12. 내용 기반 정보 접근 향상 기법

이 장의 주요 내용

- 인덱스 정의
- 단일-단계 순서 인덱싱
  - 기본 인덱스, 클러스터링 인덱스, 보조 인덱스
- 다중 레벨 인덱싱
  - B+ 트리 인덱스 – 동적 다단계 인덱스
  - 인덱스된 순차 파일
- 해시 테이블
- SQL을 이용한 인덱스 생성

내용 기반의 레코드 접근을 위한 인덱스 이용

- 파일로 관계 데이터베이스의 기능을 지원하기 위해서는 내용을 기반으로 한 레코드의 접근이 필요

- 목적 : 탐색을 최대한 효율적으로 처리하는 것 → 이를 위해 인덱싱(indexing) 방법 소개

인덱스란?

- 파일의 인덱스는 책 인덱스와 비슷

- 책 인덱스
  - 내용 기반으로 책의 각 키워드를 검색
  - 키워드- 페이지 번호들

- 파일 인덱스
  - 레코드 필드들로 구성된 검색 키 - 레코드가 저장 된 파일의 위치

인덱스란? (cont'd)

- 여러 필드(또는 애트리뷰트)로 구성된 레코드 구조를 가진 파일에서는 일반적으로 인덱스 필드(또는 인덱스 애트리뷰트)라고 부르는 파일의 한 필드에 대한 인덱스 접근 구조를 정의

- 인덱스 = 인덱스 필드의 각 값 + 그 필드 값을 포함하는 레코드들을 갖고 있는 모든 디스크 블록을 가리키는 포인터들의 리스트들

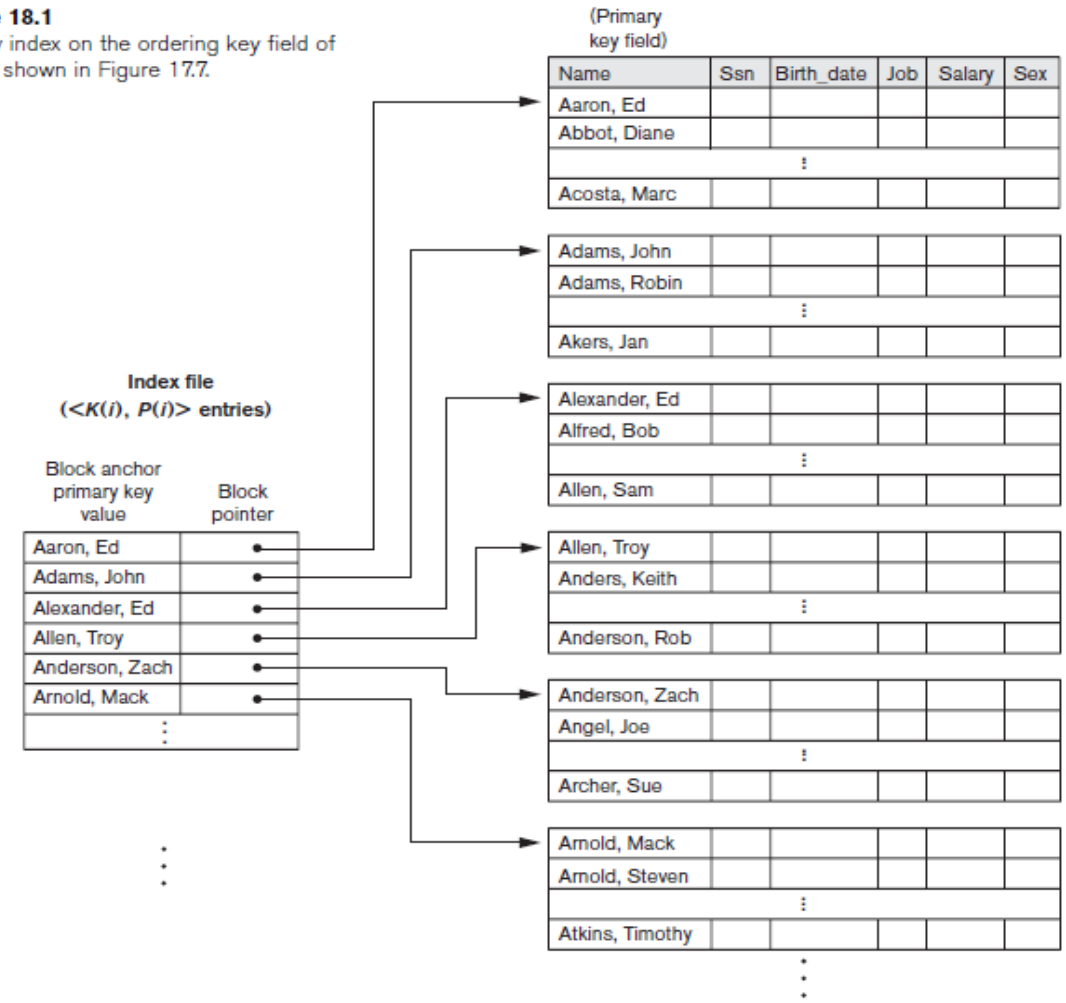
- 인덱스 값들은 쉽게 검색 될 수 있도록 트리 등 효율적인 구조를 가지도록 함

단일-단계 순서 인덱싱의 유형

- **기본 인덱스(primary index)**
  - 키 필드(key field) 에 대한 인덱스
  - 키 필드는 디스크에 있는 파일 레코드들을 물리적인 순서로 정돈하는데 이용되고, 각 레코드는 그 필드에 유일한 값을 가짐
- **보조 인덱스 (secondary index)**
  - 비 키 필드에 대해 정의
- **클러스터링 인덱스(clustering index)**
  - 필드에 대해 같은 값을 갖는 레코드들이 여러 개 있을 경우
- **한 데이터 파일은 기본 접근 방법 이외에 여러 개의 보조 인덱스가 있을 수 있음**

기본 인덱스의 예

Figure 18.1
Primary index on the ordering key field of the file shown in Figure 17.7.



클러스터링 인덱스의 예

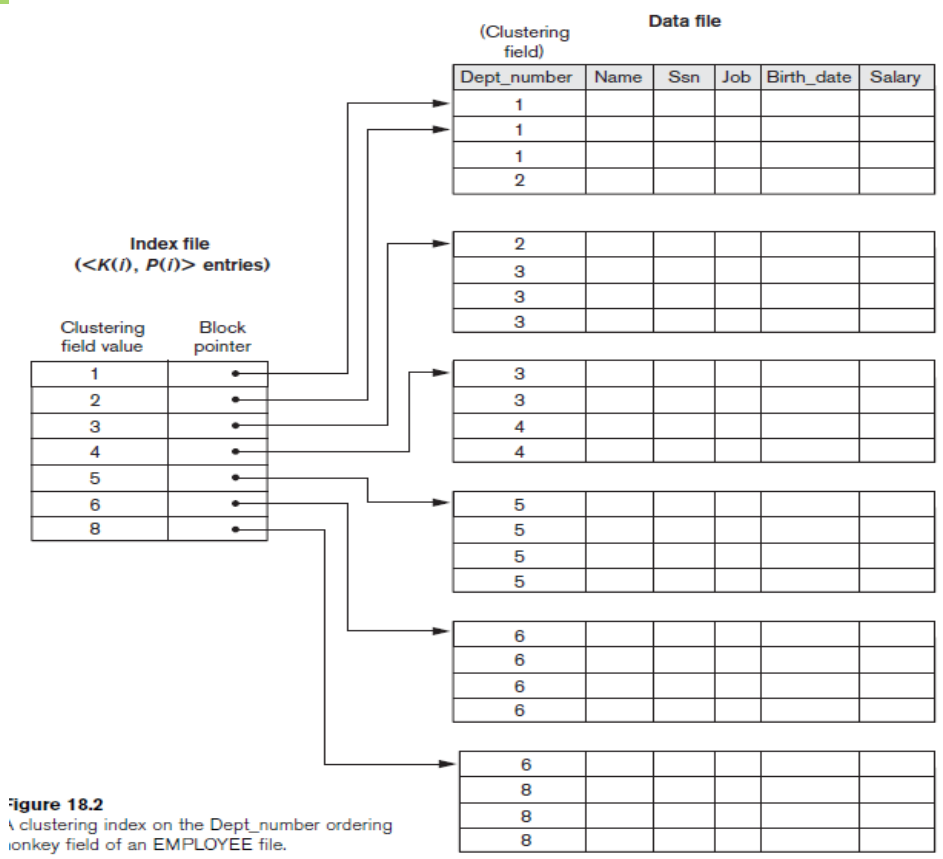
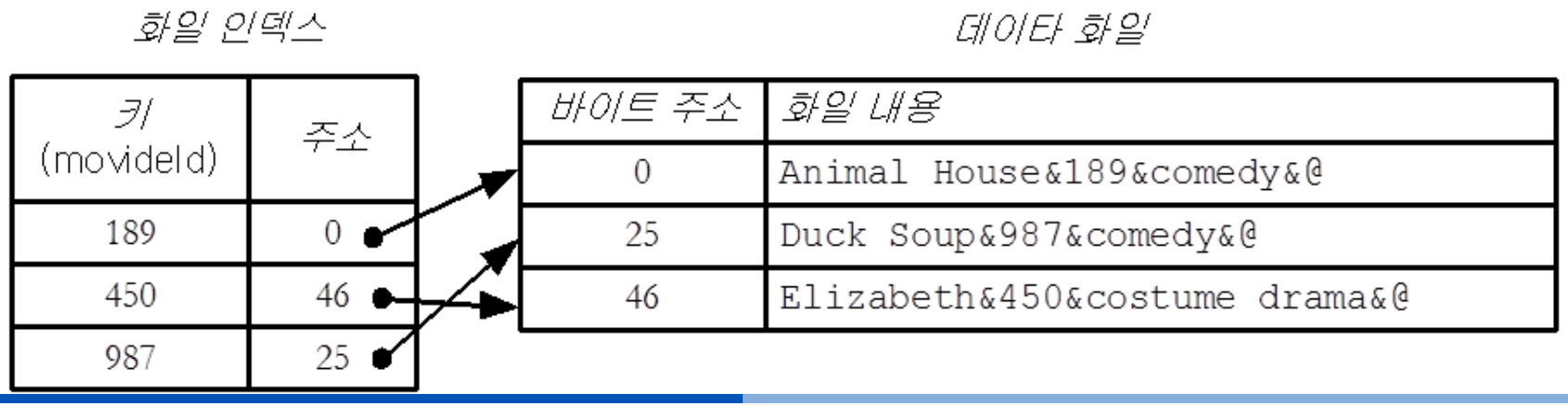


Figure 18.2
A clustering index on the Dept_number ordering nonkey field of an EMPLOYEE file.

- 장점 : 같은 값을 가지는 모든 레코드들을 검색하는데 시간을 줄일 수 있음

유일 키 필드에 대한 보조 인덱스 예

- 기본 접근 방법이 이미 존재하는 데이터 파일을 접근하는 보조 수단
- 가정 : 키는 한 개의 주소를 가지며, 인덱스에 한 번만 나타남 → 유일 키 (unique-key) 인덱스
- 3개의 Movie 객체를 가지는 파일과 키-참조(key-reference) 집합으로 구성된 인덱스



인덱스된 파일의 검색, 삽입, 삭제, 갱신 연산

- 데이터 파일에 대한 레코드의 읽기와 쓰기 연산은 인덱스를 검색하고 갱신하는 연산을 수반해야 함

- 키 값이 450인 레코드 접근 방법
  1. 인덱스에서 키 값이 450인 키를 검색
  2. 키와 관련된 주소(46)을 얻음
  3. 데이타 화일에서 주소가 46인 레코드 읽음

- 새로운 레코드를 데이타 화일에 삽입하는 연산
  1. 화일에 키가 이미 있는지를 알기 위해 인덱스 검색
  2. 없으면, 레코드를 데이타 화일에 추가하고 새로운 레코드의 주소 결정
  3. 키-참조 쌍을 인덱스에 삽입

인덱스된 파일의 검색, 삽입, 삭제, 갱신 연산 (cont'd)

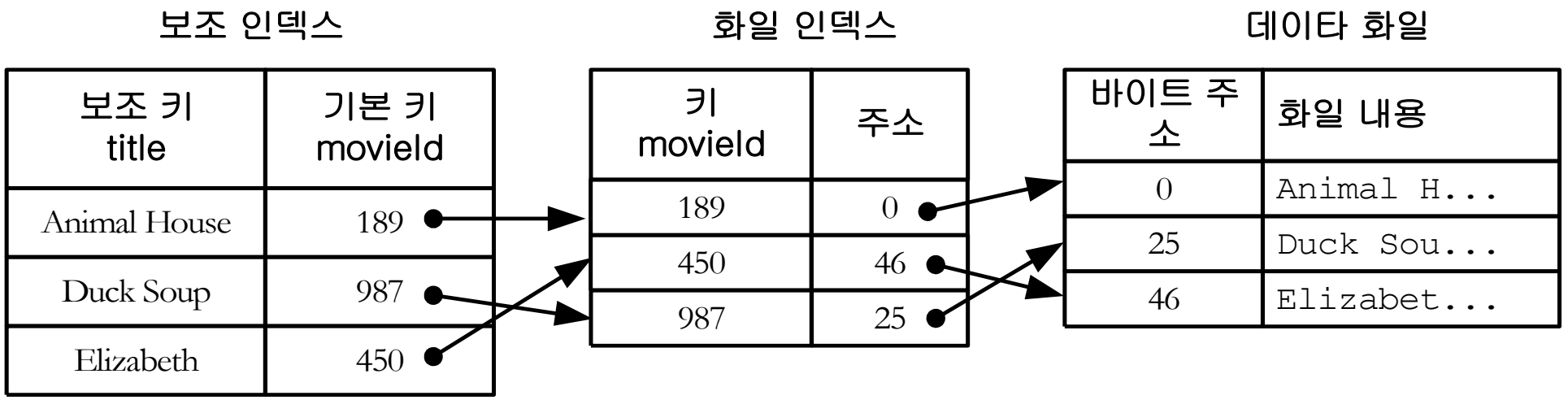
- **레코드 삭제 연산**
  - 인덱스에서 키-참조 값을 삭제

- **레코드 갱신 연산**
  - 레코드의 키 값이 변할 경우1 : 인덱스의 키 값도 변경
  - 레코드의 주소가 변할 경우 2: 인덱스의 주소 값도 변경
  - 키, 레코드의 주소가 변경 되지 않을 경우 3 : 인덱스 갱신 필요 없음

데이타 화일에 보조 인덱스를 추가하는 방법

- **방법 1**
  - 인덱스의 주소 필드들이 데이타 화일 레코드들을 직접 가리키도록 함
  - 단점 : 레코드의 주소가 변경되면 해당 보조 인덱스의 주소 필드들이 변경
- **방법 2**
  - 보조 인덱스의 주소 필드에 기본 키 값 이용
  - 데이타 레코드의 이동은 기본 키 인덱스만을 수정

보조 인덱스 추가하는 방법 2 예



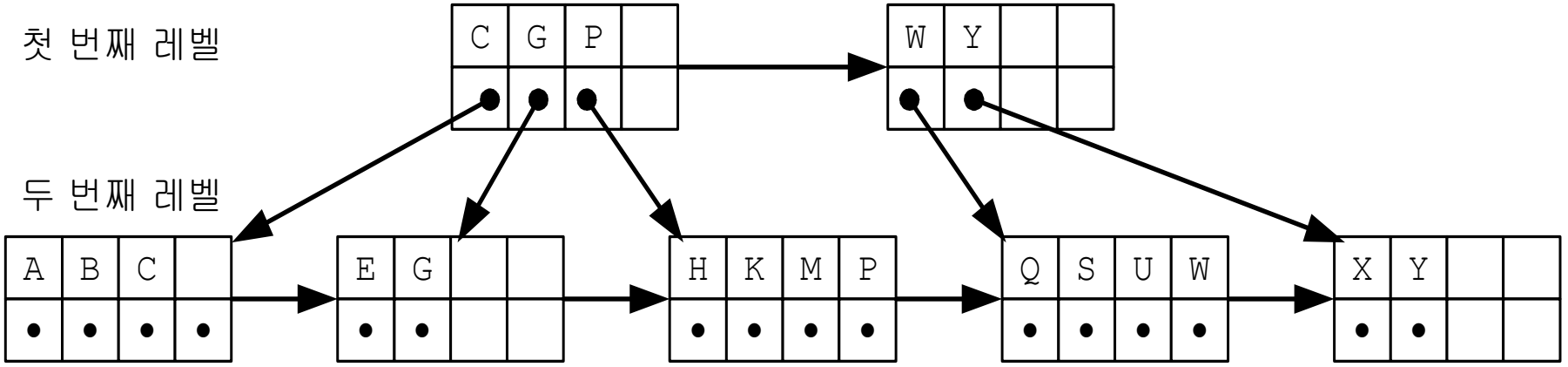
다중 레벨 인덱스와 B+ 트리

- **B+ 트리(B+ tree)**
  - B-트리의 변형
  - 높이-균형(height-balanced) 검색, 삽입, 삭제 연산의 실행 시간은 트리의 높이에 선형적
  - 매우 큰 화일에서도 빠른 검색 시간 제공
  - 다중 레벨 인덱스(multilevel index)임
    - 가장 낮은 레벨 인덱스 : 데이타 화일 인덱스
    - 다음 레벨 : 가장 낮은 레벨을 가리키는 인덱스

다중 레벨 인덱싱

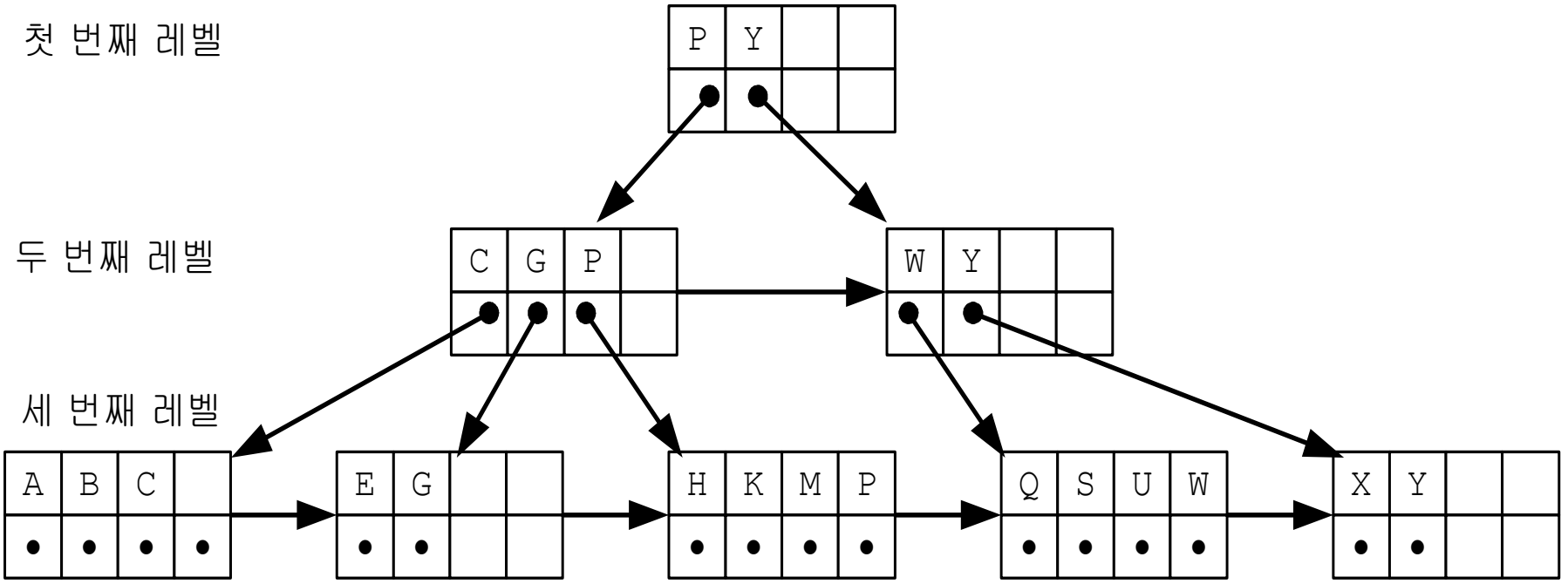
- 인덱스 레코드 파일에 대한 인덱스를 만들기 위해, 각 레코드를 대표하는 키를 선택

- 2-레벨 인덱스



다중 레벨 인덱싱 (cont'd)

- 3-레벨 인덱스



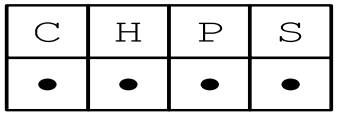
B+ 트리의 정의와 성능

- **B+ 트리**
  - 각 인덱스 레코드(또는 노드)가 가질 수 있는 최대/최소 키 개수가 정해져 있는 동적인 다중 레벨 인덱스
- **B+ 트리의 차수(order of the B+ tree)**
  - 인덱스 레코드의 최대 키 개수
- **규칙**
  - 차수가 k인 B+ 트리에서 각 노드는 k/2개에서 k개의 키를 가짐(루트 제외)
  - 내부 노드(리프 노드가 아닌)에서 참조는 다른 B+ 트리 노드들을 참조
  - 리프 노드에서 참조는 데이타 화일 레코드를 참조
  - 각 리프 노드들의 높이는 같음

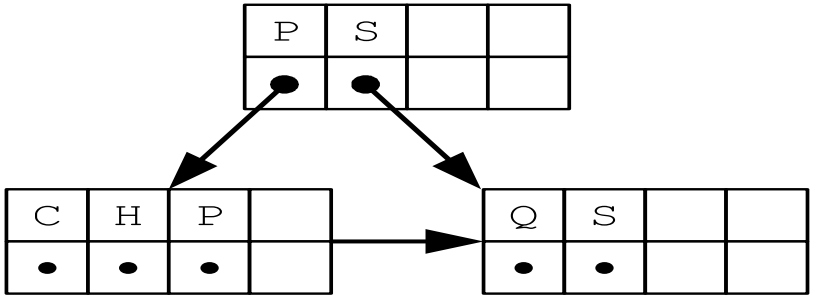
B+ 트리의 삽입

- 예 : 빈 트리 구조에서 CSHPQAGXWYBEUMK 글자 순서로 15개의 키를 트리에 삽입

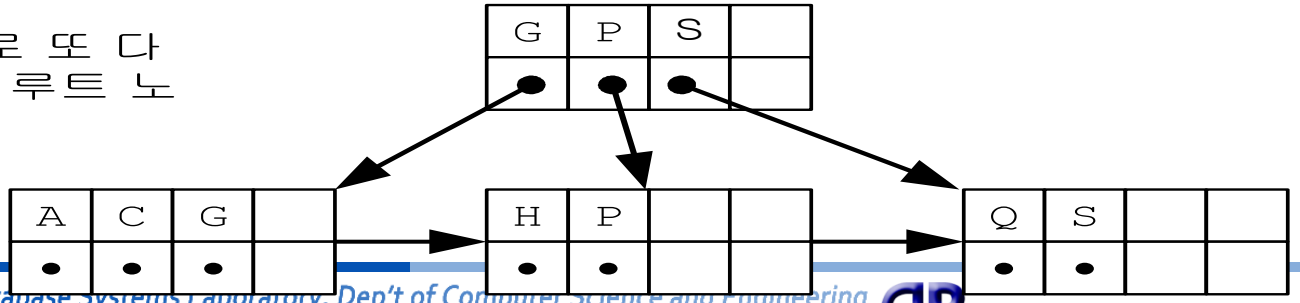
(a) 4개의 키를 삽입한 후에 루트 노트가 가득 참



(b) 다섯 번째 키의 삽입으로 루트 노드를 분할하고 새로운 루트 노드를 만듦

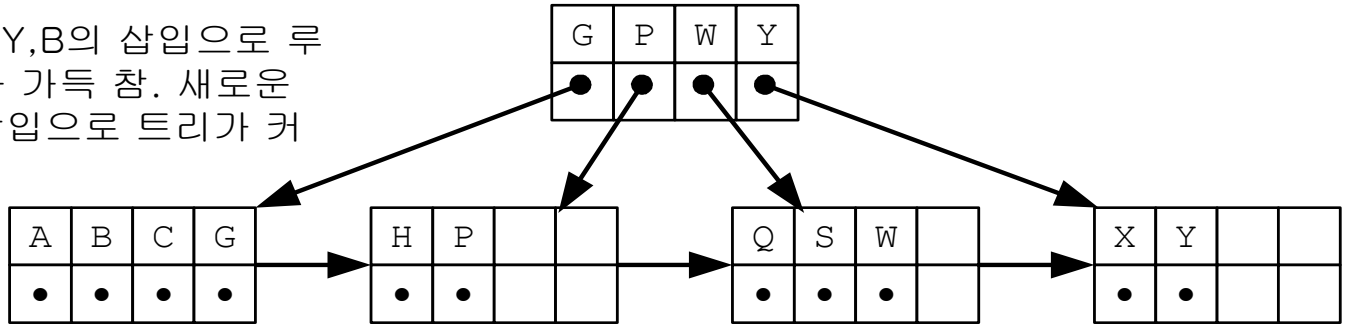


(c) A와 G의 삽입으로 또 다른 노드로 분할하고 루트 노드를 증가시킴

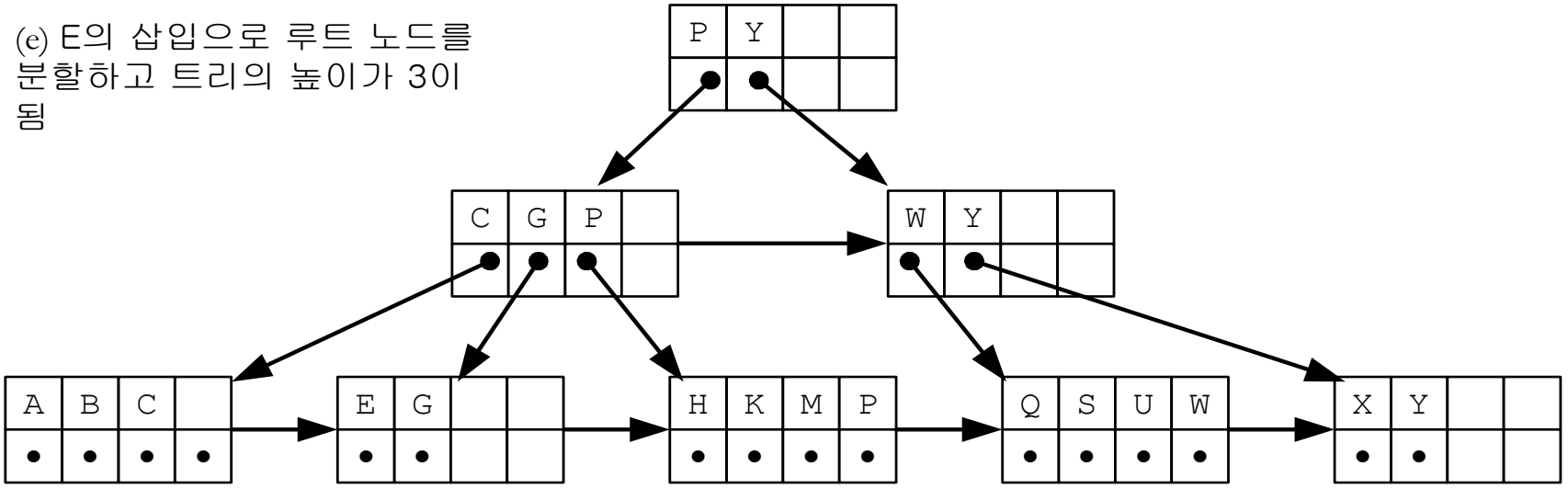


B+ 트리의 삽입 (cont'd)

(d) X, W, Y,B의 삽입으로 루트 노드가 가득 참. 새로운 키 E의 삽입으로 트리가 커져야 함



(e) E의 삽입으로 루트 노드를 분할하고 트리의 높이가 3이 됨



B+ 트리의 삭제

1. **노드에 남아 있는 키의 개수 < 최소 키 개수이면(언더플로우 조건)**
   1.1 원래 노드와 형제(sibling) 노드가 최소 키 개수를 가지기 위해, 형제 노드에서 한 개 이상의 키를 이동시키다. 이때, 형제 노드는 같은 부모를 가지는 인접한 노드를 뜻한다.
   1.2 만일 형제 노드들도 최소 키 개수를 가진다면, 형제 노드들 중에 하나와 원래의 노드를 한 개의 노드로 합병한다.

2. **만일 노드에 있는 가장 큰 키가 변경된다면, 자식 노드에서 가장 큰 키로 부모 노드에 있는 키를 바꿈**

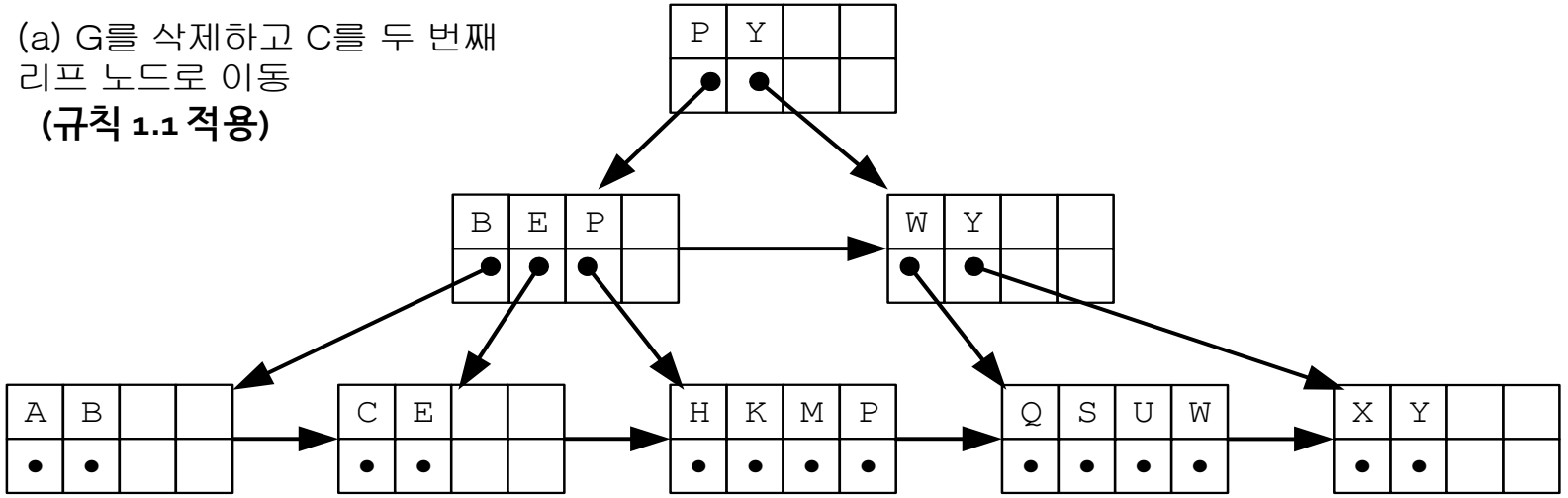
3. **만일 1단계의 결과로 노드 삭제되었을 경우, 부모 노드에서 관련된 키를 삭제하고 1-3단계를 반복**

4. **만일 루트 노드가 한 개의 자식 노드만을 가진다면, 루트 노드를 삭제하고 자식 노드를 루트 노드로 만들어 트리 높이를 1줄인다.**

B+ 트리의 삭제 (cont'd)

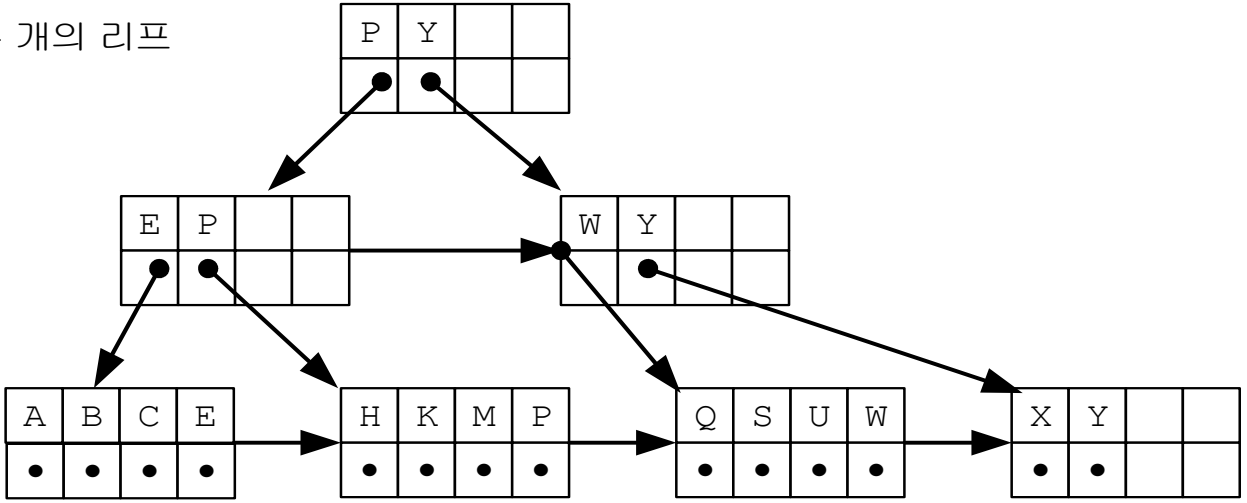
(a) G를 삭제하고 C를 두 번째 리프 노드로 이동

(규칙 1.1 적용)



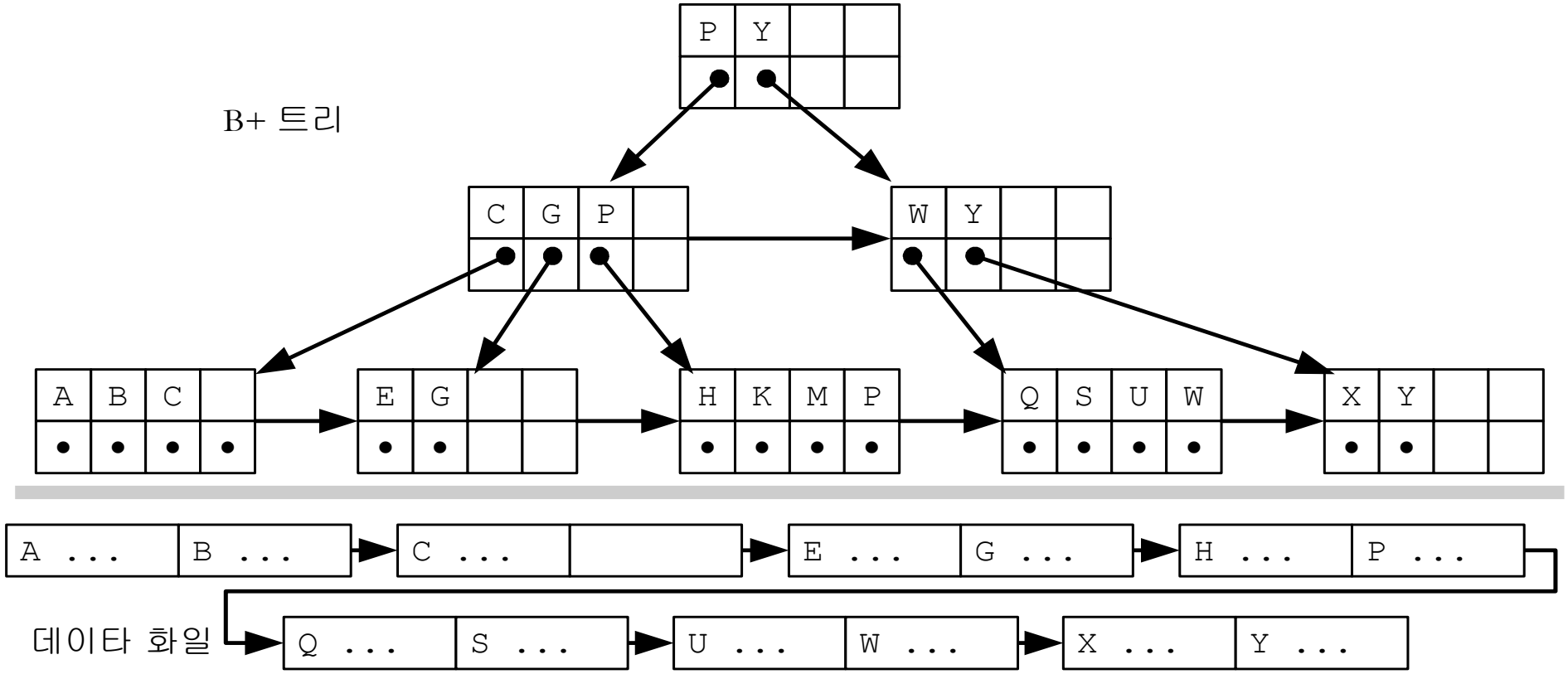
(b) G를 삭제하고 두 개의 리프 노드를 합병

(규칙 1.2 적용)



인덱스된 순차 화일과 B+ 트리

- 인덱스된 순차 화일(indexed sequential file)
  - 키 순서로 정렬되어 있으면서 기본 키에 대한 B+ 트리 구조를 가지는 화일
  - B+ 트리 인덱스와 정렬된 순차 화일의 조합



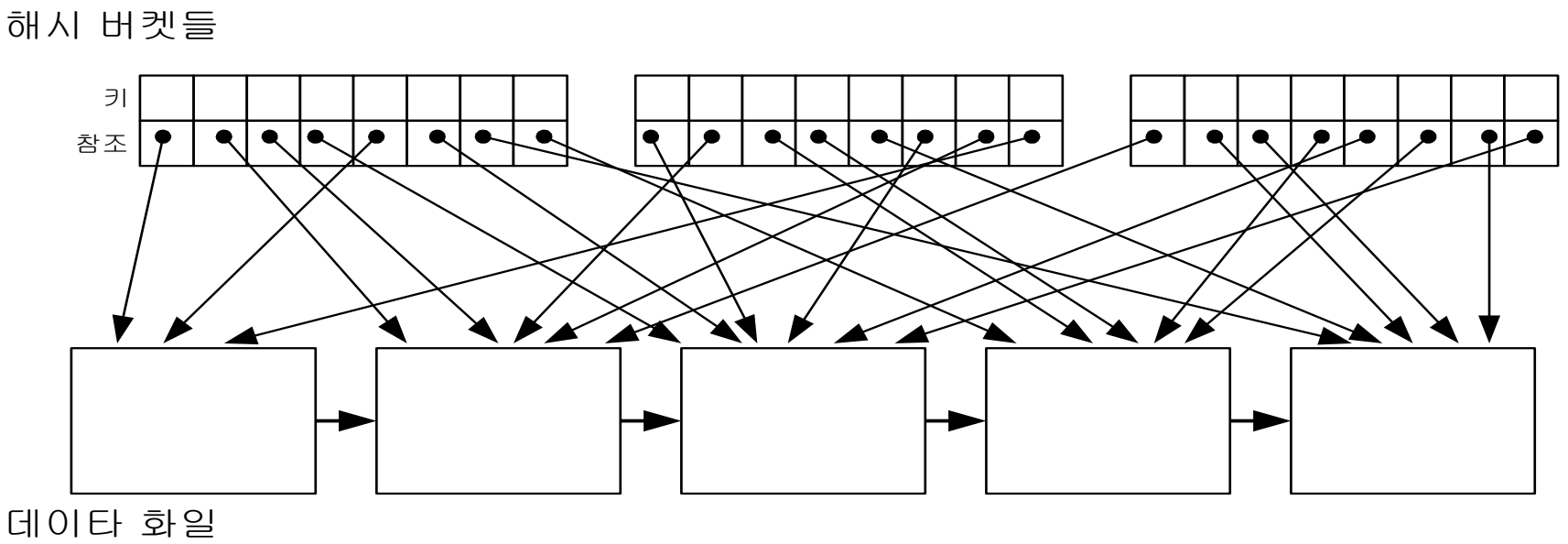
인덱스된 순차 파일의 삽입 알고리즘

1. B+ 트리에서 키에 대한 탐색
   - 데이터 파일에서 다음으로 가장 큰 키와 키의 주소를 검사
2. 다음으로 가장 큰 키의 블록과 동일한 블록 안에 데이터 레코드를 추가
3. 삽입에 따른 오버플로우는 블록을 분할하여 데이터 파일에 추가할 새로운 블록을 만듦
4. 새로운 데이타 레코드에 대한 키-참조 쌍을 B+ 트리에 삽입

- 인덱스된 순차 화일의 장점
  - 블록의 최소개수의 블록을 읽어서 특정 범위를 만족하는 키들의 모든 레코드들을 읽을 수 있음

해시 테이블을 이용한 인덱스 표현

- **해시 테이블 인덱스(hash table index)**
  - 해시 테이블(hash table)에 키-참조 쌍을 저장
  - 키-참조 쌍의 주소는 키 값을 해시 함수(hash function)에 적용하여 계산
    - 해시 함수 : 키 애트리뷰트의 도메인을 정수 값으로 매핑
  - 여러 개의 키-참조 쌍을 포함할 수 있는 블록인 버켓(bucket)으로 구성됨



해시 테이블을 이용한 인덱스 표현 (cont'd)

- 해시의 장점
  - 매우 빠른 검색 제공
  - 임의 키를 찾기 위해 기껏해야 한 번의 디스크 접근 필요

- 키들을 정렬하지 않으므로, 해시 테이블 범위에 있는 모든 키를 찾거나 키의 리스트를 순서대로 만드는데 사용될 수 없음

해시 함수

- 애트리뷰트 값을 주소로 매핑

- 좋은 해시 함수란?
  - 애트리뷰트 값들을 주소로 균등하게 분산시킴

- 일반적인 해시 함수
  - $h(k) = (a \times k + b) \bmod m$
    - m : 해시에 대한 값의 범위를 결정
    - a, b : 해시 값의 분포에 영향
  - 이 함수를 적용하기 위해서는 임의의 키를 숫자 값으로 변환하는 방법 필요

해시 테이블의 삽입

- **키-참조 쌍을 해시 테이블에 삽입하는 단계**
  1. 키를 해시 함수의 도메인 안에 있는 수로 변환하여 해시를 위한 키로 생성
  2. 키에 대한 해시 값을 계산
  3. 해시 값에서 버켓 주소를 결정
  4. 키–참조 쌍을 버켓에 삽입
  5. 버켓 오버플로우를 처리

- **충돌(collision)**
  - 두 개의 키가 같은 주소로 해시 되었을 때

물리적인 데이타베이스 특징 명세

- **물리적인 모델**
  - 시스템에 대한 성능과 신뢰성을 크게 향상 시킴
- **데이타베이스 관리자**
  - 데이타베이스에 대한 물리적인 구성을 책임
- **상업용 데이타베이스 시스템**
  - 데이타베이스 화일을 구성하고 테이블에 인덱스를 추가하는 기능 지원

SQL을 이용한 인덱스 생성

- SQL을 구현한 모든 상용 DBMS는 create index 문장을 지원
  - 상용 시스템에서는 성능 튜닝의 필요성을 중시

- Oracle8 DDL의 문법

```
create index customerAccountIdIndex on Customer (accountId);
create index customerLastNameIndex on Customer (lastName);
drop index customerLastNameIndex;
create index prevRentalAccountIdIndex on PreviousRental (accountId);
create index prevRentalAcctDataIndex
        on PrevousRental (accountId, dateRented);
```

SQL을 이용한 인덱스 생성 (cont'd)

- 예 : '1999년 5월 1일 이전에 특정 고객이 빌린 모든 비디오를 검색하시오'

```
select  lastName, firstName, videoId
from  Customer  c,  PreviousRental  p
        where  dateRented  < '01-may-1999' and c.accountId = p.accountId
```

  - 모든 고객에 대해 반복적으로 실행
  - 각 고객에 대해 날짜 조건을 만족하는 PreviousRental의 모든 행을 선택하기 위해 preRentalAcctDataIndex 이용