Databases

화일의 인덱스 구조

한국공학대학교
게임공학과
장 지 웅

Contents

화일(file)

화일

이재영 2012096138 031-8041-0551 AI
장지웅 2013081194 031-8041-0554 DB
윤정현 2013083029 031-8041-0552 C++

•

•

•

필

레코드의 특징

이재영	2012096138	031-8041-0551	AI
장지웅	2013081194	031-8041-0554	DB
윤정현	2013083029	031-8041-0552	C++
		.	
		.	
		.	

필드의 개수는 동일

각 필드의 크기는 고정

모든 레코드의 크기는 동일

기억을 되살려 봅시다.

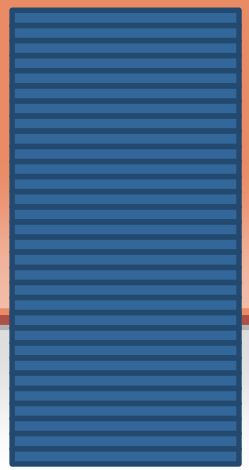
자료구조에서 성능을 결정하는 가장 중요한 요소는 무엇이었나?

- 시간복잡도 측면에서?
- 공간복잡도 측면에서?

생각해 봅시다.

?

배열과 같은 구조로 데이터 화일을
만들 수 있을까?
문제점은 무엇인가?



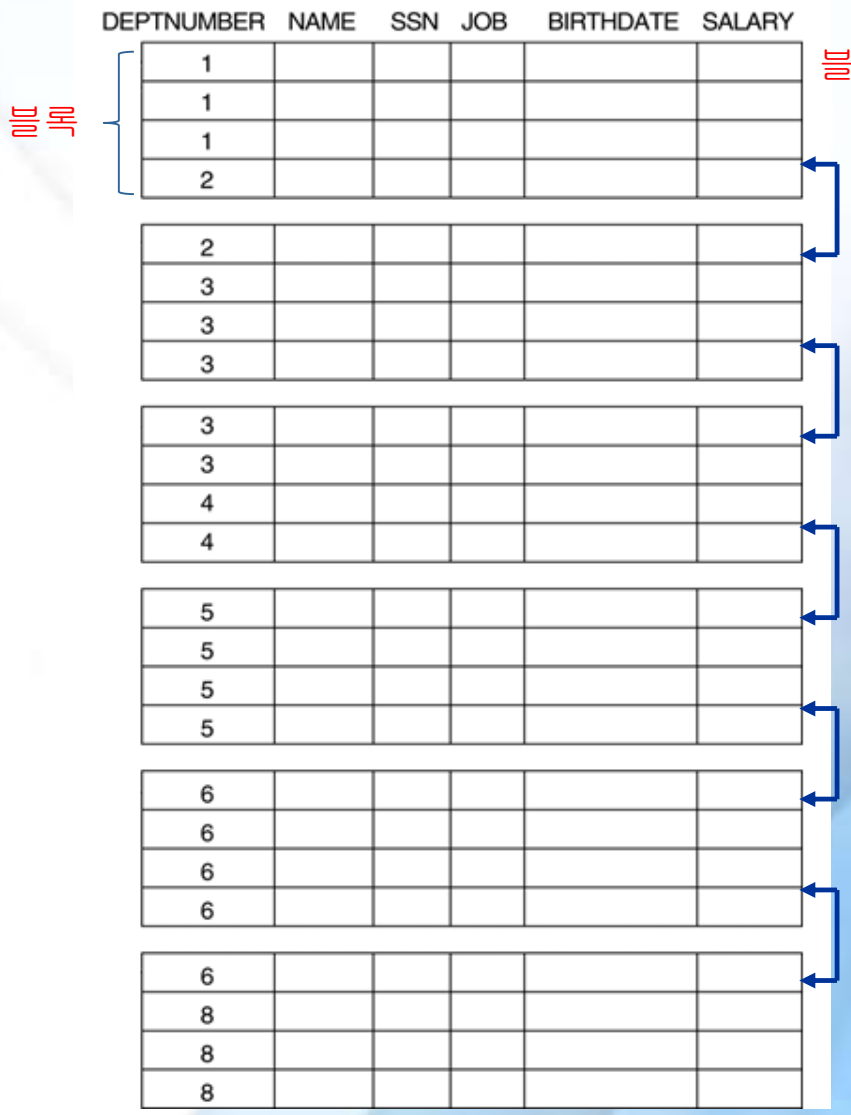
생각해 봅시다.

연결리스트 구조로 데이터 화일을
만들 수 있을까?
문제점은 무엇인가?

생각해 봅시다.

배열과 연결리스트의 단점을 보완할 수 있는 새로운 자료구조를 생각해 보자

데이터 화일의 구조



문제가 해결되었는가?
- 레코드의 추가 삽입
- 하드디스크 액세스 횟수

블록의 크기는 어느 정도가 적당한가?

생각해 봅시다.

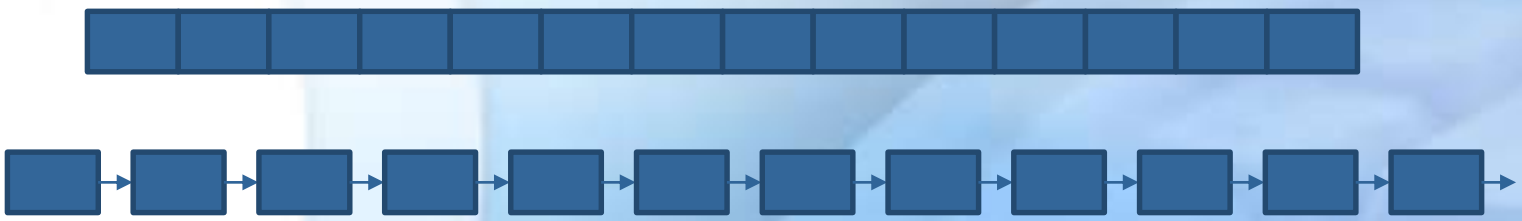
데이타 화일에서 원하는 레코드를 찾는 방법은 무엇인가?
이 때, 성능에 영향을 미치는 요인은 무엇인가?

기억나시나요?

순차검색 VS 이진검색

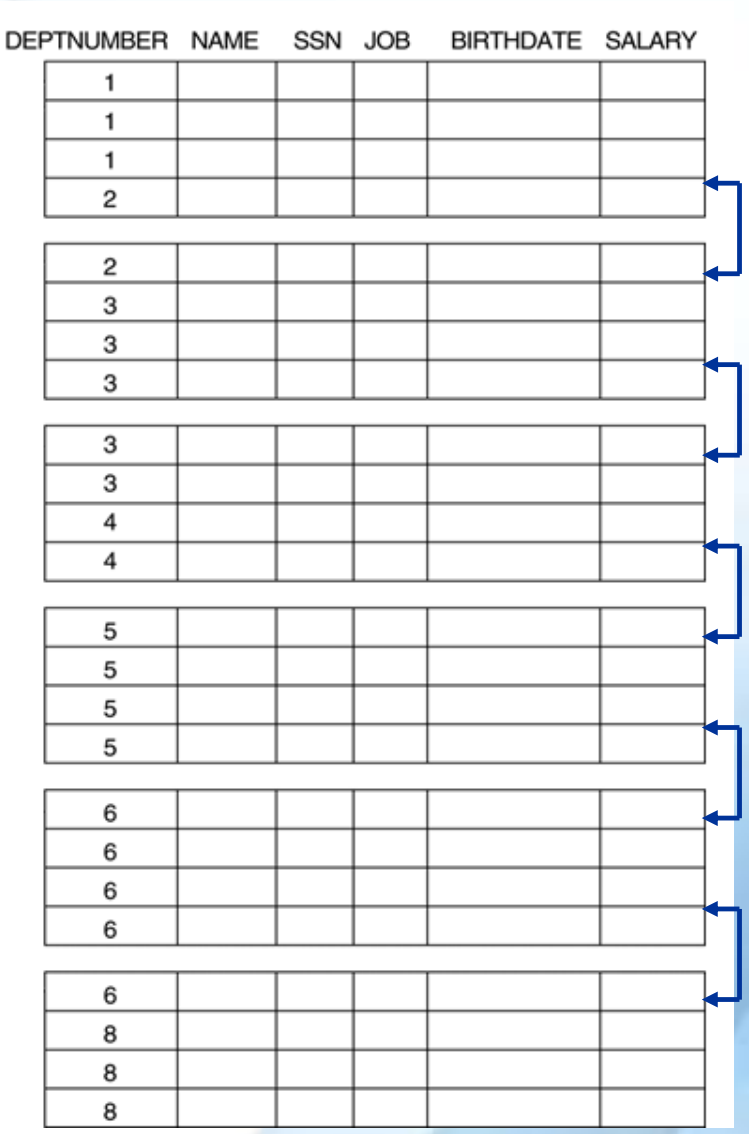
시간복잡도?

이진검색의 조건은?



순서화일에서의 검색

순서화일이란?
특정 필드 값의 크기 순으로 레코드를 저장한 화일



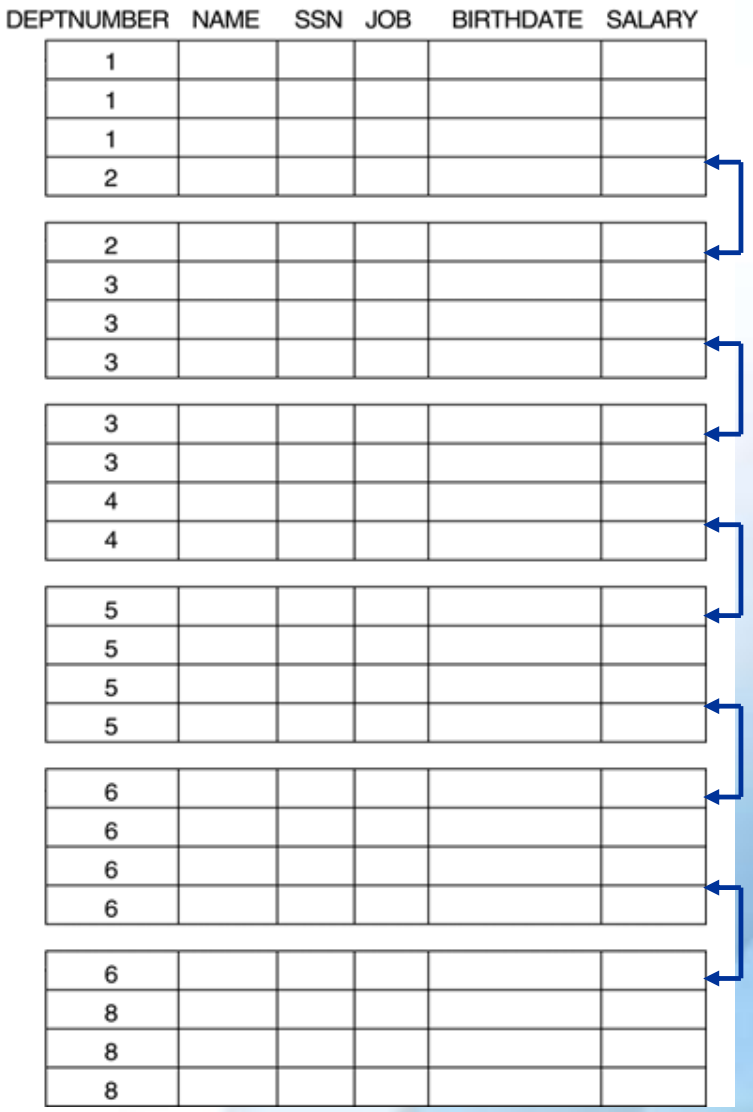
순차검색?

이진검색?

생각해 봅시다.

데이타 화일에 새로운 레코드를 삽입하는 방법은 무엇인가?
이 때, 성능에 영향을 미치는 요인은 무엇인가?

순서화일에서의 삽입



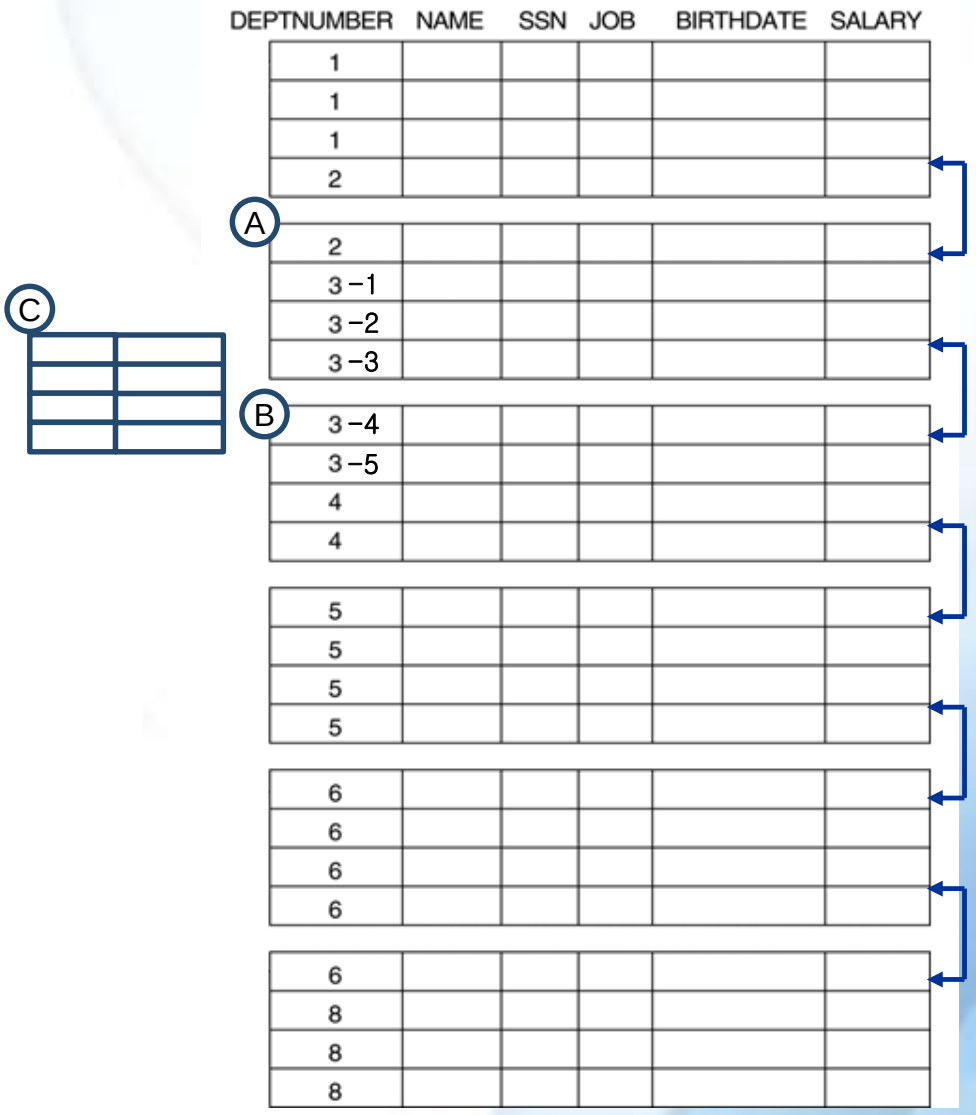
1. 삽입할 자리를 찾는다.
2. 삽입할 공간을 만든다.
3. 삽입한다.

삽입할 공간이 없으면
어떻게 만드는가?

순서화일에서의 삽입

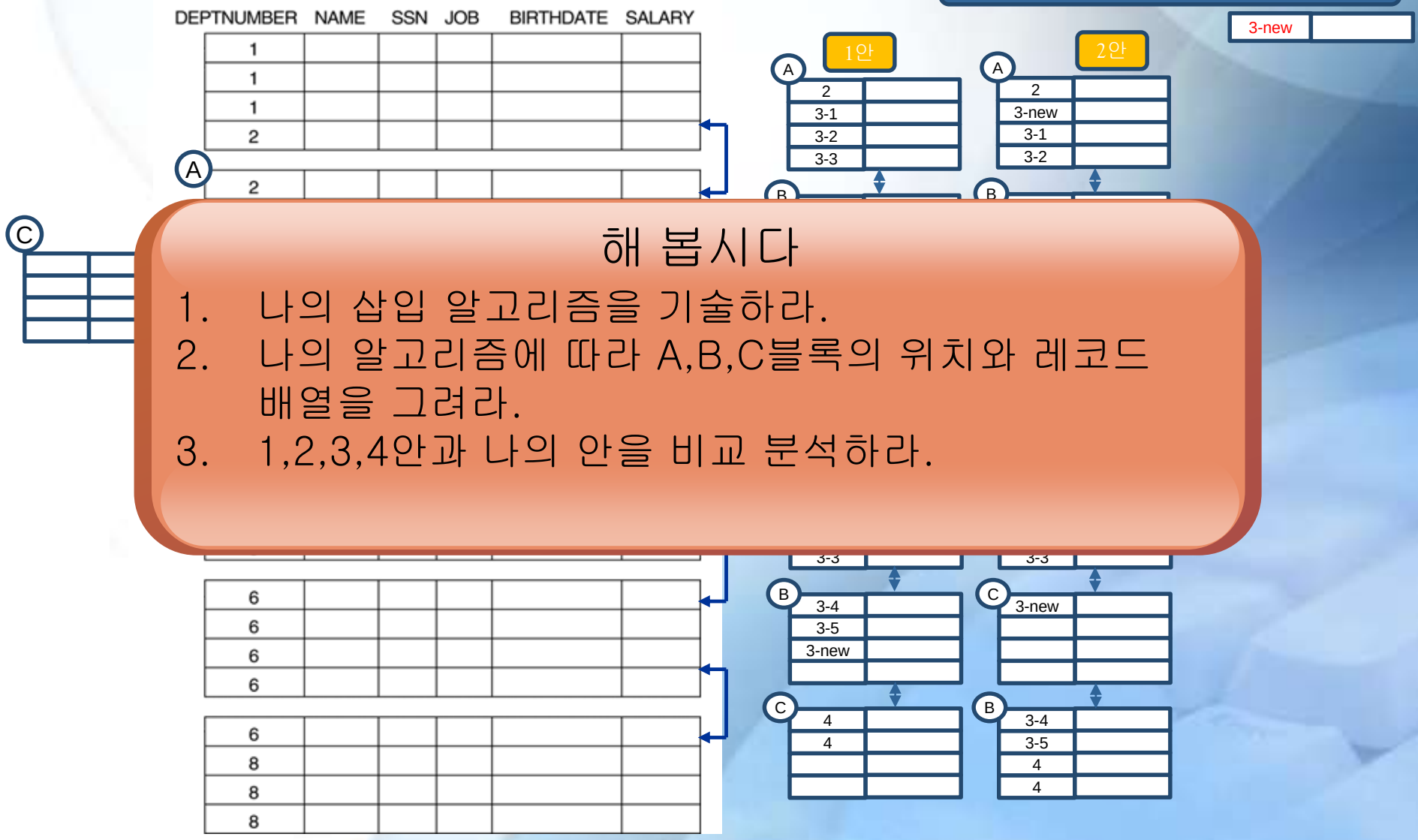
새로운 블록에 넣을 레코드는?

3-new



순서화일에서의 삽입

새로운 블록에 넣을 레코드는?

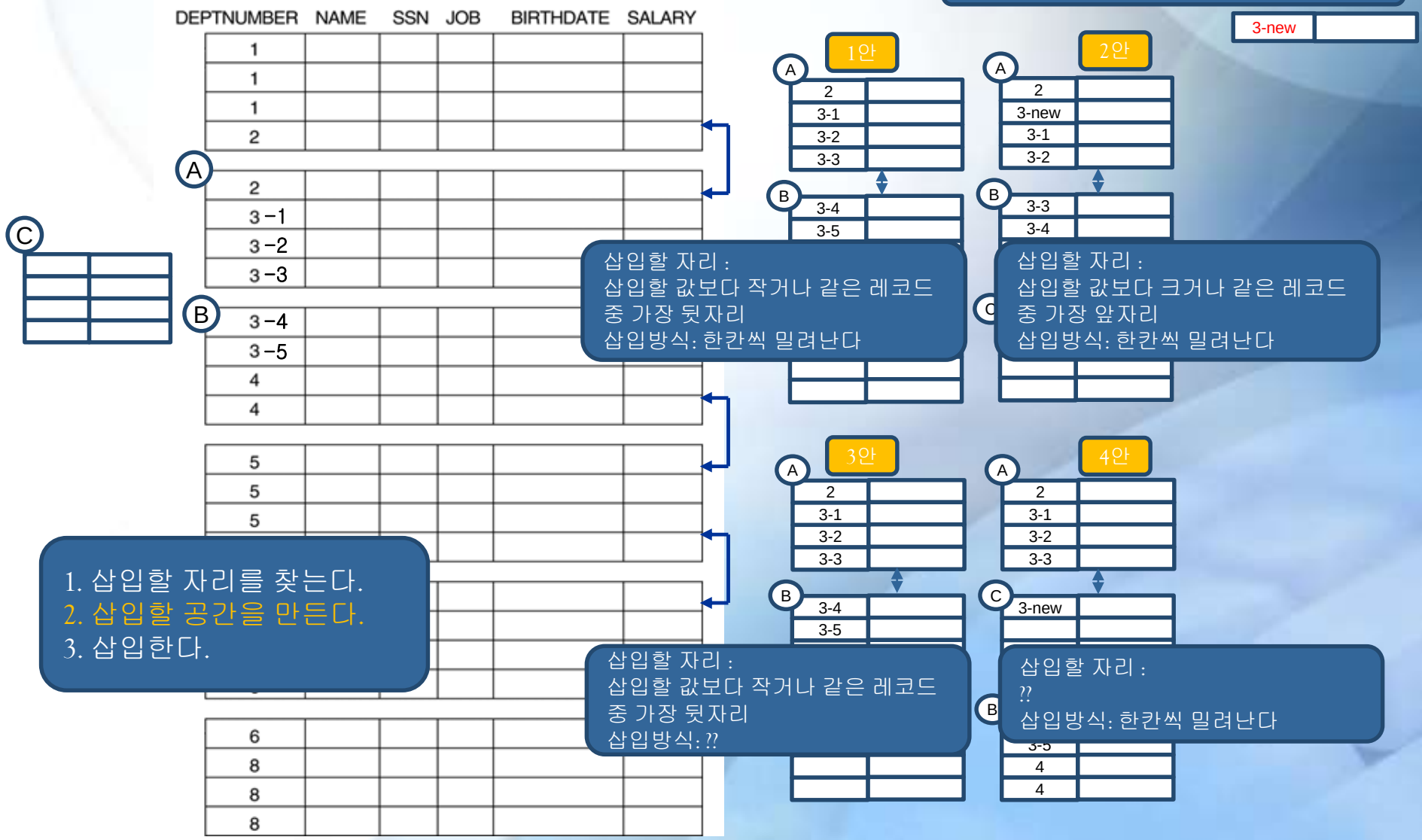


해 봅시다

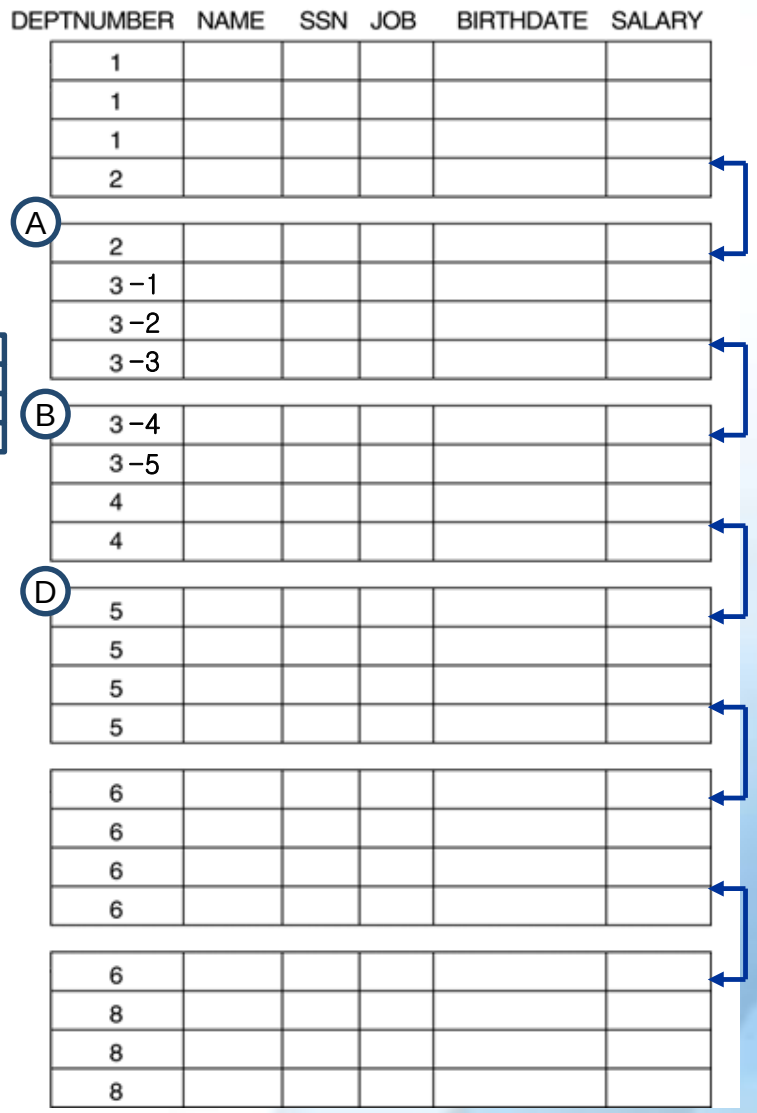
1. 나의 삽입 알고리즘을 기술하라.
2. 나의 알고리즘에 따라 A,B,C블록의 위치와 레코드 배열을 그려라.
3. 1,2,3,4안과 나의 안을 비교 분석하라.

순서화일에서의 삽입

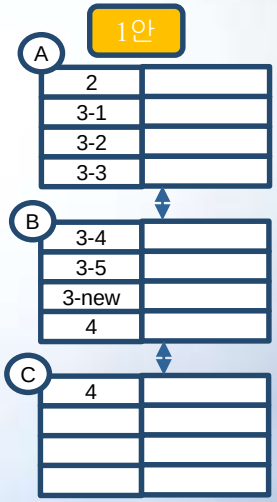
새로운 블록에 넣을 레코드



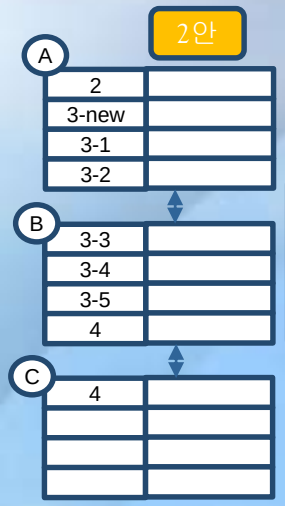
순서화일에서의 삽입안 분석1



디스크 액세스 횟수는?

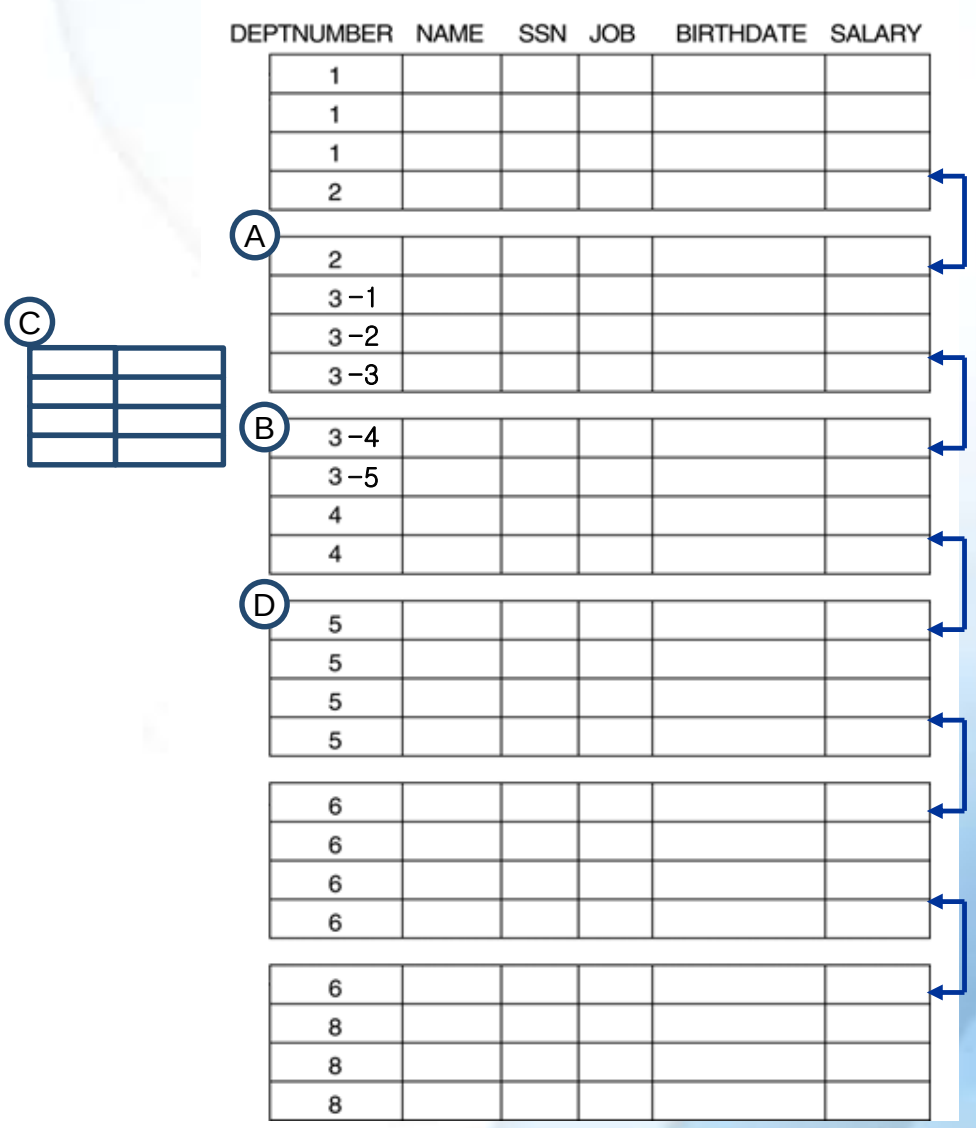


총 5회
삽입할 위치 검색 - 3회
B블록 수정(레코드, 링크) - 0회(위에 포함)
C 블록 수정 (레코드, 링크) - 1회
D블록 링크 수정 - 1회

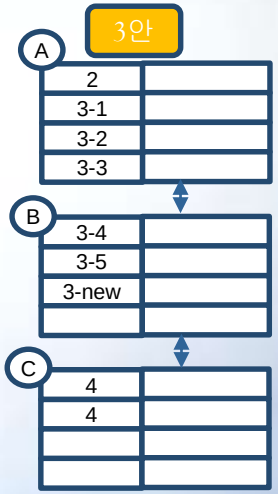


총 5회
삽입할 위치 검색 - 2회
A블록 수정 - 0회(위에 포함)
B블록 수정(레코드, 링크) - 1회
C 블록 수정 (레코드, 링크) - 1회
D블록 링크 수정 - 1회

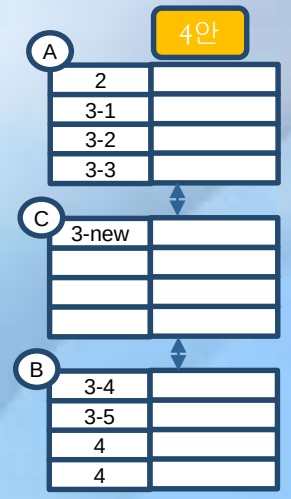
순서화일에서의 삽입분석 2



디스크 액세스 횟수는?



총 5회
삽입할 위치 검색 - 3회
B블록 수정(레코드, 링크) - 0회(위에 포함)
C 블록 수정 (레코드, 링크) - 1회
D블록 링크 수정 - 1회



총 4회
삽입할 위치 검색 - 2회
A블록 수정 - 0회(위에 포함)
B블록 수정(레코드, 링크) - 1회
C 블록 수정 (레코드, 링크) - 1회
D블록 링크 수정 - 0회

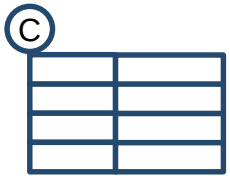
순서화일에서의 삽입분석 3

이어서 삽입할 레코드는?

3-new2

DEPTNUMBER	NAME	SSN	JOB	BIRTHDATE	SALARY
1					
1					
1					
2					
A					
2					
3 -1					
3 -2					
3 -3					
B					
3 -4					
3 -5					
4					
4					
D					

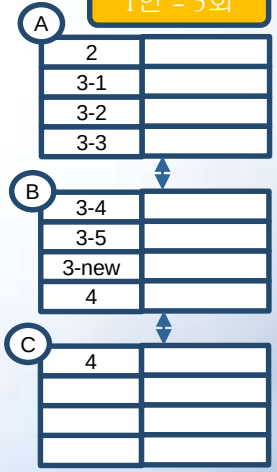
C



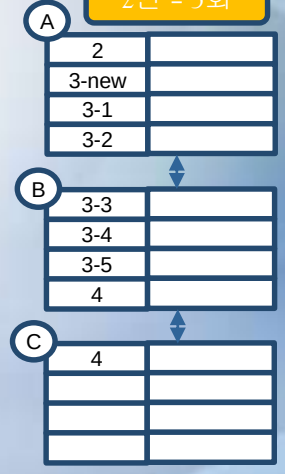
삽입 알고리즘
1. 삽입할 자리를 찾는다.
2. 삽입할 공간을 만든다.
3. 삽입한다.

삽입 알고리즘 수정
1. 삽입할 자리를 찾는다.
2. 공간이 있으면 삽입하고 종료
3. 공간이 없으면 삽입할 공간을 만든 후 삽입하고 종료

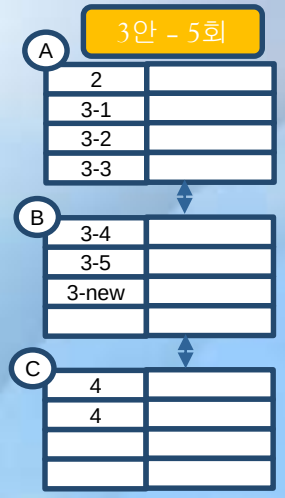
1안 - 5회



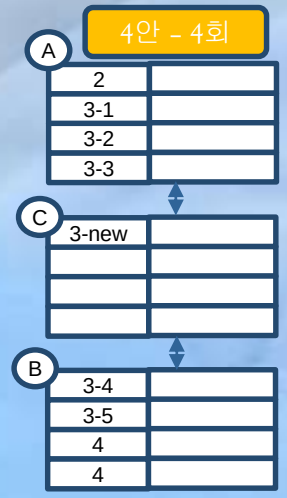
2안 - 5회



3안 - 5회



4안 - 4회

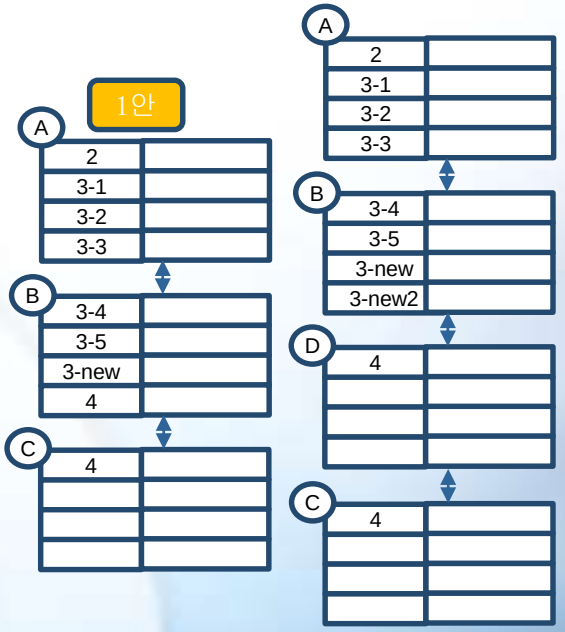


순서화일에서의 연속삽입1

DEPTNUMBER	NAME	SSN	JOB	BIRTHDATE	SALARY
1					
1					
1					
2					
2					
3-1					
3-2					
3-3					
3-4					
3-5					
4					
4					
5					
5					
5					
5					
6					
6					
6					
8					
8					

1. 삽입할 자리를 찾는다.
2. 공간이 있으면 삽입하고 종료
3. 공간이 없으면 삽입할 공간을 만든 후 삽입하고 종료

1안

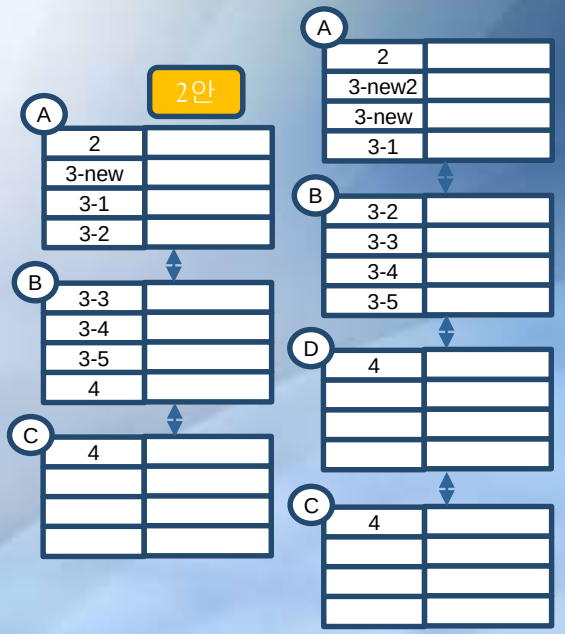


삽입할 자리 :
삽입할 값보다 작거나 같은 레코드 중 가장 뒷자리
삽입방식: 한칸씩 밀려난다

이어서 삽입할 레코드는?

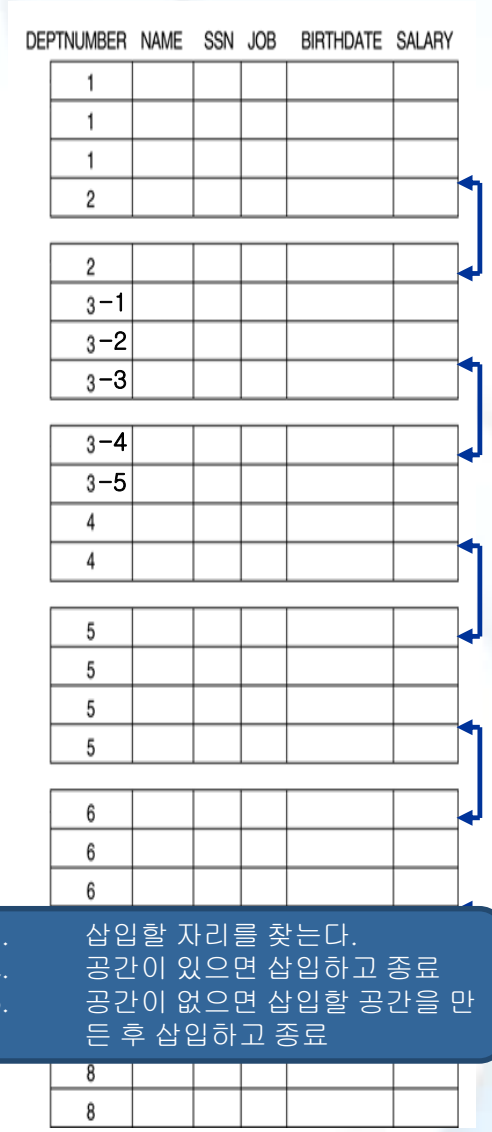
3-new2	

2안

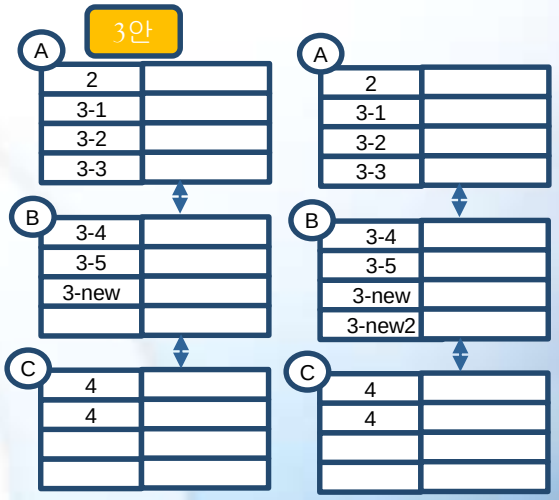


삽입할 자리 :
삽입할 값보다 크거나 같은 레코드 중 가장 앞자리
삽입방식: 한칸씩 밀려난다

순서화일에서의 연속삽입2



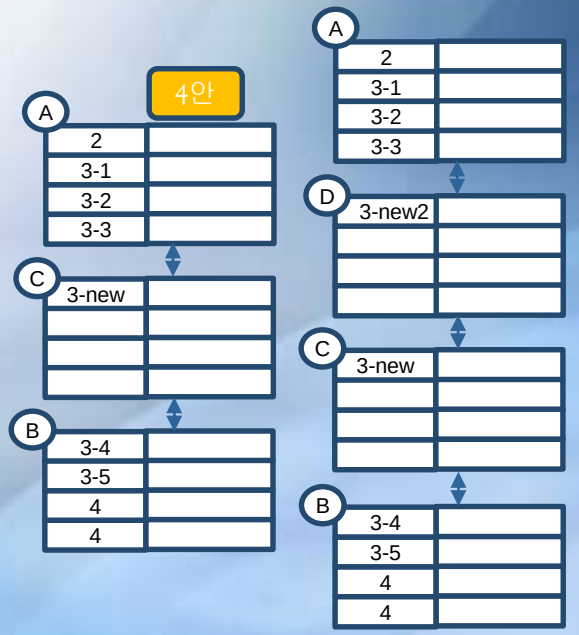
1. 삽입할 자리를 찾는다.
2. 공간이 있으면 삽입하고 종료
3. 공간이 없으면 삽입할 공간을 만든 후 삽입하고 종료



삽입할 자리 :
삽입할 값보다 작거나 같은 레코드 중 가장 뒷자리
삽입방식: ??

이어서 삽입할 레코드는?

3-new2



삽입할 자리 :
??
삽입방식: 한칸씩 밀려난다

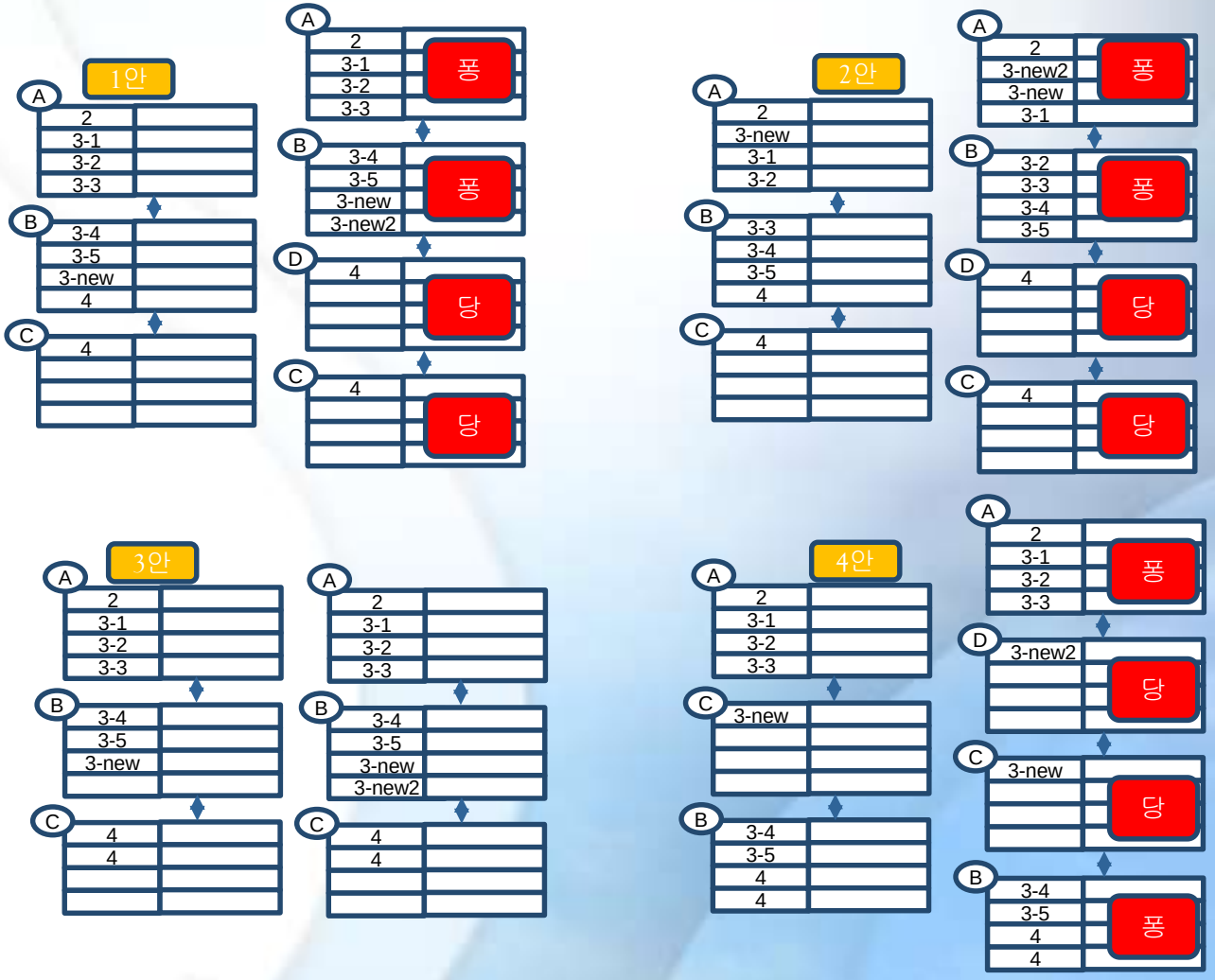
생각해봅시다

연속적인 삽입이 발생하는 경우
어떠한 일이 벌어지는가?

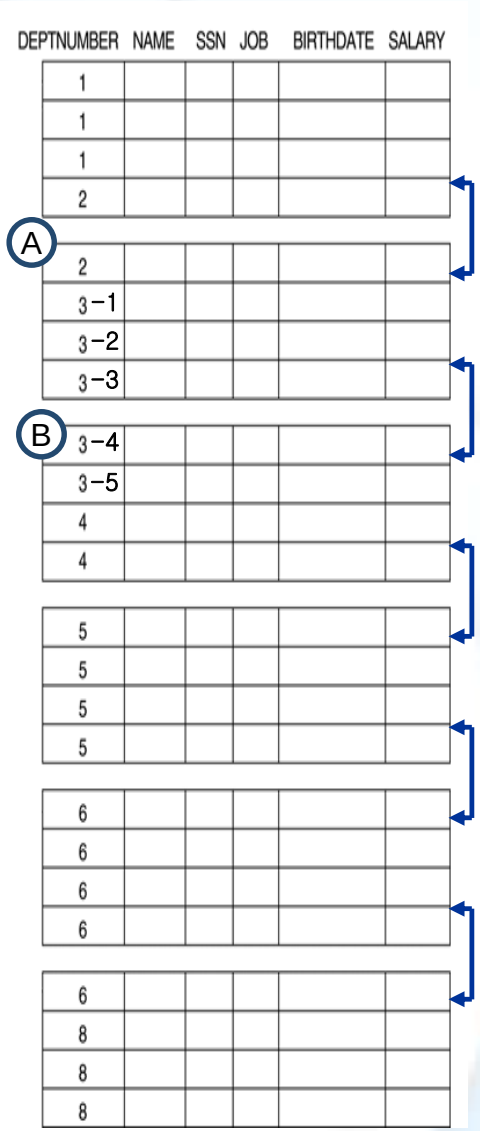
순서화일에서의 연속삽입

1. 삽입할 자리를 찾는다.
2. 공간이 있으면 삽입하고 종료
3. 공간이 없으면 삽입할 공간을 만든 후 삽입하고 종료

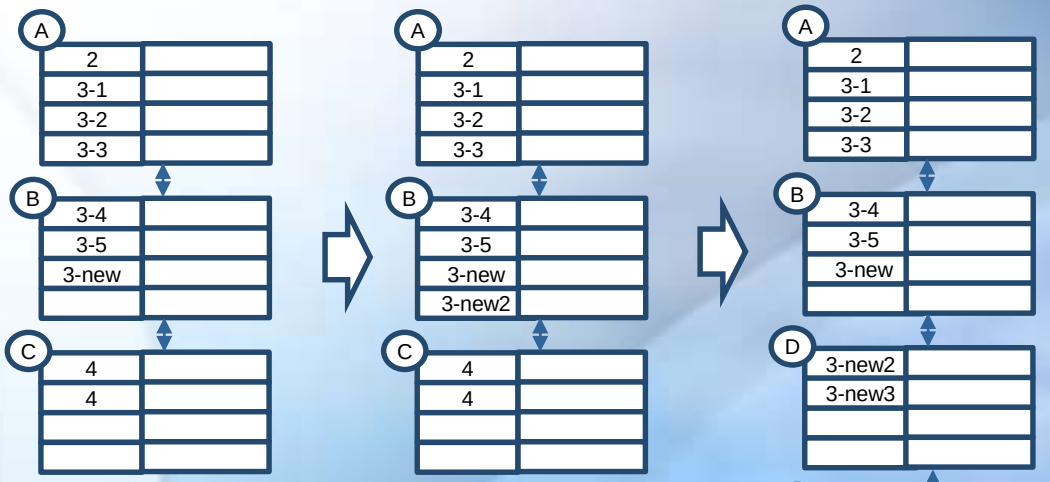
문제점
1. 저장 공간의 낭비 – 레코드 1개인 블록 증가
2. 성능 저하 – 블록의 개수 증가
   ➔ 3안에서 힌트를 얻자



순서화일에서의 삽입 다시



1. 삽입할 자리를 찾는다.
2. 찾은 블록(A)에 공간이 있으면 삽입하고 종료
3. A에 공간이 없으면
   - 3-1. 새로운 블록(B)을 할당하여 A뒤에 연결한다.
   - 3-2. A에 저장되어 있던 레코드와 삽입할 레코드를 A와 B에 절반씩 나누어 저장하고 종료한다.

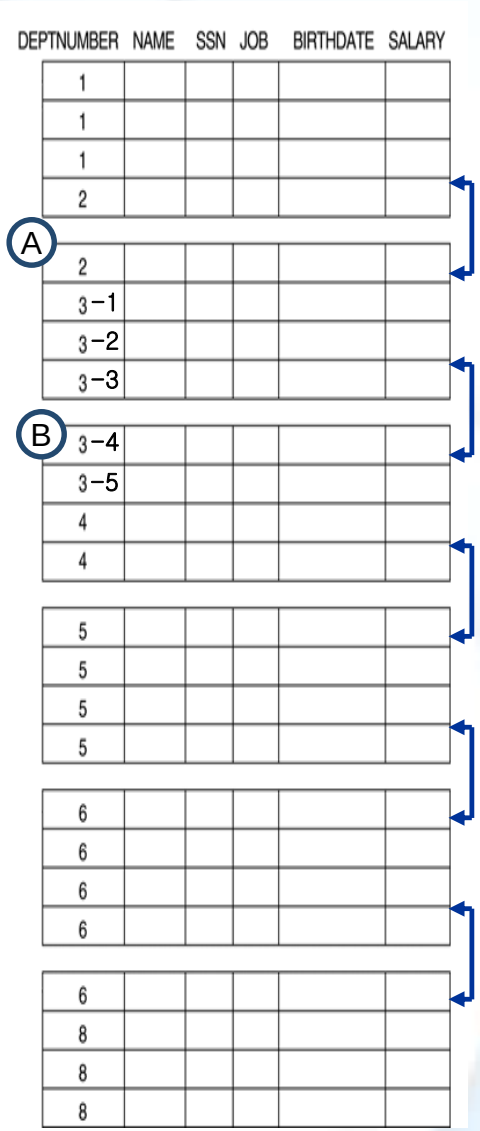


문제점은 무엇인가?

정리해봅시다

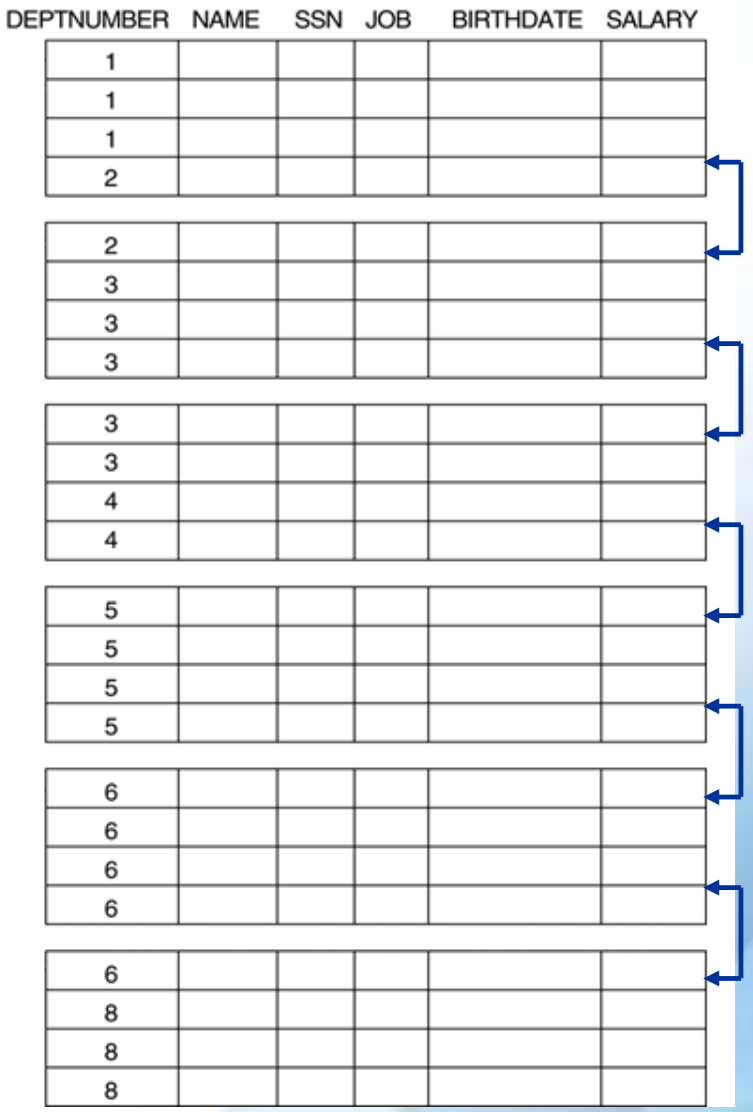
순서파일에서 검색 및 삽입
알고리즘을 정리해봅시다.

순서화일에서의 삽입



1. 삽입할 자리를 찾는다.
2. 찾은 블록(A)에 공간이 있으면 삽입하고 종료
3. A에 공간이 없으면
   - 3-1. 새로운 블록(B)을 할당하여 A뒤에 연결한다.
   - 3-2. A에 저장되어 있던 레코드와 삽입할 레코드를 A와 B에 절반씩 나누어 저장하고 종료한다.

순서화일에서의 삭제

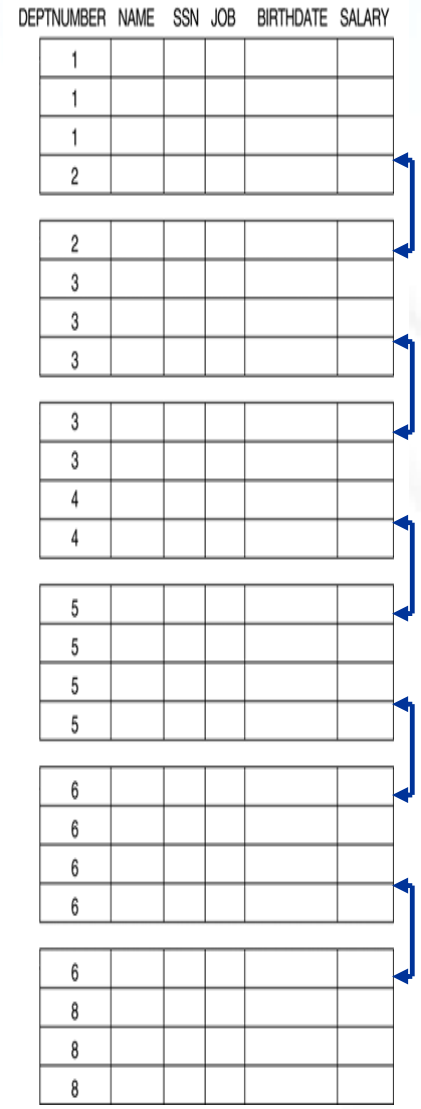


1. 삭제할 레코드를 찾는다.
2. 삭제한다.
3. 삭제한 빈 공간을 처리한다.

삭제한 빈 공간을 어떻게 처리할 것인가?

순서화일에서의 삭제

1. 삭제할 레코드를 찾는다.
2. 삭제한다.
3. 삭제한 빈 공간을 처리한다.

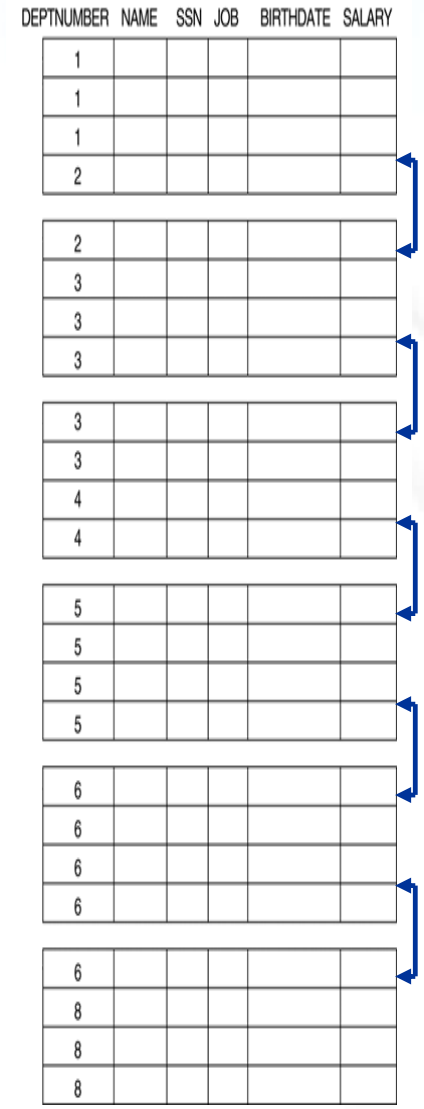


삭제한 빈 공간을 언제, 어떻게 처리할 것인가?

1. 그냥 둔다. 즉, 처리하지 않는다.
2. 블록(A) 내 레코드 개수가 0이 되면 블록을 삭제한다.
3. 블록(A) 내 레코드 개수가 X가 되면 앞뒤블록과 합친다.

순서화일에서의 삭제

1. 삭제할 레코드를 찾는다.
2. 삭제한다.
3. 삭제한 빈 공간을 처리한다.



삭제한 빈 공간을 언제, 어떻게 처리할 것인가?

1. 그냥 둔다. 즉, 처리하지 않는다.

2. 블록(A) 내 레코드 개수가 0이 되면 블록을 삭제한다.

3. 블록(A) 내 레코드 개수가 X가 되면 앞뒤블록과 합친다.

블록 사용률(a)이 50% 이하가 되면
3-1. 다음 블록(B)을 읽어 블록사용률(b)를 계산한다.
3-2. a+b <= 100%이면 합친다.
3-3. a+b > 100%이면 B에서 A로 일부 레코드를
이동하여 A,B 모두 50% 이상이 되도록 한다.

삽입, 삭제 정리해봅시다

삽입

1. 삽입할 자리를 찾는다.
2. 찾은 블록(A)에 공간이 있으면 삽입하고 종료
3. A에 공간이 없으면
   3-1. 새로운 블록(B)을 할당하여 A뒤에 연결한다.
   3-2. A에 저장되어 있던 레코드와 삽입할 레코드를 A와 B에 절반씩 나누어 저장하고 종료한다.

삭제

1. 삭제할 레코드를 찾는다. 찾은 블록을 A라한다.
2. 삭제한다.
3. 블록 사용률(a)이 50% 이하가 되면
   3-1. 다음 블록(B)을 읽어 블록사용률(b)를 계산한다.
   3-2. a+b <= 100%이면 합친다.
   3-3. a+b > 100%이면 B에서 A로 일부 레코드를 이동하여 A,B 모두 50% 이상이 되도록 한다.

이 방법의 장단점을 분석해보자.

저장공간 사용량의 관점에서?

하드디스크 액세스 횟수의 관점에서?