©2003, sangjin@kut.ac.kr

데이터베이스: 강의노트 09

A. Silberschatz, H. Korth, S. Sudarshan
Database System Concepts,
Fourth Edition, McGraw-Hill, 2002.

Part IV. Data Storage and Querying

13 질의 처리

13.1 개요

- 질의 처리 절차
  - 파싱과 번역
  - 최적화
  - 평가
- SQL로 제시된 질의는 먼저 관계 대수 형태로 번역되어 처리된다.
- SQL에 대한 파싱은 컴파일러에서 파싱하는 것과 유사하다. 이 과정에서 문법적 오류를 검사하고, 사용된 관계 이름과 속성 이름들이 올바른지 검사한다.
- 주어진 질의는 다양한 방법으로 평가할 수 있다.
  - SQL에서는 하나의 질의를 여러 형태로 표현할 수 있다.
  - SQL 문은 다양한 관계 대수식으로 번역될 수 있다.
- 예) 다음과 같은 SQL문은

  select 잔액
  from *account*
  where 잔액 < 2500

  다음과 같은 두 가지 관계 대수식으로 번역될 수 있다.
  - $\sigma_{\text{잔액}<2500}(\Pi_{\text{잔액}}(\textit{account}))$
  - $\Pi_{\text{잔액}}(\sigma_{\text{잔액}<2500}(\textit{account}))$
- 각 관계 대수식은 여러 가지 알고리즘을 이용하여 실행될 수 있다. 예를 들어 *account* 관계를 검색하여 잔액이 2500보다 작은 투플들을 찾을 수 있고, 잔액 속성에 대한 색인이 있으면 이것을 활용할 수 있다. 따라서 관계 대수식에 각 연산의 평가 방법을 나타내는 주석을 달아야 한다. 이 주석은 특정 알고리즘을 지정하거나 사용할 색인을 지정해준다.
- 평가 방법에 대한 주석이 첨가되어 있는 관계 대수 연산을 **평가 프리미티브(evaluation primitive)**라 한다.
- 일련의 프리미티브를 **실행 계획(execution plan)** 또는 **질의 평가 계획(query-evaluation plan)**이라 한다.
- 질의의 처리는 그것의 실행 계획에 따라 비용이 달라진다. 하지만 실행 계획은 사용자의 책임이 아니고, 시스템의 책임이다.

13.2 질의 처리 비용의 측정

- 질의 비용을 측정할 때 사용될 수 있는 평가 기준
  - 디스크 입출력 수: 대규모 데이터베이스에서는 가장 중요한 비용이다.
  - CPU 시간
  - 분산 환경의 경우에는 통신 비용
  - 응답 시간
- 여기서는 전송된 블록 수를 계산하여 질의 처리 비용을 측정한다. 이 때 모든 블록의 전송 시간은 같다고 가정한다. (실제는 어느 트랙의 어느 섹터에 있는지에 따라 전송 시간이 다르다.) 보다 정확하게 측정하기 위해서는 다음을 측정해야 한다.
  - 탐색 회수
  - 읽은 블록의 수
  - 쓴 블록의 수

  보통 쓰기가 읽기보다 시간이 많이 소요된다.
- 처리 비용은 또한 주기억장치에 유지되는 버퍼의 크기에 따라 다르다. 여기서는 최악의 경우를 고려하여 측정한다. 즉, 한 관계마다 하나의 블록만 버퍼에 있을 수 있다고 가정한다.

13.3 선택 연산

13.3.1 기본 알고리즘

- b_r: 파일에 있는 블록 수
- 두 가지 기본 파일 스캔(scan) 알고리즘
  - A1. **선형 검색**: 키 속성에 대한 동등 선택의 평균 비용은 $b_r/2$이며, 최악 비용은 b_r이다.
  - A2. **이진 검색**: 정렬된 속성에 대한 동등 선택은 이진 검색을 할 수 있다. 이 경우에 비용은 $\lceil \log_2(b_r) \rceil$이다.

13.3.2 색인을 이용한 선택

- 색인 구조는 다른 말로 접근 경로(access path)라 한다.
- 색인은 접근을 빠르게 해주지만 색인 구조에 대한 추가적인 접근이 필요하다.
- 색인을 이용한 스캔 알고리즘
  - A3. **주색인, 키에 대한 동등 선택**: B^+ 트리를 이용하면 "트리의 높이+1"만큼의 입출력이 필요하다. 여기서 "+1"은 실제 레코드를 추출하는 비용이다.
  - A4. **주색인, 키가 아닌 속성에 대한 동등 선택**: A3와의 차이점은 속성이 키가 아니므로 같은 값을 가진 레코드가 여러 개 있을 수 있다. 그러므로 트리의 높이에 레코드를 포함하는 블록 수를 더한 만큼의 입출력이 필요하다. 따라서 여러 레코드를 추출해야 하지만 주색인이므로 같은 값을 가진 레코드는 연속해서 저장되어 있다.
  - A5. **보조색인, 동등 선택**: 키에 대한 보조색인이면 하나의 레코드만 추출하면 되지만 키가 아닌 속성에 대한 보조색인이면 여러 레코드를 추출해야 한다. 전자의 비용은 "트리 높이 $+1$"이지만 후자의 비용은 최악의 경우 레코드마다 디스크 입출력이 소요될 수 있다. 이 비용은 선형 검색보다 더 클 수 있다.

13.3.3 비교를 포함하는 선택

- $\sigma_{A \le v}(r)$ 형태의 선택을 고려하여 보자.
- 색인을 이용한 스캔 알고리즘
  - A6. **주색인, 비교 선택**
    - $A > v$ 또는 $A \ge v$: 색인을 이용하여 $A = v$인 레코드를 찾은 다음에 단순 선형 파일 스캔을 한다.
    - $A < v$ 또는 $A \le v$: 색인을 사용하지 않고 단순 선형 파일 스캔을 한다.
  - A7. **보조색인, 비교 선택**: 보조색인을 이용한 비교는 각 영역에 해당하는 값을 이용하여 색인을 검색할 수 있다. 그러나 이 비용은 선형 스캔보다 오히려 클 수 있다.

13.3.4 복잡한 선택

- 복잡한 선택의 종류
  - 논리곱. 다음과 같은 형태의 질의를 말함

$$\sigma_{\theta_1 \wedge \theta_2 \wedge \cdots \wedge \theta_n}(r)$$

  - 논리합. 다음과 같은 형태의 질의를 말함.

$$\sigma_{\theta_1 \vee \theta_2 \vee \cdots \vee \theta_n}(r)$$

  - 부정. 다음과 같은 형태의 질의를 말함.

$$\sigma_{\neg \theta}(r)$$

- 복잡한 선택을 처리하는 알고리즘
  - A8. **하나의 색인을 이용한 논리곱 선택 연산**: 접근 경로가 있는 속성이 있으면 A2에서 A7 알고리즘 하나를 이용하여 선택을 한 다음에 주기억장치에서 나머지 조건을 검사한다. 이 때 $\sigma_{\theta_i}(r)$의 처리 비용이 가장 저렴한 θ_i와 알고리즘을 선택한다.
  - A9. **복합 색인을 이용한 논리곱 선택 연산**: 적절한 복합 색인이 존재하면 이것을 활용할 수 있다.
  - A10. **포인터의 교집합을 이용한 논리곱 선택 연산**:
  - A11. **포인터의 합집합을 이용한 논리합 선택 연산**

  A11 기법을 사용하기 위해서는 조건에 포함된 각 속성에 대한 접근 경로가 있어야 한다. A10 기법은 일부에 대한 접근 경로만 있어도 사용할 수 있으며, 이 때에는 일부에 대한 접근 경로를 이용하여 포인터의 교집합을 찾은 다음에 찾은 레코드에 대해 나머지 조건을 검사해야 한다.

13.4 정렬

- 데이터를 정렬하는 것은 다음 두 가지 이유 때문에 데이터베이스에서 중요한 역할을 한다.
  - 이유 1. SQL 질의는 결과가 정렬되기를 요구할 수 있다.
  - 이유 2. 질의 처리에서 몇 가지 관계 연산은 연산을 처리하기 전에 데이터가 정렬되어 있으면 보다 효과적으로 처리할 수 있다.
- 색인을 정렬된 순서로 유지하면 이 색인을 이용하여 정렬된 순서로 레코드를 접근할 수 있다. 그러나 레코드 자체가 물리적으로 정렬되어 있지 않기 때문에 많은 디스크 입출력이 필요할 수 있다.
- 정렬할 때 관계 전체를 주기억장치에 적재할 수 있으면 quicksort와 같은 기본 정렬 알고리즘을 사용할 수 있다. 반대로 적재할 수 없으면 외부 정렬(external sorting) 방법을 사용해야 한다. 가장 많이 사용하는 외부 정렬 방법은 외부 합병 정렬(external sort-merge) 방법이다.

13.5 조인 연산

- 조인을 계산하는 여러 알고리즘을 평가하기 위해 다음을 사용하며,

$$depositor \bowtie customer$$

  이 때 두 관계에 대해 다음과 같은 가정을 한다.

- *customer* 관계의 레코드 수: $n_c = 10,000$
- *customer* 관계의 블록 수: $b_c = 400$
- *depositor* 관계의 레코드 수: $n_d = 5,000$
- *depositor* 관계의 블록 수: $b_d = 100$

13.5.1 중첩 루프 조인

- $r \bowtie_\theta s$를 계산하는 중첩 루프(nested loop) 조인 알고리즘

  $\forall t_r \in r$마다 do
    $\forall t_s \in s$마다 do
      θ를 만족하는지 (t_r, t_s)를 검사
      만족하면 $t_r \cdot t_s$를 결과에 추가

- 여기서 r를 외부 관계라 하고 s를 내부 관계라 한다.
- 자연 조인을 계산하기 위해서는 중복을 제거해야 한다.
- 비용
  - 검사해야 하는 투플의 수: $n_r * n_s$
  - 최악의 경우 읽는 블록 수: $n_r * b_s + b_r$(버퍼에 각 관계의 한 블록만 유지할 수 있는 경우)
  - 최상의 경우 읽는 블록 수: $b_r + b_s$(버퍼에 두 관계의 모든 블록을 유지할 수 있는 경우)
- 만약 두 관계 중 하나의 모든 블록을 버퍼에 유지할 수 있으면 이 관계를 내부 관계로 사용하는 것이 좋다. (블록수: $b_r + b_s$)
- *depositor*가 외부 관계이고, *customer*가 내부 관계일 때의 비용
  - 검사해야 하는 투플의 수: $5000 * 10000 = 5 * 10^7$
  - 최악의 경우 읽는 블록 수: $5000*400+100 = 2,000,100$
  - 최상의 경우 읽는 블록 수: 500
  - 내부와 외부 관계를 바꾸었을 때 최악의 경우 읽는 블록 수: $10000 * 100 + 400 = 1,000,400$

13.5.2 블록 중첩 루프 조인

- $r \bowtie_\theta s$를 계산하는 블록 중첩 루프 조인 알고리즘

  r의 각 블록 B_r마다 do
    s의 각 블록 B_s마다 do
      $\forall t_r \in B_r$마다 do
        $\forall t_s \in B_s$마다 do
          θ를 만족하는지 (t_r, t_s)를 검사
          만족하면 $t_r \cdot t_s$를 결과에 추가

- 블록 중첩 루프 조인과 일반 중첩 루프 조인의 차이점은 일반은 투플 중심의 조인이고, 블록은 블록을 중심으로 조인하다.
- 비용
  - 검사해야 하는 투플의 수: $n_r * n_s$
  - 최악의 경우 읽는 블록 수: $b_r * b_s + b_r$(버퍼에 각 관계의 한 블록만 유지할 수 있는 경우)
  - 최상의 경우 읽는 블록 수: $b_r + b_s$(버퍼에 두 관계의 모든 블록을 유지할 수 있는 경우)
- 두 관계 모두 모든 블록을 버퍼에 유지할 수 없으면 크기가 작은 관계를 외부 관계로 사용하는 것이 좋다.
- *depositor*가 외부 관계이고, *customer*가 내부 관계일 때의 비용
  - 최악의 경우 읽는 블록 수: $100*400+100 = 40,100$
  - 최상의 경우 읽는 블록 수: 500
  - 내부와 외부 관계를 바꾸었을 때 최악의 경우 읽는 블록 수: $400 * 100 + 400 = 40,400$
- 중첩 루프 또는 블록 중첩 루프 알고리즘의 성능을 향상시키는 방법
  - 조인 속성이 내부 관계의 키이면 첫 일치를 발견하자마자 내부 루프를 중단할 수 있다.
  - 주기억장치의 버퍼가 M개의 블록을 유지할 수 있으면 이중 $M-2$ 블록을 외부 관계에 할당할 수 있다. 이렇게 하면 비용은 $\lceil b_r/M-2 \rceil * b_s + b_r$이 된다.
  - 내부 루프의 방향을 매 번 바꾸면 디스크 입출력의 수를 줄일 수 있다.
  - 내부 루프에 대한 색인이 존재하면 파일 스캔 대신에 색인을 활용할 수 있다.

13.5.3 색인된 중첩 루프 조인

- 중첩 루프 조인에서 내부 관계의 조인 속성에 대한 색인이 존재하면 파일 스캔 대신에 색인을 활용할 수 있다. 이렇게 조인하는 알고리즘을 색인된 중첩 루프 조인이라 한다.
- 비용
  - 블록 수: $b_r + n_r * c$, 여기서 c는 s에 대한 조인 조건을 이용한 단일 선택 비용이다.
  - r과 s가 모두 조인 속성에 대한 색인을 제공하면 투플의 수가 적은 것을 외부 관계로 사용하는 것이 효과적이다.
- 예) *customer* 관계는 고객명 속성에 대한 B^+ 트리 구조의 주색인을 유지하고 있다고 가정하자. 색인 노드 당 평균 20개의 항을 유지한다면 *customer* 관계의 투플 수가 10,000이므로 트리의 높이는 4이다. 따라서 $n_d = 5000$이므로 총 비용은 $100 + 5000 * 5 = 25,100$이다.

13.5.4 병합 조인

- 병합 조인(merge join) 또는 정렬-병합 조인은 조인 속성을 기준으로 두 관계가 모두 정렬되었을 때 사용할 수 있다. 만약 정렬되어 있지 않으면 정렬한 후에 이 알고리즘을 사용할 수 있다.
- 비용: 각 파일에 대해 한번의 스캔으로 조인을 할 수 있다.
  - 블록 수: $b_r + b_s$,
  - 두 관계가 모두 정렬되어 있지 않아 정렬 비용까지 고려하여도 병합 조인이 다른 조인에 비해 비용이 저렴하다.
  - 조인 속성의 값이 같은 모든 레코드가 버퍼에 적재될 수 없으면 조인 비용은 증가한다.
- 하이브리드 병합 조인 알고리즘: 한 관계는 조인 속성에 대해 정렬되어 있고, 다른 관계는 정렬되어 있지 않지만 조인 속성에 대한 보조 색인을 가지고 있을 때 적용할 수 있는 알고리즘이다. 이 알고리즘은 정렬된 관계와 정렬된 보조 색인을 합병한 다음에 보조 색인의 포인터 값을 이용하여 합병한 결과를 정렬하여 조인한다.

13.5.5 해시 조인

- 가정
  - h: 조인 속성을 $\{0, 1, \ldots, n_h\}$로 매핑하여 주는 해시함수로서 균일성과 랜덤 특성을 제공해야 한다.
  - H_{r_0}, H_{r_1}, ..., $H_{r_{n_h}}$: r 관계에 대한 분할을 나타내며, 초기에는 빈 상태이다. r의 각 투플은 조인 속성에 대해 해시값을 구하여 각 분할에 할당된다.
  - H_{s_0}, H_{s_1}, ..., $H_{s_{n_h}}$: s 관계에 대한 분할을 나타내며, 초기에는 빈 상태이다. r의 각 투플은 조인 속성에 대해 해시값을 구하여 각 분할에 할당된다.
- 이렇게 분할하면 H_{r_i}에 있는 투플은 오직 H_{s_i}에 있는 투플하고만 비교하면 된다.
- 해시 조인 절차
  - 단계 1. 두 관계를 해시함수를 이용하여 분할한다.
  - 단계 2. 각 분할에 대해 색인된 중첩 루프 조인을 한다.
    - 단계 2.1. H_{s_i}에 대한 해시 색인을 주기억장치에 만든다. 이 때 분할할 때 사용한 해시함수와 다른 해시함수를 사용한다.
    - 단계 2.2. H_{r_i}에 있는 각 투플에 대해 구축한 해시 색인을 이용하여 조인한다.
    - 이 때 H_{s_i}를 구축 입력(build input)이라 하고, H_{r_i}를 탐색 입력(probe input)이라 한다.
- n_h는 각 분할이 주기억장치 버퍼에 적재될 수 있도록 충분히 커야 한다. n_h는 최소한 $\lceil b_s/M \rceil$보다 커야 한다.
- 레코드의 수가 작은 관계를 구축 관계로 사용하는 것이 좋다.

13.5.5.1 재귀 분할

- 관계를 한번에 분할할 수 없는 경우에는 여러 번에 걸쳐 분할하여야 한다.
- 1차 분할 때 분할된 것을 각각 읽어 다시 작은 크기의 분할로 분할한다. 이렇게 하는 것을 재귀 분할(recursive partitioning)이라 한다.
- $M > n_h + 1$보다 크면 재귀 분할이 필요없다.

13.5.5.2 넘침 처리

- 분할 i에 대한 해시 테이블 넘침은 H_{s_i}에 대한 해시 색인이 주기억장치보다 클 경우에 발생한다.
- 파티션의 수를 필요한 것보다 많이 사용하면 넘침이 발생할 확률을 줄일 수 있다.
- 넘침이 발생하였을 때 해결 방법
  - 넘침 해결 방법: 넘침이 발생한 분할을 추가로 분할하는 방법
  - 넘침 회피 방법: 넘침이 발생할 수 없도록 만들 때부터 분할의 크기가 작도록 분할한 다음에 너무 작으면 분할을 결합한다.
- 많은 수의 레코드의 조인 속성 값이 같으면 넘침 문제를 효과적으로 해결할 수 없다. 이 경우에는 다른 조인 방법을 사용한다.

13.5.5.3 해시 조인의 비용

- 재귀 분할이 필요없고 넘침이 발생하지 않는다고 가정하였을 때 비용
  - 각 관계를 분할하기 위해서는 각 관계를 모두 읽은 다음 다시 써야 한다. 따라서 분할 비용은 $2(b_r + b_s)$이다.
  - 각 구축과 탐색 단계는 각 분할을 한 번 읽어야 하므로 추가로 $b_r + b_s$ 디스크 입출력이 필요하다. 그런데 파티션이 차지하는 블록 수는 $b_r + b_s$보다 클 수 있다. 최악의 경우 이 오버헤드는 각 관계마다 $2n_h$이다.
  - 따라서 총 비용은 다음과 같다.

$$3(b_r + b_s) + 4n_h$$

그런데 여기서 $4n_h$는 $b_r + b_s$보다 상대적으로 매우 작다. 따라서 무시할 수 있다.

13.6 다른 연산

13.6.1 중복 제거

- 방법 1. 정렬하여 제거한다. 이것의 비용은 정렬 비용과 같다.
- 방법 2. 해시함수를 이용하여 관계를 분할한 다음에 각 분할을 읽어 각 분할에 대한 해시 색인 구조를 만든다. 이 때 중복을 제거한다.
- 중복 제거 비용이 저렴하지 않으므로 SQL는 사용자가 중복 제거를 요청할 경우에만 중복을 제거한다.

13.6.2 집합 연산

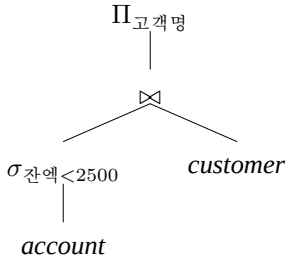
- 합집합, 교집합, 차집합 연산은 두 관계를 정렬한 다음에 평가한다. 이 때 자연스럽게 중복을 제거한다.
- 정렬 비용을 고려하지 않으면 세 연산 모두 $b_r + b_s$만큼의 디스크 입출력이 필요하다.
- 해싱 기법을 이용한 집합 연산의 처리
  - 해시함수를 이용하여 각 관계를 분할한다.
  - $r \cup s$
    - 단계 1. H_{r_i}에 대한 해시 색인 구조를 주기억장치에 만든다.
    - 단계 2. H_{s_i}의 투플이 색인 구조에 이미 없으면 색인 구조에 추가한다.
    - 단계 3. 색인 구조에 있는 투플을 결과에 추가한다.
  - $r \cap s$
    - 단계 1. H_{r_i}에 대한 해시 색인 구조를 주기억장치에 만든다.
    - 단계 2. H_{s_i}의 각 투플을 이용하여 색인 구조를 탐색하여 색인 구조에 같은 투플이 있으면 결과에 추가한다.
  - $r - s$
    - 단계 1. H_{r_i}에 대한 해시 색인 구조를 주기억장치에 만든다.
    - 단계 2. H_{s_i}의 각 투플을 이용하여 색인 구조를 탐색하여 색인 구조에 같은 투플이 있으면 색인 구조에서 삭제한다.
    - 단계 3. 색인 구조에 있는 투플을 결과에 추가한다.

13.6.3 집계

- 집계 함수의 경우에는 그룹핑이 가능하다.

$$_{\text{지점명}}\mathcal{G}_{\text{sum(잔액)}}(\textit{account})$$

- 집계 함수는 그룹핑 속성을 기준으로 정렬하거나 해싱 기법을 사용하여 분할한 다음에 처리한다.
- 그룹을 다 만든 후에 계산을 하지 않고, 그룹핑을 하면서 계산할 수 있다.



<그림 13.1> 연산 트리의 예

13.7 질의의 평가

13.7.1 실체화

- 관계 대수식의 평가 방법은 연산 트리를 보면 쉽게 알 수 있다.
- 예) 다음과 같은 관계 대수식의

$$\Pi_{\text{고객명}}((\sigma_{\text{잔액}<2500}(\textit{account})) \bowtie \textit{customer})$$

  연산 트리는 그림 13.1과 같다.
- 실체화 방식은 중간 연산의 결과를 임시 관계에 저장하는 방식이다.
- 실체화 방식으로 이 질의를 평가하면 가장 하위 수준의 연산부터 시작한다. 위 연산 트리에서 가장 하위 수준의 연산은 선택 연산이다. 이 연산부터 처리한 다음에 결과를 임시 관계에 저장한다. 이 임시 관계를 이용하여 조인 연산를 하여 또 다른 임시 관계에 저장하고, 끝으로 이 임시 관계에 대한 추출 연산을 적용하여 질의의 결과를 얻는다.
- 실체화 방식의 처리 비용은 각 연산을 처리하는 비용과 임시 관계를 저장하는 비용의 합으로 계산된다.
- 연산 결과의 레코드는 버퍼에 축적한다. 이 버퍼가 꽉 차면 버퍼를 디스크에 기록한다. 보통 한 블록 크기의 출력 버퍼를 사용한다.

13.7.2 파이프라인 기법

- 실체화 방식의 성능을 향상시키는 방법 중 하나는 임시로 생성되는 파일 수를 줄이는 것이다. 이를 위해 몇 개의 관계 연산을 파이프라인 형태로 처리한다.
- 예) $\Pi_{a_1,a_2}(r \bowtie s)$를 실체화 방식으로 처리하면 먼저 조인의 결과를 임시 관계에 저장한 다음에 이것을 다시 읽어 추출 연산을 적용한다. 이것의 성능을 향상시키기 위해 조인의 중간 결과를 저장하지 않고 부분 결과에 바로 추출 연산을 적용할 수 있다.

13.7.2.1 파이프라인의 구현

- 그림 13.1의 세 연산은 모두 하나의 파이프라인으로 구성할 수 있다. 선택의 결과를 바로 조인하고 그 다음 조인된 결과에서 추출한다.
- 파이프라인은 이전 연산이 결과를 버퍼에 기록하면 다음 연산은 이 버퍼를 이용하여 연산을 처리한다.
- 파이프라인 기법을 사용하면 연산의 입력이 한 번에 다 제공되지 않는다.
- 파이프라인을 실행하는 두 가지 방법
  - 방법 1. 요구 위주(demand driven)
  - 방법 2. 생산자 위주(producer driven)
- 요구 위주 파이프라인 기법(pulling data up)
  - 하위 연산은 상위 연산이 요구하면 투플을 생성하여 전달한다.
- 생산자 위주 파이프라인 기법(pushing data up)
  - 요청을 기다리지 않고 투플을 생성한다. 가장 하위 연산은 출력 버퍼에 공간이 없을 때까지 버퍼에 지속적으로 투플을 생성하여 적재한다. 중간 연산은 입력이 있으면 이것을 입력 버퍼에서 제거하고 출력 버퍼에 공간이 없을 때까지 투플을 생성하여 버퍼에 적재한다. 두 경우 모두 버퍼가 꽉 차면 상위 연산이 소비할 때까지 기다린다.
- 단일 프로세서 시스템에서는 버퍼가 꽉 차면 프로세스 스위치를 한다. 하지만 다중 프로세서 시스템에서는 각 연산을 동시에 수행할 수 있다.

13.7.2.1 파이프라인를 위한 평가 알고리즘

- 파이프라인 기법을 사용하면 알고리즘 선택에 제한을 받는다.
- 예) 조인 연산의 경우에는 정렬된 입력을 요구하는 병합 연산을 사용할 수 없다.
- 이 때문에 오히려 실체화 방식을 사용하는 것이 비용이 저렴할 수 있다.