조금 더 빠른방법? => 인덱스!

인덱스 => 실제로 사용 사례 (책의 목차, 책의 맨 뒷부분 색인의 단어 옆 숫자(page))

이게 뭘 의미 하느냐…? 순차검색..(앞에서부터 읽는다(O(n))),

인덱스 => 레코드를 차원이 다르게 빠르게 찾을 수 있도록 도와준다.

또다른 방법의 엑세스를 제공, 파일의 또다른 접근 방법.

인덱스 엔트리 => 인덱스 파일의 레코드

인덱스 엔트리의 구조 => 키 값들, 레코드 주소

-INDEX FILE

CLUSTERING FIELD VALUE(순차적인 값) :: BLOCK POINTER(DEPTNUMBER의 위치를 가리킨다)

# 인덱스를 이용한 Search Algorithm :

=>나의 의견 : 인덱스 Value를 고르면 그 옆 Block Pointer를 통해 원하는 값이 들어있는 곳이 DeptNumber의 몇 번째에 위치 할지를 알 수 있을 것이다.

=>남의 의견 : 찾고자 하는 키 값의 첫번째를 가리키고 있는 것.

우리가 찾고자 하는 것 : 해당 필드의 알맞은 값이 들어있는 레코드.

=>교수님 : 인덱스 엔트리의 주소 부분(Block Pointer)이 가리키고 있는 곳은 해당 레코드의 첫번째 필드의 주소이다. 그리고…

인덱스 파일에 속한 레코드 => 인덱스 엔트리

인덱스 엔트리를 순차 검색하는 것이다!

Block Pointer는 레코드!

주소값의 데이터를 읽어서 순차 검색하여 찾는다.

결론(정리) =>Index File에서 내가 찾고자 하는 키값을 가지고 있는 인덱스 엔트리를 순차 검색하여 찾는다. 그 인덱스 엔트리를 이용하여 Data File에서의 시작 부분으로 간다. 그 레코드부터 순차 검색하여 찾는다. 찾는 도중 DeptNumber의 값보다 큰 값이 나오면 그만 찾는다. Disk Access는 최소 2번 많으면 3번 (1.Index File => 2.Data File => 3.다음 DEPTNUMBER가 있는지 확인)

* 그래서.. 인덱스가 있는게 기존 서치 알고리즘 보다 빠를까??
* 내 생각 : 빠르다고 생각한다. 처음부터 찾는 서치 알고리즘과 다르게 인덱스는 인덱스 엔트리를 통해 그 시작 위치를 미리 알 수 있기 때문에 그 위치부터 순차적으로 찾을 것이다.
* 실생활에서만 봐도 인덱스를 사용하는 것이 전보다는 훨씬 빠르다.

(첫번째 블록을 찾는 것이 아니라면 전보다는 훨씬 빠를 것이다. 다만 Disk Access를 최소한은 두 번 이상 해야 한다.)

# 인덱스를 이용한 Insert Algorithm :

* 기존 insert 알고리즘에서 검색은 인덱스를 적용해서 기존보다 빠를 것이고 빈공간이 있는 경우 그 곳에 넣고 끝, 하지만 빈공간이 없으면 새로운 블록을 할당 받아서 뒤에 값들을 블록에 넣고 다음 블록과 이어준다. 그리고 그 인덱스 엔트리에 대한 인덱스 파일에서의 Insert 또한 같은 알고리즘으로 한다.

# 인덱스를 이용한 Delete Algorithm :

* 인덱스를 이용하여 찾는다. 찾고나서 그 필드를 삭제한다. 유일한 엔트리 포인터이면 엔트리 포인터를 삭제 해준다. Disk Access가 더 늘어 날 수는 있지만 인덱스를 이용한 것이 더 빠르지 않을까? 추측
* 근데…. 인덱스 구조는 쓰지 않는다.. 그냥 이해만 쉽게 해주도록 하는 것이다.