디스크와 파일

기억장치 계층 구조

- 컴퓨터 시스템의 기억장치는 계층적으로 구성됨

1. 1차 저장장치
    - 레지스터, 캐시, 주기억장치
2. 2차 저장장치
    - SSD, 자기 디스크
    - 상대적으로 느림
3. 3차 저장장치

    - 테이프
    - 가장 느린 저장장치
    - 비소멸성 저장장치

- 데이터는 디스크나 테이프에 저장하는 것이 일반적, 데이터베이스 시스템은 필요시 낮은 계층의 저장장치로부터 데이터를 가져오도록 구성됨
- 명령 실행 사이클
    1. 메모리에서 명령 인출
    - 메모리 -> cpu -> 명령레지스터
    2. 명령 레지스터에 저장
    - cpu내부에서 명령어를 저장
    3. 명령을 해독
    4. 메모리로부터 피연사자를 인출해 내부 레지스터에 저장, 피연산자에 대한 명령을 실행한 후 메모리에 저장

디스크

- 기계 장치가 포함된 저장장치, 자기를 이용해 데이터를 저장하고 읽음
- 순차 접근 방식이 아닌 직접 접근방식
- HDD
    - 자기를 이용해 플래터에 데이터를 읽거나 씀
    - 스핀들이 회전하며 디스크 헤드가 플레터에 쓰인 데이터를 읽거나 씀

SSD

- 플래시 메모리를 기반으로 한 저장 매체, Random Access 가능한 빠른 속도의 저장장치
- 모든 구성요소가 전기장치이며, 기계 장치를 가지지 않음
- 구성
    - 호스트 인터페이스 로직 : PD와 연결
    - 플래시 메모리 : 데이터 저장
    - SSD 컨트롤러 : 인터페이스와 플래시 메모리 연결하고 제어
    - 메모리 버퍼 : 외부 장치와 SSD 사이에서 버퍼 역할을 담당
- 페이지 단위로 읽기 쓰기를 함

디스크와 성능

- HDD
    - DBMS가 작업을 수행하려면 데이터는 주 기억장치에 있어야함.
    - 디스크와 주기억장치 간에 데이터 전송 단위는 HDD의 경우 블록이므로 블록 내 항목중 하나만 필요한 경우라도 블록 또는 페이지 전체 데이터가 전송되어야함
    - 블록과 페이지 입출력은 데이터 위치에 좌우
    - 접근시간 = 탐색시간 + 회전 지연시간+ 전송시간
- SSD
    - HDD와 다르게 탐색 지연 시간, 회전 지연 시간이 없이 전송 시간만 소요 되므로 임의적 읽기에서도 일정 응답 속도가 보장
    - 쓰기의 경우 비어있는 공간이 없으면 공간을 초기화하고, 이 작업 시간 동안 해당 공간에 대한 I/O 작업이 대기 상태가 됨.

운영체제 파일 시스템을 이용한 디스크 공간 관리

- 운영체제는 디스크 공간을 관리
    - 운영체제는 파일을 바이트 순서로 고수준 서비스 제공
    - 고수준의 요청을 운영체제에 따른 저수준 명령어로 바꾸어 처리함
- DBMS는 운영체제 파일 시스템을 바탕으로 데이터베이스를 관리하기도 함 대부분의 DBMS는 운영체제의 파일 시스템을 바탕으로 데이터베이스를 관리하기도 함
- 대부분의 DBMS는 운영체제의 파일 시스템에 의존하지 않음
    - 특정 운영체제의 세부적 사양에 맞추면 다양한 운영체제에서 동작하는 DBMS를 만들기 어려움
    - 운영체제는 최대 파일 크기를 제한하는 경우가 있음
    - 운영체제 파일은 여러 디스크로 분할되지 못함
    - 페이지 참조 패턴을 일반적인 운영체제보다 더 정확히 예측해야 함
    - 페이지를 디스크에 기록하는 시점에 대해 더 많은 제어를 해 주어야 함

운영체제 파일 관리 시스템에 의존하지 않는 이유

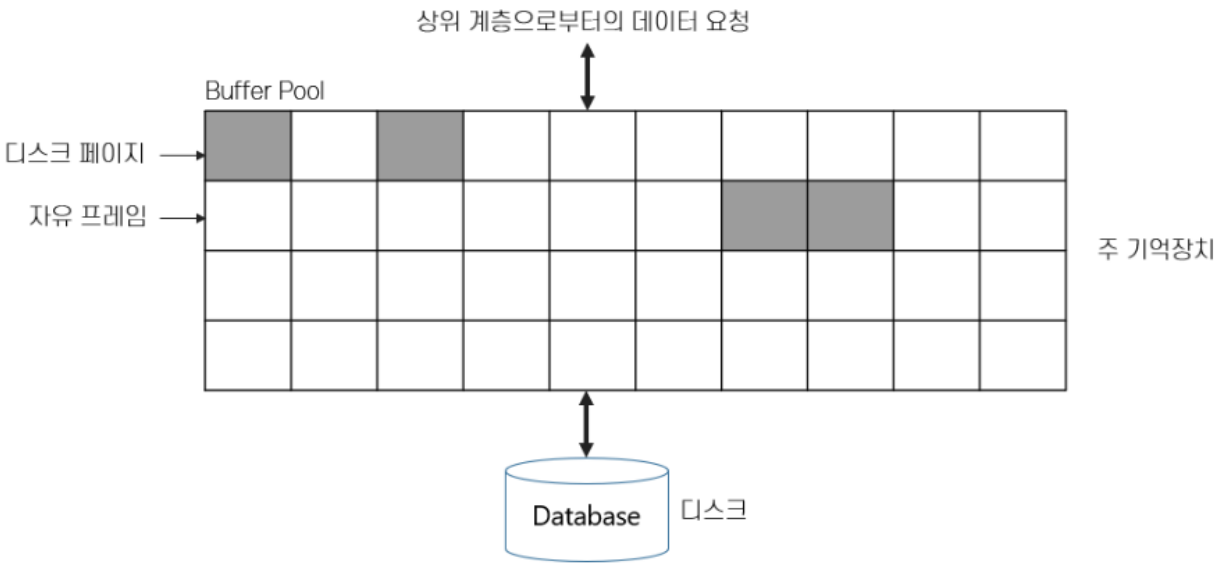
- 특정 운영체제 세부적 사양에 맞추면 다양한 운영체제에서 동작하는 DBMS를 만들기 어려움
- 운영체제는 최대 파일 크기를 제한하는 경우가 있음
- 운영체제 파일은 여러 디스크로 분할되지 못함
- 페이지 참조 패턴을 일반적인 운영체제보다 더 정확히 예측해야함
- 페이지를 디스크에 기록하는 시점에 대해 더 많은 제어가 필요함.

버퍼관리자

- 버퍼 풀
- 버퍼 교체 전략
- 버퍼 관리 기법

버퍼 풀

- 가용한 주 기억장치 공간을 페이지라는 단위로 분할한 데이터 적재 공간



- 버퍼관리자 : 사용 가능한 주 기억장치의 공간을 페이지 라는 단위들로 분할해 관리

- 버퍼풀 : 주기억장치에서 이러한 페이지가 모여 있는 공간
- 버퍼 풀내 페이지를 프레임 -프레임은 페이지를 담을 수 있는 슬롯

1. DBMS가 버퍼 관리자에게 페이지를 요청
2. 버퍼 풀 내에 페이지 적재
3. 페이지 사용 어부 표시
4. DBMS 상위 계층에서 페이지 수정후 버퍼관리자에게 전달
5. 페이지가 디스크에 기록

- 버퍼관리기에는 각 프레임마다 pin_count와 dirty라는 두 개의 변수 유지
- pin_count : 현재 프레임 사용자 수, dirty : boolean으로 페이지가 버퍼풀에 적재된 후 수정유무 표시
- 초기엔 pin_count = 0, dirty = false;

1. 버퍼 풀 점검후 요청된 페이지 있는지 조사, 페이지가 풀에 없으면 버퍼관리자가 가져옴
    1. 정해진 페이지 교체 전략에 따라 교체할 프레임을 선택
    2. 교체할 프레임의 dirty 비트가 true면 프레임의 페이지를 디스크에 저장
    3. 요청 페이지를 해당 프레임으로 로드
2. 요청 페이지를 담고 있는 프레임의 pin_count를 1 증가, 프레임의 기억장치내 주소를 반환

- Pinning :pin_count 증가
- Unpinning :pin_count 감소 > 페이지 사용을 해제 요청시
- 페이지 사용이 해제 될 때 수정된 사실을 버퍼관리자에 알리는데 이때 dirty비트가 true가 된다.
- 버퍼 교체 전략 : 요청된 페이지가 버퍼 풀에 있지 않고, 버퍼 풀에 비어있는 페이지가 없는 경우, pin_count가 0인 페이지 중 하나를 교체용으로 선정하는 전략

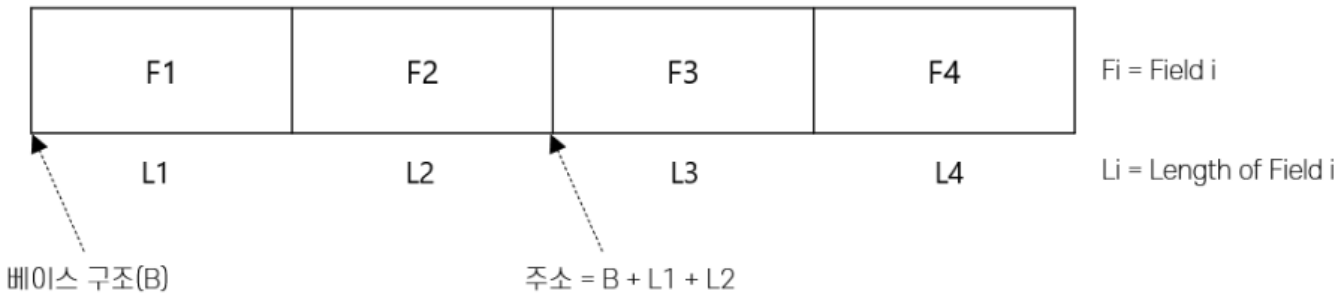
버퍼 교체 전략

- LRU(Least Recently Used) : pin_count가 0인 프레임들에 대한 포인터로 큐를 생성(가장 오래된 페이지 교체)
- Clock : LRU 변형으로, 1~N 사이의 값인 current 변수를 사용해 교체용 페이지를 선정
- FIFO(First in First out), MRU(Most Recently Used), Random등의 방식 사용

레코드 형식

- 고정 길이 레코드
- 가변 길이 레코드

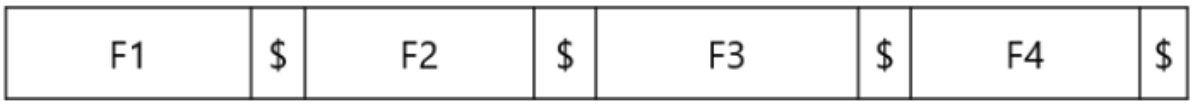
고정 길이 레코드



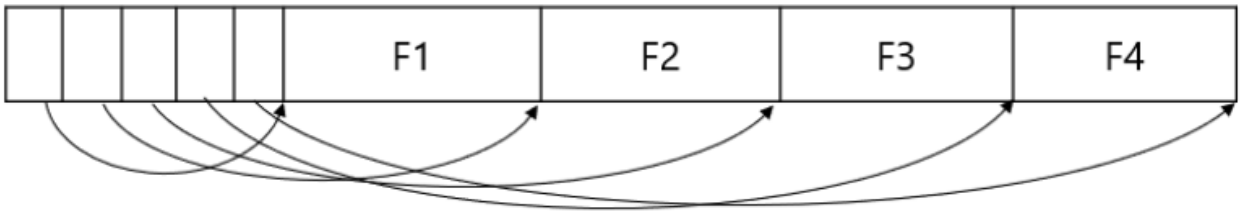
- 각 필드의 길이가 고정적이고 필드의 수도 고정된 레코드 형식 - 필드를 레코드에 연속적으로 저장 - 레코드의 주소를 얻으면 시스템 카탈로그에 있는 선행 필드들의 길이 정보를 이용해 원하는 필드의 주소를 계산해 낼 수 있어 레코드들은 연속적으로 저장 가능 - 장점 : 빠르다 - 단점 : 저장공간 낭비가 있다

가변 길이 레코드

- 필드의 길이가 가변적인 경우 해당 레코드의 길이가 가변적이 됨
- 필드의 분리자로 구분해 연속적으로 저장



- 레코드의 앞부분에 정수로 된 오프셋들을 배열로 저장



- 장점 : 저장공간 낭비가 없다
- 단점 : 느리다

페이지 형식

- 버퍼 풀에 저장되는 레코드의 집합
- 디스크에서 메모리로 데이터가 전송되는 최소 단위
- 페이지엔 각 레코드가 저장되는 슬롯이 있음

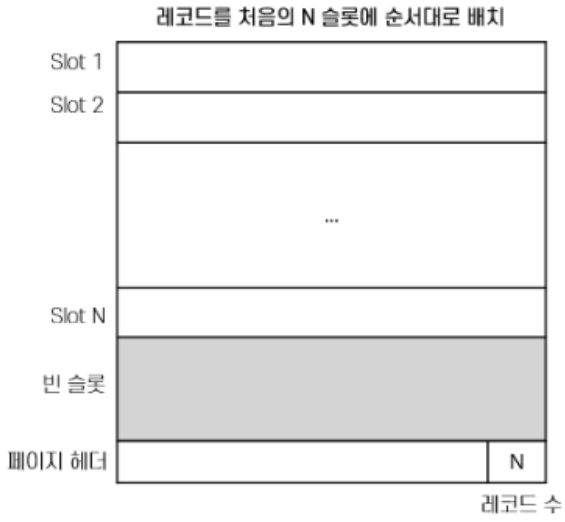
페이지 형식

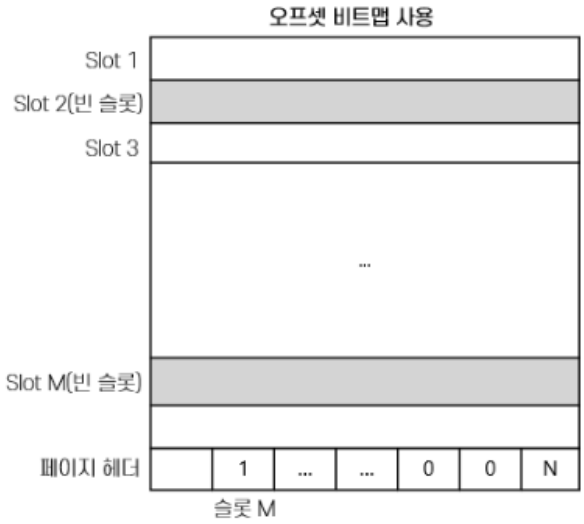
- 레코드는 <페이지 번호, 슬롯 번호>쌍
- <페이지 번호, 슬롯 번호>의 쌍을 RID<Record ID>, 레코드의 포인터 역할

고정 길이 레코드

- 고정 길이 레코드에만 탑재 될 경우 슬롯은 같은 형태, 연속적 배치 가능
- 레코드가 페이지로 삽입 될 때, 빈 슬롯을 할당하고 할당된 슬롯에 레코드 삽입

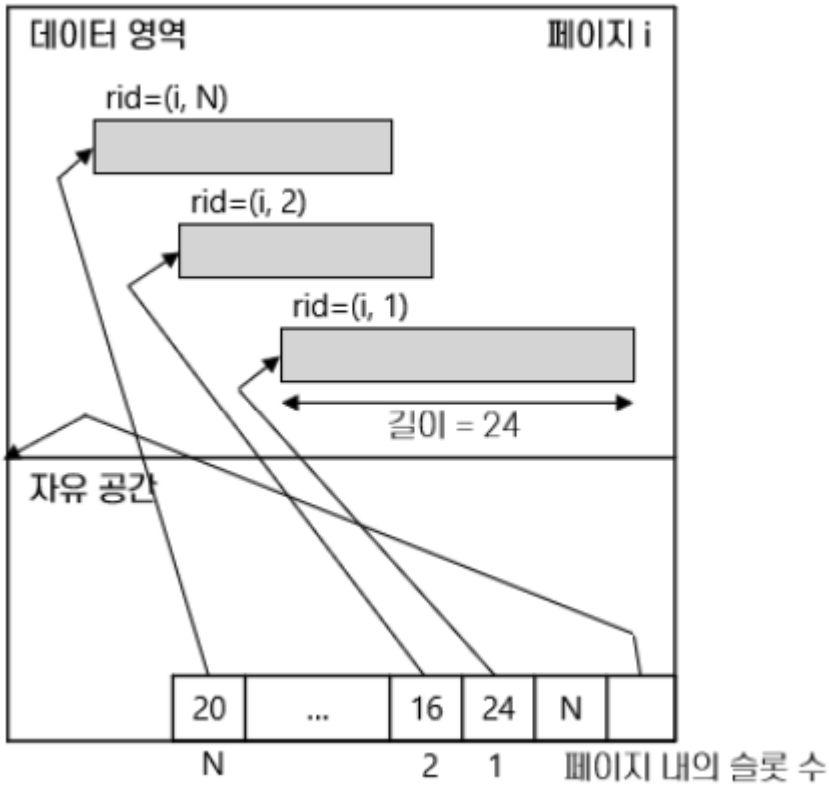
- 빈 슬롯을 어떻게 알 수 있는지에 대한 두가지 방법





가변 길이 레코드

- 레코드가 가변 길이이면 페이지를 고정된 길이의 슬롯으로 분할 불가능
- 슬롯 마다<레코드 오프셋, 레코드 길이>의 형태로 슬롯 디렉토리 유지
- 빈 공강의 시간을 가리키는 포인터 유지



파일과 인덱스

- DBMS의 상위 계층 구존느 사실상 페이지를 레코드의 집단으로 취급하고 세부적인 저장 구조에 대해서는 무시함. 레코드는 페이지에 저장되고, 페이지는 파일에 저장됨. 페이지가 파일에서 조직되는 형태에 따라 데이터베이스의 성질과 속도가 달라짐.

1. 힙파일
2. ISAM 파일
3. 인덱스 개요

힙 파일

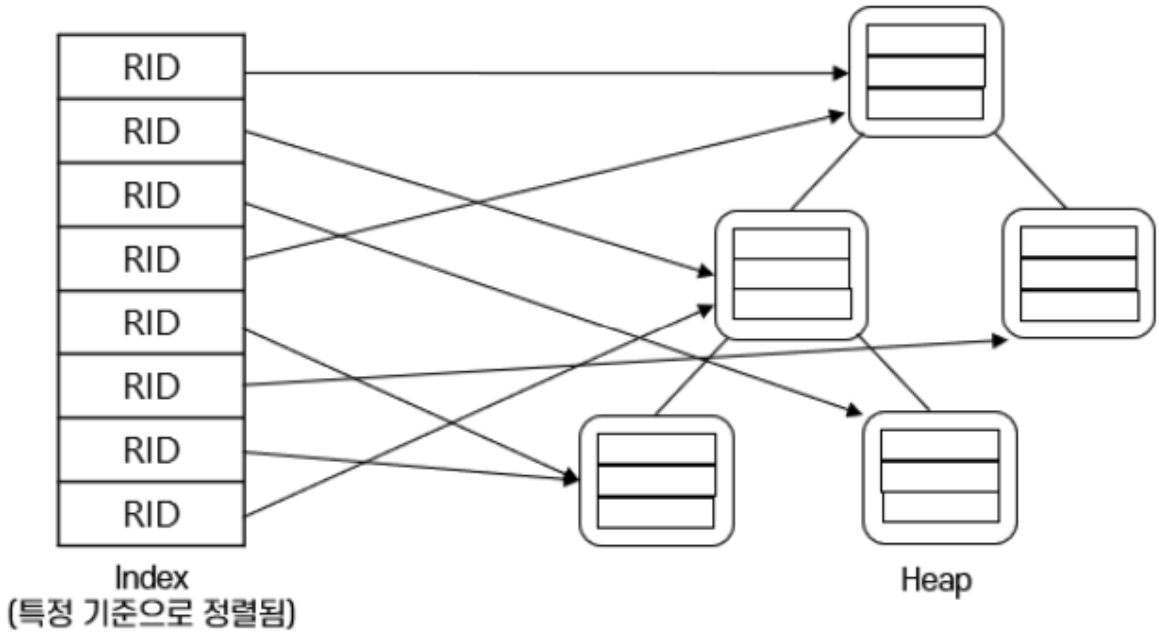
- 가장 간단한 파일구조, 레코드가 파일의 빈 공간에 순서없이 저장
- 페이지 내의 데이터가 어떠한 형태로도 정렬되지 않으며, 파일의 모든 레코드를 검색하면 다음 레코드를 되풀이해서 요청해야함
- 파일의 레코드는 각기 유일한 rid를 가지며 한 파일에 속하는 페이지는 크기가 모두 같음
    - 파일의 생성과 제거
    - 레코드의 삽입과 rid를 통한 레코드 삭제
    - rid를 통한 레코드 선택
    - 파일내의 모든 레코드 스캔 연산지원

설명

- 파일의 모든 레코드를 검색하기위해선 모든 레코드를 되풀이해서 요청
- rid를 보면 해당 레코드가 있는 페이지 번호를 알 수 있음.
- 스캔연산 - 각 힙 파일에 속한 페이지들을 알아야함
- 삽입연산 - 자유 공간이 남아있는 페이지들을 알아야함

인덱스 개요

- 대부분의 자료구조에서는 저장된 데이터의 rid를 직접 알 수 없음
- 정렬되지 않은 자료 구조에서 동등 검색을 수행할 경우, 전체 자료구조를 스캔해야함.
- 인덱스
    - 선택 조건에 맞는 rid를 구할 수 있도록 만든 보조 자료 구조



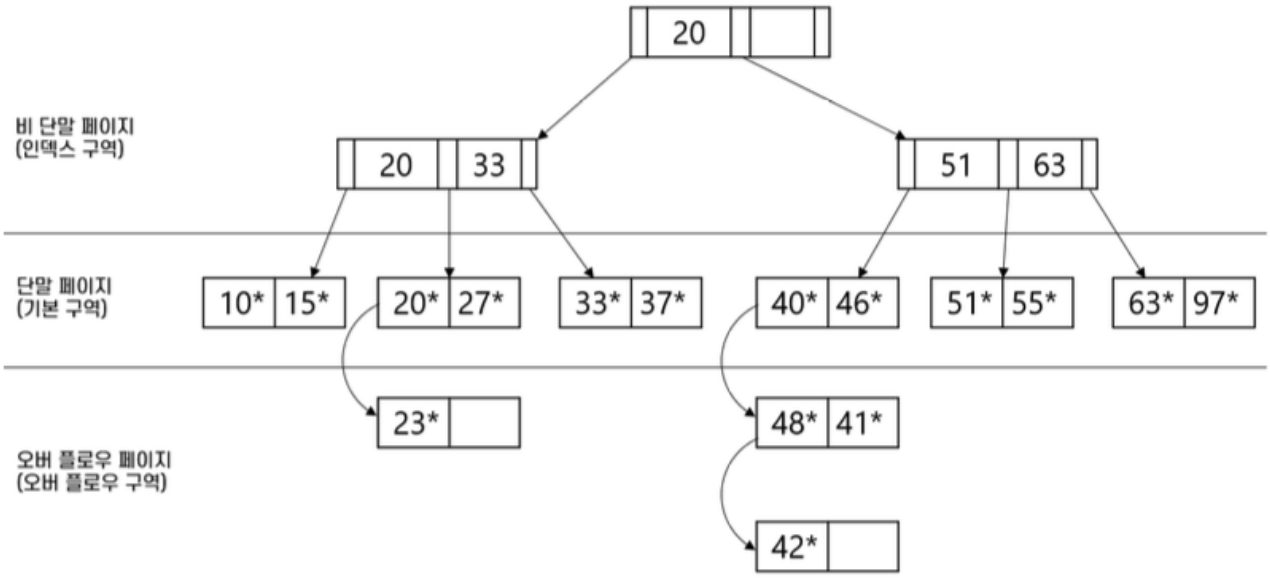
설명

> 도서관에서 분류번호를 통해 원하는 책을 찾아갈수 있다.
> 하나의 인덱스는 어느 특정한 종류의 탐색에 대해 속도를 높여줄 수 있는 보조 자료구조

- 인덱스란 데이터레코드의 요청에 대해 방향을 가리켜주는 레코드로 구성된, 또 다른 파일

ISAM(Indexed Sequential Access Method)

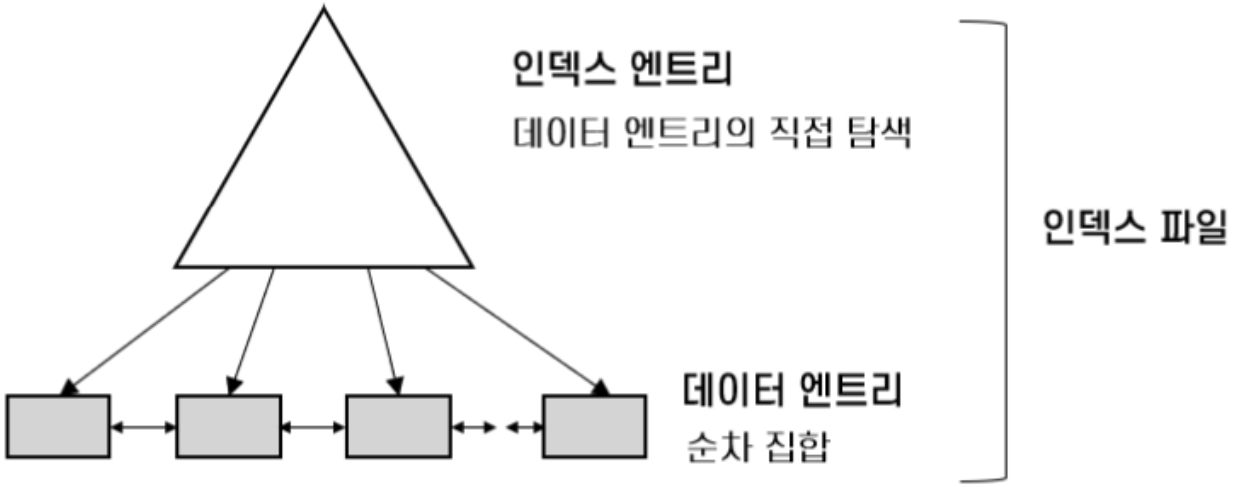
- 색인 순차 접근 방식
- 인덱스를 순차적으로 구성하는 아이디어에서 나옴
- 데이터를 순서대로 저장하거나 특정 항목을 색인으로 처리할 수 있느 ㄴ파일 처리 방법
- 인덱스를 순차적으로 구성해 큰 인덱스의 성능 문제를 해결
    - 인덱스 파일이 클 경우, 인덱스를 계층화해 인덱스에 대한 인덱스를 생성
    - 완전한 정적 구조로, 인덱스 계층의 페이지가 수정되지 않음
- 파일 구조
    - 인덱스 구역(비 단말 페이지): 기본 구역에 있는 레코드들의 위치를 찾아가는 인덱스가 기록되는 구역
    - 기본구역(기본 단말 페이지): 실제 레코드들을 기록하는 부분으로, 각 레코드는 키 값 순으로 저장
    - 오버플로우 구역(오버 플로우 페이지) : 기본 구역에 빈 공간이 없을 경우 새 레코드 삽입을 위한 예비적 구역



- 힙형식에선 모든 데이터를 다 검색해야 하지만, 파일이 정렬되어 이진검색 수행이 가능하지만 파일 크기에따라 비용이 많이 들어 isam 형식이 생김.
- 인덱스의 엔트리 크기는 페이지 크기보다 작음
- 페이지당 하나의 엔트리
- 인덱스는 데이터보다 훨씬 작음
- 인덱스 파일은 데이터 파일의 이진탐색보다 빠름 여기는 정리가 필요함. 오해의 소지가 있는 내용이 있음

B+ 트리

- ISAM의 오버 플로의 단점으 개선한 동적 트리 자료구조
- 내부 노드들이 탐색 경로를 유도하고 단말 노드들이 데이터 엔트리를 가지는 균형 트리
    - 트리에서 삽입, 삭제를 수행해도 트리의 균형이 유지
    - 레코드를 탐색시, 루트로부터 알맞은 단말까지만 가면 됨
- 일반적으로 ISAM보다 좋음



특징

- 삽입 삭제를 수행해도 트리의 균형이 유지
- 파일은 일반적으로 축소보다 확장되므로 삭제시 파일을 축소시키지 않음
- 레코드를 탐색 시, 루트로부터 알맞은 단말까지만 가면 됨. 루트로부터 단말까지 이르는 경로의 길이를 높이라고 하는데, 균형 트리이므로 어느 단말이든 이 값은 같음
- 보통 67프로 정도만큼의 공간만 채움

시스템 카탈로그

- 데이터 베이스 시스템의 근본적인 성질 중 하나는 자신이 가지고 있는 모든 데이터에 대한 설명 정보를 유지 관리, 이러한 정보를 저장한 데이터베이스를 시스템 카탈로그
- 데이터베이스 자신의 모든 데이터에 대한 설명정보를 담음
- 모든 릴레이션과 인덱스에 대한 정보를 유지관리
- 시스템 카탈로그, 데이터 사전, 마스터데이터베이스, 메타데이터라고 부름
- 저장되지 않은 뷰에 대한 정보도 유지관리 및 정의, 질의를 받으면 정의된 뷰를 이용해 뷰에 속하는 투플을 계산함

TIP

> 데이터 베이스 테이블을 빠르게 찾기 위한 자료구조