SQL 질의 처리기의 클라이언트-서버 모듈 배치에 따른 성능 비교

(Performance Comparison of SQL Query Processors with Different Client-Server Module Configuration)

여 지 황 † 김 형 주 ††

(Jee-Hwang Yeo) (Hyoung-Joo Kim)

요 약 서버 집중 방식으로 구현된 SQL 질의 처리기를 클라이언트-서버 환경에 적합하게 재구성하는데는 성능 저하를 최소화 하는 것이 중요한 문제이다.

본 논문은 클라이언트-서버 환경의 SQL 질의 처리기가 성능 개선을 위하여 택한 구조에 대하여 다루고 있다.

변화된 구조의 질의 처리기는 관계 연산자들을 서버에서 제공하여 이동되는 데이타 양을 줄임으로써 성능이 개선되었다. 또한, 변화된 구조는 다양한 접근 방법을 이용한 실행 계획 생성을 가능하게 하여, 최적화 기법을 적용시킬 수 있었다.

본 논문에서는 클라이언트-서버 프로세스 상에서, 몇가지의 상이한 DBMS 모듈 배치를 구현하고, 각각의 적용에 따른 성능평가 결과를 제시하였다.

Abstract In restructuring centralized SQL processor into client-server one, it is important to minimize its performance degradation. This paper focuses on the architecture for improving performance of client-server SQL processor.

In the proposed architecture, the server part provides a number of relational operators, which reduces data transfer between clients and server and make it possible for client to generate an executioin plan and optimize it.

In this paper, we implemented several client-server SQL processors with different module configuration and present performance benchmarking results.

1. 서 론

1.1 연구 배경 및 목적

관계형 데이타베이스는 1970년 Codd에 의해 그 모델이 정립된 이래 수학적으로 잘 정의된 연산을 바탕으로 표 형태의 일관적인 데이타 표현에 의한 데이타 독립성을 제공하고, 선언적 데이타베이스 언어를 통하여 응용을 간편하게 작성할 수 있어 지금까지도 가장 많이 사용되고 있다.

관계형 데이타베이스의 가장 큰 장점 중 하나는 관계해석에 기반을 둔 선언적 데이타베이스 언어인 SQL을 제공한다는 점이다.

관계형 데이타베이스는 향상된 성능을 기반으로 의미적 확장을 시도하여 확장된 관계형 데이타베이스(extended relational database)로 발전하고 있다.

관계형 데이타베이스의 성능은 관계 연산의 구현 방법과 해석적인 질의어의 대수 표현 과정인 질의 처리에 영향을 받는다.

관계 연산자의 여러가지 구현방법들은 다양한 물리적 접근 경로 (physical access path)를 제공할 수 있으며, 질의 처리는 대등한 대수표현으로의 논리적 변환과 가장 효율적인 물리적 접근 경로로의 변환을 수행한다.

SQL 질의 처리기의 주요역할은 해석적인 SQL 문장

· 본 논문은 '95년도 통상산업부 공업기반기술 개발 사업 과제(과제명 · "대규모 방송 정보 시스템의 설계 및 구현")에 의해 지원되었음

† 준 회 원 : 웹 인터내셔널 전임연구원

†† 종신회원 · 서울대학교 컴퓨터공학과 교수

논문접수 1996년 1월 30일

심사완료 · 1996년 12월 5일

을 가장 효율적인, 실행 가능한 대수 표현으로 변환하는 것이다.

사용자가 명시한 질의는 셀렉션, 조인, 프로젝션 등의 관계 대수 연산으로 변환되며, 이 과정에서 실행가능한 관계 대수 연산자의 트리구조인 실행계획이 생성된다.

이 때, 대수 연산자로 이루어진 여러개의 실행계획이 생성될 수 있으며 각 대수연산자에는 여러개의 물리적 접근 경로가 대응될 수 있다.

질의 최적기는 최적화 알고리즘을 써서 최소 비용의 실행계획을 생성하게 된다.

클라이언트-서버 환경에서 관계형 데이타베이스의 성능은 클라이언트 프로세스와 서버 프로세스에 어떠한 구조로 DBMS 모듈이 배치되는가에도 좌우된다[7].

클라이언트와 서버에 DBMS 모듈을 배치하는 방법에는 크게 세가지 방법이 있을 수 있다. 즉, 서버가 단순히 데이타베이스 저장장치의 역할을 담당하는 경우, 서버가 기본적인 관계 연산자를 제공하는 경우, 그리고 서버에서 질의 처리기의 역할인 파싱과 질의 변환까지 수행하는 경우이다.

후자로 진행할 수록 서버의 작업량이 커지며 진행하는 각 단계마다 클라이언트와 서버 간의 통신량 및 전송되는 데이타 종류가 달라진다.

SRP(SNU Relational DBMS Platform) SQL 질의 처리기는 객체 지향 기법을 이용하여 설계되고 구현된 질의 처리기로서 서울대학교 객체지향시스템 연구실에서 1993년부터 개발되었고 현재도 개발 중인 시스템이다[15].

본 논문에서는 단일 사용자용 SRP SQL 질의 처리기를 클라이언트-서버 구조의 다중사용자용 시스템으로 재구성하는 과정에서, 위에서 언급한 클라이언트 서버 모듈 배치 변화와 질의 최적 기법 적용에 따른 성능 평가 결과를 제시하고 분석한다.

본 연구는 다음과 같은 기여도를 가진다.

- DBMS 모듈배치에 따른 성능 변화를 수치로 제시하였다.
- 최적화에 적합한 실행계획의 구조를 설계하고 Group-By 최적화를 포함한 최적기 및 실행 모듈을 구현하였다.

1.2 논문의 구성

2 장에서는 관계형 데이타베이스 성능향상을 위한 DBMS 구조 및 질의 최적기법에 대한 관련 연구를 서술한다.

3 장에서는 다중사용자용 SRP에서, 클라이언트와 서버 간의 4가지 모듈 배치 방법을 설명하고, 4 장에서는 SRP 질의 최적기의 구조및 최적화 기법을 소개한다.

5 장에서는 4 가지 모듈배치 방법 중 3가지 배치에 대한 성능 평가 결과를 제시하고 각각을 분석한다.

마지막으로 6 장에서 연구결과를 정리하고 앞으로의 연구 방향을 제시한다.

2. 관련 연구

2.1 클라이언트-서버 계산환경

최근의 가장 두드러진 계산 환경의 변화는 소형화(downsizing) 추세와 네트워크 기술의 발전이다.

소형화란 기존의 대형컴퓨터를 가격/성능비가 우수한 PC(개인용 컴퓨터)나 웍스테이션 (Workstation)등이 대체하는 현상을 뜻한다.

이러한 계산 환경의 변화는 분산되어 있는 자원들을 공유하여 하나의 시스템 역할을 할 수 있게 하는 분산처리를 가능하게 하였다.

'클라이언트-서버 계산 환경'은 실제로 이런 분산처리를 구현하기 위해 1990년대에 나타난 하나의 조류로서 PC 및 웍스테이션 들을 LAN(Local Area Network)을 통하여 연결하여 각각의 계산 능력을 최대한 활용하며, 또한 이들을 게이트웨이 등을 거쳐 메인프레임 등과 연결하는 분산처리의 한 방법이다.

본 연구는 클라이언트-서버 환경에 적합한 SQL 질의 처리기를 구현하는데 있어 클라이언트와 서버 각각에 어떠한 기능의 모듈을 배치하는 것이 가장 효율적인가에 주목한다.

또한, 클라이언트-서버 환경의 특징이 클라이언트의 계산 능력을 활용하는데 있으므로 SQL 질의처리기의 기능들을 최대한 클라이언트에서 담당하게 하는 구조를 제안하고, 이러한 새로운 구조가 서버에서 모든 기능을 처리하는 기존의 구조와, 성능면에서 거의 차이가 없음을 보이는 것을 그 목적으로 한다.

2.2 DBMS 구조

Hagmann과 Ferrari는 INGRES DBMS를 사용하여 전위컴퓨터와 후위컴퓨터에 5가지 구조로 작업량을 분할한 후, 성능평가를 실제로 수행하였다[7].

즉, 전위컴퓨터에서 수행되는 저장장치 모듈(SD), 접근 경로 모듈(AM), 단일변수 질의처리 모듈(IL), 질의분리 및 실행 계획 모듈(Decomposition), 그리고 파서(Parser) 모듈을 차례로 후위 컴퓨터로 이전시키고 각각을 성능 평가해 보았다. SD는 후위 컴퓨터로 화일 시스템 함수들이 이전된 것이며, AM은 get, replace, find, insert, delete 등의 레코드 연산이, IL은 한 테이블에 대한 연산 처리 함수가, Decomposition은 조인을 포함한

질의 처리기 전부가, Parser는 파싱을 포함한 데이타베이스 기능 전부가, 각각 후위 컴퓨터로 이전된 형태이다.

성능 평가 기준은 디스크 사용량, CPU 사용량, 그리고 네트워크 비용이다.

5개의 분할 구조 중에서 모든 면에서 우위에 있는 구조는 없었으며, 관계형 DBMS의 경우, Parser와 Decomposition 구조가 가장 좋은 성능을 나타내었으며 AM은 가장 통신량이 많았고, IL은 모든 면에서 가장 성능이 떨어졌다.

주목할 만한 것은 디스크 입출력 비용이 예상했던 것보다 60%나 높게 나온 사실이며 이 원인을 DBMS와 운영체계 간의 기능 불일치로 해석하였다.

즉, 운영체계 화일 시스템이 DBMS가 필요한 기능보다 훨씬 많은 기능을 갖고 있어 불필요한 동작때문에 디스크 입출력 성능이 나빠진다는 것이다.

따라서, 후위 컴퓨터가 DBMS 전용으로 쓰인다면, 기존의 운영체계보다는 DBMS 전용 실행기를 배치해야 한다고 주장했다.

DBMS와 운영체계 간의 기능 불일치라는 사실은 실험에 의한 성능평가로써만 알아낼 수 있기때문에 실험에 의한 성능평가의 타당성을 주장했다.

[7]의 AM, Decomposition, Parser 구조는 본 연구의 SAM, ROP, QP 구조와 유사하며 성능 평가 결과도 거의 일치하였다.

[7]은 기존의 DBMS 코드를 크게 고치지 않고 분할에 필요한 통신모듈이나 기타 코드를 덧붙여서 각각의 구조를 구성하였으나, 본 연구에서는 기존의 DBMS의 모듈배치를 변화시킬 뿐만 아니라 코드를 상당부분 재구현하여 성능평가 하였다.

또한, [7]은 디스크, CPU, 그리고 네트워크 비용을 상세히 분석하여 운영체계와 DBMS 간의 기능 불일치를 증명하려고 했음에 반해, 본 연구에서는 모듈배치 변화와 재구현에 인한 수행방법 변화가 통신량을 어느정도 줄일 수 있는가와 클라이언트의 계산 능력을 어느 정도까지 활용할 수 있는지에 주목한다.

Delis와 Roussopoulos는 저가, 고속 하드웨어 환경에 적합한 3가지의 DBMS 구조를 제안하고 각각의 성능을 모델링, 시뮬레이션하였다[2].

첫번째는 CS 구조(Client-Server Architecture)로서, 서버만 DBMS 기능을 갖추고, 클라이언트에서는 응용프로그램만을 수행시켰다.

두번째 RU 구조(RAD-UNIFY Type of DBMS Architecture)에서는 클라이언트에 질의 처리기를 배치하고 서버는 잠금, 입출력 등의 저급 DBMS 기능만을 수행하였다.

세번째 구조는 ECS(Enhanced CS Architecture) 로서, 서버와 클라이언트 모두에 DBMS 기능을 갖추고 클라이언트에서 질의 결과를 모두 캐싱한다.

ECS 구조는 CPU와 입출력 활용도를 높인 것으로 클라이언트 또는 서버사이의 연결 정보(Binding information)를 저장하여 캐시 일관성을 유지한다.

실험결과, 갱신 연산이 없는 순수 검색 질의에 있어서는 RU가 CS보다 약간 좋은 성능을 나타내었고, ECS는 다른 두 구조보다 월등히 좋은 성능을 보였고 클라이언트 수에 비례하여 성능이 증가하였다.

갱신 비율을 높일 때에도 ECS는 다른 두 구조보다 성능이 좋았고 클라이언트 수에 대한 성능 비례 정도(scalability)가 더 높았다.

[2]는 다수 사용자 환경을 가정하며, 모듈배치 변화에 의한 성능 변화보다는 ECS 모델에서의 캐싱의 효과에 대해 분석하였다.

본 연구에서 다수 클라이언트에 대한 성능 평가를 수행하지 않았다. 그 이유는 본 연구가 하나의 클라이언트와 하나의 서버에 대해서 다양한 모듈 배치에 따른 부하조절(load balacing) 효과에 주목하기 때문이다.

따라서, 다수 클라이언트에 대한 성능에 영향을 미치는 클라이언트 캐싱은 고려하지 않았다.

2.3 질의 최적화

질의 최적화는 통신비용, 디스크 접근 비용, 디스크 저장 비용, 그리고 계산 비용을 줄이는 것을 그 목적으로 하며, 질의 표현, 논리적 변환, 실행계획 생성, 비용계산 및 최적 계획 선택의 과정으로 수행된다[9].

질의 표현은, 논리적 변환이 가능하게 내부적으로 표현하는 것으로 관계 해석, 관계 대수, 질의 그래프, 질의 표 등이 있다.

질의 변환은, 질의 표현에 대한 변환 과정으로서, 일관된 형식으로 변환하는 정규화, 중복을 제거하는 단순화, 그리고 수행성능을 높이는 개량화 등의 논리적 최적화 과정으로 이루어져 있다[9].

실행계획을 생성시키고, 비용 계산을 하여 최적의 실행계획을 선택하는 방법은 크게 두가지가 있으며 두 방법 모두 탐색 공간을 줄이는 것을 목표로 한다.

첫째, System R[13] 형식의 동적 프로그래밍 방법은 릴레이션 갯수를 늘여가며 최적 실행 계획이 될 가능성 있는 후보를 정하고 나머지를 제거해나간다.

둘째, 임의화 알고리즘[8]으로서, 임의의 계획을 연속적으로 생성시켜 국지적, 또는 전역적인 최소비용을 구하는 방법이다.

System R의 최적화 기법은 시스템 통계정보를 이용하여 기본적인 접근방법의 비용을 계산하고 죠인순서를 결정한 후, 연속적인 두 릴레이션 간의 죠인 비용을 계산하고 최적 계획 후보를 선택한다. 두 개 이상의 릴레이션의 경우 가장 최근에 선택된 최적 실행 계획 후보들에 한 릴레이션을 첨가해가며 비용계산을 하여 최적 계획들을 생성시켜 나간다.

2.4 SRP SQL 질의 처리기

그림 1은 클라이언트-서버 구조의 SRP의 구조를 나타낸다.

사용자가 응용프로그램을 통해 입력한 질의어는 C++ CLI(call level interface)의 함수 호출을 통해 질의 처리기 모듈을 실행하여 처리된다.

질의 처리기는 파싱 객체를 통해 입력된 질의어에 대한 파싱을 수행하고, 파싱 결과인 파싱 트리 객체를 생성시킨다.

파싱트리는 정당성 검사를 통하여 의미적으로 옳은 질의인지 검사하며, 검사를 통과하면 실행에 필요한 정보를 담고 있는 질의그래프 객체를 생성시킨다.

질의그래프 객체는 스캔 관계 연산자로 이루어진 실행 계획을 생성시키며, 이 과정에서 질의 최적화를 수행하게 된다.

SESS(SNU: SNU Extensible Storage System)는 관계형 DBMS용 저장시스템으로 로깅 및 잠금, 그리고 기본적인 접근 방법을 제공하며, SESS 위에 관계 연산자를 구현한 스캔 인터페이스가 존재한다.

3. 클라이언트-서버 모듈 배치

그림 1은 SRP 모듈들을 클라이언트와 서버에 배치한 4가지 구조를 보여준다.

4가지 구조를 각각 단일 프로세스(SP: Simple Process), 질의처리기/단일테이블 접근경로(SAM: Single Access Method), 질의처리기/관계연산자(ROP: Relational OPerator), 응용프로그램/질의처리기(QP: Query Processor)로 명명하기로 한다.

이어지는 절에서 각 구조를 차례로 설명하도록 한다.

3.1 단일 프로세스(SP)

클라이언트-서버 형이 아닌 단일 사용자 용 구조이다(그림1 (a)) [15].

질의 그래프의 실행모듈에서 SESS 호출은 함수 연결로 되어 있어서 전체가 하나의 프로세스로 이루어져 있다.

SESS 모듈은 단일 사용자용으로 데이타베이스의 열고 닫음, 버퍼 관리, 고정길이 순차화일의 생성, 삭제, 열고 닫음, 투플 단위의 읽기, 쓰기, 삽입, 삭제, 그리고 인덱스의 생성, 삭제 및 인덱스를 이용한 검색 등의 한 테이블에 대한 접근 경로들을 제공한다. 단일 사용자 용 SESS는 병행제어와 로깅을 지원하지 않는다.

사용자가 질의를 입력하여 파싱을 하고 질의 그래프를 생성시키는 것은 4가지 구조가 동일하다. 질의 그래프에서 데이타베이스를 접근하여 실행시키는 방법에서 차이점이 있다.

SRP SP 구조에서 수행시키는 방법은 크게 3 가지 특징이 있다.

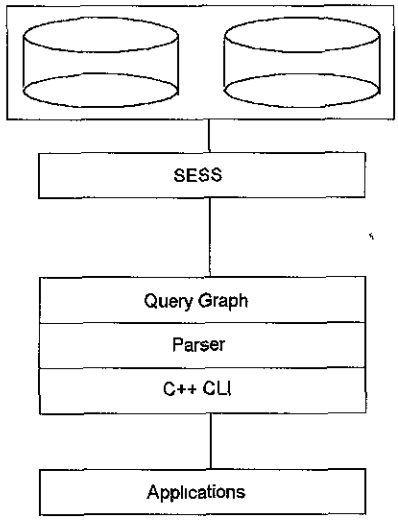
첫째, 단일 사용자용 SESS를 호출하여 질의를 수행시킨다.

단일 프로세스 구조를 제외한 나머지 세 구조들은 다중 사용자용 SESS 호출을 하며 병행제어와 로깅을 제공한다.

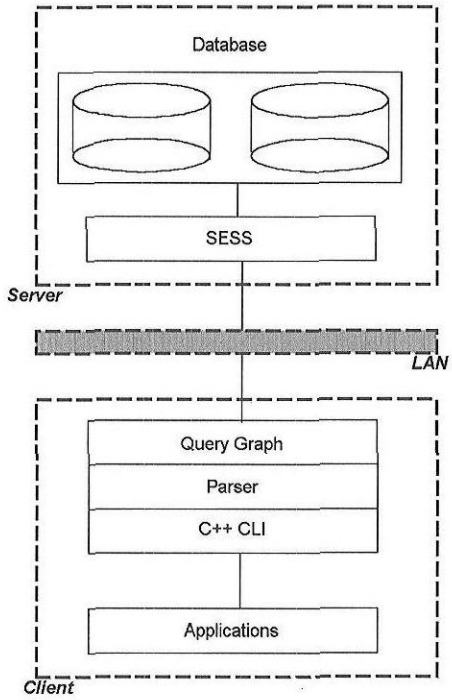
둘째, 셀렉션, 프로젝션, 죠인, 집계함수, 정렬 등의 관계 연산자가 정확히 나뉘어져 제공되지 않고 두가지 관계 연산자가 함께 수행되거나 한 연산자가 여러 곳에 중복되어 구현되어 있다. 즉, 프로젝션과 셀렉션을 동시에 수행하며, 정렬이 필요한 곳에 다른 이름의 함수들로 중복되어 제공된다.

두번째 특징은 SAM 구조에서 공통적으로 나타나며 ROP, QP 구조와는 구별되는 특징이다.

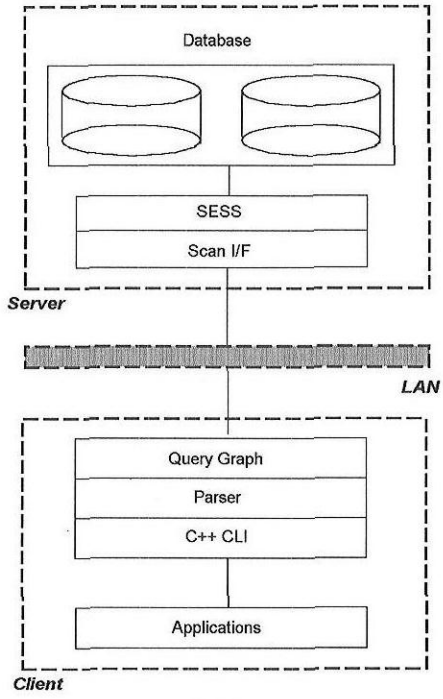
두 개 이상의 연산자를 함께 제공하면 성능이 향상되는 장점은 있지만, 관계 연산자들을 이용하여 실행 계획 트리를 구성할 수 없고, 따라서 질의 최적화가 불가능한 단점이 있다.



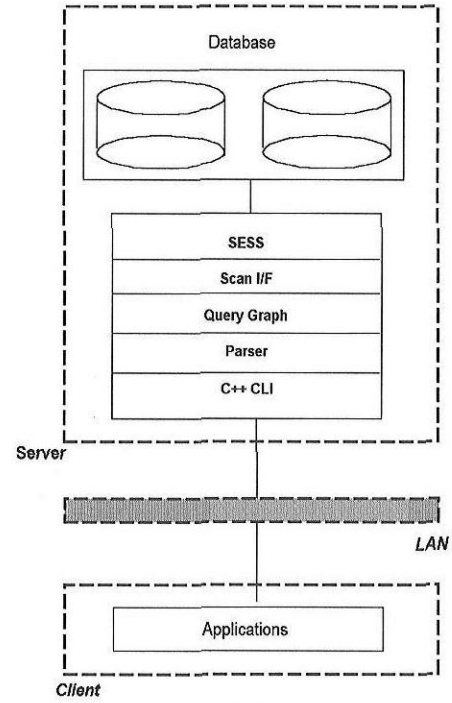
(a) SP



(b) SAM



(c) ROP



(d) QP

그림 1 SRP 클라이언트-서버 구조

세째, 관계 연산자의 실행 시 항상 임시 테이블을 만들어 저장시킨다.

이는 관계연산자의 결과도 관계(relation)이라는 관계대수의 정의에 충실하게 구현한 것이지만[10], 임시테이블을 생성시키고 조건에 맞는 투플을 삽입하는 비용이 많이 든다. 세번째 특징도 SAM 구조에서 공통적으로 나타나며 ROP, QP 구조에서는 이 비용을 줄이기 위하여 반복자(iterator) 방법[4]을 써서 스캔 연산자를 구현한다.

3.2 질의 처리기/단일테이블 접근 경로(SAM)

단일 프로세스 구조에서 함수에 의한 단일사용자용 SESS 호출에서 RPC(Remote Procedure Call)에 의한 다중사용자용 SESS 호출로 바꾸어 클라이언트-서버 구조로 만든 것이다. (그림1(b))

즉, 클라이언트의 질의 처리기는 다중 사용자 SESS를 RPC로 호출하는 것을 제외하고는 SP 구조와 동일한 모듈을 사용한다.

따라서, 3.1 절에서 설명한 3가지 특징 중 두번째, 세번째 특징이 그대로 적용된다.

ROP, QP 구조와의 공통점은 다중사용자용 SESS를 호출한다는 것이다.

다중사용자용 SESS는 데이타베이스와 로그에 대한 저장장치를 보유하며 서버프로세스는 데이타와 인덱스에 대한 동시성 제어, 페이지 할당, 그리고 회복 기법을 지원하는 역할을 담당한다.

여러개의 클라이언트 프로세스에서 요청된 서비스를 지원하기 위해 다중 쓰레드(multithread)를 가지며, 비동기적인 디스크 I/O를 위해 별개의 디스크 프로세스를 갖는다. 클라이언트 프로세스와 서버 프로세스 간의 통신(RPC)은 신뢰성있는 TCP 프로토콜과 UNIX의 소켓을 이용하여 이루어진다[3].

ROP, QP 구조와의 차이점은 첫째, SP 구조의 디스크 입출력 부분을 경계로 클라이언트와 서버가 분리된다는 점이다. 따라서, SAM 구조는 SP로부터 구현이 빠르고 쉽다.

둘째, SAM 구조는 서버의 부하는 감소시키나, 통신량을 증가시킨다.

SAM 구조에서 서버는 투플단위 저장장치 역할을 하여 한 테이블에 대한 접근경로만을 제공하고 클라이언트는 투플단위로 서버와 통신하여 관계 연산을 하고 또 그 결과 투플들을 서버로부터 전달받게 된다.

이 구조는 [7]의 접근 경로 모듈에 해당되며 [7]의 실험결과에 따르면 통신량이 가장 큰 구조이며 다른 구조에 비해 약 2배의 통신 시간이 소요되었다.

SRP에서도 연산에 쓰이는 투플(즉, 임시 테이블의 투플)과 결과 투플의 입출력으로 인해 많은 통신 시간이 요구된다.(5 장 참조)

3.3 질의 처리기/관계연산자(ROP)

이 구조는 서버에서 관계연산자를 제공하고 클라이언트에서 이를 이용하여 실행 계획 트리를 구성하여 질의 최적화에 이용할 수 있도록 만든 구조이다(그림 1 (c)).

ROP 구조의 구현상 특징은 크게 2가지로 요약된다.

첫째, 서버에서 관계연산자를 lazy evaluation 방법을 이용하여 구성하였다.

Volcano 질의 처리 시스템에서 쓰인 이 방법은 모든 관계연산자의 인터페이스가 open, close, 및 next의 세 가지로 동일하게 구성되어 있어 관계연산자의 트리를 구성하기 편하고 next가 호출되는 시점에서 결과 투플을 만들어 돌려주므로 투플들이 임시테이블에 저장될 필요가 없어 연산자 처리에 필요한 시간 및 공간 비용을 줄일 수 있다. [4]

둘째, 서버에서 제공하는 관계연산자를 이용하여 실행계획 트리를 구성하였다.

죠인 순서 및 방법 결정, Group By 연산자의 분할 배치 등의 기법을 써서 개선된 실행계획 트리를 생성한다(4 장 참조).

ROP 구조는 서버의 부하는 증가하나, 통신량이 감소되는 장점이 있다.(5 장 참조)

3.3.1 Scan 클래스

SRP ROP 구조에서 서버가 제공하는 관계연산자는 lazy evaluation 기법으로 Scan 클래스로 구현되어 있다.

모든 Scan 클래스들은 루트 클래스인 Scan의 서브클래스이며, 멤버함수로서 open(), close(), next(), reset()의 동일한 인터페이스를 갖는다.

open() 함수는 Scan 연산자를 초기화시켜주며, close는 사용을 마친 Scan 객체를 종료시키는데 사용된다. next() 함수는 각 Scan 연산자의 조건에 맞는 레코드들을 하나씩 사용자(SQL 처리기)에게 복사해 주는 역할을 하고, reset()은 이미 열려진 Scan 객체를 다시 처음 연상태로 전이시키는 기능을 갖는다.

다음은 각각의 Scan 연산자에 대한 설명이다.

● FixedSeqFile Scan :

SESS 내의 릴레이션을 순차적으로 읽기 위한 Scan이며 연산자 그래프의 단말노드가 된다.

● BTree Scan :

B+ 트리 인덱스를 써서 릴레이션을 읽기 위한 Scan이며 FixedSeqFile Scan과 마찬가지로 연산자 그래프의 단말 노드가 된다.

● Select Scan :

하나의 스캔과 조건을 인자로 받아 조건에 맞는 투플들을 검색한다.

● Project Scan :

하나의 스캔과 프로젝션되는 컬럼을 인자로 받아서 프로젝션된 투플들을 검색한다.

● Aggregate Scan :

스캔과 집계함수 종류(COUNT, MIN, MAX, SUM, AVG), GROUP-BY 컬럼들, 집계함수가 적용될 컬럼들을 인자로 받아서 GROUP-BY 컬럼들의 값이 같은 투플들에 대해 집계함수를 실행시켜 만든 투플들을 검색한다.

● Unique Scan :

하나의 정렬된 스캔과 정렬된 컬럼들을 인자로 받아 중복된 투플들을 제거한 투플들을 검색한다.

● Union Scan :

두개의 스캔을 인자로 받아 UNION한 결과 투플들을 검색한다.

● MergeSort Scan :

하나의 스캔과 정렬할 컬럼들을 인자로 받아 병합 정렬 알고리즘을 써서 정렬한다.

● Materialize Scan :

하나의 스캔과 저장할 릴레이션 이름을 인자로 받아 인자스캔을 데이타베이스 내의 실제하는 릴레이션으로 저장한다. 중첩블록 죠인시 루프의 안쪽에 위치하는 노드에 대해 *MaterializeScan* 연산자를 적용하여 반복되는 next() 호출 비용을 줄인다.

● MergeJoin Scan :

정렬된 두 스캔과 죠인컬럼들을 인자로 받아서 병합 죠인 알고리즘에 따라 죠인을 수행하고 결과 투플들을 검색한다.

● NestedBlockJoin Scan :

두 스캔과 죠인컬럼들을 인자로 받아서 중첩블록 죠인 알고리즘에 따라 죠인을 수행하고 결과 투플들을 검색한다.

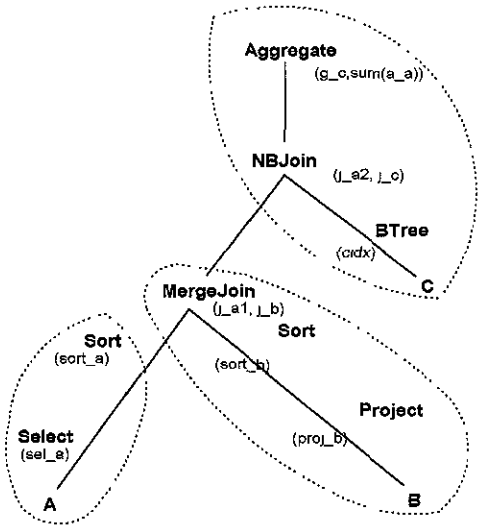


그림 2 왼쪽 깊은 연산자 트리

3.4 응용프로그램/질의 처리기(QP)

그림 1 (d)의 구조로서 클라이언트에서는 응용프로그램만을 수행시키고 DBMS 기능을 모두 서버로 이동시킨 구조이다.

QP 구조는 대부분의 시스템에서 채택하고 있는 구조이며, query shipping 구조라고도 불린다.

QP 구조는 SRP에서 아직 구현되지 않았다.

현재의 ROP 구조를 QP 구조로 재구현하려면 클라이언트 프로세스를 다중 쓰레드로 바꾸면 된다. 응용프로그램과 서버와의 통신은 SRP에서 제공되고 있는 C++ CLI를 이용하여 구현된 ODBC 드라이버 프로그램을 이용한다.

QP 구조는 [2]의 CS 구조와 일치하며, 통신량이 줄어드는 장점은 있지만, 서버의 부하가 심해진다[2].

4. SRP 질의 최적기

4.1 실행계획과 실행계획 관리자

SRP 질의 처리기의 실행계획은 연산자들의 왼쪽 깊은 트리구조(left-deep tree, 그림 2)로 만들어지며, AND로 연결된 술어가 실행계획의 대상이 된다.

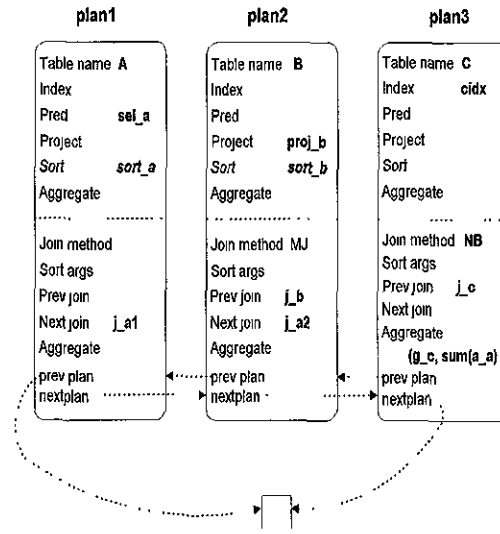


그림 3 실행계획 리스트

왼쪽 깊은 트리는 죠인 노드의 오른쪽 입력이 반드시 기본 테이블이 되는 구조로서 기본테이블 입력을 이미 죠인된 결과에 계속 덧붙여가며 실행된다.

다른 실행계획 구조로서 죠인노드의 두 입력이 모두 다른 죠인의 결과가 될 수 있는 부시 트리(bushy tree) 구조나 왼쪽입력이 기본테이블이 되는 오른쪽 깊은 트리(right-deep tree) 구조가 있다.

부시트리 구조는 보다 일반적인 실행계획을 만들 수 있고 오른쪽 깊은 트리 구조는 충분한 메모리 구조가 있는 시스템에서 효율적이다.

왼쪽 깊은 트리구조가 다른 구조에 비해서 갖는 장점은 질의 처리기의 탐색공간을 줄일 수 있다는 것이다[4].

왼쪽 깊은 트리는 링크 리스트로 표현될 수 있으며,

본 연구에서는 이중링크 리스트로 구현하였다(그림 3).

실행계획 자료구조의 한 링크는 한 기본테이블에 관련된 연산자들의 인자들을 명세하며 테이블 하나가 입력이 되는 연산자를 기술하는 기본테이블부과, 죠인 정보를 나타내는 죠인부으로 나눌 수 있다. SRP 질의 처리기 실행계획의 특징 중 하나는 Group By 최적화를 고려하여 설계되었다는 점이다.

기본테이블부는 B Tree 스캔, 셀렉션, 프로젝션, 정렬, 집계함수 연산자들의 인자를 기술하며, 죠인부가 실행되기 전에 수행된다.

죠인부는 현재의 링크의 기본테이블과 바로전 링크의 결과 간의 죠인을 기술한다. 그림 3에서 Join method는 중첩블록 죠인, 병합 죠인 중의 하나를 명세하며, Join sort args는 병합 죠인의 경우 정렬에 필요한 인자를 기술한다. Prev join args는 바로 전 링크와의 죠인에 필요한 기본테이블의 죠인 컬럼들을 명세하고 Next join args는 바로 다음 링크와의 죠인에 쓰일 죠인컬럼들을 명세한다.

실행계획 관리자는 실행계획 리스트를 순회하며 각 링크에 명세된 연산자들을 실행하며, 한 링크의 기본테이블부를 먼저 수행하고 죠인부를 만나면 바로전 링크와의 죠인을 수행한다.

그림 2와 그림 3는 왼쪽 깊은 연산자 트리와 대응되는 실행계획 리스트이다.

왼쪽 깊은 트리를 후위순행하는 것은 실행계획 리스트를 머리부터 따라 가는 것과 일치하며 그림 2에서 점선으로 둘러싸인 부분을 실행계획 리스트의 한 링크가 나타내게 된다.

실행계획 링크에서 연산자를 수행하는 것은 Scan(3.3 절)의 멤버함수인 open()을 호출하는 것이며, 트리의 루트 노드에 이르러 모든 연산자의 open이 완료된 후 결과를 돌려주기 위해 루트 노드의 Scan에 대해 next() 함수를 호출하게 된다.

하나의 실행계획은 AND로 연결된 한 술어에 적용되며 아는 다음에 기술할 최적화 기법들의 적용 단위가 된다.

각 실행 계획들이 OR로 연결될 수 있으므로 실행계획 관리자 내부에 OR로 연결된 갯수만큼의 실행계획 리스트 포인터를 유지하며, 각 실행계획 리스트를 수행한 후 각 결과들을 OR 연산한다.

5. 성능평가 방법 및 결과

이 장에서는 Wisconsin 벤치마크를 이용한 성능평가 방법을 소개하고 성능평가 결과를 제시하고 분석한다.

5.1 Wisconsin 벤치마크

Wisconsin 벤치마크는 원래 관계형 데이타베이스의 성능을 평가하기 위해 실험용으로 개발되었고, 초기의 단일 사용자용에서 다중사용자용 벤치마크로 발전하였다[5].

Wisconsin 벤치마크는 관계형 데이타베이스의 접근 방법 및 질의 최적화의 성능평가를 목적으로 하고 있고, 실제적인 데이타를 쓰지 않고 조합되어 생성된 릴레이션들을 그 대상으로 하여, 데이타 규모를 조절할 수 있고, 체계적으로 벤치마크를 수행할 수 있으며, 릴레이션 크기에 관계없이 균일한 분포를 유지할 수 있다.

총 32개 컬럼을 가진 2개의 기본 릴레이션으로 구성되며, 각 컬럼은 정해진 갯수의 구별되는 값(distinct values)을 가지도록 임의 또는 순차적으로 생성된 데이타를 가진다.

Wisconsin 벤치마크는 총 32개의 질의를 포함하고 있으며, 셀렉션, 프로젝션, 죠인, 집계함수 질의, 그리고 삽입, 삭제, 갱신를 그 내용으로 하며, 성능평가 측도로써 반응시간을 택하고 있다.

성능평가 도구로서 Wisconsin 벤치마크를 택한 이유는 다음과 같다.

첫째, 본 연구의 목적인 구조 및 최적화에 의한 성능향상 평가와 그 목적이 일치한다.

둘째, Wisconsin 벤치마크가 성능 분석보다는 성능 비교에 적합하게 만들어 졌으므로 [5] 여러 구조 간의 성능을 비교하는 본 연구에 성격에 부합된다.

세째, Wisconsin 벤치마크의 데이타 스키마 및 질의는 사용자가 의도한 질의를 쉽게 만들 수 있게 설계되어 있다.

모듈배치 성능평가의 경우는 Wisconsin 벤치마크에서 제공하는 질의를 썼으나, Group By 최적화 기능 성능평가의 경우는 릴레이션들은 그대로 쓰고 질의는 최적화 기능을 판별할 수 있게 새로 만들어 실험하였다.

5.2 시스템 환경

본 성능평가에 쓰인 하드웨어는 Sun SparcStation 10 (2 CPU) 이고, 운영체계는 Sun OS 4.1.3을 사용하였다. 네트워킹 도구로는 TCP/IP 소켓을 이용하였다.

클라이언트-서버를 다른 하드웨어에 배치하지 않고 같은 기계에서 수행시켰다.

그 이유는 성능이 다른 두 기계에서 실험할 경우 두 기계의 성능비를 고려해야하고, 또한 성능비를 알 경우라도 두 기계에 배치된 정확한 작업량을 계산해야 하는 번거로움이 있기 때문이다. 가장 좋은 선택은 같은 기종의 두 기계 상에서 클라이언트-서버를 수행시키는 것이

나, 여건상 불가능 하여 같은 기계에서 두 프로세스를 수행시키게 하였다.

5.3 모듈배치 성능평가

5.3.1 사용 질의 및 측정방법

본 성능평가에서는 20,000개의 투플을 가진 2개의 기본테이블 THOUSKTUP1, THOUSKTUP2과 2,000개의 투플을 가진 2개의 테이블 APRIME, BPRIME를 대상으로 총 32개의 질의 중 10개의 질의만을 수행하였다.

데이타베이스 총 크기는 약 9MB 이며, 10개의 질의의 내용은 셀렉션, 죠인, 집계함수, 그리고 삽입, 삭제, 갱신이다.

실험에서 제외된 질의는 클러스터 인덱스 및 부인덱스에 관한 질의와 프로젝션에 대한 질의이다.

인덱스 관련질의를 제외시킨 것은 SRP 에서 클러스터 인덱스를 지원하지 않고 또한 클라이언트-서버간의 통신량을 비교하는데 순차 검색만으로도 충분하기 때문이고, 프로젝션 질의는 DISTINCT이 들어간 프로젝션을 수행하는데 드는 정렬 부하가 각 구조에 공통적으로 너무 커서 성능평가 시간이 많이 들기 때문이다.

결과 반응시간은 10개의 질의를 2개의 테이블에 교대로 5번씩 10번을 수행하여 평균 반응시간을 계산하여 얻었다.

5.3.2 성능평가 결과 및 분석

3 장에서 제시한 4가지 구조 중 구현된 SP, SAM, ROP 3가지 구조에 대해 Wisconsin 벤치마크를 수행하였다.

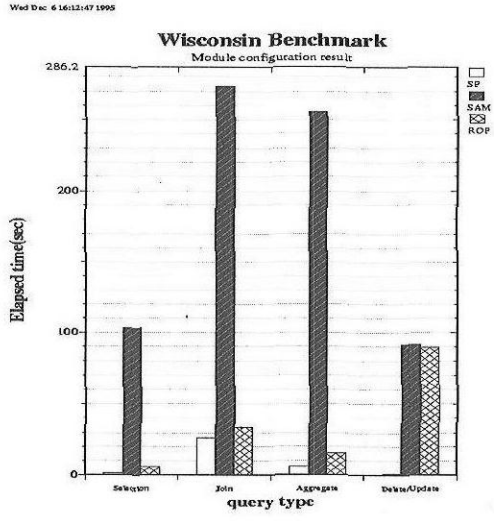


그림 4 모듈배치 성능평가 결과

다음은 각 질의에 대해서 SP, ROP, 그리고 SAM 구조에서의 반응시간 비를 나타낸다(그림 4).

1. 셀렉션 질의
   SP:ROP(SAM) = 1 : 5.8 (102.4)
2. 죠인 질의
   SP:ROP(SAM) = 1 : 1.3 (10.5)
3. 집계함수 질의
   SP:ROP(SAM) = 1 : 2.6 (43.2)
4. 삽입/갱신/삭제 질의
   SP:ROP(SAM) = 1 : 111 (114)

SP와 SAM의 성능 차이는 투플단위의 통신에 드는 비용과 로깅, 잠금 비용에 기인한다. 특히, 술어검사, 죠인 연산 및 임시테이블 저장을 위한 통신 비용이 그 대부분을 차지한다.

SP와 ROP의 성능 차이는 실행계획을 서버에 전달하고 최종 결과를 전송하는 비용과 로깅, 잠금 비용때문이다. 셀렉션, 죠인, 집계 함수 질의에서 ROP는 SP와 근소한 차이를 보인다. 그러나, 삽입/갱신/삭제 질의의 경우 스캔을 쓰지 않았기때문에 SAM과 마찬가지의 나쁜 성능을 보인다.

구현되지 않은 QP와 SP와의 성능 차이 원인을 예상해보면, QP는 SP의 성능에 질의문자열 및 질의결과 전송비용과 로깅, 잠금 비용을 추가한 것이다.

셀렉션, 죠인, 집계함수 질의에서의 SP와 ROP의 근소한 차이를 고려해볼때, 구현된 ROP 구조의 성능은 대부분 시스템에서 채택하고 있는 query shipping 구조인 QP 구조에 근접한다고 볼 수 있다.

6. 결 론

본 논문에서는 관계형 클라이언트-서버 DBMS 에 적합한 모듈배치를 실험에의해 제시하였다.

관계 연산자를 서버에서 제공하여 클라이언트-서버 구조를 구현함으로써

첫째, 클라이언트-서버 통신량을 감소시키고,

둘째, 실행계획 생성 및 질의 최적화를 가능하게 하며,

세째, 대부분의 시스템에서 채택하고 있는 query shipping 구조와 거의 비슷한 성능을 보일 수 있었다.

SRP 질의 처리기에 대한 향후 연구방향은 다음과 같다.

첫째, 캐싱을 적용하여 클라이언트-서버 구조의 성능을 개선시키는 연구를 수행할 것이다.

둘째, 서버에서 통계정보를 충분히 제공하여 비용에 기반하여 최적의 실행계획을 선택하는 질의 최적기를 구현할 것이다.

세째, 보다 많은 질의 최적화 기법을 적용하기 위하여 질의 처리기를 확장적(extensible)으로 만드는 연구가 필요하다.

참 고 문 헌

[1] C.J.Date and H. Darwen, editors. "A Guide to the SQL Standard". Addison-Wesley, U.S., 3rd edition, 1994.

[2] A. Delis and N. Roussopoulos. "Performance Comparison of Three Modern DBMS Architectures". IEEE Trans. on Software Engineering, 19(2):120-138, Feb. 1993.

[3] 이강우 외. "대규모 방송 정보 시스템의 설계 및 구현". 기술보고서, 서울대학교, Nov. 1995.

[4] G. Graefe. "Query Evaluation Techniques for Large Databases". ACM Computing Surveys, 25(2), June 1993.

[5] J. Gray, editor. "The Benchmark Handbook for Database and Transaction Processing Systems". Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1993.

[6] A. Gupta, V. Harinarayan, and D. Quass. "Aggregate-Query Processing in Data Warehousing Environments". Proc. of the Conf. on VLDB, pages 358-369, 1995.

[7] R. B. Hagmann and D. Ferrari. "Performance Analysis of Several Back-End Database Archtectures". ACM Transactions on Database Systems, 11(1):1-26, Mar. 1986.

[8] Y. E. Ioannidis and Y. C. Kang. "Randomized Algorithms for Optimizing Large Join Queries". Proceedings of SIGMOD, pages 312-321, 1990.

[9] M. Jarke and J. Kock. "Query Optimization in Database Systems". ACM Computing Surveys, 16(2):111-152, June 1984.

[10] H. F. Korth and A. Silberschartz, editors. "Database System Concepts". McGraw-Hill, Inc, 2nd edition, 1991

[11] J. Melton and A. R. Simon, editors. "Understanding the New SQL: A Complete Guide". Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1993.

[12] S. Chaudhuri and K. Shim. "Including Group-By in Query Optimization". Proc. of the Conf. on VLDB, pages 354-366, 1994.

[13] P. G. Selinger, M. Astrahan, D. Chamberlin, R. A. l

[14] A. Silberschatz, M. Stonebraker, and J. Ullman. "Database Research: Achievements and Opportunities Into 21st Century". *Report of an NSF Workshop on the Future of Database System Research*, May 1995.

[15] 송용준 외 3인. "객체지향 기법을 이용한 SQL 처리기 설계와 구현". 정보과학회논문지, 22(1). 13-22, January 1995.

[16] W.P.Yan and P.-A. Larson. "Interchanging the Order of Grouping and Join". Technical report, Department of Computer Science University of Wasterloo, Canada, 1995.

부록. Wisconsin Benchmark 데이타 및 질의어

Wisconsin Benchmark의 테이블 구조는 표 1과 같으며 규모를 원하는대로 조정할 수 있다.

표 1 THOUSKTUP1,THOUSKTUP2 테이블 스키마

Attribute Name	Range of Values	Order	Comment
unique1	0-(MAXTUPLES-1)	random	unique, random order
unique2	0-(MAXTUPLES-1)	sequential	unique, sequential
two	0-1	random	(unique1 mod 2)
four	0-3	random	(unique1 mod 4)
ten	0-9	random	(unique1 mod 10)
twenty	0-19	random	(unique1 mod 20)
onePercent	0-99	random	(unique1 mod 100)
tenPercent	0-9	random	(unique1 mod 10)
twentyPercent	0-4	random	(unique1 mod 5)
fiftyPercent	0-1	random	(unique1 mod 2)
unique3	0-(MAXTUPLES-1)	unique1	
evenOnePercent	0,2,4, ..., 198	random	(onePercent * 2)
oddOnePercent	1,3,5, . ., 199	random	(onePercent * 2) + 1
stringu1	-	random	candidate key
stringu2	-	random	candidate key
string4	-	cyclic	

다음은 실험에 사용한 질의어들이다.

● 1% Selection(S1)

```
INSERT INTO TMP
    SELECT * FROM THOUSKTUP1
    WHERE unique2D BETWEEN 0 AND 199;
```

● 10% Selection(S10)

```
INSERT INTO TMP
    SELECT * FROM THOUSKTUP1
    WHERE unique2D BETWEEN 0 AND 1999;
```

● Scalar Min Aggregate(AM)

```
INSERT INTO TMP (uniqueval)
    SELECT MIN(unique2D) FROM
```

```
THOUSKTUP1;
```

● MIN Aggregate with 100 partitions(AM100)

```
INSERT INTO TMP (uniqueval)
    SELECT MIN(unique3D) FROM THOUSKTUP1
      GROUP BY THOUSKTUP1.onePercentD;
```

● SUM Aggregate with 100 partitions(AS100)

```
INSERT INTO TMP (uniqueval)
    SELECT SUM(unique3D) FROM THOUSKTUP1
      GROUP BY THOUSKTUP1.onePercentD;
```

● 2 Table Join(J2)

```
INSERT INTO TMP
    SELECT * FROM THOUSKTUP1, THOUSKTUP2
    WHERE (THOUSKTUP1.unique2D =
            THOUSKTUP2.unique2E)
    AND (THOUSKTUP2.unique2E < 2000);
```

● 3 Table Join(J3)

```
INSERT INTO BPRIME
    SELECT * FROM THOUSKTUP2
    WHERE THOUSKTUP2.unique2E < 2000;
INSERT INTO TMP
    SELECT  THOUSKTUP1.*,BPRIME.*
    FROM BPRIME,THOUSKTUP1,THOUSKTUP2
    WHERE BPRIME.unique2E =
            THOUSKTUP1.unique2D
    AND THOUSKTUP1.unique2D =
            THOUSKTUP2.unique2E
    AND THOUSKTUP1.unique2D < 2000
    AND THOUSKTUP2.unique2E < 2000;
```

● Insert 1 Tuple(I)

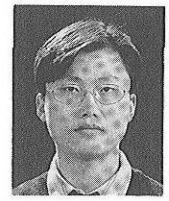
```
INSERT INTO THOUSKTUP1
    VALUES ( 'HHHHXXXXXXXXXXXXXXXXX
              XXXXXXXXXXXXXXXXXXXX
              XXX',

'PHXECXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXX',
'PHXECXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXX',
101, 100, 1000001, 1, 3, 8, 50, 10, 5, 2, 0, 1000001,
1000001);
```

● Delete 1 Tuple(D)

```
DELETE FROM THOUSKTUP1
  WHERE unique2D = 1000001;
```

● Update Key Attribute(U)

```
UPDATE THOUSKTUP1
  SET unique2D = 1000001 WHERE unique2D =
                 3999;
```



여 지 황

1994년 2월 서울대학교 컴퓨터공학과 졸업(학사). 1996년 2월 서울대학교 컴퓨터공학과 졸업(석사). 1996년 ~ 웹인터내셔널(주) 전임연구원. 관심분야는 관계형 데이타베이스 질의 처리, 객체지향 데이타베이스, 웹-데이타베이스 연동.

김 형 주

1982년 2월 서울대학교 컴퓨터공학과 졸업. 1985년 8월 Univ. of Texas at Austin, 전자계산학 석사. 1988년 5월 Univ. of Texas at Austin, 전자계산학 박사. 1988년 5월 ~ 9월 Univ. of Texas at Austin, Post-Doc. 1988년 9월 ~ 1990년 12월 Georgia Institute of Technology, 부교수. 1991년 1월 ~ 현재 서울대학교 컴퓨터공학과, 부교수 관심분야는 객체지향 시스템, 사용자 인터페이스, 데이타베이스.