관계형 데이터베이스에서 비정규화를 고려한 최적 데이터베이스 설계

장영관 · 강맹규*

An Optimal Database Design Considering Denormalization in Relational Database

Young Kwan Jang and Maing Kyu Kang

〈Abstract〉

Databases are critical to business information systems, and RDBMS is most widely used for the database system. Normalization has been designed to control various anomalies(insert, update, and delete anomalies). However, normalized database design does not account for the tradeoffs necessary for the performance.

In this research, we develop a model for database design by denormalization of duplicating attributes in order to reduce frequent join processes. In this model, we consider insert, update, delete, and query costs. The anomaly and data inconsistency are removed by additional disk I/O which is necessary for each update and insert transaction.

We propose a branch and bound method for this model, and show considerable cost reduction.

* 한양대학교 산업공학과

1. 서론

1.1 연구의 배경 및 목적

데이터는 여러 부문에서 중요한 전략적 자원이 되어 가고 있다. 이전에는 데이터를 주로 화일로 관리하여 왔으나 정보처리 요구가 증가함에 따라 DBMS 사용이 급격히 증가하고 있다. DBMS 중에서 현재 가장 많이 사용하고 있는 것은 관계형 DBMS(relational DBMS)이다.

크고 복잡한 시스템에서 관계형 DBMS의 성능(performance)은 주요한 논점이 되고 있다. 성능은 트랜잭션을 처리하는 데 소요되는 응답시간(response time)을 말하고 일반적으로 트랜잭션(transaction)을 처리하기 위해 필요한 디스크 액세스 횟수에 의해 결정된다[11, 16].

데이터베이스 설계는 일반적으로 요구의 수집 및 분석, 개념적 설계, 논리적 설계, 물리적 설계, 구현의 절차로 이루어진다[4, 16]. 논리적 설계 단계에서는 주로 E-R(entity- relationship)모델을 사용하며, 정규화(normali- zation)와 데이터의 무결성(integrity) 제약을 고려하여 모델링한다. 물리적 설계 단계에서는 논리적 설계 단계에서 설계된 릴레이션(relation)을 테이블(table)로 사상(mapping)하고 물리적 저장구조를 정하여 좋은 성능을 가지도록 한다.

물리적 저장구조를 정하는 것은 가급적 디스크 액세스(access) 횟수를 줄이는 데 그 목적이 있다. 그 방법에는 인덱싱(indexing), 클러스터링(clustering), 해싱(hashing), 비정규화(denormalization), 수직단편화(vertical fragmentaion), 수평단편화(horizontal fragmentation) 등이 있다.

물리적 설계 단계에서는 논리적 설계에서의 릴레이션을 그대로 테이블로 구현하는 것이 이상적이다. 그러나 이렇게 설계한 결과는 좋은 성능을 기대하기가 어렵다[2, 11, 16, 26]. 좋은 성능을 얻기 위해서는 많은 경우 비정규화가 필요하지만 아직까지 비정규화를 수리적으로 모델링하여 최적화하는 연구는 거의 없다.

기존 연구에서 제안한 비정규화 방법은 매우 특수한 상황에서 적용할 수 있는 지침을 제시하는 것에 그치고 있고[7, 26], 또한 비정규화에 따른 바람직하지 않은 성질인 이변(anomaly)의 제거가 보장되지 않는 방법도 있다.

본 연구에서는 관계형 데이터베이스의 논리적 설계 단계에서 완전하게 정규화한 다음 이 정규화된 테이블 설계를 다시 역으로 비정규화하는 방법을 제안한다. 본 연구의 비정규화는 참조무결성(referencial integrity) 제약을 이용하여 참조하는 테이블(referencing table)에 참조되는 테이블(referenced table)의 속성(attribute)을 복사(duplicate)하는 방법을 사용한다. 본 연구의 비정규화 방법은 이변을 명확하게 제거할 수 있으며, 또한 이변을 제거하기 위한 데이터베이스 구현도 매우 쉽다. 본 연구에서는 제안하는 비정규화 방법을 수리적으로 모델링하고 분지한계법을 사용하여 어느 테이블에 어느 테이블의 속성을 복사할 것인지를 최적으로 결정한다.

1.2 기존 연구

Corrigan과 Gurry[11]는 성능을 향상시키기 위해 데이터베이스 설계 단계에서부터 프로그램의 검증 단계까지 각각의 단계에서 고려하고 적용할 수 있는 매우 실제적인 방법을 제시하였다. 그

들은 주로 ORACLE이라는 특정 DBMS에서 적용할 수 있는 방법을 제시하였다. 그들은 데이터베이스 설계가 완전히 정규화한 설계로 오랜 동안 지속되는 것은 매우 이상적인 것이며 현실적이지 않다는 것과 비정규화가 필요하다는 것을 강조하였다.

논리적 데이터베이스 설계 단계에서 성능을 향상시킬 수 있는 방법에 대한 연구로는 Lee[20]의 연구가 있다.

Lee는 무조건 고차의 정규화 형태로 설계하는 것 대신에 비용과 수익을 고려하여 의사결정나무 방법으로 정규화형을 결정하는 방법을 제시하였다. Lee는 정규화하는 과정에서 비용과 수익을 비교하여 수익이 더 큰 경우에 정규화 과정을 계속하고, 그렇지 않은 경우에는 더 이상 정규화하지 않음으로써 정규화형을 결정하는 방법을 사용하였다. Lee는 비용 요소로서 저장비용(storage cost), 이변비용, 그리고 죠인비용(join cost)을 고려하였다. 어느 차수까지 정규화해야 최적이라는 기존 연구가 없는 상황에서 수익과 비용으로 모델링하여 정규화형을 결정한 Lee의 연구는 매우 좋은 연구라고 할 수 있다. 그러나 Lee의 연구는 비용 요소를 주어진 것으로 하였다.

논리적 데이터베이스 설계 단계와 물리적 데이터베이스 설계 단계에서 성능 향상을 위한 연구로는 Dewan[13], Cerpa[7] 등의 연구가 있다. Dewan은 논리적 데이터베이스 설계와 물리적 데이터베이스 설계를 수리모형화하여 열거법으로 해를 구하였다. 논리적 설계에서는 범용 릴레이션(universal relation)을 분해(decomposition)하는 것을 고려하고, 물리적 설계에서는 정렬과 인덱스를 고려하였다.

Cerpa는 1995년에 논리적 데이터베이스 설계 단계 이후, 물리적 데이터베이스 설계단계 이전에 논리적 데이터베이스 설계를 수정하여 성능을 향상시키는 지침을 제시하였다. Cerpa는 실제적인 데이터베이스 설계 경험에 근거한 15가지 유용한 지침을 제시하였다.

비정규화를 이용하여 데이터베이스의 성능을 향상시키기 위한 연구는 Rodgers[26], Janas[18] 등의 연구가 있으며 본 연구와 가장 관련이 많은 연구이다.

Rodgers는 1989년에 논리적 데이터베이스 설계를 그대로 물리적 데이터베이스 설계로 전환하여 적절한 성능을 얻기는 어려우므로 경우에 따라서 비정규화는 필요하다고 강조하였다.

Rodgers는 비정규화의 대상으로 두 개의 릴레이션이 1:1의 관계, 두 개의 릴레이션이 M:N의 관계, 참조 데이터, 매우 자세한 데이터를 가지는 릴레이션, 파생된 데이터 등을 제시하였다. Rodgers는 성능을 향상시키기 위해 비정규화가 필요하다고 최초로 역설하였고 그 방법으로 매우 유용한 지침을 제시하였다. 그러나 Rodgers의 연구는 비정규화에 의한 부작용인 이변을 처리하는 명확한 방안을 제시하지 않았고 수리적인 모델을 제시하지 못하였다. 본 연구는 비정규화에 따른 이변을 처리하는 명확한 방안을 제시하며 수리적으로 모형화하여 최적화하는 방법을 제시한다.

Janas는 1995년에 본 연구에서와 같이, 정규화된 테이블을 비정규화하는 방법으로서 속성을 중복시키는 방법을 제안하였는데, 먼저 정규화를 한 다음 비정규화를 하는 방법이다. 참조하는 테이블에 참조되는 테이블의 모든 속성을 중복시켜서 죠인조회(join query)의 효율을 향상시키는 것이다. 또, 참조되는 테이블의 모든 속성을 기존의 기본키(primary key)에 포함시키고 참조하는

테이블은 기존의 외부키(foreign key)에 중복시킨 속성을 추가하여 외부키로 함으로써 참조무결성 제약으로 갱신이변을 방지하였다.

Janas외 비정규화 방법은 다른 연구에 비해 정형화할 수 있는 좋은 방법이며, 특히 참조무결성 제약을 이용하여 이변을 방지한 것은 이론적으로 기여한 바가 크다고 할 수 있다. 그러나 Janas의 방법을 실제적으로 구현하는 데는 다음과 같은 문제점이 있다.

첫째, 기본키로 지정할 수 있는 속성의 개수가 제한되는 DBMS(예를 들면, ORACLE에서는 16개가 한계임)가 일반적이기 때문에 모든 속성을 중복시키는 경우 실제적으로 구현할 수 없는 경우가 있을 수 있다.

둘째, 참조되는 테이블의 모든 속성을 참조하는 테이블에 복사하기 때문에 디스크 사용량이 크게 증가한다.

셋째, 만약 참조되는 속성 중에서 죠인조회에 거의 쓰이지 않는 속성이 있으면 디스크 저장용량이 증가할 뿐만 아니라 전체적인 성능이 저하된다.

속성을 중복시키는 비정규화 방법은 최적화 기법을 사용하여 죠인조회에 빈번하게 쓰이는 속성만을 중복시키는 것이 실제적인 구현에서는 바람직하다고 할 수 있다. 본 연구에서는 참조하는 테이블에 참조되는 테이블의 속성을 복사하는 방법을 사용하지만 성능을 최적화시키는 속성만을 복사한다.

본 연구는 디스크 저장용량에 제약이 있는 경우에 먼저, 완전하게 정규화하고 다음 단계로, 정규화된 데이터베이스 설계를 비정규화하여 최적의 성능을 갖는 설계를 구하는 것이다. 본 연구에서는 최적의 성능을 보장하는 속성만을 복사하여 데이터베이스 설계를 하기 위해 수리적으로 모델링하고 분지한계법을 사용하여 최적해를 구한다.

2. 비정규화 방법

비정규화의 일반적인 절차는 가장 중요하고 빈도가 높은 트랜잭션을 대상으로 하여 액세스 경로를 분석한다. 이러한 액세스 경로를 향상시키기 위해 비정규화하는 방법 중 특정 방법을 선택하고 삽입, 갱신, 삭제, 조회, 저장 비용을 평가하고 이변에 의한 부작용을 고려한다[28]. 성능 향상을 위해 비정규화하여 테이블을 설계하는 기존의 방법은 다음과 같다 [7, 11, 18, 26].

(1) 두 개 테이블의 결합

두 개의 릴레이션이 1:1의 관계에 있을 때 두 개의 테이블을 한 개의 테이블로 만들 수 있다.

(2) 세 개 테이블의 결합

두 개의 릴레이션이 M:N의 관계에 있을 때 이를 테이블로 설계하면 세 개의 테이블이 된다. 그 중에서 한 개의 테이블을 제거하고, 제거한 테이블의 속성을 다른 테이블에 복사하여 두 개의 테이블로 비정규화할 수 있다.

(3) 속성의 복사

두 개의 릴레이션이 1:N의 관계에 있는 경우는 일반적인 경우로서 참조되는 "1" 의 관계에 있는 테이블의 속성을 "N" 의 관계에 있는 테이블에 복사하여 죠인을 줄일 수 있다[11].

(4) 부모 테이블에 자식 테이블 포함

매우 자세한 데이터를 가지는 릴레이션의 경

우 부모(parent) 테이블에 자식(child) 테이블을 포함시킨다. 자식 테이블이란 부모 테이블에 종속적인 테이블을 말한다.

(5) 파생된 데이터를 위한 속성의 추가

파생된(derived) 데이터를 한 개의 속성으로 추가하여 반복적인 계산을 피할 수 있다. 파생된 데이터는 어떤 속성의 합계나, 다른 통계 수치 등으로서 테이블의 속성값으로부터 계산될 수 있는 데이터를 말한다. 파생된 데이터는 이미 시스템 안에 필요한 정보가 포함되어 있으므로 데이터베이스 설계에서는 그러한 속성을 저장하지 않는다. 그러나 파생된 데이터를 위한 속성을 추가하여 반복적인 계산을 피함으로써 온라인 트랜잭션의 성능을 크게 향상시킬 수 있는 경우가 있다.

(6) 최근의 자식 데이터를 위한 속성의 추가

부모 테이블의 튜플이 자식 테이블의 여러 튜플과 관련이 되어 있고 자식 테이블의 튜플이 특정 순서를 가지고 있을 때 가장 최근의 자식 데이터를 부모 관계의 테이블에 복사하여 포함시킬 수 있다.

(7) 보고서용 테이블의 생성

관계형 데이터베이스의 이점 중의 하나는 여러 개의 테이블로부터 데이터를 발췌하여 최신의 보고서를 만들 수 있다는 것이다. 그러나 어떤 보고서는 그 결과를 얻는 데 많은 시간이 걸리고 시스템의 부담을 크게 주는 경우가 있다. 즉, 많은 테이블과 많은 데이터를 액세스하고 많은 계산을 해야 하는 보고서는 실시간(real time)으로 그 결과를 보기가 힘들다. 이러한 경우 보고서용으로 여러 개의 테이블로부터 데이터를 발췌하여 새로운 테이블을 만들 수 있다.

3. 제안하는 비정규화 방법

3.1 비정규화 방법

정규화는 데이터베이스 설계의 하나의 목표이며, 일반적으로 좋은 데이터베이스 설계란 중복된 데이터를 가지지 않는 설계를 말한다. 그러나 실제적으로 데이터베이스 설계가 고차의 정규형을 가지면서 좋은 성능을 얻는 것은 이상적이지만 실현하기 매우 힘들다[26, 30]. 또한 현실적으로 이론적인 완벽성보다는 좋은 성능이 더욱 더 중요하므로 정규화와 비정규화는 상호 절충이 필요하다[7].

비정규화는 액세스 횟수를 줄일 수 있지만 데이터가 중복되어 저장용량이 증가하고 데이터 무결성 제약을 유지하는 규칙이 복잡해지는 단점을 가지고 있다. 비정규화는 삽입, 갱신, 삭제이변이 발생하게 하고 이러한 이변은 반드시 제거해야 하기 때문에 이를 처리할 수 있는 명확한 방안이 제시되어야 한다.

2장에서 설명한 비정규화 방법은 특수한 상황에서만 적용할 수 있고 일반적으로 정형화하기는 매우 힘들다. 또한, 두 개의 테이블을 한 개의 테이블로 만들 경우에 삽입, 갱신, 삭제이변이 발생하여 부정확한 정보가 되거나 정보를 표현할 수 없는 경우가 발생한다. 본 연구에서의 비정규화는 먼저 완전하게 정규화를 한 다음 역으로 다시 비정규화하는 방법을 사용한다. 특히, 정규화는 실제적으로 시스템을 구축할 때 널리 쓰이는 제3정규형, 혹은 그 이상으로 정규화한다.

본 연구의 비정규화는 참조무결성 제약(referencial integrity constraint)을 이용한 비정

규화 방법으로서 참조하는 테이블에 참조되는 테이블의 속성을 복사하는 방법이다. 속성을 복사하면 참조하는 테이블에는 복사한 속성 개수만큼 속성의 개수가 증가하고 참조되는 테이블의 속성의 개수와 전체적인 테이블의 개수는 변화가 없게 된다. 속성을 복사해 두면 참조되는 테이블과 참조하는 테이블을 죠인조회하는 경우 정규화된 설계에서는 두 개의 테이블을 액세스해야 하지만 본 연구의 비정규화를 한 경우는 참조하는 테이블만을 액세스하면 되므로 전체적인 성능을 향상시킬 수 있다. 이 결과는 다음의 절충(trade-off) 관계가 있다.

- 죠인조회 비용 감소
- 디스크 사용량 증가
- 삽입, 갱신 비용 증가

본 연구에서는 디스크 저장용량이 제한되어 있는 경우에 삽입 비용, 갱신 비용, 삭제 비용, 조회 비용을 수리 모형에서 고려한다. 본 연구의 비정규화는 속성을 복사하는 방법의 비정규화이기 때문에 갱신이변이 발생하고 삽입, 삭제이변은 발생하지 않는다. 이러한 이변은 반드시 제거되어야 하기 때문에 이변을 처리하기 위한 비용을 추가되는 디스크 액세스 비용으로 모델링한다.

3.2 이변 처리 방안

기존의 연구에서 제안한 비정규화 방법은 삽입이변, 갱신이변, 삭제이변이 발생할 수 있다. 본 연구에서는 갱신이변이 발생할 수 있으므로 본 절에서는 이러한 이변을 제거하는 명확한 방안을 제시한다.

본 연구의 비정규화는 정형화된 방법이므로 이변을 제거하는 방법도 정형화되는 특성이 있다. 즉, 특정 사업규칙(business rule)이나 속성의 의미(semantic)가 변하는 경우도 본 연구의 이변 처리 방법은 같다. 또한, 데이터베이스의 규모가 확장되어 테이블에 속성이 추가되는 등의 변화에도 이변 처리 방법은 변화되지 않는다. 이변 처리는 본 연구에서 제시하는 방안에 따라 SQL 프로그램으로 구현할 수 있다.

(1) 삽입이변

본 연구의 비정규화 방법은 삽입이변은 발생하지 않지만 새로운 튜플을 삽입할 때 복사한 속성의 값의 일관성(consistency)를 유지시켜 주어야 한다. 그 방법은 속성을 복사한 참조되는 테이블을 기본키 인덱스를 이용해서 1회 액세스함으로써 데이터 일관성을 유지한다.

(2) 갱신이변

갱신이변은 참조되는 테이블의 속성이 갱신되는 경우와 참조하는 테이블의 외부키가 갱신되는 경우에 발생할 수 있다. 첫째 경우는 참조하는 테이블에서 복사된 속성을 모두 갱신함으로써 이변을 제거하고, 둘째 경우는 참조되는 테이블을 기본키 인덱스를 이용해서 1회 액세스함으로써 이변을 제거할 수 있다.

(3) 삭제이변

본 연구의 비정규화 방법은 속성을 복사하는 방법이므로 삭제이변이 발생하지 않는다.

3.3 비용 요소

본 연구의 비용 요소는 디스크의 액세스 횟수이다. 디스크 액세스 횟수를 비용 요소로 고려한 이유는 성능은 보통 각 트랜잭션을 처리하기 위해서

액세스해야 할 튜플의 개수에 의해 결정되고, 튜플을 액세스하는 데 필요한 액세스 횟수는 그 튜플의 저장구조에 의해 결정되기 때문이다[8, 10, 16].

본 연구에서의 수리 모형은 삽입, 이변에 기인한 데이터 일관성 유지 비용을 물리적인 액세스 방법에 따른 비용으로 포함시켜 수식화한다. 본 연구에서의 액세스 방법은 순차 액세스(segment scan access)와 인덱스 액세스(indexed scan access)의 두 가지를 고려한다. 순차 액세스에서의 액세스 횟수는 한 개의 튜플을 액세스해야 하는 경우나 여러 개의 튜플을 액세스 해야 하는 경우에 관계없이 테이블의 모든 튜플 즉, 테이블 전부를 액세스해야 하고 블록킹(blocking)에 의해 한 번의 액세스로 여러 블록을 액세스할 수 있기 때문에 T_i를 튜플의 크기, R_i 를 튜플의 개수, W를 (블록킹 계수×블록 크기) 라고 하면 $(T_i \times R_i)/W$ 로 표현할 수 있다. 인덱스 액세스에서의 액세스 횟수는 질의를 처리하기 위해 액세스가 필요한 튜플의 개수로 추정할 수 있다[8].

인덱스는 주로 B^*-tree 혹은 B^+-tree를 사용한다. 인덱스 액세스에서는 인덱스 블록을 인덱스 높이(height)만큼 액세스한 다음 실제 데이터가 위치한 데이터 블록을 한 번 액세스한다. 그러므로 H_{ti}를 트랜잭션 t에서 사용하는 인덱스 높이라고 하고 N_{ti}를 액세스해야 하는 튜플의 개수라고 하면 테이블 i에 대한 액세스 횟수는 $(1+H_{ti}) \times N_{ti}$로 표현할 수 있다.

4. 수리 모형

4.1 가정

본 연구에서의 가정은 다음과 같다. 이 가정은 실제적으로 중요하게 고려해야 하는 것을 가정으로 하는 것이 아니고, 매우 일반적인 사항을 가정으로 한다. 또, 특정 데이터베이스 시스템에 따라 달라지는 내용을 가정으로 함으로써 본 수리모형이 범용성을 갖도록 한다. 본 연구의 가정 중에서 (1), (5), (6)의 가정은 실제적으로 일반적인 데이터베이스 운용 상황을 고려한 것이다. 그러나 이 가정과 달리 예외적인 상황이 있을 수 있다. 그러한 경우는 장[2]의 연구를 참고할 수 있다.

(1) 기본키에 대한 인덱스가 존재한다.

기본키는 개체 무결성을 유지하는 제약으로서 관계형 데이터베이스에서는 인덱스로 구현하고 있다. 만약 기본키에 대한 제약을 하지 않으면 같은 기본키를 가지는 튜플이 존재할 수 있으므로 기본키에 대한 인덱스는 필수적이다. 본 연구는 기본키에 대한 인덱스가 없는 경우도 적용이 가능하지만 보다 현실적인 경우를 고려하기 위해 인덱스가 있는 것으로 한다.

(2) 인덱스 갱신 비용을 고려하지 않는다.

본 연구는 3장에서 논의한 바와 같이 참조하는 테이블에 참조되는 테이블의 속성을 복사하느냐, 하지 않느냐를 결정하는 문제이다. 또, 본 연구는 속성을 복사할 때 인덱스도 함께 복사하지는 않는다. 인덱스 갱신 비용은 본 연구에서 상수로서 고려 여부가 본 연구의 모델에 영향을 주지 않으므로 본 연구에서는 인덱스 갱신 비용을 고려하지 않는다.

(3) 정렬 비용은 고려하지 않는다.

조회를 실행하려면 정렬이 필요하다. 그러나 정렬을 예로 들면, 주기억에 여유가 있으면 주기

억에서 정렬을 수행할 수도 있고 디스크에서 정렬을 수행할 수도 있다. 그런데 주기억의 가용 용량은 시스템의 상황에 따라 가변적이고 또 데이터베이스 시스템의 여러 가지 환경 설정에 따라 변할 수 있다. 그러므로 가능한 특정 hardware나 DBMS와의 독립성을 유지하기 위해 정렬 비용은 고려하지 않는다.

(4) 블록에는 데이터만 저장되고, 정의된 길이보다 짧은 속성도 정의된 길이만큼 고정길이로 저장된다.

블록에는 실제적으로는 데이터만 저장되는 것은 아니다[6]. 특정 DBMS에 따라 데이터 이외의 공간에는 가변적인 공간이 있으며 또한 오버플로우(overflow)를 방지하기 위한 여유 공간이 설정될 수 있다. 또, 테이블의 속성은 가변길이 데이터형(data type)일 수 있다. 따라서, 본 연구는 특정 DBMS와의 독립성을 유지하고 범용적인 모델을 만들기 위해 모든 블록에는 순수한 데이터만 저장되어 있는 것으로 가정한다.

(5) 모든 테이블의 튜플 크기는 1 블록 보다 작다.

중대형 데이터베이스 시스템에서 블록 크기는 일반적으로 2048 byte, 4096 byte, 혹은 그 이상인 경우도 있다. 그러므로 본 연구에서 속성을 복사하여 튜플의 크기가 다소 커진다고 해도 특별한 경우가 아니면 대부분의 튜플의 크기는 1 블록 이내라고 할 수 있다.

(6) 모든 갱신, 삭제 트랜잭션은 기본키 인덱스를 액세스 경로로 하여 이루어지고 외부키에 의해 참조되는 테이블에 대한 삭제 트랜잭션은 없다.

일반적으로 온라인으로 수행되는 갱신, 삭제 트랜잭션은 기본키 인덱스를 액세스 경로로 이루어진다. 또한, 기본키 인덱스를 액세스 경로로 하지 않는 갱신, 삭제 트랜잭션은 일반적으로 일괄작업(batch job)으로서 예를 들어 예비(backup) 작업, 이전(migration), 새로운 DBMS 설치 등과 같은 경우에 수행되는 것이 보통이다. 그런데 이러한 일괄작업은 데이터베이스 시스템에 부담이 없는 시간에 이루어지는 것이 일반적이므로 온라인 트랜잭션에 비하여 상대적으로 성능이 문제가 되지 않는다고 할 수 있다. 따라서 본 연구에서는 온라인 트랜잭션을 대상으로 한다.

4.2 기호 정의

본 연구에서 사용하는 기호는 다음과 같다.

i, j, m : 테이블

k : 속성

R_i : 테이블 i의 튜플 개수

L_i : 정규화된 테이블 i의 튜플 크기

L_{ik}: 테이블 i의 속성 k의 크기

B : 블록 크기

W : 블록킹 계수×블록 크기

H_i : 테이블 i에서 기본키 인덱스의 높이

H_{ij} :테이블 j를 참조하는 테이블 i의 외부키에 대한 인덱스 높이

t : 삽입, 갱신, 삭제, 조회 트랜잭션

F_t : 트랜잭션 t의 발생 빈도

H_{ti} : 트랜잭션 t에서 사용하는 테이블 i의 인덱스 높이

V_{ij} : 테이블 i의 한 개의 튜플에 대응하는 테이블 j의 평균 튜플 개수

I_t : 삽입 트랜잭션 t의 대상이 되는 테이블

U_t : 갱신 트랜잭션 t의 대상이 되는 테이블

D_t : 삭제 트랜잭션 t의 대상이 되는 테이블

P_t : 조회 트랜잭션 t에서 인덱스 액세스하는 테이블 집합

Q_t : 조회 트랜잭션 t에서 순차 액세스하는 테이블 집합

K_{ti} : 트랜잭션 t의 대상이 되는 테이블 i의 속성 집합

N_{ti} : 조회 트랜잭션 t에서 액세스하는 테이블 i의 튜플 개수

S : 가용한 여유 디스크 저장용량

Ci : 삽입 트랜잭션을 처리하기 위해 필요한 디스크 액세스 횟수

Cu : 갱신 트랜잭션을 처리하기 위해 필요한 디스크 액세스 횟수

Cd : 삭제 트랜잭션을 처리하기 위해 필요한 디스크 액세스 횟수

Cq : 조회 트랜잭션을 처리하기 위해 필요한 디스크 액세스 횟수

I_{tij}
- 1:삽입트랜잭션 t에서 테이블 j를 참조하는 i의 외부키가 널이 아닐 때
- 0:그렇지 않은 경우

U_{tij}
- 1:갱신트랜잭션 t에서 테이블 j를 참조하는 테이블 i의 외부키를 널이 아닌 값으로 갱신할 때
- 0:그렇지 않은 경우

x_{ijk}
- 1:테이블 i에 테이블 j의 속성k가 복사되어 있을때
- 0:그렇지 않은 경우

y_{ij}, g_{ij}
- 1:테이블 i에 테이블 j의 속성이 적어도 한개 복사되어 있을때
- 0:그렇지 않은 경우

z_{tij}
- 1:조회트랜잭션 t에서 액세스하는 테이블 i의 모든 속성이 죠인되는 테이블 j에 복사되어 있을때
- 0:그렇지 않은 경우

4.3 수리 모형의 정식화

본 수리 모형은 가용한 여유 디스크 저장용량이 S로 제한되어 있는 경우에 삽입 비용(Ci), 갱신 비용(Cu), 삭제 비용(Cd), 조회 비용(Cq)의 합을 최소화하는 문제로서 다음과 같이 표현된다. 여기서 모든 비용은 각각의 삽입, 갱신, 삭제 및 조회 트랜잭션을 처리하기 위해서 필요한 보조기억장치의 디스크 액세스 횟수를 나타낸다.

$$Min\ (Ci + Cu + Cd + Cq)$$

s.t.

$$\sum_i \sum_{j \neq i} \sum_k L_{ijk} \cdot x_{ijk} \cdot R_i \leq S$$

$$\sum_k x_{ijk} - M \cdot y_{ij} \leq 0$$

$$\sum_{k \in K_s} x_{jik} - M \cdot g_{ji} \leq 0$$

$$z_{tji} = \prod_{i \in K_s} x_{jik}$$

$$z_{tji} + z_{tmj} \leq 1$$

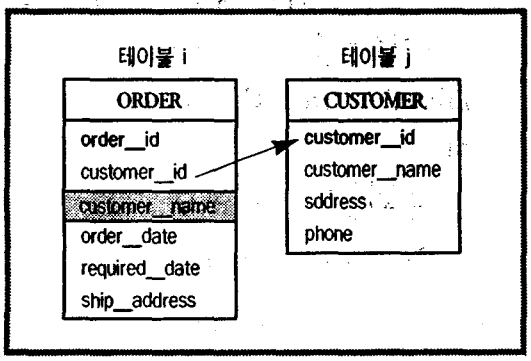
본 수리 모형에서의 x_{ijk} 는 결정변수로서 테이블 i에 테이블 j의 속성 k를 복사하는 것을 의미한다. 만약, $x_{ijk} = 1$ 이라고 결정되면 테이블 i에 새로운 속성이 한 개 추가된다. x_{ijk} 가 결정변수가 될 수 있는 조건은 다음의 세 가지가 동시에 만족할 경우이다.

- 테이블 i에는 테이블 j의 기본키를 참조하는 외부키 속성이 있어야 한다 (i⇒j로 표시함).

- 테이블 i와 테이블 j를 조인하여 조회하는 트랜잭션이 적어도 한 개 있어야 한다.
- 그 조회 트랜잭션에서 테이블 j의 속성 k에 대한 조회 요구가 있어야 한다.

〈그림 4.1〉은 비정규화된 테이블 설계의 예로서 ORDER 테이블의 customer_id는 CUSTOMER 테이블의 customer_id를 참조하는 외부키이고, ORDER 테이블의 customer_name은 CUSTOMER 테이블의 customer_name을 복사한 것이다.

〈그림 4.1〉 비정규화된 테이블 설계의 예



4.3.1 삽입 비용

삽입 비용은 테이블 i에 새로운 튜플을 삽입할 때 필요한 액세스 횟수와 3.2절에서 논의한 바와 같이 복사된 속성의 값을 유지시키기 위한 테이블 j에 대한 액세스 횟수로서 식(4.1)과 같이 표현된다.

$$Ci = \sum_{i} F_i \sum_{j \in I_i} [1 + \sum_{j=1} y_{ij} \cdot I_{ij}(1 + H_j)] \quad (4.1)$$

여기서 F_i는 테이블 j에 튜플을 삽입하는 트랜잭션의 발생 횟수이다. 속성을 복사하지 않은 정규화된 테이블에서의 삽입 트랜잭션에 대한 액세스 횟수는 1회이다. 그런데 속성을 복사하면 그 속성 값을 유지시켜 주어야 한다. 〈그림 4.1〉에서 보면 테이블 i에 한 개의 튜플이 삽입될 때 테이블 j에서 한 개의 튜플을 조회하여 customer_name의 값을 같이 삽입하여 주어야 한다. 그런데 본 연구는 참조하는 테이블에 참조되는 테이블의 속성을 복사하는 것이고 참조하는 테이블의 외부키는 참조되는 테이블의 기본키를 참조하고 있다. 본 연구의 가정에서 모든 기본키는 인덱스가 존재한다. 참조되는 테이블에서 한 개의 튜플을 조회하는 것은 일반적으로 인덱스 액세스가 순차 액세스보다 액세스 횟수가 적기 때문에 인덱스를 이용하여 액세스한다.

테이블 j에 대한 액세스는 기본키에 대한 인덱스 높이(H_j)만큼 인덱스를 액세스하고 또, 데이터 블록을 한 번 액세스한다. 따라서 테이블 j에 대한 액세스 횟수는 (H_j+ 1)회가 된다.

여기서 y_{ij}는 결정변수로서 테이블 i에 테이블 j의 속성이 하나라도 복사하면 1 이 되는 변수이다. y_{ij}는 속성을 복사하지 않은 경우는 0 이 되어 테이블 i에 튜플을 삽입할 때 테이블 j를 액세스하지 않아도 되도록 한다. 또, 〈그림 4.1〉에서 customer_name과 address 두 개의 속성을 복사한 경우를 보면 이 경우도 역시 테이블 j를 1회 액세스하면 되므로 한 개의 속성을 복사한 경우와 동일하다. 왜냐하면 디스크 액세스는 블록 단위로 액세스하고 1 블록에는 본 연구의 가정에 의해 여러 개의 튜플이 저장되어 있기 때문이다.

4.3.2 갱신 비용

갱신 비용은 복사한 테이블의 외부키가 인덱

스되어 있을 경우에 식(4.2)와 같이 표현된다. 갱신은 먼저 기본키에 대한 인덱스의 높이만큼 액세스하고, 데이터 블록을 1회 조회하고 1회 갱신하는 액세스를 해야 한다. 또한 복사해 둔 속성이 있는 경우에 데이터 일관성을 유지하기 위한 액세스 비용이 다음과 같이 추가된다. 예를 들어, 〈그림 4.1〉에서 ORDER의 customer_id가 갱신되는 경우에 데이터 일관성을 유지하기 위해 CUSTOMER 테이블에서 한 개의 튜플을 조회해야 한다. 또한 CUSTOMER 테이블의 customer_name이 갱신되는 경우에 ORDER에 복사한 customer_name을 갱신해야 한다.

$$Cu = \sum_{i} F_i \sum_{R \in U_i} [(2+H_i) + \sum_{j \neq i} y_{ij} \cdot U_{kj} (1+H_j) + \sum_{j \neq i} V_{ij} \cdot g_{ji} (2+H_{ji})] \quad (4.2)$$

식(4.2)에서 첫째 항은 속성을 복사하지 않았을 때의 갱신 트랜잭션에 대한 액세스 횟수이다. 본 연구의 가정에서 갱신 트랜잭션은 기본키에 대한 인덱스를 액세스 경로로 한다고 하였다. 갱신 트랜잭션은 갱신할 튜플을 두 번 액세스하여야 한다. 한 번은 갱신할 대상이 되는 튜플을 조회하는 액세스하는 것이고 또 다른 한 번은 갱신된 튜플을 디스크에 저장하는 데 필요한 액세스이다. 먼저 갱신할 대상이 되는 것을 조회하는 액세스 횟수는 기본키에 대한 인덱스를 이용해서 액세스하므로 (H_i + 1)이 된다. 갱신된 튜플을 다시 디스크에 저장하는 데 필요한 액세스 횟수는 1회이다. 그 이유는 갱신할 튜플을 조회하면 그 튜플이 저장되어 있는 디스크의 물리적인 주소(row-id)를 알 수 있기 때문에 갱신한 튜플을 저장하는 것은 인덱스를 다시 액세스하지 않고 실제 튜플이 저장되어 있는 블록 을 주소를 이용해 1회 액세스하면 되기 때문이다.

식(4.2)에서 둘째 항은 외부키가 갱신될 경우의 액세스 횟수를 나타낸다. 속성이 복사되어 있고(즉, y_{ij} = 1) 갱신되는 속성중에 기본키가 있으면 (즉, U_{ij} = 1) 참조되는 테이블에서 한 개의 튜플을 액세스해서 갱신한 외부키에 부합되는 속성값으로 데이터 무결성을 유지시켜 주어야 한다. 〈그림 4.1〉의 ORDER 테이블에 대한 갱신 트랜잭션에서 외부키인 customer_id를 갱신하는 경우가 이에 해당된다. customer_id가 갱신되면 참조되는 테이블을 액세스하여 customer_name을 조회하여야 한다. 그런데 참조되는 테이블에는 기본키에 대한 인덱스가 존재하므로 한 개의 튜플을 조회하려면 (H_j+1)회의 액세스가 필요하다.

셋째 항은 참조되는 테이블의 속성이 갱신되었을 때의 참조하는 테이블에 대한 액세스 횟수를 나타낸다. 〈그림 4.1〉의 CUSTOMER 테이블에서 어떤 튜플의 customer_name이 갱신되었을 경우가 이에 해당된다. 만약 갱신할 속성을 참조하는 테이블에 복사해 두었으면 참조되는 테이블의 속성을 갱신할 때 동시에 참조하는 테이블의 속성 값도 갱신하여야 한다. 참조하는 테이블에서 갱신하여야 할 튜플의 개수는 V_{ij} 이며 한 개의 튜플을 갱신하는 액세스 횟수는 위에서 설명한 바와 같이 (2 + H_j^i) 로 표현된다. 여기서 g_{ji}는 결정변수로서 만약 참조하는 테이블 i에 복사한 속성을 참조되는 테이블 j에서 갱신하면 1이 되는 변수이다.

참조되는 테이블의 속성을 복사한 참조하는 테이블의 외부키가 인덱스되어 있지 않을 경우는

순차 액세스를 하여야 하므로 식(4.3)과 같이 표현된다.

$$Cu' = \sum_t F_t \sum_{i \in D_t} [\,(2 + H_i) + \sum_{j \neq i} y_{ij} \cdot U_{ij}\,(1 + H_j) + \sum_{j \neq i} g_{ji} \cdot \left(\frac{T_j \cdot R_j}{W} + V_{ij}\right)] \quad (4.3)$$

여기서, $T_j = L_j + \sum_{m \neq j} \sum_k L_{mk} x_{jmk}$ 로서 테이블 j의 튜플 크기를 나타낸다. 즉, 비정규화하지 않았을 때의 튜플 크기(L_j)에 복사한 속성들의 크기를 더하면 비정규화된 테이블에서의 튜플 크기가 된다.

4.3.3 삭제 비용

삭제 비용은 식(4.4)와 같이 표현되며, 본 연구에서는 속성을 복사해 두는 비정규화 방법을 사용하기 때문에 삭제 이변이 발생하지 않는다. 본 연구의 가정에서 삭제 트랜잭션은 기본키에 대한 인덱스를 액세스 경로로 하므로 인덱스에 의한 한 번의 조회와 주소에 의한 한 번의 갱신으로 액세스 횟수가 표현된다.

$$Cd = \sum_t F_t \sum_{i \in D_t} (2 + H_i) \quad (4.4)$$

4.3.4 조회 비용

조회 비용은 순차 액세스와 인덱스 액세스 경우로 식(4.5)와 같이 표현된다

$$Cq = \sum_t F_t \,[\, \sum_{i \in Q_t} (1 - z_{ti}) \, \frac{T_i \cdot R_i}{W} + \sum_{i \in P_t} (1 - z_{ti}) \cdot (1 + H_{ti}) N_{ti}] \quad (4.5)$$

첫째 항은 순차 액세스 경우 테이블 i에 대한 액세스 횟수를 나타낸다. 순차 액세스는 조회해야 할 튜플의 개수에 관계없이 테이블 전체를 읽어야 하고 여러 블록을 한 번의 액세스로 읽어 올 수 있기 때문에 테이블 크기($T_i \times R_i$)에서 W(블록킹 계수×블록 크기)로 나눈 값으로 액세스 횟수가 결정된다. 또, 인덱스 액세스에서는 한 개의 튜플을 읽으려면 인덱스 블록을 먼저 높이(H_i)만큼 액세스한 다음 데이터 블록을 한 번 액세스해야 하므로 $(1 + H_i)$로 표현된다. 따라서, 만약 N_{ti}개의 튜플을 읽으려면 인덱스 액세스를 N_{ti} 번 액세스하여야 하므로 $(1 + H_i) \times N_{ti}$ 로 표현된다.

여기서 z_{ti}는 결정 변수로서 죠인조회 t에서 필요한 테이블 i의 속성이 모두 테이블 j에 복사되어 있으면 1이 되어 테이블 i를 액세스하지 않아도 죠인조회의 결과를 얻을 수 있도록 하는 변수이다.

4.3.5 제약 조건

제약 조건은 디스크 저장용량에 대한 제약과 결정변수에 대한 제약으로서 식(4.6)~(4.10)과 같이 표현된다.

식(4.6)은 디스크 저장용량에 관한 제약이다. 속성을 복사함으로써 증가되는 저장용량은 (속성의 크기(L_{ik}) × 테이블의 튜플 개수(R_i))로 표현되고 이것이 가용한 여유 저장용량 S보다 적거나 같도록 제약한다. 여기서 S를 매우 크게 하면 저장용량에 제약이 없는 경우도 적용이 가능하다.

$$\sum_i \sum_{j \neq i} \sum_k L_{ik} \cdot x_{ijk} \cdot R_i \leq S \quad (4.6)$$

식(4.7) 및 (4.8)은 결정변수 y_{ij} 및 g_{ij}에 대한 제약조건이다. 여기서 M은 상수로서 복사할 수 있는 속성의 개수보다 큰 수를 나타내며, 만약, 각각의 조건에 맞는 k에 대하여 x_{ijk}가 하나라도 1이면 y_{ij} 및 g_{ij}는 1이 되도록 제약한다.

$$\sum_{k} x_{ijk} - M \cdot y_{ij} \le 0 \qquad \forall i,j \qquad (4.7)$$

$$\sum_{k \in K_i} x_{ijk} - M \cdot g_{ij} \le 0 \qquad \forall i,j \qquad (4.8)$$

식(4.9)는 j⇒i 의 관계에 있는 테이블의 죠인 조회 트랜잭션을 처리하기 위한 제약으로서 만약, 죠인조회에서 필요한 테이블 i의 모든 속성이 j에 복사되어 있으면 z_{tji}는 1로 제약되어 그 조회를 처리하기 위한 테이블 i의 액세스 횟수는 0이 될 수 있도록 한다.

$$z_{tji} = \begin{cases} \prod_{k \in K_i} x_{ijk} & i,j \in Q_t \cup P_t,\ j \Rightarrow i, \\ & t\text{는 죠인조회 트랜잭션} \\ 0 & i,j \in Q_t \cup P_t,\ j \Rightarrow i, \\ & t\text{는단일테이블 조회 트랜잭션} \end{cases} \qquad (4.9)$$

식(4.10)은 세 개 이상의 테이블을 죠인하는 조회 트랜잭션에 대한 제약이다. 만약, 조회 트랜잭션에서 필요한 테이블 i의 속성들이 테이블 j에 복사되어 있고 또, 테이블 j의 속성들이 테이블 m에 복사되어 있다고 가정하면, 이 트랜잭션을 처리하기 위해서는 테이블 i와 테이블 j 중에서 적어도 한 개는 액세스해야만 하고 또, 테이블 j와 테이블 m 중에 적어도 한 개는 액세스해야만 한다.

$$z_{tji} + z_{tmj} \le 1 \qquad i,j \in Q_t \cup P_t,\ m \Rightarrow j,\ j \Rightarrow i,\ t\text{는죠인조회 트랜잭션} \qquad (4.10)$$

본 연구의 수리모형은 모든 결정변수가 0, 1 정수값을 갖는 비선형 정수계획법이다. 비선형 정수계획법에 대한 효율적인 알고리듬이 없으므로 본 연구에서는 분지한계법을 사용하여 최적해를 구한다.

5. 분지한계법을 이용한 알고리듬

5.1 분지한계법의 기본 개념

분지한계법(branch and bound method)은 기본적으로 열거법(enumeration)의 일종으로서 그 중에서 부분열거법에 속한다[1, 19, 24, 25]. 분지한계법은 각 단계에서 최적해가 포함되어 있을 가능성이 가장 커 보이는 부분집합을 정하여, 이 부분집합에 계산량을 줄일 수 있는 어떤 전략을 사용하여 두 개 이상의 새로운 부분집합으로 분할한다. 이를 분지(branch)라 한다.

각 분지된 문제에서 그 부분집합에서의 해의 한계를 계산한다. 만약 해의 한계를 구하는 중에 최적해를 구했으면 알고리듬을 끝낸다. 그렇지 않으면 후보문제들의 한계를 사용하여 새로 분지할 부분집합을 선정하고 위의 과정을 반복한다.

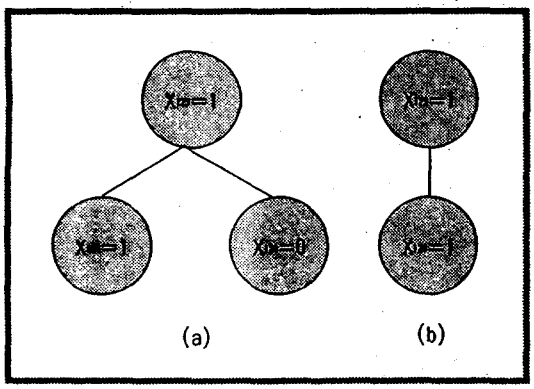
5.2 부분 분지

본 연구에서의 분지는 기본적으로 복사할 대상이 되는 속성을 복사한다(x_{ijk} = 1)와 복사하

지 않는다(x_{ijk} = 0)로 하면 된다. 본 연구에서는 x_{ijk} = 1, x_{ijk} = 0로 분지하지 않고 x_{ijk} = 1, x_{ijk} = 0 중에서 한 개로만 분지할 수 있는 경우가 있다.

본 연구에서는 부분 분지라고 한다. 〈그림 5.1〉은 죠인조회 트랜잭션 t*에서 테이블 1과 테이블 2를 죠인하는 경우 테이블 1에 테이블 2의 속성 3과 4를 복사하는 것을 분지하는 예이다. 본 연구의 수리모형은 속성을 복사하여 죠인조회 트랜잭션에서 액세스하는 테이블의 수를 줄여서 총 비용을 최소화하는 것이므로, 그 수를 줄이려면 죠인조회 트랜잭션 t*에서 필요한 모든 속성을 복사하여야 한다. 그런데 t≠t*인 죠인조회 트랜잭션 t가 테이블 1과 테이블 2의 속성 4를 죠인하지 않는다면 후보노드와 다른 값을 갖는 분지는 같은 값을 갖는 분지보다 항상 비용이 크게 된다. 왜냐하면, 만약 속성을 일부만 복사하면 삽입, 갱신 비용은 속성을 복사하지 않는 경우보다 증가하게 되고, 죠인조회 비용은 감소하지 않기 때문이다.

〈그림 5.1〉 부분 분지



본 연구의 수리모형은 다음의 정리 1이 성립한다.

[정리 1]

t≠t* 인 모든 죠인조회 트랜잭션에 대하여 $\forall K_{ij} \cap K_{t^*j^*} = \phi$ 이면 $K^1, K^2 \in K_{ij}$ 인 $X_{i^*j^*k}^1$과 $X_{i^*j^*k}^2$가 같지 않으면 최적해가 아니다.

[증명]

$X_{i^*j^*k}^1 \neq X_{i^*j^*k}^2$ 일 때의 최적해 X의 목적함수 값을 각각 Ci(삽입 비용), Cu(갱신 비용), Cd(삭제 비용), Cq(조회 비용)라고 정의하고 $X_{i^*j^*k}^1 = X_{i^*j^*k}^2 = 0$ 일 때의 최적해 X의 목적함수 값을 각각 Ci^0, Cu^0, Cd^0, Cq^0 라고 정의한다.

$X_{i^*j^*k}^1 \neq X_{i^*j^*k}^2$ 인 경우에 두 개의 변수 중에서 한 개는 1이므로 다음 제약에 의하여 $Y_{i^*j^*}=1$ 이다.

$$\sum_k x_{ijk} - M \cdot y_{ij} \leq 0$$

$Y_{i^*j^*}=1$이므로 삽입 트랜잭션에 대한 비용은 다음과 같이 표현된다.

$$\begin{aligned} Ci &= \sum_t F_t \sum_{i \in I_t} [1 + \sum_{j \neq i} y_{ij} \cdot I_{tij}(1 + H_j)] \\ &= \sum_t F_t \sum_{i \in I_t, i = i^*} [1 + \sum_{j \neq i^*} I_{ti^*j^*}(1 + H_{j^*})] \\ &\quad + \sum_t F_t \sum_{i \in I_t, i \neq i^*} [1 + \sum_{j \neq i^*} y_{ij} \cdot I_{tij}(1 + H_j)] \\ &\geq Ci^0 \end{aligned}$$

같은 방법으로, 갱신 트랜잭션에 대한 비용은 $Cu \geq Cu^0$ 이다.

$X_{i^*j^*k}^1 \neq X_{i^*j^*k}^2$ 인 경우에 두 개의 변수 중에서 한 개는 0이므로 $z_{ij} = \prod_{k \in K_{ij}} x_{ijk}$ 의 제약에 의하여 $Z_{t^*i^*j^*} = 0$ 이다. 또 t≠t* 인 모든 죠인

조회 트랜잭션에 대하여 $K_{ti} \cap K_{t'j'} = \phi$ 이므로 z_{ti} 와 $z_{t'j'}$ 는 서로 독립이다. 그러므로 조회 트랜잭션에 대한 비용은 다음과 같다.

$$
\begin{aligned}
Cq &= \sum_t F_t \Big[\sum_{j \in Q_t} (1 - z_{tij}) \frac{T_j \cdot R_j}{W} \\
&\quad + \sum_{j \in P_t} (1 - z_{tij}) \cdot (1 + H_{ti}) N_{ti} \Big] \\
&= F_{t*} \Big[\sum_{j \in Q_{t*}} \frac{T_{j*} \cdot R_{j*}}{W} \\
&\quad + \sum_{j \in P_{t*}} (1 + H_{t*j*}) N_{t*j*} \Big] \\
&\quad + \sum_{t \neq t*} F_t \Big[\sum_{j \in Q_t, j \neq j*} (1 - z_{tij}) \frac{T_j \cdot R_j}{W} \\
&\quad + \sum_{j \in P_t, j \neq j*} (1 - z_{tij}) \cdot (1 + H_{ti}) N_{ti} \Big] \\
&\geq Cq^0
\end{aligned}
$$

삭제 트랜잭션 Cd는 본 연구에서는 상수로서 Cd^0 와 같다. $Ci \geq Ci^0$, $Cu \geq Cu^0$, $Cq \geq Cq^0$, $Cd = Cd^0$ 이므로 이것은 최적해가 아니다. 그러므로 본 정리 1이 성립한다.

5.3 분지 전략

분지한계법의 효율에 큰 영향을 주는 본 연구에서 제안하는 분지 전략은 부분 분지 기준에 부합되는 노드, 동일한 K_{ti}에 속하는 노드, 죠인조회의 액세스 횟수가 큰 노드 순으로 선택하는 것으로 한다.

(1) 부분 분지 기준에 부합되는 노드를 선택

5.2절의 부분 분지 정리에 따라 부분 분지 기준에 부합되는 노드를 선택한다. 그 이유는 부분 분지 기준에 부합되면 탐색해야 할 노드의 수가 분지노드를 기준으로 하여 1/2 로 줄어들기 때문이다. 〈그림 5.2〉에서 보는 바와 같이 부분 분지 기준에 부합되는 노드의 선택 순서에 따라 탐색해야 할 노드의 개수가 크게 변하는 것을 알 수 있다. 결정변수(탐색해야 할 노드의 개수)의 수가 n이라고 하면

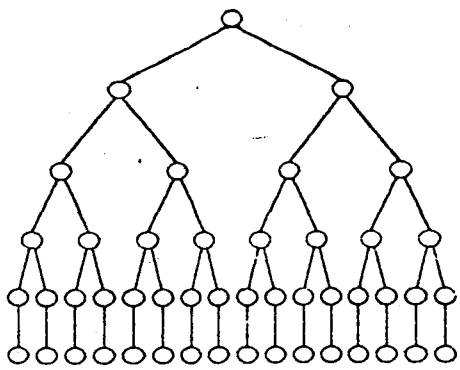
최악의 탐색노드 개수는 $2^{n-1} + \sum_{n=1}^{n-1} 2^n$

이며 최선의 탐색노드 개수는 $2 + \sum_{n=1}^{n-1} 2^n$ 이다. 최악의 탐색노드와 최선의 탐색노드 개수의 차이는 $2^{n-1}-2$ 로서 매우 크다.

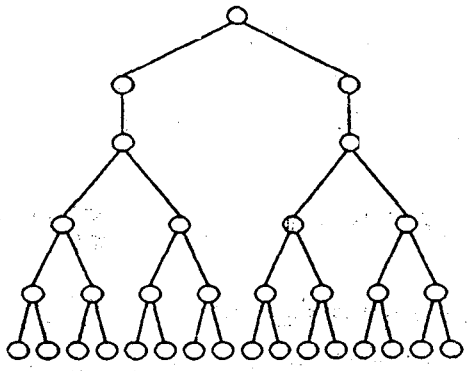
(2) 동일한 K_{ti} 에 속하는 노드를 선택

본 연구에서는 죠인조회를 처리하기 위해 액세스해야 하는 테이블의 개수를 줄이는 연구이다. 그러므로 액세스해야 하는 테이블의 개수를 줄이려면 조회트랜잭션 t에서 필요한 테이블(i)의 모든 속성 K_{ti}를 복사해야만 한다. 따라서 동일한 K_{ti}에 속하는 노드를 먼저 분지하는 것이 분지 개수를 줄일 수 있다.

〈그림 5.2〉 부분 분지 순서에 따른 분지 개수



(a) 마지막 단계에서의 부분 분지



(b) 첫번째 단계에서의 부분 분지

(3) 죠인조회 비용이 가장 큰 노드를 선택

본 연구는 속성을 복사하여 죠인조회 비용을 줄이는 연구이다. 따라서 분지는 죠인조회 비용이 가장 큰 노드를 선택한다. 즉, 죠인조회에서 참조하는 테이블과 참조되는 테이블 중에서 참조되는 테이블에 대한 액세스 비용이 가장 큰 것을 선택하여 분지노드로 한다. 그 이유는 참조되는 테이블은 속성을 복사하면 액세스하지 않아도 되기 때문이다. 죠인조회 비용이 가장 큰 노드를 선택함으로써 좋은 최저한계가 생성되게 함으로써 가능한한 분지를 억제할 수 있다.

5.4 한계 전략

최저한계란 분지 과정에서의 하나의 목적식 값을 말하는 것으로 그 목적식 값보다 더 좋은 해가 나올 수 없는 것이 보장되어야 한다. 또한, 최저한계를 구하는 방법은 간단하고 계산량이 적어야 한다. 본 연구에서는 제약식을 완화하여 완화된 문제(RP)를 만들고 그 완화된 문제의 최적해를 구하여 최저한계로 사용한다.

본 연구에서는 제약식을 완화하는 방법으로서는 제약식의 개수를 줄이는 방법을 사용하고 다음과 같이 두 개의 완화된 문제(RP1, RP2)를 정의하고 각각의 문제를 최적화하여 그 합을 최저한계로 한다.

$$RP1 = Min\ (\ Ci + Cu + Cd\ )$$

s.t.

$$\sum_{k} x_{ijk} - M \cdot y_{ij} \le 0 \qquad (i,j,k) \in \Omega \quad (5.7')$$

$$\sum_{k \in K_k} x_{jik} - M \cdot g_{ji} \le 0 \qquad (i,j,k) \in \Omega \quad (5.8')$$

여기서, Ω는 분지한 X_{ijk} 의 집합이다.

$$RP2 = Min \sum_{t} F_t \left[\sum_{i \in Q_t} (1 - z_{tji}) \frac{T_i \cdot R_i}{W} + \sum_{i \in P_t} (1 - z_{tji}) \cdot (1 + H_b) N_b \right]$$

s.t.

$$z_{tji} = \begin{cases} \prod_{j \in K_k} x_{jik} & (i,j,k) \in \Omega,\ j \Rightarrow i, \\ & t\text{는 죠인조회 트랜잭션} \\ 0 & i,j \in Q_t \cup P_t,\ j \Rightarrow i, \\ & t\text{는단일테이블 조회 트랜잭션} \end{cases} \quad (5.9')$$

$$T_i = L_i + \sum_{j=1} \sum_{k} L_{jk} x_{jik} \qquad (i,j,k) \in \Omega$$

$$z_{tji} + z_{tmj} \le 1 \qquad i,j \in Q_t \cup P_t,\ m \Rightarrow j,\ j \Rightarrow i,\ t\text{는결합조회 트랜잭션} \quad (5.10)$$

5.5 분지한계법 절차

본 연구에서 제시한 분지한계법의 절차는 다음과 같다.

단계 0: [초기화]

분지노드를 비어 있음으로 표지하고

단계 2로 간다.

단계 1: [분지노드 결정]
아직 분지하지 않은 분지노드 중에서 최저한계가 가장 작은 노드를 찾는다. 만약 그 노드의 최저한계가 현재까지의 최적해보다 크거나 같으면 끝낸다. 만약 분지하지 않은 분지노드가 없으면 끝낸다.

단계 2: [후보노드 결정]
단계 1에서 분지노드를 x_{ijk}라고 하면 x_{ijk}를 사용하는 죠인조회 트랜잭션 중에서 아직까지 분지하지 않은 속성(k_c)을 후보노드로 한다. 만약 그 k_c 중에서 부분분지 기준에 부합되는 것이 있으면 그것을 후보노드로 하고 분지는 분지노드 x_{ijk}의 결정변수 값과 동일하게 분지한다. 그러한 k_c가 없으면 5.3절의 분지전략에 의해 후보노드를 결정한다.

단계 3: [후보노드의 최저한계 계산]
5.4절의 한계전략을 사용하여 가능성이 있는 최저한계를 계산한다.

단계 4: [가능해 여부 결정]
분지가 잎노드(leaf node)까지 되었고 가용한 저장용량보다 적으면 가능해로, 아니면 불능해로 표지한다. 가능해이고 현재까지의 최적해보다 적으면 최적해를 갱신한다.

단계 5: [분지노드 집합에 분지한 노드 삽입]
분지한 후보노드를 분지노드에 삽입하고 단계 1로 간다.

데이터베이스 설계는 빈번하게 하는 것이 아니고 그 설계 주기가 매우 크므로 최적해를 구하는 시간이 다소 많이 걸리더라도 최적해를 구하는 것이 바람직하다.

본 연구에서 제안한 분지한계법에서 탐색해야 할 노드의 개수는 4.3절에서 논의한 조건에 맞는 결정변수 X_{ijk} 의 개수를 n이라고 하면 $\sum_{m=1}^{n} 2^m = 2^{n+1} - 2$ 이다. 즉, 최악의 계산량(complexity)은 $O(2^n)$으로 지수형이다.

그러나 본 연구의 비정규화는 참조하는 테이블에 참조되는 테이블의 속성을 복사하는 방법이며 그러한 참조관계의 개수는 많지 않다.

〈그림 6.1〉은 테이블의 개수가 네 개인 경우로서 참조관계는 ORDER⇒CUSTOMER, ORDER_DETAIL⇒ORDER, ORDER_DETAIL⇒PRODUCT 의 세 개이다. 그런데 〈그림 6.1〉의 테이블들은 실제적으로 매우 밀접한 관계에 있는 테이블이다.

또, 삽입, 갱신, 삭제, 조회 트랜잭션을 분석하고 수집할 때는 일반적으로 80-20 법칙이 적용된다[14]. 이 법칙은 보통의 경우 20% 정도의 트랜잭션이 전체 시스템이 처리해야 하는 양의 80% 정도를 차지한다는 법칙이다. 그러므로 20% 정도의 트랜잭션만을 고려하면 된다. 또한, 본 수리모형의 해법으로 본 장에서 제안한 효율적인 분지 전략과 한계 전략을 사용하므로 실제적인 계산량은 크게 줄 수 있다.

6. 수치 예제

본 수치 예제는 〈그림 6.1〉과 〈그림 6.2〉와 같이 정규화하고 그 정규화한 테이블을 가지고 역으로 비정규화하여 최적의 성능을 가지는 테이블 설계를 구하는 문제이다.

〈그림 6.1〉에서 화살표의 시작점은 외부키를 표시한 것이고, 화살표의 끝점은 외부키가 참조하

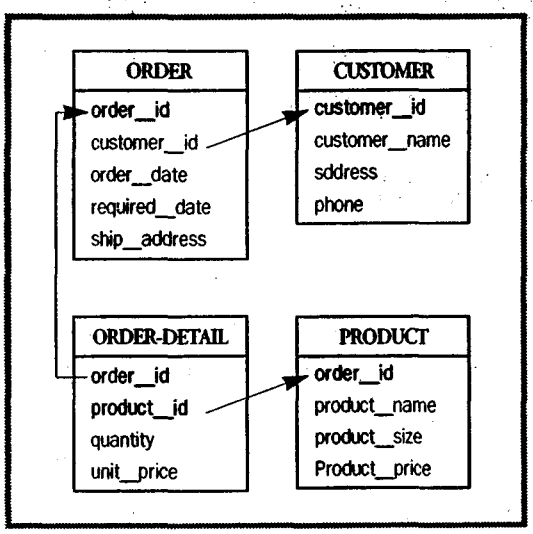
는 기본키를 표시한 것이며, 또 굵은 글씨로 되어 있는 것은 기본키를 표시한 것이다. 즉, ORDER 테이블의 customer_id는 CUSTMER 테이블의 기본키인 customer_id를 참조하는 외부키이고 ORDER-DETAIL 의 order_id는 ORDER 테이블의 기본키인 order_id를 참조하는 외부키이다. 또, ORDER-DETAIL 의 product_id는 PRODUCT 테이블의 기본키인 product_id를 참조하는 외부키이다.

〈그림 6.2〉는 각 테이블의 튜플 개수, 기본키 높이, 속성의 길이를 나타낸다. 그림에서 속성 이름이 굵게 되어 있는 것은 외부키를 나타낸다. 본 예제의 모든 외부키는 인덱스되어 있고 그 높이는 기본키에 대한 인덱스 높이와 같으며 널이 아닌 것으로 한다.

〈그림 6.3〉은 각 테이블에서 발생하는 삽입 트랜잭션, 갱신 트랜잭션, 삭제 트랜잭션, 그리고 조회 트랜잭션의 자료이다. 조회는 한 개의 테이블을 조회하는 트랜잭션도 있고 여러 개의 테이블을 죠인하는 트랜잭션도 있다. 블록 크기는 1024 byte, 블록킹 계수는 4로 하고, 각 테이블 간에 한 개의 튜플에 대응하는 평균 튜플의 개수는 각각 $V_{13}=1$, $V_{24}=5$, $V_{34}=5$ 이다. 또, 가용한 여유 저장용량 S는 1,500 Kbyte 이다.

본 예제의 정규화한 테이블 설계에서 총 액세스 횟수는 본 수리모형을 사용해서 해를 구하면 3252.7 이고 본 연구에서 제안한 비정규화한 설계의 최적해는 〈그림 6.4〉에서 보는 바와 같이 2798.9 으로서 453.8 회가 감소하였다. 최적해는 〈그림 6.5〉에서 보는 바와 같이 ORDER-DETAIL에 PRODUCT의 product_name, product_size, 또 ORDER에 CUSTOMER의 customer_name을 복사해 두는 비정규화된 테

〈그림 6.1〉 정규화된 테이블 설계의 예



〈그림 6.2〉 예제의 테이블 구조

테이블번호	테이블 이름	튜플 개수	기본키 높이	속성번호	속성이름	길이
1	customer	5,000	2	1	customer_id	5
				2	customer name	20
				3	address	100
				4	phone	15
2	product	5,000	2	1	product_size	5
				2	product_name	30
				3	Droduct_size	5
				4	standard_price	10
3	order	5,000	2	1	order_id	5
				2	customer_id	5
				3	order_date	7
				4	required_date	7
				5	ship_address	100
4	order-detail	25,000	3	1	order_id	5
				2	product_id	5
				3	quantity	5
				4	unit_price	10

이블 설계이다.

본 예제에서 만약 삽입, 갱신 트랜잭션의 발생 횟수가 죠인조회 트랜잭션에 비하여 많은 경우를 살펴보면, 최적해는 비정규화하지 않은 정규화한 테이블 설계가 된다. 이러한 경우는 매우 이상적이며 본 수리모형의 최적해인 정규화한 테이블 설계를 그대로 사용하면 된다.

본 예제에서 결정변수가 6개이므로 탐색해야 할 노드의 개수는 5.5절에서 설명한 바와 같이 126개다. 그런데 〈그림 6.4〉에서 보는 바와 같이 노드를 16개 탐색하여 최적해를 구하였다. 이렇게 탐색해야 할 노드의 개수가 크게 감소한 것은 본 연구에서 제안한 알고리듬이 매우 효율적임을 보여준다.

〈그림 6.3〉 예제의 트랜잭션 종류

〈그림 6.4〉에서 5.2절에서 설명한 본 연구의

트랜잭션 번호	테이블 번호	발생 횟수
1	1	1
2	2	1
3	3	10
4	4	50

(a) 예제의 삽입 트랜잭션

트랜잭션 번호	테이블 번호	속성 번호	발생 횟수
1	2	3	1
2	3	4	1
3	4	3,4	2

(b) 예제의 갱신 트랜잭션

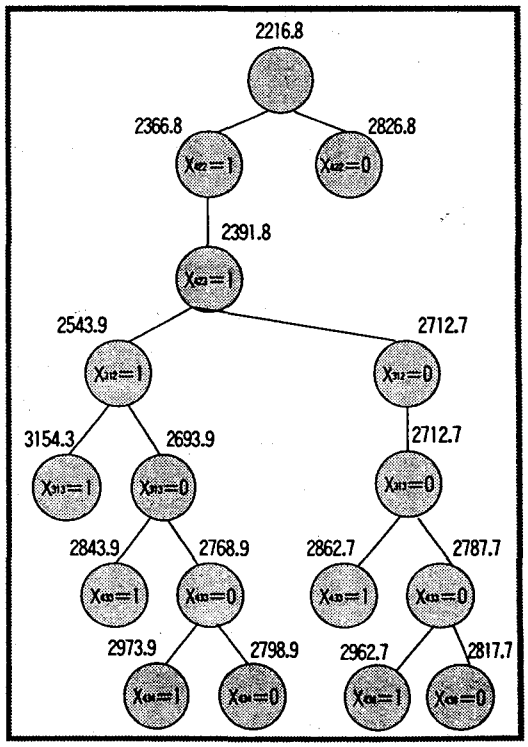
트랜잭션 번호	테이블 번호	발생 횟수
1	4	2

(c) 예제의 삭제 트랜잭션

트랜잭션 번호	테이블 번호	속성 번호	액세스 방법	인덱스 높이	액세스 튜플 개수	발생 횟수
1	2	3	인덱스	2	1	100
2	3	3	인덱스	2	1	50
	1	2,3	인덱스	2	1	
3	4	3	인덱스	3	10	10
	2	2,3	순차	—	5,000	
4	4	3	인덱스	3	1	5
	3	3	인덱스	2	5	
5	1	2,3,4	인덱스	2	1	100
6	3	1,2,3,4,5	순차	—	5,000	5
7	4	2,3	인덱스	3	50	1
	3	4	인덱스	2	10	
	1	2	순차	—	5,000	

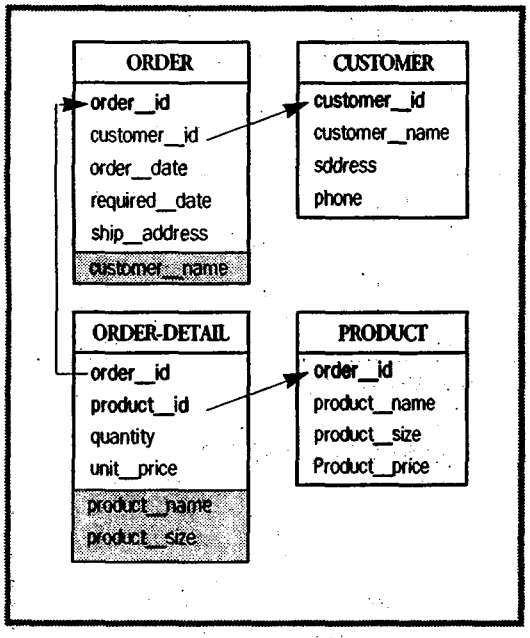
(d) 예제의 조회 트랜잭션

〈그림 6.4〉 분지한계법을 이용한 예법



분지 전략인 부분 분지는 두 번 있었다. 이 부분 분지에 의해 감소한 탐색 분지의 개수는 20개이고 나머지 26개는 5.4절에서 제안한 한계 전략에 의해 감소한 것이다.

〈그림 6.5〉 최적으로 비정규한 테이블 설계



7. 결론

데이터는 여러 부문에서 중요한 전략적 자원이 되어 가고 있다. 이전에는 데이터를 주로 화일로 관리하여 왔으나 정보처리 요구가 증가함에 따라 DBMS 사용이 급격히 증가하고 있다. 크고 복잡한 시스템에서 관계형 DBMS의 성능은 주요한 논점이 되고 있다.

본 연구는 가용한 저장용량에 제약이 있는 경우에 먼저, 완전하게 정규화하고 다음 단계로, 정규화된 데이터베이스를 비정규화하여 최적의 성능을 가지는 데이터베이스를 설계하는 방법이다. 본 연구는 참조하는 테이블에 참조되는 테이블의 속성을 복사하는 비정규화 방법을 사용하였다. 본 연구에서는 최적의 성능을 보장하는 속성만을 복사하여 데이터베이스 설계를 하기 위해 수리적으로 모델링하고 분지한계법을 사용하여 최적해를 구하였다. 기존의 연구에서 제안한 비정규화 방법은 매우 특수한 경우에 제한적으로 사용할 수 있는 정형화되어 있지 않은 방법이며, 또한 데이터베이스에서 반드시 제거되어야 할 삽입, 갱신, 삭제이변의 제거가 보장되지 않는 경우도 있다.

본 연구는 참조하는 테이블에 참조되는 테이블의 속성을 복사해 둠으로써 조인을 방지하여 성능을 향상시키는 방법을 제안하였다. 본 연구는 대부분의 데이터베이스에서 지원하고 있는 참조무결성 제약을 이용한 비정규화 방법이고, 이 참조무결성 제약을 이용함으로써 본 연구의 비정규화 방법은 다른 연구에서 제안한 방법과는 달리 매우 정형적인 비정규화 방법이 된다. 본 연구의 비정규화는 이변이 발생할 수 있지만 그 이변의 제거가 보장되는 명확한 방안이 제시되었다. 이변을 처리하는 데 드는 비용은 기존의 연구에서와 같이 주어지는 것으로 가정하는 것이 아니라 발생할 수 있는 이변을 제거하기 위해 필요한 추가적인 액세스 횟수로 모델링하였다. 본 연구의 비정규화는 조회뿐만 아니라 삽입, 갱신, 삭제 트랜잭션을 수행하기 위한 비용을 액세스 횟수로 고려하였다. 본 연구의 비정규화 방법은 데이터베이스 설계 단계뿐만 아니라 실제로 운용 중인 데이터베이스를 재설계할 때도 쉽게 적용이 가능하다.

데이터베이스 설계는 빈번하게 하는 것이 아니고 그 설계 주기가 매우 크므로 최적해를 구하는 시간이 다소 많이 걸리더라도 최적해를 구하는 것이 바람직하다. 본 연구의 수리모형에서 최악의

계산량은 지수형이지만 복사할 수 있는 조건에 맞는 속성의 개수는 많지 않으며, 또한 중요한 트랜잭션만 고려하면 되기 때문에 복사할 수 있는 속성의 개수가 줄어들 수 있다. 또한, 본 연구에서 제시한 효율적인 분지한계법을 사용함으로써 수치 예제에서 확인한 바와 같이 계산량을 크게 줄일수 있으므로 본 연구의 최적해법은 매우 유용한 해법이라고 할 수 있다.

본 연구의 비정규화 방법은 향후 DBMS 기능중에 포함시켜 DBMS가 자동적으로 비정규화할 수도 있을 것으로 생각한다. 왜냐하면 본 연구는 외부키를 이용한 것이고, 이 외부키에 대한 정의는 DBMS의 시스템 카타로그 또는 데이터 사전에 그 정보가 저장되어 있기 때문이다[5, 6]. 그러나 다른 방법의 비정규화는 그 방법이 정형화되어 있지 않기 때문에 DBMS가 그 정보를 알기 힘들다. 또, 실제 운용 중인 시스템의 액세스 상황은 DBMS가 모니터하여 그 정보를 알 수 있다. 그러므로 외부키에 대한 정보, 시스템의 액세스 상황에 대한 정보, 그리고 데이터베이스의 트리거(trigger) 기능을 이용하여 이변을 방지할 수 있기 때문에 향후 본 연구의 비정규화 방법을 발전시켜 DBMS 스스로 하도록 할 수 있을 것이다.

〈참고문헌〉

[1] 강맹규, 「네트워크와 알고리듬」, 박영사, 서울, 1991.

[2] 장영관, "관계형 데이터베이스의 성능 향상을 위한 비정규화 방법," 박사학위 논문, 한양대학교 대학원, 서울, 1996.

[3] Armstrong, E., S. Bobrowski, J. Frazzini, B. Linden, and M. Pratt, *ORACLE7 Server Application Developer's Guide, Oracle* Corporation, Redwood, CA, 1992.

[4] Batini, C., S. Ceri, and S. B. Navathe, *Conceptual Database Design*, The Benjamin/Cummings Publishing Company, Inc., Redwood, CA, 1994.

[5] Bobrowski, S., E. Armstrong, C. Closkey, and B. Linden, *ORACLE7 Server Administors Guide*, Oracle Corporation, Redwood, CA, 1992.

[6] Bobrowski, S., E. Armstrong, C. Closkey, B. Linden, and M. Pratt, *ORACLE7 Server Concepts Manual*, Oracle Corporation, Redwood, CA, 1992.

[7] Cerpa, N., Pre-physical Data Base Design Heuristics, *Information & Management*, Vol. 28, No. 6, pp. 351-359, 1995.

[8] Chu, W. W. and I. T. Ieong, "A Transaction-based Approach to Vertical Partitioning for Relational Database Systems," *IEEE Transactions on Software Engineering*, Vol. 19, No. 8, pp. 804-812, 1993.

[9] Codd, E. F., Further Normalization of the Data Base Relational Model, *Data Base Systems*, Courant Computer Science Symposia Series, Vol. 6, Prentice Hall, Englewood Cliffs, NJ, 1972.

[10] Cornell, D. W. and P. S. Yu, An Effective Approach to Vertical Partitioning for Physical Design of Relational Databases, *IEEE Transactions on Software Engineering*, Vol. 16, No. 2, pp. 248-258, 1990.

[11] Corrigan, P. and M. Gurry, *ORACLE Performance Tuning*, O'Reilly & Associates, Inc., Sebastopol, CA, 1993.

[12] Date, C. J. An *Introduction to Database Systems*, Addison-Wesley Publishing Company, Menlo Park, CA, 1995.

[13] Dewan, R. M. and B. Gavish, "Models for the Combined Logical and Physical Design of Databases," *IEEE Transactions on Computers*, Vol. 38, No. 7, pp. 955-967, 1989.

[14] Elmasri, R. and S. B. Navathe, *Fundamentals of Database System*, The Benjamin/Cummings Publishing Company, Inc., Redwood, CA, 1994.

[15] Finkelstein, S., M. Schkolnick, and P. Tiberio, Physical Database Design for Relational Databases, *ACM Transactions on Database Systems*, Vol. 13, No. 1, pp. 91-128, 1988.

[16] Hawryszkiewycz, I. T., *Relational Database Design*, Prentice Hall, Englewood Cliffs, NJ, 1990.

[17] Hoffer, J. A., "An Integer Programming Formulation of Computer Data Base Design Problems," *Information Sciences*, Vol. 11, pp. 29-48, 1976.

[18] Janas, J. M., A Systematic Approach to Database Denormalization, *iti95. Proceedings of the 17th International Conference on Information Technology Interfaces*, pp. 323-328, 1995.

[19] Lawler, E. L., J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem*, John Wiley & Sons, NY, NY, 1985.

[20] Lee, H., "Justifying Database Normalization: A Cost/Benefit Model, *Information Processing & Management*, Vol. 31, No. 1, pp. 59-67, 1995.

[21] Linden, B., *ORACLE7 Server SQL Language Reference Manual*, Oracle Corporation, Redwood, CA, 1992.

[22] Martello, S. and P. Toth, *Knapsack Problems*, John Wiley & Sons, NY, NY, 1990.

[23] Nemhauser, G. L. and L. A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley & Sons, NY, NY, 1988.

[24] Nemhauser, G. L., A. H. G. Rinnooy Kan, and M. J. Todd, *Handbooks in Operations Research and Management Science*, Vol. 1, North-Holland, Amsterdam, Netherlands, 1989.

[25] Reingold, E. M., J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood, NJ, 1977.

[26] Rodgers, U., "Denormalization: Why, What, and How?," *Database Programming & Design*, Vol. 2, No. 12, pp. 46-53, 1989.

[27] Schkolnick, M. and P. Tiberio, Estimating the Cost of Updates in a Relational Databases, *ACM Transactions on Database Systems*, Vol. 10, No. 2, pp. 163-179, 1985.

[28] Teorey, T. J., *Database Modeling & Design: The Fundamental Principles*, Morgan Kaufmann Publishers, San Francisco, CA, 1994.

[29] Ullman, J. D., *Database Systmems*, Computer Science Press, Rockville, MD, 1982.

[30] Westland, J. C., "Economic Incentives forDatabase Normalization," *Information Processing & Management*, Vol. 28, No. 5, pp. 647-662, 1992.