데이터베이스

연산자의 구현

학습목표

- SELECT, PROJECT 연산자의 동작 원리를 설명할 수 있다.
- 다양한 조인 연산자의 구현 방안을 설명할 수 있다.
- 집합 연산자와 집단 연산자의 구현 방안을 설명할 수 있다.

학습내용

- SELECT, PROJECT 연산자의 구현
- 조인 연산자의 구현
- 집합 연산자와 집단 연산자의 구현

SELECT, PROJECT 연산자의 구현

외부 합병 정렬

01 외부 합병 정렬

정렬 알고리즘의 필요성

01 ORDER BY절, SELECT절의 DISTINCT, 조인 연산의 처리에서 사용

02 집합 연산(합집합, 교집합, 차집합)의 처리에서도 사용

릴레이션이 메모리 크기보다 작은 경우

퀵 정렬(Quick sort)이나 힙 정렬(Heap Sort)을 사용

릴레이션이 메모리 크기보다 큰 경우

디스크에 저장된 레코드들로 구성되며 주기억장치에 한꺼번에 수용할 수 없는 대규모 파일들에 대해서는 **외부 합병 정렬**을 사용

SELECT, PROJECT 연산자의 구현

외부 합병 정렬

01 외부 합병 정렬

특징

- 주 파일을 런(Run)이라고 하는 작은 부파일(Subfile)들로 나누고 이들을 정렬
- 정렬한 런(Run)들을 합병하여 더 큰 규모의 정렬된 부파일(Subfile)들을 생성하는 과정을 반복

주기억장치에 버퍼 공간을 필요로 하며 그 버퍼 공간에서 런(Run)들의 실제 정렬과 합병이 수행됨

정렬 단계

- 이용 가능한 버퍼 공간에 들어갈 수 있는 파일의 런(Run)들을 주기억장치로 읽어온 뒤, 내부 정렬 알고리즘을 이용하여 정렬한 후 이들을 임시 정렬된 부파일(또는 런(Run))로써 디스크에 저장
- Read RUN; Sort; Write RUN

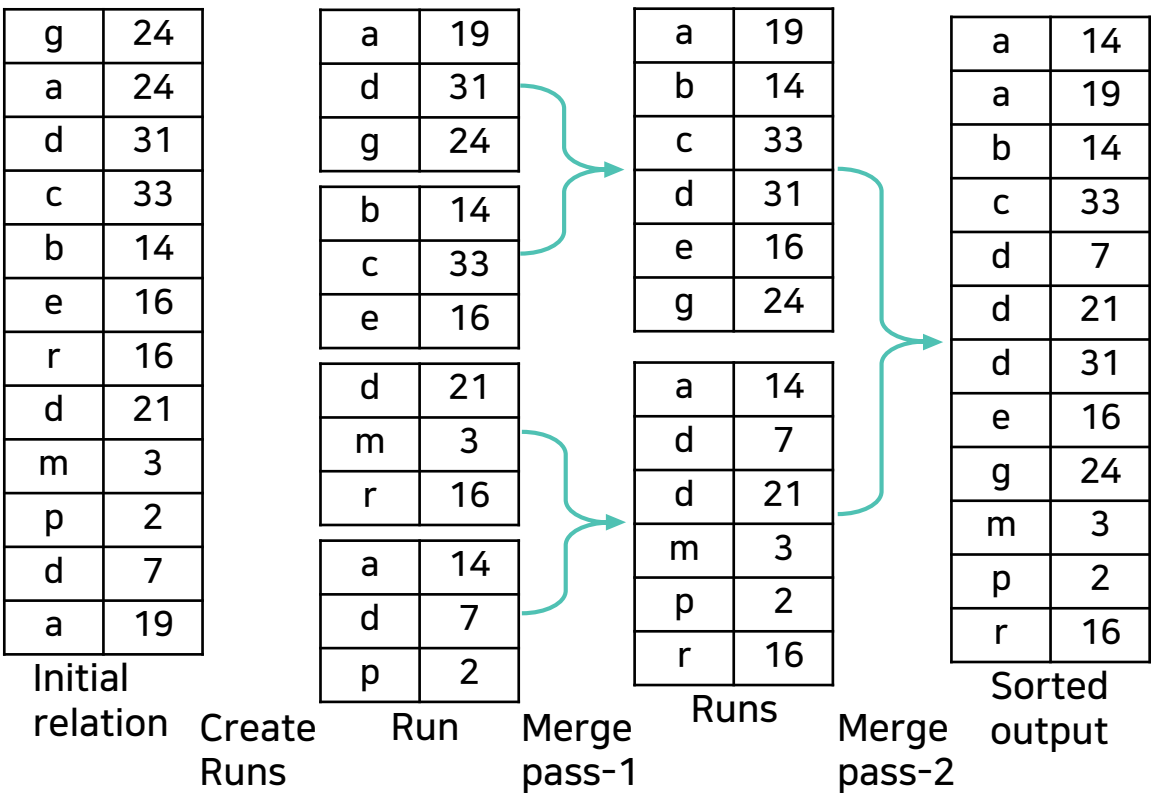
합병 단계

- 정렬된 런(Run)들이 한 번 이상의 패스(Pass)를 거쳐 합병
  → 4-way 합병: 4개의 Run들을 합병하여 하나의 Run을 생성
- 하나의 런(Run)으로 될 때까지 합병을 반복 수행

SELECT, PROJECT 연산자의 구현

외부 합병 정렬

01 외부 합병 정렬



SELECT, PROJECT 연산자의 구현

SELECT 연산자의 구현

01 관계 대수 연산자

- 질의문에서 쓰인 관계 대수 연산자를 구현할 저급 연산 프로시저가 필요함
  - 하나의 연산자는 여러 가지로 구현 가능
  - 하나의 접근 루틴은 특정 저장 구조와 접근 경로에 적용

» 예 | 색인 있다면 이를 사용할 것인가, 말 것인가?

02 SELECT 연산자

- 특정 조건을 만족하는 튜플들을 선발

SELECT, PROJECT 연산자의 구현

SELECT 연산자의 구현

03 SELECT 연산자를 위한 탐색 기법

선형 탐색 (Linear Search)

- 가장 단순한 방법(Brute Force)
- 데이터에 대한 기본 요구 조건이 없음
- 테이블의 튜플들을 처음부터 끝까지 검사하는 방법

이원 탐색 (Binary Search)

- 동등 비교, 정렬된 데이터
- 튜플들이 검색 조건 속성값을 기준으로 정렬되어 있다면 가능한 방법

- 기본 색인 또는 해시를 통해 하나의 레코드를 탐색
- 기본 색인을 이용해서 복수 레코드를 탐색
  - >, ≥, <, ≤
  - 검색 조건 속성값이 기본키(Primary Key)인 경우 기본 색인 있음
- 보조(B+ tree) 인덱스를 이용
  - 검색 조건 속성값이 기본키는 아니지만 색인이 구성 되어 있는 경우

SELECT, PROJECT 연산자의 구현

SELECT 연산자의 구현

04 SELECT 연산자의 시간 복잡도

튜플의 수를 m이라고 가정

선형탐색	m번 비교
이원탐색	$\log_2 m$번 비교
색인탐색	$\log_x m+1$번 비교

- $\log_x m$는 색인 트리의 깊이
- 1은 마지막에 실제 튜플에 접근

SELECT, PROJECT 연산자의 구현

PROJECT 연산자의 구현

01 PROJECT 연산자

튜플에서 특정 속성만을 추출하는 연산자

- SQL문의 SELECT절에 나타남
- 일반적으로는 SQL은 BAG임으로 PROJECT 연산을 함
- 결과물에 대하여 중복을 검사하지 않음
  - SELECT DISTINCT의 경우 중복을 검사해야 함

```
/* 프로젝트 애트리뷰트 : attr-list */
Do i = 1 to n ;                    /*  |R| = n */
   add R(i)[attr-list] to T' ;    /* attr-list로 프로젝트 */
end;

If('DISTINCT' 가 아님)
then T ← T' ;
else {
    If (R.key ⊆ attr-list)    /* 중복 가능성 검사 */
    then T ← T' ;
    else {
        T'의 투플을 정렬 ;
        set i := 1, j := 2 ;
        while (i < n)     do {
            add the tuple T'[i] to T ;
            while (T[i] = T'[j]) do j := j+1;    /* 중복 제거 */
             i := j ;     j := j+1 ;
        }
    }
}
/* T는 중복이 제거된 프로젝트 결과를 포함 */
```

SELECT, PROJECT 연산자의 구현

PROJECT 연산자의 구현

02 PROJECT 연산자의 시간 복잡도

- 튜플의 수를 m이라고 가정

DISTINCT가 아닌 경우	m번
DISTINCT인 경우	m+m log m

  - 정렬 비용 m log m

조인 연산자의 구현

중첩 루프 조인

01 조인 연산자

- 두 개의 테이블을 연결하는 조인 조건을 만족하는 튜플 쌍만을 추출하는 연산자

02 조인 연산자의 구현

중첩 루프 조인(Nested loop Join)

인덱스 조인(Indexed Lookup Join)

해시 조인(Hash-Lookup Join)

정렬 합병 조인(Sort Merge Join)

조인 연산자의 구현

중첩 루프 조인

03 중첩 루프 조인

- 가장 단순한 방법
- 테이블에 대한 어떠한 조건도 없음

$$R \bowtie_{R.A=S.A} S$$

  - 단점: 가장 느림

- 가능한 모든 쌍을 만들고 각 쌍에 대하여 조인 조건을 검사

```
/*  |R| = n,  |S| = m */
      for i = 1 to n;              /* outer loop */
        for j = 1 to m;            /* inner loop */
          if R(i).A = S(j).A then  /* join condition check
            add R(i)·S(j) to result;
        end;
      end;
```

조인 연산자의 구현

중첩 루프 조인

04 인덱스 조인

- S.A 대하여 B-tree와 같은 색인이 구성되어 있는 경우 적용 가능

$$R \bowtie_{R.A=S.A} S$$

  - R의 한 튜플 속성 A의 값을 가지고 S.A 색인을 검사하여 동일한 값을 가지는 S의 튜플들을 추출

```
/*  |R| = n,  |S| = m */
for i = 1 to n do  {              /* outer loop */
      S_Index = Index_lookup(R(i).A); /* Index lookup
      /* R(i).A와 같은 값을 가진 인덱스 엔트리가 k개, 즉
      |S_Index| = k 라고 가정*/
      for j = 1 to k {            /* inner loop */
            add R(i)·S_Index(j) to result;
      }
}
```

조인 연산자의 구현

중첩 루프 조인

05 해시 조인

- S.A에 대하여 해시 테이블이 구성되어 있을 경우 적용 가능

$$R \bowtie_{R.A=S.A} S$$

- R의 한 튜플의 속성 A의 값을 가지고 S.A 해시를 검사하여 동일한 값을 가지는 S의 튜플들을 추출
  - 해시는 충돌이 발생할 수 있음으로 실제 같은 값인지 확인 필요
- 해싱(Hashing)은 기본적으로 값의 크기 비교를 하지 못하므로, 동등 조인에 대해서만 적용 가능

```
/*  |R| = n,  |S| = m */
/* assume hash table H on S.A */
    for i = 1 to n;                /* outer loop */
        k = hash(R(i).A);
        /* H(k)에 h개의 투플, S(1), S(2), …, S(h)가 있다고
        가정*/
        for j = 1 to h;            /* inner loop */
            if S(j).A = R(i).A then  /* 같은 값인지 확인*/
                add R(i)·S(j) to result;
        end;
    end;
```

조인 연산자의 구현

중첩 루프 조인

06 정렬 합병 조인

- R과 S를 각각 속성 A에 대하여 오름차순으로 정렬

$$R \bowtie_{R.A=S.A} S$$

```
R = sort(R, R.A);
S = sort(S, S.A);
k= 1
for i = 1 to n do{
   for j = k to m do{
      if (S(j).A = R(i).A then
          add R(i)S(j) to result
      if (S(j).A > R(i).A then
          break
   }
   k = j
}
```

조인 연산자의 구현

중첩 루프 조인

06 정렬 합병 조인

- 두 테이블의 튜플 수는 각각 n, m이라고 가정
  - 조인 연산자

중첩루프	n*m
인덱스검사	$n*\log_x m*k$
해시검사	n*h //h는 동일 해시값을 가지는 S튜플의 평균수
정렬합병	nlogn + mlogm + m + n

집합 연산자와 집단 연산자의 구현

집합 연산자의 구현

01 합집합, 교집합, 차집합

합병 호환성을 만족하는 테이블들

01 중복된 튜플을 제거해야 함

02 정렬이나 해싱(Hashing)을 이용함

집합 연산자와 집단 연산자의 구현

집합 연산자의 구현

01 해싱(Hashing)을 이용한 집합 연산자의 구현

```
R U S
for every r in R
   insert r into hash table H
for every s in S
   if( H[h(s)] == empty) insert s into H
```

동일한 해시키 값을 가지는 S 튜플은 중복된 것임

```
R ∩ S
for every r in R
   insert r into hash table H
for every s in S
   if( H[h(s)] != empty) insert s into Result
```

동일한 해시키 값을 가지는 S 튜플은 R에도 있음

집합 연산자와 집단 연산자의 구현

집합 연산자의 구현

01 해싱(Hashing)을 이용한 집합 연산자의 구현

```
R - S
for every r in R
   insert r into hash table H
for every s in S
   if( H[h(s)] != empty) remove H[h(s)]
```

동일한 해시키 값을 가지는 S 튜플은 R에도 있음

집합 연산자와 집단 연산자의 구현

집단 연산자의 구현

01 MIN, MAX, COUNT, AVERAGE, SUM

대상 테이블에 색인이 있는 경우	대상 테이블에 색인이 없는 경우
색인을 이용하여 집단 연산을 수행	전체 스캔 필요

02 GROUP BY절

- 정렬이나 해싱(Hashing)을 이용하여 그룹핑 속성에 따라서 그룹을 생성
- 각 그룹에 대하여 집단 연산자를 적용

학습정리

1 SELECT, PROJECT 연산자의 구현

✓ SELECT

- 선형 탐색: 가장 단순하며 처음부터 끝까지 찾음
- 이진 탐색: 정렬되어 있을 경우에 적용
- 색인 탐색: 탐색 대상인 속성에 색인이 구성되어 있을 경우 적용

✓ PROJECT

- DISTINCT가 아닌 경우: 해당 속성 부분만 추출
- DISTINCT인 경우: 중복을 제거

2 조인 연산자의 구현

✓ 중첩 루프 조인, 인덱스 조인, 해시 조인, 정렬 합병 조인

3 집합 연산자와 집단 연산자의 구현

✓ 해쉬나 정렬기법을 이용하여 구현 가능