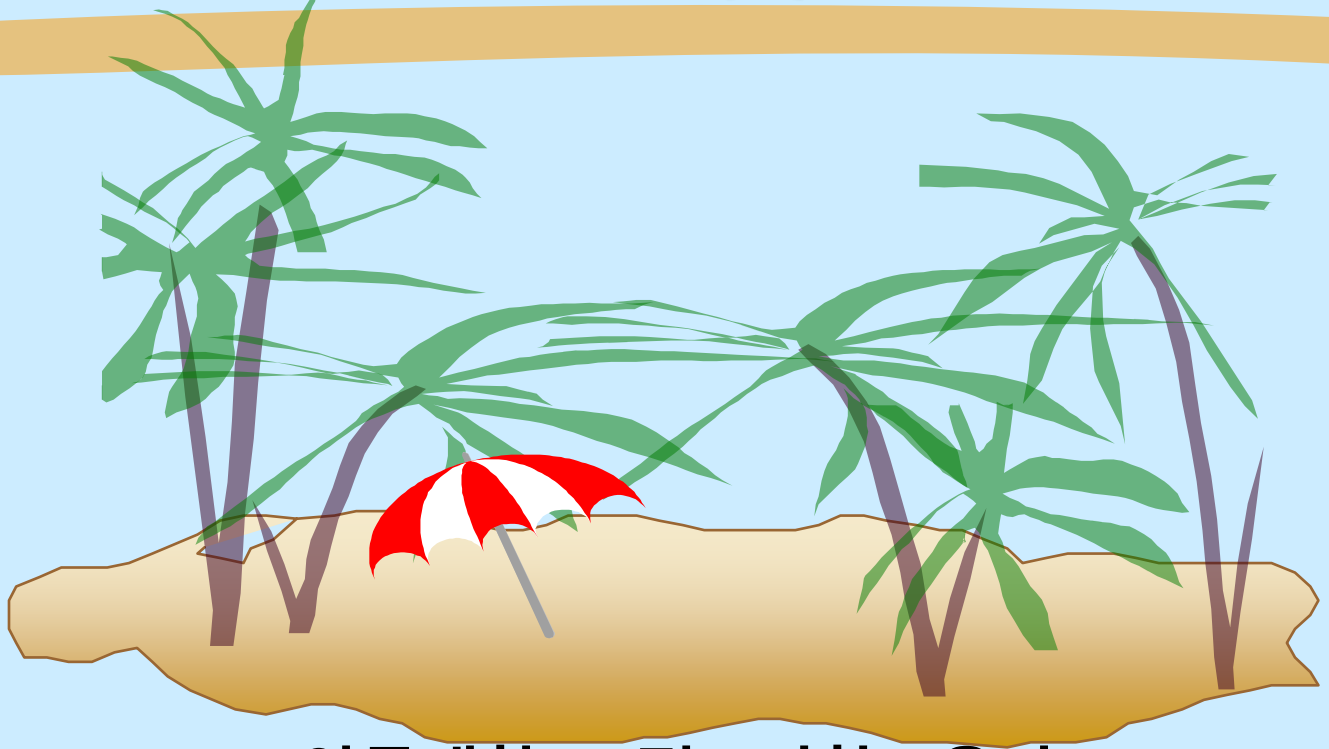
질의 처리 및 최적화

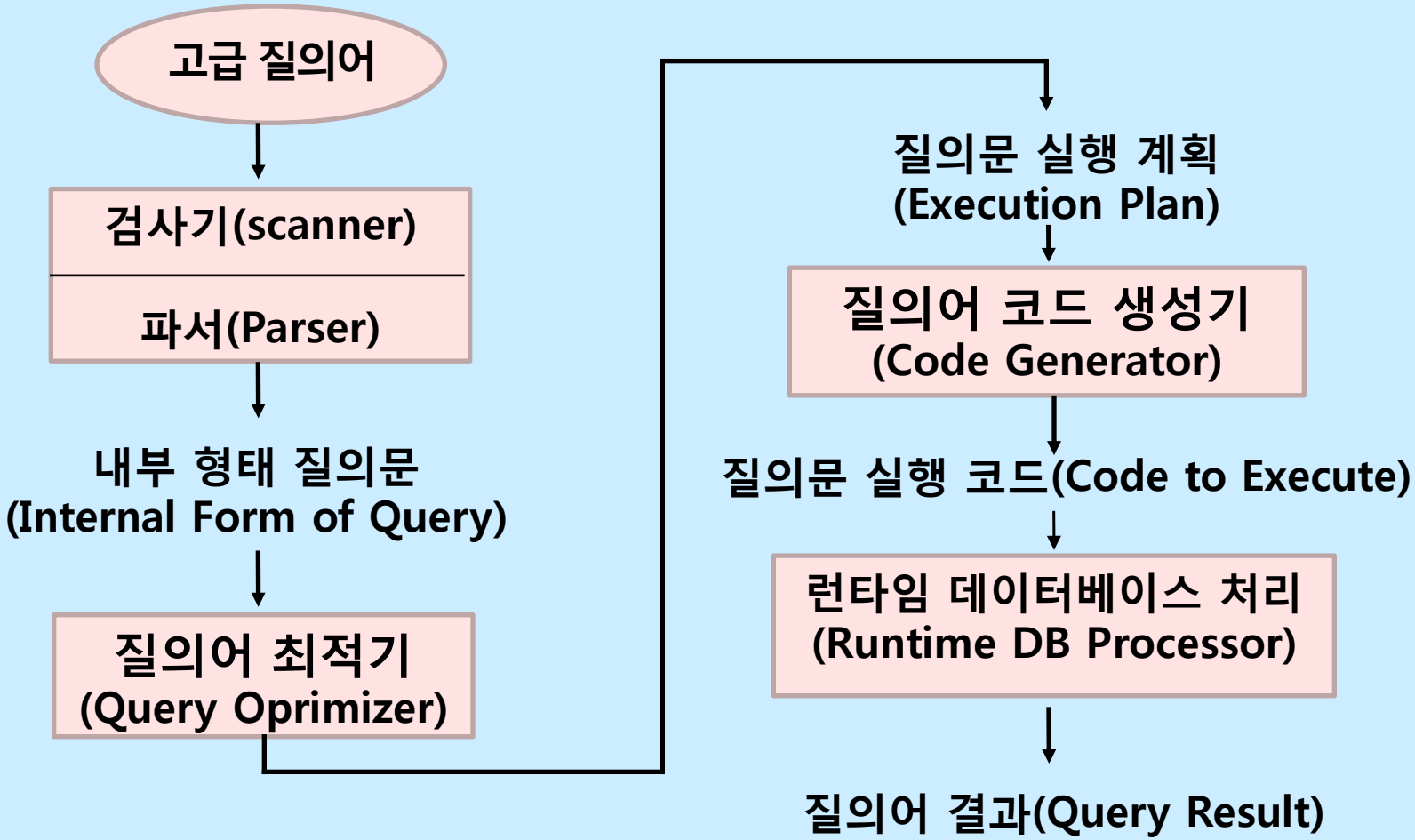


안동대학교 정보과학교육과

배울 내용

- 질의어 처리 단계
- 질의어 최적화
- 내부 형태 변환 규칙
- 관계대수 연산자 구현

질의어 처리 단계(Query Processing Steps)



질의어 최적화(Query Optimization)

- 최적화(Optimization)
  - 효율적인 실행 전략의 계획
  - 최적화 : 더 효율적인 실행
  - 시스템 레벨의 최적화 : 고급 질의어
- 성능측정(Performance metrics)
  - 디스크 I/O 횟수(튜플, 페이지, 블럭…)
  - 중간결과의 크기(Size of intermediate result)
  - 응답시간(Response time)
- 질의어 최적화 과정(Stages in Query Processing)
  - ① 질의문의 내부표현
  - ② 효율적인 내부 형태로 변환
  - ③ 후보 프로시저 선정
  - ④ 질의문 계획의 평가 및 결정

질의어 최적화 과정: 1) 질의문의 내부 표현(1/3)

- 내부표현(Internal Representation)
  - 사용자의 질의문을 컴퓨터가 처리하기에 적절한 어떤 내부 형태로 변환
  - 사용자가 질의문을 작성하기에 편리하도록 첨가시킨 부수적인 구문이라든가 실제 처리하는 데는 필요하지 않을 것을 제거하는 작업 포함

- 형식론(Formalism)
  - 시스템의 질의어로 표현할 수 있는 것은 모두 이 형식으로 표현 가능해야 함
  - 다음 최적화 단계에서 편향된 영향을 주지 않고 중립적인 것이어야 함
  - 관계대수나 관계해석 등 여러가지 가능

질의어 최적화 과정: 1) 질의문의 내부 표현(2/3)

- 질의문 트리(Query tree)
  - 질의문 내부표현형태는 보통 트리 구조로 표현가능
  - 관계대수(algebraic expression) 사용
  - 하나의 질의문 트리는 하나의 질의문을 표현
  - 단말노드(Leaf nodes) : 피연산자(operands)인 릴레이션
  - 내부노드(Internal nodes) : 관계 대수 연산자(operators)

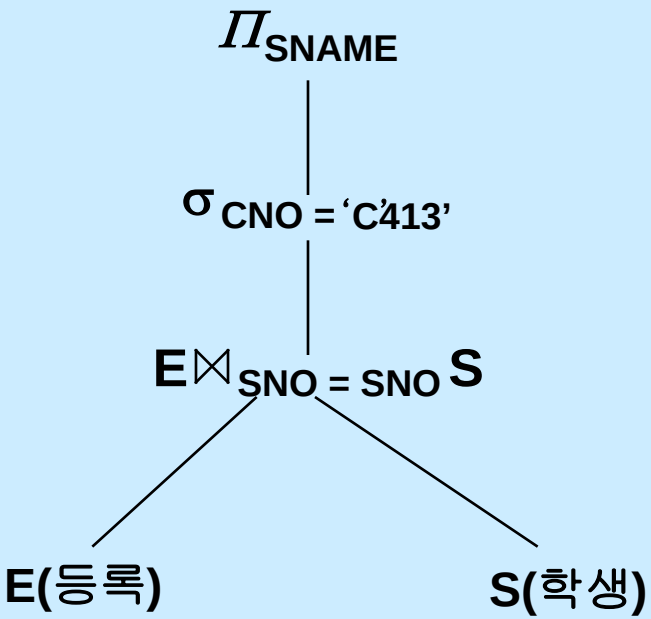
- 질의문 트리의 실행(Execution of the query tree)
  - 필요한 피연산자가 모두 사용 가능한 서브트리에 대해 먼저 내부노드 연산을 실행
  - 그 결과 릴레이션을 실행된 서브트리와 대체
  - 루트노드 실행 : 질의문의 실행 결과

질의어 최적화 과정: 1) 질의문의 내부 표현 (3/3)

질의문 트리의 예)

"과목 'C413'에 등록한 학생의 이름(SNAME)을 검색하라"

$$\Pi_{sname}(\sigma_{cno='C413'}(E \bowtie_{sno=sno} S))$$



S(학생)

S

SNO	SNAME	YEAR	DEPT

E(등록)

E

SNO	CNO	GRADE	MIDTERM	FINAL

질의어 최적화 과정: 2) 효율적 내부형태로 변환

- 일반적으로 하나의 질의문을 표현하는 관계대수식,
  즉 질의문 트리는 하나 이상 있을 수 있음
  - 효율성(Efficiency)
  - 질의문 처리의 성능(Cost of query processing)

- 변환규칙(Transformation rules)
  - 질의문 내부 표현을 동등하면서도 처리에 효율적인 형태로 변환시킨다.
    - 문법(Syntax)
    - 추론(Inference)
    - 의미(Semantics)
  - 예
    - $\sigma_{Sc}(R \bowtie S) \rightarrow R \bowtie (\sigma_{Sc}(S))$

▶ 예 (1/2)

```
SELECT   SNAME
FROM     S, E
WHERE    S.SNO=E.SNO   AND   CNO='C413'
```

|S|=100, |E|=10000, |E.CNO='C413'|=50
메인메모리는 50개의 튜플을 가질수 있다고 가정

S(학생)

S

SNO	SNAME	YEAR	DEPT

E(등록)

E

SNO	CNO	GRADE	MIDTERM	FINAL

▶ 예 (2/2)

❑ 방법 1

① $R1 \leftarrow E \bowtie_{SNO=SNO} S$
No. of tuple I/O = 10000(E)+100(S)*10000+10000(R1)

② $R2 \leftarrow \sigma_{CNO='C413'}(R1)$
No. of tuple I/O = 10000(R1)

③ $\pi_{SNAME}(R2)$

➡ No. of tuple I/O : 1030000

❑ 방법 2

① $R1 \leftarrow \sigma_{CNO='C413'}(E)$
No. of tuple I/O = 10000(E)

② $R2 \leftarrow R1 \bowtie_{SNO=SNO} S$
No. of tuple I/O = 100(S)

③ $\pi_{SNAME}(R2)$

➡ No. of tuple I/O : 10100

질의어 최적화 과정: 3) 후보 프로시저 선정 (1/2)

- 이 질의문 계획의 기본 전략은 주어진 최종 내부 표현을 일련의 저급 연산, 다시 말해 조인 프로시저, 셀렉트 프로시저 등으로 명세 하는 것

- 최종 내부표현을 실제로 어떻게 실행시킬 것인가 하는 질의문 계획 결정시 고려해야 할 사항
  - 인덱스나 기타 다른 접근 경로의 존재여부
  - 저장 데이터 값의 분포
  - 레코드들의 물리적 집중

- 각 저급 연산에 대해 최적기는 보통 미리 구현시켜 놓은 몇 개의 프로시저들을 포함

질의어 최적화 과정: 3) 후보 프로시저 선정 (2/2)

- 최적기는 내부 표현에 사용된 각 연산자에 대해 하나 이상의 후보 프로시저 선정 가능
  - 접근 경로 선택 : 특정 레코드를 찾기 위한 구조
    - 조인 프로시저, 셀렉트 프로시저, ...
    - 이 일련의 저급 연산들은 상호 의존성을 가짐
  - 미리 정의된 프로시저 고려
    - 예 : Selection
      - 후보키(candidate key) 비교에 기초한 프로시저
      - 인덱스 필드에 기초한 프로시저
      - 물리적으로 집중된 프로시저

질의어 최적화 과정: 4) 질의문 계획의 평가 및 결정 (1/2)

- 질의문을 실행할 수 있는 후보 질의문 계획을 평가하고 그 중에서 최상의, 즉 최소 비용의 계획을 결정하는 것
  - 각 질의문 계획은 최적화된 내부 표현에 있는 각 연산에 대해 하나의 실행 프로시저를 골라 이들을 모두 조합해서 구성

- 질의문 하나에 대해 보통 여러 개의 후보 질의문 계획 존재 가능
  - 제한된 수의 질의문 계획을 생성
  - 탐색공간의 축소
  - 가장 비용이 적게 드는 계획을 선택

질의어 최적화 과정: 4) 질의문 계획의 평가 및 결정 (2/2)

- 비용식(Cost Formula)
  - 디스크 입출력 비용
    - 디스크에 있는 데이터 블록을 판독하고 기록하는 비용
  - 저장 비용
    - 질의문을 실행하는 과정에서 생성되는 중간 결과를 저장하는데 드는 비용
  - 계산 비용
    - 정렬, 조인을 위한 병합, 그리고 필드 값에 대한 계산 포함
  - 통신 비용
    - 질의문을 처리한 결과를 송신하고 수신하는 데 드는 비용뿐 만 아니라 질의문 실행을 위해 필요한 경우 다른 곳으로부터의 데이터 전송 비용까지도 모두 포함

  ➔ 주로 디스크 입출력의 횟수를 고려

내부형태 변환 규칙 (1/4)

효율적이면서 동등한 관계 대수식으로 변환

R1. 논리곱으로 연결된 선택조건 → 일련의 개별적인 선택 조건

$$\sigma_{c1\ AND\ c2\ AND \cdots\ cn}(R) \equiv \sigma_{c1}(\sigma_{c2}(\ldots(\sigma_{cn}(R))\ldots))$$

R2. 선택연산은 교환적

$$\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$$

R3. 연속적인 프로젝트 연산(P) → 마지막 것만 실행

$$P_1(P_2(\ldots(P_n(R))\ldots)) \equiv P_1(R)$$

R4. 셀렉트의 조건 c가 프로젝트 애트리뷰트만 포함하고 있다면 이들은 교환적

$$\sigma_c(P(R)) \equiv (P(\sigma_c(R)))$$

R5. 셀렉트의 조건이 카티션 프로덕트(×)에 관련된 릴레이션 하나에만 국한 → 조인조건

$$\sigma_c(R \times S) \equiv R \bowtie_c S$$

$$\sigma_{c1}(R \bowtie_{c2} S) \equiv R \bowtie_{c1 \wedge c2} S$$

내부형태 변환 규칙 (2/4)

R6. 셀렉트의 조건이 조인 또는 카티션 프로덕트에 관련된 릴레이션 하나와만 관련이 되어있을 때

- $\sigma_c (R \bowtie S) \equiv \sigma_c(R) \bowtie S$
- $\sigma_c (R \times S) \equiv \sigma_c(R) \times S$

R7. c_1은 릴레이션 R과 관련되어 있고, c_2는 릴레이션 S와 관련이 되어 있을때 $c = (c_1 \text{ AND } c_2)$

- $\sigma_c (R \bowtie S) \equiv (\sigma_{c1} (R)) \bowtie (\sigma_{c2}(S))$
- $\sigma_c (R \times S) \equiv (\sigma_{c1} (R)) \times (\sigma_{c2}(S))$

R8. $\times$, $\cup$, $\cap$, $\bowtie$ 는 교환적

- $R \times S \equiv S \times R$
- $R \cup S \equiv S \cup R$
- $R \cap S \equiv S \cap R$
- $R \bowtie S \equiv S \bowtie R$

내부형태 변환 규칙 (3/4)

R9. L_1은 릴레이션 R에 관련되어있고, L_2는 릴레이션 S에 관련되어 있을때 $L=(L_1, L_2)$

- $\Pi_L(R \bowtie S) \equiv (\Pi_{L1}(R)) \bowtie (\Pi_{L2}(S))$
- $\Pi_L(R \times S) \equiv (\Pi_{L1}(R)) \times (\Pi_{L2}(S))$

R10. 집합연산과 관련된 셀렉트의 변환

- $\sigma_c(R \cup S) \equiv \sigma_c(R) \cup \sigma_c(S)$
- $\sigma_c(R \cap S) \equiv \sigma_c(R) \cap \sigma_c(S)$
- $\sigma_c(R-S) \equiv \sigma_c(R) - \sigma_c(S)$

R11. 합집합과 관련된 프로젝트의 변환

- $\Pi(R \cup S) \equiv (\Pi(R)) \cup (\Pi(S))$

내부형태 변환 규칙 (4/4)

R12. $\cup$, $\cap$, $\times$, $\bowtie$ 는 연합적

- $(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$
- $(R \cup S) \cup T \equiv R \cup (S \cup T)$
- $(R \cap S) \cap T \equiv R \cap (S \cap T)$
- $(R \times S) \times T \equiv R \times (S \times T)$

R13. OR로 연결된 조건식을 AND로 연결된 논리곱 정형식(conjunctive nomal form)으로 변환

- C_1 OR (C_2 AND C_3) → (C_1 OR C_2) AND (C_1 OR C_3)

초기 트리를 최적화 된 트리로의 변환방법

① 논리곱으로 된 조건을 가진 셀렉트 연산은 분해, 일련의 개별적 셀렉트 연산으로 변환

② 셀렉트 연산의 교환법칙을 이용해서 셀렉트 연산을 트리의 가능한 한 아래까지 내림

③ 가장 제한적인 셀렉트 연산이 가장 먼저 수행될 수 있도록 단말 노드를 정렬

- 가장 적은 투플 수 또는 가장 작은 선택도(selectivity)

④ 카티션 프로덕트와 해당 셀렉트 연산을 조인연산으로 통합

⑤ 프로젝트 연산은 가능한 한 프로젝트 애트리뷰트를 분해하여 개별적 프로젝트로 만들어 이를 먼저 실행할 수 있도록 트리의 아래로 내림

⑥ OR로 연결된 조건식은 논리곱 정형식으로 변환

- 중간결과물의 최소화
- 논리곱 정형식을 통한 쉬운 failure detect방법

예(I)

EMPLOYEE	SSN	NAME	BDATE	ADDR	SAL	DNO	SEX	SUPERSSN

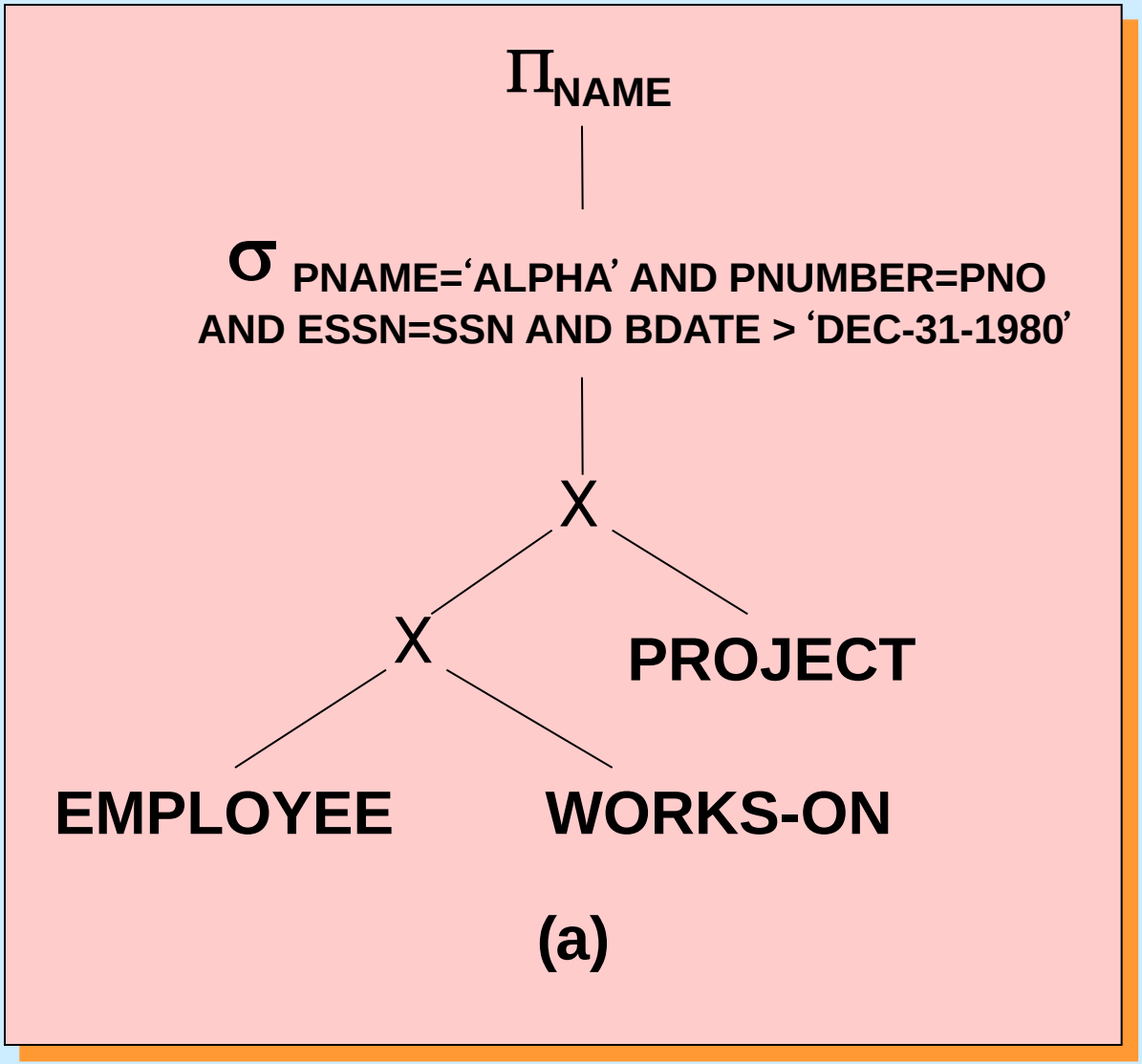
PROJECT	PNUMBER	PNAME	PLOCATION	DNAME

WORKS-ON	ESSN	PNO	HOURS

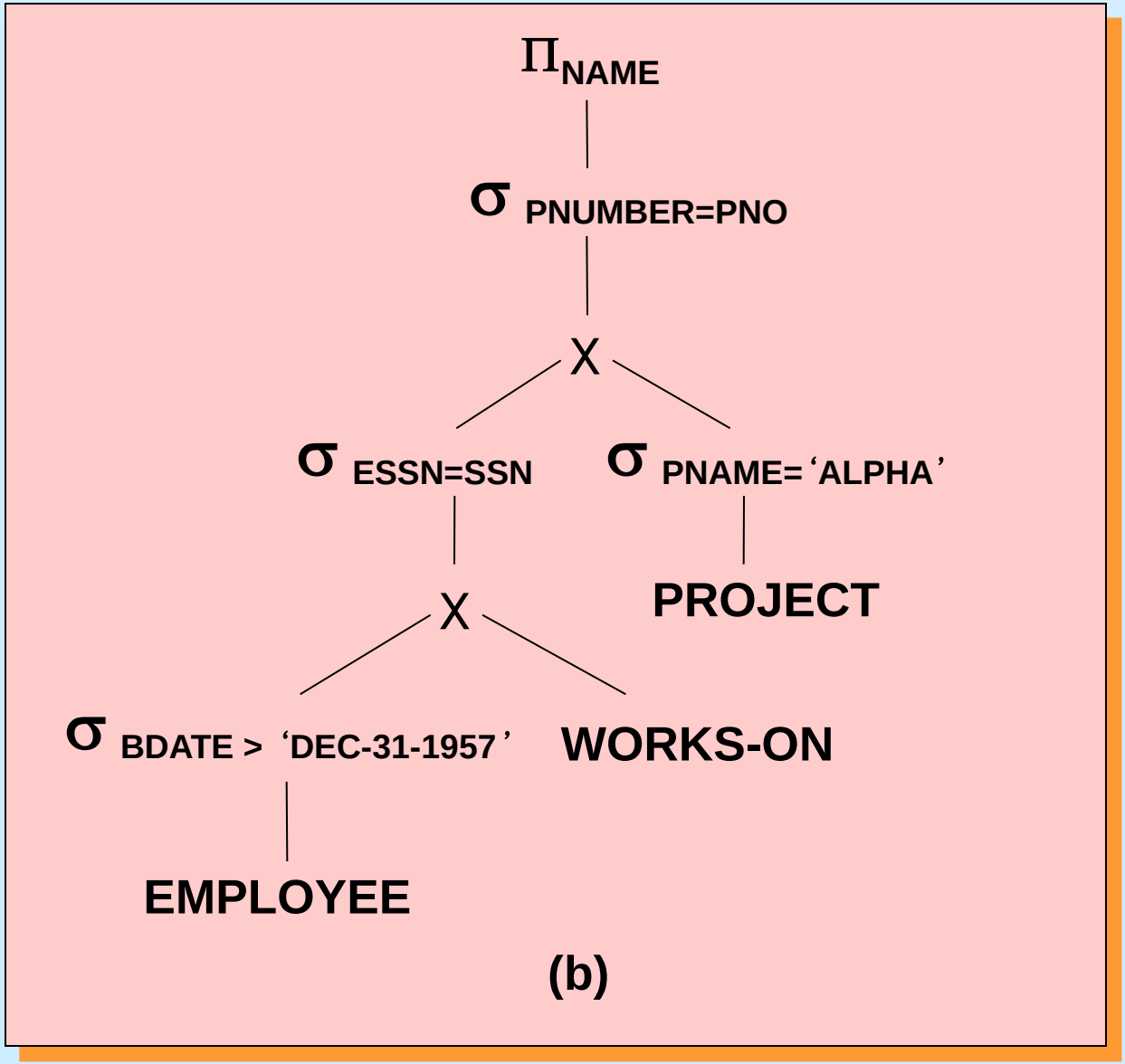
프로젝트 ALPHA에 참여한 사람중 1957년 이후에 태어난 사람의 이름을 셀렉트하라

```
SELECT NAME
FROM EMPLOYEE, WORKS-ON, PROJECT
WHERE PNAME = 'ALPHA'
  AND PNUMBER = PNO
  AND ESSN = SSN
  AND BDATE > 'DEC-31-1957'
```

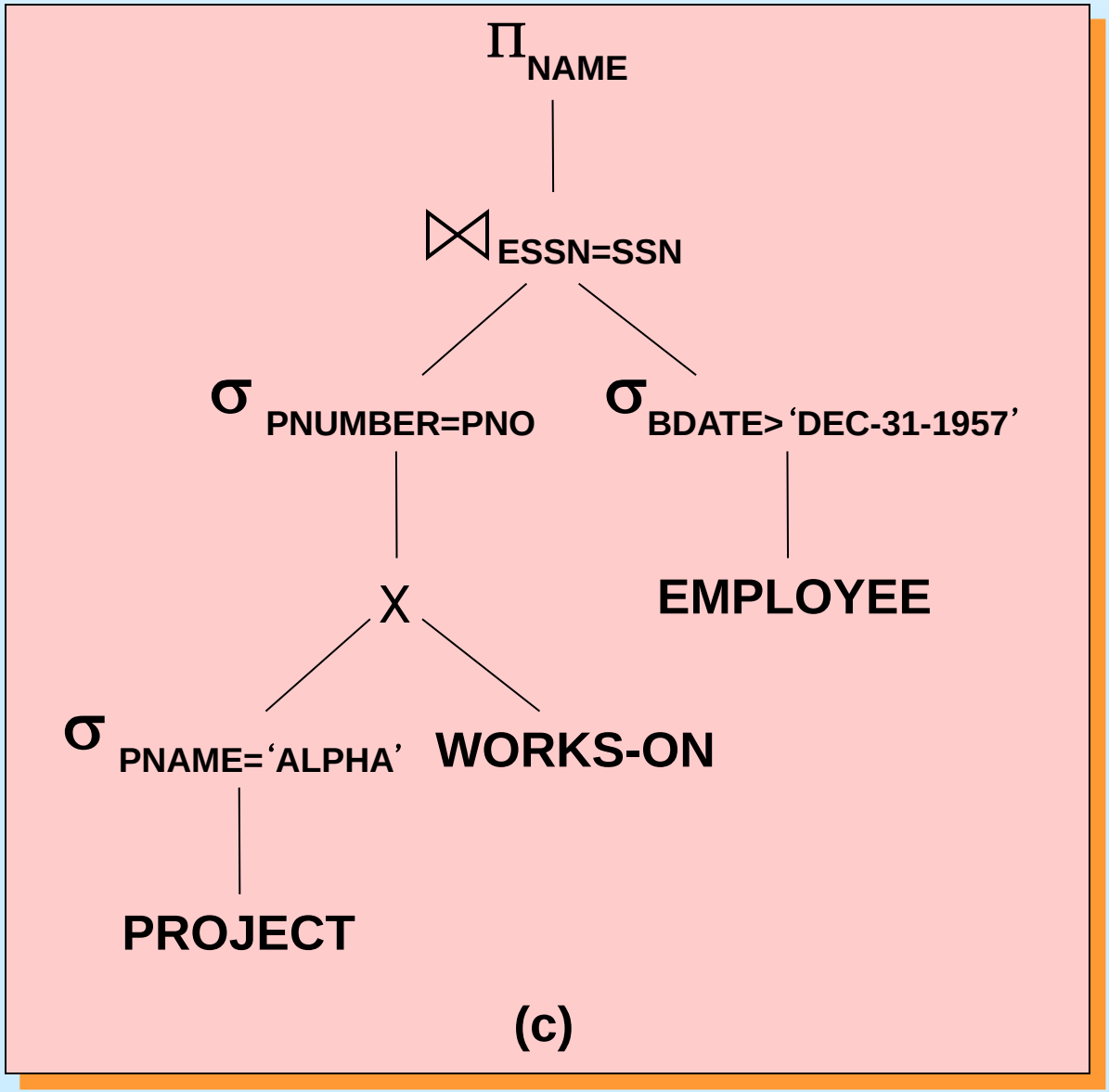
예(II)


Π_{NAME}
σ PNAME='ALPHA' AND PNUMBER=PNO
AND ESSN=SSN AND BDATE > 'DEC-31-1980'
X
X
PROJECT
EMPLOYEE
WORKS-ON
(a)

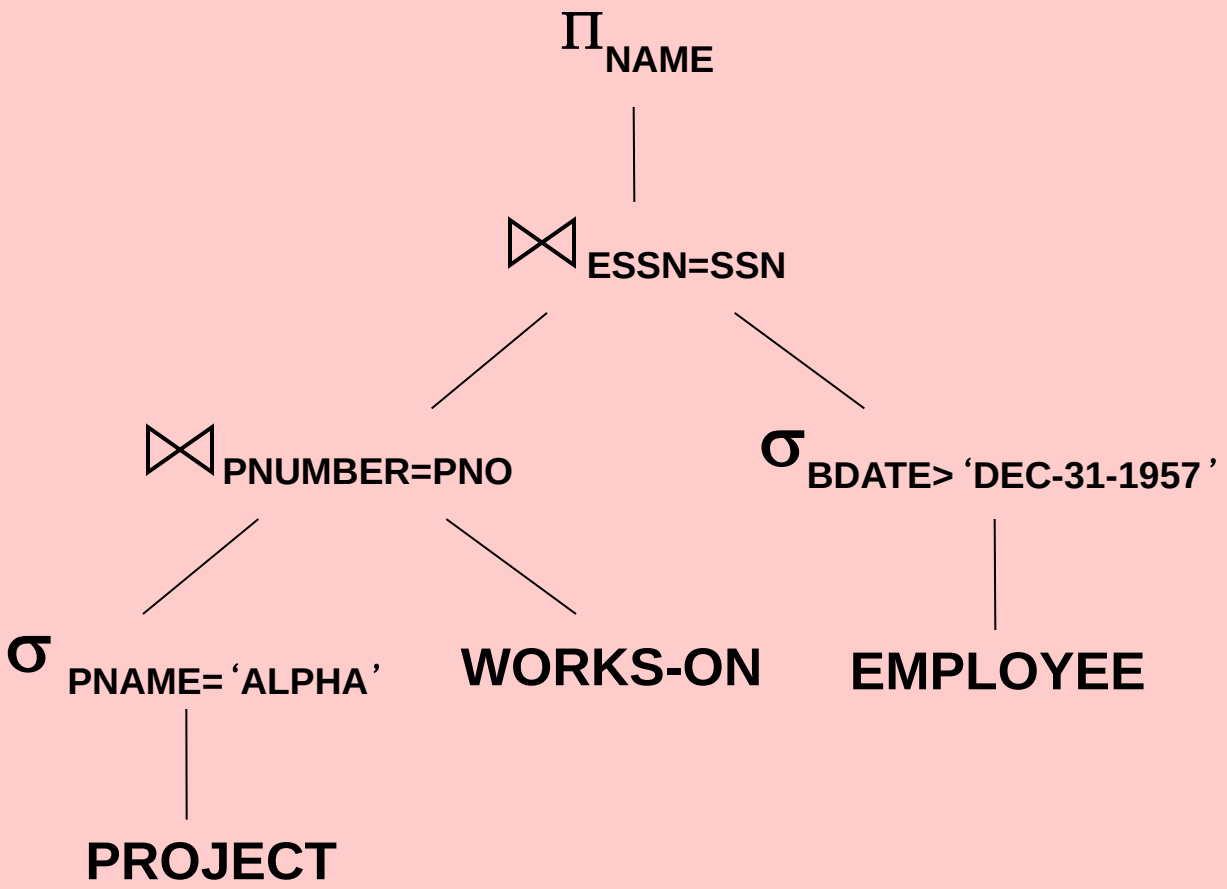
예(III)


Π_{NAME}
σ PNUMBER=PNO
X
σ ESSN=SSN
σ PNAME= 'ALPHA'
X
PROJECT
σ BDATE > 'DEC-31-1957'
WORKS-ON
EMPLOYEE
(b)

예(IV)

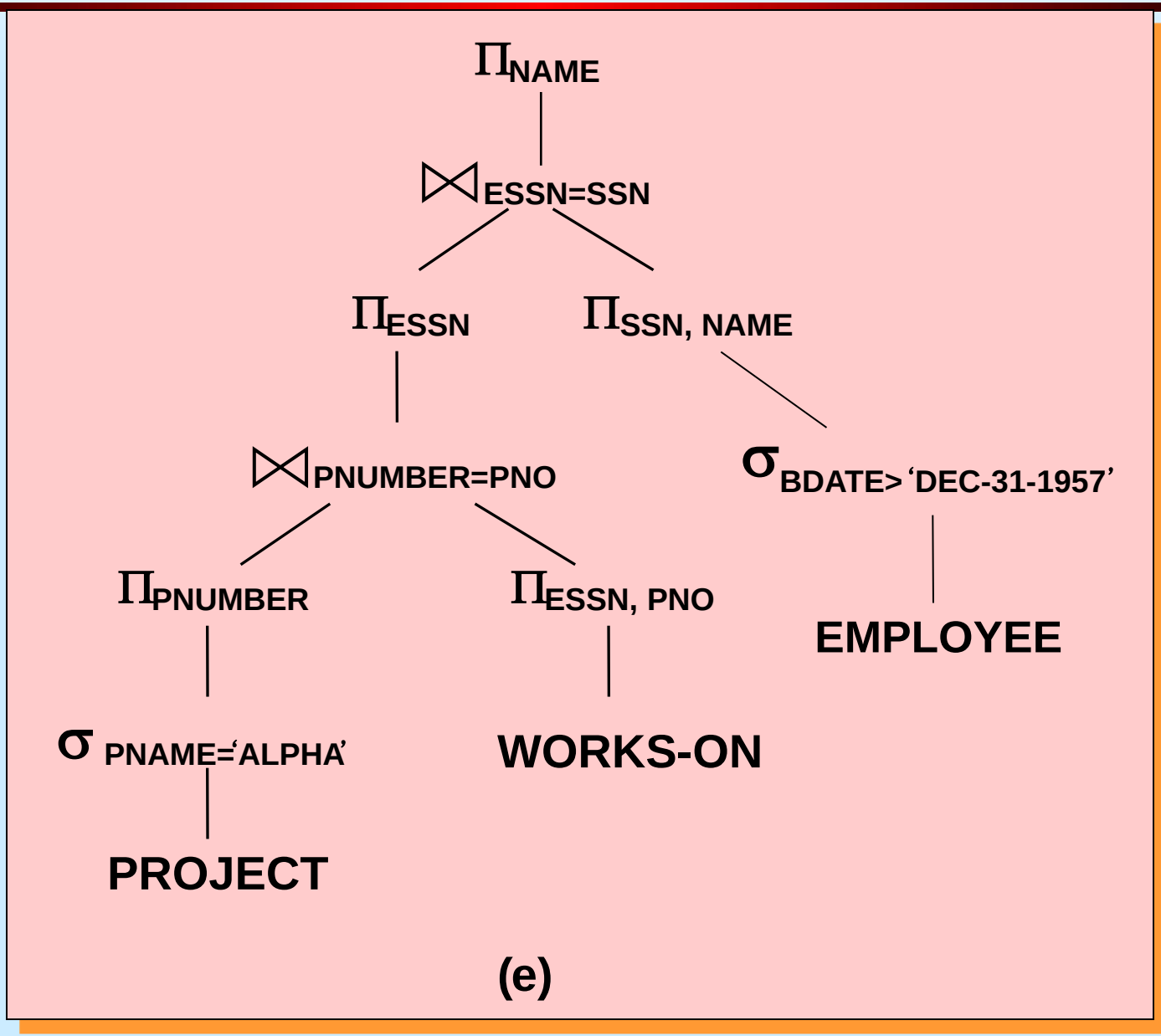

π NAME
⋈ ESSN=SSN
σ PNUMBER=PNO
σ BDATE> 'DEC-31-1957'
X
EMPLOYEE
σ PNAME='ALPHA'
WORKS-ON
PROJECT
(c)

예(V)


Π_{NAME}
$\bowtie_{ESSN=SSN}$
$\bowtie_{PNUMBER=PNO}$
$\sigma_{BDATE>'DEC-31-1957'}$
$\sigma_{PNAME='ALPHA'}$
WORKS-ON
EMPLOYEE
PROJECT

(d)

예(VI)


Π_{NAME}
$\bowtie_{ESSN=SSN}$
Π_{ESSN}
$\Pi_{SSN, NAME}$
$\bowtie_{PNUMBER=PNO}$
$\sigma_{BDATE> 'DEC-31-1957'}$
$\Pi_{PNUMBER}$
$\Pi_{ESSN, PNO}$
EMPLOYEE
$\sigma_{PNAME='ALPHA'}$
WORKS-ON
PROJECT
(e)

관계 대수 연산자의 구현(I)

- 질의문에서 쓰인 관계 대수 연산을 구현한 저급 연산 프로시저의 필요
- 관계 연산자
  - 관계 대수 연산
  - 조합 연산
  - 집단 함수의 실행 루틴
- 각 연산의 접근 루틴은 여러 가지로 구현 가능
- 하나의 접근 루틴은 특정 저장구조와 접근 경로에 적용

관계 대수 연산자의 구현(II)

- 셀렉트 연산의 구현
  - 선형 탐색 : 주먹구구식 방법(brute force)
  - 이원 탐색 : 동등 비교, 정렬된 데이터
  - 기본 인덱스 또는 해싱키를 통해 하나의 레코드를 탐색
  - 기본 인덱스를 이용해서 복수 레코드를 탐색

    >, ≥, <, ≤
  - 집중 인덱스를 이용해서 복수레코드를 탐색
  - 보조 (B+ tree) 인덱스 이용
    - 유일성 인덱스(key index field) : 단일레코드
    - 비유일성 인덱스(non-key index field) : 복수레코드

관계 대수 연산자의 구현(III)

- 조인 연산의 구현 : 특정 속성값을 기준으로 투플들을 그룹화
  - 중첩루프(Nested Loop)
  - 인덱스 검사(Index Lookup)
  - 해시 검사(Hash Lookup)
  - 정렬/합병(Sort/Merge)
  - 해싱(Hashing)
  - 종합(Hybrid method)

(1) 중첩루프

R : 외부 릴레이션
S : 내부 릴레이션
R.A = S.A

```
/*  |R| = n,  |S| = m */
do i := 1 to n;                  /* outer loop */
   do j := 1 to m;               /* inner loop */
      if R[i].A = S[j].A then
         add R[i]·S[j] to result;
   end;
end;
```

(2) 인덱스 검사

```
 /* assume index X on S.A */
do i := 1 to n;                    /* outer loop */
     /* let there be k index entries X[1], X[2], ... X[k] */
     /* with indexed attribute value = R[i].A                    */
     do j := 1 to k;               /* inner loop */
          /* let tuple of S indexed by X[j] be S[j] */
          add R[i]·S[j] to result;
     end;
end;
```

R : 외부 릴레이션
S : 내부 릴레이션
R.A = S.A

Note : 인덱스는 영구적이거나 임시적 이다.

(3) 해시 검색(hash lookup)

```
/* assume hash table H on S.A */
do i := 1 to n;                    /* outer loop */
    k := hash(R[i].A);
    /* let there be h tuples S[1], S[2], ... S[h] stored at H[k] */
    do j := 1 to h;                /* inner loop */
        if S[j].A = R[i].A then
            add R[i]·S[j] to result;
    end;
end;
```

R : 외부 릴레이션
S : 내부 릴레이션
R.A = S.A

(4) 정렬/합병

```
/* assume R and S are both sorted on attribute A ;  */
/* following code assume join is many-to-many ;     */
r := 1 ;
s := 1 ;
do while r ≤ n and s ≤ m ;                    /* outer loop */
      v := R[r].A ;
      do j := s by 1 while S[j].A < v ;
      end ;
      s := j ;
      do j := s by 1 while S[j].A = v ;       /* main inner loop */
            do i :=r by 1 while R[i].A = v ;
                  add R[i].S[j] to result ;
            end ;
      end ;
      s := j ;
      do i := r by 1 while R[i].A = v ;
      end ;
      r := i ;
end ;
```

R : 외부 릴레이션
S : 내부 릴레이션
R.A = S.A

(5) 해싱

```
/* build hash table H on S.A */
do j := 1 to m;
    k := hash(S[j].A);
    add S[j] to hash table entry H[k];
end;

/* Continue with hash lookup */
```

R : 외부 릴레이션
S : 내부 릴레이션
R.A = S.A

관계 대수 연산자의 구현(IV)

- 프로젝트 연산의 구현
  - 프로젝트 속성에 키가 포함되면
    - 원 릴레이션 투플 수와 동일한 개수의 투플 포함
  - 프로젝트 속성에 키가 포함되지 않으면
    - 중복 투플 제거 작업 추가 수행이 필요함

프로젝트 연산자의 구현

```
/* Project attribute list : attr-list */
Do i := 1 to n ;                          /*  |R| = n */
    add R[i][attr-list] to T' ;    /* project on attr-list */
end ;

if (R.key ∈ attr-list)    /* no duplicates */
then T := T' ;
else
    sort tuples of T' ;
    set i := 1, j := 2 ;
    do while i ≤ n ;
        add the tuple T'[i] to T ;
        do while T[i] = T'[j] ;   /* check duplicates */
            j := j+1;                    /* remove duplicates */
        end ;
        i := j ;     j := j+1 ;
    end ;
end ;
/* T contains the project result without duplication */
```

비용 함수 (1/2)

- 질의문 계획의 비용을 계산
- 비용 함수가 필요로 하는 정보를 유지 : 카탈로그
  - 파일의 크기
  - 레코드의 수 (r), 블록 수 (b), 그리고 파일에 대한 블록 인수 (b_f)
  - 기본 접근 방법과 기본 접근 애트리뷰트
    - 파일의 레코드들로 정렬이 되어 있지 않거나 키 애트리뷰트로 인덱스나 해시되어 정렬되어 있을 수 있다
    - 다단계 인덱스에 대해서는 단계 수 (x)가 필요 – 이 정보들은 질의문 실행 시 블록 접근 수를 계산하는데 필요한 매개변수가 됨
  - 저장된 인덱스 애트리뷰트 값에 대해 서로 다른 상이한 값의 수 (d)
    - 검색 투플 수 (s), 즉 이 속성값을 선택조건으로 할 때 만족하는 평균 투플 수를 계산할 때 필요함

    예)

      - 모든 키 애트리뷰트 : s=1
      - 키가 아닌 애트리뷰트 : s=r/d

비용 함수 (2/2)

- 디스크 입출력 수만 고려

- 비용(Cost)
  - 선형 탐색(Linear search)
    - 키 아닌 애트리뷰트(Non-key attribute(multiple records)): b
    - 키 애트리뷰트(Key attribute): b/2
  - 이원 탐색(Binary search) : $\log_2 b$
  - 기본 인덱스 이용 : x+1
  - 해싱 함수를 이용한 비용 : 1

의미적 질의어 최적화

- 구문 변환 규칙과 스키마의 제약조건을 혼용하여 질의문을 변환하는 방법

- 예

```
SELECT   SNAME
FROM     S
WHERE    YEAR≥5

무결성 제약조건 :  1≤YEAR≤4
⇒ 결과가 공백임을 예측 가능
```

Π_{CNO} (S $\bowtie_{SNO=SNO}$ E)

① SNO 는 S의 기본인덱스
② SNO 는 E의 외래키이고 널이 아님
⇒ Π_{CNO} (E)

요약

- 질의어 처리 과정
- 질의어 최적화
- 관계대수 연산자 구현

다음 배울 내용 : 트랜잭션