DOI: 10.3745/KIPSTD.2009.16-D.4.463

관계형 데이터베이스에서 PIVOT 연산과 차등 파일을 이용한 수평 뷰의 점진적인 관리

신 성 현[†] · 김 진 호[††] · 문 양 세[†††] · 김 상 욱[††††]

요 약

OLAP 이나 e-비즈니스 환경에서는 다차원 데이터의 효율적인 분석을 위하여, 데이터를 여러 형태로 표현하거나 저장한다. 이러한 다차원 구조의 데이터를 차원 애트리뷰트들의 값으로 표시되는 넓은 형태의 수평 뷰로 표현한다. 수평 뷰는 여러 소스로부터 수집한 요약 정보를 유지하도록 실체 뷰로서 저장되며, 복잡한 질의들을 효율적으로 처리하기 위해 사용된다. 그러나, 소스 데이터가 변경될 경우 수평 뷰들의 내용도 수정해야 하는 데, 소스 데이터들이 여러 사이트에 분산되어 있기 때문에 수평 뷰를 관리하는 것은 어렵다. 따라서, 본 연구에서는 점진적인 뷰 관리 방법 중의 하나로 차등 파일을 이용하여 수평 뷰를 관리하는 효율적인 방법을 제시한다. 이러한 방법은 상용 RDBMS에서 제공하는 PIVOT 연산을 이용하여 저장된 수직 형태의 소스 테이블을 수평 뷰로 변환하고, 수직 테이블의 변경 사항을 반영한 차등 파일을 이용하여 수평 뷰에서도 동일하게 적용하는 방법이다. 이를 위해, 우선 저장된 수직 테이블에서 수평 뷰로 변환하여 처리하는 전체적인 프레임워크를 제안한다. 제안한 프레임워크 하에서 수직 테이블을 수평 뷰로 변환하는 PIVOT 연산을 정의한다. 다음으로, 수직 테이블로부터 데이터가 변경될 경우, 데이터에 대한 변경 사항을 차등 파일로 저장한 후, 이를 이용하여 수평 뷰를 갱신하는 방법을 제안한다. 특히, 차등 파일의 구조는 수평 뷰의 구조와 다르기 때문에, 수평 뷰에 적합하도록 변경 사항을 변환해야 한다. 마지막으로 실험을 통하여 제안한 방법은 다른 방법에 비해서 평균 1.2~5.0배까지 성능을 향상시킴을 보인다.

키워드 : OLAP, 데이터 웨어하우스, 다차원 데이터, PIVOT, 차등 파일

Incremental Maintenance of Horizontal Views Using a PIVOT Operation and a Differential File in Relational DBMSs

Sung-Hyun Shin[†] · Jinho Kim[††] · Yang-Sae Moon[†††] · Sang-Wook Kim[††††]

ABSTRACT

To analyze multidimensional data conveniently and efficiently, OLAP (On-Line Analytical Processing) systems or e-business are widely using views in a horizontal form to represent measurement values over multiple dimensions. These views can be stored as materialized views derived from several sources in order to support accesses to the integrated data. The horizontal views can provide effective accesses to complex queries of OLAP or e-business. However, we have a problem of occurring maintenance of the horizontal views since data sources are distributed over remote sites. We need a method that propagates the changes of source tables to the corresponding horizontal views. In this paper, we address incremental maintenance of horizontal views that makes it possible to reflect the changes of source tables efficiently. We first propose an overall framework that processes queries over horizontal views transformed from source tables in a vertical form. Under the proposed framework, we propagate the change of vertical tables to the corresponding horizontal views. In order to execute this view maintenance process efficiently, we keep every change of vertical tables in a differential file and then modify the horizontal views with the differential file. Because the differential file is represented as a vertical form, its tuples should be converted to those in a horizontal form to apply them to the out-of-date horizontal view. With this mechanism, horizontal views can be efficiently refreshed with the changes in a differential file without accessing source tables. Experimental results show that the proposed method improves average performance by 1.2~5.0 times over the existing methods.

Keywords : OLAP, Data Warehouse, Multidimensional Data, PIVOT, Differential File

※ 본 논문은 2008년도 정부(교육과학기술부)의 재원으로 한국과학재단의 지원(R01-2008-000-20872-0) 및 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업(IITA-2009-C1090-0902-0040)의 지원을 받아 수행된 연구임.
† 정 회 원 : 한양대학교 BK21 사업단 정보기술분야 Post-Doc.
†† 정 회 원 : 강원대학교 컴퓨터과학전공 교수
††† 종신회원 : 강원대학교 컴퓨터과학전공 부교수
†††† 종신회원 : 한양대학교 정보통신학부 교수
논문접수 : 2008년 11월 17일
수 정 일 : 1차 2009년 4월 20일, 2차 2009년 5월 22일
심사완료 : 2009년 5월 22일

1. 서 론

OLAP(On-Line Analytical Processing)은 데이터 웨어하우스에 저장된 대용량 데이터에서 여러 관점의 다양한 정보를 추출하기 위해 다차원 데이터 분석을 지원하는 기법이다[2]. OLAP에서는 다차원 데이터 분석을 편리하고 효율적으

로 분석할 수 있도록 뷰를 여러 형태로 표현하거나 저장한다[12]. 이러한 데이터 저장 및 표현 방법 중의 하나가 다양한 차원에 대해 측정값을 유지하는 수평 뷰(*horizontal view*)[1, 3]이다. 이러한 수평 뷰는 마이크로소프트 엑셀의 PIVOT 테이블이나, 통계 데이터 표현에 사용되는 교차 테이블(cross table)과 유사한 형태 구조로서, 애트리뷰트들의 값을 테이블의 행과 열의 색인 값으로 갖는 이차원 테이블을 의미한다[7, 12]. (그림 1(a))는 매장 상품 판매량을 표시한 수평 뷰의 예를 나타낸다. 그림의 수평 뷰를 보면, 각 매장별 혹은 상품별 상품 판매량을 쉽게 파악할 수 있어, 수평 뷰가 다차원 분석에 편리한 뷰로 제공함을 알 수 있다. 그러나, 기존 관계형 DBMS에서는 이러한 수평 뷰를 효과적으로 지원하지 못한다. 그 이유는 기존 RDBMS에서는 (그림 1(b))와 같이 소수의 애트리뷰트(열)과 다수의 튜플(행)로 구성되는 수직 테이블(*vertical table*)을 주로 다루며[1, 3, 5], 대부분의 RDBMS는 테이블이 가질 수 있는 애트리뷰트 개수를 제한하고 있다. 예를 들어, SQLServer 2005와 오라클의 Oracle 9i 등은 테이블이 가질 수 있는 애트리뷰트 개수를 최대 1,024개로 제한하고 있다. 이에 따라, 상용 RDBMS에 저장된 수직 테이블을 수평 뷰로 변환하여 실행하는 방법이 필요하다. Agrawal 등 [1]은 수평 뷰에 대한 질의를 수직 테이블로 변환하는 방법을 제안하였다. 그리고, 수평 뷰를 직접 저장하고, 질의를 처리하는 방법보다 수직 테이블로 저장하고 질의 처리하는 경우가 성능을 향상시킴을 보였다. 이외에도 수평 뷰에 대한 여러 연구가 진행되었다[3, 5, 7]. 그러나, 이들 연구는 수직 테이블에서 갱신된 정보를 수용하고, 수평 뷰에 갱신된 정보를 반영하기 위한 연구가 미흡하다.

수직 테이블에서 PIVOT 연산에 의해 변환된 수평 뷰는 사용자 질의에 대한 빠른 응답을 제공하기 위해서 실체 뷰(*materialized view*)로 저장된다[6]. 이러한 실체 뷰는 자주 요구되는 질의에 관계되는 튜플들이 요약된 형태로 실제 저장된 테이블이다. 또한, 실체 뷰는 수직 테이블에서 추출된 결과를 별도로 저장하고 있으므로, 저장된 수직 테이블이 변경되면 이를 반영하기 위해 수직 테이블의 변경 사항과 동일하게 변경되어야 한다. 이러한 이유는 뷰 내의 데이터가 여러 데이터베이스에 분산되어 저장되어 있기 때문에 많은 시간이 소요되기도 하며, 네트워크 상황에 따라 데이터 소스에 대한 접근이 불가능할 수도 있어 뷰 관리가 절대적으로 필요하다.

이러한 뷰를 관리하기 위한 기법으로는 재정의 기법(view redefinition)과 점진적 뷰 관리 기법(incremental view maintenance)이 연구되었다. 재정의 기법은 데이터 소스의 변경 사항이 발생시, 뷰의 정의에 따라 수직 테이블로부터 뷰를 재계산하는 방법이다[4]. 그러나, 뷰 재정의 기법은 수직 테이블의 접근으로 인해 많은 계산 비용이 소요된다는 단점이 있다. 이에 비해, 점진적 뷰 관리 기법은 수직 테이블의 변경 사항만을 가지고 차등 파일(*differential file*)을 이용하여 뷰를 관리하는 방법이다. 차등 파일을 사용하는 방법은 소스 데이터의 갱신(삽입, 삭제, 갱신)에 대해 데이터 소스의 접근 없이도 뷰에 변경 사항을 반영하는 뷰의 자체적 관리 기법이다[4, 8, 10]. 이러한 뷰의 자체적인 관리는 단지 실체 뷰의 내용과 수직 테이블의 변경 사항만을 가지고 뷰를 관리한다. 따라서 갱신된 데이터 소스에 대한 직접적인 접근이 없기 때문에 질의에 대한 빠른 응답과 높은 이용 가능성을 제공하므로 대형의 뷰들을 관리하는 데 매우 효율적이다.

본 논문에서는 차등 파일을 이용하여 효과적으로 수평적인 실체 뷰를 관리하는 방법을 제안한다. 이를 위해, 상용 RDBMS에서 제공하는 PIVOT 연산을 이용하여 수평 뷰를 생성하고, 이를 실체 뷰로 저장한다. 그리고, 수직 테이블에 대한 변경 사항을 차등 파일로 유지한 후에 이를 수평 뷰로 갱신하는 방법을 제시한다. 이 과정에서, 차등 파일은 많은 양의 데이터 변경이 발생하게 되므로, 변경된 최종의 내용만을 유지하는 알고리즘을 제시한다. 또한, 변경 사항은 수평 뷰에 그대로 적용할 수 없으므로, 수평 뷰에 적용하기 위해 적합한 형태로 변환하여 처리하는 방법과 이를 위한 전체적인 알고리즘을 제안한다.

본 논문의 구성은 다음과 같다. 제2장에서는 본 연구와 관련된 기존 연구를 소개한다. 제3장에서는 본 논문에서 제안하는 수평 뷰를 점진적으로 관리하기 위한 전체적인 처리 과정을 설명한다. 제4장에서는 차등 파일을 이용하여 수평 뷰에 데이터 변경시 발생하는 연산의 정의와 뷰 관리 기법을 소개하고 이에 대한 알고리즘을 제안한다. 제5장에서는 제시된 수평 뷰의 점진적인 관리 기법에 대한 비용 결과를 제시하고 마지막으로, 제6장에서 결론을 맺는다.

ShopID	TV	Radio	Computer	Camera
1	280	⊥	220	150
2	120	145	⊥	110
3	240	⊥	125	⊥

(a) A Horizontal Table (***SalesH***)

ShopID	Product	Sales
1	TV	280
1	Computer	220
1	Camera	150
2	TV	120
2	Radio	145
2	Camera	110
3	TV	240
3	Computer	125

(b) A Vertical Table (***SalesV***)

(그림 1) 수평 테이블(*SalesH*)와 수직 테이블(*SalesV*)의 예.

2. 관련 연구

본 장에서는 수직 테이블에서 수평 뷰로 변환하기 위한 기존 연구와 수직 테이블에서 갱신되는 사항을 뷰에서도 동일하게 유지되는 기존 연구에 대해 설명한다.

Lakshmanan 등 [9]에서는 수직 테이블에서 수평 뷰로 변환하는 방법을 먼저 제안하였다. 이들은 수평 뷰를 다루기 위한 방법으로 *fold, unfold*을 제안하였다. 이들 연산은 수직 테이블에서 수평 뷰로, 혹은 반대로 처리하는 연산이다. 이들 연구를 기반으로, Agrawal 등 [1]은 수직 테이블을 수평 뷰를 변환하여 처리하는 체계적인 방법을 제안하였다. 이들의 방법은 사용자에게 수평 뷰로 제공하고, 실제 저장과 질의는 수직 테이블을 사용하는 방법이다. 이외에도 수직 테이블을 수평 뷰로 제공하는 여러 연구가 진행되었다. 우선, Witkowsk 등 [13, 14]는 다차원 데이터를 스프레드시트 형태의 수평 뷰로 정의하고 조작하는 연구를 수행하였다. 이를 통하여 OLAP 분석 기능을 편리하게 사용할 수 있도록 하였다. 다음으로, Cunningham 등 [5]은 수직 테이블을 수평 뷰로 또는 반대로 변환하는 PIVOT과 UNPIVOT 연산을 관계 대수로 정의하였다. 또한, Chen 등 [3]은 PIVOT 연산을 일반화하여 GPIVOT 연산을 제안하고, 이를 이용하여 수평 뷰에 대한 실체화 뷰(materialized view)를 점진점으로 갱신하는 방법을 제안하였다. 그러나, 이들 연구는 수직 테이블에서 발생한 갱신 정보를 앞서 생성된 수평 뷰에 동일하게 변경되는 기법에 대한 연구가 미흡하다.

다음으로 앞서 생성된 뷰를 수직 테이블의 변경 사항과 동일하게 적용되고, 수직 테이블의 변경에 따라 뷰를 최신의 정보로 유지하는 연구들이 연구되었다. 우선, Colby 등 [4]는 수직 테이블로부터 뷰를 재정의하는 방법을 제안하였다. 이 방법은 수직 테이블의 데이터가 변경 사항이 발생하면, 뷰의 정의에 따라 수직 테이블에서 뷰를 재계산하는 방법이다. 그러나, 이들 연구는 뷰를 갱신하기 위해, 방대한 수직 테이블을 매번 접근해야 하므로 많은 계산 비용이 소요된다. 뷰 재정의 기법의 단점을 보완하기 위해, Lee[10]과 Mohania 등[11]은 수직 테이블의 갱신 사항만을 이용하기 위해 보조 릴레이션과 차등 파일을 제안하였다. 이 방법은 수직 테이블의 접근없이도 뷰에 변경 사항을 반영하는 뷰의 자체적 관리 기법이다. 이들 연구는 변경된 수직 테이블에 대한 직접적인 접근이 없기 때문에 데이터 변경이 발생할 경우의 빠른 질의 처리와 대형의 뷰들을 관리하는 데 매우 효율적이다.

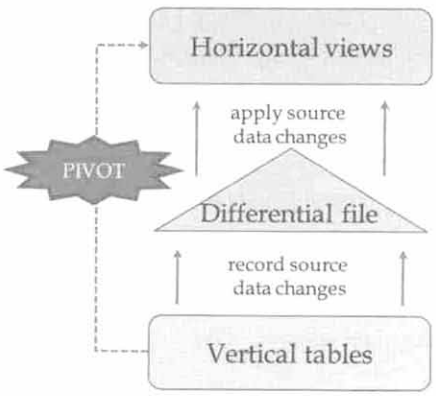
3. 수평 뷰 관리를 위한 전체적인 과정

본 장에서는 차등 파일을 이용한 프레임워크, 수직 테이블에서 수평 뷰로 변환 및 실체 뷰로의 저장하는 방법, 수직 테이블에서의 변경 사항을 저장한 차등 파일 그리고, 전체적인 처리 과정을 설명한다.

첫째, 수직 테이블과 수평 뷰에 갱신된 값을 처리하는 전체적인 프레임워크를 설명한다. (그림 2)는 차등 파일을 이용하여 뷰 관리를 하는 전체적인 프레임워크이다. 그림을 보면, 수직 테이블은 PIVOT 연산을 적용하여 수평 뷰를 생성함을 알 수 있다. 그림에서 오른편은 새로이 갱신된 변경 사항을 유지하는 차등 파일을 나타내며, 차등 파일은 수직 테이블의 변경 사항이 발생된 튜플의 정보를 저장한다. 본 연구에서는 제안한 프레임워크를 기반으로 차등 파일을 이용하여 실체 수평 뷰를 자체적으로 관리하는 방법을 제안한다.

둘째, 데이터베이스에 저장한 수직 테이블과 수평 형태의 뷰에 대해 설명한다. 이를 위해, 본 논문에서 사용하는 표기법은 표 1과 같다. 표 1에서와 같이, 수직 테이블 RT_i는 (Oid,A,M)의 스키마를 가지며, RT_i에 저장되는 각 튜플은 (O_i,A_j,M_j)로 표현된다. 여기서, O_i는 식별자이고, A_j는 애트리뷰트 A의 값이며, M_j는 애트리뷰트 M의 값이다. 다음으로, 수직 테이블에서 유도되고, OLAP에서 표현되는 수평 뷰 테이블 V_{RTi}는 $(Oid,A_1,A_2,...,A_n)$의 스키마를 가지며, V_{RTi}에 표현되는 튜플은 $(O_i,M_1,M_2,...,M_n)$으로 표현된다.

셋째, 수직 테이블에서 수평 뷰로 변환하는 과정에 대해 설명한다. 수직 테이블 RT_i를 뷰 테이블 V_{RTi}의 형태로 표현하는 방법으로 PIVOT 연산을 이용할 수 있다. 마이크로소프트의 SQL Server 2005 등과 같은 상용 RDBMS에서도 제공하는 PIVOT 연산은 데이터베이스에 저장된 수직 테이



(그림 2) 전체적인 프레임워크.

〈표 1〉 주요 표기법

기호	정 의
RT_i	수직 테이블(소스 테이블)
V_{RTi}	RT_i에서 유도된 수평 뷰 테이블
dRT_i	수직 테이블의 변경 사항을 저장한 차등 파일
Oid	튜플 식별자 애트리뷰트
O_i	튜플 식별자의 값(수평 뷰 $VRTi$의 i-번째 튜플)
A. M	수직 테이블의 RTi의 애트리뷰트(A와 M의 값은 각각 A_j와 M_j)
A_j	수평 뷰 V_{RTi}의 j-번째 애트리뷰트 (수직 테이블 RT_i와 차등 파일 dRT_i의 애트리뷰트인 A의 값)
M_j	수평 뷰 VRTi에서 j-번째 애트리뷰트 값 (수직 테이블 RT_i와 차등 파일 dRT_i의 애트리뷰트인 M의 값)

블을 분석이 용이한 수평 뷰로 표현하는 기능을 한다. Chen 등 [3]은 PIVOT 연산을 다음 수식 (1)과 같은 관계 대수식으로 정형적으로 정의하였다.

$$V_{RTi} = \mathrm{PIVOT}^{[A1,A2,\ldots,An]}_{A \text{ on } M}(RT_i) = [\ \bowtie_{j=1}^{n} \pi_{Oid,M}(\sigma_{A=A_j}(RT_i))] \quad (1)$$

수식 (1)을 보면, 수직 테이블 RT_i를 애트리뷰트 A의 값 $[A_1,A_2,\ldots,A_n]$에 대해 PIVOT하면, 수평 뷰 V_{RTi}로 변환된 것을 알 수 있다. 이 과정은 먼저 애트리뷰트 A의 값이 A_j인 튜플 (O_i,A_j,M_j)들을 추출하고, 이들을 완전 외부 조인($\bowtie$)으로 결합하여 수평 뷰로 표현한다. 또한, 생성된 수평 뷰는 자주 요구되는 질의에 대한 빠른 응답을 위해, 실체 뷰(materialize view)로 저장할 수 있다. 이러한 실체 뷰는 자주 요구되는 질의에 관계되는 튜플들이 요약된 형태로서, 실제로 데이터베이스에 저장된 테이블을 의미한다. (그림 3)은 이러한 수평 뷰에 대한 실체 뷰의 예를 나타낸다.

다음으로, 수직 테이블의 변경 사항만을 가지는 차등 파일에 대해 설명한다. 차등 파일은 수직 테이블에 대한 변경 사항을 별도로 유지하며, 이를 이용하여 수직 테이블의 접근없이 뷰를 갱신할 수 있다. 이러한 차등 파일은 dRT_i(*Oid, A,M,optype,time*)의 스키마를 가진다. 여기서, *optype*은 튜플 변경에 적용된 연산을 나타내며, *insert*와 *delete* 중의 하나를 가진다. *update*의 경우, *insert*와 *delete*의 조합으로 고려한다. 특히, *insert*는 수식 (2)와 같이, ΔI로 표기하며, 수직 테이블 RT_i의 튜플(*Oid,A,M*)에 속하지 않으면서 차등 파일 dRT_i에 속하는 튜플(*Oid,A,M*)을 의미한다. 그리고, *delete*는 수식 (3)과 같이, ∇D로 표기하며, 삭제하고자 하는 튜플(*Oid,A,M*)이 수직 테이블에 속하면서 차등 파일에 속하는 튜플(*Oid,A,M*)을 의미한다. *time*은 수직 테이블에서 튜플이 변경되는 시간(time-stamp)을 나타낸다. 이러한 타임스탬프 값을 사용하여 데이터들 간의 입력 순서를 파악할 수 있고, 오래된 데이터를 폐기할 수 있다[10, 11].

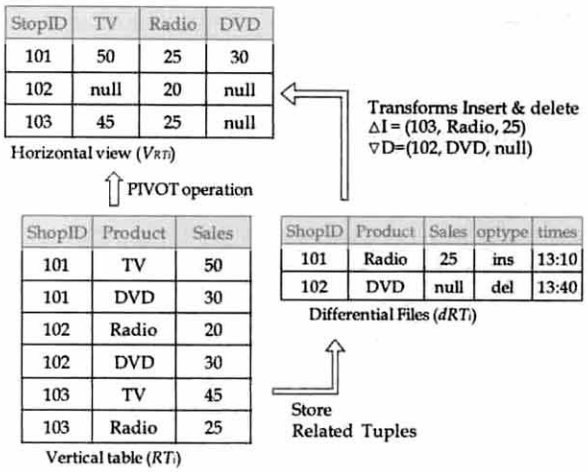
$$\Delta I = \{(Oid,A,M) \mid (Oid,A,M) \notin \pi_{Oid,A,M}(RT_i) \wedge (Oid,A,M) \in \pi_{Oid,A,M}(dRT_i)\} \quad (2)$$

$$\nabla D = \{(Oid,A,M) \mid (Oid,A,M) \in \pi_{Oid,A,M}(RT_i) \wedge (Oid,A,M) \in \pi_{Oid,A,M}(dRT_i)\} \quad (3)$$

마지막으로, 수직 테이블과 수평 뷰에 갱신된 값을 처리하는 전체적인 처리 과정을 설명한다. (그림 4)는 본 논문에서 차등 파일을 이용한 방법의 처리 과정을 나타낸다. 그림을 보면, 수직 테이블 RT_i은 PIVOT 연산을 적용하여 수평 뷰 V_{RTi}로 생성함을 알 수 있다. 앞서 언급한 수식 (1)을 이용한다. 그림에서 오른편은 새로이 갱신된 변경 사항을 유지하는 차등 파일 dRT_i를 나타내며, 차등 파일은 수직 테이블의 변경 사항이 발생된 튜플의 정보를 저장한다. 이러한

CREATE VIEW V_{dRTi}(*Oid*, A_1, A_2, ⋯, A_n) AS
SELECT *Oid*, A_1, A_2, ⋯, A_n
FROM RT_i
PIVOT (SUM(M) FOR A IN(*Oid*, A_1, A_2, ⋯, A_n)) AS PVT

(그림 3) 수평 뷰를 정의하는 구문



(그림 4) 차등 파일을 이용한 전체적인 처리 과정

처리 과정을 기반으로, 본 연구에서는 차등 파일을 이용하여 실체 수평 뷰를 자체적으로 관리하는 방법을 제안한다.

4. 차등 파일을 이용한 수평 뷰의 관리

본 장에서는 수직 테이블의 변경 사항에 따라 수평 뷰를 최신 정보로 유지하기 위한 방법을 제안한다. 제4.1절에서는 수평 뷰를 위해 차등 파일을 관리하는 방법과 알고리즘을 제시한다. 그리고, 제4.2절에서는 차등 파일의 튜플을 수평 뷰에 갱신할 수 있도록 연산의 변환과 이에 대한 전체적인 알고리즘을 제안한다.

4.1 차등 파일의 관리

본 절에서는 수직 테이블의 변경 사항을 기록한 일련의 정보인 차등 파일에 대해 설명한다. (그림 4)를 보면, 차등 파일은 수직 테이블마다 하나씩 존재하며, 각 수직 테이블의 변경 사항을 유지한다. 이러한 차등 파일 내의 변경 사항은 뷰를 갱신한 시간부터 앞으로 갱신되는 시간까지 유지된다. 일단, 뷰가 갱신되면 차등 파일 내에 변경 사항은 삭제되고, 갱신 후에는 다시 기록한다. (그림 5(a))는 이러한 차등 파일의 예를 나타낸다. 그림에서, 수직 테이블에 적용된 튜플 (*101, Radio, 25*)은 변경된 시간 *time*이 13:10에 삽입되었으며, 다른 튜플 (*102, TV, null*)은 변경된 시간 *time*이 13:40에 삭제되었음을 알 수 있다. 여기서, 널(*null*)은 값이 삭제됨을 묵시적으로 나타낸다.

Oid	Product	Sales	optype	times
101	Radio	25	ins	13:10
102	TV	40	ins	13:13
102	TV	null	del	13:22
103	DVD	30	ins	13:25
102	TV	35	ins	13:34
103	Radio	null	del	13:37
102	TV	null	del	13:40

(a) Before eliminating duplicated data

Oid	Product	Sales	optype	times
101	Radio	25	ins	13:10
102	TV	40	ins	13:13
103	DVD	30	ins	13:25
103	Radio	null	del	13:37
102	TV	null	del	13:40

(b) After eliminating duplicated data

(그림 5) 차등 파일의 예

수직 테이블의 데이터들이 연속적으로 변경된다면, 차등 파일 내에는 동일한 애트리뷰트 *Oid*와 *A*를 갖는 튜플의 연속적인 갱신이 존재할 수 있다. 차등 파일 내의 정보 중 동일한 애트리뷰트의 값을 갖는 튜플들이 연속적으로 변경된다면, 차등 파일에는 처음과 나중의 변경 사항만을 제외하고 중간 과정의 변경 사항은 뷰 갱신에 대해 불필요한 과정으로 존재하게 된다. 예를 들어, 동일한 애트리뷰트의 값을 갖는 튜플이 연속적으로 *ins*→*del*→*ins*→*del* 된다면, 연산 정보의 처음 *ins*와 마지막 *del* 이외에, 중간 과정은 불필요하게 된다. 따라서, 이러한 과정을 중복 삭제(*duplicate elimination*)이라 하며, 차등 파일 내에서 같은 튜플에 대해 처음과 나중의 정보만을 남기고, 나머지는 모두 제거한다. (그림 5(a))는 중복 튜플을 삭제하기 이전의 차등 파일을 나타낸다. 여기서, 튜플(*102, TV, ...*)는 처음과 나중의 값만을 남겨두고 13:22분의 *del*과 13:34분의 *ins* 튜플은 삭제된다. (그림 5(b))는 중복 삭제한 후의 차등 파일의 예를 보이고 있다. 이러한 과정으로, 뷰에 전달되는 튜플 수는 중복 제거되는 비율에 따라 많은 양이 감소하게 된다.

다음으로, (그림 6)은 차등 파일에서 불필요한 튜플의 중복 제거를 위한 알고리즘을 나타낸다. 라인 1에서는 차등 파일 dRT_i을 정렬하고, 라인 2-5에서는 select 문을 이용하여 차등 파일에서 튜플 *Oid, A, M, optype, time*을 검색하고, 이들 데이터를 변수에 저장한다. 라인 6에서는 차등 파일에서 중복된 값을 검사하는 *CheckOpCount*를 사용한다. 라인 7-15에서는 이전 튜플의 키와 현재 튜플을 서로 비교하여 중복된 튜플을 제거하고, 초기의 튜플과 최종의 튜플을 보존하는 방법을 수행한다. 여기서, 라인 8-12에서는 커서에 의해 튜플을 검색하여 컬럼의 집합 T_{Cur}에 저장하고, 이전 튜플의 키와 현재 튜플의 키를 비교하여 서로 동일하면 *CheckOPCount*를 하나씩 증가시킨다. 여기서, *CheckOPCount*가 증가되면, 중복된 튜플이므로 현재 튜플을 삭제한다. 라인 13에서는 기존과 현재의 튜플의 키가 서로 다르게 되므로, 기존 튜플은 최종 튜플로 남아있게 되고 현재 튜플은 새로운 최초 튜플이 된다. 라인 14에서는 현재 튜플 T_{Cur}는 기존 튜플 T로 재배치한다. 마지막으로 라인 15에서는 모든 과정이 끝나게 되므로, 활성화된 커서를 종료한다.

```
Algorithm DuplicateElimination (Table dRTi)
01  Sorting the tuples stored in the dRTi;
02  Declare Cursor for
03      Select Oid, A, M, optype, time from dRTi;
04  Open Cursor;
05  Fetch from Cursor T = {Oid, A, M, optype, time};
06  CheckOpCount = 1;
07  While Cursor is not null
08      Fetch from Cursor TCur = {OidCur, ACur, MCur, optypeCur, timeCur};
09      if (Oid = OidCur ∧ A = ACur) {
10          CheckOpCount = ++i;
11          if(CheckOpCount > 2) delete a tuple T;
12      }
13      else CheckOpCount = 1;
14      replace T by TCur;
15  close Cursor;
```

(그림 6) *DuplicateElimination* 알고리즘

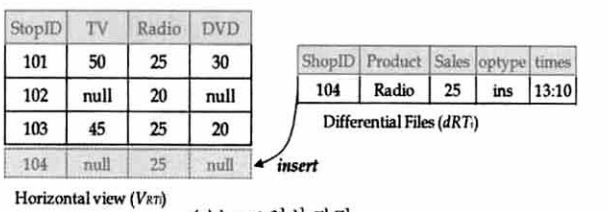
4.2 수평 뷰의 관리

본 절에서는 차등 파일을 이용하여 수직 테이블로부터 변환된 수평 뷰를 관리하는 방법에 대해 제시한다. 수직 테이블의 변경 사항을 기록한 차등 파일 내의 변경 사항은 뷰에 주기적으로 갱신을 해야 한다. 그러나, 차등 파일 내의 변경 사항은 PIVOT 연산이 적용된 수평형태의 뷰로 직접 적용할 수 없으므로, 이를 적용하기 위해서는 변경 연산을 수평 뷰에 적합한 형태로 변환해야 한다. 이에 따라, 본 논문에서는 변경 연산인 삽입, 삭제 그리고 갱신 연산으로서, 이를 이용한 관리 방법을 제시한다. (그림 7)은 차등 파일을 이용하여 수평 뷰에 여러 연산들을 적용하는 방법을 나타낸다.

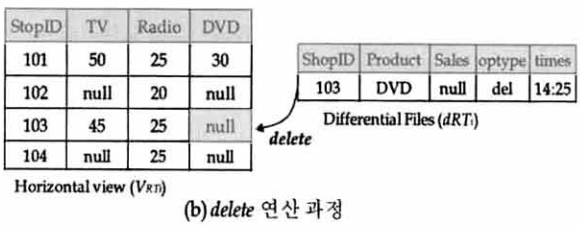
먼저, 수평 뷰에 대한 삽입 연산에 대해 설명한다. 삽입 연산은 차등 파일 dRT_i의 연산 정보 *optype*에 의하여 수평 뷰 V_{RTi}에 변환하여 적용할 수 있다. 다음 정리 1은 이러한 삽입연산에 대한 변환 규칙을 나타낸다.

[정리 1] 수직 테이블 RT_i가 변환된 수평 뷰를 V_{RTi}라 할 때, V_{RTi}에 대한 삽입 연산은 다음 수식 (6)에 의해 차등 파일 dRT_i와 수평 뷰 V_{RTi}와 조인한 후 합집합으로 적용할 수 있다. 단, *Oid*는 O_j이며, A_j는 널이다.

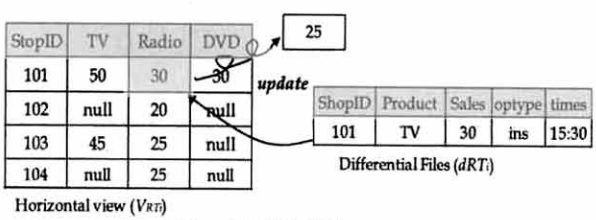
$$V_{RTi}^{new} = V_{RTi} + \Delta I(dRT_i) = V_{RTi} \cup [\cup_{j=1}^{m} V_{RTi} \bowtie \pi_{Oid,M}(\sigma_{optype='ins' \wedge A='A_j'}(dRT_i))] \quad (4)$$



(a) *insert* 연산 과정



(b) *delete* 연산 과정



(c) *update* 연산 과정

(그림 7) 수평 뷰에서 차등 파일을 이용한 변경 사항의 적용

[증명] 우선, V_{RTi}을 V로, $\pi_{Oid,M}(\sigma_{optype='ins' \wedge A='A_j'}(dRT_i))$를 R로 표기한다고 가정하자. 여기서, $(Oid, M_1, ..., M_k)$는 V에 속하고, (Oid, M_x)는 R에 속한다면, $V \bowtie\!\sqsubset R$는 다음과 같이 표기할 수 있다.

$$
\begin{aligned}
&V \bowtie\!\sqsubset R \\
&= \{(Oid, M_1, ..., M_k, M_x) \mid (Oid, M_1, ..., M_k) \in V \wedge (Oid, M_x) \in R\} \\
&\quad + \{(Oid, \perp, ..., \perp, M_x) \mid (Oid, M_x) \in R \wedge (\perp = null)\} \\
&= \{(Oid, M_1, ..., M_k, M_x) \mid ((Oid, M_1, ..., M_k) \in V \\
&\quad \vee (M_1 = \perp \wedge ... \wedge M_k = \perp)) \wedge (Oid, M_x) \in R\}
\end{aligned} \tag{5}
$$

수식 (5)는 $V \bowtie\!\sqsubset R$의 경우, 튜플 $(Oid, M_1, ..., M_k)$와 (Oid, M_x)에서 대응되는 키 Oid와 결합한 튜플들의 결과는 $(Oid, M_1, ..., M_k, M_x)$이다. 만일, V에 관련된 튜플들이 없으면 새로운 튜플들이 생성되는 것으로 튜플들의 결과는 $(Oid, \perp, ..., \perp, M_x)$와 같이 널 값과 함께 생성된다. 그런 후, 생성된 중간 결과 튜플을 기존의 V_{RTi}에 합집합 연산 $V_{RTi} \cup [V \bowtie\!\sqsubset R]$을 적용하면, 삽입 연산은 수식 (4)로 나타낼 수 있으므로, 증명이 완료된다. □

정리 1을 보면, 차등 파일 dRT_i에서 애트리뷰트 $optype$의 값이 ins이다. 먼저, dRT_i에서 애트리뷰트 A의 값 A_j인 튜플(Oid, M_j)들을 추출하고, 이들 추출한 결과를 수평 뷰 V_{RTi}와 오른쪽 외부 조인$(\bowtie\!\sqsubset)$으로 결합한 후, 이를 다시 V_{RTi}와 합집합하면, 변경된 수평 뷰 V_{RTi}^{new}로 수행됨을 알 수 있다. (그림 7(a))는 차등 파일 $dRTi$의 변경 정보에 의해 수평 뷰 V_{RTi}에 새로운 튜플을 삽입한 예로 나타낸다.

다음으로, 수평 뷰에 대한 삭제 연산에 대해 설명한다. 수직 테이블 RT_i와 수평 뷰 V_{RTi}는 차등 파일의 연산 정보 $optype$의 값이 del일 경우 삭제에 대한 변경 사항을 적용할 수 있다. 이러한 삭제 연산은 정리 2로 나타낼 수 있다.

[정리 2] 수직 테이블 RT_i가 변환된 수평 뷰를 V_{RTi}라 할 때, V_{RTi}에 대한 삭제 연산은 다음 수식 (6)에 의해 차등 파일 dRT_i를 이용하여 수평 뷰 V_{RTi}에서 특정 튜플을 삭제하고, 널이 포함한 새로운 튜플을 포함하는 것으로 적용할 수 있다. 단, dRT_i에서 M의 값은 $null$이다.

$$
\begin{aligned}
V_{RTi}^{new} &= V_{RTi} + \nabla D(dRT_i) \\
&= V_{RTi} - [V_{RTi} \bowtie \pi_{Oid}(\sigma_{optype='del' \wedge A='A_j'}(dRT_i))] \\
&\quad + [\cup_{j=1}^{m} V_{RTi} \bowtie\!\sqsubset \pi_{Oid,null}(\sigma_{optype='del' \wedge A='A_j'}(dRT_i))]
\end{aligned} \tag{6}
$$

[증명] 수식 (6)에서 우선, V_{RTi}을 V라 하고, $\pi_{Oid}(\sigma_{optype='ins' \wedge A='A_j'}(dRT_i))$를 R이라 하자. 여기서, $(Oid, M_1, ..., M_k)$는 V에 속하고, Oid는 R에 속한다면, $V \bowtie R$는 다음과 같이 표기할 수 있다.

$$
\begin{aligned}
&V \bowtie R \\
&= \{(Oid, M_1, ..., M_k, M_x) \mid (Oid, M_1, ..., M_k) \in V \wedge (Oid) \in R\}
\end{aligned} \tag{7}
$$

다음으로, $\pi_{Oid,null}(\sigma_{optype='\nabla' \wedge A='A_j'}(dRT_i))$을 R'라 하면, V_{RTi}^{new}는 다음 수식 (8)이 성립한다. 성립하는 이유는 새로운 뷰 V_{RTi}^{new}는 수식 (7)에 의해 특정 Oid가 포함한 튜플이 삭제되고, 수식 (5)에 의하여 널이 포함한 새로운 튜플로 변경됨으로써 본 증명이 완료된다.

$$
\begin{aligned}
&V_{RTi}^{new} \\
&= \{(Oid, M_1, ..., M_k, M_x) \mid (Oid, M_1, ..., M_k, M_x) \in V\} \\
&\quad - \{(Oid, M_1, ..., M_k, M_x) \mid (Oid, M_1, ..., M_k) \in V \wedge (Oid, M_x) \in R\} \\
&\quad + \{(Oid, M_1, ..., M_k, \perp) \mid ((Oid, M_1, ..., M_k) \in V \\
&\quad \wedge ((Oid, \perp) \in R' \wedge \perp = null\}
\end{aligned} \tag{8}
$$

□

정리 2를 보면, 수평 뷰는 다음과 같은 과정으로 변경 사항을 적용한다: 1) 수평 뷰 V_{RTi}와 차등 파일 $dRTi$의 애트리뷰트 $optype$와 A의 값이 각각 del과 A_j인 튜플들을 추출하여 조인$(\bowtie)$한 후, V_{RTi}에서 해당 튜플을 삭제한다. 그런 후, 2) 수평 뷰 V_{RTi}와 차등 파일 $dRTi$에서 애트리뷰트 $optype$과 A의 값이 각각 del과 A_j인 튜플 (O_i, M_j)를 오른쪽 외부 조인$(\bowtie\!\sqsubset)$으로 결합한다. 마지막으로, 1)과 2)의 과정에서의 임시 결과들을 V_{RTi}에 적용하면 기존 값이 $null$로 변경됨을 알 수 있다. (그림 7(b))는 수평 뷰 V_{RTi}에 저장된 데이터의 삭제를 차등 파일 $dRTi$에 저장된 변경 정보에 의해 삭제된 예를 나타낸다.

마지막으로, 수평 뷰에 대한 갱신 연산에 대해 설명한다. 갱신 연산은 삭제와 삽입 연산의 조합으로 간주할 수 있지만, 구조가 다른 수평 뷰에 대해서는 다르게 고려해야 한다. 이에 따라, 갱신 연산은 수평 뷰에서 해당 튜플의 값이 존재할 때 값을 변경하기 위한 대상으로 적용한다. 다음 정리 3은 갱신 연산에 대한 변환 규칙을 나타낸다.

[정리 3] 수직 테이블 RT_i가 변환된 수평 뷰를 V_{RTi}라 할 때, V_{RTi}에 대한 갱신 연산은 다음 수식 (9)에 의해 차등 파일 dRT_i와 조인하여 수평 뷰 V_{RTi}에서 특정 튜플을 삭제하고, 갱신된 값이 포함한 새로운 튜플로 변경하는 것으로 적용할 수 있다. 단, $Oid=O_i$이며 dRT_i에서 $M \neq null$이다.

$$
\begin{aligned}
V_{RTi}^{new} &= V_{RTi} + \Delta I(dRT_i) \\
&= V_{RTi} - [V_{RTi} \bowtie \pi_{Oid}(\sigma_{optype='ins' \wedge A='A_j'}(dRT_i))] \\
&\quad + [\cup_{j=1}^{m} V_{RTi} \bowtie\!\sqsubset \pi_{Oid,M}(\sigma_{optype='ins' \wedge A='A_j'}(dRT_i))]
\end{aligned} \tag{9}
$$

[증명] 수식 (5)와 (8)을 사용하여 널 값 대신 갱신한 값으로 사용하는 것을 제외하고는 정리 2에 대한 증명과 동일한 과정으로 증명할 수 있으므로 수식 (9)가 성립한다. □

수식 3을 보면, 갱신 연산의 과정은 먼저, 수평 뷰 V_{RTi}와 차등 파일 dRT_i에서 애트리뷰트 $optype$과 A의 값이 각각 ins와 A_j인 튜플들을 조인한 후, 수평 뷰 V_{RTi}와 조인$(\bowtie)$하여 해당 튜플들을 추출하여 삭제한다. 다음으로 V_{RTi}와 dRT_i에서 튜플 (O_i, M_j)를 추출하여 오른쪽 외부 조인$(\bowtie\!\sqsubset)$으로 결합한 임시 결과를 V_{RTi}로 튜플의 기존 값과 변경된

값을 함께 삽입하는 과정임을 알 수 있다. (그림 7(c))는 차등 파일 *dRTi*에 저장된 변경 정보를 이용하여 수평 뷰 V_{RTi}에 저장된 데이터를 새로운 값으로 변경함을 나타낸다.

이제, (그림 8)은 차등 파일 *dRTi*를 이용하여 수평 뷰 V_{RTi}를 관리하는 알고리즘을 나타낸다. 우선, 라인 1에서는 차등 파일 *dRTi*을 정렬하고, 라인 2-3에서는 select 구문을 이용하여 한 튜플(*Oid, A, M, optype, time*)을 가리키는 커서 *Cursor*를 선언한다. 다음으로, 라인 4-6에서는 차등 파일 *dRTi*의 튜플 하나를 추출하여 데이터 변수에 저장한다. 다음으로, 라인 7에서는 차등 파일 *dRTi*에서 애트리뷰트 *optype*의 값을 판별하여, *optype*의 값이 *ins*이면 라인 8-12에서 수행하고, 반대로 *del* 값을 가지면, 라인 13-15에서 수행한다. 삽입 연산에 대해, 라인 8-9에서는 수평 뷰 V_{dRTi}의 값이 널(*null*)이거나 해당 *Oid*가 존재하지 않을 때, 수식 (6)을 이용하여 차등 파일 *dRTi*의 튜플 *Oid*와 *M*을 변경하여 수평 뷰 V_{dRTi}의 값을 삽입한다. 갱신 연산에 대해, 라인 10-12에서는 수평 뷰 V_{dRTi}에서 기존 값을 삭제하고 새로운 값으로 변경한다. 여기서, *dRTi*에 저장된 튜플을 수평 뷰 V_{dRTi}에 적용하기 위해 값을 변경하기 위한 식 (9)을 적용한다. 삭제 연산에 대해, 라인 13-15에서는 수평 뷰 V_{dRTi}에 특정 튜플을 검색한 후, 수식 (8)을 이용하여 기존의 값을 삭제한다. 결국, 이런 과정으로 수직 테이블의 변경 사항은 차등 파일을 통해 수평 뷰를 갱신할 수 있다.

결론적으로 요약하면, 수평 뷰 테이블의 내용을 최신 상태로 유지하기 위해서는 수직 테이블이 갱신되었을 때 뷰 테이블도 동일하게 갱신되어야 한다. 이때, 수직 테이블에서 갱신된 데이터를 차등 파일로 저장하고, 차등 파일을 이용하여 수평 뷰 테이블로 적용하는 과정이 필요하다. 특히, 차등 파일은 동일한 갱신 데이터들이 필요 이상으로 중복되거나, 불필요한 저장 공간이 증가될 수 있다. 따라서, 차등 파일의 갱신 데이터를 수평 뷰로 적용하기 전에, 이들 데이터들을 제거하는 알고리즘 *DuplicateElimination*을 이용하여 차등 파일 내에 불필요한 데이터를 제거한다. 그런 후, 정제된 차등 파일을 이용하여 수평 뷰 테이블 내에 삽입, 삭제, 갱신 연산을 결정하기 위해, 알고리즘 *ViewManagement*를 이용하여 특정 데이터를 갱신한다. 이러한 수평 뷰 테이블은 수직 테이블에서 다른 연산이 동안 유지된다.

```
Algorithm ViewManagement (View Vi, Differential File dRTi)
01 Sorting the tuples in the dRTi;
02 Declare Cursor For
03  Select Oid, A, M, optype, time From dRTi;
04 Open Cursor;
05 While Cursor is not null
06   Fetch Cursor into Oid, A, M, optype, and time;
07   If (optype = 'ins')
08     If (a value in Vi is null or a Oid in Vi is no values)
09       Insert into Vi values dRTi.Oid and dRTi.M using formula (4);
10     Else
11       Find suitable values in Vi to update ones in dRTi;
12       Update values in Vi with identical values in dRTi using formula (9);
13   Else // (optype = 'del')
14     Find a suitable value to delete in Vi;
15     Then, delete values in Vi with identical values in dRTi using formula(6);
16 Close Cursor;
```

(그림 8) ViewManagement 알고리즘

5. 성능 평가

본 장에서는 제안한 방법과 기존의 방법들에 대한 성능을 평가한다. 제5.1절에서는 성능을 측정하기 위한 실험 데이터와 실험 환경에 대해 기술하고, 제5.2절에서는 실험 결과를 제시한다.

5.1 실험 환경

실험 대상은 수직 테이블에서 차등 파일을 적용하는 방법(Source), 차등 파일을 이용하되 중복 데이터를 제거하지 않는 상태에서 적용한 방법(Dup-Method), 그리고, 제안한 방법(DF-Method) 방법으로 성능을 측정하였다. 즉, 다음과 같이 세 가지 방법을 실험하였다.

- Source: 변경 사항은 수직 테이블에 직접 반영 [4]하고, 이를 다시 수평 뷰로 변환 [1,3]하는 방법이다.
- Dup-Method: 차등 파일을 이용하는 방법으로, 수직 테이블에 적용한 변경 사항이 차등 파일에 적용 [10,11]하고, 이러한 차등 파일은 중복 삭제 과정 없이 수평 뷰에 적용하는 방법이다.
- DF-Method: 본 논문에서 제안한 방법으로서, 수직 테이블에 발생한 갱신 데이터를 중복 삭제 과정을 통해 차등 파일을 유지하고, 이를 수평 뷰에 적용하는 방법이다.

실험을 수행한 하드웨어 플랫폼은 Intel Core2 1.86GHz CPU, 1GB RAM을 장착한 PC이며, 소프트웨어 플랫폼은 마이크로소프트의 Window 운영체제, SQL Server 2005 DBMS이다. 매개 변수로는 Agrawal 등 [1]의 실험에서와 같이, 데이터 집합, 널 값의 비율 등을 이용하여 실험하였다. 데이터 집합은 수평 뷰을 기준으로 열의 수와 행의 수(*#cols*×*#rows*)인 200×100K로 구성하였고, 널 값 비율은 90%, 95%를 사용하였다[1]. 그리고 제안한 방법에서 이용한 차등 파일은 전체 튜플 중 중복 데이터의 비율을 20%, 40%, 80%로 달리하면서 실험하였다.

5.2 실험 결과

본 절에서는 세 가지 방법에 대한 실험 결과를 설명한다. 먼저, 실험 1)에서는 삽입 연산이 수행할 경우에 수행 시간을 측정하였고, 실험 2)에서는 삭제 연산이 수행할 경우에 수행 시간을 실험하였다. 또한, 실험 3)에서는 삽입 및 삭제 연산에 대한 비교 실험을 수행하였다.

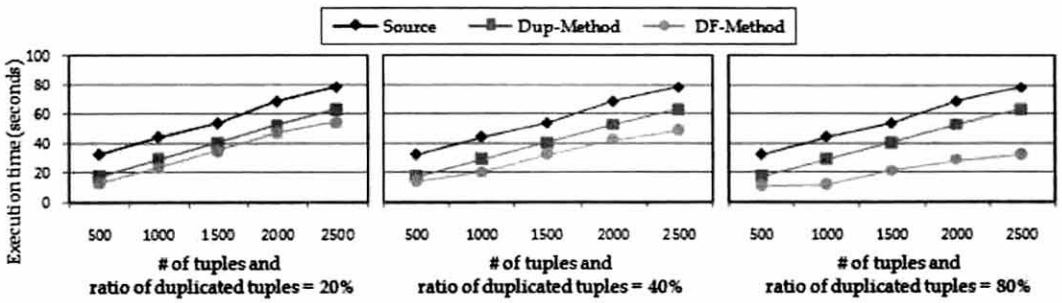
실험 1) 삽입 연산에 대한 수행 시간

(그림 9)는 데이터 집합에 차등 파일의 튜플 수를 증가에 따라 세 가지 방법의 삽입 실험 결과를 나타낸다. (그림 9(a))는 널 값이 90%인 경우의 실험 결과를, (그림 9(b))는 95%인 경우의 실험 결과를 각각 나타낸다. 각 실험에서는 차등 파일의 중복된 튜플 수의 비율을 변경하여 삽입 및 갱신 연산에 대한 실제 수행시간을 측정하였다. (그림 9)를 보면, 전체적으로 DF-Method가 Source 방법과 Dup-Method보다 성능을 향상시킨 것으로 나타낸다. 우선, Source 및 Dup-Method가 DF-Method보다 성능이 저하된 이유는 Source 방법은 수직 테이블에 다수의 갱신 데이터를 직접 적용한 후, 수평 뷰로 제공하기 위해 전체 수직 테이블에서 다시 수평 뷰로 변환하기 때문이다. 이 과정에서 수평 뷰로 변환할 때 수직 테이블의 전체 튜플들을 액세스하게 되므로 수행 시간이 오래 걸린다. 그리고, Dup-Method는 갱신 데이터를 저장한 차등 파일을 이용하기는 하나, 중복 데이터를 제거하는 과정이 없으므로 불필요한 데이터를 포함할 가능성이 있다. 이로 인해 차등 파일의 저장 공간이 크며, 이들을 수평 뷰에 적용하는 것 또한 수행 시간이 오래 걸린다.

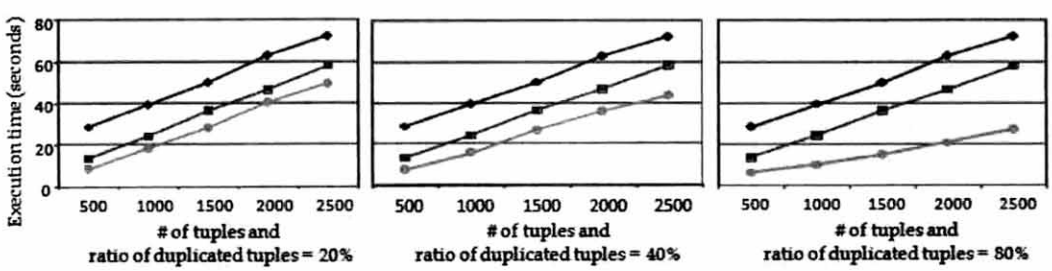
(그림 9)의 결과를 요약하면, DF-Method는 차등 파일에서 중복된 데이터의 비율을 20%로 설정할 경우, Source 방법에 비해 평균 1.7배, 최대 2.9배까지 성능을 향상시켰고, Dup-Method에 비해 평균 1.2배, 최대 1.5배까지 성능을 향상시키고, 차등 파일에서 중복 데이터의 비율을 40%로 설정할 경우, Source 방법에 비해 평균 1.9배, 최대 3.0배로, Dup-Method에 비해 평균 1.3배, 최대 1.6배로 성능을 향상시킨 것으로 나타냈다. 또한, 차등 파일에서 중복된 데이터의 비율을 80%로 설정할 경우, Source 방법에 비해 평균 2.9배, 최대 4.2배로, Dup-Method에 비해 평균 2.1배, 최대 2.5로 성능을 향상시킨 것으로 나타났다. 결과적으로, 제안한 방법은 삽입 연산에 대해 중복된 데이터의 비율이 높을 수록 Source는 물론 Dup-Method에 비해 성능을 크게 향상시킴을 알 수 있다.

실험 2) 삭제 연산에 대한 수행 시간

(그림 10)은 차등 파일을 이용한 세 가지 방법의 삭제 실험 결과를 나타낸다. (그림 10(a))는 널값 비율이 90%이고, (그림 10(b))는 95%인 경우의 실험 결과를 각각 나타낸다. 각 실험에서, 차등 파일의 중복 데이터 비율을 변경하면서 실제 수행시간을 측정하였다. (그림 10)을 보면, Source 방법과 차등 파일에서 중복 삭제를 적용하지 않는 Dup-Method가 성능이 저조함을 보이고 있다. Source 방법은 삭제되는 튜플이 직접적으로 적용되고, 이를 다시 수평 뷰로 변환하기 때문에 수행 시간이 오래 걸린다. 반면에, DF-Method와 Dup- Method는 차등 파일을 이용하여 삭제 정보를 가진 튜플을 이미 생성된 수평 뷰로 직접 적용하므로 수행 시간이 Source 방법보다 우수하다. 결과적으로 DF-Method는 차등 파일에서 중복된 데이터의 비율이 높을수록 Source 방법과 Dup- Method에 비해 성능을 향상시켰음을 나타낸다. 삭제에 대한 실험 결과를 요약하면, 차등 파일의 중복된 데이터 비율이 20%일 경우, 제안한 DF-Method는 Source에 비하여 평균 3.2배, 최대 3.3배로, Dup-Method에 비해 평균 1.2배, 최대 1.3배로 성능을 향상시킨 것으로 나타났고, 차등 파일에서 중복 데이터의 비율이 40%일 경우, Source에 비해 평균 3.6배, 최대 3.9배로, Dup-Method에 비해 평균 1.4배, 최대 1.5배로 제안한 DF-Method이 성능을 향상시킨 것으로 나타났다. 또한, 차등 파일의 중복된 데이터 비율이 80%일 경우, Source에 비

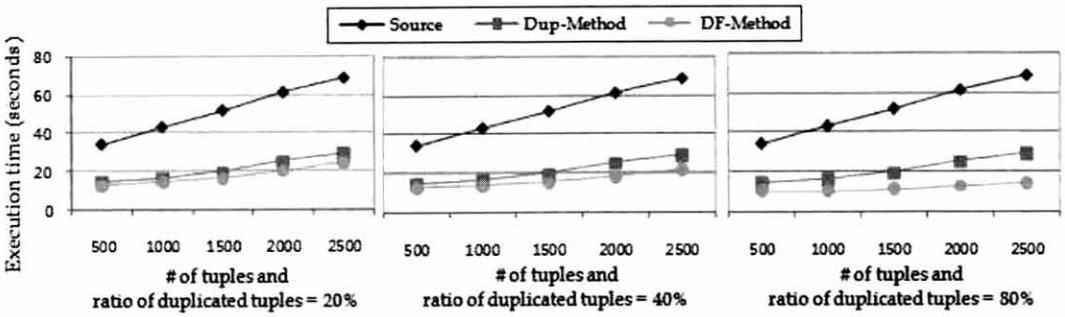


(a) Data set = 200 ×100K and ratio of null values = 90%

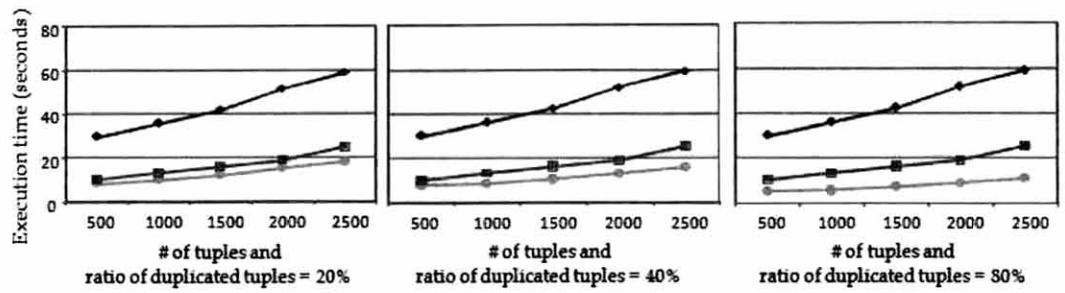


(b) Data set = 200 ×100K and ratio of null values = 95%

(그림 9) 삽입 연산에 대한 수행 시간 비교



(a) Data set = 200 ×100K and ratio of null values = 90%



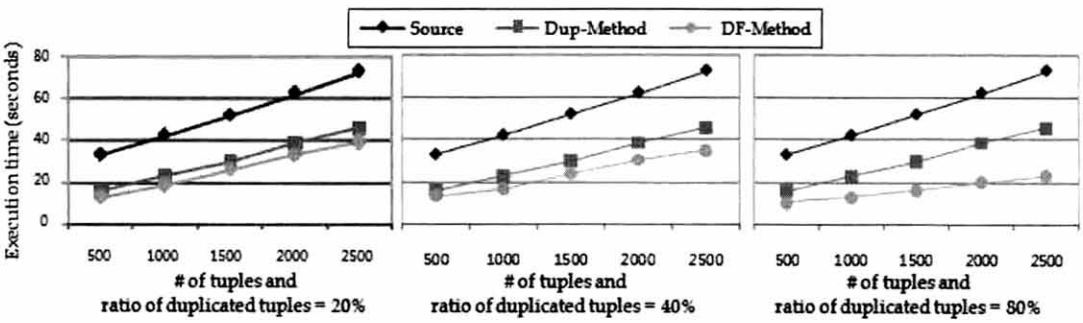
(b) Data set = 200 ×100K and ratio of null values = 95%

(그림 10) 삭제 연산에 대한 수행 시간 비교

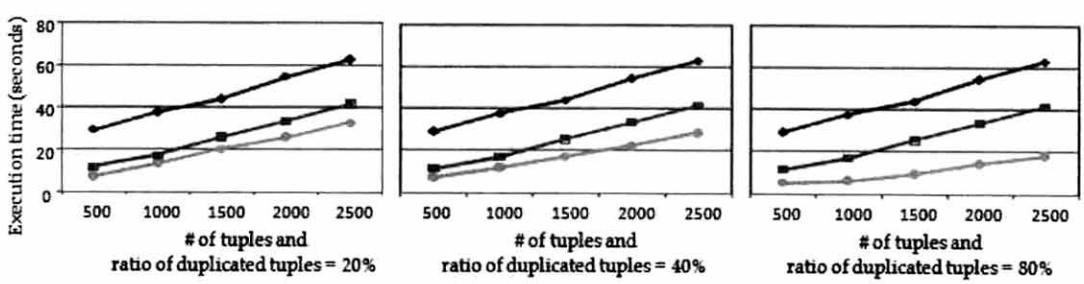
해 평균 5.0배, 최대 5.6배로, Dup-Method의 경우 평균 2.0배, 최대 2.2배로 성능이 향상시킨 것으로 나타났다. 결과적으로, 제안한 DF-Method 방법은 삭제 연산에 대하여 Source 방법은 물론 Dup-Method에 비해 성능을 크게 향상시킨 것으로 나타났다.

실험 3) 삽입 및 삭제에 대한 수행 시간

(그림 11)은 갱신 정보를 삽입 및 삭제 정보로 구성하여 세 가지 방법의 수행 시간을 측정한 결과이다. (그림 11(a))는 널 비율 값이 90%이고, (그림 11(b))는 95%인 실험 결과를 각각 나타낸다. 각 실험에서, 차등 파일의 전체 비율 중



(a) Data set = 200 ×100K and ratio of null values = 90%



(a) Data set = 200 ×100K and ratio of null values = 95%

(그림 11) 삽입과 삭제 연산에 대한 수행 시간 비교

삽입 정보와 삭제 정보를 각각 50%로 설정하여 실험한 결과이다. 그림을 보면, 제안한 DF-Method가 Source 방법과 Dup-Method에 비해 성능을 크게 향상시켰음을 알 수 있다. 이는 앞서의 삽입과 삭제의 실험 결과에서와 같이 제안한 DF-Method가 가장 좋은 성능을 보였기 때문이다. (그림 11)의 실험 결과를 요약하면, 차등 파일에서 중복 데이터의 비율이 20%일 경우, 제안한 DF-Method는 Source 방법에 비해 평균 2.2배, 최대 3.2배로, Dup-Method에 비해 평균 1.2배, 최대 1.4배로 성능이 향상시켰으며, 차등 파일에서 중복 데이터의 비율이 40%일 경우, 제안한 DF-Method는 Source 방법에 비해 평균 2.4배, 최대 3.2배로, Dup-Method에 비해 평균 1.4배, 최대 1.5배로 성능이 향상시킨 것으로 나타났다. 또한, 차등 파일에서 중복 데이터의 비율이 80%일 경우, 제안한 DF-Method는 Source 방법에 비해 평균 3.6배, 최대 4.6배로, Dup- Method에 비해 평균 2.1배, 최대 2.3배로 성능이 향상시킨 것으로 나타났다. 결국, 제안한 DF-Method는 삽입과 삭제 연산이 혼합된 환경에서도 Source 방법은 물론 Dup-Method에 비해 성능을 크게 향상 시킨다고 말할 수 있다.

이제 제안한 방법(DF-Method)과 기존의 단순한 방법(Souce)의 수행 시간 복잡도를 비교해 보자. 이를 위해 다음과 같은 표기법을 사용하기로 한다: 수직 테이블의 튜플의 수는 t_RT_i, 수평 뷰의 애트리뷰트의 수는 n_V_{RTi}, 수평 뷰의 튜플의 수는 t_V_{RTi}, 그리고 차등 파일의 튜플의 수는 t_dRT_i로 각각 표현한다. 우선, 초기에 수직 테이블 RT_i에서 수평 뷰 V_{RTi}를 생성하는 비용을 계산해 보자. 수식 (1)을 보면 수평 뷰는 수직 테이블을 n번 외부 조인하여 얻을 수 있다. 만약 외부 조인을 정렬 합병 조인(sort merge join)을 사용한다고 가정하면, 초기 수평 뷰를 생성하는 비용은 수직 테이블 RT_i를 한번 정렬한 후 ($O(t\_RT_i \log t\_RT_i)$ 소요) 이를 $n-1$번 합병($O((n\_V_{RTi}-1)*(2t\_RT_i))$ 소요)해서 구할 수 있다. 따라서 전체 수행 시간 복잡도는 아래와 같다:

$$O(t\_RT_i \log t\_RT_i + n\_V_{RTi} * t\_RT_i) \quad (10)$$

이 초기 파일에 t_dRT_i개의 튜플이 갱신됐을 경우, 단순한 방법(Source 방법)은 수평 뷰 전체를 다시 계산하는 것이다. 갱신된 수직 테이블의 튜플 수가 $t_RT_i + t_dRT_i$에 비례하므로, 이 방법의 수행 시간 복잡도는 아래와 같다:

$$O((t\_RT_i + t\_dRT_i)\log(t\_RT_i + t\_dRT_i)$$
$$\log(t\_RT_i + t\_dRT_i) + n\_V_{RT_i} * (t\_RT_i + t\_dRT_i)) \quad (11)$$

하지만, 제안된 DF-Method는 (1) 먼저 차등 파일에 있는 갱신 내용을 수평적인 형태의 튜플로 변환한 후, (2) 이 차등 수평 튜플을 수평 테이블에 반영한다. 단계 (1)은 초기 수평테이블 만드는 것과 동일한 과정을 거치므로 $O(t_dRT_i \log t_dRT_i + n_V_{RT_i} * t_dRT_i)$의 복잡도를 갖는다. 단계 (2)의 경우, 이진 탐색(또는 색인)을 사용하여 수평 테이블의 한 튜플을 갱신하는 비용이 $O(\log t_V_{RTi})$이므로, 총 $O(t_dRT_i * \log t_V_{RTi})$이다. 따라서 DF-Method의 전체 시간 복잡도는 이 두 비용을 합친 아래 수식 (12)와 같다:

$$O(t\_dRT_i \log t\_dRT_i + n\_V_{RT_i} * t\_dRT_i + t\_dRT_i * \log t\_V_{RT_i}) \quad (12)$$

결과적으로, DF-Method (수식 (12))와 Souce 방법(수식 (11))의 비용 차이는 총 수행 비용에서 차등 파일의 튜플의 수 t_RT_i가 수직 테이블의 튜플의 수 t_dRT_i보다 작으므로, DF-Method의 수행 비용이 우수하며, 성능 개선의 정도는 수직 테이블의 튜플 수와 수평 테이블의 튜플 수에 대비하여 차등 파일의 튜플 수의 비율에 의해 결정된다. 앞의 실험 결과, DF-Method가 Source 방법보다 우수하였지만 이론적인 시간 복잡도와는 다소 차이가 있다. 그 이유는, 실험에서는 실제 갱신 비용 외에 데이터베이스 시스템의 통신 비용 및 질의 처리 오버헤드가 포함되었기 때문에 수행 복잡도와 실험 결과는 약간의 차이를 보일 수 있다.

6. 결 론

e-비지니스 환경이나 OLAP 응용에서는 다차원 데이터의 효율적인 분석을 위해 수평 형태의 뷰가 널리 사용되고 있다. 반면에, RDBMS에서는 수직적인 수직 테이블을 지원하고 있다. 이에 따라, 최근 연구에서는 수직 테이블에서 수평 뷰로 변환하여 처리하는 연구들이 진행되었다. 그러나, 이들 연구에서는 수평 뷰로 변환한 후에는 수직 테이블에서 변경된 최신의 정보를 반영하는 연구가 미흡하다. 본 연구에서는 상용 RDBMS에서 제공하는 최적화된 PIVOT 연산을 적용하고, 최신 갱신 정보를 적용하기 위해 차등 파일을 이용하여 수직 테이블의 변경 사항과 동일하게 갱신한 점진적인 방법을 제안하였다. 이를 위해, 본 연구의 공헌은 다음과 같다. 첫째, 수직 테이블의 갱신에 따라 수평 뷰를 최신의 정보로 유지하기 위한 전체적인 프레임워크를 제안하였다. 둘째, 최신 갱신 정보를 유지를 위해 차등 파일에 대한 알고리즘을 제안하였다. 셋째, 차등 파일을 이용하여 수직 테이블에 갱신된 내용을 수평 뷰에서도 동일하게 적용하기 위한 변경 연산들을 제시하였다. 넷째, 차등 파일을 이용한 전체적인 뷰 관리 알고리즘을 제안하였다. 마지막으로, 실험을 통해 제안한 방법이 다른 기존 방법에 비해 성능을 향상시켰음을 확인하였다.

이러한 결과를 볼 때, 제안한 방법은 수직 테이블의 갱신에 따라 수평 뷰를 최신 정보로 유지하는 점진적인 관리하는 데 효율적인 방법이라 사료된다. 본 연구의 향후 연구로는 복합 뷰를 관리하거나, 조인 연산이나 집계 연산을 처리하는 방법을 보이는 것이다.

참 고 문 헌

[1] Agrawal, R., Somani, A., and Xu, Y., "Storage and Querying of E-Commerce Data,"' In Proc. the 27th Int'l Conf. on Very Large Data Bases (VLDB), Roma, Italy, pp.149-158, Sept., 2001.

[2] Chaudhuri, S. and Dayal, U., "An Overview of Data Warehousing and OLAP Technology," ACM SIGMOD Record, Vol.26, No.1, pp.65-74, Mar., 1997.

[3] Chen, S. and Rundensteiner, E. A., "GPIVOT: Efficient Incremental Maintenance of Complex ROLAP Views," In Proc. the 21st Int'l Conf. on Data Engineering (ICDE), IEEE, Tokyo, Japan, pp.552-563, Apr., 2005.

[4] Colby, L., et al., "Supporting Multiple View Maintenance Polices," In Proc. of ACM SIGMOD Conf., pp.405-416, 1997.

[5] Cunningham, C., Graefe, G., and Galindo-legaria, C. A., "PIVOT and UNPIVOT: Optimization and Execution Strategies in an RDBMS," In Proc. the 30th Int'l Conf. on Very Large Data Bases (VLDB), Toronto, Canada, pp.998-1009, Aug., 2004.

[6] Elmasri, R. and Navathe, S., Fundamentals of Database Systems, 3rd Ed., Addison-Wesley, 2000.

[7] Gray, J., et al., "Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals," Data Mining and Knowledge Discovery, Vol.1, No.1, pp.29-53, Mar., 1997.

[8] Gupta, A. and Mumick, I., S., "Maintenance of Materialized Views: Problems, Techniques, and Applications," IEEE Data Engineering Bulletin, Vol.18, No.2, pp.3-18, 2005.

[9] Lakshmanan, L. V. S., Sadri, F., and Subramanian, S. N., "On Efficiently Implementing SchemaSQL on an SQL Database System," In Proc. the 25th Int'l Conf. on Very Large Data Bases (VLDB), Edinburgh, Scotland, pp.471-482, Sept., 1999.

[10] Lee. W., "On the Independence of Data Warehouse from Databases in Maintaining Join views," Lecture Notes in Computer Science, pp.86-109, 1999.

[11] Mohania, M and Kambayashi, Y., "Making Aggregate Views Self- Maintainable," Data and Knowledge Eng., Vol.32, No.1, pp.87-109, 2000.

[12] Mohania, M., Samtani, S., Roddick, J., and Kambayashi, Y., "Advances and Research Directions in Data Warehousing Technology," Australian Journal of Information Systems, Vol.7, No.1, pp.41-59, 1999.

[13] Witkowski, A., et al., "Spreadsheets in RDBMS for OLAP," In Proc. Int'l Conf. on Management of Data, ACM SIGMOD, San Diego, California, pp.52-63, June, 2003.

[14] Witkowski, A., et al., "Business Modeling Using SQL Spreadsheets," In Proc. the 29th Int'l Conf. on Very Large Data Bases (VLDB), Berlin, Germany, pp.1117-1120, Sept., 2003.

신 성 현

e-mail : shshin@agape.hanyang.ac.kr

2000년 관동대학교 컴퓨터공학과(공학사)

2002년 강원대학교 컴퓨터과학과(이학석사)

2007년 강원대학교 컴퓨터과학과(이학박사)

2000년~2007년 KAIST 첨단정보기술연구센터 연구보조원

2007년~현 재 한양대학교 BK21 사업단 정보기술분야 Post-Doc.

관심분야 : 데이터 웨어하우스, OLAP, 데이터 마이닝, 도로 네트워크 데이터베이스, XML 데이터베이스 등

김 진 호

e-mail : jhkim@kangwon.ac.kr

1982년 경북대학교 전자공학과(공학사)

1985년 KAIST 전산학과(공학석사)

1990년 KAIST 전산학과(공학박사)

1995년~1996년 미국 미시건 대학교 방문교수

2003년~2004년 미국 드렉셀 대학교 방문 교수

2006년~2008년 강원대학교 중앙교육연구전산원 원장

1990년~현 재 강원대학교 컴퓨터과학과 교수

관심분야 : 데이터 웨어하우스, 데이터 마이닝, 임베디드/실시간 데이터베이스, 유비쿼터스 센서 데이터베이스, 정보 검색, 데이터 모델링 등

문 양 세

e-mail : ysmoon@kangwon.ac.kr

1991년 KAIST 전산학과(학사)

1993년 KAIST 전산학과(공학석사)

2001년 KAIST 전자전산학과 전산학전공(공학박사)

1993년~1997년 현대전자산업(주) 통신사업본부 주임연구원

2001년~2002년 (주)현대시스콤 호처리개발실 선임연구원

2002년~2005년 (주)인프라밸리 기술연구소 기술위원(이사)

2005년~2008년 KAIST 첨단정보기술연구센터 연구원

2008년~2009년 Purdue University 방문연구원

2005년~현 재 강원대학교 컴퓨터과학전공 부교수

관심분야 : 데이터 마이닝, Knowledge Discovery, 스트림 데이터, 데이터베이스 응용, 프라이버시 & 보호, 이동/무선 통신 시스템 등

김 상 욱

e-mail : wook@hanyang.ac.kr

1989년 서울대학교 컴퓨터공학과(학사)

1991년 KAIST 전산학과(공학석사)

1994년 KAIST 전산학과(공학박사)

1991년 7월~8월 미국 Stanford University, Computer Science Department 방문연구원

1994년~1995년 KAIST 정보전자연구소 전문연구원

1999년~2000년 미국 IBM T.J. Watson Research Center, Post-Doc.

1995년~2003년 강원대학교 정보통신공학과 부교수

2003년~현 재 한양대학교 정보통신학부 교수

2009년~현 재 미국 Carnegie Mellon University, Visiting Scholar

관심분야 : 데이터베이스 시스템, 저장 시스템, 트랜잭션 관리, 데이터 마이닝, 멀티미디어 정보 검색, 공간 데이터베이스/GIS, 주기억장치 데이터베이스, 이동 객체 데이터베이스/텔레매틱스, 사회 연결망 분석, 웹 데이터 분석 등