- **9.** 인덱싱과 해싱
  - **1.** 인덱싱의 이해**[305]**
    - 인덱스**(index)**
      - **DBMS**에서 요청된 레코드에 빠르게 접근할 수 있도록 데이터와 관련된 부가적인 구조인 인덱스를 관리
      - 순서를 가지고 정렬되어 있음
    - 인덱싱**(indexing)**
      - 인덱스를 디자인하고 생성
    - 탐색키**(search key)**
      - 한 파일에서 레코드를 찾는 데 사용되는 컬럼이나 컬럼의 집합
    - 인덱스의 종류
      - 순서 인덱스**(ordered index)**
        - 탐색키값에 대해 정렬된 순서에 기초함
      - 해시 인덱스**(hash index)**
        - 버킷의 범위 안에서 값의 균일한 분포에 기초함
        - 어떤 값이 어느 버킷에 할당되는지는 해시 함수**(hash function)**라는 함수에 의해 결정됨
    - 인덱스의 평가 기준
      - 접근 시간**(access time)**
        - 사용자가 원하는 데이터를 찾는 데 걸리는 시간
      - 유지 비용**(maintenenance cost)**
        - 새로운 데이터 삽입 및 기존 데이터 삭제 연산으로 인한 인덱스 구조 갱신 비용
      - 공간 비용**(space cost)**
        - 인덱스 구조에 의해 사용되는 부가적인 공간 비용
  - **2.** 순서 인덱스 **[308]**
    - 순서 인덱스
      - 데이터가 정렬되어 있는 가장 기본적인 형태의 인덱스
      - 지정된 탐색키값으로 정렬된 순차 파일**(sequential file)**에서는 데이터 레코드에 대한 빠른 임의 접근**(random access)**이 가능하도록 순서 인덱스를 사용할 수 있음
        - 일반적으로 기본키 순서로 정렬
    - 순서 인덱스의 구성
      - 순차 파일
        - 순차적으로 정렬**(sort)**된 순차 파일
        - 레코드 집합 전체에 대한 순차 접근을 지원하는 데 사용
      - 순서 인덱스
        - 각 레코드에 대한 포인터를 가지고 있는 순서 인덱스
        - 어느 한 특정 레코드에 대한 직접 접근을 위해 사용
        - 클러스터드 인덱스**(clustered index)**
          - 인덱스의 순서와 레코드가 저장된 순서가 동일한 인덱스
      - 순차 접근과 직접 접근을 동시에 지원하는 방식
        - 각 페이지의 처음과 마지막 단어를 표시한 페이지 헤딩을 이용하여 일단 이 단어가 있는 페이지를 찾은 다음에 그 페이지에서 원하는 단어를 순차적으로 찾아 나간다
          - ex) daisy를 찾을 때 'd'
    - **1.** 밀집과 희소 인덱스**[309]**
      - 인덱스 엔트리**(index entry)**
        - **<**탐색키 값**,** 포인터**>** 쌍으로 구성
        - 포인터
          - 탐색키에 대응되는 한 개 이상의 레코드에 대한 포인터
          - 블럭**ID** : 데이터 레코드가 저장되어 있는 디스크 블럭의 식별자
          - 오프셋**(offset)**: 블럭 안에서 레코드의 위치를 가리키는 오프셋
      - 밀집 인덱스**(dense index)**
        - 모든 탐색키값에 대해 인덱스 엔트리를 생성

- 크기가 훨씬 작은 밀집 인덱스를 메모리로 읽어 해당하는 탐색키 인덱스 엔트리를 찾은 후 레코드에 대한 포인터에 따라 '데이터베이스 시스템'에 접근
- 적은 양의 디스크-메모리 입출력으로 질의 처리 속도를 향상
- 장점
  - 인덱스만으로도 요청된 탐색키에 해당하는 레코드의 블럭 내 위치를 정확하게 찾을 수 있기 때문에 검색의 속도가 희소 인덱스에 비해 빠르다
- 단점
  - 희소 인덱스에 비해 많은 공간을 사용한다는 점에서 비효율적

- 희소 인덱스(sparse index)
  - 일부의 탐색키값만을 대상으로 인덱스 엔트리를 생성
  - 'COM44' 찾기
    - 희소 인덱스에는 모든 탐색키가 존재하는 것이 아니기 때문에 찾고자 하는 탐색키보다 작거나 같은 값 중에서 가장 큰 키값을 찾는다.
      - 'COM31'이 이에 해당하는 키값이므로 1) 이 인덱스 엔트리의 포인터를 따라 '데이터베이스 시스템'에 해당하는 레코드를 찾은 후 2) 주어진 'COM44'에 해당하는 레코드를 발견할 때까지 순차적으로 다음 레코드를 읽어 '클라우드 컴퓨팅'에 해당하는 레코드를 찾게 된다.
  - 장점
    - 인덱스의 크기가 작아 저장공간을 작게 차지
    - 인덱스를 적은 저장장치 접근 횟수로도 사용
  - 단점
    - 요청 탐색키를 찾기 위해 인덱스 엔트리가 가리키는 블럭 내부에서 레코드에 대한 작은 규모의 검색을 다시 해야 한다

- 2. 다단계 인덱스 [311]
  - 일반적으로 인덱스는 디스크에 저장되며, 인덱스의 크기가 작아 메모리에 모두 적재될 수 있다면 인덱스 탐색 시간이 짧아서 문제가 되지 않는다.
  - 그러나 인덱스 파일 크기가 메모리 용량보다 커지면 인덱스 사용 시 디스크로부터 인덱스 블럭을 읽어 들이는 입출력(I/O) 비용이 증가하여 탐색 시간이 증가하게 된다.
  - 데이터 접근 비용을 절감하기 위해 밀집 인덱스와 희소 인덱스의 장단점을 적절하게 결합한 다단계 인덱스를 구성할 수 있으며, 90도 회전하면 트리 모양의 인덱스가 된다.
    - 내부 인덱스(inner index)
      - 레코드를 직접 가리키는 인덱스 엔트리의 집합
    - 외부 인덱스(outer index)
      - 내부 인덱스를 가리키는 인덱스 엔트리의 집합
    - 외부 인덱스에 대해서는 희소 인덱스 방식으로, 내부 인덱스에 대해서는 밀집 인덱스 방식으로 인덱스 블럭을 읽어 들인다.
      - ① 외부 인덱스에서 원하는 엔트리의 탐색키값보다 작거나 같은 탐색키값 중 에서 가장 큰 값을 갖는 인덱스 엔트리를 찾는다. ② 이 인덱스 엔트리의 포인터가 가리키는 블록을 적재한다. ③ 내부 인덱스에 도달할 때까지 1~2 과정을 반복한다. ④ 내부 인덱스에서 조건과 일치하는 인덱스 엔트리의 탐색키를 찾아 데이터 블록을 적재한다. ⑤ 해당 레코드를 사용자에게 반환한다.

- B+ -트리 인덱스[313]
  - B+트리 인덱스
    - 현대 상용 DBMS에서 가장 널리 사용되는 순서 인덱스의 일종
    - 경로(path)의 길이가 같은 높이 균형 트리(height balanced tree) 형태로 구성되어 검색의 속도를 일정하게 향상시켜 안정적 데이터 검색이 이루어질 수 있도록 고려한 인덱스 구조
  - 1. B+ - 트리 구조
    - 루트 노드(root node)로부터 모든 단말 노드(leaf node)에 이르는 경로의 길이가 같은 높이 균형 트리
    - 다단계 인덱스의 일종
    - 이진 검색 트리(binary search tree)의 확장된 버전
    - 중간 노드(internal node)
      - 단말 노드나 루트 노드가 아닌 그 사이의 노드
      - ⌈n/2⌉과 n 사이의 자식을 갖는다
        - n은 노드가 포함할 수 있는 인덱스 엔트리의 최댓값이다
    - 그림9-7
      - [그림 9-7]은 일반적인 B^+-트리를 구성하는 노드 구조를 나타내며, n-1개의 탐색키값 K_1, K_2, ..., K_{n-1}과 자식 노드를 가리키는 n개의 포인터 P_1, P_2, ..., P_n 을 포함한다. 노드 안의 탐색키값은 정렬된 순서로 유지되어 있으며, 한 노드에 저장되는 최대 포인터의 개수는 B^+-트리의 차수(order)에 의해 결정된다. [그림 9-7]은 차수가 n인 일반적인 B^+-트리의 노드이다.

- 왜 포인터가 탐색키보다 하나 더 많을까?
- 그림 9-8
  - B+트리에서 포인터는 자식 노드나 레코드의 저장 위치를 가리킨다.
  - 이 때 한 노드가 가질 수 있는 자식 노드의 개수 m은⌈차수/2⌉ <= m <= 차수를 만족해야 한다.
    - 예) 차수가 3인 B+-트리의 경우 m은 2 이상 3 이하로, 최소 2개 혹은 최대 3개의 자식 노드를 갖는다.
  - B+ 트리 구성 [315]
    - 인덱스 세트(index set)
      - 단말이 아닌 노드로 이루어진 인덱스 세트
      - 인덱스 세트에 있는 자식 노드에는 탐색키 값만 있다
        - 단말 노드에 있는 탐색키 값을 신속하게 찾아갈 수 있는 경로를 제공한다
    - 순차 세트(ordered set)
      - 단말 노드로만 구성된 순차 세트
      - 순차 세트에는 모든 탐색키값과 연관된 데이터 레코드가 저장된 주소가 있음 (그림 9-6)
        - 모든 노드가 연결 리스트 형태로 순차적으로 연결되어 있으며, 이것은 파일에 저장된 레코드를 탐색키값 순서에 따라 효율적으로 접근할 수 있도록 한다.
      - 단말 노드의 구조를 살펴보면 i=1, 2, ..., n-1일 때, P_i는 탐색키 값 K_i를 가지는 레코드를 가리킨다.
      - 마지막 포인터 Pn은 단말 노드를 탐색키 순서로 연결하기 위해 다음 단말 노드를 가리키는 데 사용 된다.
    - 그림 9-10
      - 비단말 노드(루트 및 중간 노드)는 포인터를 최대 n개까지 가질 수 있음 최소한 ⌈차수/2⌉개의 포인터는 유지
      - 팬아웃(fanout)
        - 한 노드의 포인터 수
      - 예외적으로 루트 노드는 다른 비단말노드와 달리 최소한 ⌈차수/2⌉개보다 더 적은 포인터를 가질 수 있다
        - 루트는 0, 2 또는 ⌈차수/2⌉에서 차수 사이의 자식 노드(포인터)를 갖는다.
      - 모든 단말 노드는 같은 레벨에 있다. = 루트로부터 같은 거리에 있다
      - 단말 노드가 아닌 노드에 있는 탐색키값의 수는 그 노드의 자식 노드 수보다 하나 적다
      - 단말 노드는 데이터 파일의 순차 세트를 나타내며, 모두 리스트로 연결되어 있다.
      - 단말 노드는 적어도 ⌈(차수-1)/2⌉개의 탐색키값을 포함해야 한다.

B+ -트리 인덱스 (2) [317]

2. B+-트리에서 검색, 삽입, 삭제 연산

1. B+-트리상에서 검색

- 특정 탐색키값에 대한 b+트리 검색 연산은 인덱스 세트를 거쳐 순차 세트에 도달하여 탐색키에 해당하는 레코드 포인터를 획득하는 것으로 종료된다.
- 루트부터 단말 노드 도달까지 아래 방향으로 다음 단계 반복
  - 탐색키값 V에 대한 검색 연산을 수행할 때, 하나의 노드에 m개의 포인터가 존재한다고 가정하고, 현재의 노드를 N이라고 가정. 초기의 N은 루트 노드이다.
    - 단계 1: 현재 탐색 대상인 N을 조사하여 V와 같거나 큰 탐색키 중 가장 작은 키 Ki를 찾는다.
    - 단계 2: 아래의 조건에 따라 다음 노드를 결정한다. ① Ki = V일 경우, 포인터 P_{i+1}이 가리키는 노드를 N으로 결정 ② Ki > V일 경우, P_i가 가리키는 노드를 N으로 결정 ③ V보다 큰 탐색키가 없는 경우, 노드 내의 NULL이 아닌 마지막 포인터가 가리키는 노드를 N으로 결정
    - 단계 3: N이 단말 노드가 아니면, 단계 1부터 반복한다. N이 단말 노드일 경우, 탐색키값 Ki와 V가 같은 포인터 P_i를 반환한다.
    - 만약 검색 알고리즘을 통해 단말 노드에서 V를 찾지 못한다면, V를 가지는 레코드는 존재하지 않음을 의미
  - 예시