



인택싱



컴퓨터과학과 정재화

학습목차

— ① 인덱스의 이해

— ② 순서 인덱스

— ③ B⁺-트리 인덱스



01

인덱스의 이해

- 인덱스의 개념
- 인덱스 기반의 검색 과정
- 인덱스의 종류

일반적인 데이터 검색 과정



'COM44'의
과목명은?

'클라우드 컴퓨팅'

SQL



DBMS

COM11	컴퓨터의 이해	...
COM12	파이썬 프로그래밍 기초	...
COM31	데이터베이스 시스템	...
COM34	알고리즘	...
COM44	클라우드 컴퓨팅	...
ECE24	놀이지도	...
ECE31	유아언어교육	...
HE14	패션과 문화	...


디스크

COM34	알고리즘	...
COM44	클라우드 컴퓨팅	...
ECE24	놀이지도	...

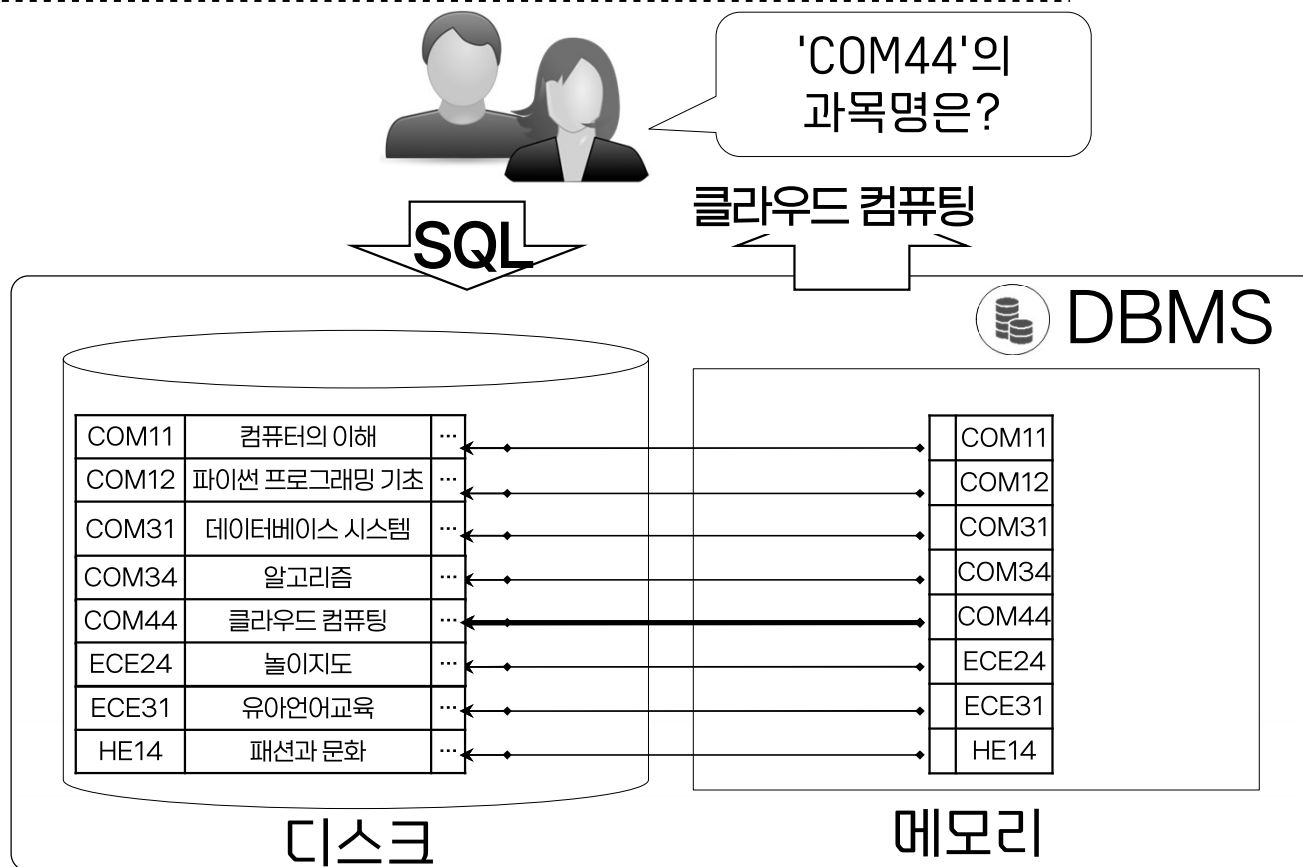
메모리

인덱스의 개념

- ▷ 데이터 검색 시 발생하는 비효율적인 데이터 입출력 문제를 해결하기 위한 목적으로 시작
 - + 인덱스: 요청된 레코드에 빠르게 접근할 수 있도록 지원하는 데이터와 관련된 추가적인 구조
 - + 인덱싱: 인덱스를 구성하고 생성하는 작업
- ▷ 인덱스의 탐색키를 이용하여 해당 레코드가 저장된 블록을 위치를 파악하고 해당 블록을 빠르게 적재

 탐색키(검색키) _____
파일에서 레코드를 찾는데 사용되는
컬럼이나 컬럼의 집합

인덱스 기반의 데이터 검색 과정



인덱스의 종류와 평가기준

▷ 인덱스의 종류

- + 순서 인덱스: 특정 값에 대해 정렬된 순서 구조
- + 해시 인덱스: 버킷의 범위 안에서 값의 균일한 분포에 기초한 구조로 해시 함수가 어떤 값이 어느 버킷에 할당되는지 결정

▷ 인덱스의 평가기준

- + 접근 시간: 데이터를 찾는 데 소요되는 시간
- + 유지 비용: 새로운 데이터 삽입 및 기존 데이터 삭제 연산으로 인한 인덱스 구조 갱신 비용
- + 공간 비용: 인덱스 구조에 의해 사용되는 추가적인 공간 비용

02

순서 인덱스

- 순서 인덱스의 특징
- 인덱스의 구성
- 밀집·희소·다단계 인덱스



순서 인덱스의 특징

- ▷ 탐색키로 정렬된 순차 파일에 대하여 레코드에 대한 빠른 접근이 가능하도록 구성한 인덱스
 - ⊕ 탐색키를 정렬하여 해당 탐색키와 탐색키에 대한 레코드와의 연계를 통하여 인덱스 생성
 - ⊕ 순서 인덱스의 종류
 - 밀집 인덱스
 - 희소 인덱스
 - 다단계 인덱스

☆ 순차 파일 구조

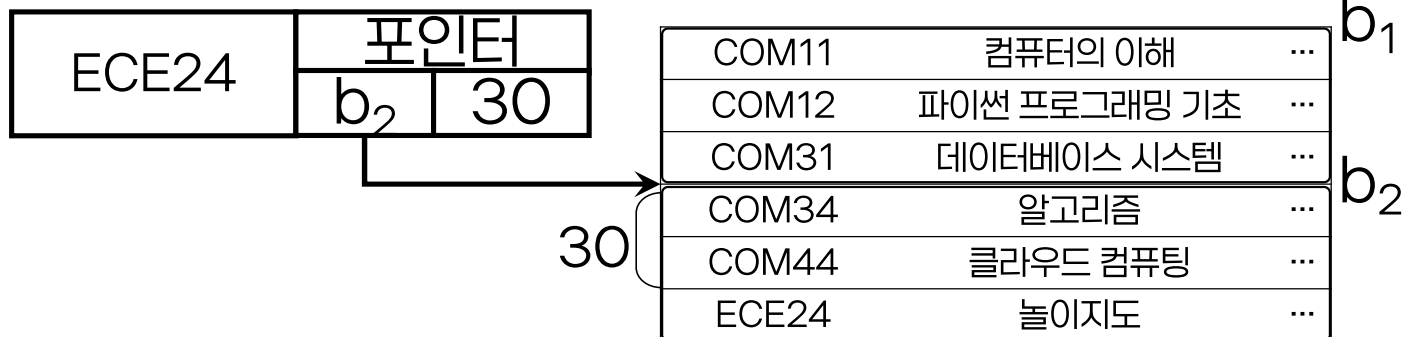
과목코드	과목명	학점	선수과목	이수구분	교수번호
COM11	컴퓨터의 이해	3		교양	194634-810228
COM12	파이썬 프로그래밍 기초	3		교양	194834-760517
COM24	자료구조	3	COM12	전공필수	194634-810228
COM31	데이터베이스 시스템	3	COM24	전공필수	194834-760517
COM34	알고리즘	3	COM24	일반선택	194634-810228
COM44	클라우드 컴퓨팅	3		전공필수	194834-760517
ECE24	놀이지도	3		전공필수	210315-549413
○ ○ ○					
LAW21	세법개론	3		전공필수	189414-790829
PA05	행정학개론	3		일반선택	191924-730620

인덱스 엔트리

인덱스 엔트리 구조

탐색키 값	포인터	
	블록 ID	오프셋

인덱스 엔트리 구성



⋮

밀집 인덱스 (dense index)


모든 레코드에 대해

탐색키 값	포인터
-------	-----

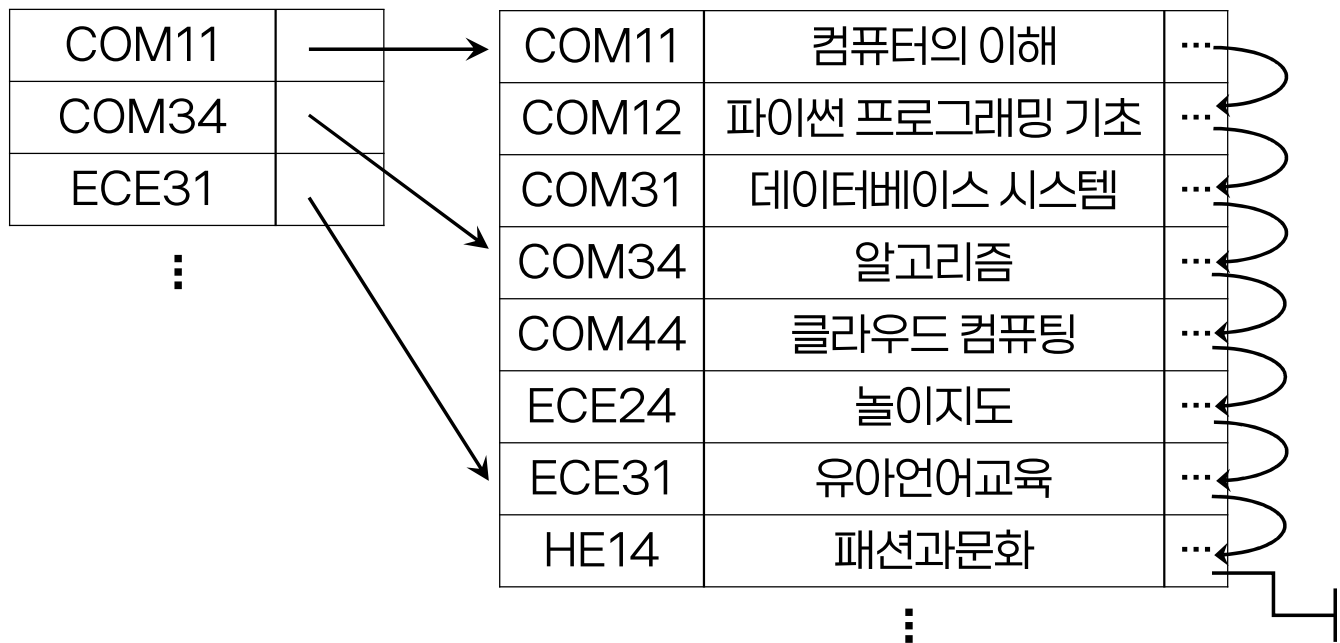
쌍을 유지

COM11	→	COM11	컴퓨터의 이해	...
COM12	→	COM12	파이썬 프로그래밍 기초	...
COM31	→	COM31	데이터베이스 시스템	...
COM34	→	COM34	알고리즘	...
COM44	→	COM44	클라우드 컴퓨팅	...
ECE24	→	ECE24	놀이지도	...
ECE31	→	ECE31	유아언어교육	...
HE14	→	HE14	패션과 문화	...
⋮		⋮		⋮

☆ 희소 인덱스 (sparse index)

▷ 인덱스의 엔트리가 일부의 탐색키 값만을 유지

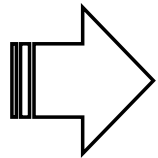
- ⊕ 요청된 탐색키보다 작거나 같은 인덱스의 탐색키 값 중 가장 큰 인덱스 엔트리의 포인터가 가리키는 블록을 스캔





다단계 인덱스의 필요

- ▷ 4KB 크기의 한 블록에 100개의 엔트리가 삽입될 때,
100,000,000 개의 레코드에 대한 밀집 인덱스
 - ⊕ 1,000,000개의 블록 = 4GB의 공간 필요
- ▷ 인덱스 크기에 따른 검색 성능
 - ⊕ 인덱스 크기 > 메모리 크기
 - 저장된 블록을 여러 번 나누어 읽어야 하기 때문에
디스크 I/O 비용이 증가하여 탐색 시간이 증가
 - ⊕ 인덱스 크기 < 메모리 크기
 - 디스크 I/O가 줄어 탐색 시간이 축소

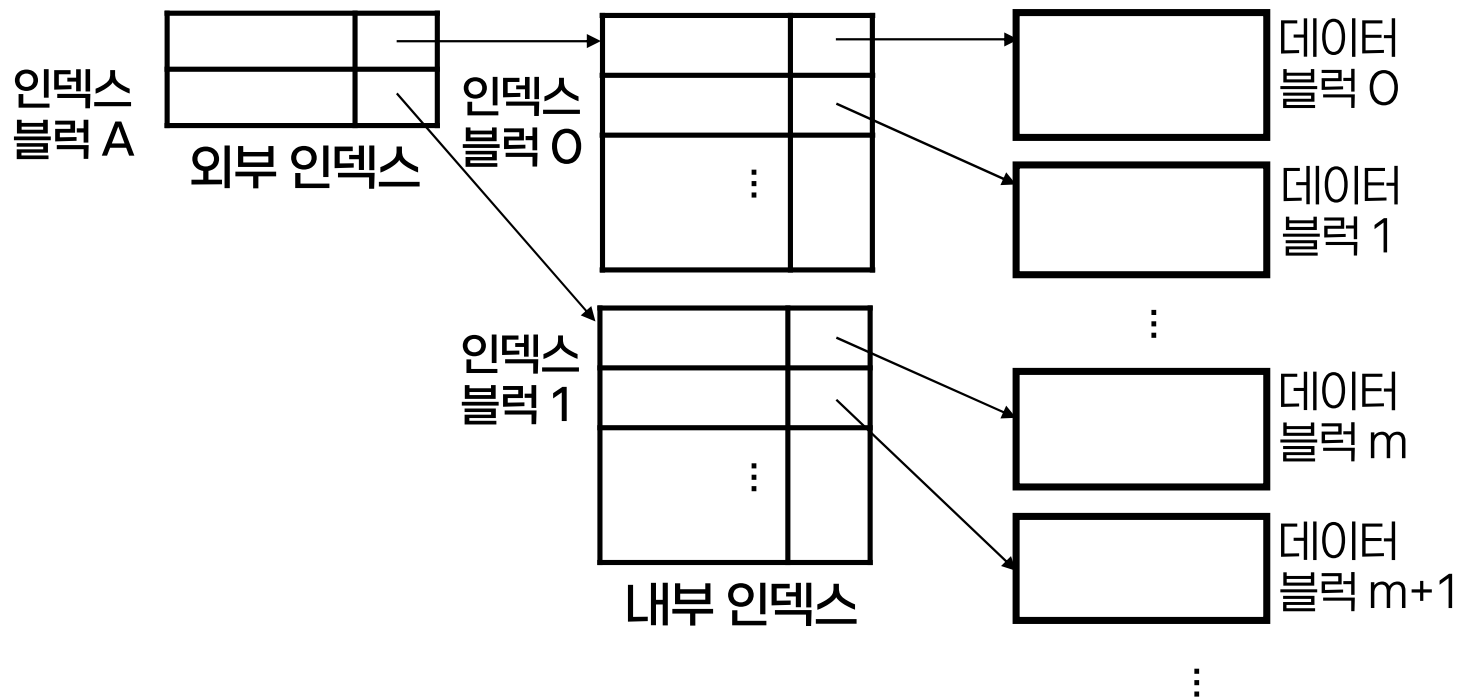


복수 계층의 인덱스를 구성

다단계 인덱스의 구조

▷ 내부 인덱스와 외부 인덱스로 구성

- ⊕ 외부 인덱스를 내부 인덱스보다 희소한 인덱스로 구성하여 엔트리의 포인터가 내부 인덱스 블록을 지칭

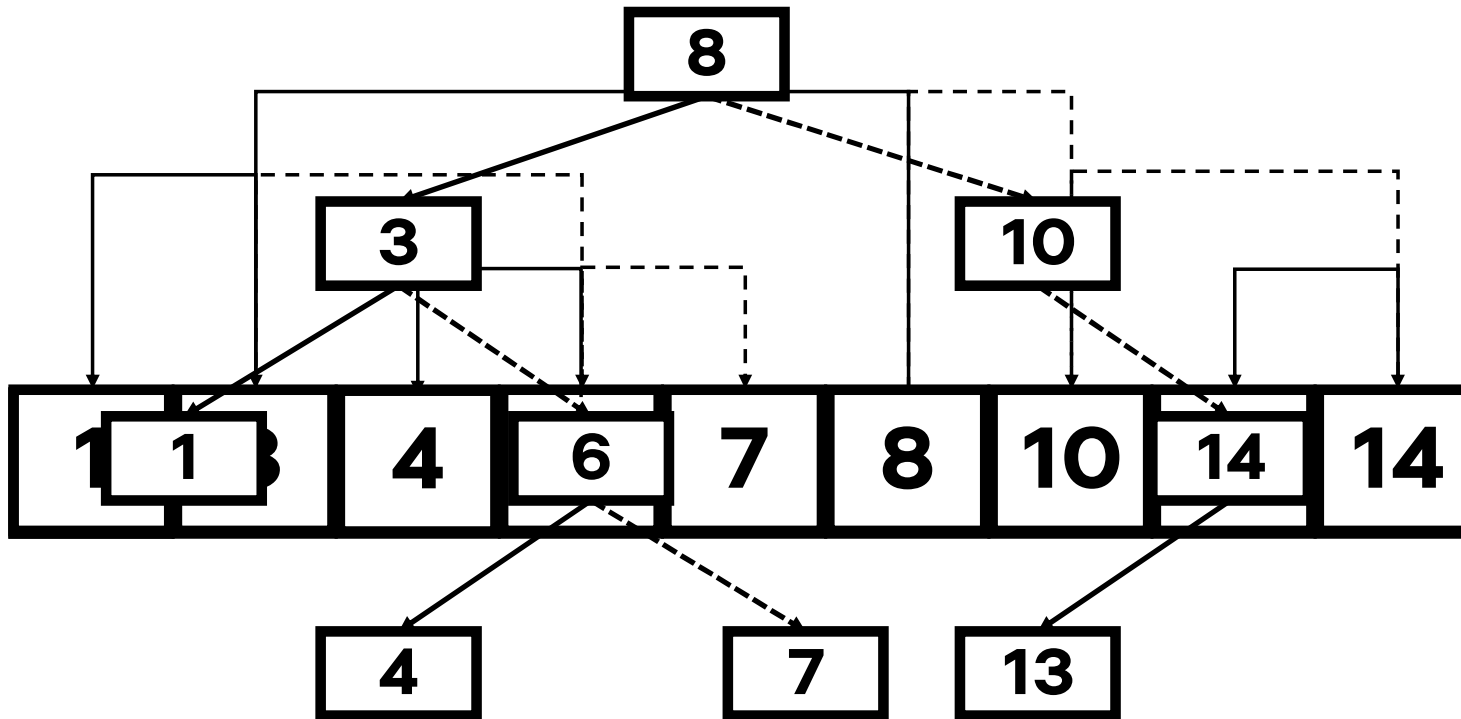


03

B⁺-트리 인덱스

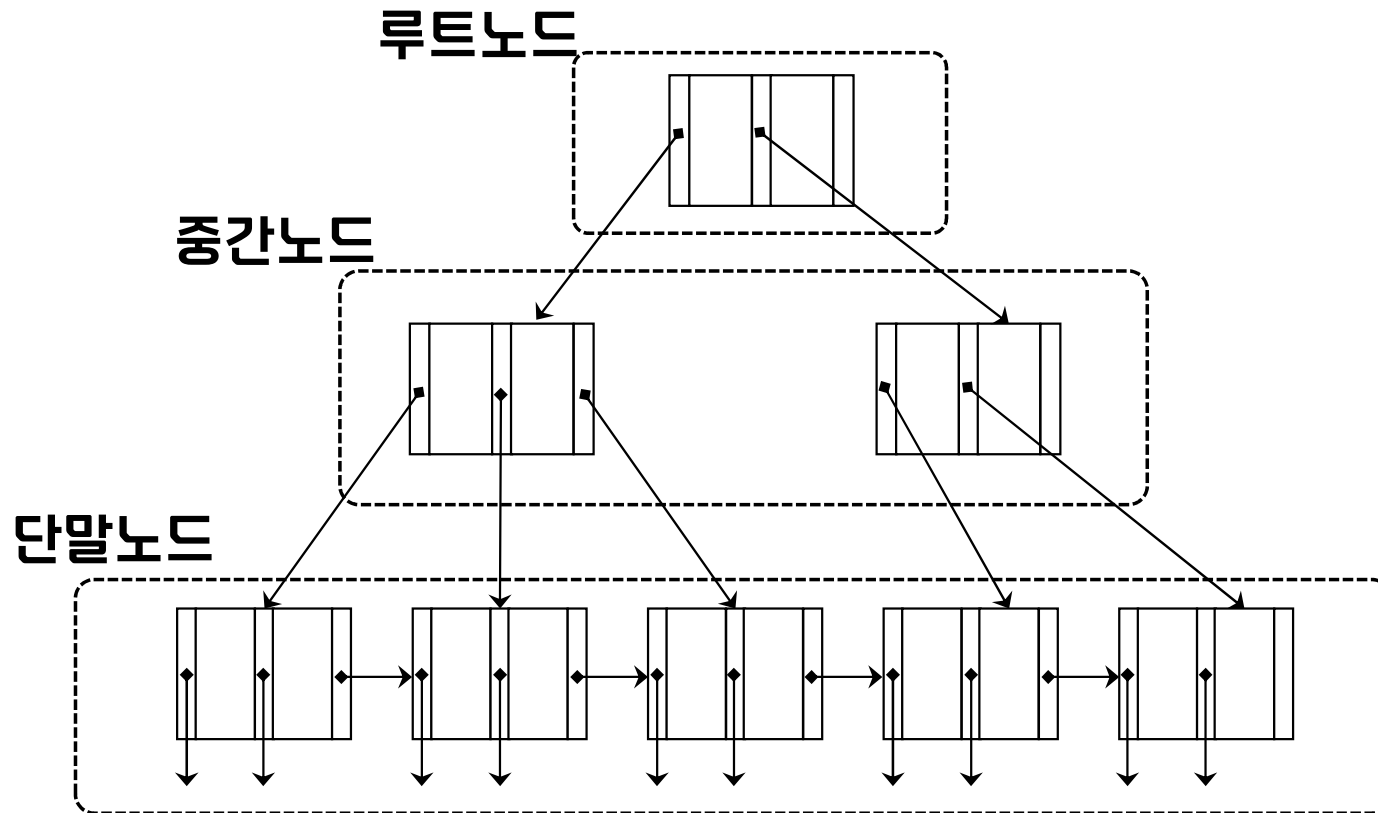
- B⁺-트리의 구조
- B⁺-트리의 구성 요소
- B⁺-트리 기반 검색
- 탐색키 삽입과 삭제

카드 찾기



이진 탐색 트리(binary search tree)

☆ B^+ -트리의 구조



☆ B^+ -트리의 구조

◇ 루트노드부터 모든 단말노드에 이르는 경로의 길이가 같은 높이 균형 트리

- ⊕ 순서 인덱스는 파일이 커질수록 데이터 탐색에 있어서 접근 비용이 커지는 문제점을 해결하기 위해 제안
- ⊕ 상용DBMS에서도 널리 사용되는 대표적인 순서 인덱스

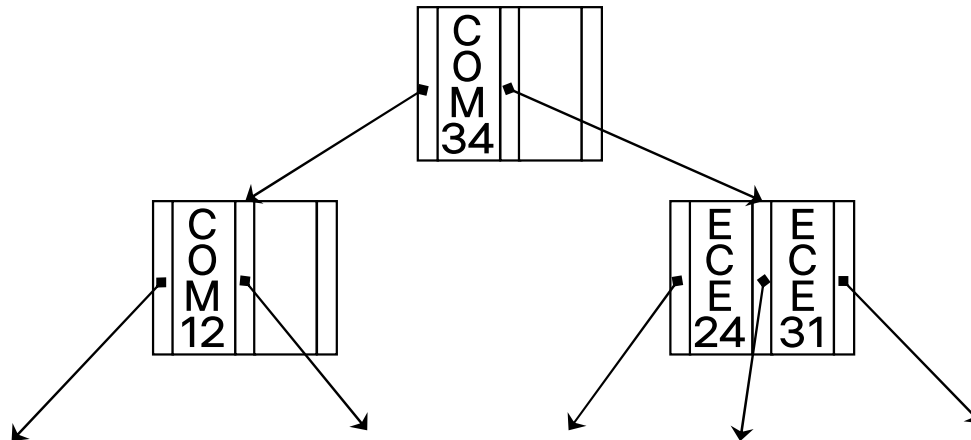
◇ B^+ -트리의 노드 구조

팬아웃(fanout)



☆ B^+ -트리의 구성 요소

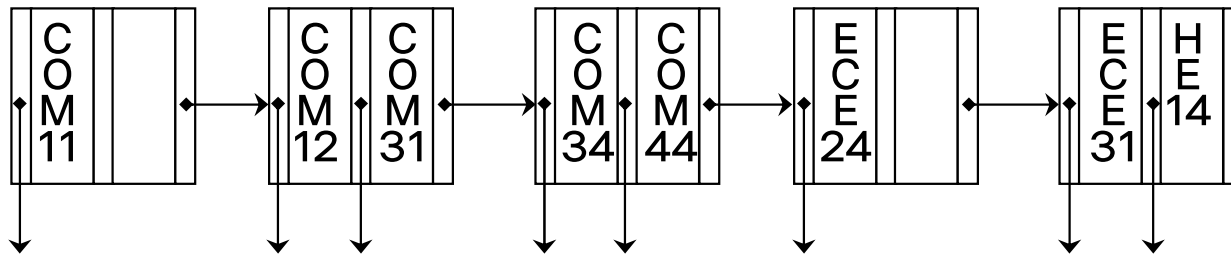
- ◇ 인덱스 세트: 루트노드와 중간노드로 구성
 - ⊕ 단말노드에 있는 탐색키 값을 신속하게 찾아갈 수 있도록 경로를 제공하는 목적으로 사용
 - ⊕ $[n/2] \sim n$ 사이의 개수를 자식으로 소유



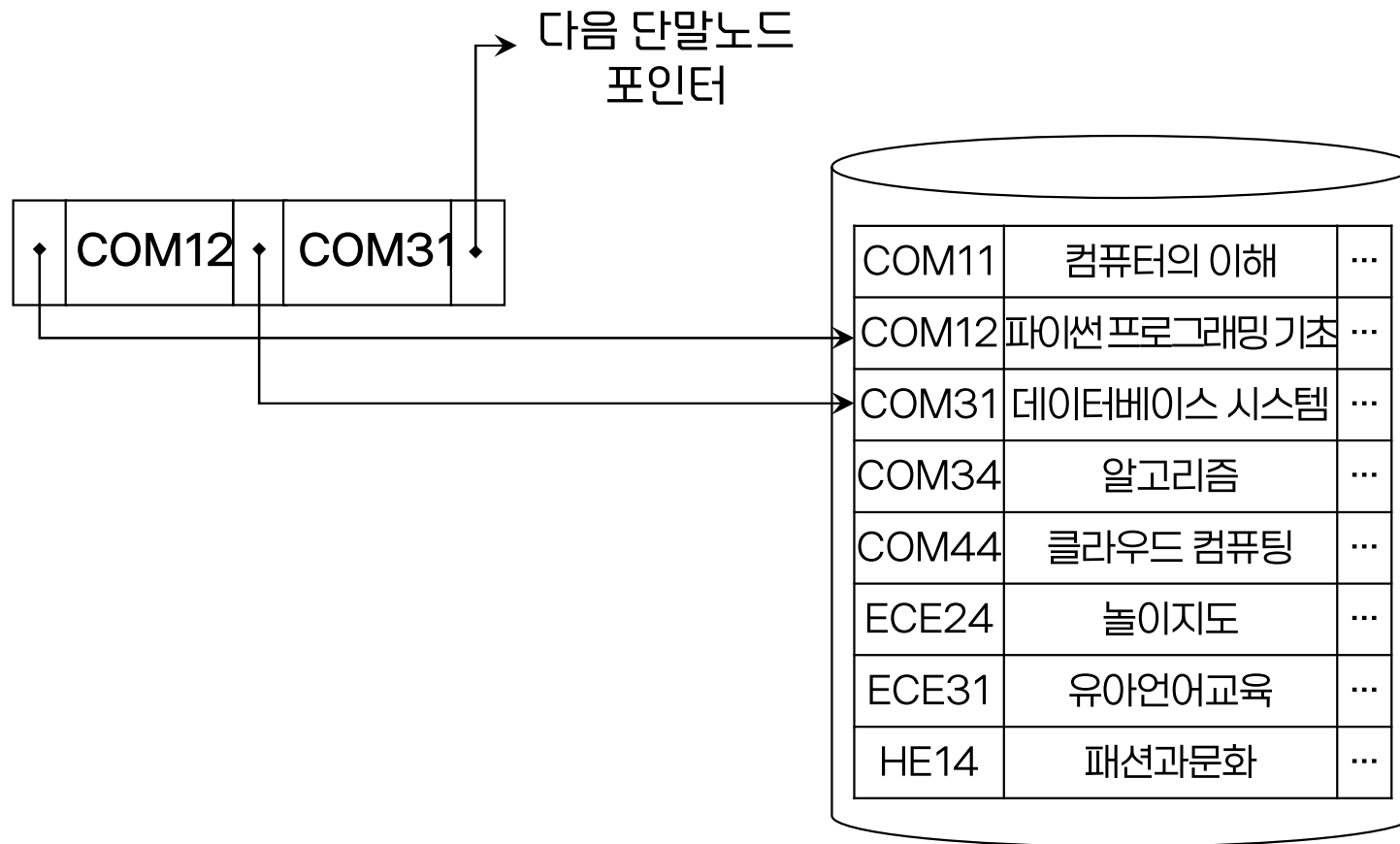
☆ B^+ -트리의 구성 요소

▷ 순차 세트: 단말노드로 구성

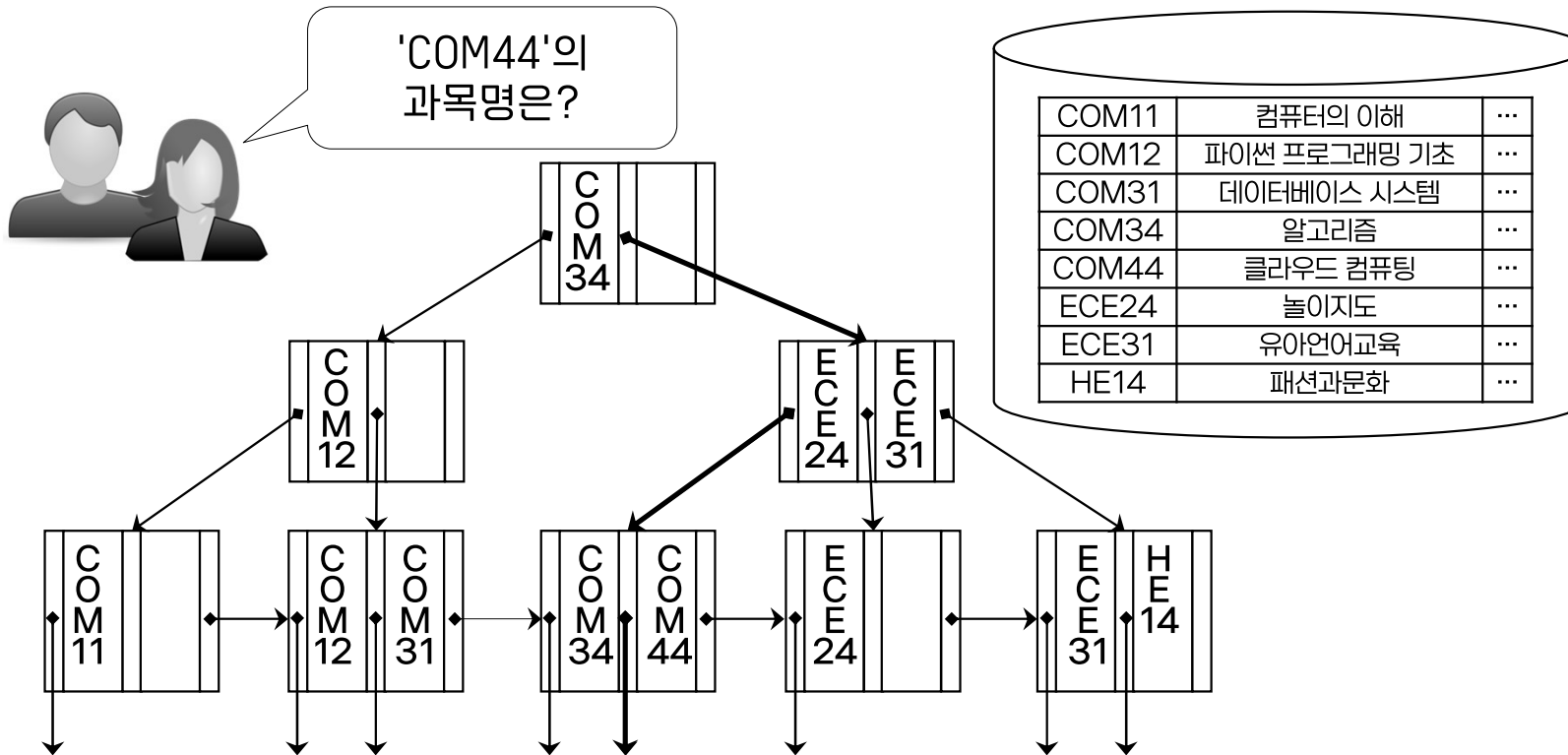
- + 모든 노드가 순차적으로 서로 연결
- + 적어도 $\lceil (n-1)/2 \rceil$ 개의 탐색키를 포함
- + 탐색키에 대한 실제 레코드를 지칭하는 포인터를 제공



단말노드의 구성



☆ B⁺-트리의 예



<https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html>

☆ B^+ -트리 상에서의 삽입, 삭제

▷ 레코드 삽입, 삭제 시 B^+ -트리 수정

+ 레코드 삽입:

- 노드에서 유지해야 할 탐색키와 포인터 수 증가로 인해 노드를 분할해야 하는 상황이 발생

+ 레코드 삭제

- 노드에서 유지해야 할 탐색키 값과 포인터 수 감소로 형제 노드와 키를 재분배 또는 병합해야 하는 상황이 발생

+ 높이 균형 유지

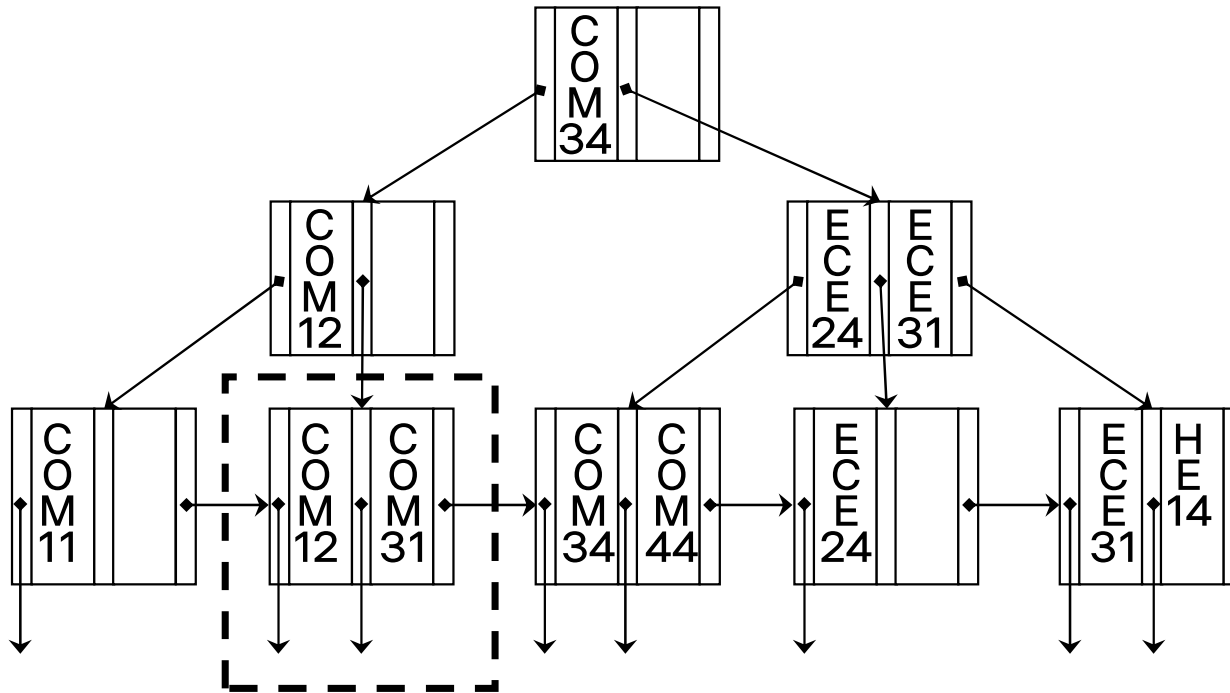
- 노드가 분할되거나 병합되면서 높이의 균형이 맞지 않는 상황이 발생

☆ B^+ -트리 상에서의 삽입과 삭제

- ◇ 삽입: 검색과 같은 방법을 사용하여 삽입되는 레코드의 탐색키 값이 속할 단말 노드를 탐색
 - ⊕ 해당 단말 노드에 <탐색키, 포인터> 쌍을 삽입
 - ⊕ 삽입 시 탐색키가 순서를 유지
- ◇ 삭제: 삭제될 레코드의 탐색키를 통해 삭제될 탐색키와 포인터를 포함한 단말 노드를 탐색
 - ⊕ 같은 탐색키 값을 가지는 다중 엔트리가 존재할 경우, 삭제될 레코드를 가리키는 엔트리를 찾을 때까지 탐색 후 단말노드에서 제거
 - ⊕ 단말노드에서 제거된 엔트리의 오른쪽에 있는 엔트리들은 빈 공간이 없도록 왼쪽으로 이동

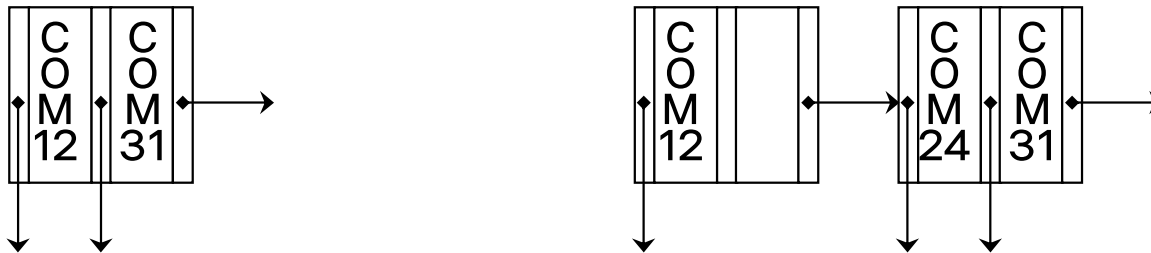
노드가 분할되는 삽입

▷ 'COM24' 삽입



노드가 분할되는 삽입

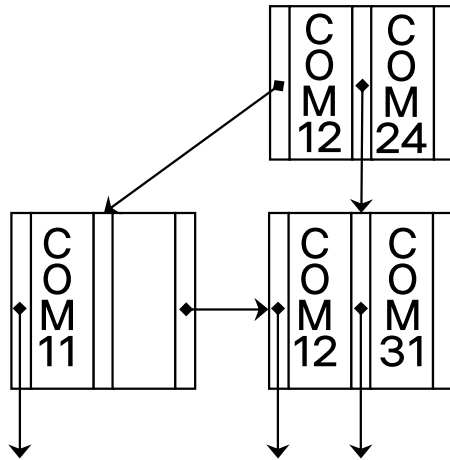
▷ 'COM24' 삽입



- ⊕ 삽입 대상 노드에 추가적인 저장할 공간 부족:
노드 분할
 - COM12를 하나의 단말 노드로 구성
 - COM24와 COM31이 하나의 단말 노드로 구성

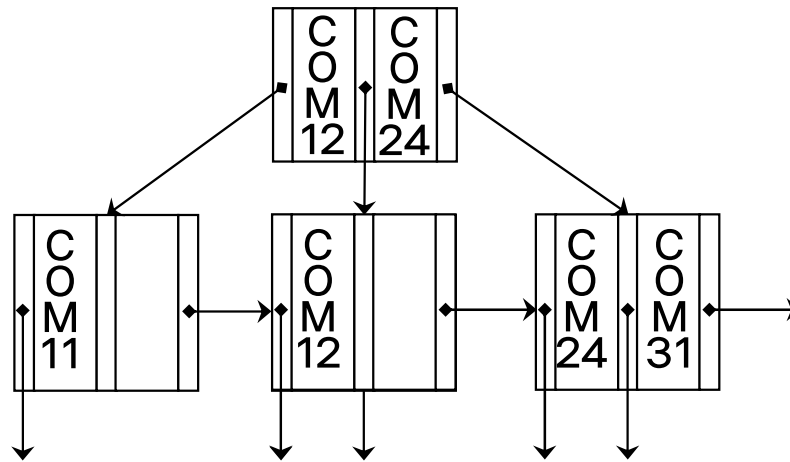
노드가 분할되는 삽입

- ▶ 부모 노드에 탐색키를 조정하고 추가된 노드에 대한 포인터를 삽입



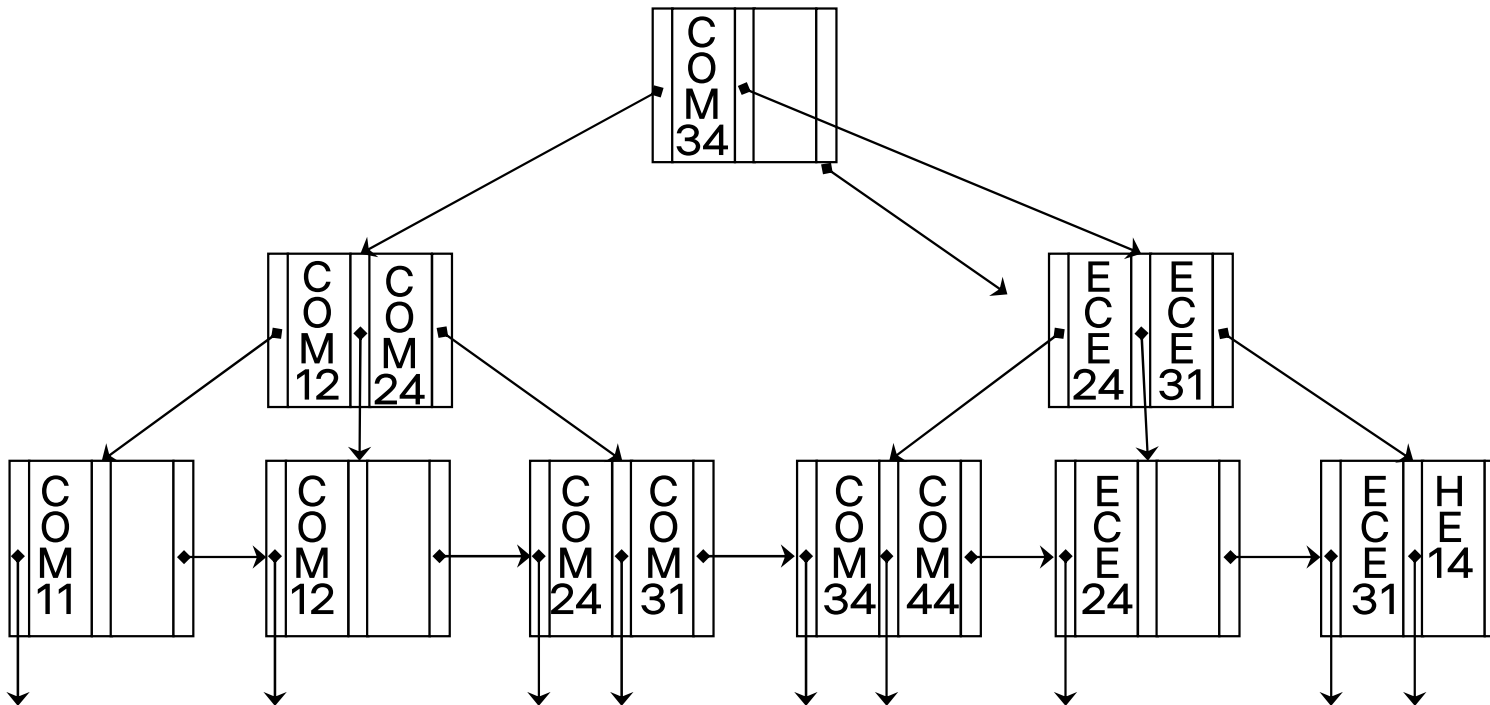
노드가 분할되는 삽입

- ▶ 부모 노드에 탐색키를 조정하고 추가된 노드에 대한 포인터를 삽입



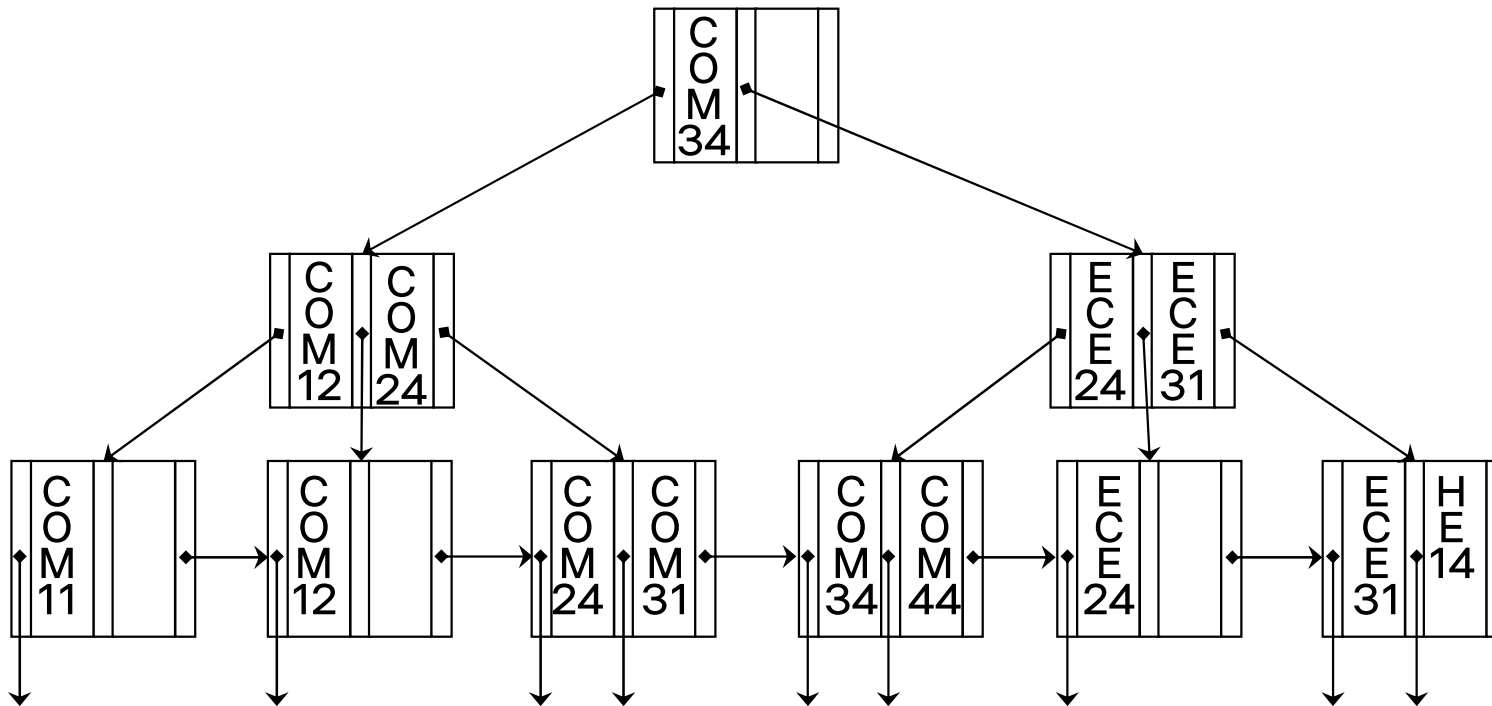
노드가 분할되는 삽입

- ▶ 부모 노드에 탐색키를 조정하고 추가된 노드에 대한 포인터를 삽입



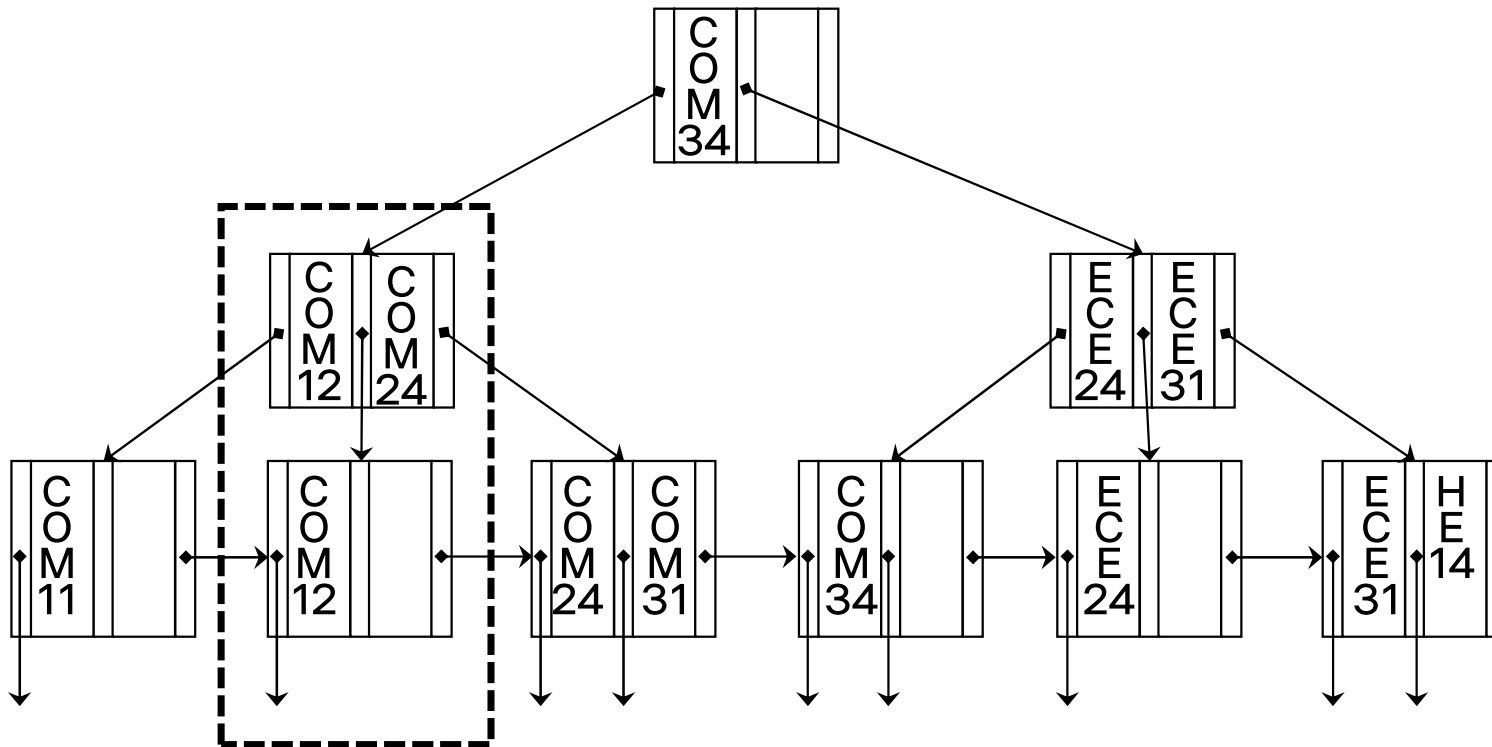
탐색키의 삭제

▷ 'COM44' 삭제



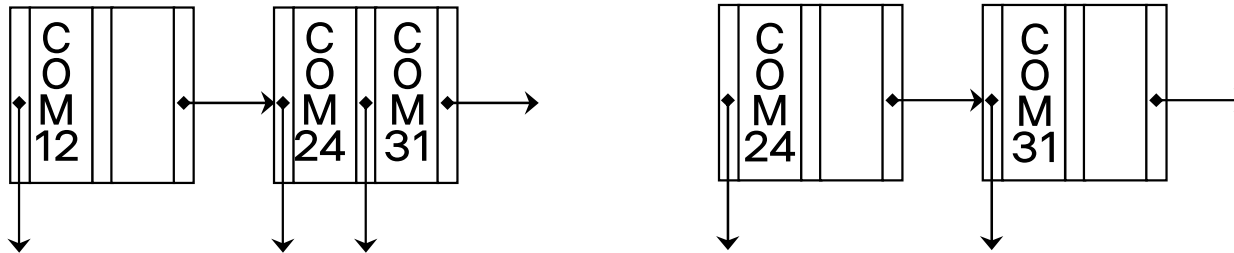
탐색키의 삭제

▷ 'COM12' 삭제



★ 탐색키가 재분배되는 삭제

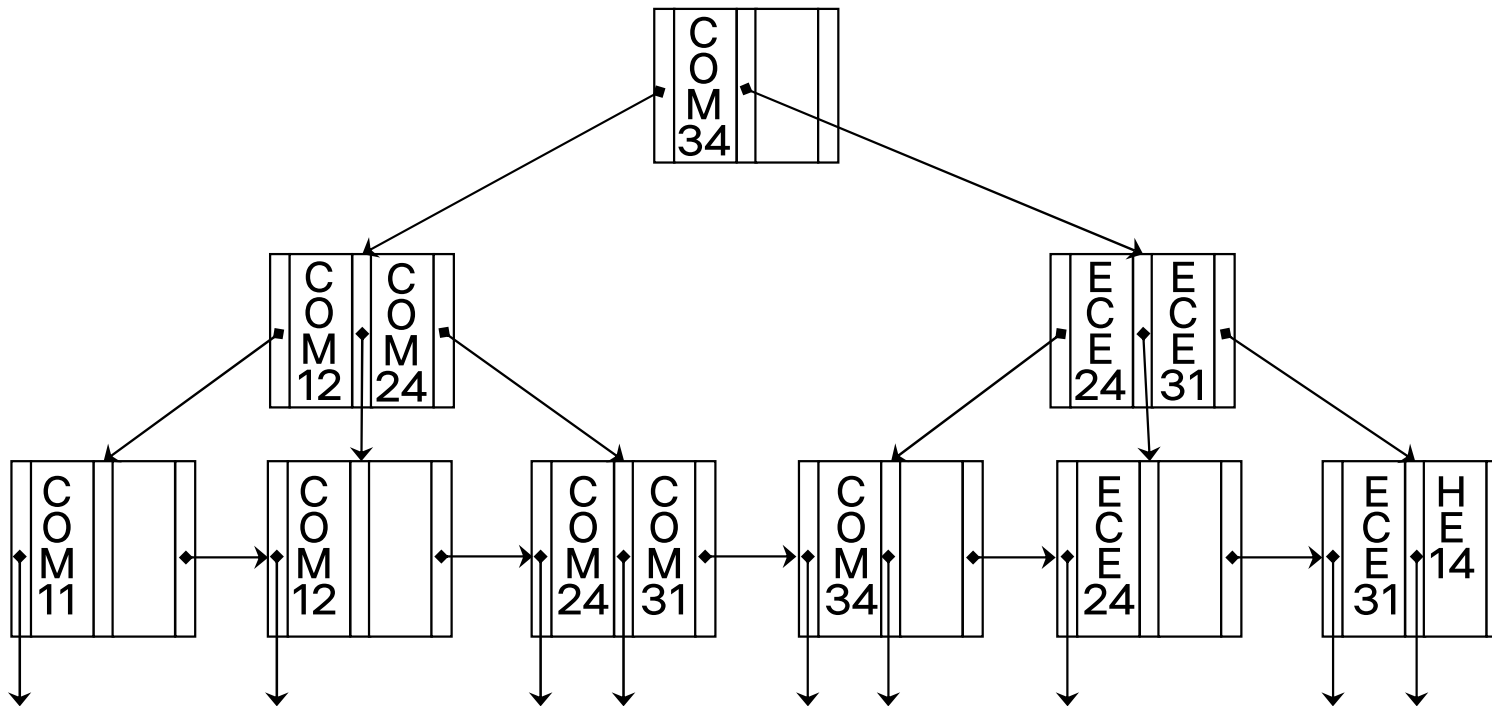
▷ 'COM12' 삭제



- ⊕ 'COM12'가 있는 단말 노드를 검색하고 탐색키를 삭제
 - 해당 단말 노드는 삭제 후 탐색키가 존재하지 않음
 - $\lceil (n - 1) / 2 \rceil$ 개 보다 탐색키가 적으므로 다른 노드와 별도의 재구조화 작업이 필요
- ⊕ 'COM12'가 저장된 노드의 오른쪽쪽의 형제 노드와 키를 재분배

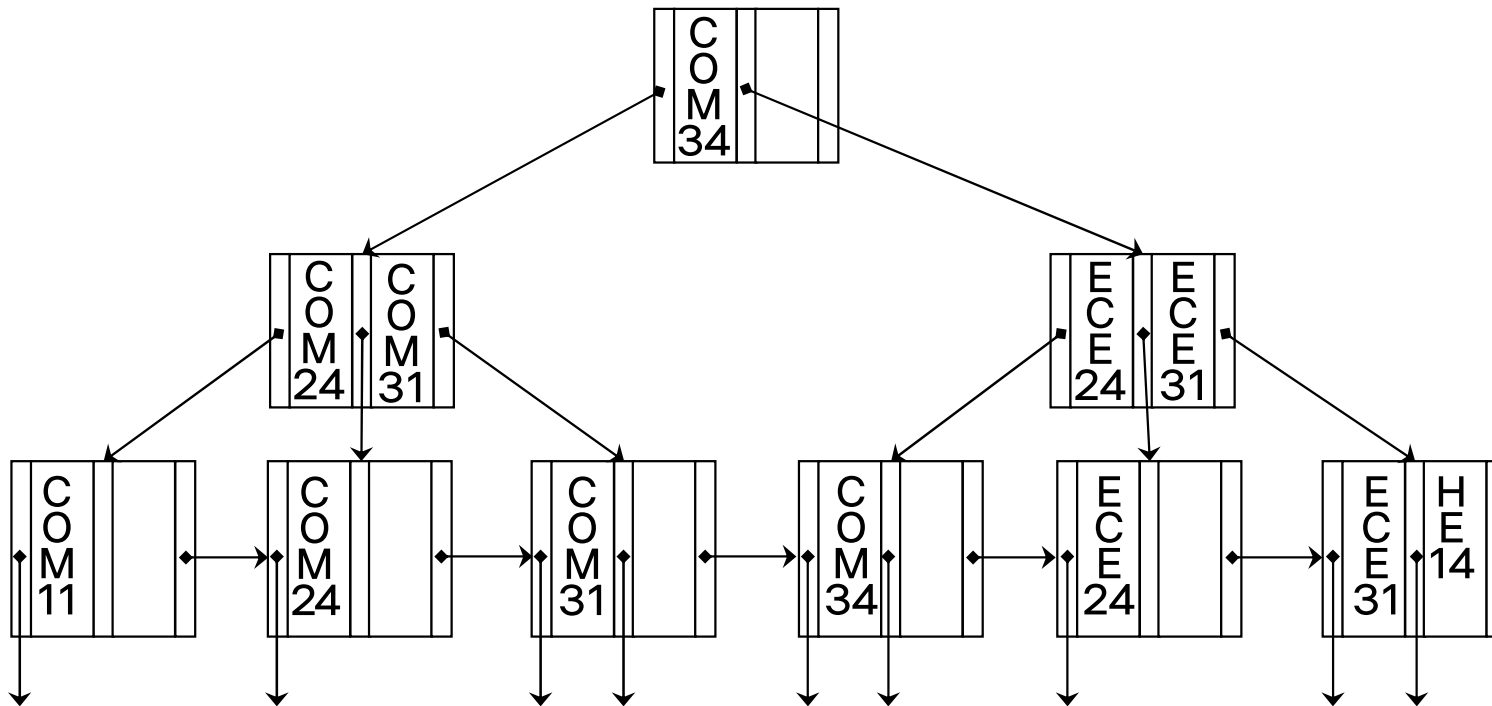
☆ 탐색기가 재분배되는 삭제

▷ 'COM12' 삭제



☆ 탐색기가 재분배되는 삭제

▷ 'COM12' 삭제



다음 시간



해싱과 특수 인덱스