하이퍼큐브 데이타베이스 컴퓨터 COREDB에서의 질의처리

박 휴찬*, 안 명수*, 김 탁곤*, 박 규호*
* 한국과학기술원, 전기 및 전자공학과, 컴퓨터공학연구실

Query Processing on A Hypercube Database Computer COREDB

Hyu-Chan Park, Myoung-Soo Ahn, Tag-Gon Kim, and Kyu-Ho Park
Computer Research Engineering Lab., Dept. of Electrical Eng., KAIST

요약

현재 개발 진행 중인 하이퍼큐브형 병렬 데이타베이스 컴퓨터인 COREDB의 질의 처리 부분에 관하여 기술하였다.

DBMS의 병렬화에 따라 기존의 비 병렬 DBMS에서는 발생하지 않던 새로운 문제점들이 발생하게 된다. 즉, 실제 병렬 연산을 수행하는 각 노드 컴퓨터와 이들을 제어, 관리하는 큐브매니저의 역할과 상호 작용, 질의 수행시 병렬성을 증가시켜 효율적으로 처리하기 위한 기법, 또한 단위 연산의 병렬화등의 문제점이 제기된다.

본 논문에서는 COREDB의 여러 구성요소 중에서 효율적인 질의 처리를 위한 여러가지 설계상의 문제점을 알아보고 구현상의 기법에 대하여 논하였다. 특히, SQL질의를 관계형대수의 기본 연산으로 변환, 분해하기 위한 두 단계 접근 방법, 즉 본래 SQL의 모든 부질의(sub-query)를 표현할 수 있는 기본 부질의로 변환한 후 최종적으로 관계형대수 기본연산으로 분해하는 방법과 이 과정에서 추가될 수 있는 질의 최적화 기법에 관하여 기술하였다. 또한, 기본 연산의 병렬화 및 효율적인 수행 전략에 대하여 설명하기로 한다. 위와 같은 일련의 과정에서 사용되는 내부 표현으로 질의 그래프 구조(query graph structure)를 사용하였으며, 릴레이션의 튜플들을 수평 분할하여 각 노드 컴퓨터들에 분산시킴으로써 병렬 처리의 효과를 증대시켰다.

I 서론

COREDB 프로젝트는 1990년 부터 시작되었으며 하이퍼큐브 컴퓨터 상에서 병렬 관계형 데이타베이스 관리 시스템의 개발을 목표로 1993년 까지 진행될 예정이다. 이 프로젝트의 목적은 가격 대비 성능면에서 우수하고 확장성이 뛰어난 데이타베이스 시스템의 설계와 구현에 있다. 시스템은 각각이 자신의 디스크를 가지는 노드컴퓨터들이 확장성이 뛰어난 하이퍼큐브 통신망으로 연결된 형태로 구성된다. 현재, 그림1과 같은 큐브매니저와 8노드 및 8디스크를 갖는 3차원 하이퍼큐브를 개발중에 있다. 큐브매니저와 노드컴퓨터는 68030 CPU, 8M DRAM, 디스크 콘트롤러 및 통신채널로 구성되어 있으며 운영체제는 UNIX System V 를 채택하고 있다.

본 논문에서는 COREDB 중에서 질의 처리 부분의 설계, 구현과 관련된 사항들을 논한다. 현재, 릴레이션의 튜플들을 각 노드컴퓨터에 분산하기 위한 병렬 분할 방법으로 해시(hash) 분할 기법을 이용하고 있다. 데이타를 각 노드 컴퓨터에 분산 저장함으로써 선택(select), 조인(join)등의 기본 연산의 병렬화 뿐만 아니라 질의간(inter-query) 병렬화도 쉽게 구현할 수 있다.

전체적인 질의 처리 방법으로는 먼저 SQL질의를 관계형 대수의 기본 연산으로 분해한 후 이들의 수행 순서를 결정함에 있어서 비용이 최소가 되는 수행전략을 찾게된다. SQL질의를 관계형대수의 기본 연산으로 변환, 분해하기 위한 2단계 접근 방법을 채택하였다. 즉, 본래 SQL의 모든 부질의(sub-query)를 표현할 수 있는 5가지의 기본 부질의로 변환한 후 최종적으로 관계형대수 기본연산으로 분해하게 된다. 이렇게 하면 복잡한 네스티드(nested)질의인 경우에도

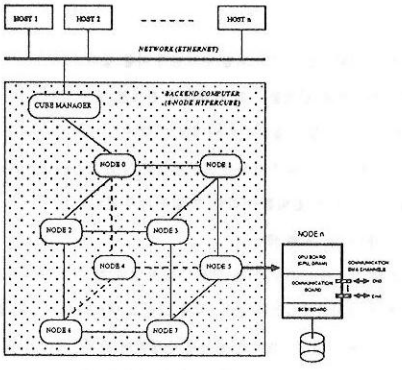

그림 1: 하이퍼큐브 컴퓨터

문제의 복잡도를 낮출 수 있을 뿐만 아니라 이 과정에서 여러가지 최적화 기법을 적용할 수 있다. 이와 같은 일련의 과정에서 유용한 내부 표현으로 질의그래프구조(query graph structure)를 사용하였다. 또한, 기본 연산의 병렬화 및 효율적인 수행 전략을 구현하였다.

2장에서는 COREDB의 각 프로세스(process)들의 구성과 그들의 역할에 대하여, 3장에서는 SQL로 주어진 질의를 관계형대수의 기본연산으로 변환하는 방법에 대하여, 4장에서는 각 기본연산의 구현 기법에 대하여 설명하고 5장에서 결론을 맺기로 한다.

II 프로세스 구조

본 장에서는 COREDB의 구성 요소인 호스터 컴퓨터, 큐브매니저, 노드 컴퓨터에서의 질의 처리를 위한 여러 프로세스(process)들의 역할과 그들의 상호 작용에 관하여 설명하기로 한다 (그림 2).

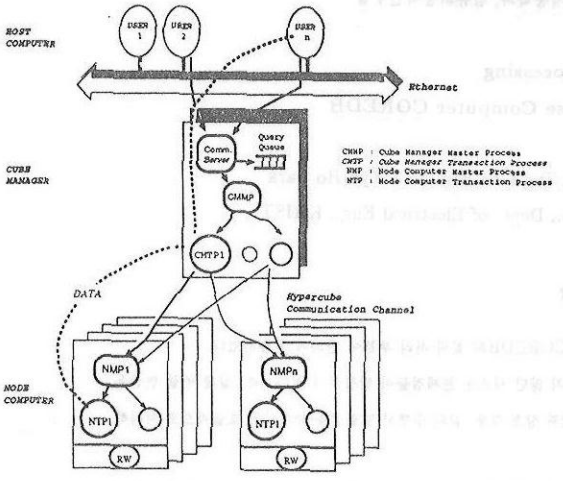

그림 2: 프로세스 구조

2.1 큐브매니저의 프로세스

큐브매니저 마스터 프로세스(CMMP)는 메세지큐(message queue)로 부터 질의를 하나 꺼집어내어 자신이 처리해야 할 질의이면 직접 처리하고, 트랜잭션 프로세스가 처리해야할 질의이면 큐브매니저 트랜잭션 프로세스(CMTP)를 하나 생성한 후 이 질의를 넘겨주고 자신은 다음 질의를 메세지큐로 부터 읽어들인다. CMTP가 수행해야할 질의들은 데이타베이스의 열기, 닫기와 데이타베이스의 생성, 삭제 같은 것으로 노드의 마스터 프로세스와 상호 통신하면서 처리하게 된다.

CMTP는 CMMP로 부터 질의를 넘겨받아 이를 질의변환기에서 어휘분석, 구문분석, 의미해석등의 단계를 거쳐 여러개의 관계형 대수 기본연산 단위로 나누어, 각 기본연산의 수행시 필요로하는 정보와 함께 노드 트랜잭션 프로세스에게 보내준다. 이 때는 노드의 트랜잭션 프로세스와 상호 통신하면서 주어진 질의를 처리하게 되며 주어진 질의를 처리한 후 없어지게 된다.

2.2 노드컴퓨터의 프로세스

노드 마스터 프로세스(NMP)는 컴퓨터가 부팅될 때 각 노드컴퓨터에서 수행되기 시작하여 데이타베이스의 생성, 삭제 그리고 데이타베이스의 열기, 닫기 등과 같은 명령들을 CMMP로부터 받아서 수행하여 준다. 그리고 새로운 트랜잭션이 발생하면 CMTP로부터 *MSG_START* 메세지를 받아 노드 트랜잭션 프로세스(NTP)를 하나 만들어 그 트랜잭션을 처리하게 한다. NMP는 현재 열려진 데이타베이스등과 같은 정보를 NTP에게 넘겨준다.

NTP는 트랜잭션을 수행하기 위한 관계형 대수 기본연산들을 받아서 처리해 준다. CMTP로부터 트랜잭션이 성공적으로 완료되어 결과가 데이타베이스에 반영되어도 좋다는 MSG_COMMIT 메세지를 받거나, 트랜잭션 수행도중 오류가 발생되어 트랜잭션 수행을 무효화하라는 MSG_ABORT 메세지를 받으면 필요한 뒷작을 행한 후 exit한다.

2.3 큐브매니저에서의 트랜잭션과 프로세스 관리

큐브매니저에서 트랜잭션을 처리하기 위한 프로세스의 생성 방법은 전체 시스템의 성능에 상당한 영향을 미치게 된다. 프로세스의 생성 방법을 크게 두 가지로 나누어 생각해볼 수 있다. 첫번째 방식은 데이타베이스를 사용하는 각각의 사용자마다 한개씩의 프로세스를 생성시켜 사용자의 질의를 처리해 주는 방법으로 프로세스의 생성 및 관리 방법이 간단하여 구현하기가 용이하다는 장점이 있지만 이 방식의 단점은 사용자가 DBMS환경에 들어와 있는 동안에는 사용자가 질의를 발생시키지 않더라도 계속 프로세스가 살아 있어야 하므로 전체 시스템의 성능을 저하시킬 수 있다.

이 방식을 보완하기 위한 다른 방법은 사용자들에 의해 발생된 질의가 있을 때만 이 질의를 처리하기 위한 프로세스를 생성하는 것이다. 즉, 사용자가 DBMS 환경에 있더라도 질의를 발생시키지 않으면 큐브매니저와 노드에서 프로세스가 생성되지 않으므로 데이타베이스 컴퓨터의 성능에는 나쁜 영향을 주지 않는다. COREDB에서는 두번째 방법을구현하였다. 그러나, 동시에 너무 많은 질의가 발생되면 그에 비례하여 CMTP와 NTP가 생성되어 이로 인한 전체 시스템의 성능저하를 초래할 수도 있으므로 트랜잭션 프로세스의 갯수를 시스템의 성능에 따라 제한된다.

이와 같이 메세지큐를 사용하여 질의를 처리했을 때의 또다른 장점은 여러개의 질의를 모아두었다가 동시에 처리할 수 있으므로 다중질의최적화 기법을 이용하여 질의 처리능력을 향상시킬 수도 있다는 것이다.

III 질의 변환 및 최적화

질의 변환은 사용자가 기술한 SQL을 관계형 대수(relational algebra)로 분해, 변환하는 과정으로서 주로 큐브매니저의 트랜잭션 프로세스가 담당하는 역할이다. 이와 같이 변환하는 이유는 SQL같은 질의어는 사람이 사용하기에는 적합하지만 질의의 내부 표현으로 시스템이 사용하기에는 부적합하므로 질의의 좀 더 유용한 내부 표현인 관계형 대수로 바꾸는 것이다. 즉, 사용자에의해서 주어진 *SQL*질의는 관계형 대수의 기본연산인 선택, 죠인, 추출등으로 변환된다.

COREDB에서의 질의 처리 과정은 [CERI85]에서 제안된 바와 같이 크게 두 단계로 나누어지며 전체적인 구성은 (그림3)과 같다. 첫번째 단계는 사용자가 준 SQL질의를 제한된 부질의(sub-query)만을 갖는 제한SQL(restricted SQL)로 변환하는 pre-processing과정이다. 두번째 단계는 제한SQL을 관계형 대수의 기본연산으로 변환하는 meaning과정이다. 일반적으로 질의처리 과정에서 가장 어려운 단계는 두번째 과정으로 알려져 있다. 위와 같이 질의처리 과정을 두 단계로 나누면, 어려운 두번째 과정에서 제한된 부질의만을 고려하면 되므로 문제의 난이도를 상당히 낮출 수 있다.

이 변환 과정에서 특히 부질의(subquery)를 갖는 네스티드(nested) 질의를 위한 효율적인 수행계획에 비중을 두었다. 일반적으로 이러한 네스티드 질의는 네스티드-루프(nested-loop) 죠인 형태로 수행된다. 즉, 외부질의 블럭의 각 튜플에 대해서 내부질의 블럭의 모든 테이블을 스캔(scan)해야한다. 이러한 비효율성을 제거하기 위하여 네스티드 질의는 이와 동등한 비 네스티드 질의로 변환된 후 질의 최적화 과정을 거치게된다. 이와 같은 일련의 과정에서 사용되는 자료구조로서 질의그래프구조(query graph structure: QGS)를 내부 표현으로 사용하였으며 이 구조는 강력하고 다양한 기능을 질의변환과 최적화를 위하여 제공한다.

변환 과정에서 질의 최적화 기법을 적용할 수 있으며 다음과 같은 최적화가 구현 완료 또는 진행 중에 있다. COREDB에서의 죠인은 분할에 기초한 알고리즘을 사용하고 있는데 이 경우에 다중 죠인을 위한 죠인 순서를 결정할 때 재

분할을 고려한 최적화가 되어야한다. 즉, 중간결과가 죠인 애트리뷰트에 대해서 분할 되어있지 않으면 이 중간 결과를 재 분할해야하며 이러한 재 분할을위한 통신량의 증가로인해 전체 시스템의 성능 저하를 초래하게 된다. 따라서, 다중 죠인 최적화를 위하여 레프트 디프 트리(left deep tree) 구조를 이용하였으며 죠인의 결과가 또다른 죠인의 입력으로 사용되는 경우에는 재 분할을 최소화하기 위한 죠인 수행 순서를 결정한다.

각 과정을 설명하기 위한 예제에 사용될 스키마와 질의는 다음과 같다.

```
s  (sno, sname, city)
sp (pno, sno, qty)

SELECT x.sname FROM s x
WHERE  sno NOT IN
  (SELECT sno FROM sp
   WHERE pno = 'p2')
```

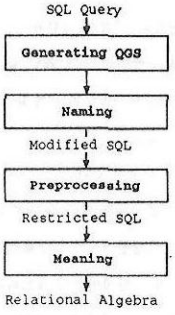

그림 3: 질의 변환 과정

3.1 질의그래프구조(QGS) 생성

주어진 SQL질의를 LEX와 YACC를 이용하여 어휘 및 구문분석을 하고 목록 화일을 참조하여 질의의 의미상 오류를 검증한다. 오류가 없으면 다음의 여러과정에서 이용될 내부 표현인 질의그래프구조(QGS)를 생성한다. 이 QGS는 SQL의 각 질을 내부적으로 표현하기가 용이하며 임의의 깊이를 갖는 부질의(subquery)를 표현할 수 있도록 구성되어 있어 특히 질의 변환에 유용하게 사용된다.

3.2 Naming Transformation

질의그래프구조로 표현된 SQL질의에서 모호함을 제거하게 되며 중요한 변환 규칙 및 그 효과는 다음과 같다.

- 동일 릴레이션을 서로다른 깊이의 부질의 사이에서 참조한 경우 이들의 이름을 구분해주고, 릴레이션과 관련된 변수가 있는 경우 이 변수는 해당 릴레이션 이름으로 치환된다.
- 애트리뷰트 이름이 그 애트리뷰트가 속해 있는 릴레이션 이름과 결합된다. 즉, 애트리뷰트 이름에 릴레이션 이름이 주어지지 않은경우 경우에 그것을 relation_name.attribute_name 으로 확장 시킨다.

```
SELECT s.sname FROM s
WHERE  s.sno NOT IN
  (SELECT sp.sno FROM sp
   WHERE  sp.pno = 'p2')
```

변수 x가 s로 변환되었고 애트리뷰트들이 릴레이션 이름과 결합되었다.

3.3 Pre-Processing

주어진 SQL 질의를 제한SQL 질의로 변환하는 과정이다. 위와 같은 변환의 결과는 모든 네스티드 질의(nested query)의 부질의가 제한된 다섯가지 기본적인 부질의인 simple_query, groupby_query, exists_query, complex_query, binary_query로 변환되는 것이다 (이 부질의에 관한 자세한 내용은 [김92] 참조). 또한, 조건절을 디스정티브 노말폼으로 변환하게 된다.

```
(SELECT s.sname FROM s)
MINUS
(SELECT s.sname FROM s
 WHERE  s.sno = (SELECT sp.sno FROM sp
                 WHERE  sp.pno = 'p2')
```

not_in 부질의가 기본 부질의 중 하나인 binary_query로 변환 되었다.

3.4 Meaning Analysis

기본적인 부질의로 변환된 질의들을 관계형 대수로 분해한다. 이와 같이 preprocessing과 meaning으로 구별한 이유는 본래의 SQL에서 모든 종류의 부질의(subquery)의 의미를 해석하는 것이 아주 어렵기 때문이다. 따라서, 모든 다른 부질의를 다 표현할 수 있는 기본적인 부질의의 부분집합을 정하여 그것으로 변환한 후 의미를 해석하게 된다. 해석된 결과 관계형 대수는 수행 모듈에서 제공되는 기본연산들을 이용하여 최종 결과를 얻게 된다. 이 단계에서 여러가지 최적화 기법을 추가할 수 있다.

```
(PJ[s.name] s)
DF
(PJ[s.sname] (SL[s.sno=sp.sno]
              (PJ[sp.sno] SL[sp.pno='p2'] sp)))
```

최종적으로 관계형대수의 기본연산인 추출(PJ), 선택(SL), 차집합(DF)으로 분해된다.

IV 실행 모듈

실행모듈은 질의 변환의 결과로 나오는 관계형 대수의 여러 기본 연산을 최적화 과정을 거쳐 효율적으로 수행하고 데이타베이스의 생성, 제거 및 변경 기능을 제공하는 여러 명령들을 처리한다. 또한, 컴퓨터에 구축된 모든 데이타베이스에 관한 정보를 유지하는 목록화일들을 관리하여 질의 변환이나 최적화시에 사용할 수 있도록 하는 기능을 제공한다.

4.1 데이타 저장 구조

현재 릴레이션을 분할하기 위하여 선택된 분할 애트리뷰트값에 해시(hash) 함수를 적용하여 어느 노드컴퓨터에 저장할 것인가를 결정한다. 각 노드컴퓨터에 저장된 릴레이션의 부분을 프래그먼트(fragment)라고 부른다.

현재의 해시 분할 방법은 분할 애트리뷰트의 값들이 균일 분포를 갖지 않을때 각 노드컴퓨터에 할당되는 튜플의 갯수가 많은 차이를 보이는 데이타 집중(data skew) 현상을 유발할 수 있으므로 이를 보완하기 위하여 여러가지 분할 방법이 고려되고 있다.

큐브매니저와 각 노드컴퓨터는 시스템에 저장되어 있는 데이타베이스에 관한 스키마 정보와 통계 자료를 저장하기 위하여 목록 화일을 유지한다. 큐브매니저의 목록 화일은 시스템에 저장되어 있는 모든 데이타베이스에 관한 정보를 저장하며 각 노드컴퓨터는 노드컴퓨터에 저장되어 있는 데이타에 관한 정보만을 유지한다.

4.2 일관연산

큐브매니저의 실행모듈이 질의 변환기로 부터 선택 연산을 받으면 어떤 접근 경로(access path)를 사용하여 연산을 수행할 것인가와 연산에 참여할 노드컴퓨터들을 결정한다. 선택 연산의 조건부에 분할 애트리뷰트가 사용됐다면 어떤 노드컴퓨터에서는 연산을 수행하여도 결과 튜플이 없을 수 있으므로 이러한 노

드컴퓨터에는 연산 자체를 전송하지 않음으로써 불필요한 처리 비용을 줄인다

접근 경로는 연산의 조건부를 조사하여 다음 여러가지 방법중에서 가장 비용이 적게 요구되는 방법을 선택하게 된다

방법 1 조건부에 '='(equal) 연산자가 사용된 에트리뷰트에 군집(clustered) 인덱스가 구축되어 있으면 이 인덱스를 사용

방법 2 조건부에 '=' 연산자가 사용된 에트리뷰트에 비군집(*nonclustered*) 인덱스가 구축되어 있으면 이 인덱스를 사용

방법 3 군집인덱스가 구축된 에트리뷰트가 조건부에 사용되었으나 연산자가 '>', '<','<=','>='과 같은 영역 연산자가 사용된 경우에 이 군집인덱스를 사용

방법 4 비군집인덱스가 구축된 에트리뷰트가 조건부에 사용되었으나 연산자가 영역 연산자인 경우에 이 비군집인덱스를 사용

방법 5 릴레이션의 모든 튜플들을 읽어서 조건부를 만족하는지 검사

선택 연산을 수행하기 위한 접근 경로와 연산에 참여할 노드컴퓨터들이 결정되면 퓨브매니저의 실행모듈은 연산 수행 명령을 해당되는 노드컴퓨터들에 방송하여 각 노드컴퓨터에서 병렬로 연산이 수행되도록 지시한다

각 노드컴퓨터들의 연산 결과 튜플들은 각 노드컴퓨터에 임시로 저장되어 다른 연산의 입력 데이타로 사용될 수도 있고 사용자 질의의 최종 결과일 경우에는 퓨브매니저의 명령에 의해 모든 노드의 결과 튜플들이 퓨브매니저로 전송된 후 사용자에게 출력 된다 사용자가 결과 튜플들을 특정 에트리뷰트에 대한 정렬된 순서로 출력되기를 원할때는 최종 결과 튜플들이 전역(global) 정렬 알고리즘을 사용하여 정렬된 후에 출력된다

4.3 이진연산

병렬 데이타베이스 컴퓨터에서 일반적으로 조인연산을 수행하기 위하여 데이타 전송 단계와 지역(*local*) 조인 단계를 분리하여 조인 연산을 수행하는 두단계 처리 방법이 COREDB에서도 사용되었다 COREDB에서 조인연산은 두번의 최적화 과정을 거쳐서 최소의 처리비용으로 처리되어 진다 첫번째 최적화 과정에서는 데이타 전송 단계에 사용되는 데이타 전송 방법을 결정하게 되며 두번째 과정에서는 지역 조인 알고리즘을 선택한다

첫번째 최적화 과정에서 노드컴퓨터사이에 데이타를 전송하기 위한 통신비용을 고려하여 선택할 수 있는 데이타 전송 방법으로 다음의 4가지 방법이 고려된다

방법 1 *local join*

조인에 참여하는 두 릴레이션이 모두 조인 에트리뷰트를 분할 에트리뷰트로 사용하여 같은 분할 함수에 의해 노드컴퓨터들에 분할 저장되어 있다면 조인 연산은 데이타 전송없이 각 노드컴퓨터에 저장된 프래그먼트 사이에서 수행하면 된다

방법 2 *repartition of one relation*

두 릴레이션중에서 한 릴레이션이 조인 에트리뷰트에 의해 이미 분할되어 있으면 다른 릴레이션을 같은 분할 함수를 사용하여 재분할한다

방법 3 *repartition of both relations*

두 릴레이션을 적당한 분할 함수를 사용하여 각 노드컴퓨터에 재분할한다

방법 4 *broadcast of one relation*

한 릴레이션을 모든 노드컴퓨터에 방송하여 각 노드컴퓨터에서는 방송되지 않은 릴레이션의 프래그먼트와 방송된 릴레이션의 모든 튜플사이에 조인연산을 수행하게 된다 이 방법을 고려할 때에는 분할에 기초한 방법보다 추가로 요구되는 지역처리비용이 비용 계산에 포함되어져야 한다 이 방법은 조인 연산자가 '=' 연산자가 아닌 비등가 조인(nonequi-join)이나 카티션 곱(cartesian product)연산시에 이용될 수 있다

위의 4가지 방법중에서 방법1~3은 전체 조인 알고리즘으로 분할에 기초한 알고리즘을 사용하는 것이고 방법4는 Nested-loop에 기초한 알고리즘을 이용하는 것이다 퓨브매니저에서 위의 여러 방법중에서 비용이 가장 적게드는 방법을 선택하여 각 노드컴퓨터에 전송하면 노드컴퓨터 사이에 데이타 전송 단계가 시작된다

데이타 전송 단계가 완료되면 각 노드컴퓨터는 조인연산의 두번째 최적화 과정인 지역 최적화 과정을 수행하여 어떤 조인 알고리즘을 사용하여 노드컴퓨터에 할당된 두 프래그먼트사이의 조인을 수행할 것인가를 결정한다 현재 지역 조인 알고리즘으로 nested-loop 알고리즘과 sort-merge 알고리즘, 그리고 hybrid-hash 알고리즘이 구현되어 있다

합집합, 교집합, 차집합 연산등과 같은 집합 연산도 조인 연산과 마찬가지로 분할에 기초한 방법에 의해 수행된다 먼저 두릴레이션중에서 큰 릴레이션의 분할 에트리뷰트에 해당하는 작은 릴레이션의 에트리뷰트를 분할 에트리뷰트로 선택하여 작은 릴레이션을 큰 릴레이션과 같은 분할 방법을 사용하여 재분할한 후 각 노드컴퓨터에 저장된 프래그먼트 사이에서 연산을 수행하면 된다

V 결론

대용량 데이타베이스의 관리를 효율적으로 하기위한 하이퍼큐브형 데이타베이스 컴퓨터인 COREDB의 여러가지 구성 요소 중에서 질의 처리와 관련된 몇 가지 사항들에 관하여 논하였다

각 프로세스들의 구성은 질의 처리의 효율성을 증대 시킬 수 있도록 구성되어 있다 SQL질의를 관계형대수의 기본 연산으로 변환, 분해하기 위한 두 단계 접근 방법은 문제의 복잡도를 낮출 수 있으며 최적화 기법을 구현하기에도 유용하다 전체 시스템의 병렬성을 증대시키기 위한 릴레이션의 병렬 분할 방법으로 해시 분할을 채택하였으며, 이를 이용한 단위 연산의 병렬화가 완료되었다

현재, 단위 질의 최적화 뿐만아니라 다중 질의 최적화 기능을 추가하고 있으며 시스템의 성능 향상을 위하여 병렬화 과정에서 요구되는 비용을 최소화하고 있다

참고문헌

[1] Stefano Ceri and Georg Gottlab, "Translating SQL Into Relational Algebra Optimization, Semantics, and Equivalence of SQL Queries," *IEEE Trans on Software Engineering*, SE-11(4), pp 324-345, April 1985

[2] 김명환 등, 하이퍼큐브형 데이타베이스 컴퓨터 개발 *(2차년도)*, 연구보고서, January, 1992

[3] Y C Kim, et al, "A Hypercube Database Computer - COREDB," submitted in the 1992 Phoenix Conference on Computers and Communications