> CREATE INDEX IDX_EMP_IDX_JOB
  2  ON EMP_IDX(JOB) ;

인덱스가 생성되었습니다.
```

```
SQL> SELECT INDEX_NAME, TABLE_NAME
  2  FROM USER_INDEXES
  3  WHERE TABLE_NAME IN ('EMP_IDX') ;

INDEX_NAME                     TABLE_NAME
------------------------------ ------------
IDX_EMP_IDX_JOB                EMP_IDX
```

6-1장. 인덱스 실습

❑ 인덱스의 개요

05.인덱스를 사용해야 하는 경우 판단하기

무조건 이덱스를 사용한다고 해서 검색 속도가 빨라 지는것은 아니므로 계획성 없이 너무 많은 인덱스를 지정하면 오히려 성능을 저하시키는 요인이 된다.

인덱스를 사용해야 하는 경우	인덱스를 사용하지 않아야 하는 경우
테이블에 행의 수가 많을 때	테이블의 행의 수가 적을 때
WHERE문에 해당 컬럼이 많이 사용 될 때	WHERE문에 해당 컬럼이 많이 사용되지 않을 때
검색결과가 전체 데이터의 약2~4% 정도일때	검색결과가 전체 데이터의 약10~15% 정도일 때
JOIN에 자주 사용되는 컬럼이나 NULL을 포함하는 컬럼이 많은 경우	테이블에 DML작업이 많은 경우 즉,입력,수정, 삭제 등이 자주 일어 날 때

6-1장. 인덱스 실습

❑ 인덱스의 개요

05.인덱스를 사용해야 하는 경우 판단하기

예:다음과 같은 조건에서 사원 테이블의 부서번호를 인덱스를 거는 것이 좋을까?

[조건]

테이블의 전체 행의 수가 10000건입니다.

위의 쿼리문이 전체 쿼리문 중에서 95%를 사용합니다.

쿼리문의 결과로 구해지는 행은 10건 정도입니다.

① 인덱스 전 검색

```
SQL> SELECT DISTINCT DEPTNO
  2  FROM EMP_IDX
  3  WHERE DEPTNO=10 ;

    DEPTNO
----------
        10

경    과: 00:00:00.03
```

② 인덱스 생성

```
SQL> CREATE INDEX IDX_EMP_IDX_DEPTNO
  2  ON EMP_IDX(DEPTNO) ;

인덱스가 생성되었습니다.
```

③ 인덱스 후 검색

```
SQL> SELECT DISTINCT DEPTNO
  2  FROM EMP_IDX
  3  WHERE DEPTNO=10 ;

    DEPTNO
----------
        10

경    과: 00:00:00.01
```