6. 데이타베이스의 내부적 운영

6.1 데이타베이스의 저장

- 데이타베이스의 내부적 운영
  - 데이타를 저장하는 방법과 접근에 관련된 작업
  - 디스크(DASD) 사용
    - 디스크 접근(디스크 I/O) 횟수를 최소화

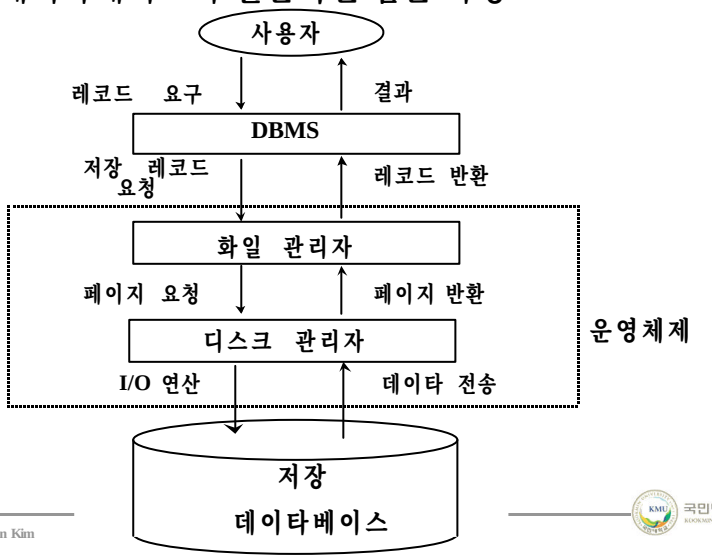
- 디스크 접근시간(access time)
  - 헤드가 원하는 트랙에 있는 레코드를 찾아 전송하는데 걸리는 시간
  - 탐구 시간(seek time)
  - 회전지연 시간(rotational delay)
  - 데이타 전송 시간(transfer time)
  - 주기억장치 접근시간에 비해 느림
    - I/O 횟수의 최소화가 가장 중요한 성능 개선 방법
    - 디스크에의 배치, 저장이 중요한 문제

□ 저장구조(storage structure)

- 디스크에 데이타가 배치, 저장되는 형식
- 다양한 저장구조 지원
  - ◆ DB의 부분별로 적절한 저장
  - ◆ 성능요건 변경 시 저장 구조 변경
- 데이타베이스의 물리적 설계
  - ◆ DB의 사용 방법, 응용, 응용 실행 빈도수에따라 적절한 저장방식을 선정하는 과정

6.2 데이타베이스의 접근

□ 데이타베이스의 일반적인 접근 과정



▶ 디스크 관리자

- 기본 I/O 서비스 (basic I/O service)
  - 모든 물리적 I/O 연산에 대한 책임
  - 운영체제의 한 구성요소
- 물리적 디스크 주소
- 화일 관리자 지원
  - 디스크를 일정 크기의 페이지로 구성된 페이지 세트들의 논리적 집단으로 취급하도록 지원
  - 데이타 페이지 세트와 하나의 자유공간 페이지 세트
  - 페이지 세트 : 유일한 페이지 세트 ID
  - 페이지 : 해당 디스크 내에서 유일한 페이지번호
- 디스크 관리
  - 페이지 번호 (사상) 물리적 디스크 주소
    → 화일 관리자를 장비에서 독립
  - 화일 관리자의 요청에 따라 페이지 세트에 대한 페이지의 할당과 회수

▶ 디스크 관리자(2)

- 디스크 관리자의 페이지 관리 연산
  - 화일 관리자가 명령할 수 있는 연산
    - ✶ 페이지 세트 S 로부터 페이지 P의 검색.
    - ✸ 페이지 세트 S 내에서 페이지 P 의 교체.
    - ✹ 페이지 세트 S 에 새로운 페이지 P 의 첨가 (자유공간 페이지 세트의 빈 페이지 할당)
    - ❋ 페이지 세트 S 서 페이지 P 의 제거 (자유공간 페이지 세트에 반납)
- Notes
  - 화일관리자가 필요로 하는 페이지 I/O 연산 : ✶ ✸
  - 페이지 세트들을 증감 시키는 연산 : ✹ ❋

▶ 화일 관리자

- DBMS가 디스크를 저장 화일들의 집합으로 취급할 수 있도록 지원
- 저장화일(stored file)
  - 한 타입의 저장레코드 어커런스들의 집합
  - 한 페이지 셑은 하나 이상의 저장화일을 포함
  - 화일이름 또는 화일 ID로 식별
- 저장 레코드는 레코드번호 또는 레코드 ID(RID: Record Identifier)로 식별
  - 전체 디스크 내에서 유일
  - (페이지 번호, 페이지 오프셋)
- OS의 한 구성요소 또는 DBMS와 함께 패키지화

▶ 화일 관리자(2)

- 화일 관리자의 화일 관리 연산
  - DBMS가 화일관리자에 명령할 수 있는 연산
    - ✶ 저장화일 f에서 저장레코드 r의 검색
    - ✷ 저장화일 f 에 있는 저장레코드 r 의 대체
    - ✹ 저장화일 f 에 새로운 레코드를 첨가하고
    - ✹ 저장화일 f에 새로운 레코드를 첨가하고 새로운 레코드 ID, r 을 부여
    - ❋ 저장화일 f에서 저장레코드 r 의 제거
    - ⌖ 저장화일 f 의 생성
    - ⌖ 저장화일 f 의 제거

6.3 페이지 세트와 화일

- 디스크 관리자
  - 화일관리자가 물리적 디스크 I/O가 아닌 논리적인 페이지 I/O 로 관리할 수 있게끔 지원
  - 페이지 관리(page management)
- 예
  - 저장화일들은 28개의 페이지로 구성된 페이지 세트에 저장
  - 각 레코드들은 하나의 페이지를 차지

▶ 대학 데이타베이스

	학번	이름	학년	학과
S1:	100	나 연묵	4	컴퓨터
S2:	200	이 찬영	3	전기
S3:	300	정 기태	1	컴퓨터
S4:	400	송 병호	4	컴퓨터
S5:	500	박 종화	2	산공

학생

	과목번호	과목이름	학점	담당교수
C1:	C123	프로그래밍	3	김 성기
C2:	C312	자료 구조	3	황 수찬
C3:	C324	화일 처리	3	이 규철
C4:	C413	데이타베이스	3	이 석호
C5:	E412	반도체	3	홍 봉희

과목

	학번	과목번호	성적
E1:	100	C413	A
E2:	100	E412	A
E3:	200	C123	B
E4:	300	C312	A
E5:	300	C324	C
E6:	300	C413	A

	학번	과목번호	성적
E7:	400	C312	A
E8:	400	C312	A
E9:	400	C413	B
E10:	400	C412	C
E11:	500	C312	B

등록

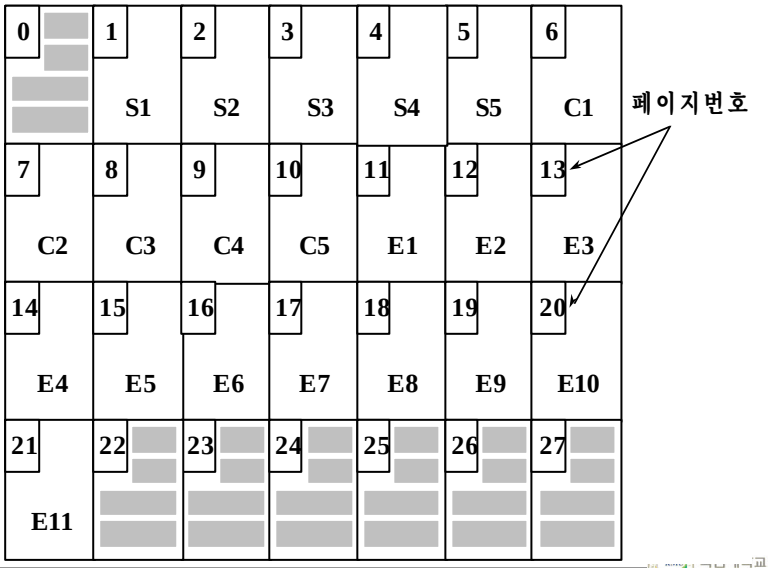
※ 연산

✶ 처음(빈 디스크) :
- 하나의 자유 공간 페이지 세트만 존재(1 ~ 27)
- 페이지 0 제외 : 디스크 디렉토리

✷ 화일관리자 : 학생 화일에 있는 5개의레코드 적재
- 디스크관리자 : 자유공간 페이지 세트의 페이지 1에서 5까지를 "학생 페이지 세트" 라고 이름을 붙이고 할당

✹ 과목과 등록 화일에 대한 페이지 세트를 할당
- 4개의 페이지 세트가 만들어짐
- "학생"(1~5), "과목"(6~10), "등록"(11~21),"자유공간" 페이지 세트 (페이지 22~27)

▶ 대학 데이타베이스의 초기 적재 후의 디스크 배치도



※ 연산

- ❋ 화일관리자 : 새로운 학생 S6 (학번 600)을 삽입
  - 디스크 관리자 : 첫번째 자유 페이지 (페이지 22)를 자유공간 페이지 세트에서 찾아서 학생 페이지 세트에 첨가
- ⌗ 화일 관리자 : S2 (학번 200)를 삭제
  - 디스크 관리자 : 이 레코드가 저장되어 있던 페이지 (페이지 2)를 자유공간 페이지 세트로 반납
- ⌖ 화일 관리자 : 새로운 과목 C6 (E 515)를 삽입
  - 디스크 관리자 : 자유공간 페이지 세트에서 첫번째 자유페이지 (페이지 2)를 찾아서 과목 페이지 세트에 첨가
- ✧ 화일 관리자 : S4를 삭제
  - 디스크 관리자 : S4가 저장되어 있던 페이지 (페이지 4)를 자유공간 페이지 세트에 반납

▶ 삽입, 삭제 연산 실행된 뒤의 디스크 배치도

I : S6
D : S2
I : C6
D : S4

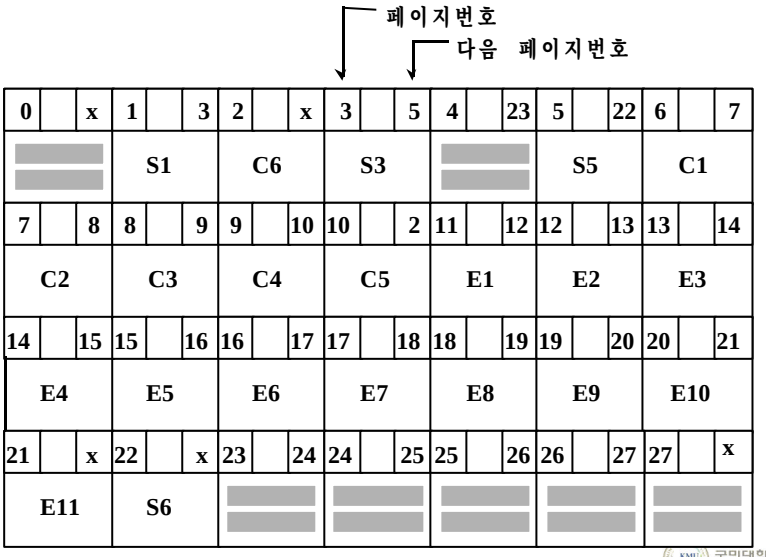
0	1 S1	2 C6	3 S3	4	5 S5	6 C1
7 C2	8 C3	9 C4	10 C5	11 E1	12 E2	13 E3
14 E4	15 E5	16 E6	17 E7	18 E8	19 E9	20 E10
21 E11	22 S6	23	24	25	26	27

- 삽입, 삭제 연산 실행 후에는 페이지들의 물리적 인접성이 없어짐

▶ 포인터 표현 방법

- 한 페이지 세트에서 페이지의 논리적 순서가 물리적 인접으로 표현되지 않음
  - 페이지
    : 페이지 헤드 - 제어정보 저장
  - 포인터
    : 논리적 순서에 따른 다음 페이지의 물리적 주소
  - 다음 페이지 포인터는 디스크 관리자가 관리
    (화일 관리자는 무관)

- 페이지 헤드에 "다음 페이지" 포인터가 포함되어 있는 경우의 디스크 배치도



▶ 포인터 표현 방법(2)

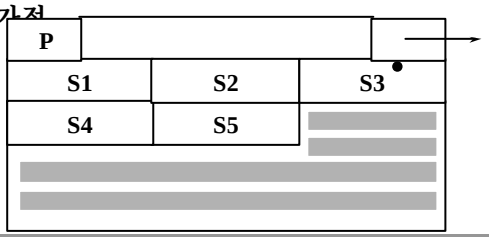
- 디스크 디렉토리(페이지 세트 디렉토리)
  - 실린더 0, 트랙 0에 위치
  - 디스크에 있는 모든 페이지 세트의 리스트와 각 페이지 세트의 첫번째 페이지에 대한 포인터 저장
- 디스크 디렉토리 (페이지 0)

페이지 세트	주 소
자유공간	4
학 생	1
과 목	6
등 록	11

▶ 화일 관리자

- 저장 레코드 관리 (stored record management)
  - DBMS가 페이지 I/O 에 대한 세부적인 사항에 대해 알 필요 없이 저장화일과 저장 레코드만으로 동작하게 함
- 예
  - 하나의 페이지에 여러 개의 레코드 저장
  - 학생 레코드에 대한 논리적 순서는 학번 순

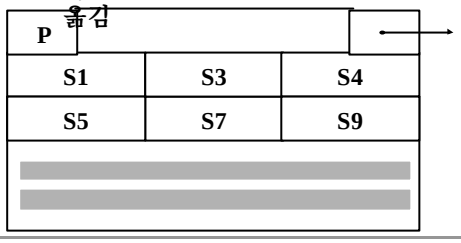
★ 페이지 p에 5개의 학생레코드(S1~ S5)가 삽입되어 있다고 가정



5개의 학생 레코드를 처음 적재한 페이지 P의 배치도

▶ 화일 관리자(2)

- DBMS : 학생 레코드 S9(학번 900)의 삽입 요청
  - 페이지 p의 학생레코드 S5 바로 다음에 저장
- DBMS : 레코드 S2의 삭제 요청
  - 페이지 p에 있는 학생 레코드 S2를 삭제하고 뒤에 있는 레코드들을 모두 앞으로 당김
- DBMS : 레코드 S7(학번 700)의 삽입 요청
  - 학생레코드 S5 다음에 들어가야 되므로 학생 레코드 S9를 뒤로 옮김

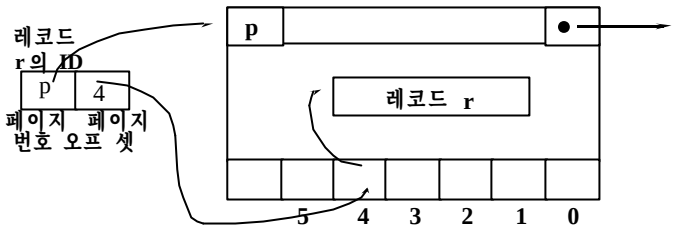


S2가 삭제되고
S9와 S7이 삽입된 후
의 페이지 P 의 배치도

※ Note

- 한 페이지 내에서 저장레코드의 논리적 순서는 그 페이지 내에서의 물리적 순서로 표현 가능
  - 레코드들이 페이지 내에서 이동
  - 레코드들을 모두 페이지 윗 쪽으로 저장
    - 아래쪽은 계속적으로 자유공간으로 유지

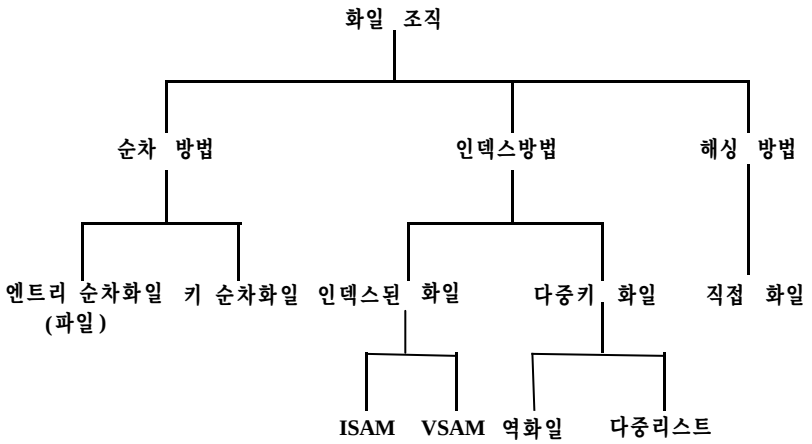
▶ RID의 구현



- RID = (페이지 번호 p, 오프 셋)
- 페이지 오프 셋 = 페이지 내에서의 레코드 위치(byte)
- 레코드가 한 페이지 내에서 이동할 때마다 RID의 변경 없이 페이지 오프 셋의 내용(포인터)만 변경
- 최악의 경우 두 번째 접근으로 원하는 레코드 검색가능
  - 두 번 접근 : 해당 페이지가 오버플로우가 되어 다른 페이지로 저장된 경우

6.4 화일의 조직 방법

- 화일조직 : 레코드 저장과 접근 방법 결정



▶ 파일 조직의 기본 방법

- 순차 방법
- 인덱스 조직
  - B-트리
  - B+-트리
- 해싱
  - 버켓 해싱
  - 확장 해싱

▶ 순차 방법

- 레코드들의 논리적 순서가 저장 순서와 동일
  - 파일(pile) : 엔트리 순차(entry-sequence) 화일
  - 순차 화일 : 키 순차(key-sequence) 화일
- 레코드 접근 - 물리적 순서
- 화일 복사, 순차적 일괄 처리(batch processing) 응용

S4	S1	S2
S5	S3	

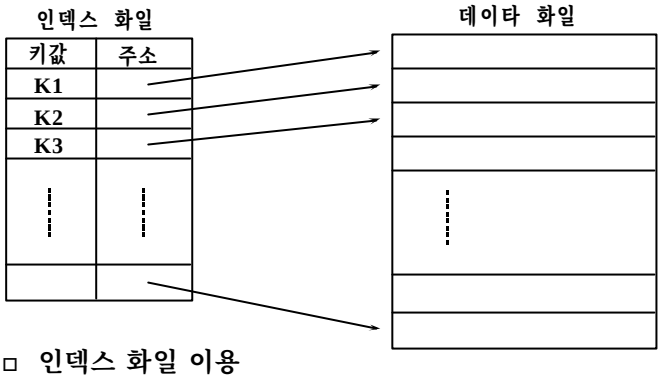
파일(엔트리순차) 화일

S1(100)	S2(200)	S3(300)
S4(400)	S5(400)	

키(학번)순차 화일

▶ 인덱스 방법

- 인덱스를 통해 데이타 레코드를 접근



- 인덱스 화일 이용
  - 인덱스된 순차 화일 : 하나의 인덱스
  - 다중 키 화일 : 다수 인덱스 동시 사용

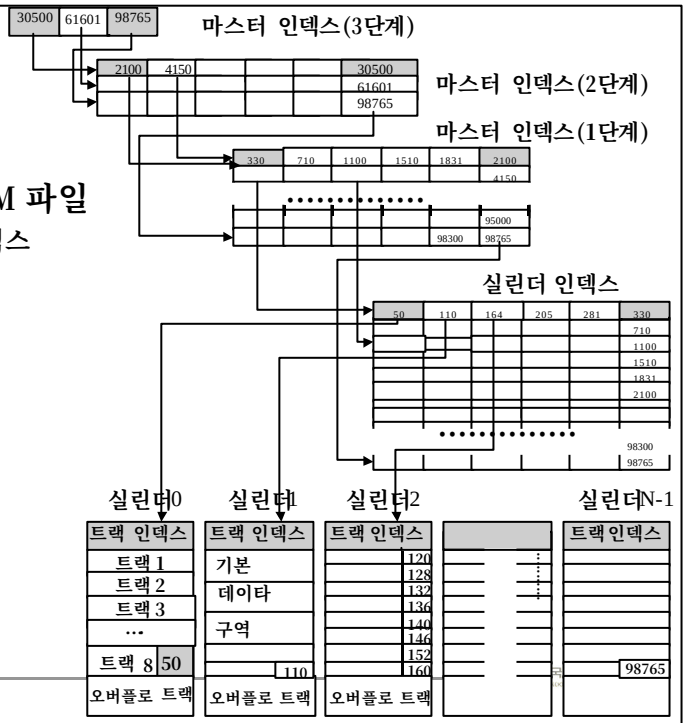
▶ 인덱스된 순차화일 (indexed sequential file)

- 키 값에 따라 정렬된 레코드를 순차적으로 접근
- 주어진 키값(인덱스)을 가지고 직접 접근
- 인덱스 구성 방법
  ① 정적 인덱스 방법
  ② 동적 인덱스 방법
  삭제 시 레코드의 순서 유지 및 인덱스 갱신 방법의 차이

(1) 정적 인덱스 방법

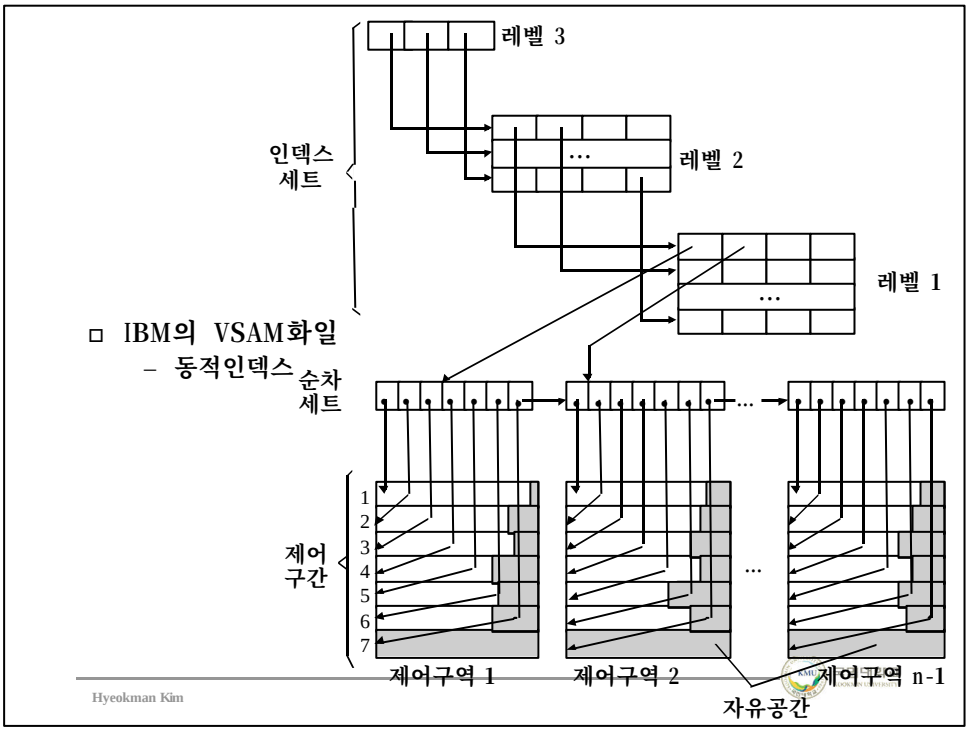
- 삽입, 삭제시
  - 인덱스 내용 변경
  - 인덱스의 구조는 변경되지 않음
- 오버플로우 구역(overflow area)
- 인덱스는 물리적 특성(실린더, 트랙)에 맞게 설계
- IBM의 ISAM 화일

- IBM의 ISAM 파일
  정적 인덱스



(2) 동적 인덱스 방법

- 인덱스나 데이타 화일을 블록으로 구성
- 각 블록에는 빈 공간을 예비
- 블록의 동적 분열(split)및 합병(merge)
- 인덱스는 트리 구조의 형태로 하드웨어와 독립적으로 구성
- IBM의 VSAM 화일
  - 제어구간(블록), 제어구역

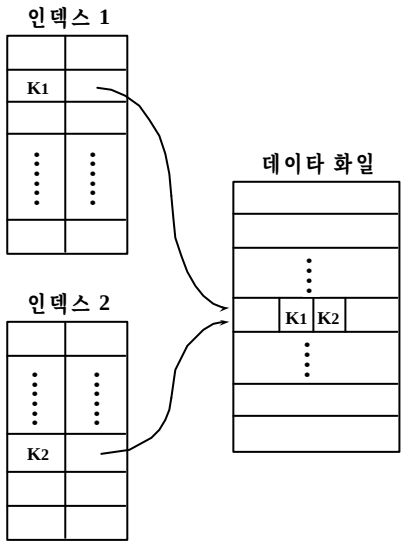


- IBM의 VSAM화일
  - 동적인덱스

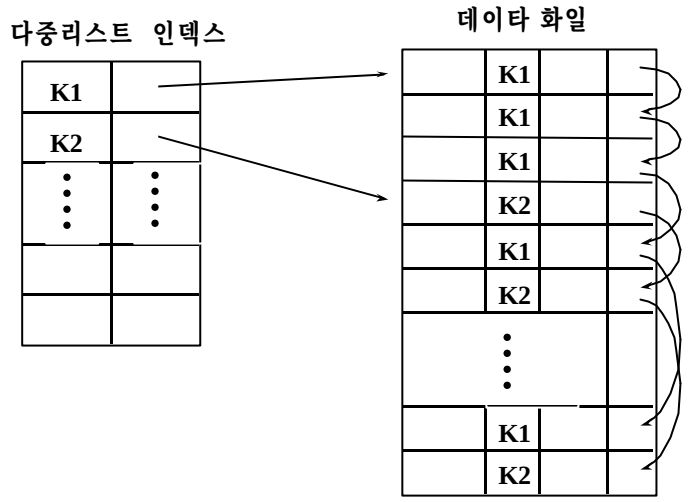
▶ 다중 키 화일(multikey file)

- 하나의 데이타 화일에 여러 개의 상이한 접근 방법 지원
- 데이타 레코드에 대한 다중 접근 경로
  - 역화일(inverted file)
    - 각 응용에 적합한 인덱스를 별도 구현
  - 다중 리스트 화일(multilist file)
    - 레코드들 사이의 다중 리스트 구축

(1) 역화일



(2) 다중리스트 화일



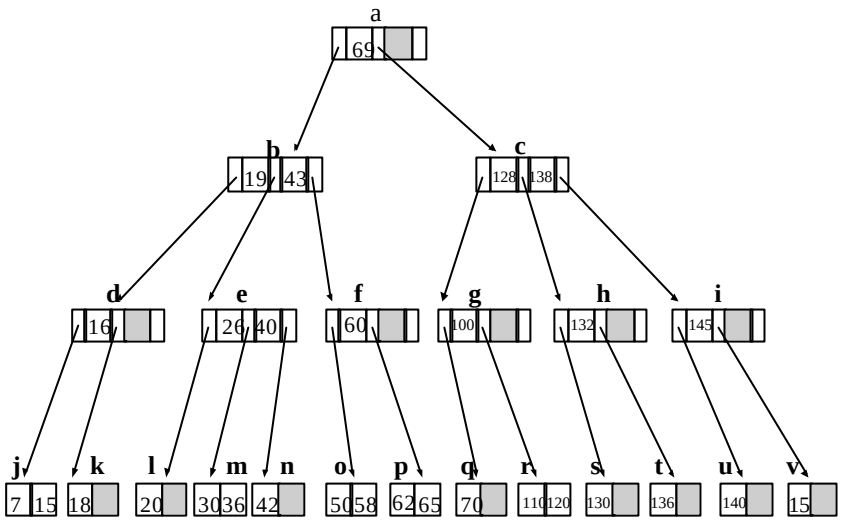
▶ 인덱스 조직

□ B-트리

- 균형 m-원 탐색 트리
- 차수 m인 B-트리의 특성
  - ◆ 루트와 리프를 제외한 노드의 서브트리 수 ≥ 2
    - $\lceil m/2 \rceil \le$ 개수 $\le m$
  - ◆ 모든 리프는 같은 레벨
  - ◆ 키값의 수
    - 리프 : $\lceil m/2 \rceil - 1$ ~ $(m-1)$
    - 리프가 아닌 노드 : 서브트리수 - 1
  - ◆ 한 노드 내의 키값 : 오름차순
- 노드 구조
  - ◆ $K_i \rightarrow (K_i, A_i)$: 데이타 화일의 주소($A_i$)

m	P_1	K_1	…	P_{m-1}	K_{m-1}	P_m

※ 3차 B-트리



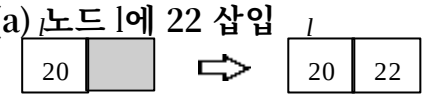
▶ 연산

- 직접 탐색 : 키 값에 의존한 분기
  - 순차 탐색 : 중위 순회
  - 삽입, 삭제 : 트리의 균형 유지
  - 분할 높이 증가
  - 합병 높이 감소

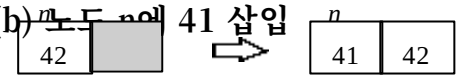
- 삽입
  - 리프노드
    - ◆ 빈 공간이 있는 경우: 순 삽입
    - ◆ 오버플로
      - ✶ 두 노드로 분열(split)
      - ✷ ⌈m/2⌉ 째의 키 값 → 부모노드
      - ✹ 나머지는 반씩 나눔 (왼쪽, 오른쪽 서브트리)

※ 삽입 예

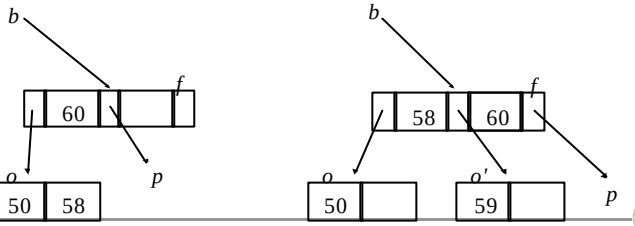
(a) 노드 l에 22 삽입



(b) 노드 n에 41 삽입



(c) 노드 o에 59 삽입



Hyeokman Kim

국민대학교 KOOKMIN UNIVERSITY

▶ 삭제

- 리프노드
- 삭제키가 리프가 아닌 노드에 존재
  - 후행키 값과 자리교환(후행키 - 항상 리프에)
  - 리프노드에서 삭제

- 언더플로 : 키수 < $m/2 - 1$
  - 재분배 (redistribution)
    - 최소키 수 이상을 포함한 형제노드에서 이동
      (형제노드의 키 → 부모노드 → 언더플로노드)
  - 합병 (merge)
    - 재분배 불가능시 이용
      (형제노드 + 부모노드의 키 + 언더플로노드)

Hyeokman Kim

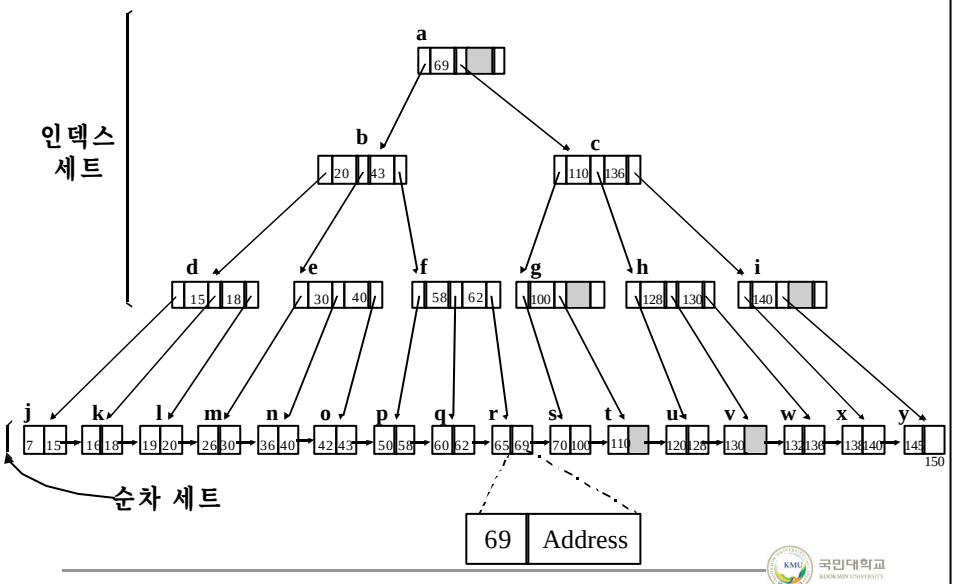
국민대학교 KOOKMIN UNIVERSITY

▶ B^+-트리

- 인덱스 세트 (index set)
  - 내부 노드
  - 리프에 있는 키들에 대한 경로 제공
  - 직접처리 지원

- 순차 세트 (sequence set)
  - 리프 노드
  - 모든 키 값들을 포함
  - 순차 세트는 순차적으로 연결
    → 순차처리 지원
  - 내부 노드와 다른 구조

❖ 차수가 3인 B^+-트리



▶ B^+-트리(2)

□ 특성

- 루트의 서브트리 : 0, 2, m/2 ~ m
- 노드의 서브트리 (루트,리프제외) : m/2 ~ m
- 모든 리프는 동일 레벨
- 리프가 아닌 노드의 키 값 수 : 서브트리수 - 1
- 리프노드 : 데이타 화일의 순차세트
  (리스트로 연결)

▶ B^+-트리(3)

□ 연산

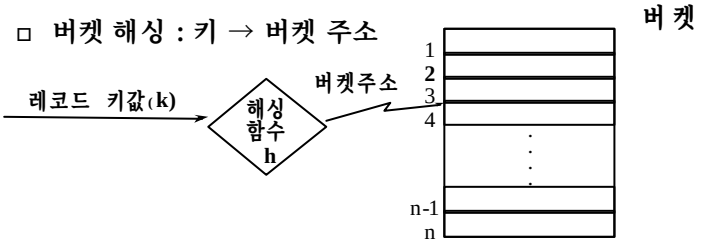
- 탐색
  - B^+-트리의 인덱스 셑 = m-원 탐색 트리
  - 리프에서 검색
- 삽입
  - B-트리와 유사
  - 오버플로우 (분열) → 부모 노드, 분열노드 모두에 키 값 존재
- 삭제
  - 리프에서만 삭제 (재분배, 합병 필요 없는 경우)
  - 재분배: 인덱스 키 값 변화, 트리구조 유지
  - 합병 : 인덱스의 키 값도 삭제

▶ 해싱 방법

- 다른 레코드 참조 없이 목표 레코드 직접 접근
  - 직접 화일(direct file)
- 키값과 레코드 주소 사이의 관계 설정
- 해싱 함수(hashing function)
  - 키 값으로부터 주소를 계산
  - 사상 함수(mapping function) : 키 → 주소
  - 삽입, 검색에도 이용

(1) 버켓 해싱

- 버켓(bucket) : 하나의 주소를 가지면서 하나 이상의 레코드 를 저장할 수 있는 화일의 한 구역
  - 버켓 크기 : 저장장치의 물리적 특성과 한번 접근으로 채취 가능한 레코드수 고려
- 버켓 해싱 : 키 → 버켓 주소



- 충돌(collision) : 상이한 레코드들이 같은 주소(버켓)로 변환
  - 버켓 만원 - 오버플로우 버켓
  - 한번의 I/O를 추가

(2) 확장성 해싱

- 충돌 문제에 대처하기 위해 제안된 기법
- 특정 레코드 검색 - 1~2번의 디스크 접근
- 기본키 사용

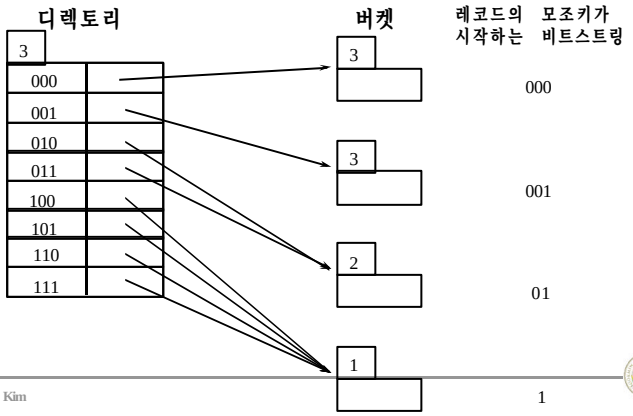
- 2단계 구조 : 디렉토리와 버켓

- 디렉토리
  - 정수값 d를 포함하는 헤더와 버켓들을 지시하는 2^d 개의 포인터로 구성
    - d = 디렉토리 깊이(depth)
  - 디스크에 저장

- 모조키(pseudokey)
  - 확장성 해싱 함수:
    키 값 일정 길이의 비트 스트링(모조키)
  - 모조키의 처음 d 비트를 디렉토리 접근에 사용

(2) 확장성 해싱

- 버켓
  - 정수 값 d'(d)가 저장된 헤더 존재
    - d' = 버켓 깊이
      = 버켓에 저장된 레코드들의 모조키들이 처음부터 d'비트까지 모두 동일



※ 확장성 해싱의 연산 예

□ 검색

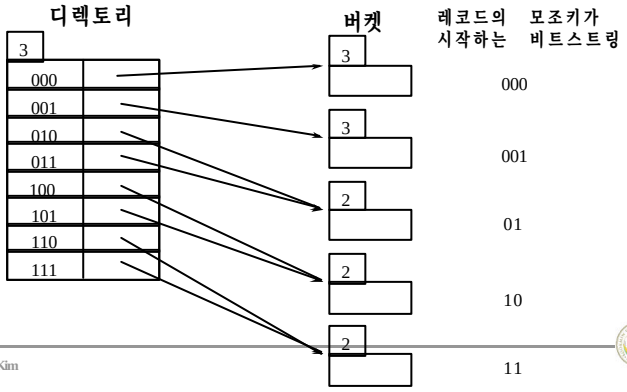
- ✶ 키값 k 모조키 101000010001
- ✷ 모조키의 처음 3비트(=d) 사용
  - ◆ 디렉토리의 6번째(101) 엔트리를 접근
- ✹ 엔트리는 4번째 버켓에 대한 포인터
  - ◆ 키값 k를 가지고 있는 레코드가 저장
  - ◆ d'= 1 : 모조키 비트 1로 시작하는 레코드 저장

□ 저장

- – 모조키의 처음 d 비트를 이용 디렉토리 접근
- – 포인터가 지시하는 버켓에 저장

※ 버킷 분할 뒤의 확장성 해싱 화일

- – 버켓 4가 만원인 상태에서 모조키가 10으로 시작하는 레코드 삽입
- – 버켓을 분할 : 빈 버켓 할당
  - ◆ 모조키 11로 시작되는 레코드를 새 버켓으로 이동
  - ◆ 원래의 버켓에 저장
- – 디렉토리의 110과 111의 포인터 값은 새 버켓을 지시하도록 변경
- – 분할된 버켓의 깊이는 2로 증가



※ 디렉토리 오버플로우

□ 첫번째 버켓(000)이 만원인 경우에 레코드 삽입

– 디렉토리 깊이(d) = 버켓 깊이(d')

– d를 증가시켜 디렉토리 확장 : 2배 증가
  - ◆ 빈 버켓을 할당받아 모조키가 0001로
  - ◆ 시작되는 레코드를 이동
  - ◆ 디렉토리 헤더와 버켓 헤더에 있는 깊이 값 증가
  - ◆ 포인터 조정