관계형 데이타베이스에서 지식관리에 의한 질의 최적화

남 인 길† 이 두 한††

요 약

본 논문에서는 세 가지 종류의 지식을 적절하게 표현하여 데이타베이스 시스템에 저장하고 이를 사용하여 질의를 의미적으로 등가이며 보다 처리 효율이 뛰어난 질의로 변환하는 기법을 제시하였다. 또한 제안된 지식을 사용하여 필수적인 성분이나 연산이 부분적으로 생략된 단순화된 질의를 완전한 질의로 변환할 수 있는 기법을 제시하여 사용자로 하여금 보다 단순화된 질의를 사용할 수 있는 환경을 제공하였다. 단순화된 질의로부터 변환과 최적화를 위해 다루는 지식은 크게 세 가지로 대별되는데, 의미적 무결성 규정과 도메인 무결성 규정을 포함하는 의미적 지식과 관계형 데이타베이스에서의 릴레이션간의 물리적 관계를 표현하는 구조적 지식 그리고 속성의 도메인 정보를 유지하는 도메인 정의이다. 제안된 시스템에서는 이들 지식을 사용하여 질의어의 조건 절에 있는 불필요하거나 중복적인 제한연산(restrictions)이나 조인연산(join)을 제거하거나, 다른 효율적인 연산으로의 대체, 혹은 보다 나은 효율을 위해 부가적인 제한연산이나 조인연산을 추가하여 질의 최적화를 이루게 된다.

Query Optimization with Knowledge Management in Relational Database

In Gil Nam† and Doo Han Lee††

ABSTRACT

In this paper, we propose a mechanism to transform more effective and semantically equivalent queries by using appropriately represented three kinds of knowledge. Also we proposed a mechanism which transforms partially omitted components or expressions into complete queries so that users can use more simple queries. The knowledges used to transform and optimize are semantic, structural and domain knowledge. Semantic knowledge includes semantic integrity constraints and domain integrity constraints. Structural knowledge represents physical relationship between relations. And domain knowledge maintains the domain information of attributes. The proposed system optimizes to more effective queries by eliminating/adding/replacing unnecessary or redundant restrictions/joins.

1. 서 론

대단위 정보를 처리하는 데이타베이스 시스템에서 보다 효율적인 질의 처리를 위한 질의 최적화 기법은 데이타베이스의 고전적 문제로서 오랫동안 꾸준히 연구되어 왔으며 많은 성과가 있었다[1, 2, 3, 4, 5]. 특히 1980년대 중반 이후부터는 의미적 질의 최적화(semantic query optimization)에 관한 연구에 관심이 집중되어왔다[6, 7, 8, 10]. 의미적 질의 최적화란 현재의 데이타베이스에서 적절하게 표현되지 못하는 의미적 무결성 법칙(semantic integrity constraints)에 관한 정보를 체계적으로 표현하여 저장하고 이를 효율적인 질의로 변환하는 기법을 의미한다[7].

질의 최적화의 이론적 분야에서 여러 논문들이 발표되었는데[6, 7, 8], 특히 Wei Sun[7]은 트리

• 이 논문은 1993학년 대구대학교 학술연구조성비에 의한 논문임.
† 정 회 원 : 대구대학교 공과대학 전자계산학과 교수
†† 정 회 원 : 경북대학교 대학원 컴퓨터공학과 박사과정 수료
논문접수 : 1995년 2월 6일, 심사완료 : 1995년 10월 4일.

와 체인질의에서 의미적 질의최적화 기법을 이론적으로 정리하였으며, 본 논문은 이를 바탕으로 관계형 데이타베이스 시스템에서 효율적인 질의 최적화를 이루고자 하였다. 본 논문에서는 [7]에서 사용된 의미적 지식 외에 구조적 지식을 관리하고 이러한 지식을 적절하게 표현할 수 있는 기법을 제안하였다. 또한 제안된 지식을 사용하여 필수적인 성분이나 연산이 부분적으로 생략된 단순화된 질의를 완전한 질의로 변환할 수 있는 기법을 제시하여 사용자로 하여금 단순화된 질의를 사용할 수 있는 환경을 제공하였다.

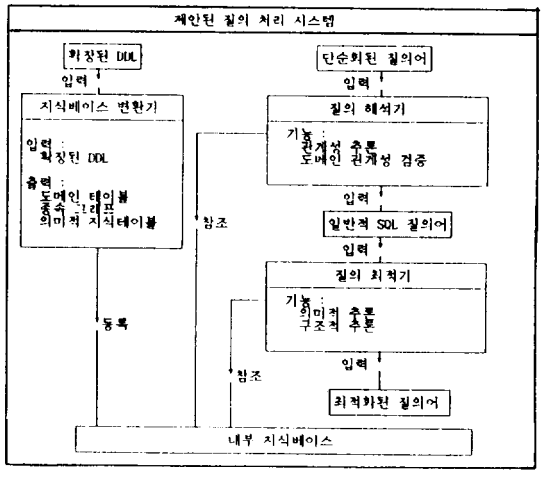
제안한 시스템에서는 질의 최적화를 위해 세 가지 지식을 사용하는데, 먼저 의미적 지식은 도메인 무결성 법칙과 의미적 무결성 법칙을 포함한다. 이러한 의미적 지식은 제안한 시스템에서 질의의 조건 절에 있는 불필요하거나 중복적인 성질을 지닌 제한연산(restrictions)이나 조인연산(join)을 제거하거나, 다른 효율적인 연산으로의 대체, 또는 보다 나은 효율을 위해 부가적인 제한연산이나 조인연산의 추가에 사용되어 진다. 두 번째 지식은 관계형 데이타베이스에서의 릴레이션간의 물리적 관계를 표현하는 구조적 지식이다. 구조적 지식은 바로 데이타베이스 관리자에 의해 생성되고 사용자에게는 어느 정도 은폐되어 있는 스키마 구조와 기본 키(primary key)와 외래 키(foreign key)에 의해 유지되는 관계성 정보를 표현한다. 이러한 지식을 시스템 내부에 이식함으로써 사용자의 스키마에 대한 부담을 줄여 단순화된 질의의 사용을 가능하게 한다. 세 번째는 의미적 지식과 구조적 지식을 구축하기 위한 기반 지식으로 속성의 도메인 정보를 유지하는 지식이다. 이는 확장된 DDL을 통해 데이타베이스 관리자에 의해 제안되며, 단순화된 질의의 변환과 최적화에 이용되어 진다.

본 논문에서는 이들 지식을 질의 최적화에 사용할 수 있도록 하기 위하여 일반적인 관계형 데이타베이스 시스템을 확장하였다. 확장 시스템의 내부 지식베이스는 앞서 설명한 세 가지 형태의 지식을 저장하기 위한 저장구조이며, 확장된 DDL은 지식을 구성하기 위해 필요한 제반 정보를 데이타 정의 단계에서 기술하는데 이용된다. 질의 해석기는 단순화된 질의를 일반적인 질의로 변환하며, 질의 최적기는 지식베이스에 저장된 세 가지 지식을 사용하여 입력된 질의를 보다 효율적인 질의로 변환한다.

본 논문의 구성은 다음과 같다. 2장에서는 시스템의 전반적인 구조를 기술하고, 3장에서는 질의 처리기법에서 사용되어지는 도메인 테이블, 종속그래프와 의미적 지식테이블을 정의한다. 4장에서는 단순화된 질의어의 변환 기법을 기술하고, 5장에서는 질의에 대한 의미적 최적화 기법을 제시한다. 6장에서는 실험을 통하여 제안된 기법들을 검증하고, 7장에서는 본 연구를 통하여 얻어진 결론과 앞으로의 연구방향을 서술한다.

2. 시스템 구조

본 논문에서 제안한 시스템은 크게 지식베이스 변환기와 질의 해석기 그리고 질의 최적기로 구성되어 있으며, 지식베이스 변환기는 확장된 DDL로 정의된 릴레이션의 정의를 입력하여 내부지식베이스를 구축한다. 확장된 DDL은 지식베이스 구축에 필요한 도메인 정보와 기본 키와 외부 키의 정의를 포함하는데 본 논문에서는 [11]에서 소개된 형식을 사용하였다. 질의 해석기는 단순화된 질의를 지식베이스의 검색을 통하여 일반적인 SQL질의어로 번역하는 기능을 지닌다. 그리고 질의 최적기는 SQL질의를 최적화



(그림 1) 시스템 구조
(Fig. 1) System structure

하여 최적화된 탐색경로를 지니는 질의어를 생성시킨다. (그림 1)은 본 논문에서 제안한 질의처리 시스템의 전체적인 처리 구성도이다.

(그림 1)에서와 같이 본 시스템에서 사용될 데이타베이스는 확장된 DDL로 정의되어야 하며, 데이타베이스의 정의는 지식베이스의 변환기를 통하여 내부지식베이스에 저장된다. 이러한 내부지식베이스는 동작시 주기억장치에 유지된다. 릴레이션의 정의가 이루어지면 레코드의 입력이 이루어지고 이러한 레코드에 대해 질의어를 통한 접근이 이루어진다. 단순화된 질의는 질의 해석기에 의해 완전한 SQL로 번역이 이루어지며, 이 과정에서 내부 지식베이스에 구축된 구조적 정보가 이용된다. 번역된 SQL질의어는 질의 최적기를 통하여 최적화 되며 이 과정에서 지식베이스의 의미적 지식이 사용된다.

3. 지식의 정의

지식의 구성요소로는 기본 키를 인식하고 도메인 무결성을 유지하기 위한 도메인 테이블과 질의 해석 시에 릴레이션과의 관계형 정보를 추출하기 위한 종속 그래프와 의미적 지식을 표현하는 의미지식 테이블을 포함한다. 또한 이러한 지식은 내부 지식베이스에 유지되며, 처리 시에 주기억장치에 상주하게 된다. (그림 2)는 제안한 지식들의 설명을 위해 공급자와 부품의 속성명과 도메인 정보를 유지하고 있는 확장된 DDL로 정의한 릴레이션의 예이다.

```
CREATE DOMAIN S_NAME        CHAR(20);
CREATE DOMAIN STATUS        SMALLINT;
CREATE DOMAIN CITY  CHAR(15);
CREATE DOMAIN P_NAME        CHAR(10)
CREATE DOMAIN COLOR CHAR(10);
CREATE DOMAIN WEIGHT        SMALLINT;

CREATE TABLE S
(SNAME      DOMAIN (S_NAME) NOT NULL,
 STATUS DOMAIN (STATUS),
 CITY   DOMAIN (CITY)),
 PRIMARY KEY (SNAME);


CREATE TABLE SP
(SNAME  DOMAIN (S_NAME) NOT NULL,
 PNAME  DOMAIN (P_NAME) NOT NULL),
 PRIMARY KEY (SNAME,PNAME)
 FOREIGN KEY SNAME, PNAME;

CREATE TABLE P
(PNAME  DOMAIN (P_NAME) NOT NULL,
 COLOR  DOMAIN (COLOR),
 WEIGHT DOMAIN (WEIGHT),
 CITY   DOMAIN (CITY)),
 PRIMARY KEY (PNAME);
```

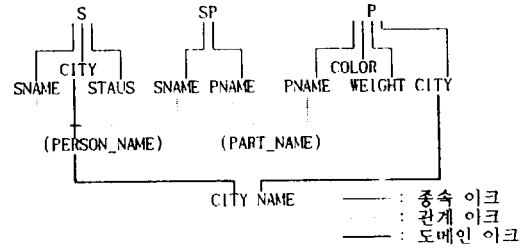
(그림 2) 릴레이션의 예
(Fig. 2) A sample relation

3.1 도메인 테이블

도메인은 하나 혹은 그 이상의 임의의 속성이 가질 수 있는 값들의 풀(pool)로 정의되며, 지금까지의 대부분의 데이타베이스 시스템에서 도메인의 개념은 전적으로 사용자에게 의존하였으나, 본 논문에서는 도메인 테이블을 정의하여 시스템 목록에서 관리하도록 구성하였다. 도메인 테이블에서는 속성명과 도메인명외에도 그 속성이 속한 릴레이션명과 기본.키 여부를 지시하는 플래그가 존재하며, 이러한 테이블의 생성은 시스템의 정의 단계에서 이루어진다.

3.2 종속 그래프

종속 그래프는 관계형 스키마를 표현한 것으로, 이는 데이타베이스 내의 릴레이션의 구조와 릴레이션간의 구조적 관계를 나타낸다. (그림 3)은 (그림 2)를 종속 그래프로 나타낸 것이다.



(그림 3) 종속 그래프
(Fig. 3) Dependence graph

종속 그래프에서는 각각의 릴레이션명과 속성명 그리고 도메인명이 하나의 노드가 되며 각 속성과 속성이 소속된 릴레이션간에는 아크(arc)가 존재하며 이를 종속 아크라 부른다. 또한 도메인이 같은 속성끼리는 같은 도메인임을 명시하는 도메인 노드에 대한 아크가 존재하며 이를 도메인 아크라 부른다. 또한 이들 도메인 아크 중 일부 기본 키와 외래 키의 관계를 갖는 두 노드를 연결하는 아크가 있으며, 이는 도메인 아크이

면서 또 다른 정보를 표현하게 되는데, 이를 관계 아크라 부르며 관계 아크가 포함된 종속그래프는 임의의 두개의 속성간의 관계성을 유추하는데 사용된다. 즉 별개의 두 노드 X와 Y간의 관계성을 추론함에 있어서 그래프 상에 X, Y를 잇는 경로(path)가 존재할 때, 'X와 Y는 관계가 있다'고 하고 이는 X의 값으로 Y의 값을 추출할 수 있음을 의미하고, 그 역도 성립한다. 즉 관계 아크에 의해 생성되는 이러한 관계성은 대칭적(symmetric)이며, 반사적(reflexive)이며, 전이적(transitive)이다.

(그림 3)에서는 편의상 차수(degree)가 1이하인 도메인 노드는 생략하였는데 이는 이론의 전개상 차수가 1이하인 도메인 노드는 전혀 필요치 않기 때문이다.

실제 프로그램 상에서 각 노드는 (그림 4)와 같이 정의 되었다. 세가지 종류의 노드에 존재하는 필드 sibling은 같은 종류의 다음 노드로의 포인터를 유지하며, 이는 각 노드의 검색에서 이용되어 진다. 전역변수로 선언된 F attr, F dmn, F rln은 각 종류의 최초노드로의 포인터를 유지하며, L attr, L dmn, L rln은 각 종류의 마지막 노드로의 포인터를 유지한다. 도메인 노드(domain node)에서 *attribute[NOOFATTR]은 임의의 도메인에 소속된 모든 속성들로의 포인터를 유지하며, 속성노드(attribute_node)에서는 속성이 소속된 릴레이션노드으로의 포인터(*relation)와 소속된 도메인노드로의 포인터(*domain)를 각각 유지하고 있다.

```
struct relation_node {
   char mark;
   char relation[TOKENSIZE];
   struct attribute_node far *attribute[NOOFATTR];
   struct relation_node  far *sibling;
};

struct domain_node {
   char mark;
   char domain[TOKENSIZE];
   struct attribute_node far *attribute[NOOFATTR];
   struct domain_node    far *sibling;
};

struct attribute_node {
   char mark;
   char attribute[TOKENSIZE];
   char primaryflag;
   char foreignflag;
   struct relation_node  far *relation;
   struct domain_node    far *domain;
   struct attribute_node far *sibling;
};

struct attribute_node    far *F_attr, *L_attr;
struct domain_node       far *F_dmn, *L_dmn;
struct relation_node     far *F_rln, *L_rln;
```

(그림 4) 종속그래프 상의 노드정의
(Fig. 4) Node Definition in dependence graph

3.3 의미지식 테이블

의미지식은 속성과 속성간의 의미적 연결과 관련성을 나타내는 지식으로 SQL의 WHERE절에서 사용되는 집계함수, LIKE와 IS NULL과 같은 특별한 조건연산을 제외한 모든 구문과 →(if-then)과 ↔(if-and-only-if)를 사용하여 정의된다.

구문의 정확한 용법을 BNF문법을 사용하여 정의하면 다음과 같다.

```
rule
   : : =expression connective expression
expression
   : : =term | NOT expression | (expression) | ex-
         pression logical-op term
term
   : : =factor comparison-op factor
factor
   : : =constanct | attribute
connective
   : : =→ | ↔
logical-op
   : : =AND | OR
comparison-op
   : : : ==  | < > | > | >= | < | <=
```

(그림 2)와 (그림 3)에서 정의한 릴레이션을 바탕으로 의미 지식의 종류와 구조를 예시하면 다음과 같다.

예 1) 서울에 사는 공급자는 상태가 10이다.
;S.CITY='seoul'→P.STATUS=10

예 2) 부산에 살거나 대구에 사는 공급자의 상태는 50보다 크다.
;P.CITY='pusan' OR P.CITY='taegu'→ S.STATUS>50

4. 단순화된 질의어의 변환

4.1 단순화된 질의어의 특징

본 논문에서 제안한 질의어의 특징은 다음과

같다.

첫째, 기존의 SQL언어의 일반적인 형태인 SFW절에서 이용되는 연산자와 호환성을 갖는다.(단, 집계함수와 [NOT] LIKE ……와 IS [NOT] NULL과 같은 특별한 조건은 제외)

둘째, SFW절에서 FROM부분은 생략이 가능하다.(생략하였을 경우 구조적 지식으로 추론하여 재구성한다.)

셋째, 릴레이션명은 모든 기술에서 모호성(ambiguity)이 없는 한 생략이 가능하다.

넷째, 기본 키와 외래 키로 연결되는 조인 부분은 생략가능하다.

이상과 기본 특징을 지니는 질의어의 자세한 표현 기법을 (그림 1)에서 제시된 데이타베이스를 바탕으로 설명하면 다음과 같다.

먼저 다음과 같은 자연어로 이루어진 질의를 생각하여 보기로 한다.

"적어도 하나 이상의 검은 색 부품을 공급하는 공급자의 이름과 상태를 구하라."

결론적으로 위와 같은 질의에서 데이타베이스에서의 기술적인 측면에서 관심이 있는 문장 성분만을 골라 정리하면 다음과 같다.

COLOR(색)=black(검정)
SNAME(공급자 이름)
STATUS(공급자 상태)

즉 위와 같이 토큰은 자연어로 된 질의어에서 데이타 값(여기서는 black)과 메타데이타(COLOR, SNAME, STATUS 등과 같은 속성 값과 릴레이션명)들이 되며, 이러한 토큰은 몇 가지 간단한 관계연산자 및 논리연산자와 결합하여 질의를 형성하게 되며 이는 다음과 같다.

```
SELECT SNAME, STATUS
WHERE COLOR='black'
```

다음은 단순화된 질의의 일반형이다.

```
select [X1, X2, X3, ···]
where U1 ROP1 V1 [LOP1 U2 ROP2 V2 ···]
```

(D : 데이타 값, M : 속성 명이라 할 때, ROP={=, >, <, >=, <=, <>}, LOP={AND, OR, NOT}이며, U∈M, V∈(DUM), X∈M이다.)

4.2 질의어의 변환기법

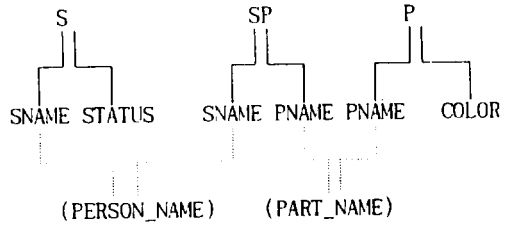
질의어의 변환 기법은 상이한 두 가지 문제로 분리하여 고려할 수 있다. 그것은 질의의 일반형인 'U ROP V'에서 V가 데이타 값일 경우와 V가 속성일 경우이다. 전자는 기본 키와 외래 키에 의한 관계성 연결의 경우이고 후자는 동일한 도메인으로 연결된 경우이다. 본 절에서는 각 경우의 변환 기법을 기술한다.

4.2.1 기본 키와 외래 키에 의한 관계성 연결의 경우

먼저 전자의 경우를 다음에 언급한 질의의 예를 통하여 질의어의 변환기법을 기술한다.

```
SELECT SNAME, STATUS
WHERE COLOR='black';
```

위 질의는 '적어도 하나 이상의 검은 색 부품을 공급하는 공급자의 이름과 상태를 구하여라'에 대한 단순화된 질의어이다. 위 질의에서는 일반적인 SQL질의어에서 표현되어야할 FROM절이 생략되어 있으며 COLOR가 'black'일 때 SNAME, STATUS를 구할 수 있는 연결 고리인 기본 키와 외래 키로 연결되는 조인연산이 생략될 수 있는 것이다. 우리는 이러한 생략된 정보를 지식베이스에 저장된 지식을 사용하여 파악할 수 있다. 그 과정은 우선 각 속성이 소속된 릴레이션 명을 찾는 것으로 이루어지는데, 이는 도메인 테이블을 통하여 쉽게 파악할 수 있다. 밝혀진 각 속성의 릴레이션명은 큐에 저장이 된다. 그리고 SELECT절에 있는 속성들과 조건 절에 기술되어 있는 속성들 사이에 관계아크와 종속아크만으로 이루어진 최단경로를 찾는데, 제시된 예의 경우에는 (그림 6)과 같다. 이러한 경로는 결국 외래 키와 기본 키로 이루어지는 조인연산을 표현하게 될 것이며 이 조인연산은 경로의 길이에 따라 여러개가 될 수도 있다. 또한 이러한



(그림 5) SNAME과 PNAME 사이의 경로
(Fig. 5) Path between SNAME and PNAME

경로 상에 나타난 릴레이션명은 앞에서 구한 큐에 저장된 릴레이션명과 더불어 FROM절을 구성하게 된다. 이 상과 같이 구해진 조인연산과 릴레이션이 바로 단순화된 질의어에서 생략된 조건이 되며 이를 부가하여 일반적인 질의어를 생성할 수 있다.

결국 이러한 과정을 거쳤을 때, 다음과 같은 질의어가 생성될 수 있다.

```
SELECT S.SNAME, S.STATUS          ← 원래의 결론부
FROM   S, SP, P                   ← 생성된 FROM절
WHERE P.COLOR='black' AND         ← 원래의 연산
      P.PNAME=SP.PNAME AND ┐      ← 추가된 연산
      SP.SNAME=S.SNAME ; ←─┘
```

위 결과에서 생성된 FROM절은 프로그램상에서 결론부의 모든 속성을 도메인 테이블상에서 찾아 발견된 테이블명(알고리즘에서 GRQ)과 발견된 경로상의 속성들의 테이블명(알고리즘에서 CRQ)을 합하여 구성되어 지며, SELECT절과 WHERE절에서 생략된 테이블명 역시 도메인테이블상에서 발견하여 부가된다. 추가된 조인연산은 탐색된 경로상(알고리즘에서 P)에서 동일한 도메인 노드로 연결된 속성의 '=' 연산으로 구성되어 진다.

이러한 변환기법의 알고리즘은 다음과 같다.

```
queue GRQ, CRQ;
// GRQ : Queue for relation name in Goal
// CRQ : Queue for relation name in Condition
Explicity(char lopd[ ],char ropd[ ])
// lopd : left operand of a expression in where clause
// ropd : right operand of a expression in where clause
{
  char temp[TOKENSIZE],token[TOKENSIZE];

  construct a GRQ;// select절의 모든 속성들의 릴레이션명을
  GRQ에 ADD한다.
  if (lopd is literal?) {
    if (relation name explicit in ropd?) {
    extract a relation name N in ropd;
    if (not exist N in GRQ and CRQ?)
      add N in CRQ;
}
else if (search relation name N attached a attribute on ropd,
       in dependency graph) {
       if (not exist N in GRQ and CRQ?)
         add N in CRQ;
    }
    else
      ErrorMessage("Error : Not Found Relation name !!!\n");
}
if (ropd is literal?) {
  if (relation name explicit in lopd) {
    extract a relation name N in ropd;
    if (not exist N in GRQ and CRQ?)
      add N in CRQ;
    }
    else if (search relation name N attached a attribute on lopd,
          in dependency graph) {
          if (not exist N in GRQ and CRQ?)
            add N in CRQ;
    }
    else
      ErrorMessage("Error : Not Found Relation name !!!\n");
}
  for (each relation name : N in GRQ) {
    if (relation name : N is exist in GRQ?)
      continue;
    else {
      if (find a path P between N and a relation name in
        GRQ?)
        the path P is a additional join operation
      else
        ErrorMessage("Error : Not Found any path!!!\n");
    }
  }
}
```

4.2.2 동일한 도메인으로 연결된 경우

V가 속성일 경우는 다음과 같은 질의어를 통하여 기술한다.

'부품과 공급자의 도시가 같을 경우 공급자의 이름과 부품이름의 쌍을 구하여라.'

위 질의를 단순화된 질의어로 표현하면 다음과 같다.

```
SELECT SNAME, PNAME
WHERE S.CITY=P.CITY;
```

위 질의는 메타 데이타간의 직접 연결이 조건부에 존재하는 경우이며, 추론의 여지가 없이 질의의 합법성을 검사하기 위해 단지 두 속성간의 도메인의 비교 검사를 하면 된다. 처리하는 과정은 다음과 같다. 먼저 SELECT절에 포함된 속성들의 릴레이션을 도메인 테이블로부터 취득하여 GRQ에 넣고 WHERE절에 포함된 속성들의 릴레이션을 추출하여 CRQ에 넣는다. 그리고 WHERE절에 나타난 두 피연산자(속성)를 종속그래프 상에서 찾아 이 두 노드를 연결하는 도메인 아크로 이어진 경로를 찾는다. 경로가 존재할 경우 이 연산은 합법적인 것으로 인정되어지며, 만

일 경로가 존재하지 않는다면 질의는 시스템에 의해 거부되어 진다.

알고리즘을 통하여 생성된 결과는 다음과 같다.

```
SELECT S.SNAME, P.PNAME
FROMS,P                          ← 추가된 연산
WHERE S.CITY=P.CITY
```

알고리즘 상에서 추가된 FROM절은 SELECT절의 속성들이 소속된 테이블명(알고리즘에서 GRQ)과 WHERE절의 속성들이 소속된 테이블(알고리즘에서 CRQ)명을 도메인테이블 상에서 발견하여 구성되어지며, 원질의에서 생략된 테이블명 역시 도메인테이블상에서 찾아 부가된다.

이와 같은 과정을 알고리즘으로 표현하면 다음과 같다.

```
queue GRQ, CRQ;
// GRQ : Queue for relation name in Goal
// CRQ : Queue for relation name in Condition
Implicity(char lopd[ ],char ropd[ ])
// lopd : left operand of a expression in where clause
// ropd : right operand of a expression in where clause
{

  if (exist relation name in ropd) {
    extract relation name RR in ropd;
    extract attribute name RA in ropd;
    if (not exist RR in GRQ and CRQ?)
      add RR in CRQ;
  }
  else if (search relation name RR attached a attribute on ropd,
          in dependency graph) {
        move ropd to RA;
        if (not exist RR in GRQ and CRQ?)
          add RR in CRQ;
      }
      else
        ErrorMessage("   : Not Found Relation name!!!\n");
  if (exist relation name in lopd) {
    extract relation name LR in lopd;
    extract attribute name LA in lopd;
    if (not exist LR in GRQ and CRQ?)
      add LR in CRQ;
  }
  else if (search relation name LR attached a attribute on lopd,
          in dependency graph) {
        move lopd to LA;
        if (not exist LR in GRQ and CRQ?)
          add LR in CRQ;
    }
    else
      ErrorMessage("   : Not Found Relation name !!!\n");
    if (find a path constructed only domain edges between
        RR.RA and LR.LA
        in dependancy graph)
        domain integrity check is OK;
    else
      domain integrity check is FAIL;
}
```

5. 질의 최적화

입력된 단순화된 질의어는 4장에서 기술한 알고리즘을 거쳐 일반적인 SQL언어로 변환되며 변환된 SQL질의어는 질의 최적화 과정을 거치게 된다.

질의 최적화의 종류는 크게 세 가지가 있다. 그것은 불필요한 연산의 제거, 보다 효율적인 연산으로의 대체, 그리고 효율을 증진시킬 수 있는 부가적인 연산의 추가가 있다. 이 세 가지 경우 모두 본 연구에서 제안된 기술들이 사용된다.

다음은 5.1, 5.2, 5.3절에서 제시한 질의 최적화 기법을 운용하고 결과를 출력하는 알고리즘이다.

```
void Optimize : : Process()
{

  for (each condition in WHERE clause) {
    Delete();
    Replace();
    Append();
  }
  for (each condition in WHERE clause)
    Find relation name of attributes in condition;
  Make FROM clause with table names;
  Output object code;
}
```

5.1 불필요한 연산의 제거

불필요한 연산의 제거는 중복되거나 혹은 관계성 정보를 통한 불필요한 조인연산의 경우가 된다. 중복되는 연산의 제거를 예를 들어 설명하면 다음과 같다.

```
질의 : SELECT S.SNAME
       FROM   S, SP, P
       WHERE S.STATUS > 50 AND
                P.COLOR='black' AND
                P.PNAME=SP.PNAME AND
                SP.SNAME=S.SNAME;
```

위와 같은 질의가 있고, 또한 지식베이스 내에 '공급자의 상태가 50보다 크면 그 공급자는 검은

색의 부품을 공급한다'는 의미지식이 존재할 경우 이 의미지식에 의해 P.COLOR='black' AND P.PNAME=SP.PNAME AND SP.SNAME=S.SNAME은 중복되는 조건이 되므로 제거되어 질 수 있다. 이에 따른 이득은 두개의 조인연산과 한 개의 제한 연산만큼의 비용절감이 따를 것이다. 다음은 불필요한 연산의 삭제알고리즘이다.

```
int Optimize : : Delete(C)
{
  flag=NO;
  for (each semantic rule : R in semantic table) {
    if (C=R) {
      for (each conditon : G in WHERE clause) {
        if (G=R) {
          delete unnessary condition;
          flag=YES;
          return flag;
        }
      }
    }
  }
}
```

5.2 효율적인 연산으로의 대체

효율적인 연산으로의 대체는 다음과 같은 예를 통하여 설명될 수 있다.

```
질의 : SELECT P.PNAME
       FROM P
       WHERE P.WEIGHT > 5;
```

위의 질의어가 있고 지식베이스 내에 '부품의 WEIGHT가 5인 부품의 COLOR는 모두 'black'이며, 부품의 COLOR가 'black'이면 부품의 weight가 5이다'란 의미적 지식이 있다고 하자. 또한 이 경우 부품 테이블이 COLOR로 인덱싱 되어 있다면, 의미지식에 의해 P.WEIGHT>5라는 조건은 인덱싱을 사용할 수 있는 보다 효율적인 연산이 가능한 조건 COLOR='black'으로 대체될 수 있다.

다음은 효율적인 연산으로의 대체하는 알고리즘이다.

```
int Optimize : : Replace(C)
{
    flag = NO;
    for (each semantic rule : R in semantic table) {
      if (C=R) {
        if (connetive of R='↔' AND
          findindex(attribute of goal of R) {
          replace C with goal of R;
          flag=YES;
          return flag;
        }
      }
    }
}
```

5.3 효율증진을 위한 연산의 추가

실제로 이러한 대체나 불필요한 연산의 삭제 외에도 새로운 연산이 추가되었을 때, 오히려 그 효율이 증대되는 경우도 존재할 수 있다.

```
질의 : SELECT S.STATUS
       FROM   S,SP,P
       WHERE  P.CITY='seoul' AND
              P.PNAME=SP.PNAME AND
              S.SNAME=SP.SNAME;
```

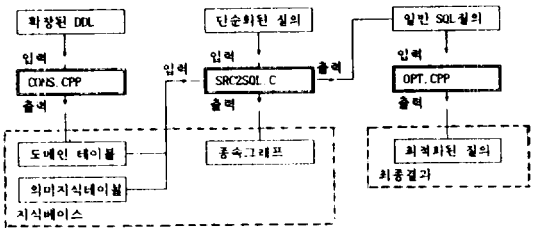
위의 질의에 대해 '부품의 CITY가 seoul이면 부품의 COLOR는 black이다'라는 의미적 지식이 존재하고 또한 부품 테이블이 COLOR로 인덱싱이 되어 있다면, 'P.COLOR='black' AND라는 부가적인 조건이 추가될 때, 인덱싱을 사용하여 이 조건에 부합하는 튜플을 우선적으로 선택한 다음 이들을 대상으로 처리할 수 있으므로 비용의 절감이 가능하여 진다.

다음은 연산의 추가를 위한 알고리즘이다.

```
int Optimize : : Append(int i)
{
  flag=NO;
  for (each semantic rule : R in semantic table) {
    if (C = R) {
      if (opr='↔' AND findindex(attribute of goal of R) {
        append a condtion with goal of R;
        flag=YES;
        return flag;
      }
    }
  }
}
```

6. 실험 및 결과

본 논문에서 제안한 기법은 IBM-PC 호환기종에서 C와 C++로 구현되었으며, 프로그램의 각 모듈은 (그림 6)과 같은 구성을 지닌다.



(그림 6) 프로그램의 구성도
(Fig. 6) Structure of program

프로그램은 CONS.CPP와 SRC2SQL.C와 OPT.CPP로 구성되며, CONS.CPP는 (그림 2)에서 제시된 바와 같은 확장된 DDL로 기술된 릴레이션 정의를 입력으로하여 도메인 테이블을 구성하며, SRC2SQL.C는 도메인 테이블과 단순화된 질의어를 입력으로 하여 생략된 부분이 첨가된 일반질의어를 출력한다. OPT.CPP는 변환된 SQL 질의어를 입력으로 하여 최적화된 질의어를 생성하게 된다.

본문 내용에서 예시한 각 질의를 구성한 프로그램을 통하여 처리한 결과는 다음과 같다.

□ CONS.CPP의 실행결과 생성된 도메인 테이블

테이블명	속 성 명	도메인명	기 본 키	외 래 키
S	SNAME	S_NAME	PRIMARY	X
S	STATUS	STATUS	X	X
S	CITY	CITY	X	X
SP	SNAME	S_NAME	PRIMARY	FOREIGN
SP	PNAME	P_NAME	PRIMARY	FOREIGN
SP	QTY	QTY	X	X
P	PNAME	P_NAME	PRIMARY	X
P	COLOR	COLOR	X	X
P	WEIGHT	WEIGHT	X	X
P	CITY	CITY	X	X

* 프로그램의 실행에서 도메인 테이블은 그림2에서 제시한 확장된 DDL기초로 하여 생성되었다.

□ 실행에 사용된 의미적 지식테이블

```
s.status > 50 → p.color = 'white';
P.WEIGHT > 5 ↔ P.COLOR='BLACK';
P.CITY='SEOUL' → P.COLOR='RED';
```

□ 실행 예 1 : 최적화 단계에서 불필요한 연산의 삭제

```
// src2sql의 실행 결과
>>> Source File
SELECT SNAME,STATUS
WHERE COLOR='black';
```
← 단순화된 질의

```
***     SELECT SNAME,STATUS
        attribute : SNAME → relation : S
        attribute : STATUS → relation : S

***     WHERE STATUS > 50 AND
        Implicit Query
        attribute : STATUS → relation : S

***     COLOR='WHITE';
        Implicit Query
        attribute : COLOR → relation : P
        Inferenced condition :
        AND

+       P.PNAME=SP.PNAME AND
+       SP.SNAME=S.SNAME

>>>     SQL Code
SELECT  S.SNAME, S.STATUS
FROMS,  P, SP                          ← 추가된 FROM절
WHERE   STATUS > 50 AND
+       P.COLOR='WHITE' AND            ← 추가된 조인연산
+       P.PNAME=SP.PNAME AND
        SP.SNAME=S.SNAME;

// opt의 실행결과
>>>     Delete Process
        delete conditions
        P.COLOR='WHITE'
        + P.PNAME=SP.PNAME
        + SP.SNAME=S.SNAME

>>>     Object Code
SELECT  S.SNAME, S.STATUS              ← 최적화된 질의어
FROM    S
WHERE   S.STATUS > 50;
```

처리결과 분석 : 단순화된 질의에서 생략된 정보인 FROM 절과, WHERE절에서 생략된 조인연산(P.PNAME=SP.PNAME AND SP.SNAME=S.SNAME) 그리고 각 속성의 소속 릴레이션명이 추가된 SQL이 생성되었으며, 이를 바탕으로 의미적 지식(S.STATUS > 50→P.COLOR='WHITE';)을 통하여 WHERE절에서 불필요한 조인연산(P.COLOR='WHITE' AND P.PNAME=SP.PNAME AND SP.SNAME=S.SNAME)이 삭제된 최적화된 질의어가 생성되었음.

□ 실행 예 2 : 효율적인 연산으로의 대체

```
// src2sql의 실행결과
>>>  Source File :
SELECT  PNAME                          ← 단순화된 질의
WHERE   WEIGHT > 5;

***     SELECT P.PNAME
        attribute : P.PNAME → relation : P
```

```
***     WHERE WEIGHT > 5;
        Implicit Query
        attribute : WEIGHT → relation : P

>>>     SQL Code
SELECT  P.PNAME
FROM    P                          ——— 추가된 FROM절
WHERE
        P.WEIGHT > 5;

// opt의 실행결과
>>>     Replace Process
        replace condition
        P.WEIGHT > 5 →
        P.COLOR = 'BLACK'

>>>     Object Code
SELECT  P.PNAME                    ——— 최적화된 질의어
FROM    P
WHERE
        P.COLOR = 'BLACK';
```

처리결과 분석 : 효율적인 연산으로의 대체를 검증하기 위한 예제이며, 최초 입력화일인 단순화된 질의어에서 생략된 릴레이션명과 FROM절이 SRC2SQL의 실행후에 추가되어 완성된 SQL 질의어가 생성되었으며, OPT의 실행후 의미적 지식(P.WEIGHT > 5 ↔ P.COLOR = 'BLACK';)을 바탕으로 WHERE절에서 'P.WEIGHT > 5'가 'P.COLOR = 'BLACK"로 대체되었음을 확인할 수 있다.

□ 실행 예 3 : 효율증진을 위한 연산의 추가

```
// src2sql의 실행결과
>>>     Source File :
SELECT  S.STATUS                   ——— 단순화된 질의
WHERE   P.CITY = 'SEOUL';

***     SELECT S.STATUS
        attribute : S.STATUS → relation : S

***     WHERE P.CITY = 'SEOUL';
        Implicit Query
        attribute : P.CITY → relation : P
        Inferenced condition : AND
-       P.PNAME = SP.PNAME AND
-       SP.SNAME = S.SNAME

>>>     SQL Code
SELECT  S.STATUS
FROM    S, P, SP                   ——— 추가된 FROM절
WHERE
        P.CITY = 'SEOUL' AND
-       P.PNAME = SP.PNAME AND
-       SP.SNAME = S.SNAME;

// opt의 실행결과
>>>     Append Process
        append condition
        P.COLOR = 'RED' AND

>>>     Object Code
SELECT  S.STATUS
FROM    S, P, SP
WHERE
        P.CITY = 'SEOUL' AND       ——— 최적화된 질의어
        P.COLOR = 'RED' AND
        P.PNAME = SP.PNAME AND
        SP.SNAME = S.SNAME;
```

처리결과 분석 : SRC2SQL의 실행후 생략된 FROM절과 WHERE에서의 조인연산(P.PNAME = SP.PNAME AND SP.SNAME = S.SNAME) 그리고 각 속성들의 릴레이션명이 부가된 SQL 질의어가 생성되었으며, OPT의 실행후 의미적 지식(P.CITY = 'SEOUL' → P.COLOR = 'RED';)을 사용하여 효율을 증진할 수 있는 조건절(P.COLOP = 'RED')이 추가 되었음을 확인할 수 있다(본문 5.3절 참조).

7. 결 론

본 논문에서는 질의 최적화를 위해 세 가지의 지식을 표현하고 다루는 방법을 제시하였다. 세 가지 지식 중 구조적 지식은 관계형 데이타베이스에서의 릴레이션간의 물리적 관계를 표현하며 이는 기본 키 와 외래 키에 의해 유지되는 관계성 정보를 내포한다. 의미적 지식은 도메인 무결성 법칙과 의미적 무결성 법칙을 포함하고 이는 내부 지식베이스에 유지되어 질의 최적화 단계에서 이용된다. 도메인 테이블은 구조적 지식과 의미적 지식을 구축하고 질의 처리를 위한 보조지식으로 사용된다. 또한 본 논문에서는 제안된 지식들을 사용하여 필수적인 성분이나 연산이 부분적으로 생략된 단순화된 질의를 완전한 질의로 변환할 수 있는 기법을 제시하여 사용자로 하여금 보다 단순한 질의를 사용할 수 있는 환경을 제공하였다.

본 논문에서 제안한 구조적 의미적 정보를 활용한 최적화 기법은 지적인 데이타베이스 시스템을 구축하는데 유용하게 응용될 수 있고 대단위 지식을 다루는 지식베이스 시스템이나 전문가 시

스템에서 정보검색시 효율적인 탐색경로를 추출하는데도 이용될 수 있다.

향후 연구 과제는 질의의 결론에 도달하는 두 개 이상의 경로가 있을 때, 이들 중 최적의 경로를 선택할 수 있는 최적화 기법에 관한 연구가 이루어져야 할 것이다.

참 고 문 헌

[1] P. J. Hayes and J. G. Carbonell, "Multi-Strategy Construction Specific Parsing for Flexible Data Base Query and Update," Proceedings of the Seventh International Joint Conference on Artificial Intelligence, pp. 432-439, 1981.

[2] L. Haas, J. C. Freytag, G. M. Lohman, and H. Pirahesh, "Extensible Query Processing in Starburst," Proceedings of ACM SIGMOD International Conference on the Management of Data Portland, Oregon, pp. 377-388, June 1989.

[3] A. Ohori, P. Buneman, and V. Breazu-Tannen,"Database Programming in Machiavelli-a Polymorphic Language with Static Type Inference," Proceedings of ACM SIGMOD International Conference on the Management of Data Portland, Oregon, pp. 46-57, June 1989.

[4] S. Chaudhuri,"Query Languages and Processing," Proceedings of Sixth International Conference on Data Engineering, pp. 138-145, 1990.

[5] J. Annevelink,"Database Programming Languages : A Functional Approach," Proceedings of ACM SIGMOD International Conference on the Management of Data Portland,Oregon, pp. 318-327, 1991.

[6] U. Chakravarthy, D. H. Fishman, and J. Minker, "Semantic Query Optimization in Expert Systems and Database Systems," in Expert Database system, L. Kerschberg, Ed. Culver city, CA : Benjamin/Cummings Rublishing Co., Inc., 1986.

[7] Wei Sun, Clement T. Yu, "Semantic Query Optimization for Tree and Chain Queries," IEEE Transactions on Knowledge and Data Engineering, Vol. 6, No. 1, Feb. 1994.

[8] U. Chakravarthy, J. Minker, and J. Grant, "Semantic Query Optimization : Additional Constraints and Control Strategies," in Expert Database System, L. Kerschberg, Ed. Culver City, CA : Benjamin/Cummings Publishing Co., Inc., 1987.

[9] Amihai Motro, "Constructing Queries from Tokens," Proceedings of ACM SIGMOD International Conference on the Management of Data Portland, Oregon, pp. 120-131, 1991.

[10] 성종진, 박종태, "객체지향 데이타베이스에서 의미지식을 이용한 효과적인 질의어 처리," 한국정보과학회 논문지, 제20권 2호, pp. 159-168, 1993년 2월.

[11] C. J. Date, An Introduction to Database systems, Vol. I Fourth Ed. Addison-wesley, Reading, Massachusetts, 1986.

남 인 길

1978년 경북대학교 전자공학과 전자계산전공(학사)
1981년 영남대학교 대학원 전자공학과 계산기 전공(공학석사)
1992년 경북대학교 대학원 전자공학과 전산공학 전공(공학박사)
1978년~80년 대구은행 전산부
1980년~90년 경북산업대학 전자계산학과 부교수
1990년~현재 대구대학교 공과대학 전자계산학과 교수
관심분야 : 데이타베이스

이 두 한

1987년 경북대학교 전자공학과 (학사)
1991년 경북대학교 대학원 전자공학과(공학석사)
1993년 경북대학교 대학원 컴퓨터공학과 박사과정 수료
관심분야 : 객체지향데이타베이스, 인공지능