# 객체지향 기법을 이용한 SQL 처리기 설계와 구현[1]
# (An Object-Oriented Design and Implementation of SQL Processor)

서울대학교 컴퓨터공학과 송용준(Yong-Jun Song)
서울대학교 컴퓨터공학과 이상원(Sang-Weon Lee)
서울대학교 컴퓨터공학과 안정호(Jung-Ho Ahn)
서울대학교 컴퓨터공학과 김형주(Hyoung-Joo Kim)

## 요약

데이타베이스 질의처리기를 설계하고 구현할때 성능, 확장성, 그리고 표준에의 적합성을 고려해야 한다. 본 논문은 확장성과 재사용성을 장점으로 하는 객체지향 개념을 이용해 SQL의 질의 처리기를 설계및 구현한 내용, SQL 표준에 대한 적합성 평가, 상용 데이타베이스 시스템인 INFORMIX와 성능 비교결과를 다루고 있다. 이와 같이 객체지향적으로 설계 및 구현된 SQL 처리기는 정보중복의 제거, 구현의 편의, 확장성 등의 많은 장점을 갖는다.

## Abstract

Performance, extensibility, and conformance are important issues in designing and implementing a database query processor. This paper describes the object-oriented design and object-oriented implementation of SQL processor using object-oriented concept, and shows the results of its conformance test and performance comparison with a commercial relational database management system, INFORMIX. SQL processor implemented in this paper has a number of advantages, such as avoidance of information redundancy, ease of implementation, and extensibility.

[1]본 논문은 상공부 공기반 과제 "대규모 방송정보시스템의 설계및 구현"의 일부 연구비 지원에 의한 것임.

## 1 서론

컴퓨터를 이용하여 정보를 처리하기 시작한 이래로 데이타베이스 기술은 많은 발전을 거듭하여 지금 현재는 제 4세대 시스템이라 할 수 있는 관계형 데이타베이스 관리 시스템(Relational DataBase Management System)이 가장 널리 사용되고 있다. 그 이유는 관계형 데이타 모델은 행과 열로 구성되는 테이블로 데이타가 표현되고 수학적으로 잘 정의된 연산들을 기반으로 하기 때문에 간단 명료하며, ANSI, ISO에서 표준화한 SQL[1][2]이라는 데이타 언어를 제공하기 때문이다.

본 논문에서는, 재사용성과 확장성을 제공하므로써 소프트웨어 산업 전반에서 각광을 받고 있는 객체지향 개념을 이용해서, 표준 SQL1[2]을 지원하는 질의 처리기(이하에서는 SNU-SQL-Engine이라 함.)를 설계하고 구현한 내용을 기술한다. 첫째, 질의 처리기의 각 구성 요소를 객체를 이용해서 모델링한 내용과, 둘째, 효율적인 SQL1 처리를 위해 제공하는 여러 기능들, 적합성 평가, 성능 평가의 결과와 마지막으로 설계, 구현과정에서 객체지향 개념이 제공하는 장점을 다룬다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 기존의 SQL 처리기의 설계방법과 객체지향 기법을 이용한 설계방법의 개념적인 차이점과 장단점을 설명한다. 3장에서는 2장의 개념적인 설계를 바탕으로 구현한 SQL 처리기의 전체적인 구조와 각 구성요소별로 그 기능들을 설명하고, 효율적인 질의 처리를 위해 SNU-SQL-Engine에서 제공하는 최적화 전략들을 소개한다. SQL1 표준안에 대한 적합성 평가와 INFORMIX[3]와의 간단한 성능비교 평가를 4장에서 보이고, 5장에서는 SNU-SQL-Engine의 객체지향적 설계 및 구현의 여러가지 장점을 기술한다. 마지막으로, 6장에서는 결론과 향후 연구 과제를 제시한다.

## 2 객체 지향 기법에 기초한 SQL 처리기

E.F.Codd가 1970년에 제안한 관계형 데이타 모델을 기반으로 한 여러 상용 관계형 데이타베이스 시스템이 널리 사용되고 있는데, 이들 시스템은 데이타의 생성, 유지, 제어할 수 있는 데이타 언어로서 대부분 SQL을 사용하고 있다. IBM의 System R의 SEQUEL을 모체로 하는 현재의 SQL은 오늘날 관계형 데이타베이스 시스템의 표준 데이타 언어로서 사용되고 있다.

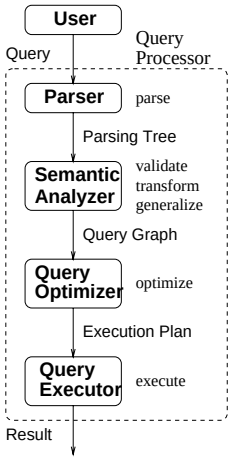


그림 1: 기존 SQL 처리기의 논리적 구조

사용자가 작성한 SQL문은 실제로 시스템 내에서 그 의미에 맞게 처리되는데, 이와 같이 시스템에서 입력된 SQL문의 처리를 담당하는 부분을 따로 SQL 처리기라 한다. 이 장에서는 먼저 기존 SQL 처리기 설계기법에 대해 알아보고, 이를 객체지향적으로 설계한 개념적인 구조를 제안한다.

### 2.1 기존 SQL 처리기의 설계기법

일반적으로 기존의 SQL 처리기는 그림 1과 같이 SQL문에 관련된 데이타들을 처리하는 4개의 관리자들로 구성된다.

SQL 처리기를 구성하는 첫번째 관리자는 구문분석기(parser)로서 사용자로부터 입력받은 SQL문을 어휘 분석과 구문 분석을 통하여 문법적 오류를 검사하고, 오류가 없을 경우에 그 결과 데이타로서 파싱 트리를 생성한다. 두번째 관리자는 파싱 트리를 처리하는 의미 분석기로서 파싱 트리가 올바른 의미를 갖는지 검사하고, 뷰 변환과 술어들의 정규화, 중첩 질의의 단일 레벨 질의들로의 변환 등을 수행하고, 파싱 트리를 시스템이 처리하기 편리한 내부적인 표현으로 바꾸는 일반화 작업을 수행하여 질의 그래프[8]를 생성한다. 세번째 관리자인 질의 최적기는 질의 그래프를 입력으로 하여 그것을 처리하기 위한 여러가지 방법들을 찾고, 그것들 중에서 가장 효율적으로 수행될 수 있는 것을 선정하여, 실제로 결과를 생성하기까지의 자세한 과정을 표현하는 실행 계획을 생성하게 된다. 마지막 관리자는 질의 실행자로서 실행 계획에 따라 정해진 과정

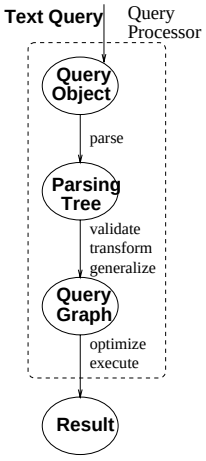


그림 2: SQL 처리기의 객체지향적 설계

을 실제로 수행하고, 그 결과를 생성하게 된다.

### 2.2 SQL 처리기의 객체지향적인 설계

객체 지향 시스템은 객체, 이들 객체들 간의 메시지(message) 전달, 그리고 메시지를 받은 객체가 메시지에 해당하는 메소드(method)를 실행시킴으로써 이루어지는데, 이런 관점에서 SQL 처리기도 SQL문에 관련된 객체 위주로 설계하고 그것을 처리하는 모든 동작들은 각 객체의 메소드로 구현하고 프로그램의 흐름은 멤버함수의 호출로 이루어지는 것이 자연스럽다.

SNU-SQL-Engine은 개념적으로는 그림 2와 같이 크게 3개의 SQL문 관련 클래스의 인스탄스 객체들로 구성된다고 볼수 있다. SQL 처리기를 구성하는 첫번째 객체는 사용자가 입력한 SQL문을 객체화한 질의 객체(query object)로서 그것의 멤버 함수를 통하여 구문분석을 수행하여 파싱 트리 객체를 생성하게 된다. 파싱 트리 객체도 정당성 검사, 뷰 변환과 중첩질의의 일반화를 위한 멤버 함수들을 갖는데 그것들을 차례로 호출함으로써 질의 그래프 객체를 생성한다. 질의 그래프 객체는 최적화와 실행을 담당하는 멤버함수를 갖는데, 그것들을 차례로 수행함으로써 최종 결과 객체를 생성하게 된다.

## 3 전체 시스템의 구성 및 기능

여기서는 앞에서 설명한 SQL 처리기의 개념적인 구조를 실제로 구현한 내용을 다루고 있다. 전

체 시스템의 구성과 주요 구성객체의 기능을 기술한 후, SNU-SQL-Engine에서 제공하는 질의처리를 위한 다양한 최적화 기법을 설명한다.

### 3.1 전체 시스템의 구성

본 SQL 처리기는, 하부 저장 시스템으로 서울대학교 객체지향시스템 연구실에서 개발한 확장 용이 저장 시스템인 SESS(SNU Extensible Storage System)[4]를 사용한다. 그림 3은 SQL 처리기를 이루는 주요 구성요소들과 이들 간의 관계를 보여주는 전체적인 시스템 구성도를 나타낸다.

SNU-SQL-Engine은 앞서 설명했듯이 질의 객체, 파싱 트리 객체, 질의 그래프 객체와 같은 SQL문 관련 클래스들의 인스탄스 객체들을 중심으로 구성되며 그 객체들이 갖는 멤버함수들의 수행에 필요한 시스템 카탈로그들과 테이블들을 저장 시스템으로부터 메모리 내의 처리 가능한 객체들로 변환하여 관리하는 카탈로그 관리자와 테이블 관리자를 포함하며, 권한 검사를 위한 권한 관리자를 포함한다.

SNU-SQL-Engine은 UNIX 환경에서 객체 지향 기법을 이용하여 세분화된 클래스들로 설계되고 현재 가장 널리 사용되는 객체 지향 프로그래밍 언어인 C++로 구현되어 확장성과 재사용성이 좋은 것이 특징이다. 현재 200개 이상의 C++ 클래스정의와 멤버함수를 포함하여 약 35,000줄 정도의 코드로 이루어져 있다. C++는 클래스를 통한 데이타추상화를 지원하고, 상속과 가상함수에 의한 함수의 동적인 바인딩을 지원하기 때문에 소프트웨어 공학 측면에서, C 언어를 이용해서 질의 처리기를 구현했을때 보다, 더 높은 수준의 모듈화, 재사용성, 그리고 확장성을 제공한다.

SNU-SQL-Engine은 그림 3에 나타난 객체들을 위한 클래스들 뿐만 아니라 질의 처리 과정 중에 필요한 수많은 객체들을 위한 클래스들로 구성되어 있다. 이 글에서는 그것들 중에서 가장 기본이 되는 테이블과 투플에 관련된 클래스에 대해서만 3.2.7절에서 설명하기로 한다.

### 3.2 각 구성 객체의 기능

#### 3.2.1 사용자(User)

현재 SNU-SQL-Engine은 대화형 SQL만을 지원하는 처리기이므로 사용자는 시스템을 사용하는 사람이 된다. 사용자는 입력으로 텍스트 질의를 넘겨주고, 결과로 주어진 질의의 결과 투플들의 집합인 테이블이나 에러메시지를 돌려받게 된다.

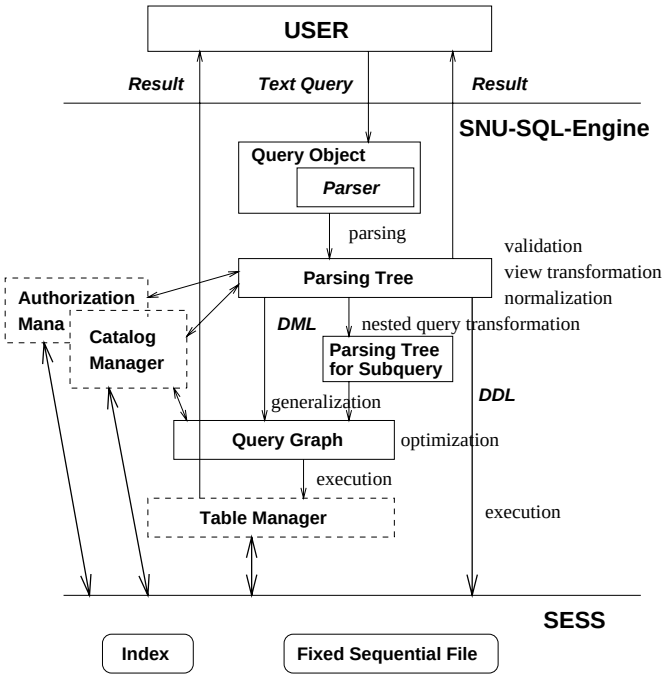


그림 3: SNU-SQL-Engine의 구조

### 3.2.2 질의 객체(Query Object)

질의 객체는 텍스트 질의에 해당하는 파싱 트리 객체를 생성하는 파싱 멤버함수를 갖는데, 해당 질의에 문법적인 오류가 없을 경우에 파싱 트리 객체를 생성한다.

### 3.2.3 파싱 트리 객체(Parsing Tree Object)

각 파싱 트리 객체들은 검색 조건들의 정규화를 담당하는 멤버함수, 유효한 테이블, 컬럼, 데이타 값 등의 사용 여부와 사용자의 권한검사를 담당하는 정당성 검사 멤버함수를 갖고 있다. 정당성 검사 멤버함수 수행시, 중첩된 SQL문에 대한 파싱 트리를 단일 레벨 SQL문의 파싱 트리들로 변환하는 작업도 동시에 수행된다.

이외에 파싱 트리 객체는 뷰를 포함한 파싱 트리 객체를 뷰의 기본 테이블에 대한 파싱트리로 변환하는 뷰 변환 멤버함수를 갖는다. SNU-SQL-Engine은 질의 변환 방법[5]에 의해 뷰를 지원하는데, 뷰를 정의하는 SQL문을 저장해둔 뒤에 그 뷰가 질의에 사용되는 경우에 저장된 뷰 정의문을 읽어 파싱 트리 객체를 생성하여 질의로부터 생성된 파싱 트리 객체를 기본 테이블에 대한

파싱 트리 객체로 변환한다. 그리고 질의 변환 방법에 의한 뷰 외에도 통계뷰라는 특정타입의 뷰를 참고문헌 [6]에 나와 있는 count 방법을 통해 저장뷰(materialized view)로 생성한다. 통계뷰는 SQL에서 지원하는 통계함수 AVG, MAX, MIN, SUM, COUNT를 이용해서 정의한 뷰를 일컫는데, 이들은 자주 사용되는 유용한 정보를 가지면서도 대개 테이블의 크기는 작아서 보다 효율적인 질의 처리가 가능하다.

데이타 조작어(DML)에 대한 파싱 트리 객체는 보다 효율적인 실행을 위해서 질의 그래프 객체를 생성하는 일반화 과정을 거치는데, 이 과정도 정당성 검사와 병행하여 수행된다. 정당성 검사를 오류없이 완료하고 나면 파싱 트리 객체에 대한 질의 그래프 객체가 생성된다.

데이타 정의어(DDL)에 대한 파싱 트리 객체는 직접 처리하는 것이 보다 효율적이므로 따로 질의 그래프 객체를 생성하지 않고 실행을 위한 멤버함수를 직접 갖고 있다.

#### 3.2.4 질의 그래프 객체(Query Graph Object)

데이타 조작어의 경우 파싱 트리 객체로부터 생성된 질의 그래프 객체에는 보다 효율적인 실행을 위해 질의 그래프의 내용을 변경하는 최적화 멤버함수, 실제 질의 수행을 담당하는 실행 멤버함수 등이 있다. 질의 그래프 객체를 위한 클래스들은 파싱 트리 객체의 의미를 그대로 포함하면서도 시스템이 보다 처리하기 쉽도록 설계되어 있다.

#### 3.2.5 카탈로그 관리자(Catalog Manager)

현재 SNU-SQL-Engine에서는 데이타베이스내의 모든 테이블과 칼럼 정보를 저장하는 테이블 카탈로그, 칼럼 카탈로그, 각 테이블에 생성된 인덱스 정보를 저장하는 인덱스 카탈로그, 뷰 정의 SQL문을 저장하는 뷰 정의 카탈로그, 뷰의 기본 테이블에 대한 종속성을 저장하는 뷰 종속성 카탈로그, 테이블과 컬럼에 대한 사용자의 권한정보를 저장하는 테이블 권한 카탈로그, 컬럼 권한 카탈로그를 포함한 7개의 시스템 카탈로그를 유지한다.

카타로그 관리자는 저장 시스템에 문자열 형태로 저장되어 있는 카탈로그 투플들을 실제로 메모리 내에서 이용가능한 논리적인 카탈로그 객체들로 변환하는 역할을 담당하는 멤버 함수를 갖고 있다. 또한, 이들 카탈로그 객체들을 메모리 내에 유지하고 교체하는 작업을 담당한다.

### 3.2.6 권한 관리자(Authorization Manager)

데이타베이스는 여러명의 사용자가 접근하기 때문에 시스템은 정보보안을 위해 정당한 자격이 있는 사용자만이 정보에 접근할 수 있도록 해야 한다. SNU-SQL-Engine에서는 대부분의 관계형 데이타베이스가 지원하는 방법인 Grant/Revoke 모델에 기반한 권한모델[7]을 지원하는데, 테이블과 뷰 단위의 권한관리 뿐만 아니라 각 컬럼 단위로도 권한관리를 지원한다.

### 3.2.7 테이블 관리자(Table Manager)

테이블이란 실제로 디스크에 화일로 저장되어 있는 릴레이션으로서 투플들의 집합이다. 디스크에 저장되어 있는 투플들은 단순한 문자열로서 그 자체로는 아무런 의미를 갖지 못하고 시스템 카탈로그들로부터 필요한 정보를 찾아서 메모리 내에서 하나의 논리적인 테이블 객체를 생성한 뒤에야 비로소 그 의미를 갖게 된다. 다음은 메모리내의 투플과 테이블 객체를 위한 C++ 클래스 구조의 예를 보여주고 있다.

```
class Tuple        // 테이블 내의 투플을 위한 클래스
{
protected:
   char* record;   // 실제 투플값이 저장된 메모리에 대한 포인터
   RID rid;        // 테이블 내의 투플 ID
   ⋮
};

class Table                  // 테이블을 위한 클래스
{
   Tuple row[MaxTupleNum];   // 테이블 내에 투플들을 저장하기 위한 배열
   int tuplesize;            // 투플의 크기
   int tuplenum;             // 배열에 저장된 투플수
   Tblcat_node* tblinfo;     // 테이블 카탈로그에 대한 포인터
   FixedSeqFile tblfile;     // 테이블이 실제로 저장되어 있는 화일
   ⋮
};
```

클래스 `Tuple`은 실제 투플을 위한 메모리에 대한 포인터와 실제 테이블 화일에서의 레코드 식별자를 인스탄스 변수로 갖는다. 클래스 `Table`은 한 투플의 크기와 MaxTupleNum개의 투플들을 위한 기억공간, 그리고 현재 객체 내에 포함된 투플의 갯수를 위한 인스탄스 변수들을 갖는다. 또한 테이블에 관련된 모든 카탈로그들의 모임인 클래스 `Tblcat_node`의 객체에 대한 포인터와 저장시스템의 테이블 화일을 위한 객체를 포함한다.

테이블 관리자는 저장 시스템으로부터 읽어들인 물리적인 테이블로부터 각 카탈로그 관리자를 이용하여 메모리 내의 논리적인 테이블 객체를 생성하고, 필요한 투플들을 저장 시스템으로부터 읽어들이고, 변경되거나 새로 생성된 투플들을 저장 시스템에 기록하는 작업을 담당하는 멤버함수를 갖고 있다.

#### 3.2.8 하부 저장 시스템(SESS)

SNU-SQL-Engine에서 하부 저장 시스템으로 사용하고 있는 SESS[4]는 객체 지향 개념을 사용하여 C++로 구현한 확장 용이 저장 시스템으로서 새로운 형태의 화일 구조 추가나 새로운 버퍼 관리 알고리즘의 추가가 용이하다.

SESS는 저장 시스템에서 볼 수 있는 화일 관점에 따라서 전체적으로 크게 3 단계 구조를 가지고 있다. 가장 하위 단계인 페이지 화일 단계는 화일을 단순히 페이지들의 집합으로서 다룬다. 따라서 이 단계에서는 페이지의 내부 구조나 페이지 사이의 연결 상태 등은 다루지 않는다. 두번째 단계는 구조화된 화일 단계로서 다양한 화일 구조를 갖는 화일 객체들과 데이타베이스 객체로 이루어 진다. 여기서 구조화된 화일은 $B^+$ 트리 구조나 해쉬 구조와 같이 일정한 데이타 구조를 이루는 레코드들의 집합으로서 존재한다. 따라서 화일에 대한 연산도 페이지 단위가 아닌 레코드 단위로 이루어진다. 가장 상위의 단계인 인덱스된 화일 단계에서는 구조화된 화일 단계에서 제공하는 데이타 화일과 $B^+$ 트리와 같은 인덱스 화일을 결합하여 이를 가상의 인덱스된 화일로써 보여준다. 또한 이 단계에서는 탐색 조건을 지정하여 원하는 레코드들을 쉽게 검색할 수 있는 방법도 제공한다.

SESS에서는 $B^+$ 트리와 extendible 해쉬 두가지의 단일 키 인덱스를 지원하고 임의의 사용자 정의 키를 레코드 탐색 키로 사용할 수 있다. 이외에 K-D-B, Grid, R-트리와 같은 다중 키 인덱스는 구현 중이다. 임의의 사용자 그리고 SESS의 버퍼는 전역적인 관리는 물론, 각 화일 별로 버퍼를 할당 관리할 수 도 있다.

표 1은 SESS와 위스콘신 대학에서 개발한 저장시스템인 WISS의 성능을 Sun SPARC, SunOS 4.1.2 상에서 같이 실행하여 비교한 내용이다. WISS에서는 고정 길이 순차 화일을 지원하지 않아 이에 대한 비교는 수행하지 못했다. 표에서 알 수 있듯이 SESS의 성능이 전반적으로 WISS에 비해 뛰어나다.

표 1: SESS와 WISS의 성능 비교

| | 테스트 연산 | 경과 시간 (초) | | 성능비 |
|---|---|---|---|---|
| | | SESS(A) | WISS(B) | (A/B) |
| 가변 길이 순차 화일 | 1,200개 레코드에 대해 삽입, 추가, 삭제, 갱신 | 2.80 | 3.20 | 0.86 |
| B+ 트리 | 40,000개의 정수 키에 대해 삽입, 검색, 삭제 | 137.40 | 155.30 | 0.88 |
| B+ 트리 (같은 키값을 허용) | 1000개의 같은 키에 대해 삽입, 검색, 삭제 | 6.10 | 69.70 | 0.86 |
| Extensible 해쉬 | 40,000개의 정수키에 대해 삽입, 검색, 삭제 | 32.70 | 26.70 | 1.22 |
| Large object | 1,000개의 세그먼트의 삽입, 삭제, 읽기 | 3.20 | 3.50 | 0.91 |

### 3.3 다양한 최적화 전략

여기서는 SNU-SQL-Engine에서 제공하는 다양한 최적화 전략을 설명한다. 우선 다음과 같은 기본적인 최적화 전략을 제공하고 있다.

1. 프로젝션은 될 수 있는 한 빨리 수행하여 중간 결과 크기를 줄인다[8].
2. 보다 적은 중간 결과를 생성하는 술어들을 먼저 처리하여 중간 결과의 크기를 줄인다[8].
3. 중첩 질의를 여러 개의 단일 레벨 질의들로 변환한 다음에 처리하여 중복되는 계산 과정을 제거한다[9][10].

위와 같은 간단한 최적화 전략이외에 SNU-SQL-Engine은 몇몇 상용시스템이나 프로토타입들에서 지원하는 다양한 고급의 최적화 전략을 제공하므로써 질의의 타입에 따른 효율적인 수행을 가능하게 해준다.

1. 다양한 인덱스를 사용할 수 있다. SNU-SQL-Engine는 인덱스를 생성하는 CREATE INDEX 문에서 사용자가 원하는 타입의 인덱스를 명시할 수 있도록 구문을 다음과 같이 확장하였다.

   CREATE [UNIQUE] INDEX 인덱스-이름 ON 테이블-이름 (컬럼 [, 컬럼... ]) [인덱스-타입]

   현재 인덱스-타입으로는 $B^+$ 트리 인덱스와 extendible 해쉬 인덱스가 있는데 아무런 명시가 없을 경우에는 $B^+$ 트리 인덱스를 생성한다. 또한 클래스의 상속성과 가상 함수를 이용하여

각 인덱스 타입에 대한 클래스를 설계하여 새로운 타입의 인덱스를 추가하기가 쉽다는 특징을 갖는다.

2. 다양한 죠인 기법들을 사용한다. SNU-SQL-Engine는 중첩-블럭 죠인, 정렬-합병 죠인, 하나의 인덱스를 사용한 죠인, 두개의 인덱스를 사용한 죠인과 같은 4가지 죠인 기법들[11]을 제공하는데, 실제 죠인을 수행할 때에는 죠인의 대상이 되는 테이블들의 투플수와 죠인에 사용되는 컬럼들에 인덱스가 있는지에 따라 그 중에서 가장 효율적인 것을 선택하여 수행하게 된다.

3. 다양한 테이블 정렬 방법들을 사용한다. SNU-SQL-Engine가 사용하는 테이블 정렬 방법으로는 2원 균형 합병 정렬과 $B^+$ 트리 인덱스를 이용한 정렬 방법의 2가지가 있어서 정렬의 기준이 되는 컬럼 수와 대상 테이블의 크기에 따라 보다 효율적인 방법을 선택하여 사용한다.

이상에서 구현한 질의 처리의 최적화 전략들이외에, 새로운 최적화 전략을 질의그래프 객체의 멤버함수로 정의하므로써 쉽게 추가할 수 있다. 이는 객체지향 기법을 이용해서 설계 및 구현한 SNU-SQL-Engine의 큰 장점 중의 하나이다.

# 4 적합성 및 성능 평가

3장에서는 SNU-SQL-Engine의 전체 구성과 각 구성요소들의 기능에 대해 알아보았다. 여기서는 구현된 SQL 처리기의 SQL 표준에 대한 적합성과 특정 상용 데이타베이스 시스템과의 성능평가 결과를 보이겠다.

## 4.1 적합성 평가

SQL 처리기의 적합성이란 그것이 SQL 표준을 얼마나 따르는지를 나타내는 것으로서 적합성이 높을수록 사용이 용이하고, 다른 DBMS의 데이타들을 사용하거나 다른 DBMS로 대체하기가 쉽다. 따라서 새로 개발되는 SQL 처리기에 대해서는 적합성 검사를 거쳐서 보다 좋은 적합성을 갖도록 개선하는 과정이 반드시 필요하다.

### 4.1.1 적합성 평가 방법

SNU-SQL-Engine은 SQL1을 기본으로 구현되었으므로 적합성을 평가하기 위해서는 SQL1에 맞는 다양한 질의들을 올바르게 처리할 수 있는지 검사해야 하는데, 그 질의들은 임의로 만들어 사용하는 것이 아니라 적합성 평가를 위한 공인된 표준 프로그램을 사용한다.

SNU-SQL-Engine의 적합성 평가에는 SQL1을 채택한 미국의 FIPS PUB 127-1의 적합성을 검사하기 위해서 NIST에서 개발한 SQL Test Suite[12]를 사용하였는데, SQL Test Suite는 테스트 데이타베이스를 위한 스키마와 데이타들, 그리고 다양한 SQL문과 그 실행 결과로서 구성된다. 검사용 SQL문에는 대화형 SQL에 관한 것 뿐만 아니라 내장(embedded) SQL에 관한 것들도 있는데, SNU-SQL-Engine은 대화형 SQL만을 지원하므로 그것에 대해서만 적합성 평가를 실시하였다.

### 4.1.2 적합성 평가 결과

NIST SQL Test Suite의 대화형 SQL에 대한 검사용 SQL문에는 데이타 조작어에 관한 것과 제약조건(integrity constraint)에 관한 것들이 있는데, 여기서는 데이타 조작어에 관한 199가지 SQL문 중에서 SQL1 표준에 포함되어 있지 않은 21가지 경우를 제외한 178가지 경우에 대해서만 적합성 검사를 실시하였다. 그 결과 SNU-SQL-Engine은 135개의 SQL 문을 제대로 수행하여 약 75.8%의 적합성을 보였다. 구현한 것들 중에서 중첩된 EXISTS 연산과 같이 구현상의 오류로 제대로 처리하지 못하는 기능, 아직 구현하지 않은 기능 등이다.

SNU-SQL-Engine에서 제공하지 않는 데이타 타입은 정밀도(precision)와 기수(base)를 갖는 DOUBLE, FLOAT, NUMERIC, DECIMAL 등이 있는데, 현재로서는 C++에서 제공하는 데이타 타입(int, char, float ...)들을 그대로 사용하기 때문에 임의의 정밀도와 기수를 제공하기가 어렵다.

COMMIT과 ROLLBACK 기능을 따로 제공하지 않으며, 별도의 스키마 기능을 제공하지 않고 개념적으로 전체 데이타베이스를 하나의 스키마로 취급한다.

WHERE절에서 술어의 왼쪽 피연산자가 수식인 경우나 컬럼이 아닌 리터럴인 경우를 처리하지 못하고, HAVING절에서는 후자의 경우만을 처리하지 못한다. 또한 선택절에 컬럼이 아닌 문자열이나 숫자가 단독으로 사용된 리터럴의 경우도 처리하지 못한다.

표 2: 테스트용 테이블들의 정보

| 테이블 이름 | 투플 크기(Byte) | 투플수 | 테이블 크기(Byte) |
|---|---|---|---|
| 소속 | 92 | 616 | 56,672 |
| 강의시간 | 73 | 26,140 | 1,908,220 |
| 교과과목 | 162 | 14,329 | 2,321,298 |

### 4.2 성능 평가 및 결과

시스템의 성능은 일반적으로 벤치마크에 의해 평가되지만, SNU-SQL-Engine은 완전한 데이타베이스 시스템이 아니기 때문에 UNIX의 *time* 명령어를 사용하여 SPARC2 SunOS 4.2에서 대략적인 성능을 측정하였다. *time* 명령어는 프로그램이 실행되는데 걸린 시간과 디스크 I/O 횟수, 페이지 오류와 스왑핑이 발생한 횟수 등을 계산한다. UNIX와 같은 다중 태스킹 환경에서는 다른 태스크들이 시스템 성능에 많은 영향을 미치기 때문에, 이와 같은 방법에 의한 성능 평가가 타당성을 갖기 위해서는 다른 태스크들이 거의 없는 시간을 택해서 실제 응용에서 사용되는 많은 양의 데이타를 대상으로 다양한 경우의 질의를 여러번 수행시키고 그 결과를, 똑같은 조건에서 다른 데이타베이스 시스템(여기서는 INFORMIX-ISQL를 이용했다.)의 수행 결과와 비교하는 것이 적절하다.

#### 4.2.1 테스트용 데이타베이스 및 질의

성능 평가에 사용된 데이타베이스는 서울대학교 정보시스템에서 실제로 사용하고 있는 수강편람 데이타베이스로서 3개의 테이블로 구성되는데, 각 테이블들에 대한 정보는 표 2에 나타나있다.

성능 평가에 사용된 질의는 가장 기본적인 3가지 경우에 대한 것으로서 다음과 같이 단일 테이블에 대한 질의와 두개의 테이블에 대한 질의, 그리고 세개의 테이블에 대한 질의로 구성된다. 질의 2에서 과목번호는 교과과목 테이블에서는 키이고, 강의시간 테이블에서는 키가 아니다.

**질의1**
```
SELECT COUNT(*)
FROM 강의시간
WHERE 학과 BETWEEN '400' and '600';
```

**질의2**
```
SELECT COUNT(*)
FROM 교과과목 TA, 강의시간 TB
WHERE TA.과목번호 = TB.과목번호;
```

**질의3** SELECT COUNT(*)
FROM 소속, 교과과목, 강의시간
WHERE 소속.학과 = 교과과목.개설학과
AND 강의시간.과목번호 = 교과과목.과목번호;

#### 4.2.2 성능 평가 방법 및 결과 비교

SNU-SQL-Engine과 INFORMIX I-SQL(버전 4.0)의 성능 평가 절차는 우선 각 시스템에 수강편람 데이타베이스를 적재한 다음에 각 질의를 10번씩 수행시켜 각각의 경과시간을 측정한 뒤에 최소값과 최대값을 제외한 나머지 8가지 경우의 평균값을 계산한다. 다음으로 각 질의에서 사용할 수 있는 모든 인덱스를 생성하여 앞의 과정을 반복한다. INFORMIX는 $B^+$ 트리만을 사용할 수 있지만 SNU-SQL-Engine은 그 외에도 extendible 해슁을 사용할 수 있으므로 사용할 수 있는 인덱스들의 조합이 여러가지가 되는데, 실제로 질의2에 대한 성능 평가에서는 모든 경우에 대해 검사했다.

우선, 성능 평가의 결과의 이해를 돕기 위해, 죠인 칼럼에 대한 인덱스 유무, 인덱스 종류 그리고 테이블의 투플수에 SNU-SQL-Engine의 죠인 전략을 기술한다.

- 죠인에 관련된 두 칼럼 중 하나에만 인덱스가 존재하는 경우: 인덱스가 존재하는 테이블이 내부블럭이 된다.
- 두 칼럼 모두 같은 종류의 인덱스를 갖는 경우: 투플의 수가 작은 테이블이 외부블럭이 된다. 이때 인덱스가 $B^+$ 트리인 경우에는 외부블럭에 대해 $B^+$ 트리를 이용한 순차접근을 하고 해쉬인 경우에는 외부블럭에 대한 인덱스를 사용하지 않고 테이블에 대한 직접적인 순차접근을 수행한다.
- 두 조인 칼럼에 각각 해쉬와 $B^+$ 트리 인덱스가 존재하는 경우, $B^+$ 트리 인덱스가 존재하는 테이블이 외부블럭이 되고 마찬가지로 $B^+$ 트리를 이용한 순차접근을 한다.
- 인덱스가 존재하지 않는 경우, 투플의 수가 많은 테이블이 외부 블럭이 된다.

표 3는 성능평가의 결과를 보여주고 있다.

표 3에서 인덱스가 없을 경우의 질의 처리 성능은 SNU-SQL-Engine이 INFORMIX I-SQL 보다는 약 2배 정도 빠른 것으로 나타났다. 인덱스가 있을 경우에 질의1의 처리 속도는 SNU-SQL-Engine이 다소 느린 것으로 나타났는데, 그 이유 중의 한가지는 SNU-SQL-Engine에서는

표 3: SNU-SQL-Engine과 INFORMIX I-SQL의 성능 비교

| 질의 | 인덱스 | 경과 시간 (초) | | 성능비 |
|---|---|---|---|---|
| | | SNU-SQL-Engine(A) | INFORMIX(B) | (A/B) |
| 질의1 | 없음 | 3.1 | 5.7 | 0.65 |
| | 학과에 $B^+$ 트리 | 3.0 | 1.7 | 1.76 |
| 질의2 | 없음 | 31.63 | 76.75 | 0.41 |
| | TA에만 $B^+$ 트리(1) | 19.13 | 21.88 | 0.87 |
| | TB에만 $B^+$ 트리(2) | 10.63 | 14.5 | 0.73 |
| | TA,TB 둘다 $B^+$ 트리(3) | 12.0 | 12.5 | 0.96 |
| | TA에만 해쉬(4) | 16.13 | | |
| | TB에만 해쉬(5) | 12.5 | | |
| | TA,TB 둘다 해쉬(6) | 12.5 | | |
| | TA에 $B^+$ 트리,TB에 해쉬(7) | 13.88 | | |
| | TA에 해쉬,TB에 $B^+$ 트리(8) | 8.25 | | |
| 질의3 | 없음 | 44.13 | 92.37 | 0.48 |
| | 사용되는 모든 컬럼에 $B^+$ 트리 | 18 | 28 | 0.64 |

BETWEEN 술어를 두 개의 비교연산자를 가진 술어들로 변경하여 처리하는데 아직까지는 하나의 술어에 대해서만 인덱스를 사용할 수 있기 때문에 처리 속도가 다소 느리다. 질의2와 질의3의 수행 결과 둘 다 SNU-SQL-Engine이 INFORMIX I-SQL보다 나은 성능을 갖는 것으로 측정되었다. 질의 2에서, (1)과 (2) 경우 모두 $B^+$ 트리 인덱스가 있는 테이블이 내부블럭이 된다. 이때, 이 두 경우의 성능차이는 (1)의 외부 블럭인 TB의 투플수가 (2)의 외부블럭인 TA의 투플수보다 많다는 사실에 기인한다. (3)의 경우에는 외부블럭 TA에 대해 $B^+$ 트리를 이용한 순차접근을 수행하므로 (2)의 테이블에 대한 직접적인 순차접근보다 효율이 떨어짐을 알수 있다. (5)와 (6)의 경우에는 둘다 TA를 외부 블럭으로 순차접근을 하고 TB에 대한 해쉬를 이용하므로 결과는 동일한 반면, (4)의 경우 순차접근하는 외부 블럭인 TB가 (5)와 (6)의 외부 블럭인 TA보다 크기때문에 속도가 떨어짐을 알 수 있다. TA에 해쉬 인덱스, TB에 $B^+$ 트리 인덱스를 갖는 (8)이 가장 좋은 성능을 보이는데, 이는 TB를 외부블럭으로 해서 $B^+$ 트리를 이용한 순차접근을 수행하고 내부블럭인 TA의 레코드들을 해쉬 인덱스를 이용해서 조인하기 때문이다. (7)의 경우, TA를 외부블럭으로 해서 $B^+$ 트리 인덱스를 이용한 순차접근을 수행하는데, (8)의 경우 보다 시간이 많이 걸림을 볼 수 있다. (8)의 경우 과목번호가 같은 여러개의 투플이 $B^+$ 트리를 통해서 한꺼번에 죠인되기 때문에[2] (7) 보다 속도가 빠르다. 표에서 (6)의 경우가 (7)보다 성능이 나은 것으로 나타났는

[2]강의시간 테이블 TB는 과목번호와 강의시간에 대해 클러스터링되어 있음.

데, 이는 TB가 내부 블럭으로 사용되고 앞의 경우와 마찬가지로 해쉬 인덱스의 성능이 $B^+$ 트리 인덱스보다 낫기 때문이다. 마지막으로 (4)와 (8)의 경우 둘 다 TB를 외부 블럭으로 해서 내부 블럭 TA에 대한 해쉬를 이용한 접근을 수행하는데, (8)의 경우 앞에서 설명한 것처럼 과목번호에 대한 클러스트링이 되어 있기 때문에 이 경우에도 속도 차이가 난다.

성능 평가 결과는 SNU-SQL-Engine이 INFORMIX I-SQL보다 다소 좋은 성능을 보이고 있는데 그것은 다음과 같은 이유때문이다.

- SNU-SQL-Engine에서는 여러 죠인 기법과 인덱스 타입, 그리고 테이블 정렬 기법 등 다양한 질의 처리기법들을 제공한다.
- 표 1에서 알 수 있듯이 저장시스템 SESS의 성능이 우수하다.
- SNU-SQL-Engine은 단일 사용자용 대화형 SQL 처리기로서 최소한의 필요한 기능만을 제공하고 있는데 비해서 INFORMIX는 다중사용자용 DBMS로서 갖추어야 하는 추가 기능들로 인한 실행부담(예를 들어, 동시성제어, 고장회복, 권한관리 등)을 안고 있기 때문이다.[3]

## 5 객체지향적 설계 및 구현의 장점

SNU-SQL-Engine의 실제 구현은 200개 이상의 C++ 클래스들의 계층구조를 이루어져 있는데, 이 클래스들은 주로 파싱 트리 객체, 질의 그래프 객체, 테이블 및 카타로그 관리자, SQL에서 사용되는 수식이나 술어 등의 표현을 위한 것들이다. 이렇게 많은 클래스가 존재하는 이유는 객체지향적 방법론의 원칙에 따라 SQL 질의에 존재할 수 있는 모든 가능한 IS-A 관계를 찾아서 이를 클래스 계층구조에 반영했기 때문이다. 객체지향 개념을 통한 SNU-SQL-Engine 구현의 장점으로는 정보의 중복을 피할 수 있고, 구현이 쉽고, 확장성이 좋고, 프로그램의 이해가 쉽다는 점 등을 들 수 있는데, 이와 같은 특징들을 파싱 트리 객체를 위한 클래스를 예로 들어 설명한다.

### 5.1 정보중복의 제거

그림 4는 SNU-SQL-Engine의 SELECT문과 관련한 파싱 트리 객체의 부분적인 클래스 계층구조를 보여주고 있다. 이는 각각의 SQL 문에 반드시 필요한 자료구조만을 제공하므로 불필요한

[3] 하지만, 평가에 사용된 질의들이 SELECT문들로만 이루어졌기 때문에 이러한 실행부담이 그리 크지 않다.

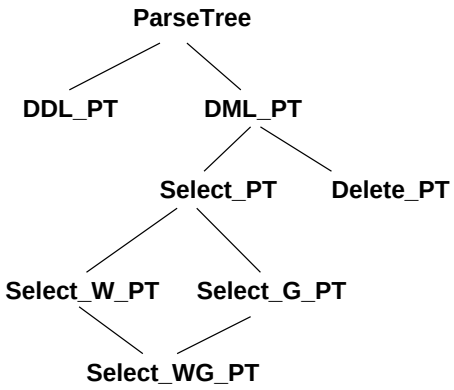


그림 4: 파싱 트리의 계층구조

기억공간의 낭비를 방지해준다. 예를 들어, WHERE절을 포함하지 않는 SELECT문의 경우에 SELECT절과 FROM절을 표현할수 있는 SELECT 파싱 트리 클래스의 객체가 생성된다. 반면에 기존의 SQL 처리기의 구현은 가장 일반적인 파싱 트리의 자료구조를 정하고, 질의의 종류에 관계없이 이 자료구조를 통해서 파싱 트리를 표현한다. 또한, 이와 같이 해당 질의에 반드시 필요한 자료구조만을 갖는 파싱 트리 객체를 생성하므로써 각각의 멤버함수의 구현이 간단해지고, 질의의 목적에 맞는 최적화를 할 수 있는 부가적인 장점이 있다. 기존의 SQL 처리기의 구현방법을 따르면 각각의 자료구조에 해당 정보가 있는지를 검사하는 case 문이 많을수 밖에 없는 반면, 본 SQL 처리기의 각 멤버함수는 단지 해당 파싱 트리 객체에 필요한 동작만을 수행하므로써 불필요한 case 분석을 줄일수 있다.

### 5.2 구현의 편의

앞 절에서 설명한 멤버함수에서의 case 분석의 제거이외에도, SQL 처리기에 필요한 모든 자료구조를 객체화하므로써 전체적으로 프로그램의 구현이 쉬워진다는 장점이 있다. 그림 5는 뷰 변환을 위한 클래스 계층구조와 멤버함수의 인터페이스를 보여주고 있다. 그림 5에서 Select_PT의 뷰 변환을 담당하는 멤버함수 transform()은 자신의 멤버인 target과 tableList의 멤버함수 transform()을 호출하고, 이들은 마찬가지로 자신의 멤버인 Column과 Table의 transform() 멤버함수를 호출한다. 이들 멤버함수들은 실제 자신이 표현하는 칼럼과 테이블 이름을 기본테이블의 칼럼 이름과 테이블 이름으로 바꾼다. 즉, 하나의 함수가 전체적인 파싱 트리의 구성요소들을 관리하면서 뷰 변환을 담당하는 것이 아니라 각각의 객체가 뷰 변환시 자신이 담당해야 할 역할만

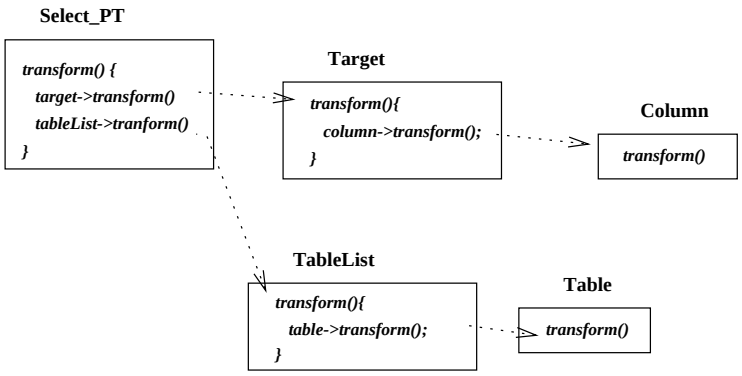


그림 5: 뷰 변환을 위한 파싱 트리의 계층구조

을 transform() 멤버함수에 코딩하면 되므로 멤버함수의 구현이 쉬워진다.

### 5.3 확장성

SNU-SQL-Engine의 확장성과 관련해서 3.3절에서 질의어 처리를 위한 새로운 최적화 전략을 질의 그래프 객체의 멤버함수로 쉽게 추가할 수 있음을 설명했다. 이외에도 객체지향적 설계 및 구현은 다음 예와 같은 확장성을 제공한다. SNU-SQL-Engine에서는 처음에 뷰를 지원하지 않다가 나중에 뷰 기능을 추가하게 되었다. 그런데, 개념적으로 '뷰 is-a 테이블'의 관계가 성립하므로, 뷰 생성을 위한 파싱 트리 클래스를 테이블 생성을 위한 파싱 트리 클래스에서 상속을 통해서 추가할 수 있었다. 이때 뷰의 기본 테이블로서 수행해야할 기능들을 테이블 클래스에서 상속받은 멤버함수를 이용하고, 기본테이블의 멤버함수중에서 무시(override)해야할 부분(예를 들어, 실제 저장시스템에 해당 테이블을 위한 화일을 생성하는 작업)을 재정의하고, 뷰 생성에 고유한 작업(예를 들어, 뷰와 관련한 시스템 카타로그에 정보를 추가하는 작업)을 위한 멤버함수를 추가하므로써 뷰 생성 파싱 트리 클래스가 클래스 계층구조에 쉽게 추가될 수 있었다. 이와 같이 이전에 지원하지 않던 SQL의 새로운 표준기능을 객체지향적으로 설계하고 구현하므로써 쉽게 추가할 수 있음을 알 수 있다.

### 5.4 프로그램의 쉬운 이해

객체지향적으로 설계된 SNU-SQL-Engine의 경우 클래스 계층구조를 통해 전체적인 SQL 처리기의 구조를 쉽게 파악할 수 있고, 각 멤버함수의 구현내용을 이해하기가 쉬워진다. 5.1절에서 설

명한 것처럼 각각의 파싱 트리 객체나 질의 그래프 객체는 해당 질의 처리를 위해 필요한 자료구조만을 유지하고, 뷰 변환, 최적화, 실행 등을 담당하는 멤버함수들은 지역적으로 수행해야할 기능만을 구현하고 있다. 또한, 5.2절의 뷰 변환 예에서 알수 있듯이 멤버함수의 구현이 개념적인 알고리즘과 유사하게 이루어지고 있다. 이러한 이유 때문에 소프트웨어 공학의 측면에서 원시코드 작성자이외의 다른 사람이 이해하기가 쉽다.

## 6 결론

본 논문에서는 SNU-SQL-Engine의 객체지향적인 설계 및 구현 내용과 이전의 SQL 처리기 구조와의 차이점을 다루었다. 또한, SNU-SQL-Engine에 대한 표준 대화형 SQL1의 데이타 조작어 기능의 적합성검사 결과를 보이고, 학교에서 실제로 사용되는 구체적인 데이타베이스를 이용한 질의 수행결과를 다중 사용자용 INFORMIX I-SQL(버전 4.0)과 비교한 결과를 보였다. 그리고, 기존의 SQL 처리기 설계와 구현 방식에 비해, 객체지향적인 설계 방법이 갖는 여러가지 장점, 즉 정보중복의 제거, 구현의 편의성, 확장성, 프로그램의 쉬운 이해 등을 기술했다.

앞으로 SNU-SQL-Engine을 보다 개선하기 위해 다음과 같은 작업들을 수행할 예정이다. 첫째, 아직 제공하지 않는 표준 기능들을 추가한다. 보다 완전한 SQL을 지원하는 SQL 처리기를 구현하고, 나아가서 SQL2로의 확장도 필요하다. 예를 들어, 외래(foreign) 키와 같은 무결성 제약조건의 효율적인 지원을 위한 트리거 기능의 추가는 반드시 필요하다. 둘째, 카타로그 관리자 객체나 테이블 관리자 객체 등은 제어위주의 역할을 담당하는 객체로 모델링되어 있는데, 이는 기존의 순차적 패러다임을 완전히 배제하지 못한 설계 방법이다. 앞으로 이들 객체를 데이타 위주의 객체로 모델링하는 방안에 관한 연구가 필요하다. 세째, 시스템의 성능향상을 들 수 있다. 아직까지는 인덱스를 이용하는 방법에서 개선해야 할 점이 있다. 의미분석을 통한 효율적인 질의 처리가 가능하도록 하는 최적화 기법도 필요하다. 특히 이 과정에서 앞에서 주장한 바와 같이 객체지향 개념의 장점인 확장성을 최대한 활용할 수 있을 것이다. 네째로 시스템의 안정성을 높인다. 어떠한 환경에서 어떤 질의에 대해서도 시스템이 올바르게 동작하고 예기치 못한 경우도 적절하게 처리할 수 있도록 해야 한다. 마지막으로, SNU-SQL-Engine의 설계 및 구현의 경험을 ODMG[13]에서 제안한 OQL 처리를 위한 SNU-OQL-Engine의 개발에 적용할 것이다.

## 참고 문헌


[1] C.J.Date, A Guide to the SQL Standard, 2nd Ed., Addison-Wesley, 1989

[2] ISO, Information Technology-Database Language SQL, 1989

[3] 한글informix-SQL 사용자지침서, 다우기술, 1991

[4] 안정호, 확장 용이 저장 시스템의 객체 지향 설계, 서울대학교 석사학위논문, 1992

[5] M.Stonebraker, E.Wong, "Access Control in a Relational Database Management Systems by Query Modification," Proceedings of the ACM National Conference, Vol.3, pp. 180-187 1974

[6] Mumick, "Maintenance View Incrementally," Proceedings of SIGMOD, Vol.22, pp. 157-166, 1993

[7] R.Fagin, "On an Authorization Mechanism", ACM Transactions On Database Systems, Vol.3, No.3,pp.310-319, 1978

[8] M.Jarke, J.Koch, "Query Optimization in Database Systems," ACM Computing Surveys, Vol.16, No.2, pp.111-152, 1984

[9] W.Kim, "On Optimizing an SQL-like Nested Query," ACM Transactions On Database Systems, Vol.7, No.3, pp.443-468, 1982

[10] R.A.Ganski, H.K.T.Wong, "Optimization of Nested SQL Queries Revisited," ACM SIGMOD, pp.23-44, 1987

[11] P.Mishra, M.H.Eich, "Join Processing in Relational Database," ACM Computing Surveys, Vol.24, No.1, pp.63-113, 1992

[12] NIST, SQL Test Suite for FIPS 127-1, Ver.2.0, 1989

[13] Cattel, et al., Object Database Standard:ODMG-93, Morgan Kaufmann Publishers, 1993