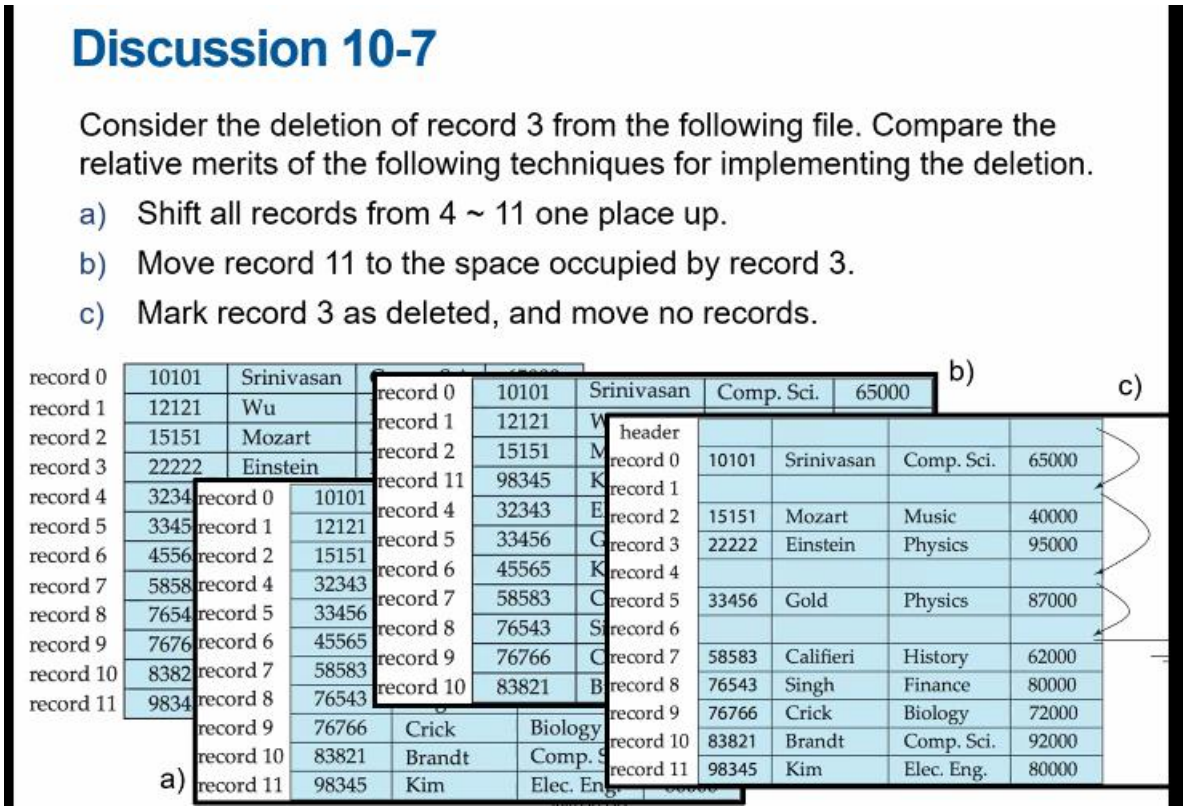
Discussion 05/18



a)의 장점은 sequential read 속도가 빠르다. Record 순서가 보장된다. Maximum available space 보장. 단점은 record 하나가 지워지면 그 밑의 모든 record 를 shift 해야 한다.

b)의 장점은 delete 의 overhead 가 줄어듦. Maximum available space 보장. 단점은 record 순서가 지켜지지 않음.

c)의 장점은 delete 나 insert 가 상수 시간. Record 순서 보장. 단점은 record 사이에 빈 공간이 생기는 것.

Discussion 10-8

A **sequential file** is designed for efficient processing of records in sorted order based on some search key. Ideally, all records should be stored in sorted order, but this is not always feasible after updates. Discuss how pointers can be used to support efficient sequential access employing the different techniques after deletion of record 3.

~~a) Shift all records from 4 ~ 11 one place up.~~

b) Move record 11 to the space occupied by record 3.

c) Mark record 3 as deleted, and move no records.

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

Header 에 시작과 끝을 저장하는 pointer 를 둔다. 시작 pointer 는 id 가 가장 작은 record, 끝 pointer 는 가장 큰 record 를 가리키게 하고, 각 record 마다도 포인터를 둬서 id 의 크기를 따라 다음 record 를 가리키게 한다. Insert 를 할 경우 id 값을 보고 pointer 끼리의 관계에서 맞는 곳에 pointer 를 insert 하고, Delete 가 되면 그냥 해당 record 의 pointer 도 delete 하고 앞 뒤 관계를 갱신해주면 된다.

이러한 방식으로 포인터를 사용하면 id 의 정렬된 순서를 보장할 수 있다.

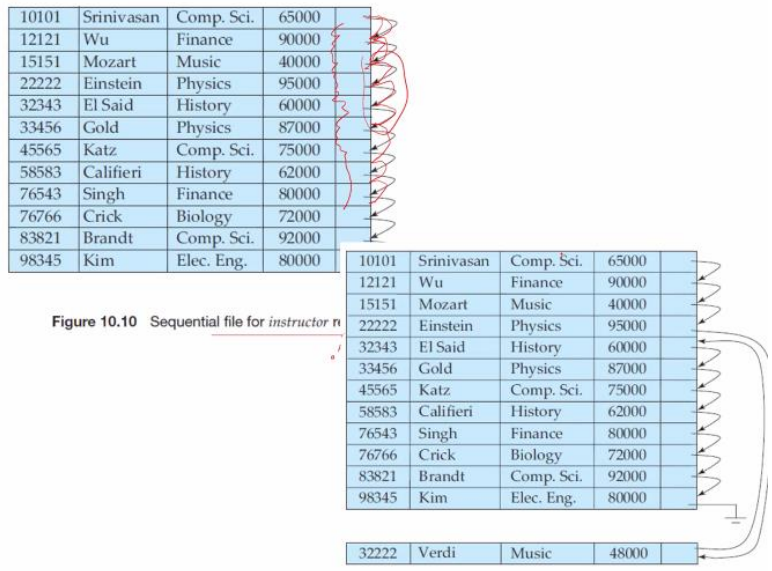


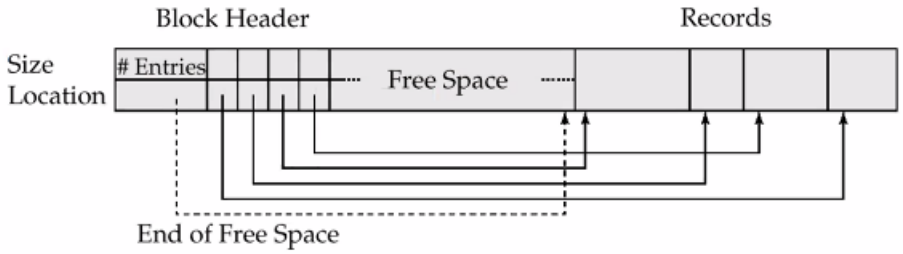
Figure 10.10 Sequential file for *instructor r*

=〉 이런 식으로 pointer 사용!

Discussion 10-9

Show the status of the block in **slotted page structure** is after performing the following operations in order. Assume the block size is 500B and empty initially.

1. insert record A (length 20B)
2. insert record B (length 40B)
3. insert record C (length 30B)
4. delete record B
5. insert record D (length 50B)



Entry 수가 3 개, block header 는 A, 빈 공간, C, D 의 위치와 크기 등 정보를 가지고 있음.

오른쪽 record 공간은 오른쪽에서부터 순서대로 A 20B, 빈 공간 40B (B 가 삭제됐기 때문에), C 30B, D 50B 가 저장되어 있음.

가운데에는 free space 가 있음.

=〉 delete 연산이 되면 블락 내부에서는 shift 를 함! Free space 를 한 덩어리로 크게 보장하는 것이 핵심. Record 를 shift 해서 빈 공간을 없애야 함. Header 는 shift 하지 않고 그냥 -1 등 삭제됐다는 표시만 해줌.

Discussion 10-10

A record in a disk is usually identified logically by < *fileID, block#, rec#*> rather than using the physical address <*track#, sector#, offset*>.

a. What may be the reason for this?

b. Describe the information that should be kept in a file header that keeps track of the blocks in the file.

a. 일관성을 유지하기 위해서. Logical address 를 사용하면 OS 가 하드웨어 구조에 따라 적절하게 physical 로 맵핑을 해주지만 physical address 를 사용하면 하드웨어에 따라 구조가 달라져야 함.

=〉 교수님 설명: 대형 DBMS 들은 physical address 도 관리할 수 있음. 근데도 쓰는 이유는 record 들이 옮겨질 수도 있고, record 의 size 가 변경될 수도 있음. 이렇게 데이터가 변경되는 순간 해당 데이터를 참조하고 있는 정보들의 위치가 모두 바뀌어야 함. Logical 로 관리하면 physical 을 바꿀 필요가 없음.

b. logical address 와 physical address 의 맵핑 정보가 있어야 함.

Discussion 10-11

Suppose transaction *T1* requests **write**(*X*) on a database item *X*, and the DBMS subsequently responds with a success message. Does this mean that *X* is updated in the disk? Explain.

그럴 수도 있고 아닐 수도 있다.

DBMS 에서 write 를 할 때 메인 메모리의 buffer 에 block 을 불러와서 한다. 그 후 다른 transaction 이 X 를 참조할 경우가 생기면 바로 disk 에 write 를 하지 않고 그 작업까지 buffer 의 block 에서 모두 끝낸 후 disk 에 update 를 한다.

=〉 디스크에 반영이 되지 않은 상태로 메모리에만 저장되어 있다가 문제가 생길 경우 recovery system 이 해결을 해줘야 한다고 한다. 16 장에서 배울 예정.