실시간 연관규칙 탐사를 위한 능동적 후보항목 관리 모델

신 예 호[†]·류 근 호[††]

요 약

미디어의 발달과 생활 패턴의 변화를 토대로 새롭게 나타나고 있는 다양한 판매 패턴들을 분석하는데 있어 단일한 분석 방법을 적용하는 것은 효과적이지 못하다. 특히 신선 식품이나 기념일 주변에서 집중적인 매출이 발생하는 품목들은 제한된 시간 내에 판매를 최대로 해야 하는 시간적 제약을 갖는다. 그러나 기존의 연관규칙 탐사 기법은 대규모 거래 데이터베이스로부터 반복적 스캔 연산을 통해 연관규칙 탐사를 수행하기 때문에 제한된 시간 안에서 빈번히 필요로 하는 패턴을 분석하기에는 비효율적이기 때문이다. 따라서 이 논문에서는 시간 제약을 갖는 특수한 판매 패턴에 대한 실시간 연관규칙 탐사가 가능하도록 하기 위해 트리거와 저장 프로시저를 이용한 점진적 후보항목 관리 모델을 제안한다. 아울러 이 논문에서는 제안 모델의 구현 및 실험을 통해 그 성능 특성의 분석도 수행한다. 특히 이 논문에서 제안하는 방법은 이중 해쉬 기법을 이용함으로써 연산의 성능을 향상시킨다.

An Active Candidate Set Management Model for Realtime Association Rule Discovery

Ye Ho Shin[†]·Keun Ho Ryu[††]

ABSTRACT

Considering the rapid process of media's breakthrough and diverse patterns of consumption's analysis, a uniform analysis might be much rooms to be desired for interpretation of new phenomena. In special, the products happening intensive sails on around an anniversary or fresh food have the restricted marketing hours. Moreover, traditional association rule discovery algorithms might not be appropriate for analysis of sales pattern given in a specific time because existing approaches require iterative scan operation to find association rule in large scale transaction databases. In this paper, we propose an incremental candidate set management model based on twin-hashing technique to find association rule in special sales pattern using database trigger and stored procedure. We also prove performance of the proposed model through implementation and experiment.

키워드 : 능동 데이터베이스(Active Database), 능동 마이닝(Active Mining), 실시간 연관규칙 탐사 모델(Realtime Association Rule Discovery Model), 온라인 마이닝(On-Line Mining), 데이터베이스 트리거(Database Trigger)

1. 서 론

경제 규모의 증가와 소비 패턴의 변화를 토대로 최근의 유통 환경은 점차 대형화하고 있는 특성을 나타내고 있다. 특히 대형 할인점의 등장은 박리 다매에 의한 이익의 극대화라는 측면에서 공급자 및 소비자 모두에게 매우 의미있는 유통 환경으로 평가되고 있으나 이익의 극대화를 위해 정확한 판매 패턴의 분석과 재고량의 적정화를 필수적으로 요구하고 있다. 이와 같은 요구사항은 데이터 마이닝을 통해 해결할 수 있다. 특히 판매 패턴과 관련하여 주목할 만한 기법으로 연관규칙 탐사 기법을 들 수 있는데 이는 대용량의 거래 데이터베이스로부터 연관성이 강한 항목을 찾아내고 이들을 지식화 함으로써 판매전략의 수립이나 진열대 배치의 조정 등을 통해 매출의 극대화를 꾀할 수 있도록 하기 때문이다[1].

그러나 대형 유통 환경에서 나타나는 다양한 현상들은 일률적으로 규정할 수 없는 문제가 있다. 예를 들어 생선, 과일, 육류 및 야채류와 같은 신선 식품들의 경우 유통 기간이 매우 짧고 계절적 환경에 민감한 특성을 갖고 있다. 마찬가지로 기념일과 관련된 제품들의 경우 해당 기념일 주변에서 집중적인 판매가 발생하는 특성을 갖고 있다. 이와 같이 매우 제한적이고 특수한 매출 형태를 나타내고 있는 품목들에 대해 일괄적으로 연관규칙 탐사 기법을 적용할 경우 왜곡된 판매 패턴을 생성해 낼 수 있으며 이는 결국 판매 전략을 수립하는데 있어 부적절한 영향을 끼치는 문제를 발생시킨다.

이와 같은 문제 즉 시간적 제약에 의해 발생하는 제한적이고 특수한 판매 형태를 나타내는 품목들에 대한 판매 패턴의

※ 이 논문은 한국과학재단 목적기초연구(R01-1999-00243)지원으로 수행되었음.
† 정 회 원 : 극동대학교 정보통신학부 전임강사
†† 종신회원 : 충북대학교 전기전자및컴퓨터공학부 교수
논문접수 : 2001년 10월 8일, 심사완료 : 2001년 11월 8일

분석을 효과적으로 수행하기 위해서는 일반적인 연관규칙 탐사가 아닌 시간적 특수성을 흡수할 수 있는 실시간 연관규칙 탐사 기법이 필요하다. 왜냐하면 시간적 제약 조건을 갖는 특수한 판매 형태들은 일반적으로 매우 제한적인 시간 내에 매출을 극대화해야 하며 해당 시기가 지나게 되면 상품가치가 급격히 저하되는 특성을 갖기 때문에 필요한 시점에 최대한 신속히 패턴을 분석해 낼 수 있어야 하기 때문이다. 그러나 기존의 연관규칙 탐사 기법들은 특정 시점의 데이터베이스 전체를 대상으로 연관규칙 탐사를 수행함으로써 연관규칙 탐사에 의한 연산 부하가 대단히 높고 매우 많은 연산 시간을 필요로 한다[2]. 또한 기존의 방법들은 연관규칙의 탐사를 시작한 시점 이후에 발생한 데이터들은 전혀 연관규칙 탐사 결과에 반영할 수 없기 때문에 트랜잭션 발생 주기가 매우 짧은 동적 환경 하에서 시간적 제약을 갖고 있음으로 해서 수시로 판매 패턴에 대한 분석을 필요로 하는 형태의 품목에 대해서는 적용이 불가능한 문제를 갖고 있다. 이와 같은 문제들을 해결하기 위해 이 논문에서는 트랜잭션 데이터의 삽입과 동시에 연관규칙의 탐사가 가능하도록 하기 위해 데이터베이스 트리거(database trigger)와 이중 해쉬 기법을 이용한 실시간 점진적 갱신기법(realtime incremental updating technique)을 수용한 능동적 후보항목 관리 모델을 제시하고 이의 구현 및 실험을 통해 제안 모델의 특성을 분석한다.

이를 위한 논문의 구성은 다음과 같다. 먼저 2장에서 이 논문의 연구 기반이 되는 온라인 연관규칙 탐사기법 및 점진적 갱신을 기반으로 한 마이닝 기법 그리고 트리거 및 트리거를 이용한 마이닝 기법에 대한 기존 연구를 고찰한다. 다음으로 3장에서는 트리거와 점진적 갱신 기법을 결합한 능동적 후보항목 갱신 모델을 제시하고 4장에서 이의 동작 시나리오를 기술한다. 그리고 5장에서 구현 모델 및 구현 알고리즘에 대해 기술하고 6장에서 실험 및 성능 평가를 수행한 후 7장에서 결론 및 향후 연구방향을 제시한다.

2. 관련 연구

온라인 연관규칙 탐사모델에는 접근 방식에 따라 몇 가지 형태로 분류할 수 있다. 이와 같은 형태 중 첫 번째로 고려할 수 있는 방식이 질의어 수정에 의한 방식인데 이는 질의어의 연산을 규정하는 대수(algebra)를 연관규칙 탐사에 적합하게 확장하는 방식으로서 기존 데이터베이스 시스템에 대한 수정을 거의 하지 않고도 연관규칙을 탐사할 수 있도록 하는 방법이다[3]. 두 번째 유형으로는 기존 질의어와는 별도로 마이닝 연산을 수행할 수 있는 마이닝 질의어를 정의하고 이를 기존 시스템에 통합시키는 방법이다. [4]에서 제기한 이 방법은 연관규칙을 비롯한 여섯 가지 마이닝 지식을 탐사할 수 있는 마이닝 질의어를 정의함으로써 별도의 프로세스없이 이 마이닝 질의어만으로 마이닝 지식을 탐사할 수 있도록 하였다. 또 다른 방법으로 [2]에서 제시한 저장 프로시져를 이용한 연관규칙 탐사 기법을 들 수 있다. 이는 온라인 연관규칙 탐사 과정에서 데이터베이스 시스템 프로세스와 연관규칙 탐사 프로세스 사이에 발생하는 과도한 데이터 전송 비용을 상쇄하기 위한 방안으로 제시되었는데 실험 결과 저장 프로시져를 이용하는 방법이 호스트언어를 이용하는 방법보다 세배 정도의 성능향상을 가져 왔음을 보고하고 있다.

연관규칙 탐사 모델에서 성능 향상을 위한 연구도 다수 존재한다. 앞에서 설명한 [2]의 저장 프로시져를 이용한 방법 역시 주요한 이슈는 연관규칙 탐사 연산의 성능 향상이 주요한 초점이 되고 있다. 성능 향상에 있어 [2]가 구현 기법에 초점을 맞춘 방안이라면 알고리즘 자체를 개선하여 성능 향상을 추구하는 연구들로는 [5, 6] 등의 연구를 들 수 있다. [5]는 해쉬 기법을 이용하여 성능 향상을 꾀하고 있으며 [6]은 연관규칙 탐사를 점진적으로 수행함으로써 전체적인 연산 부하를 감소시키려는 방법이다.

데이터베이스에 마이닝 기능을 통합하려는 또 다른 시도로 마이닝과 트리거를 결합시키려는 연구가 있다. [7] 및 [8]에서 제시한 이 방법들은 연관규칙 탐사 프로세스에 의해 탐사된 연관규칙을 관리하기 위한 지식베이스 운영 과정에 트리거를 이용함으로써 마이닝 지식베이스의 관리를 지능화하기 위한 모델을 제시하고 있다. [7] 및 [8]에서 마이닝 지식의 관리 과정에 적용한 트리거는 데이터베이스의 상태 변화에 대한 자동 대응 기능으로서 80년대 초반 이후 데이터베이스 분야에서 광범위한 연구가 진행되었다[9, 10]. 이 트리거는 데이터베이스에 대한 수정연산에 대응해서 미리 정의된 적절한 조건을 충족시킬 경우 이에 대한 조치를 데이터베이스 스스로 취할 수 있게 하는 조건부 조치기능을 갖는 것으로서[11-13] 이미 SQL3 표준에 기본기능으로 채택될 만큼 데이터베이스 분야에서 대중적 기술로 인식되고 있다[14]. 이와 같은 데이터베이스 트리거의 자동 대응기능은 시스템 수준에서는 제공하기 힘든 다양한 무결성 문제는 물론 보안 수준의 강화 등에 효과적으로 적용할 수 있다[12, 15].

그러나 온라인 마이닝 또는 데이터베이스 시스템에 마이닝 기능을 통합하기 위한 이와 같은 방법들에는 일정한 문제점들을 내포하고 있다. 예를 들어 [6]에서 제시한 점진적 연관규칙 탐사 기법은 부분 연산을 수행하기 위한 차분 데이터베이스가 오차를 희석시킬 수 있을 만큼 충분히 커야 하며 따라서 차분 데이터베이스가 일정량 이상 축적될 때까지는 점진적 연산의 의미가 불충분해 진다는 문제를 갖고 있으며 [5]에서 제시한 방법은 여전히 일시에 집중적인 데이터베이스 검색 연산이 발생한다는 문제를 내포하고 있다. 뿐만 아니라 [4]의 방법은 마이닝 기능을 데이터베이스 시스템에 통합하기 위해 데이터베이스 엔진에 대한 수정을 요구하고 있다. 아울러 [7, 8]에서 제시한 능동적 마이닝 모델 역시 트리거를 연관규칙 탐사 과정에서 적용하지 못하고 단지 탐사된 지식의 관

리에만 적용함으로써 완전한 능동적 연관규칙 탐사 모델을 제시하지 못하고 있다.

한편 [16]에서는 이 논문에서 제시하고 있는 구조와 유사하게 트리거와 저장 프로시져를 이용한 실시간 점진적 후보 항목 갱신 모델에 대한 연구 결과를 제시하고 있다. 그러나 [16]에서는 항목들 사이의 충돌 해결을 위한 방법으로 단일 해쉬를 사용함으로써 과도한 디스크 입출력이 발생하여 연산 성능이 불규칙적인 문제를 나타낸다.

이와 같이 실시간 연관규칙 탐사와 관련된 온라인 연관규칙 탐사 모델 및 데이터베이스 시스템과 마이닝 기능의 통합 모델에서 갖는 문제점들을 해결하기 위해 이 논문에서는 트리거를 연관규칙 탐사 단계에 적용하고 연관규칙 탐사를 기술하는 트리거의 조치 절을 저장 프로시져로 구현함으로서 데이터베이스 시스템에 대한 수정 없이 실시간 연관규칙 탐사를 수행할 수 있는 실시간 연관규칙 탐사 모델을 제안한다. 특히 이 논문에서 제안하는 점진적 후보 항목 관리 모델은 [16]에서 제안하고 있는 단일 해쉬 방법과 달리 이중 해쉬 기법을 이용함으로써 전체적인 성능의 안정성을 확보할 수 있도록 하였다. 아울러 이 논문에서는 제안 모델의 구현 및 실험을 통해 제안 모델의 타당성을 검증한다.

3. 후보항목 및 빈발계수의 점진적 갱신 모델

연관규칙 탐사에 대한 연산비용 문제에 있어 가장 핵심적 부분은 후보 항목의 생성 부분이다[17, 18]. 이는 후보 항목 생성을 위해 데이터베이스 스캔을 필요로 하기 때문이다. 따라서 기존의 방법들에서는 후보 항목 생성 비용을 줄이기 위해 빈발항목으로부터 후보 항목 생성 과정에서 가능한 적은 수의 후보 항목을 생성하도록 하는 방법을 사용하고 있다. 그러나 이 논문에서와 같이 실시간 온라인 연관규칙 탐사를 위해서는 기존의 방법들을 사용할 수 없다. 왜냐하면 기존의 방법들에서는 실시간 탐사를 고려하지 않고 있으며 따라서 특정 시점의 전체 데이터베이스를 대상으로 연관규칙 탐사를 수행하고 있을 뿐 끊임없이 변화하는 동적 환경 하에서 변경된 상태를 즉시 연관규칙 탐사 과정에 반영할 수 있는 방안을 전혀 고려하고 있지 않기 때문이다. 따라서 이와 같은 문제를 해결하기 위해 이 논문에서는 새로운 트랜잭션 데이터의 삽입 즉시 새로 삽입된 데이터로부터 후보 항목을 생성하고 이를 연관규칙 탐사 과정에 반영할 수 있는 점진적 후보 항목 갱신 모델을 기술한다.

점진적 후보항목 갱신 모델의 핵심이 되는 점진적 연산은 시간의 흐름에 따라 후보항목에 나타나는 차분(difference)을 포착해서 이를 후보항목에 반영하는 연산이다. 이와 같은 점진적 후보항목 갱신 연산은 시간의 흐름에 따라 나타나는 차분 연산을 수행하기 위해 시점 인덱스를 포함해야 한다. 다음의 <표 1>은 이와 같은 점진적 연산을 수행하는데 있어 적용되는 파라메터들을 요약한 것이다.

〈표 1〉 점진적 후보항목 연산을 위한 파라메터

기 호	기호 의미
t	점진적 후보항목 연산 수행 시점을 나타내는 시점 인덱스
$Card(Tr)$	트랜잭션 내에 포함되어 있는 단일항목의 수
${}_tMaxTr$	$0 \sim t$ 시점 사이에 발생한 트랜잭션 중 $Card(Tr)$ 값이 가장 큰 트랜잭션
k	연산대상 후보 항목 집합의 항목 크기(항목을 구성하는 단일항목의 수) $1 \le k \le Card(Tr)$
i_l^k	k 후보항목 집합 내에 있는 l 번째 원소항목
$freq\_count$	$0 \sim t$ 시점사이에 항목이 발생한 빈도수
l	항목 집합 내의 항목 인덱스
I^k	k 크기 항목들의 집합
z	항목집합 내에 최대로 포함될 수 있는 항목의 수

다음의 정의 1은 <표 1>에서 설명하고 있는 기호들을 이용하여 점진적 후보항목관리 연산에 참여하는 후보항목의 의미를 정의한 것이다.

[정의 1] 시점 인덱스 t를 포함하는 k 크기 후보 항목 집합 ${}_tI^{k+1}$

t 시점에서 k크기의 후보항목 집합은 t 시점에 발생한 트랜잭션으로부터 조합 가능한 모든 항목들의 집합으로 정의하며 다음과 같이 표기한다.

$$t \text{ 시점에서의 } k\text{\_후보항목집합} = {}_tI^{k+1} =$$
$$\{({}_ti_l^k, freq\_count_l) \mid \forall o,p, 1 \le o,p \le z, o \ne p \rightarrow {}_ti_o^k \ne {}_ti_p^k$$
$$\wedge 1 \le l \le z \wedge z = {}_{Card({}_tMaxTr)}C_k\}$$

여기서 항목집합에 대한 시점인덱스 t를 항목집합 기호 I의 앞쪽 아래 첨자로 표기하고 항목이 포함하는 단일항목의 수인 항목 크기 k는 항목집합 기호의 윗첨자로 표기하였다. 아울러 변수 z는 해당 크기의 후보 항목집합이 갖을 수 있는 최대 원소 수로서 이는 t 시점에 전체 트랜잭션 데이터베이스에서 단일 항목을 가장 많이 갖고 있는 ${}_tMaxTr$로부터 k개의 항목을 추출하여 조합할 수 있는 경우의 수 즉 ${}_{Card({}_tMaxTr)}C_k$를 넘지 못한다.

한편 전체 후보항목 집합은 위 정의 1에서 정의한 k크기의 후보 항목 집합들의 합집합으로 정의되는데 여기서 항목집합의 항목 크기를 나타내는 k는 최소 크기 1에서 최대 크기 $Card({}_tMaxTr)$를 넘지 못한다. 이와 같은 사항을 반영한 전체 후보항목집합에 대한 정의를 다음의 보조정의 1.1에서 하였다.

[보조정의 1.1] t 시점에서 점진적 후보항목 연산을 위한 후보항목 전체집합 ${}_tCand\_I$

t 시점에서 점진적 후보항목 연산을 위한 후보항목의 전체

집합은 k 크기 항목 집합의 합집합으로 정의하며 그 의미는 다음과 같다.

$$\text{후보항목집합 } {}_tCand\_I = \{ \bigcup_k {}_tI^{k+1} \mid 1 \le k \le Card({}_tMaxTr) \}$$

앞의 [정의 1] 및 [보조정의 1.1]에서 점진적 연산과정에 중요하게 참여하는 후보항목집합에 대한 정의를 내렸다. [정의 1] 및 [보조정의 1.1]의 정의는 전체 후보항목 집합 관점에서 내린 정의이다. 그러나 점진적 연산은 차분을 이용한 연산을 수행해야 하며 이 차분은 새롭게 발생한 트랜잭션에 의해 생성되는 항목들을 이용하여 생성된다. 따라서 차분 집합에 대한 정의를 추가로 내릴 필요가 있다. 이를 반영한 정의가 다음의 [정의 2]이다.

[정의 2] t 시점에서의 갱신 차분 ${}_t\Delta_U$, 삽입 차분 ${}_t\Delta_A$ 및 차분 집합 ${}_t\Delta$

새로운 트랜잭션의 발생에 의해 생성되는 차분 집합은 새로 발생한 트랜잭션의 단일 항목들을 이용하여 조합 가능한 모든 항목들의 집합으로 정의하며 다음과 같은 의미를 갖는다.

$$\text{차분집합 } {}_t\Delta = \{ {}_ti_l^k \mid \forall o, p, 1 \le o, p \le {}_{Card({}_tTr)}C_k,\ o \ne p \Rightarrow {}_ti_o^k \ne {}_ti_p^k \wedge 1 \le k \le Card({}_tTr) \}$$

$$\text{빈발계수 갱신 차분 } {}_t\Delta_U = {}_t\Delta \cap {}_{(t-1)}Cand\_I$$

$$\text{빈발계수 삽입 차분 } {}_t\Delta_A = {}_t\Delta - {}_t\Delta_U$$

[정의 2]가 [정의 1] 및 [보조정의 1.1]과 다른 점은 [정의 1] 및 [보조정의 1.1]이 전체 트랜잭션 데이터베이스를 대상으로 후보항목의 의미를 규정하는 반면 [정의 2]는 t 시점에 발생한 트랜잭션에 한정해서 후보항목의 의미를 정의한다는 점이다. [정의 2]에서 차분 집합은 t 시점에 발생한 트랜잭션으로부터 생성할 수 있는 모든 항목들의 집합으로 정의하고 있으며 갱신 차분은 전체 차분 집합에서 $t-1$ 시점까지의 후보항목과 교집합을 이루는 부분으로 한정하고 있다. 아울러 삽입 차분은 전체 차분집합에서 갱신차분을 제외한 부분을 의미함도 알 수 있다. 이와 같은 차분 의미를 토대로 점진적 후보항목 갱신 연산을 정의하면 다음의 [정의 3]과 같다.

[정의 3] 후보 항목의 점진적 갱신

후보 항목의 점진적 갱신은 이전 시점의 후보항목 집합에 새로 삽입된 트랜잭션 데이터로부터 생성되는 후보 항목 집합의 차분을 합산하는 것으로 정의하며 다음과 같이 표현한다.

$$ {}_tCand\_I = {}_{(t-1)}Cand\_I \oplus {}_t\Delta$$

여기서 차분 연산자 $\oplus$가 의미하는 바는 차분을 구성하는 입력 차분과 갱신 차분을 모두 직전 시점의 후보 항목 집합에 합산한다는 의미를 내포한다. 따라서 차분 연산자 $\oplus$는 갱신 차분에 의해 발생하는 이전 시점의 후보 항목들에 대한 빈발 계수 갱신 연산과 삽입 차분에 의해 발생하는 빈발계수 1인 후보 항목들의 삽입 연산을 모두 포함한다.

이와 같이 차분 연산을 이용한 후보 항목의 점진적 갱신 연산은 단지 새로 삽입되는 트랜잭션 데이터에 대해서만 연산을 수행하고 이를 이전 시점까지 유지하고 있던 후보 항목 집합에 합산하는 연산만을 수행하면 된다. 따라서 후보 항목 집합에 대한 점진적 갱신 연산은 후보 항목 집합의 빈발계수를 계산하기 위해 전체 데이터베이스를 스캔해야 하는 기존의 방법에 비해 동적 환경 하에서 데이터베이스 상태변화에 대응하여 즉각적으로 변경된 상태를 후보 항목 집합에 반영하는데 있어 매우 높은 효율성을 갖을 수 있다. 또한 점진적 갱신 연산에 의해 유지되는 후보 항목 집합의 상태는 동적 환경에 대응해서 항상 최신의 상태를 반영할 수 있는 빈발항목 추출이 가능하도록 완전한 후보 항목집합을 유지한다.

4. 점진적 후보 항목 갱신 시나리오

이 장에서는 3장에서 기술한 후보항목의 점진적 갱신 모델에 따라 점진적 후보 항목의 갱신을 위한 시나리오와 그 절차를 기술한다. 트리거를 이용한 점진적 후보 항목 갱신 기법의 주요한 특성은 트리거를 이용해 새로운 트랜잭션의 발생 즉시 이를 후보 항목에 반영할 수 있다는 점과 전체 데이터베이스에 대한 스캔 없이 단지 새로 발생한 트랜잭션에 대응한 연산만으로 전체 데이터베이스를 스캔한 것과 동일한 효과를 얻을 수 있다는 점이다. 또한 트리거에 의해 촉발되는 점진적 후보 항목 갱신 프로시저는 데이터베이스 시스템 주소공간에서 수행되는 저장프로시저를 이용하기 때문에 보다 높은 성능 특성을 얻을 수 있다. 이 트리거를 이용한 점진적 후보 항목 갱신 기법의 수행 절차를 요약하면 다음과 같다.

단계 1. 새로운 트랜잭션의 발생과 발생된 트랜잭션 데이터의 데이터베이스 삽입 연산 수행

단계 2. 트랜잭션 데이터의 삽입 연산에 의한 후보 항목 관리 트리거의 사건 발생

단계 3. 사건 발생에 따라 기동되는 후보항목 관리트리거의 조치절에 의한 점진적 연산프로시저 호출

  단계 3.1 삽입된 트랜잭션 데이터를 이용하여 조합 가능한 후보 항목 생성

  단계 3.2 생성된 후보 항목들을 삽입 차분과 갱신 차분으로 분류

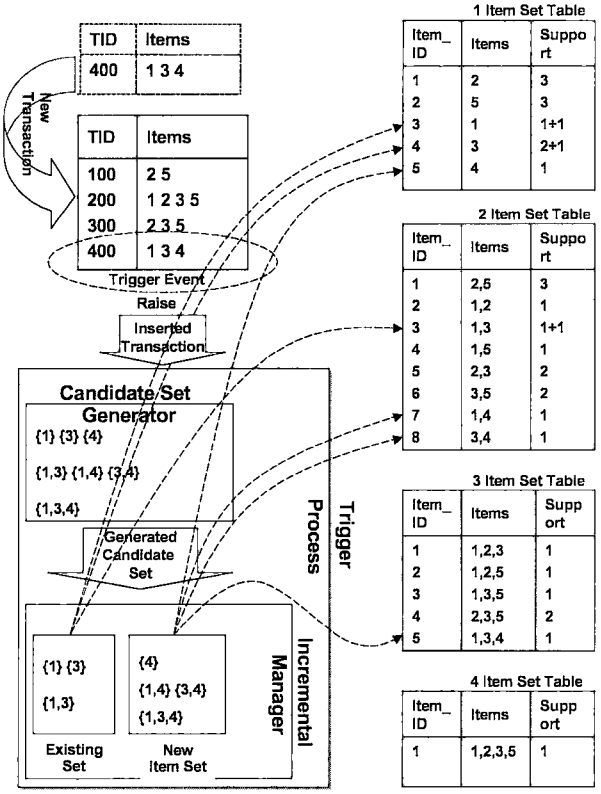
  단계 3.3 삽입 차분의 삽입 연산 수행

  단계 3.4 갱신 차분의 갱신 연산 수행

단계 4 트랜잭션 커밋

여기서 단계 3과 그 하위단계들이 트리거에 의해 컨트롤되는 영역으로서 실제 트리거 자체는 새로운 트랜잭션의 삽입에 따른 사건 발생과 트리거 조치에 의한 후보 항목 관리 프

로시저를 호출하는 것으로 한정되지만 프로시저 자체가 트리거의 컨트롤 범위 내에 있기 때문에 전 과정이 트리거에 의해 컨트롤되는 것으로 파악할 수 있다. 트리거에 의해 컨트롤되는 과정을 다음의 (그림 1)에 도시하였다.



(그림 1) 트리거를 이용한 후보항목집합의 점진적 관리 시나리오

(그림 1)에서 볼 수 있는바와 같이 새로운 트랜잭션의 삽입이 발생하게 될 경우 트리거 사건이 발생하게 된다. 트리거 사건에 의해 기동되는 트리거는 먼저 후보항목집합 생성기(candidate set generator)에 의해 조합 가능한 모든 후보 항목들을 생성하고 생성된 후보 항목들을 점진적 후보항목 관리기(incremental manager)에 의해 지지도 갱신 가능 항목과 새로 추가될 후보 항목으로 분류한 다음 이들을 일괄적으로 후보항목집합 데이터베이스에 반영한다.

이와 같은 연산은 모두 한 트랜잭션의 삽입에 의해 발생되며 이 연산이 완료되었을 때 삽입 트랜잭션의 종료가 완료된다. 그러므로 모든 후보 항목 집합에 대한 연산은 트랜잭션 단위로 발생하게 되고 새로운 트랜잭션이 발생할 때마다 발생된 트랜잭션의 상태를 이미 유지하고 있는 후보 항목 집합 테이블에 반영하는 연산을 수행한다. 따라서 특정 시점의 전체 데이터베이스를 대상으로 하는 기존의 연관규칙 탐사 기법과는 달리 제안하는 방법은 트랜잭션 단위로 후보 항목에 대한 상태 변화를 관리하는 점진적으로 후보 항목을 관리하는 기법이다. 이 때 트랜잭션 단위의 점진적 연산을 수행할 수 있게 하는 핵심적 요소는 바로 트리거에 의해 후보 항목의 점진적 갱신 연산을 기동(invoke) 시키는 것이다.

한편 단계 3의 트리거와 저장 프로시저를 이용한 점진적 후보항목 갱신 연산에서 가장 중요한 부분은 새로운 트랜잭션에 의해 생성되는 후보 항목들을 삽입 차분과 갱신 차분으로 분류하는 단계 3.2이다. 왜냐하면 새로 삽입된 트랜잭션 데이터를 이용하여 생성된 후보 항목들을 삽입 차분과 갱신 차분으로 분류하기 위해서는 이미 생성되어 관리 유지되고 있는 기존의 차분들과의 비교연산이 필수적이기 때문이다. 여기서 주목할 점은 점진적 연산을 위해 모든 가능한 후보 항목들을 유지해야 한다는 점인데 이렇게 할 경우 빈발 빈도가 매우 적은 후보 항목들까지 지속적으로 유지해야 하며 이는 불필요한 저장 공간의 낭비 및 연산 시간의 과도한 부하를 유발시킬 수 있다. 이와 같은 문제를 해결하기 위해 일정정도 이상의 빈발계수를 갖는 후보 항목들을 주메모리상에 유지하면서 해쉬 기법과 결합하여 데이터베이스에 대한 접근을 최소화할 수 있도록 하는 전략을 채택한다.

5. 점진적 갱신 연산을 위한 구현 모델

3장에서 후보 항목 집합에 대한 점진적 갱신 모델을 기술하고 4장에서 모델의 수행 시나리오에 대해 기술하였다. 이 장에서는 이 점진적 갱신 모델을 수행 시나리오에 따라 동작하도록 하기 위한 구현 모델을 기술한다. 실제로 이 장에서 기술하는 구현 모델은 오라클 8에 통합되어 있는 트리거와 저장 프로시저 언어인 PL/SQL[19]을 이용하여 구현하고 있으며 따라서 구현 모델 역시 이들 도구를 기반으로 한다.

5.1 연관규칙 탐사 트리거의 정의

이 절에서는 오라클 8에 통합되어 있는 트리거를 이용하여 연관규칙 탐사를 위한 트리거를 정의한다. 오라클 8의 트리거는 SQL3의 트리거 모델을 따르고 있으나 몇 가지 특성들은 성능 측면에서 배제하고 있다. 예를 들어 트리거의 재귀적 호출을 지원하지 않음으로써 트리거의 연쇄에 의한 트리거 비 종료 문제를 원천적으로 차단하고 있다. 그러나 복합 사건(composite event)의 정의, 트리거 수행 시점(trigger action time), 조건절(condition clause), 트리거 실행단위(trigger execution granularity) 등에 대해서는 충실히 지원하고 있다. 특히 조치로서 PL/SQL 블록은 물론 외부 프로시저를 호출할 수 있도록 허용함으로써 매우 유연한 트리거 수행 모델을 제공한다[19].

이와 같은 오라클 트리거를 이용하여 연관규칙 탐사를 위해서는 두 개의 트리거가 쌍으로 정의되어야 한다. 왜냐하면 실제 데이터베이스 삽입 연산은 행(row) 단위로 이루어지지만 연관규칙 탐사는 트랜잭션 단위로 수행되어야 하기 때문에 트리거의 적용에 있어 실행 단위가 달라지기 때문이다. 따

라서 행 단위 트리거를 이용하여 트랜잭션 내에서 데이터베이스에 삽입되는 행들을 수집한다. 그런 다음 트랜잭션 단위 트리거에서는 행 단위 트리거에서 준비된 데이터를 파라메터로 하여 점진적 후보 항목 갱신을 위한 연산을 호출한다. 다음의 (그림 2)는 행 단위로 삽입되는 데이터를 수집하기 위한 트리거 정의를 기술한다.

```
CREATE TRIGGER Catch_Trigger
IS
BEFORE INSERT ON Tran_DB
REFERENCING NEW as New_TR
BEGIN
    Mine.ADD_New_Item( : New_TR)
END ;
```

(그림 2) 행 단위 삽입 데이터 수집 트리거

다음으로 후보 항목의 점진적 갱신을 수행하는 저장 프로시저를 호출하는 트랜잭션 단위 트리거의 정의는 다음의 (그림 3)과 같다.

```
CREATE TRIGGER Discovery_Association_Rule
IS
AFTER INSERT ON Tran_DB
BEGIN
    Mine.Start_Package ;
END ;
```

(그림 3) 트랜잭션 단위 트리거 정의

여기서 두 트리거의 사건 부분에 집중할 필요가 있다. 행 단위 트리거가 트리거 수행 시점으로 BEFORE로 한 반면 트랜잭션 단위 트리거는 수행 시점으로 AFTER를 지정하고 있다. 이는 점진적 생신을 수행하기 이전에 데이터베이스에 삽입되는 모든 행들을 수집하기 위해 행 단위 트리거가 트랜잭션 단위 트리거보다 높은 우선순위를 갖도록 지정하기 위해 설정한 것이다. 이렇게 할 경우 행 단위 트리거는 데이터베이스에 행을 삽입하기 전에 트리거 연산을 먼저 수행하고 난 다음에 데이터베이스 연산을 수행한다. 반면 트랜잭션 단위 트리거는 데이터베이스 연산이 모두 끝나고 난 다음에 트리거 연산을 수행한다.

5.2 후보항목 생성 알고리즘

이 절에서는 4장의 시나리오에서 후보항목 생성기 부분에 대한 구현 알고리즘을 기술한다. 후보 항목 집합 생성 연산은 삽입된 트랜잭션으로부터 조합 가능한 모든 항목들을 생성하는 연산으로서 트랜잭션이 n개의 단일 항목으로 구성되어 있다면 생성되는 후보 항목은 집합론에 의거해서 2^n-1개의 항목으로 조합된다. 다시 말해 한 트랜잭션 내에서 생성 가능한 후보 항목의 수는 해당 트랜잭션 내 단일 항목의 수에 따라 지수적으로 증가한다. 이 때 이 논문에서 제안하는 트리거와 저장 프로시저를 이용한 점진적 후보항목 관리기법은 트랜잭션 발생 즉시 트랜잭션 내에서 후보 항목에 대한 모든 연산을 수행해야 하므로 트랜잭션 내에서 처리해야할 후보 항목의 수가 지나치게 많을 경우 성능에 심각한 영향을 미칠 수 있다. 따라서 이와 같은 문제를 해결하기 위해 이 논문에서는 트랜잭션 내 최대 항목 수를 12개 이하로 제한한다. 이는 이 연구에서 적용 대상으로 하고 있는 항목들이 시간적 제약 조건을 갖는 특수한 항목들로 제한하고 있으며 이들 특수 항목의 경우 품목의 수가 많지 않기 때문이다. 뿐만 아니라 트랜잭션 사이즈가 지나치게 클 경우 트랜잭션에 의해 생성 가능한 후보 항목의 수가 너무 많아져 메모리 요구량 및 전체 시스템 성능 측면에서 많은 부하가 가중될 수 있게 된다.

```
procedure Make_Candidate_Item_set( New_tr_buf, tr_size )
begin
 bnd_1 : = tr_size - 1 ;
 for cnt_1 in 1..bnd_1 loop
   for cnt_2 in cnt_1+1..tr_size loop
     jmp_2 : = jmp_2 + 1 ;
     buf 2_1(jmp_2) : = New_tr_buf(cnt_1) ;
     buf 2_2(jmp_2) : = New_tr_buf(cnt_2) ;
     for cnt_3 in cnt_2+1..tr_size loop
         --
         for cnt_7 in cnt_6+1..tr_size loop
           jmp_7 : = jmp_7 + 1 ;
           buf 7_1(jmp_7) : = New_tr_buf(cnt_1) ;
               --
           buf 7_7(jmp_7) : = New_tr_buf(cnt_7) ;
           for cnt_8 in cnt_7+1..tr_size loop
             jmp_8 : = jmp_8 + 1 ;
             buf 8_7(jmp_7) : = jmp_7 ;
             buf 8_8(jmp_8) : = New_tr_buf(cnt_8) ;
             for cnt_9 in cnt_8+1..tr_size loop
                 --
               for cnt_12 in cnt_11+1..tr_size loop
                   --
               end loop ;                    -- 12
             --
             end loop ;                      -- 09
           end loop ;                        -- 08
         end loop ;                          -- 07
     --
   end loop ;                                -- 03
  end loop ;                                 -- 02
 end loop ;                                  -- 01
end Make_Candidate_Item_set ;
```

(그림 4) 트랜잭션 단위 후보 항목 생성 알고리즘

또한 동일 트랜잭션 내에 포함된 단일 항목들을 이용하여 조합되는 후보 항목들은 많은 중복(redundancy) 현상이 발생한다. 이와 같은 현상은 트랜잭션 사이즈가 커질수록 그 정도가 심해진다. 예를 들어 a, b, c, d, e 다섯 개의 항목을 갖는 트랜잭션에 대해 이를 이용한 후보 항목을 조합할 경우 31개의 후보 항목들을 생성할 수 있으며 이 중 {a,b} 항목이 8번 나타난다. 이것은 거의 전체 항목에 대해 25%정도의 빈도율을 갖는 것으로서 이를 모두 실제 항목에 포함시킬 경우

메모리 낭비는 물론 조합 과정에서의 연산 부하를 초래하게 된다. 이와 같은 문제를 해결하기 위해 이 논문에서는 알고리즘 적용 최대 트랜잭션 크기의 중간인 8을 기점으로 8보다 큰 트랜잭션에 의해 생성되는 후보 항목들은 8 이하에서 조합된 항목들을 그대로 사용하는 기법을 적용한다. 아울러 트랜잭션 내의 단일 항목들을 조합하는 기법으로 중첩 순환(nested loop) 기법을 이용한다. 이와 같은 과정을 정리하면 다음과 같다.

(그림 4)에서 후보항목을 저장하기 위한 버퍼 변수들은 모두 1차원 배열로서 이는 이 논문에서 구현 플랫폼으로 이용하고 있는 오라클 8의 PL/SQL에서 제공하는 복합 데이터 타입이 다차원 배열을 허용하지 않기 때문에 취한 것이다. 그러나 후보항목 집합은 2차원 데이터 구조를 필요로 한다. 따라서 1차원 구조를 이용한 2차원 구조 모사(simulation) 기법이 필요한데 이 논문에서는 이를 1차원 배열을 필요한 수만큼 병렬로 결합시키는 기법을 이용하였다. 아울러 항목 크기 8이상의 후보항목에 대해서 항목의 일곱 번째 까지 항목 요소는 동일한 내용을 포함하고 있는 7 크기의 후보항목에 대한 포인터만을 저장하고 실제의 항목 내용은 생성하지 않도록 하였다. (그림 4)에서 8항목 생성 부분을 보면 buf8_7 변수에 적용하는 배열 인덱스가 7항목 배열의 인덱스 값을 갖도록 하고 있는데 이와 같이 하는 이유는 8 크기 항목의 일곱 번째까지 요소가 7 크기 항목의 jmp_7 위치에 있는 내용과 일치하기 때문에 다시 생성할 필요가 없어지기 때문이다.

5.3 점진적 후보항목 관리기

앞 절에서 기술한 트랜잭션 단위 후보항목 생성 알고리즘에 의해 생성된 후보 항목들은 이제 점진적 후보 항목 관리기(Incremental Manager)에 의해 지지도 갱신 대상 항목들과 새로 삽입될 항목들로 분류한다. 여기서 항목 분류기의 처리 과정은 전체 알고리즘의 성능을 결정하는 가장 중요한 부분이다. 왜냐하면 이 항목 분류기에서 현재 테스트 중인 항목이 후보 항목 데이터베이스에 새로 삽입되어야 할 항목인지 아니면 기존에 존재하고 있는 항목이어서 지지도만을 갱신해야 할 항목인지를 결정해야 하는데 이와 같은 사항을 결정하기 위해 최악의 경우 최대 트랜잭션 크기로 지정한 12에 대응해서 $2^{12}-1$번의 데이터베이스 스캔을 필요로 할 수도 있기 때문이다. 다음의 (그림 5)는 분류기에 대한 구조이다.

```
Procedure Item_set_divider is
begin
   bnd : = number of n_size_items ;
   for cnt in 1..bnd loop
      classifier : = Get_From_Hash( cnt ) ;
      if classifier.tag = new or classifier.tag = conflicting then
         make insert_item_set for classifier.item_id
      elsif classifier.tag = existing then
         make update_item_set for classifier.item_id ;
      end if ;
   end loop ;
end ;
```

(그림 5) 후보 항목 분류기 골격

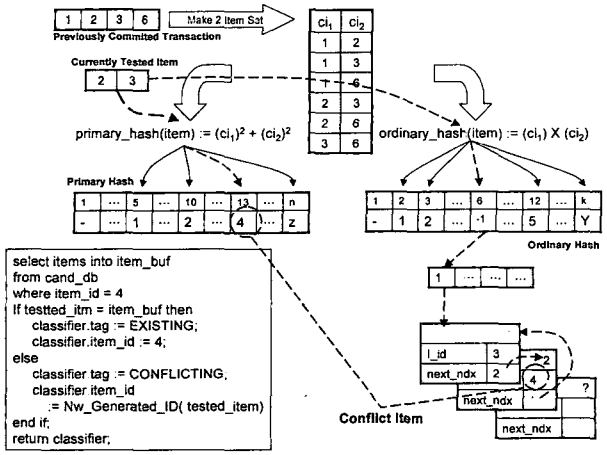
테스트 할 항목은 (그림 4)의 알고리즘에 의해 생성된 후 전역변수로 존재한다. 따라서 항목 분류기에서는 전역변수로 존재하는 후보 항목들의 위치만을 이용하여 식별할 수 있다. 항목 분류기는 먼저 테스트 할 항목의 위치 식별자를 이용하여 Get_From_Hash용하여 함수를 호출한다. 그러면 Get_From_Hash 함수는 테스트 할 항목이 삽입 항목인지, 아니면 갱신 항목인지를 결정하여 결정값을 classifier 변수에 반환한다. classifier 변수는 항목 식별자와 충돌 태그를 갖고 있는 복합 데이터 타입의 변수이다. Get_From_Hash 함수로부터 classifier로 반환된 값을 확인해서 해당 항목을 삽입 차분 집합인 insert_item_set 또는 갱신 차분인 update_item_set에 할당함으로써 분류 과정을 완료한다.

Get_From_Hash 함수는 앞에서 언급한 데이터베이스 스캔 문제를 해결하기 위해 두 개의 해쉬 테이블을 이용하여 데이터베이스 스캔을 최대한 억제하면서 항목의 소속을 결정하는 역할을 수행한다. 해쉬 테이블을 이용하는 이유는 데이터베이스에 대한 스캔 없이 단지 해쉬 테이블관을 조사함으로써 항목의 존재 여부를 결정할 수 있기 때문이다.

그러나 하나의 해쉬 테이블만을 이용하여 후보 항목을 관리하고자 할 경우에는 해쉬 테이블 상에서 발생하는 많은 충돌 문제를 해결하기가 어렵고 또 명시적 테스트 중인 항목의 데이터 값들을 이용하여 생성한 해쉬 키 위치가 비어 있을 경우에만 데이터베이스 스캔 없이 항목의 종류를 삽입 대상 항목으로 결정 할 수 있을 뿐 이외의 경우에는 모두 후보항목 데이터베이스에 대한 스캔 연산을 수행해야 한다. 따라서 하나의 해쉬 테이블만을 사용하는 방법은 성능상의 잇점을 충분히 발휘하지 못한다. 이와 같은 문제를 해결하기 위해 이 논문에서는 기준 해쉬 테이블(primary hash table : p_table)과 보조 해쉬 테이블(ordinary hash table : o_table) 두 개의 해쉬 테이블을 이용한다. 이 때 이 두 해쉬 테이블의 해쉬 키 생성 알고리즘을 서로 다르게 구성하여 하나의 항목으로부터 생성되는 해쉬 키가 서로 일치하지 않도록 한다. 이와 같이 구성하게 되면 데이터베이스 스캔을 상당 부분 줄일 수 있다.

다음의 (그림 6)은 두 개의 해쉬 테이블을 이용하여 테스트 중인 후보 항목의 삽입 또는 갱신 여부를 결정하는 시나리오를 표현하고 있다. (그림 6)에서 해쉬 테이블 값이 존재하지 않는 경우는 아직 해당 해쉬 키 위치에 어떤 항목도 할당되지 않았었음을 의미하며 0보다 큰 값이 있을 때는 이전에 해당 해쉬키 위치에 어떤 항목 값이 할당되었으나 충돌은 발생하지 않은 상태를 의미하고 0보다 작은 값은 이미 해당 해쉬 키 위치에 두 개 이상의 항목들이 할당 된 충돌 상태를 의미한다. 또한 (그림 6)에서 가는 실선은 이전에 커밋 완료된 트랜잭션에 의해 이미 후보 데이터베이스 및 두 개의 해쉬 테이블에 반영되어 있는 후보 항목들에 대한 컨트롤 과정

을 지정하고 있다. 그리고 굵은 점선은 현재 테스트되고 있는 하나의 항목에 대한 컨트롤 과정을 나타내고 있다.



(그림 6) 두 개의 해쉬를 이용한 후보 항목 테스트 시나리오

(그림 6)에서 표현하고자 하는 상황은 이미 커밋 완료된 트랜잭션에 의해 구성되어 있는 해쉬 테이블 상에 새로운 항목 (2,3)이 삽입되었을 때 이 항목이 어떻게 해쉬 테이블들에 반영되는가를 보여 준다. (그림 6)에서 (2,3)이라는 새로운 테스트 항목이 삽입되었을 때 기준 해쉬 테이블의 해쉬 키를 결정하는 해쉬 함수에 의해 해쉬 키 값 13이 생성되고 이 위치에 4라는 항목 식별자를 발견한다. 그런 다음 보조 해쉬 테이블의 해쉬 키를 결정하는 해쉬 함수에 의해 해쉬 키 값 6이 결정되면서 -1이라는 충돌집합 식별자를 발견한다. 그러면 알고리즘은 충돌집합에 있는 항목 식별자들을 모두 읽어와서 기준 해쉬 테이블에서 획득한 항목 식별자와 같은 식별자가 있는지 비교해 보고 같은 항목 식별자가 있을 경우 충돌 판정을 한다. (그림 6)의 좌측 하단 알고리즘은 충돌 평가를 위한 알고리즘이다. 반면 기준 해쉬 테이블 또는 보조 해쉬 테이블 둘 중 어느 하나라도 비어 있거나 아니면 일치하는 항목 식별자를 발견하지 못하면 삽입되어야 할 항목으로 결정한다. 따라서 두 개의 해쉬 테이블을 사용하게 될 경우 하나의 해쉬 만을 사용하는 것에 비해 더 많은 데이터베이스에 대한 스캔 비용을 줄일 수 있다. 다음의 <표 2>는 두 개의 해쉬 테이블에 대한 상태에 따라 데이터베이스 스캔 여부를 결정하는 결정 인자를 정리해 놓은 것이다.

〈표 2〉 두 해쉬를 이용한 데이터베이스 스캔 결정 인자

p_hash \ o_hash	Empty	Positive	Negative
Empty	× ①	× ②	× ③
Positive	× ④	○/× ⑤	○/× ⑥
Negative	× ⑦	○/× ⑧	○/× ⑨

<표 2>에서 ○/×로 표시된 부분은 일단 두 해쉬 테이블에서 일치하는 항목 식별자가 존재하는지를 조사한 후 일치하는 항목 식별자가 존재할 경우 데이터베이스 스캔을 수행하고 그렇지 않을 경우 데이터베이스 스캔을 하지 않는 선택

```
function Get_From_Hash(in tested_item) return class_rec
is
    p_hash              number(11) ;
    o_hash              number(11) ;
    classifier          class_rec ;
    item_buf            item_buf_t ;
    items               c_itm_2_tab ;
    classifier          class_rec ;
    itm_id              number ;
begin
    seed_num_make( tested_item, p_hash, o_hash ) ;    /* 해쉬 키 생성 프로시져 호출 */
    if not p_table.exists(p_hash) or not o_table.exists(o_hash) then /* <표 1>의 ① ② ③ ④ ⑦ 의 경우 */
        make and assign new item_id for tested_item ;
        managing conflict with appropriate hash table ;
    else                                          /* 두 해쉬 테이블 모두에 반드시 어떤 값이 존재 할 경우 */
       if p_table(p_hash) * o_table(o_hash) < 0 then
          if existing matched_item_id between p_table and o_table then      /* <표 1> ⑥ ③의 ○ 부분 */
             select items into item_buf form cand_db where item_id = matched_item_id ;    /* DB Scan */
             compare between tested_item and item_buf ;
             if compared_result is true then
                set existing item_id into calssifier.item_id and classifier.tag = existing ;
             else
                make and assign new item_id for tested_item and managing conflict with appropriate hash table ;
             end if ;
       else                                                    /* <표 1> ⑥ ⑧의 × 부분 */
          make and assign new item_id for tested_item ;
          managing conflict with appropriate hash table ;
       end if
```

(그림 7)(a) Get_From_Hash 함수에서 <표 1>의 ① ② ③ ④ ⑥ ⑦ ⑧ 부분 알고리즘

```
        elsif p_table(p_hash)*o_table(o_hash) > 0 then
            if p_table(p_hash) > 0 then                    /* <표 1>의 ⑤ 부분 */
                if p_table(p_hash) = o_table(o_hash) then  /* <표 1> ⑤의 ○부분 */
                    select items into item_buf from cand_db where item_id = p_table(p_hash) ; /* DB Scan */
                    compare between tested_item and item_buf ;
                    if compared_result is true then        /* 테스트 항목이 후보항목 테이블 상에 이미 존재하고 있다면 */
                        set existing item_id into calssifier.item_id and classifier.tag = existing ;
                    else                                   /* 충돌 처리 */
                        make and assign new item_id for tested_item and managing conflict with appropriate hash table ;
                    end if ;
                else                                       /* <표 1>의 ⑤ 의 ×부분 */
                    make and assign new item_id for tested_item and managing conflict with appropriate hash table ;
                end if ;
            else                                           /* <표 1>의 ⑨ 부분 */
                compare with two hash_table ;
                if compare result is not empty then        /* <표 1> ⑨의 ○ 부분 */
                    select items into item_buf from cand_db where item_id = p_table(p_hash) ;
                    compare between tested_item and item_buf ;
                    if compared_result is true then        /* 테스트 항목이 후보항목 테이블 상에 이미 존재하고 있다면 */
                        set existing item_id into calssifier.item_id and classifier.tag = existing ;
                    else                                   /* 충돌 처리 */
                        make and assign new item_id for tested_item and managing conflict with appropriate hash table ;
                    end if ;
                else                                       /* <표 1> ⑨의 ×부분 */
                    make and assign new item_id for tested_item and managing conflict with appropriate hash table ;
            end if ;
        end if ;
    end if ;
    return classifier ;
end get_from_hash_2 ;
```

(그림 7)(b) Get_From_Hash함수에서 <표 1>의 ⑤ ⑨ 부분 알고리즘

적 스캔 영역임을 나타낸다. 또한 <표 2>에서 원 번호는 뒤에 알고리즘과 결부지어 설명하기 쉽도록 하기 위해 붙여 놓은 셀 번호이다. <표 1>에서 볼 수 있는 바와 같이 두 개의 해쉬를 사용하게 되면 두 개의 해쉬에서 동일한 항목 식별자를 발견하지 못할 경우 데이터베이스 스캔을 하지 않게 되므로 데이터베이스 스캔 연산을 획기적으로 줄일 수 있게 됨으로써 성능을 극대화 할 수 있게 한다. 다음의 (그림 7) (a)는 이와 같은 과정을 수행하는 Get_From_Hash 함수에서 <표 2>의 ① ② ③ ④ ⑥ ⑦ ⑧ 부분을 다루는 알고리즘을 기술한다.

(그림 7)(a)에서 먼저 o_table과 p_table로 식별되는 해쉬 테이블 연산을 수행하기 위한 기본 조건으로 해쉬 키 값을 생성하는 프로시져인 seed_num_make를 호출한다. 이 프로시져의 호출에 의해 생성된 해쉬 키 o_hash와 p_hash는 이후 해쉬 테이블 연산에 있어 핵심적 역할을 한다. 일단 생성된 해쉬 키를 이용하여 해당 테이블의 해쉬키 위치가 비어 있는지 아니면 유효한 값으로 채워져 있는지 검사한다. 만일 두 해쉬 테이블 중 하나라도 비어 있다면 이 경우에는 무조건 테스트 중인 항목은 삽입 항목으로 처리된다. 반면 두 해쉬 테이블에 모두 유효한 값으로 채워져 있다면 두 해쉬 테이블 중 하나에만 충돌이 발생한 경우와 모두 충돌이 발생하지 않은 경우 그리고 모두 충돌이 발생한 경우에 대해서 각각 조사를 한다. (그림 7)(a)는 두 해쉬 테이블 중 하나에만 충돌이 발생하고 다른 하나는 충돌이 발생하지 않은 유효한 값으로 채워져 있는 부분에 대해서도 기술하고 있다. 반면 다음의 (그림 7)(b)는 모두 충돌이 발생하지 않은 경우와 모두 충돌이 발생한 경우에 대해서 기술하고 있다.

(그림 7)(b)에서 두 해쉬 테이블에 충돌이 발생하지 않은 경우에는 두 해쉬 테이블의 값이 일치하는지를 비교 해 보고 일치한다면 해쉬 테이블 값을 이용하여 데이터베이스 스캔을 수행한다. 그렇지 않다면 데이터베이스 스캔을 필요하지 않다. 반면 두 해쉬 테이블이 모두 충돌 상태라면 먼저 충돌 집합으로부터 일치하는 항목 식별자가 있는지를 먼저 확인 해 보아야 한다. 만일 두 해쉬테이블에 의해 관리되는 충돌집합에서 일치하는 항목 식별자를 발견한다면 데이터베이스 스캔을 해야 한다. 그렇지 않다면 데이터베이스 스캔은 필요하지 않다.

6. 실험 및 성능 평가

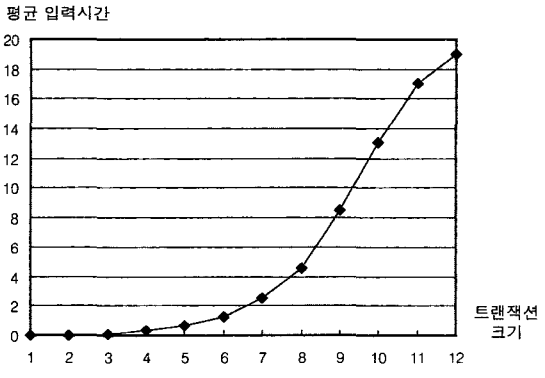
이 논문에서 트리거를 이용한 점진적 연관규칙 탐사 기법의 성능 평가를 데이터 집합은 다음과 같은 특성을 갖고 있다.

전체 트랜잭션 수	전체 항목수	트랜잭션당 평균 항목수	빈발항목 집합			
			패턴의 총 수	패턴의 평균길이	신뢰도	신뢰도 분산
20,000	99	8	4,000	5	75%	10%

이와 같은 데이터를 이용한 실험 환경은 다음과 같다.

시스템 : Sun (TM) Enterprise 250 (UltraSPARC-II 400MHz 1 CPU)
메모리 : 524288K
DBMS : Oracle8i Enterprise Edition
구현도구 : Oracle 8 Trigger & PL/SQL

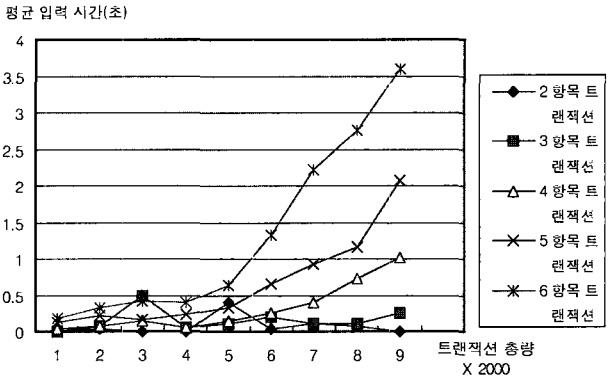
이 논문에서 제시하는 트리거와 저장 프로시저를 이용한 점진적 후보항목 갱신 기법은 데이터 입력이 완전히 이루어진 상태에서 탐사하는 것이 아니라 트랜잭션 데이터의 삽입이 발생할 때마다 동적으로 재구성을 해야 한다. 따라서 빈 트랜잭션 데이터베이스에서부터 시작해서 내장 프로시저를 이용한 연관규칙 탐사를 수행한 데이터베이스와 동일한 형태의 데이터베이스가 형성될 때까지 연속적으로 데이터를 삽입하면서 실험을 수행하였다. 이 때 트랜잭션 항목의 크기에 따라 실제 데이터가 삽입되면서 후보항목을 생성하는 트리거가 완전히 수행될 때까지의 시간에 대한 평균치를 측정한 결과가 다음의 (그림 8)에 나타나 있다. 그림에서 보는 바와 같이 트랜잭션의 크기가 8을 초과하면서부터 연산 부하가 현격히 증가함을 발견할 수 있다.



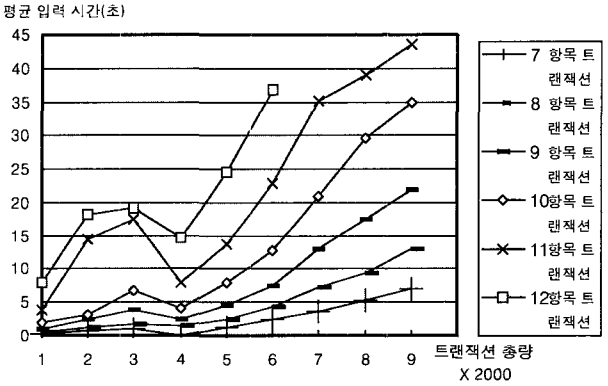
(그림 8) 트랜잭션 크기에 따른 평균 입력 시간의 변화 추이

(그림 8)의 결과는 트랜잭션의 크기가 증가할수록 갱신 연산에 참여하는 후보 항목의 수가 기하 급수적으로 증가하기 때문에 발생하는 결과로 풀이할 수 있다. 다음으로 (그림 9)(a)와 (그림 9)(b)는 입력된 트랜잭션이 양이 증가하는 동안 k 크기 트랜잭션에 나타나는 평균 입력 시간을 측정한 것이다. (그림 9)(a)는 항목 크기 2~6 사이의 트랜잭션에 대한 평균 입력 시간을 측정한 것이고 (그림 9)(b)는 항목 크기 7~12 사이의 트랜잭션에 대한 평균 입력 시간을 측정한 것이다. 그림을 (a)와 (b)로 구분한 것은 트랜잭션의 크기가 6을 초과하는 부분과 그 이하 부분 사이의 평균 입력 시간의 차가 너무 크기 때문에 동일 그래프로 표현하는데 매우 부적합한 특성을 나타내기 때문이다.

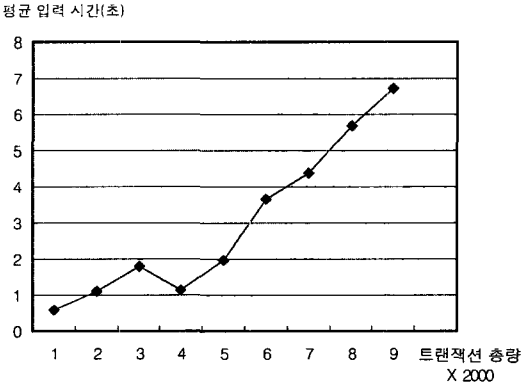
(그림 9)에서 항목크기 1인 트랜잭션을 포함하지 않은 이유는 모두 0으로 나오기 때문에 평가 대상으로 포함할 필요가 없기 때문이다. 아울러 (그림 9)(b)에서 항목 크기 9, 10, 11, 12인 트랜잭션들이 트랜잭션 총량 8000 주변에서 급격한 변화를 보이는 이유는 이 영역의 트랜잭션들이 입력되는 과정에서 충돌 발생률이 극히 적은 관계로 (그림 6)에서 설명하고 있는 테스트 연산의 연산 부하가 많지 않기 때문으로 풀이할 수 있다. 이에 대해서 다음의 (그림 10)과 (그림 11)에 대한 분석에서 보충 설명을 추가한다.



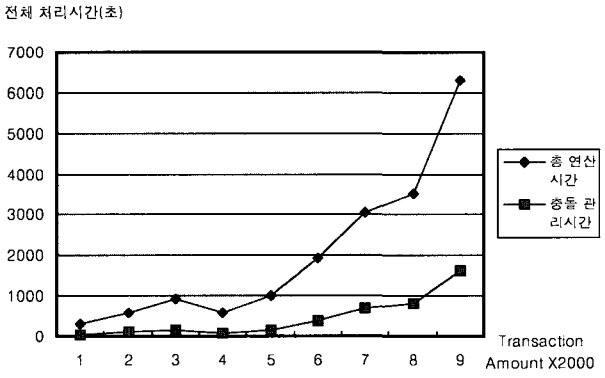
(그림 9)(a) 항목크기 2~6 사이의 트랜잭션에 대한 평균 입력시간 변화 추이



(그림 9)(b) 항목크기 7~12 사이의 트랜잭션에 대한 평균 입력시간 변화 추이



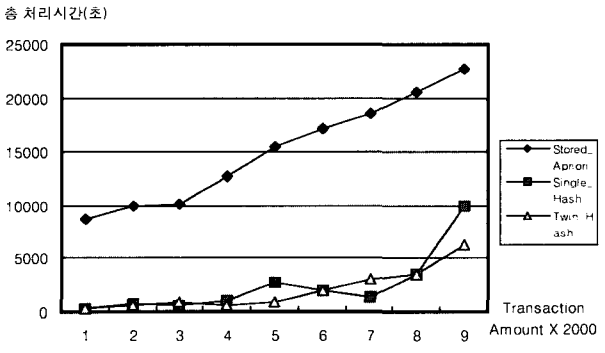
(그림 10) 입력 트랜잭션 총량의 증가에 따른 평균 입력 시간의 변화 추이



(그림 11) 트랜잭션 입력 성능에 항목 충돌이 미치는 영향 비교 분석

(그림 10)은 트랜잭션의 총량이 증가해 감에 따라 나타나는 평균 입력 시간의 증가 추이를 나타낸 것이다. 트랜잭션의 총량이 증가해 감에 따라 내부 연산 비용이 증가하게 되어 점진적으로 처리비용이 증가해 가는 특성이 나타남을 파악할 수 있다. 다음으로 (그림 11)은 트랜잭션 총량이 증감에 따라 나타나는 전체 처리시간과 전체 처리시간 내에 포함된 항목 충돌 처리시간 사이의 상호 관계를 비교하는 그래프이다. (그림 11)에서 트랜잭션 총량 8000 부근과 16000 부근에서 충돌 처리시간이 상대적으로 적게 나타나면서 전체 처리시간에 상당한 영향을 주고 있음을 확인할 수 있다. 이는 (그림 9)(b)의 8000 부근에서 나타나는 현상에 대한 원인을 설명하는 중요한 단서라 할 수 있다.

다음으로 (그림 12)는 [2]에서 제안한 저장 프로시져를 이용한 Apriori 알고리즘과 [16]에서 수행한 단일 해쉬를 이용한 방법 및 이 논문에서 제안하고 있는 이중 해쉬를 이용한 방법 사이의 성능 특성에 대한 비교 분석을 나타낸다.



(그림 12) 저장 프로시져로 작성한 Apriori 알고리즘 및 단일 해쉬 기법의 알고리즘과 성능 비교

(그림 12)에서 단일 해쉬 방법을 사용하는 알고리즘과 이중 해쉬를 사용하는 알고리즘은 정량적 비교 분석이 가능하다. 왜냐하면 이 두 방법은 모두 트리거와 저장 프로시져를 이용하여 후보항목을 실시간 점진적으로 생성 관리하는 모델을 구현하고 있기 때문이다. 그러나 [2]에서 제안한 저장 프로시져를 이용한 방법은 이미 입력 완료된 상태의 데이터베이스를 대상으로 일시에 데이터베이스 스캔을 통해 후보항목 및 빈발항목을 생성하는 연산을 수행한다. 따라서 이 논문에서 제시하는 방법과 정량적 비교 분석 자체가 의미를 갖을 수 없다. 다만 (그림 12)에 [2]의 방법을 포함하여 제시한 것은 이 논문에서 제시하고 있는 방법이 기존에 제시되었던 방법과 비교하여 어떤 성능 특성을 나타내는지에 대한 객관적 자료로서의 의미를 갖을 뿐이다. 한편 [16]에서 제시한 단일 해쉬를 이용한 방법과 비교하여 이 논문에서 제시하는 이중 해쉬를 이용한 방법이 성능상으로 좀 더 안정적인 특성을 나타냄을 알 수 있다. 그림에서 볼 수 있는 바와 같이 트랜잭션 총량의 변화에 따라 나타나는 총 수행시간의 변화 추이가 단일 해쉬 방법이 다소 불규칙적인 상태인데 반하여 이 논문에서 제안하는 이중 해쉬를 이용하는 방법은 선형적인 증가 추세를 나타내는 것을 확인할 수 있다. 이는 이 논문에서 제안하는 방법이 이중 해쉬를 이용함으로써 디스크 입출력 빈도를 감소시켜 보다 안정적 성능을 발휘하도록 하는 것으로 해석할 수 있다.

7 결 론

이 논문에서는 실시간 연관규칙 탐사를 수행하는 데 있어 연산비용이 가장 높은 후보 항목 갱신 연산을 트리거와 점진적 갱신 기법을 이용하여 수행하는 수행 모델을 개발하고 이의 구현 및 성능 특성을 분석하였다. 이 논문에서 제안하는 실시간 연관규칙 탐사를 위한 능동적 후보항목 갱신 모델은 트랜잭션의 삽입과 동시에 삽입된 트랜잭션에 대응해서 점진적 후보항목 갱신 연산을 자동으로 수행하도록 하는 모델이다. 이를 위해 이 논문에서는 트리거를 이용하여 실시간 후보항목 갱신 연산을 자동으로 호출하도록 하였으며 저장 프로시져를 이용하여 모든 연산을 구현함으로써 데이터베이스와 후보항목 갱신 프로시져 사이의 프로세스 변환에 따른 부하를 최소로 하였다. 뿐만 아니라 데이터베이스 시스템에서 지원하는 기능 이외의 어떤 기능이나 도구도 사용하지 않았으므로 구현 및 이식성이 우수한 특성을 갖는다. 또한 후보 항목의 실시간 갱신에 있어 데이터베이스 연산 부하를 최소화하기 위해 두 개의 해쉬 테이블을 이용하는 기법을 적용함으로써 후보항목 갱신 과정에서 발생하는 과도한 데이터베이스 스캔 비용을 현저히 낮추어 성능상의 문제를 해결하였다. 특히 성능상의 문제는 6장 실험 및 평가에서 볼 수 있는 바와 같이 트랜잭션 당 평균 처리 시간이 최대 10초를 초과하지 않는 특성을 나타냄으로써 시간적 제약을 갖는 특수한 품목이나 거래 패턴이 불규칙적으로 나타나는 인터넷 전자 상거래 등에서 실시간적으로 연관규칙을 탐사할 수 있도록 하는데 효과적으로 적용할 수 있게 하였다.

한편 이 논문의 기본 목적이 트리거와 마이닝의 결합 가능

성을 검증하는데 두고 있기 때문에 실시간 연관규칙 탐사가 가능하도록 하는 핵심 요소인 트리거와 저장 프로시저를 이용한 점진적 후보항목 갱신 알고리즘을 제시하였다. 따라서 이 논문에서 제시된 가능성을 토대로 다른 마이닝 기법들에 해당하는 분류화(classification), 군집화(clustering), 순차 패턴(sequential pattern) 등을 탐사하는데 확장 적용하기 위한 방안의 연구가 추가적으로 수행될 필요가 있다.

참 고 문 헌

[1] R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. of the 20th Int'l Conference on Very Large Databases, 1994.

[2] R. Agrawal, K. Shim, "Developing Tightly-Coupled Data Mining Applications on a Relational Database System," Proc. of the 2nd Int'l Conference on Knowledge Discovery in Databases and Data Mining, Portland, Oregon, August, 1996.

[3] S. Sarawagi, S. Thomas, R. Agrawal, "Integrating association rule mining with databases : alternatives and implications," Proc. of the ACM SIGMOD Int'l Conference on Management of Data, Seattle, Washington, June, 1998.

[4] J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaiane, "DMQL : A Data Mining Query Language for Relational Databases," 1996 SIGMOD'96 Workshop. on Research Issues on Data Mining and Knowledge Discovery (DMKD'96), Montreal, Canada, June, 1996.

[5] Jong Soo Park, Ming-Syan Chen, Philip S. Yu, "An Effective Hash Based Algorithm for Mining Association Rules," SIGMOD Conference, 1995.

[6] D. Cheung, J. Han, V. Ng and C.Y. Wong, "Maintenance of Discovered Association Rules in Large Databases : An Incremental Updating Technique," Proc. of 1996 Int'l Conf. on Data Engineering (ICDE'96), New Orleans, Louisiana, USA, 1996.

[7] R. Agrawal, G. Psaila, "Active Data Mining," Proc. of the 1st Int'l Conference on Knowledge Discovery and Data Mining, Montreal, August, 1995.

[8] J. Han, S. Nishio and H. Kawano, "Knowledge Discovery in Object-Oriented and Active Databases," F. Fuchi and T. Yokoi (eds.), Knowledge Building and Knowledge Sharing, Ohmsha, Ltd. and IOS Press, 1994.

[9] Jennifer Widom, Stefano Ceri, "Chapter 1 : Introduction to Active Database Systems," Active Database Systems, Morgan Kaufmann Publishing Inc, 1996.

[10] Norman W. Paton, Andrew Dinn, M. Howard Williams, "Chapter 4 : Optimization," Active Rules in Database Systems, Springer, 1999.

[11] Umeshwar Dayal, Barbara T. Blaustein, Alejandro P. Buchmann, Upen S. Chakravarthy, M. Hsu, R. Ledin, Dennis R. McCarthy, Arnon Rosenthal, Sunil K. Sarin, Michael J. Carey, Miron Livny, Rajiv Jauhari, "The HiPAC Project : Combining Active Databases and Timing Constraints," SIGMOD Record 17(1), 1988.

[12] Eric N. Hanson, "Rule Condition Testing and Action Execution in Ariel," SIGMOD Conference, 1992.

[13] J. S. Park, Y. H. Shin, K. W. Nam, K. H. Ryu, "Incremental Condition Evaluation for ActiveTemporal Rule," Journal of KISS, (B) 26(4), 1999.

[14] ANSI X3H2-99-079/WG3 : YGJ-011 (ANSI/ISO Working Drft) Foundation(SQL/Foundation), March, 1999.

[15] Spyros Potamianos, Michael Stonebraker, "Chapter 2 : The POSTGRES Rule System," Active Database Systems, Morgan Kaufmann Publishing Inc, 1996.

[16] J. H. Whang, Y. H. Shin, K. H. Ryu, "An Active Candidate Set Management Model on Association Rule Discovery using Database Trigger and Incremental Update Technique," Journal of KISS, (B), 29(1), 2002.

[17] Y. J Lee, S. B. Seo, K. H. Ryu, "Discovering Temporal Relation Rules from Temporal Interval Data," Journal of KISS, (B), 28(9), 2001.

[18] J. S Song, Y. J. Lee, K. H. Ryu, "Discovering Relationship rule from Interval Data," to be submitted the ETRI Journal, 2001.

[19] Oracle 8i Development Guide : PL/SQL, Oracle Press, 2000.

신 예 호

e-mail : snowman@db.ab.chungbuk.ac.kr

1996년 군산대학교 컴퓨터과학과 졸업(학사)

1998년 충북대학교 대학원 전자계산학과 졸업(석사)

2002년 충북대학교 대학원 전자계산학과 졸업(박사)

현재 극동대학교 정보통신학부 전임강사

관심분야 : 능동 데이터베이스, 시간 데이터베이스, 공간 데이터베이스, 데이터 마이닝

류 근 호

e-mail : khryu@dblab.chungbuk.ac.kr

1976년 숭실대학교 전산학과(이학사)

1980년 연세대학교 산업대학원 전산전공(공학석사)

1988년 연세대학교 대학원 전산전공(공학박사)

1976~1986년 육군군수 지원사 전산실(ROTC 장교), 한국전자통신 연구원(연구원), 한국방송통신대 전산학과(조교수) 근무

1989년~1991년 Univ. of Arizona Research Staff(TempIS 연구원, Temporal DB)

1986년~현재 충북대학교 전기전자및컴퓨터공학부 교수

관심분야 : 시간 데이터베이스, 시공간 데이터베이스, Temporal GIS, 객체 및 지식기반 시스템, 지식기반 정보검색 시스템, 데이터 마이닝, 데이터베이스 보안 및 Bio-Informatics