- 8. 데이터 저장과 파일
  - 1. 물리적 저장장치 구조[288]
    - 개요
      - 휘발성(volatile) 기억장치
        - 전원 공급이 중단되면 기억장치 안의 모든 데이터가 소멸
      - 비휘발성 기억장치
        - 데이터 보관하기 위한 용도로 사용
  - 2. 파일의 구성 [289]
    - 개요
      - DBMS는 데이터의 영구적 저장을 위해 운영체제가 제공하는 파일 시스템을 이용하며, 이를 통해 저장장치에 접근하고 파일에 저장된 데이터를 사용한다.
      - 파일(file)
        - 데이터를 영구적으로 저장하기 위해 사용하는 컴퓨터 시스템의 가장 기초적인 구조
        - 파일은 논리적 구조이기 때문에 물리적으로는 여러 개로 분할하여 저장될 수도 있다.
        - 분할되더라도 사용자에게 제공될 때는 완전한 내용을 갖춘 형태로 재구성된다.
      - 블럭(block)
        - 파일을 고정적 길이로 분할하여 생기는 균등한 크기의 데이터 묶음
        - 파일은 블럭 단위로 분할되어 기록된다.
        - 사례
          - 블럭 하나의 크기가 4KB라고 할 때, 용량이 12KB 파일은 3개의 블럭으로 나뉘어 저장된다.
          - 파일 전체에 대한 읽기 요청을 처리하기 위해서는 파일을 구성하는 3개의 블럭을 디스크로부터 읽어 들여 메모리에 적재해야 한다.
          - 12KB 파일을 디스크에 기록할 경우, 메인 메모리에서 디스크로 3개의 블럭을 분할하여 전송
        - 블럭의 단위
          - 운영체제가 프로그램(즉 사용자)에 데이터를 전달하기 위해 디스크에서 메모리로 적재(페이징 인)하는 최소한이ㅡ 기본 단위
      - 파일(1) : 블럭 (N) 블럭(1) : 레코드(N)
        - 상황에 따라 한 레코드가 블럭 크기보다 커서 여러 블럭에 나뉘어 저장되는 경우가 있다
        - 관계형 모델에서 레코드
          - 더 이상 분리되어 저장되지 않는 최소 데이터 저장 단위로, 논리적으로 연관되어 있는 데이터 집합
    - 1. 고정 길이 레코드
      - 고정 길이 레코드(fix-length records)
        - 각 레코드에 고정적인 바이트 수를 할당하는 방법
        - 어떠한 블럭 내에서 세 번째 사원 레코드를 읽고 싶을 경우 앞 2개의 레코드 길이 84바이트(42바이트x2)를 건너뛰고 85번째 바이트부터 42바이트를 읽어 들이면 된다.
      - 장점
        - 구현이 단순
        - 탐색 방법이 매우 직관적
      - 한계
        - 블럭의 길이가 레코드 길이로 정확히 나뉘어 떨어지지 않는다면, 어떤 레코드들은 2개의 블럭에 나뉘어 저장되어야 한다.
          - 블럭의 길이가 레코드 길이로 정확히 나뉘어 떨어지지 않는다면, 어떤 레코드들은 2개의 블럭에 나뉘어 저장되어야 한다.
          - 블럭 내의 남은 공간은 공백으로 남김으로써 공간을 낭비하게 된다.
          - 레코드를 두 블럭에 나누어 저장하는 것을 허용하는 시스템인 경우 하나의 레코드를 읽거나 쓰기 위해 2개의 블럭에 접근해야 하는 상황이 발생하기 때문에 DBMS 성능이 저하된다.
          - 한 블럭에만 저장하는 시스템의 경우 블럭의 일부 공간을 비워 놓는 공간 낭비가 발생한다.
        - 레코드 삭제 작업 후 빈 공간에 대한 관리가 어렵다.
          - 특정 레코드가 삭제되었을 때 레코드가 저장되었던 자리에 빈 공간이 생성된다.
            - 1. 가장 마지막에 위치한 레코드를 빈 공간에 재할당
            - 2. 무의미 문자로 채워놓기
            - 3. 삭제된 레코드 이후의 모든 레코드를 한 칸씩 위로 이동
              - 레코드의 순서를 훼손하지 않지만 많은 수는 디스크 입출력을 동반
          - 가용 리스트(free list)

- 발생한 빈 공간을 표시하는 것만으로는 삽입 시 가용 공간을 찾는 것이 어려워지기 때문에 가용 공간을 관리하는 가용 리스트라는 추가적인 구조를 사용
- 아직 사용되지 않았거나 기존 레코드가 삭제되어 비어있는 공간에 대한 정보를 관리하는 자료구조
- 파일 헤더(file header)
  - 파일의 시작점에 있는 파일과 관련된 다양한 정보가 저장됨
  - 가용 리스트 정보를 위치시킨다.
  - 새로운 레코드 삽입 시 파일 헤더가 가리키고 있는 빈 공간에 레코드를 삽입하고, 두 번째 빈 공간의 주소를 파일 헤더에 저장한다.
  - 만약 가용 리스트에 빈 공간이 없다면 파일의 맨 끝에 레코드를 삽입한다.

2. 파일의 구성 (2) [293]

2. 가변 길이 레코드

- 가변 길이 레코드(variable-length records)
  - 블럭에 저장되는 레코드의 길이가 서로 다를 때 레코드에 가변적인 길이를 할당하는 방법
  - 사용처
    1. 한 블럭 내에 저장되는 레코드가 서로 다른 테이블의 것일 때
    2. 길이가 고정되지 않은 컬럼의 개수가 하나 이상일 때
       - VARCHAR, TEXT, JSON, MULTISET, ARRAY 등
    3. 레코드가 멀티셋(multiset)을 허용하는 컬럼을 가질 때
       - 컬럼에 해당하는 값이 여러 개일 수 있는 경우
- 일반적으로 고정 길이 컬럼은 레코드의 앞쪽 부분에, 가변 길이 컬럼은 레코드의 뒤쪽 부분에 위치시켜 컬럼의 길이 변화에 필요한 작업을 최소화한다.
- 가변 길이 레코드의 가장 첫 부분은 가변 길이 컬럼에 대한 위치 정보를 저장한다.
  - 레코드 앞 부분의 (27, 6)은 27번째 바이트부터 6바이트가 가변 길이 컬럼인 것을 나타낸다.
  - 고정 길이 컬럼과 가변 길이 컬럼들 사이에는 1바이트의 널 바이트가 추가되는데 두 유형의 컬럼을 구분해주는 역할을 수행한다.
- 하나의 블럭에 여러 개의 가변 길이 레코드를 저장할 때, 고정 길이 레코드와 달리 각 레코드의 길이가 불규칙하기 때문에 각 레코드의 시작과 끝 위치를 나타내는 등 레코드의 접근 및 관리를 위한 별도의 자료 구조가 필요하다.
  - 슬롯 페이지 구조(slotted-page structure)
    - 블럭의 초반부는 블럭 헤더로 구성
    - 슬롯(Slot)
      - 각 레코드의 위치 및 길이 요약 정보를 담고 있는 테이블. 레코드가 어디에 저장되어 있는지 가리키는 포인터
    - 블럭 헤더
      - 블럭에 대한 요약 정보를 담고 있음
        - 저장된 레코드 수, 각 레코드의 블럭 내 위치, 크기에 대한 정보를 가지고 있어 레코드를 빠르게 찾기 위해 사용된다.
    - 신규 레코드는 슬롯 페이지의 중간에 있는 가용 공간의 뒤쪽부터 저장되는데 이를 위해 블럭 헤더가 가용 공간의 끝에 대한 위치가 저장된다.
      - 가용 공간의 뒤쪽부터 신규 레코드가 저장되는 이유
        1. 슬롯과 레코드의 충돌을 방지 슬롯(메타데이터)은 앞쪽부터, 레코드(실제 데이터)는 뒤쪽부터 채움으로써
        2. 레코드를 앞쪽부터 채우면 가변 길이 레코드가 기존 레코드를 밀어야 하는 상황이 자주 발생
        3. 슬롯만 수정하면 되므로 성능 향상
        4. 압축/정리(compaction) 시 유리
    - 새로운 레코드들은 가용 공간의 끝 부분에서부터 역방향으로 저장된다.
    - 레코드 삭제 시 해당 레코드가 차지하고 있던 공간이 반환된다.
      - 이때 생성되는 빈 공간을 메우기 위해 삭제된 레코드 이전에 존재하는 모든 레코드의 위치를 당겨서 저장한다.
        - 한 블럭 내 레코드들의 위치만 이동하기 때문에 많은 비용을 발생시키지 않음
        - 블럭 내 레코드들의 위치가 조정된 후 레코드 이동상황을 블럭 헤더에 반영한다.

3. 파일 구조

- 파일 구조(file organization)
  - 특정 레코드에 대한 접근을 위해 어떤 레코드가 어느 블럭에 저장되어야 하는지 관리
  - 유형
    - 힙(heap) 파일 구조
      - 파일 내 임의의 블럭에 저장될 수 있는 방식으로, 각 레코드들의 저장 순서를 고려하지 않음

- 순차(sequential) 파일 구조
  - 레코드들이 특정 컬럼에 대한 값을 기준으로 정렬되어 저장되는 방식으로, 일반적으로 정렬키로 탐색키를 사용한다.
- 해시(hash) 파일 구조
  - 해시 함수를 이용하는 방식으로, 해시 함수는 레코드 탐색키를 입력받아 레코드가 저장될 블럭 주소를 반환하고 해당 주소에 레코드를 저장한다.

- 순차 파일
  - 레코드가 탐색키(search key) 순서대로 정렬되어 저장
    - 탐색키는 반드시 기본키일 필요 없음
  - 키 순차 파일(key-sequence file)
    - 각 레코드의 탐색키값 순서로 물리적 레코드를 정렬하여 생성되는 파일
    - 레코드 접근은 레코드가 저장되어 있는 물리적 순서에 따르기 때문에 데이터 탐색 시 블럭의 접근 횟수를 최소화할 수 잇다.
    - 보다 빠른 접근을 위해 포인터로 순차적 연결한 연결 리스트 형태로 순차 파일을 구성할 수 있다.
  - 장점
    - 탐색키에 대한 정렬 연산이 불필요하므로 정렬된 키 값들의 순서로 레코드를 판독하는 연산이 매우 효율적
    - 현재 레코드에서 정렬된 키 순서로 다음 레코드를 찾을 때 부가적인 블럭 접근을 필요로 하지 않는다.
    - 탐색키로 사용된 컬럼의 값을 기반으로 탐색할 때 이진 탐색을 사용하면 원하는 레코드를 더 빠르게 검색할 수 있다.
  - 단점
    - 탐색키가 아닌 컬럼의 값을 이용하여 탐색하는 것이 비효율적이다.
      - 임의 접근을 위해서는 선형 탐색을 사용하고, 비순차 컬럼의 순서대로 레코드에 접근하기 위해서는 파일을 비순차 컬럼 순으로 정렬한 사본을 별도로 만들어 사용해야 한다.
    - 레코드 삽입과 삭제 연산 비용이 매우 크다
  - 특징
    - 레코드가 물리적으로 정렬된 상태로 유지되어야 하므로 레코드 삽입 시 순차 컬럼값(탐색키)을 기반으로 삽입할 블럭을 찾고, 그 위치에 삽입할 충분한 공간이 있으면 삽입을 완료한다.
    - 오버플로 블럭(overflow block)
      - 만약 공간이 없다면 신규 블럭에 삽입하고 신규 레코드의 이전 레코드가 신규를 링크하고, 신규 레코드 다음 레코드를 신규 레코드가 링크하도록 포인터가 재조정한다. 이 때 레코드 삽입을 위해 사용한 신규 블럭을 오버플로 블럭(overflow block)이라고 한다.

3. 저장장치 접근 [298]

- 개요
  - DBMS에서 관리하는 파일들은 영속성을 위해 보조기억장치인 디스크에 저장된다.
  - 파일은 논리적 관점에서의 저장 단위이기 때문에 실제 저장될 때는 여러 개의 물리적 단위인 '블럭'으로 분할되어 저장된다.
  - 블럭의 크기
    - 시스템과 운영체제에 따라 다르지만 일반적으로 2KB, 4KB, 8KB, 16KB, 32KB가 주로 사용된다
  - 블럭
    - 디스크와 메인 메모리 사이의 기본 데이터 전송의 단위
  - 블럭을 찾는 과정
    - 사용자에 의해 특정 레코드가 데이터베이스 서버로 전송 요청 시, 운영체제는 DBMS로 요청을 전달하고 DBMS는 요청에 포함된 조건을 파악하여 조건에 해당하는 레코드가 포함된 블럭을 찾는다.
    - DBMS는 찾은 블럭을 메인 메모리에 전송, 적재하고, 최종적으로 클라이언트에게 요청 레코드를 전달한다.
    - 주의사항
      - 디스크와 메인 메모리 사이에서 하나의 블럭이 전송되는 속도는 프로세서(CPU)의 연산 속도에 비해 매우 느리다
    - DBMS는 블럭 전송 횟수를 최소화하여 입출력에 소요되는 시간을 단축하고 질의에 대한 반응 속도를 최소화할 필요가 있다
  - 블럭 전송 횟수의 최소화
    - 메인 메모리에 사용 중인 블럭을 적재해 놓는 방법을 사용할 수 있다.
    - 특정 파일에 접근할 때마다 디스크에서 블럭을 읽어 들이고 수정 내용을 곧바로 디스크에 기록하는 작업에는 오랜 시간이 소요된다.
      - 사용 중인 블럭을 메모리에 적재시켜 놓고 메인 메모리 내에서 작업한다면 블럭 전송, 즉 디스크 입출력을 줄일 수 있다.
  - 보통 메인 메모리보다 DBMS 관리 데이터 용량이 크다.
    - 필요에 따라 특정 블럭 할당을 해지하여 새로운 공간을 만들기 위해 해당 블럭을 디스크에 기록해주는 기술이 필요하다.
  - 버퍼 매니저의 역할

- 버퍼(buffer) / 버퍼 매니저(buffer manager)
  - DBMS는 메모리 내부에 버퍼라는 공간을 만들어 디스크로부터 가져온 블럭들을 저장하며, 하위 시스템인 버퍼 관리자(buffer manager)를 통해 버퍼를 효율적으로 관리한다.
- DBMS 상의 프로그램이 필요한 블럭이 있을 때 버퍼 관리자에 해당 블럭을 요청
  - 요청된 블럭이 이미 버퍼에 있다면, 버퍼 관리자는 블럭이 위치한 메모리 내의 주소를 요청한 프로그램에 전달한다
  - 요청한 블럭이 버퍼에 없다면 버퍼 관리자는 요청한 블럭을 적재하기 위한 공간을 새로 할당한다
  - 적재할 공간이 없다면 블럭 할당 공간을 만들기 위해 버퍼에 잇는 기존 블럭 하나를 선택하여 할당을 해지해야 한다.
  - 선택된 블럭이 메모리에 적재될 당시와 동일한 상태의 블럭이라면 해당 블럭을 디스크에 재기록할 필요가 없다.
  - 선택된 블럭이 수정된 적이 있다면 변경 내용을 반영하기 위해 디스크에 기록한다.
  - 버퍼 관리자는 프로그램이 요청한 블럭을 디스크로부터 읽어 들인 후 새로 생긴 공간에 해당 블럭을 적재한다. 그리고 블럭이 적재된 메모리 주소를 프로그램에 전달한다.
    - 버퍼 관리자에 의해 추상화되기 때문에 애플리케이션은 상세 내용에 대해 고려할 필요 없음
- 버퍼 교체 전략, 고정 블럭, 블럭 강제 출력 등의 기능을 수행한다.
  - 버퍼 교체 전략(buffer replacement strategy)
    - 버퍼에 더 이상 가용 공간이 없을 경우, 새로운 블럭을 적재하기 위해 기존에 적재된 블럭 하나를 선택하여 할당을 해제
    - LRU(Least Recently Used)
      - 참조된지 가장 오래된 블럭을 교체
    - MRU(Most Recently Used)
      - 최근에 참조된 블럭을 우선 교체
  - 고정 블럭(pinned blocks)
    - 하드웨어 혹은 소프트웨어 결함으로 인하여 휘발성 기억장치의 데이터가 손실되어 작업이 중단될 경우, 중단된 작업의 중간 결과물이 디스크에 기록되는 것을 방지
      - 임시적으로 디스크 블럭이 교체되는 것을 제한할 필요가 있음
    - 복구 시스템은 블럭이 갱신되고 있는 동안에는 블럭이 교체되는 것을 제한하기 위해 특정 블럭을 고정시키는데 이를 고정 블럭이라고 한다
  - 블럭 강제 출력(forced output of blocks)
    - 시스템 로그와 같은 중요한 데이터는 안정적인 시스템 운용을 위하여 디스크에 영구적으로 기록되어야 한다.
    - 중요 블럭은 버퍼 공간이 부족하지 않은 상태임에도 필요에 따라 강제로 디스크에 기록됨