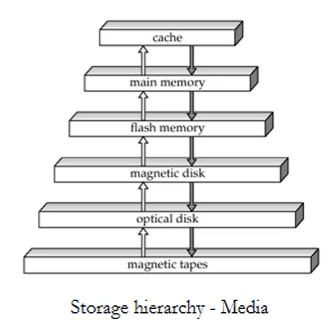
**- Chapter 10: Storage & File Struct**

**Physical Storage Media**

물리적인 저장매체. Disk, Memory, Flash Memory, CD-ROM 등이 있음.

데이터 액세스 스피드, 데이터 저장 비용, 신뢰성 (파워가 꺼지거나 시스템 충돌 시 데이터를 잃는지), 지속성 (volatile, non-volatile) 등으로 나눠짐.

**Storage Hierarchy – Operational**



- Primary Storage

가장 빠르지만 volatile임. Cache와 main memory.

- Secondary Storage

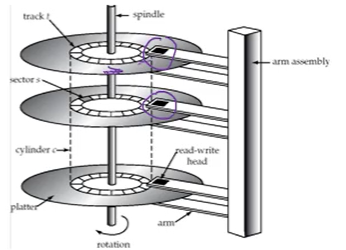
Non-volatile. On-line storage라고도 함. Flash Memory, magnetic disk가 있음.

프로그램을 수행되면서 수시로 access가 가능한 storage를 on-line이라고 함. Primary도 당연히 online.

- Tertiary Storage

Non-volatile. 느림. Off-line storage. Magnetic tape, optical storage 등이 있음.

**Magnetic Disks (하드디스크)**

****

Read-write head가 있음. 각 platter는 track으로 나누어져 있음. 각각의 track은 sector로 나누어져 있음. 주로 섹터의 크기는 512byte. 보통 트랙당 안 쪽은 500, 바깥 쪽은 1000개의 섹터가 있음.

섹터를 읽으려면 disk arm이 맞는 트랙으로 움직여야 함.

같은 지름의 track들을 Cylinder라고 함 - 모든 플래터의 i번째 트랙.

Head-Crash는disk에서 외부 충격이나 먼지가 껴서 손상이 되면 못 읽음.

**Disk Controller**

컴퓨터 시스템과 HDD 사이의 인터페이스. Read, Write 수행.

Robustness를 관리함

- 각 섹터마다 checksum을 계산하고 붙여둠.

- Bad sector를 remapping하는 기능을 수행함.

**Performance Measures of Disks**

- Access Time

Read나 write가 요청된 시간부터 데이터 전송이 시작된 시간 사이의 길이

Seek time – arm이 알맞은 트랙까지 가는 시간. 보통 4~10ms.

Rotational Latancy – 섹터가 head 밑으로 오기까지 걸리는 시간. 보통 4~11ms.

- Data-transfer Time

데이터가 disk로부터 받아오거나 저장되는 비율.

보통 25~100MB/s임.

- Mean time to failure (MTTF) or MTBF

디스크가 failure 없이 꾸준히 동작하기로 기대되는 시간. 보통 3~5년.

새 디스크의 경우 고장날 확률이 낮음. 나이가 들면서 수명이 줄어듦.

**Optimization of Disk-Block Access**

**Block**

하나의 track에 있는 연속된 sector들의 시퀀스

데이터가 디스크와 메모리 사이에 전송되는 크기가 블락임.

사이즈는 512byte부터 몇 kb까지 다양함. Block이 작으면 disk로부터 많이 전송받아야 하고, 블락이 크면 공간 낭비가 심함.

요즘엔 보통 4~16kb.

**Disk-arm-scheduling**

Disk arm의 이동을 최소화하기위해 명령들을 queue 형태로 미뤄놓음.

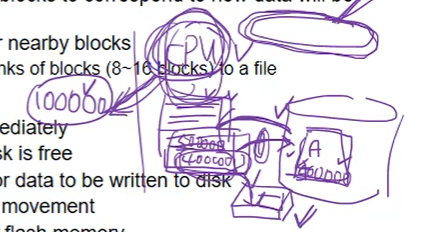
Elevator algorithm

- 디스크를 한 방향으로 움직임. 움직이는 방향에 있는 요청들 수행.

**File Organization**

파일을 어떻게 저장하느냐에 따라 block access time을 최적화할 수 있음. 보통 같은 파일을 같거나 근처의 block에 저장함.

**Nonvolatile write buffers**

****

block들을 non-volatile RAM 버퍼에 바로 씀.

**Log Disk**

Block update의 연속적인 log를 쓰기 위해 전용의 disk를 둠.

Non-volatile RAM과 비슷한데, HW가 따로 필요 없음. Seek이 필요 없기 때문에 write가 빠름.

**RAID**

Redundant Arrays of Independent Disks. 여러 개의 디스크를 묶어서 하나처럼 사용하는 것.

원래 I는 inexpensive였음.

High capacity, high speed, high reliability.

**Reliability via Redundancy**

Redundancy란 추가적으로 정보를 저장하는 것.

- Mirroring (or shadowing)

모든 디스크를 복제함. 미러 디스크의 MTTF는 57000년. 수리 시간이나 failure의 독립성 등에 의존함. 이를 full duplication이라고 함.

- Parity

모든 n개의 데이터 비트(혹은 블락)마다 parity bit를 저장해둠.

**Performance via Parallelism**

다수의 데이터로부터 data를 stripe함. striping이란 데이터를 나누어 담는 것.

- Bit-level striping

각각의 바이트의 비트들을 여러 개의 디스크에 나눠서 저장함.

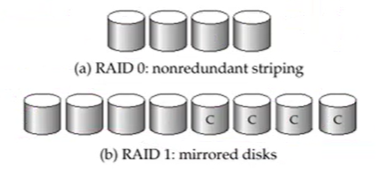
만약 8개의 디스크가 있으면 bit i를 각각의 디스크 i에 저장하면 됨.

- Block-level striping

n개의 디스크에, 파일의 i번째 block이 디스크로 들어감.

(i mod n) + 1.

**RAID Levels**

****

RAID Level 0: Block stripping; non-redundant. 데이터 손실이 치명적이지 않은 high performance 시스템에서 사용.

RAID Level 1: Mirrored Disk with block striping. 레벨 0의 방법과 같은데 그걸 통째로 mirroring 하는 것. Write 퍼포먼스는 최고. DB의 로그 파일을 저장하는 등의 용도로 많이 씀.

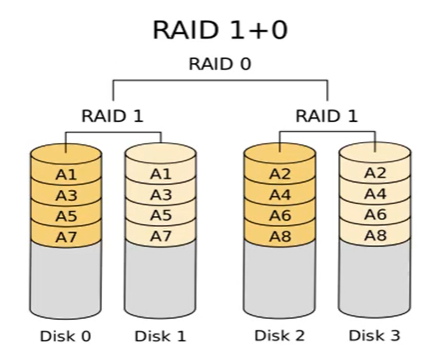
다른 책에서는 Level 1을 striping을 생각하지 않기도 함. 하지만 우리는 이 책을 따름.

RAID Level 5: Block-interleaved distributed parity

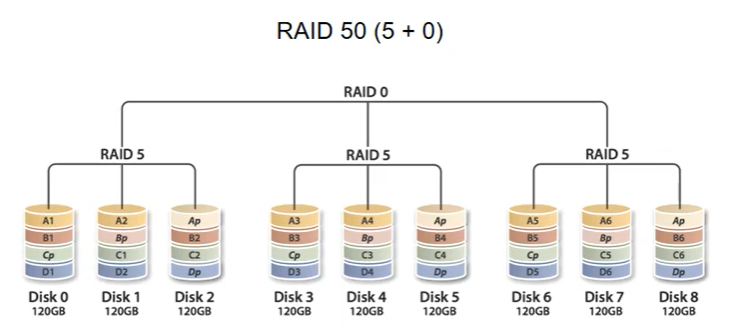
레벨 4는 parity block을 하나의 disk에 저장하는 건데, 얘의 단점을 보완한 것. Parity를 디스크 여러 개에 나눠서 저장.

레벨 1과 비교하면 storage overhead가 적고, write 오버헤드가 큼. Reliability도 조금 떨어짐.

**Non-Standard RAID Levels – Nested/Hybrid**



만약 Mirrored Disk만을 RAID 1이라고 하면 책에서 언급한 RAID 1의 개념은 위의 방식으로 RAID 1+0으로 나타낼 수 있음.



RAID 5에 0을 덮은 것.

**File Organization**

데이터베이스는 files의 collection으로 저장됨.

각각 파일은 record의 sequence.

Record는field의 sequence.

얘네는 block들의 단위로 저장됨.

Fixed length records

Record 크기가 고정되어 있다고 가정.

각각의 파일은 1가지 타입으로만 이루어진 레코드들을 가짐

다른 파일들은 다른 relation을 위해 저장됨.

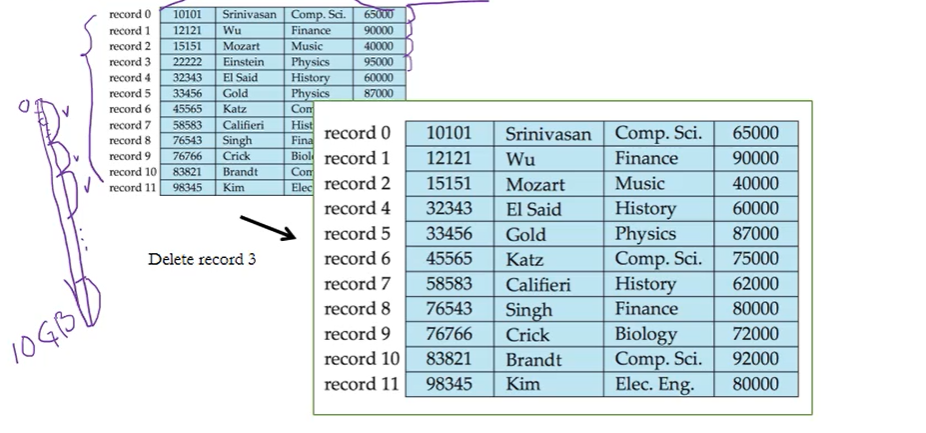
Variable length records

다양한 record 타입을 파일에 저장

하나 이상의 가변 길이를 가진 record type을 허용

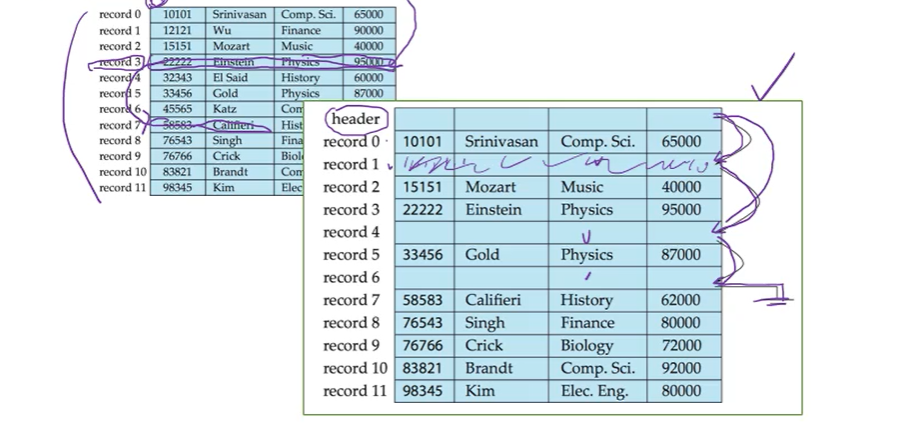
반복되는 field의 record type 허용.

**Fixed Length Records**



Record가 delete되면 순차적으로 shift함. 단점은 하나의 record가 delete되었는데 다 shift해야함. 이 때 sequential read 성능이 좋음. Maximum available space 보장.

다른 방법은 마지막 record를 delete한 자리로 옮김. 장점은 위와 같고 단점은 순서가 흐트러짐.



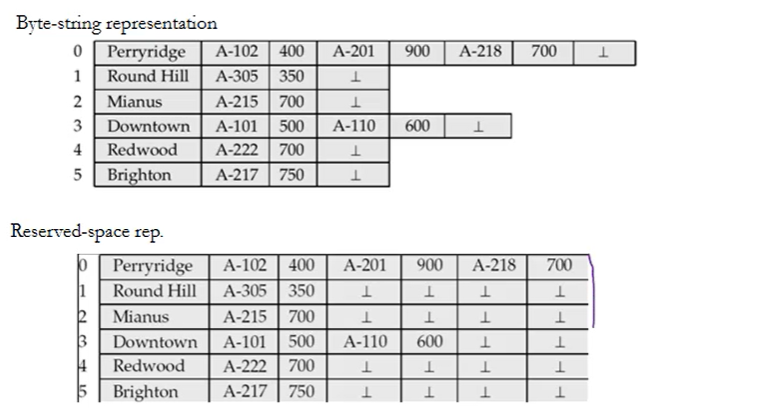
또 다른 방법은 해당 delete된 부분을 빈 공간으로 놔두는 것. 그리고 맨 앞에 available 공간에 대한 pointer를 두고 빈 공간들만 링크드 리스트처럼 연결함.

장점은 delete나 insert가 constant. 단점은 빈 공간이 숭숭 뚫리는 것.

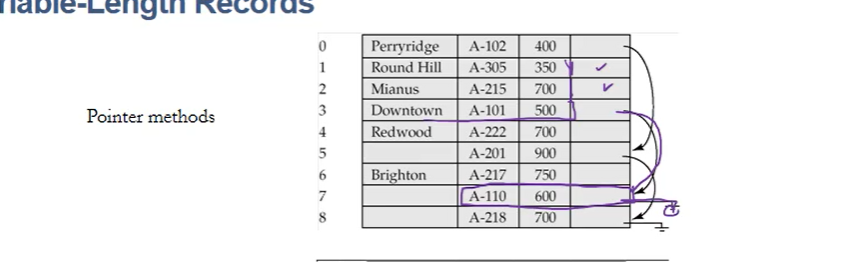
중요한 것이 무엇이냐에 따라 다른 방법을 선택해야 함.

보통 블락 내에서 shift하고 빈 공간은 블락 내에서 최대한 보장하는 방법을 많이 사용.

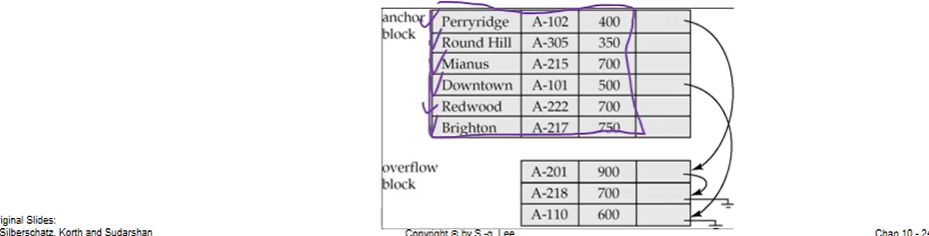
**Variable-Length Records**



위 방법은 record 길이를 모두 읽어봐야 함. 밑은 좀 더 단순한 방법으로 variable이지만 fixed 처럼 사용. 단점은 공간 낭비.



Variable 부분을 이런 식으로 pointer로 관리할 수도 있음. Fixed length로 공간을 관리하지만 retrieve할 때는 pointer 사용.



또 다른 방법으로 fixed를 anchor block에 놔두고, variable 부분은 overflow block에 놔둠. 단점은 block을 두 개 읽어야 함.

하나의 레코드를 같은 블락에서 access하는 것이 중요한지, 헤더 부분을 access하는 것이 중요한지에 따라 골라야 함. 현재는 overflow 방법은 잘 사용하지 않음.

**Variable-Length Records**

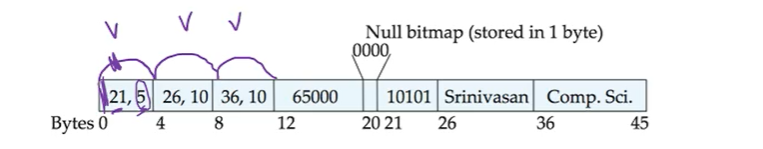
Attribute는 순서대로 저장됨.

Variable length attributes

- record의 시작 부분에 fixed size(offset, length)로 나타남.

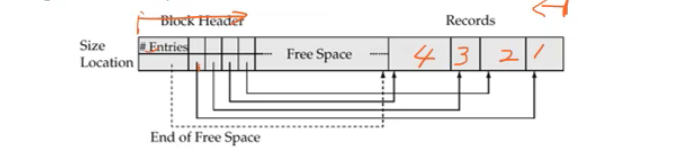
- 실제 데이터는 모든 fixed length attribute 이후에 저장됨.

Null value는 null-value bitmap에 의해 표현됨. 밑의 예시는 지금은 null이 없음. 만약 2번째 attribute가 null이면 0100 형태로 저장.



각 record의 시작점만 알면 모든 attribute를 상수 시간에 접근 가능.

**Slotted Page Structure**

****

Variable-length record는 Byte-string 으로 나타냄.

Record가 block보다 작다고 가정.

Page header는 이런 것들을 담음.

- record entry의 개수

- 해당 block의 free space의 끝

- 각 record의 location와 size.

장점

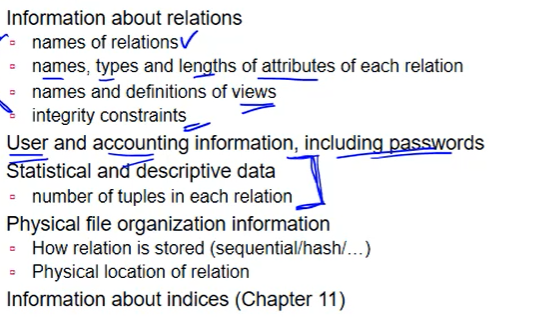
Record에 대한 indirect 포인터를 지원.

블락 안에서의 공간의 파편화 방지.

뭔가 stack과 heap처럼 서로 반대 방향으로 쌓인다.

**Data Dictionary Storage**

Data dictionary (system catalog로도 불림)가 metedata를 저장함. (데이터에 대한 데이터) 밑의 정보 같은걸 담고 있음. 테이블들에 저장.



**Storage Access**

데이터베이스 파일은 block이라고 불리는 고정 길이의 storage unit으로 나뉘어져 있음.

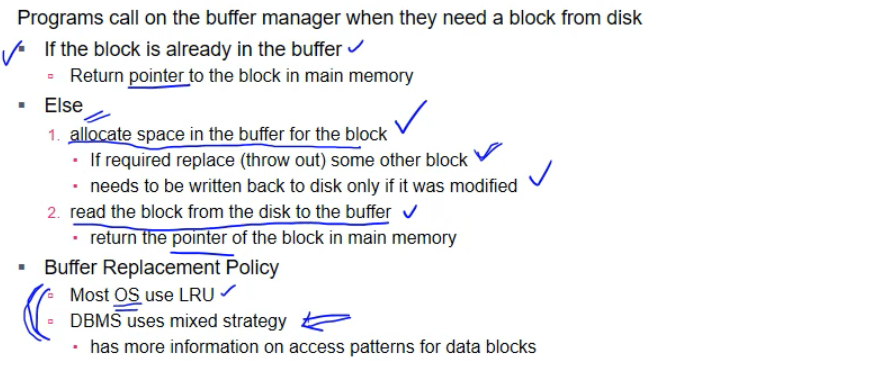
block들은 storage allocation과 data transfer 둘 다의 기준 unit임.

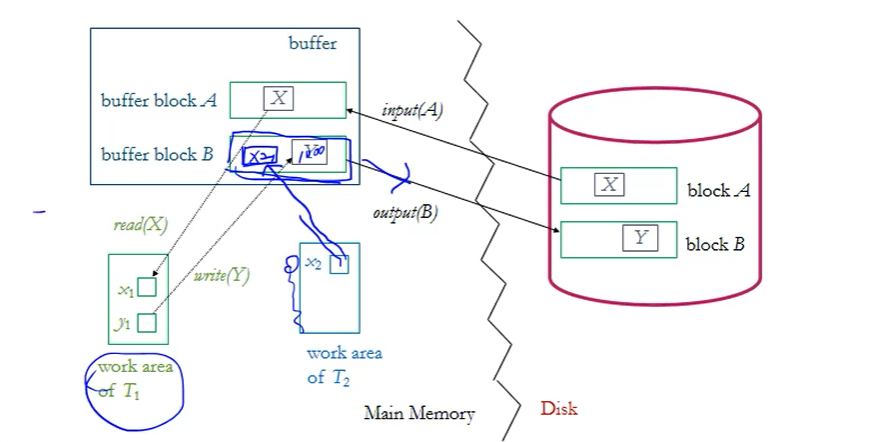
메모리와 디스크 사이의 Block transfer의 회수를 줄여야 함.

Buffer – 메인 메모리의 일정 부분을 disk block을 복사해놓고 사용

Buffer manager – 메인 메모리에 버퍼 공간을 할당하는걸 관리함.

**Buffer Manager**





프로그램 입장에서는 buffer나 디스크에 상관 없음. DBMS에서 버퍼와 디스크 관리.