트리거와 점진적 갱신기법을 이용한 연관규칙 탐사의 능동적 후보항목 관리 모델

(An Active Candidate Set Management Model on Association Rule Discovery using Database Trigger and Incremental Update Technique)

황 정 희 [+] 신 예 호 [++] 류 근 호 [+++]
(Jeong Hee Hwang) (Ye Ho Shin) (Keun Ho Ryu)

요 약 연관규칙 탐사는 지지도와 신뢰도를 바탕으로 연관성이 강한 항목들을 탐사한다. 탐사된 연관규칙은 장바구니 분석 등과 같이 전자 상거래 및 대형 소매점 등의 판매 패턴에 대한 분석에 유용하게 적용될 수 있다. 이와 같은 연관규칙 탐사는 대규모로 축적되어 있는 트랜잭션 데이타를 대상으로 하는 기법으로서 대규모 데이타에 대한 반복적 스캔연산을 수반한다. 그러므로 매우 높은 연산 부하를 안고 있으며 이로 인해 동적 환경에서 실시간 연관규칙 탐사에 대한 시도를 하지 못하고 있다. 따라서 이 논문에서는 연관규칙 탐사의 비 실 시간적 제한사항을 극복하기 위하여 트리거와 점진적 갱신 기법을 이용한 능동적 후보항목 관리 모델을 제안하였다. 아울러 제안 모델을 구현하기 위해 점진적 갱신 연산의 구현 모델을 제시하고 이의 구현 및 실험을 통해 성능 특성을 분석하였다.

키워드 : 능동 마이닝, 능동 데이타베이스, 능동규칙, 트리거 모델

Abstract Association rule discovery is a method of mining for the associated item set on large databases based on support and confidence threshold. The discovered association rules can be applied to the marketing pattern analysis in E-commerce, large shopping mall and so on. The association rule discovery makes multiple scan over the database storing large transaction data, thus, the algorithm requiring very high overhead might not be useful in real-time association rule discovery in dynamic environment. Therefore this paper proposes an active candidate set management model based on trigger and incremental update mechanism to overcome non realtime limitation of association rule discovery. In order to implement the proposed model, we not only describe an implementation model for incremental updating operation, but also evaluate the performance characteristics of this model through the experiment.

Key words: Active Mining, Active Database, Active Rule, Trigger Model

1. 서 론

연관규칙 탐사는 트랜잭션 데이타베이스를 바탕으로 지지도(support)와 신뢰도(confidence)라는 두 제한자(threshold)를 이용하여 연관성이 강한 항목들을 찾아내는 것으로 정의할 수 있다[1]. 이와 같은 연관규칙 탐사의 가장 전형적인 응용은 전자 상거래 및 대형 할인점 등과 같이 대규모 거래가 이루어지는 유통환경에서의 장바구니 분석을 들 수 있다. 이와 같은 유통 환경 하에서 장바구니 분석을 통해 탐사된 연관규칙은 진열대(show case)의 진열 형태를 최적화함으로써 매출의 신장에 기여하는 수단으로 활용되어 왔다.

한편 대형 유통환경에서 계절 또는 시간대 및 품목의 특수성에 따라 발생하는 특수한 판매 환경을 적절히 조절함으로써 매출의 극대화를 꾀할 수 있다. 예를 들어

· 이 논문은 한국과학재단 목적기초연구(R01-1999-00243)지원으로 수행되었음.
+ 학생회원 : 충북대학교 전자계산학과
jhhwang@dblab.chungbuk.ac.kr
++ 비 회 원 : 충북대학교 전자계산학과
snowman@dblab.chungbuk.ac.kr
+++ 종신회원 : 충북대학교 전기전자컴퓨터공학부 교수
khryu@dblab.chungbuk.ac.kr
논문접수 : 2001년 5월 28일
심사완료 : 2001년 10월 15일

생선 및 채소류와 같은 신선 식품의 경우 일정 시간 이내에 제고를 소진 시켜야만 물류비 및 저장 비용을 최소화 할 수 있으며 이는 결국 매출의 극대화와 직결되는 문제가 된다. 또한 전자 상거래에서 시간대별로 구매 패턴을 분석 해 낼 수 있다면 해당 시간대에 가장 적합한 광고를 게재할 수 있게 됨으로써 매출의 신장과 연결시킬 수 있다. 이와 같이 극도로 제한적인 조건 하에서 역동적으로 변화하는 거래 형태들을 적기에 분석해 내기 위해서는 실시간 연관규칙 탐사기법이 필요하다.

그러나 기존의 연관규칙 탐사 기법들은 특정 시점의 데이타베이스 상태 전체를 대상으로 하고 있으며 연관규칙 탐사를 위해 과도한 데이타베이스 연산을 수반한다. 뿐만 아니라 연관규칙의 탐사시점 이후에 발생하는 트랜잭션은 전혀 탐사 대상으로 고려할 수 없기 때문에 앞에서 예시한 바와 같이 시간적 제약을 받아서 실시간 연관규칙 탐사를 필요로 하는 동적 환경 하에서 연관규칙의 실시간 탐사는 불가능에 가까워지는 문제점을 갖고 있다. 따라서 이 논문에서는 이와 같은 문제 즉 시간적 제약을 받는 특수한 판매 형태에 대한 분석을 효과적으로 수행할 수 있도록 하기 위한 방안으로 트랜잭션 데이타의 삽입과 동시에 연관규칙의 탐사가 가능하도록 하기 위해 데이타베이스 트리거(database trigger)와 점진적 갱신기법(incremental updating technique)을 결합한 능동적 후보항목 관리 모델을 제시하고 이의 구현 및 실험을 통해 제안 모델의 타당성을 검증한다.

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 연관규칙 탐사에 적용될 수 있는 트리거 모델을 제시하고 4장에서는 트리거와 점진적 갱신 기법을 결합한 능동적 후보항목 갱신 모델에 대한 정형의미를 기술한다. 다음으로 5장에서 이의 구현 모델 및 구현 알고리즘에 대해 기술을 하고 6장에서 실험 및 성능 평가를 수행한 후 7장에서 결론 및 향후 연구방향에 대해 기술한다.

2. 관련연구

연관규칙의 탐사는 특성상 데이타베이스에 대한 반복적 스캔을 필요로 하며, 연관규칙 탐사 프로세스와 데이타베이스 프로세스 사이에 대규모 데이타 전송을 필요로 한다. 특히 온라인 연관규칙 탐사는 데이타베이스 프로세스와 연관규칙 탐사 프로세스가 병행 수행되면서 데이타베이스 내의 데이타를 연관규칙 탐사 프로세스가 직접 참조하므로 연관규칙 탐사 프로세스와 데이타베이스 프로세스 사이의 데이타 전송량은 기하급수적으로 증가하게 된다. 이러한 문제점을 해결하기 위해 [2]에서는 연관규칙 탐사 프로세스를 호스트 언어를 사용하지 않고 데이타베이스의 저장 프로시저(stored procedure)를 이용하여 수행하는 방법을 제시하였다. 이는 데이타베이스 시스템의 구조를 변경하지 않고도 효과적으로 데이타베이스 시스템과 연관규칙 탐사 프로세스를 통합할 수 있는 방안으로서 데이타변환이나 프로세스 사이의 데이타 전송을 필요로 하지 않기 때문에 호스트 언어(host language)를 이용하는 것 보다 더 적은 노력으로 연관규칙 탐사 프로세스를 구현할 수 있다. 뿐만 아니라 연관규칙 탐사과정에서 프로세스간 통신량의 감소로 훨씬 높은 성능 특성을 나타낼 수 있다[2].

데이타베이스 시스템과 연관규칙 탐사 프로세스의 통합에 대한 또 다른 관점은 질의어를 이용한 연관규칙 탐사 기법이다. 여기서 질의어를 이용한 연관규칙의 탐사가 의미하는 바는 질의어가 단순히 목표 데이타를 추출하는 수단으로만 적용하는 것이 아니라 질의어 자체만으로 연관규칙의 탐사를 수행할 수 있도록 하는 것을 의미한다. 이와 같은 연구의 선구적 업적은 [3]에서 찾아볼 수 있는데 여기서 Han등은 연관규칙 등의 탐사를 위한 별도의 질의어인 DMQL(Data Mining Query Language)을 제안하고 이를 통해 별도의 마이닝 프로세스 없이 데이타베이스로부터 일반화 규칙(generalized rule), 특성화 규칙(characteristics rule), 구분 규칙(discriminant rule), 분류 규칙(classification rule), 연관규칙(association rule)을 DMQL을 이용하여 직접 발견할 수 있도록 하였다. 이와 유사한 연구가 [4]에서 수행되었는데 [3]과의 차이는 탐사대상 지식을 연관규칙 및 순차패턴 탐사로 한정한다는 점과 데이타베이스 시스템에서 지원하는 질의어만을 이용하였다는 점이다. [4]는 데이타베이스 질의어를 마이닝에 적용하기 위해 질의어를 마이닝에 적합하도록 정형화하는 방안을 제시하였다. 이 두 연구는 모두 온라인 마이닝을 수행하는 데 있어서 데이타베이스의 외부적 요인을 최대한 배제한 상태에서 마이닝을 수행할 수 있는 방향을 제시하였다는 점에서 매우 의미 있는 연구라 할 수 있다.

이와는 별도로 마이닝 프로세스의 성능 향상에 대한 연구가 다수 진행되었는데 이 중 주목할 만한 연구로서 점진적 마이닝 기법을 고찰할 필요가 있다. 점진적 마이닝이란 k 시점의 마이닝은 k-1시점까지 누적되어 있는 마이닝 결과에 k시점에 변경된 데이타베이스 상태에 대해서만 마이닝을 수행하여 이를 다시 k-1시점까지의 누

적된 마이닝 결과에 합산함으로써 데이타베이스 전체에 대해 마이닝 한 결과를 도출해 내는 방법으로서 [5]에서는 연관규칙 탐사에 대한 점진적 마이닝 알고리즘을 소개하고 Apriori[1] 및 DHP[6]와의 비교를 통한 성능 평가를 수행하였다.

한편 데이타베이스의 상태 변화에 대한 자동 대응 기능으로 정의될 수 있는 데이타베이스 트리거에 대한 연구가 광범위하게 진행되었다[7, 8]. 이 트리거는 데이타베이스에 대한 수정연산에 대응해서 이 수정 연산이 미리 정의된 적절한 조건을 충족시킬 경우 이에 대한 조치를 데이타베이스 스스로 취할 수 있게 하는 조건부 조치기능을 갖는 것으로서[9, 10, 11] 이미 SQL3 표준에 기본기능으로 채택될 만큼 데이타베이스 분야에서 대중적 기술로 인식되고 있다[12]. 이 데이타베이스 트리거의 자동 대응기능은 시스템 수준에서는 제공하기 힘든 다양한 무결성 문제는 물론 보안 수준의 강화 등에 효과적으로 적용할 수 있다[10, 13].

[14] 및 [15]에서는 데이타베이스 트리거의 자동대응 기능을 마이닝과 결합시키기 위한 방안을 제시하고 있다. 이들 연구에서 제시하고 있는 방안은 우선 마이닝 프로세스를 적용하여 마이닝 지식을 찾아낸 다음 찾아낸 마이닝 지식을 지식베이스에 저장할 때 지식의 저장에 의한 지식베이스의 갱신을 관리하기 위해 트리거를 적용하며 이를 능동적 마이닝 모델로 정의하고 있다. 그러나 이들 연구는 트리거를 마이닝 수행단계에서 적용하지 못하고 단지 마이닝 프로세스에 의해 생성된 마이닝 지식을 관리하는 데 있어 한정적으로 적용하고 있을 뿐이다. 이는 마이닝을 수행하는 데 있어 트리거의 장점을 충분히 활용하지 못하는 문제를 갖고 있다.

이상과 같이 온라인 마이닝과 관련된 기존 연구들에는 몇 가지 잠재된 문제점들을 갖고 있다. 우선 온라인 마이닝 방법의 가장 기본적인 메커니즘은 마이닝 시점에 데이타베이스 전체를 스캔해야 한다는 문제점을 갖고 있다. 이는 앞에서도 여러 번 지적한 바와 같이 데이타베이스 시스템에 상당한 연산 부하를 가중시킬 수 있다. 다음으로 점진적 마이닝의 문제점으로 k시점의 차분을 관리하기 위한 △ 릴레이션의 관리방법에 일정한 제약을 내포하고 있다는 점이다. 이는 차분의 축적시간에 대한 문제와 차분의 양에 대한 문제로 [5]에서 제시한 방법은 k-1시점과 k시점 사이의 시간 간격이 일정시간 이상이어야 한다는 점과 k시점의 차분을 관리하기 위한 △ 릴레이션의 크기가 오차를 희석시킬 수 있을 만큼 충분히 커져야만 정당한 결과를 도출할 수 있다는 점이다. 마지막으로 트리거를 이용한 마이닝 모델들은 트리거를 직접 마이닝에 적용하지 못하고 있으며 오히려 마이닝 자체는 오프라인 마이닝을 가정하고 있기도 하다. 이에 이 논문에서는 이상과 같은 문제들을 해결하기 위해 데이타베이스 수정이 발생한 때마다 수정된 데이타 단위로 트리거를 이용하여 차분 연산 및 마이닝 연산을 수행하는 효율적 마이닝 방법을 제시하고 이를 연관규칙 탐사에 적용하여 그 성능 특성을 분석한다.

3. 트리거를 이용한 연관규칙 탐사 모델

데이타베이스 트리거는 삽입, 삭제, 갱신 등과 같은 데이타베이스 수정 연산에 대응해서 자동으로 조건부조치 기능을 수행한다. 이 논문에서는 데이타베이스 트리거의 자동대응기능을 연관규칙 탐사와 결합하여 실시간 연관규칙을 위한 능동적 후보항목 갱신 모델을 제시한다. 트리거를 연관규칙 탐사에 결합시키기 위한 모델은 대략적으로 두 가지 모델을 가정 할 수 있다. 이 중 하나는 트리거의 조치에 의해 또 다른 트리거를 연쇄적으로 트리거 시킬 수 있도록 하는 재귀적 모델(recursive trigger model)로서 다음의 그림 1과 같은 시나리오를 갖는다.

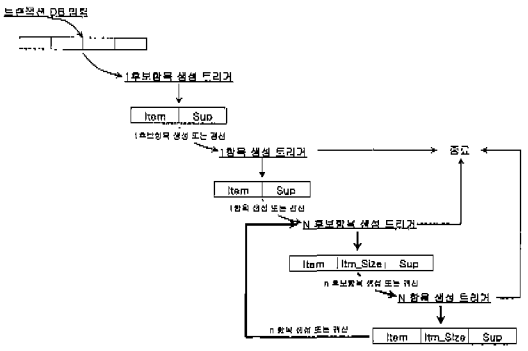

그림 1 재귀적 트리거를 이용한 연관규칙 탐사 시나리오

트리거의 재귀적 모델은 연관규칙 탐사 연산을 단위 연산들로 잘게 분해한 뒤 이들 분해된 연산들을 트리거를 통해 구현하고 구현된 트리거들이 완전한 연산 수행을 위해 재귀적으로 상호 호출하도록 구성하는 것이다. 이렇게 할 경우 트리거만을 이용해서 연관규칙 탐사 연산을 구성할 수 있을 뿐만 아니라 연관규칙 탐사 연산을 구성하는 개별적 트리거들의 규모가 작아져서 이들 단위 연산을 담당하는 트리거의 수행 속도를 증가시킬 수 있다. 그러나 전체적으로는 개별적 트리거들의 트리거 연산 시간이 추가되어 오히려 성능을 저하시키는 원

인이 될 수 있다. 아울러 재귀적 호출은 그림 1의 진한 실선으로 표시된 부분과 같이 트리거의 재귀적 호출로 인해 트리거들 사이의 순환적 사이클을 형성 할 수 있다. 이렇게 형성되는 트리거들 사이의 순환적 사이클은 트리거의 종료를 보장하지 못하는 비 종료 상태를 유발할 수 있다. 이는 트리거의 처리 부하를 가중시켜 시스템 성능에 문제를 발생시킬 수 있을 뿐만 아니라 트리거의 정의 자체를 매우 어렵게 한다[16]. 또한 트리거 수행에 따라 후보항목 테이블 및 빈발항목 테이블을 반복적으로 스캔해야 하므로 트리거 수행 비용 역시 매우 높은 문제점을 갖고 있다.

트리거 구현에 대한 또 다른 모델로 반복적 모델(iterative trigger model)을 들 수 있다. 반복적 모델은 재귀적 모델이 연산을 작게 분해한 뒤 이를 여러 개의 트리거에 의해 분산 구현하는 대신 이들을 하나의 트리거에서 통합 구현하는 구조로서 재귀적 모델이 갖는 대부분의 단점들을 극복할 수 있다. 그러나 이 반복적 모델은 하나의 트리거에 완전한 연산의미들을 구현해야 하므로 트리거 조치 절의 규모가 지나치게 증가하는 문제가 있다. 또한 하나의 트리거 안에 모든 연산들을 구현해야 하므로 구현이 어려워진다. 이 반복적 트리거 모델을 채택하는 연관규칙 탐사는 다시 두 가지의 형태를 고려할 수 있다.

반복적 트리거 모델 중 첫 번째로 고려할 수 있는 방법은 트리거와 [2]에서 제시한 저장 프로시저의 결합 모델이다. 이 모델의 장점은 저장 프로시저를 이용하는 방법이 이미 검증되어 있고 트리거에서 저장 프로시저의 직접 호출이 가능하므로 이미 검증된 방법을 활용할 수 있는 장점을 갖는다. 그러나 트리거는 데이타베이스 상태변화에 즉시 반응하는 특성이 있기 때문에 데이타베이스 갱신이 자주 발생하는 환경에서는 성능상의 부하가 지나치게 증가하여 적용이 불가능한 문제점을 갖고 있다.

반복적 모델의 다른 형태는 트리거와 변형된 저장 프로시저를 결합하는 방법이다. [2]에서 제시한 저장 프로시저는 Apriori 알고리즘을 단순하게 저장 프로시저로 구현했기 때문에 프로시저가 수행될 때마다 전체 데이타베이스에 대한 반복적 재스캔을 필요로 한다. 따라서 한번 수행될 때마다 매우 높은 연산 비용을 필요로 한다. 그러나 여기서 제시하는 변형된 저장 프로시저 모델은 전체 데이타베이스를 스캔하지 않고 단지 새로 삽입된 트랜잭션에 대해서만 연산을 하고 이를 이전의 연산 결과에 합산하는 점진적 방법을 채택한다. 트리거는 새로운 트랜잭션이 발생했을 때 이 새로 발생한 트랜잭션을 매개 변수로 이 논문에서 제시하는 변형된 프로시저를 호출하도록 하는 역할을 담당한다. 이를 정형화하여 표현하면 다음과 같다.

Association_Rule:=Incremental_Recomputation_
Trigger(Insert(New_TR));

여기서 점진적 갱신 프로시저의 처리 절차를 개시하도록 지정하는 트리거에 대한 구조는 다음과 같다.

```
CREATE TRIGGER Incremental_Recomputation_Trigger
   AFTER INSERT ON Tran_DB
REFERENCING NEW as New_TR
BEGIN
   New_Designed_Stored_Procedure( :New_TR );
END;
```

그림 2 연관규칙 탐사 트리거 템플릿

이와 같은 점진적 연산 전략은 데이타베이스에 대한 스캔을 최소화하면서 효과적으로 연관규칙을 탐사할 수 있도록 한다. 트리거와 결합하기 위한 변형된 저장 프로시저의 점진적 갱신 기법에 대한 자세한 기술은 다음의 4장에서 논의한다.

4. 점진적 갱신 연산의 정형 모델

앞에서 데이타베이스 상태 변화가 빈번히 발생하는 동적 환경 하에서 트리거를 이용한 실시간 온라인 연관규칙 탐사 기법으로서 트리거와 변형된 저장 프로시저의 결합 모델이 가장 타당한 기법임을 제시하였다. 이 장에서는 [17]에서 제시한 바와 같이 이 모델의 핵심인 변형된 저장 프로시저를 이용한 점진적 연관규칙 탐사 연산의 정형적 검증을 통해 제안 모델의 타당성을 검토한다.

4.1 후보항목 및 빈발계수의 점진적 갱신 모델

연관규칙 탐사에 대한 연산비용 문제에 있어 가장 핵심적 부분은 후보 항목의 생성 부분이다. 이는 후보 항목 생성을 위해 데이타베이스 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 데이타베이스를 대상으로 연관규칙 탐사를 수행하고 있을 뿐 끊임없이 변화하는 동적 환경 하에서 변경된 상

태를 즉시 연관규칙 탐사 과정에 반영할 수 있는 방안을 전혀 고려하고 있지 않기 때문이다. 따라서 이와 같은 문제를 해결하기 위해 이 논문에서는 새로운 트랜잭션 데이타의 삽입 즉시 새로 삽입된 데이타로부터 후보 항목을 생성하고 이를 연관규칙 탐사 과정에 반영할 수 있는 점진적 후보 항목 갱신 모델을 기술한다. 이를 위해 먼저 빈발계수를 갖으면서 k개의 단일 항목들의 조합으로 구성되는 k 후보 항목에 대한 정의를 내리면 다음과 같다.

[정의 1] 빈발계수를 포함하는 k 후보 항목 집합 I^{k+1}

후보 항목은 k개의 단일 항목으로 구성되는 후보 항목과 1개의 빈발 계수로 구성된다. 항목의 크기를 결정하는 변수를 k 라하고 항목 집합 내에 최대로 포함될 수 있는 항목의 수를 z라 할 때 k 크기의 후보항목 집합 I^{k+1} 의 의미는 다음과 같다.

$$I^{k+1}=\{(i_l^k, freq\_count_l)|\forall o,p,1\le o,p\le z, o\neq p\Rightarrow i_o^l\neq i_p^k \wedge 1\le l\le z\wedge z=_{Card(Max\_Tr\_Size)}C_k\}$$

여기서 항목 집합 I 의 카디널리티를 $k+1$ 로 잡은 이유는 k 개의 단일 항목들이 모여 하나의 k 항목을 구성하고 여기에 다시 하나의 빈발계수인 $freq\_count$ 이 포함되기 때문에 항목 크기 k 에 빈발계수 1이 추가되기 때문이다. 아울러 첨자 o,p 및 l 은 항목 내 임의의 후보 항목을 지정하는 첨자이며 $Card(Max\_Tr.\_Size)$ 는 데이타베이스 내에 가장 큰 트랜잭션이 포함하는 단일 항목들의 수를 의미한다. 그리고 k 항목에 포함될 수 있는 최대한의 항목 수는 이 최대 크기의 트랜잭션에 포함된 단일항목의 개수에서 k개의 단일 항목을 뽑아 조합할 수 있는 수인 $_{Card(Max\_Tr\_Size)}C_k$ 를 넘지 못한다.

정의 1 에서 k 항목 집합의 원소들은 모두 빈발 계수를 포함하도록 구성되었음을 알 수 있다. 이 k 후보 항목 집합의 정의를 바탕으로 후보항목 집합에 대한 정의를 내리면 다음의 정의 2와 같다.

[정의 2] 후보 항목 집합 $Cand\_I$

후보 항목 집합은 k 항목 집합들의 합집합으로 정의하며 다음과 같은 정형의미를 갖는다.

$$Cand\_I=\{\bigcup_k I^k | 1\le k\le Card(Max\_Transaction\_Size)\}$$

여기서 k 크기의 항목 집합 I^k이 의미하는 바는 정의 1과 같으며 k의 최대 크기는 트랜잭션 데이타베이스 내에서 단일 항목을 가장 많이 갖는 트랜잭션의 단일 항목 수를 초과 할 수 없다.

이 정의는 데이타베이스의 일정 부분만을 대상으로 하지 않고 전체 데이타베이스를 대상으로 하는 정의이다. 그러나 점진적 연산을 수행하기 위해서는 이렇게 전체 데이타베이스를 대상으로 하는 후보 항목 집합은 이용하지 않는다. 왜냐하면 이 논문에서 제시하는 점진적 후보 항목 갱신기법은 전체 데이타베이스를 대상으로 하지 않고 새로 삽입되는 트랜잭션만을 대상으로 점진적 연산을 수행하기 때문이다.

한편 동적 환경에서 트리거를 이용한 점진적 연관규칙 탐사 연산은 연관규칙 탐사 과정에 포함되는 후보 항목 집합의 생성 및 관리, 빈발 항목 집합의 생성 및 관리 그리고 연관규칙의 탐사 및 관리에 이르는 전 과정을 점진적 연산의 대상으로 하지는 않는다. 왜냐하면 연관규칙을 탐사하기 위한 전체 과정 속에서 빈발 항목 집합은 후보 항목 집합에서 빈발계수 값이 최소 지지도를 초과하는 후보 항목들을 선택적으로 포함하도록 하는데 동적 환경에서는 최소 지지도를 만족하는 빈발계수의 값이 고정된 값을 갖지 않고 새로운 트랜잭션 데이타의 삽입에 따라 계속해서 변화하기 때문이다. 따라서 이 논문에서는 트랜잭션 데이타의 삽입에 대응한 점진적 연산은 후보 항목의 생성과 생성된 후보 항목의 빈발계수를 유지하는 것으로 한정한다.

점진적 연산은 시간의 흐름에 따라 변화하는 차분을 포착해야 하기 때문에 시점 인덱스를 포함해야 한다. 따라서 전체 데이타베이스를 대상으로 하는 후보 항목 집합에 대해 정의를 내리고 있는 정의 1과 정의 2를 점진적 연산에 대응시키기 위해 시점 인덱스를 적용하여 확장 정의하면 다음의 정의 3과 같다.

[정의 3] 시점 인덱스 t 를 포함하는 k 크기 후보 항목 집합 및 전체 항목 집합

t시점에서의 k_후보항목집합 $_tI^{k+1}$
$=\{(_ti_l^k, freq\_count_l)|\forall o,p,1\le o,p\le z, onot$
$=p\Rightarrow {}_ti_o^k\neq {}_ti_p^k\wedge 1\le l\le z\wedge z$
$=_{Card(Max\_Tr\_Size)}C_k\}$

t시점에서의 후보항목집합 $_tCand\_I$
$=\{\bigcup_k {}_tI^k | 1\le k\le Card(_tMax\_Tr\_Size)\}$

여기서 시점 인덱스 t 를 갖는 k 항목 집합의 의미는 최초 시점에서부터 t 시점까지 트랜잭션 데이타베이스에서 나타난 k 크기 항목들이 모두 누적되어 있음을 의미하고 t 시점의 후보 항목 집합은 t 시점까지의 모든 후보 항목들을 포함하는 집합임을 의미한다. 따라서 새로운 트랜잭션 데이타가 삽입될 경우 정의 3 에서 규정하고 있는 후보 항목 집합에 새로 삽입된 트랜잭션 데이타로부터 발생하는 차분만을 합산하면 된다. 이 때 새로 삽입된 트랜잭션에 의해 발생하는 차분은 두 가지 유형을 가질 수 있다. 첫 번째 유형은 이전 시점까지

누적되어 있는 후보 항목집합에 존재하는 항목들로 구성되는 항목 집합 유형이고 두 번째 유형은 이전 시점까지 누적되어 있는 후보 항목 집합에 존재하지 않는 항목들로 구성되는 항목 집합 유형이다. 이 두 항목 집합 사이에는 서로소 관계를 갖고 있으며 모두 새로 삽입된 트랜잭션으로부터 유도되는 항목들로 구성된다. 다음 정의 4는 후보 항목 집합을 정의한다.

[정의 4] t 시점에서의 갱신 차분 ${}_t\Delta_U$, 삽입 차분 ${}_t\Delta_A$ 및 차분 집합 ${}_t\Delta$

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

$$\text{차분집합 } {}_t\Delta = \{ {}_t i_o^{k_p} \mid \forall o, p, 1 \le o, p \le {}_{Card(New\_TR)}C_k, o \neq p \Rightarrow {}_t i_o^k \neq {}_t i_p^k \wedge 1 \le k \le Card(New\_TR) \}$$

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

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

이와 같은 정의들을 토대로 이 논문에서 제시하고 있는 후보 항목의 점진적 갱신 연산을 정의하면 다음의 정의 5와 같다.

[정의 5] 후보 항목의 점진적 갱신

후보 항목의 점진적 갱신은 이전 시점의 후보항목 집합에 새로 삽입된 트랜잭션 데이타로부터 생성되는 후보 항목 집합의 차분을 합산하는 것으로 정의하면 다음과 같다.

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

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 연산에 의해 유지되는 후보 항목 집합의 상태는 동적 환경에 대응해서 항상 최신의 상태를 반영할 수 있는 빈발항목 추출이 가능하도록 완전한 후보 항목집합을 유지한다. 다음의 정리 1 은 후보 항목 집합의 점진적 갱신 연산이 갖는 완전성을 정리한 것이다.

[정리 1] 후보 항목 집합에 대한 점진적 갱신 연산의 완전성

후보 항목 집합의 점진적 갱신 연산은 항상 데이타베이스의 최신 상태를 반영할 수 있는 빈발항목 추출이 가능하도록 완전한 후보 항목집합을 유지한다.

[증명]

t 시점에서의 후보 항목 집합은 t 시점에 삽입된 트랜잭션의 단일 항목들을 조합해서 얻을 수 있는 모든 항목의 집합이다 (정의 3). 이 때 t 시점에 생성된 후보 항목들은 갱신 차분과 삽입 차분으로 구성되며 이들 사이에는 서로소 관계에 있다 (정의 4).

한편 이 논문에서 제안하는 점진적 후보항목 갱신 기법에서는 Apriori 알고리즘에서 적용하는 항목의 축약 (pruning) 연산을 적용하지 않기 때문에 $t-1$ 시점까지 누산되어 있는 후보 항목집합과 $t-1$ 시점의 데이타베이스 상태에 Apriori 알고리즘을 적용하여 얻을 수 있는 후보 항목 집합 사이에는 다음과 같은 관계가 성립한다.

$$ {}_{(t-1)}Cand\_I \supseteq {}_{(t-1)}Cand\_Apriori \quad (1)$$

다음으로 t시점의 트랜잭션에 의해 생성된 후보 항목들은 t 시점에 Apriori 알고리즘을 적용하여 얻은 후보 항목과 $t-1$시점에 Apriori 알고리즘을 적용하여 얻은 후보 항목의 차집합에 대해 다음과 같은 관계를 만족한다.

$$ {}_tCand\_I \supseteq \{ {}_tCand\_Apriori - {}_{(t-1)}Cand\_Apriori \} \quad (2)$$

따라서 (식 1)과 (식 2)에 의해 후보 항목의 점진적 갱신 연산은 Apriori 알고리즘에서 적용하는 모든 의미들을 내포하는 연산의 완전성을 보장한다. ■

5. 점진적 갱신 연산을 위한 구현 모델

4 장에서 후보 항목 집합에 대한 점진적 갱신 모델을 기술하였다. 이 장에서는 이 점진적 갱신 모델을 구현하기 위한 구현 모델을 기술한다. 실제로 이 장에서 기술하는 구현 모델은 오라클 8에 통합되어 있는 트리거와 저장 프로시저 언어인 PL/SQL을 이용하여 구현하고 있으며 따라서 구현 모델 역시 이들 도구를 기반으로 한다.

5.1 점진적 후보 항목 갱신 시나리오

트리거를 이용한 점진적 후보 항목 갱신 기법의 주요한 특성은 트리거를 이용해 새로운 트랜잭션의 발생 즉시 이를 후보 항목에 반영할 수 있다는 점과 전체 데이타베이스에 대한 스캔 없이 단지 새로 발생한 트랜잭션에 대응한 연산만으로 전체 데이타베이스를 스캔한 것과 동일한 효과를 얻을 수 있다는 점이다. 또한 트리거에 의해 촉발되는 점진적 후보 항목 갱신 프로시저는 데이타베이스 시스템 주소공간에서 수행되는 저장프로시저를 이용하기 때문에 보다 높은 성능 특성을 얻을 수 있다. 이 트리거를 이용한 점진적 후보 항목 갱신 기

법의 수행 절차를 요약하면 다음과 같다.

단계 1. 새로운 트랜잭션의 발생과 발생된 트랜잭션 데이타의 데이타베이스 삽입 연산 수행

단계 2. 트랜잭션 데이타의 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 데이타를 이용하여 조합 가능한 후보 항목 생성

단계 3.2. 생성된 후보 항목들을 삽입차분과 갱신 차분으로 분류

단계 3.3. 삽입 차분의 삽입 연산 수행

단계 3.4. 갱신 차분의 갱신 연산 수행

단계 4. 트랜잭션 완료

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

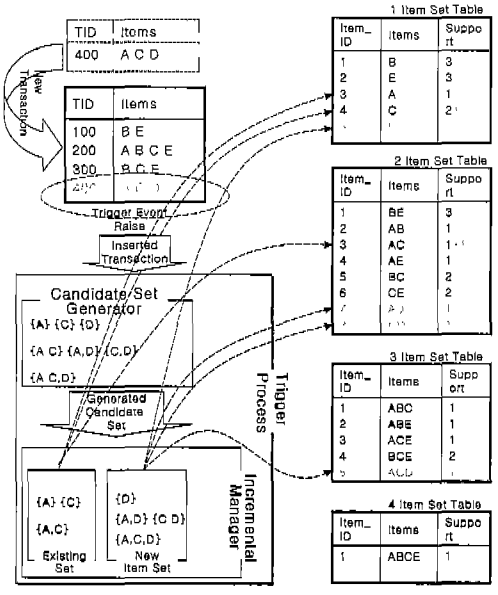

그림 3 트리거를 이용한 후보항목집합의 점진적 관리 시나리오

그림 3에서 볼 수 있는바와 같이 새로운 트랜잭션의 삽입이 발생하게 될 경우 트리거 사건이 발생하게 된다. 트리거 사건에 의해 기동되는 트리거는 먼저 후보항목 집합 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 데이타베이스에 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 데이타베이스를 대상으로 하는 기존의 연관규칙 탐사 기법과는 달리 제안하는 방법은 트랜잭션 단위로 후보 항목에 대한 상태 변화를 관리하는 점진적으로 후보 항목을 관리하는 기법이다. 이때 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 항목들의 차분들인 삽입 차분과 갱신 차분으로 분류하는 단계 3.2 이다. 왜냐하면 새로 삽입된 트랜잭션 데이타를 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 주기억장치상에 유지하면서 해쉬 기법과 결합하여 데이타베이스에 대한 접근을 최소화 할 수 있도록 하는 전략을 채택한다.

5.2 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[18].

이와 같은 오라클 트리거를 이용하여 연관규칙 탐사를 위해서는 두 개의 트리거가 쌍으로 정의되어야 한다. 왜냐하면 실제 데이타베이스 삽입 연산은 행(row) 단위로 이루어지지만 연관규칙 탐사는 트랜잭션 단위로 수행되어야 하기 때문에 트리거의 적용에 있어 실행 단위가 달라지기 때문이다. 따라서 행 단위 트리거를 이용하여 트랜잭션 내에서 데이타베이스에 삽입되는 행들을 수집한다. 그런 다음 트랜잭션 단위 트리거에서는 행 단위 트리거에서 준비된 데이타를 파라메터로 하여 점진적 후보 항목 갱신을 위한 연산을 호출한다. 다음은 행 단위로 삽입되는 데이타를 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.Item )
END;
```

다음으로 후보 항목의 점진적 갱신을 수행하는 저장 프로시저를 호출하는 트랜잭션 단위 트리거의 정의는 다음과 같다.

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

여기서 두 트리거의 사건 부분을 집중할 필요가 있다. 행 단위 트리거가 트리거 수행 시점으로 BEFORE로 한 반면 트랜잭션 단위 트리거는 수행 시점으로 AFTER를 지정하고 있다. 이는 점진적 갱신을 수행하기 이전에 데이타베이스에 삽입되는 모든 행들을 수집하기 위해 행 단위 트리거가 트랜잭션 단위 트리거보다 높은 우선순위를 갖도록 지정하기 위해 설정한 것이다. 이렇게 할 경우 행 단위 트리거는 데이타베이스에 행을 삽입하기 전에 트리거 연산을 먼저 수행하고 난 다음에 데이타베이스 연산을 수행한다. 반면 트랜잭션 단위 트리거는 행단위 트리거에 의한 행들의 수집 연산을 모두 수행하고 난 후 점진적 후보항목 갱신연산 저장 프로시저를 호출함으로써 트리거 연산을 수행한다.

5.3 후보항목 생성 알고리즘

이 절에서는 5.1절의 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 $2^n - 1$ 개의 항목으로 조합된다. 다시 말해 한 트랜잭션 내에서 생성 가능한 후보 항목의 수는 해당 트랜잭션 내 단일 항목의 수에 따라 지수 적으로 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 15개 이하로 제한한다. 이는 구현 시스템의 성능과 메모리 등의 요건에 따라 변경될 수 있으나 하나의 트랜잭션을 처리하는데 최대 20초를 초과하지 않는 범위로 한정하기 위해서이다.

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}를 포함하는 항목이 여덟번 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 앞에서 제시한 최대 트랜잭션 크기의 중간 크기인 8을 기점으로 8보다 큰 트랜잭션에 의해 생성되는 후보 항목들은 트랜잭션 크기 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 다음의 그림 4와 같다.

그림 4에서 후보항목을 저장하기 위한 버퍼 변수들은 모두 1차원 배열로서 이는 오라클 8의 PL/SQL에서 제공하는 복합 데이타 타입이 다차원 배열을 허용하지 않기 때문에

```
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;
        buf2_1(jmp_2) := New_tr_buf(cnt_1);
        buf2_2(jmp_2) := New_tr_buf(cnt_2);
        for cnt_3 in cnt_2+1..tr_size loop
                  --
                for cnt_8 in cnt_7+1..tr_size loop
                  jmp_8 := jmp_8 + 1;
                  buf8_1(jmp_8) := New_tr_buf(cnt_1);
                    --
                  buf8_8(jmp_8) := New_tr_buf(cnt_8);
                  for cnt_9 in cnt_8+1..tr_size loop
                    jmp_9 := jmp_9 + 1;
                    buf9_8(jmp_9) := jmp_8;
                    buf9_9(jmp_9) := New_tr_buf(cnt_9);
                    for cnt_10 in cnt_9+1..tr_size loop
                              --
                          for cnt_15 in cnt_14+1..tr_size loop
                              --
                          end loop;                          15
                      -
                    end loop;                        -- 10
                  end loop;                            09
                end loop;                        -- 08

      end loop;                        -- 03
    end loop;                        -- 02
  end loop;                        -- 01
end Make_Candidate_Item_set;
```

그림 4 트랜잭션 단위 후보 항목 생성 알고리즘

취한 것이다. 알고리즘에서 New_tr_buf는 트랜잭션에 포함된 단일 항목들을 유지하는 1차원 배열이다.

5.4 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 항목의 분류 과정은 알고리즘의 성능을 결정하는 가장 중요한 요소이다. 왜냐하면 앞 절에서 언급한 바와 같이 후보 항목 생성기에 의해 생성되는 후보 항목들이 대규모로 발생하게 되며 이와 같이 대량으로 발생하는 후보 항목들을 지지도 갱신 대상 항목과 새로 삽입할 항목으로 분류하기 위해서는 생성된 후보 항목이 이미 존재하고 있는 항목인지 아니면 존재하지 않는 항목인지를 결정해야 하기 때문이다. 이 때 이와 같은 사항을 결정하기 위해서는 저장되어 있는 후보 항목들에 대한 검색이 필요하게 되며 이를 위해서는 저장 데이타 상태로 유지되는 후보항목들에 대한 스캔 연산을 해야 한다. 이와 같은 연산을 수행하기 위해 최악의 경우 $2^{15}-1$ 번의 데이타베이스 스캔을 필요로 할 수 있다. 이는 성능에 치명적 영향을 끼칠 수 있는 문제이다. 다음의 그림 5는 분류기에 대한 구조를 보여준다.

```
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 then
            assign new_item_id to cnt th item;
            make insert item for cnt th item;
        elsif classifier.tag = existing then
            make update item for classifier.id;
        elsif classifier.tag = conflicting then
            assign new item id to cnt th item;
            add new_item_id into conflict set;
            make insert item for cnt th item;
        end if;
    end loop;
end;
```

그림 5 후보 항목 분류기 골격

그림 5의 분류기에서 Get_From_Hash 함수는 해당 항목이 해쉬상에 존재하는지 여부를 결정하기 위한 함수이다. Get_From_Hash 함수에서 반환하는 classifier는 id와 tag로 구성되는 레코드 구조이다. classifier의 id는 항목 식별자이거나 또는 충돌집합의 색인 값을 나타낸다. 그리고 tag는 new, existing, conflicting 세 가지 값 중 하나를 가질 수 있으며 이 tag 값에 따라 해당 항목의 처리 방법이 결정된다. 따라서 Get_From_Hash 함수의 classifier 결정 과정이 알고리즘의 성능을 결정하는 중요한 특성을 갖는다. 다음의 그림 6은 Get_From_Hash함수의 구조이다.

여기서 Get_From_Hash함수는 해쉬 테이블의 상태에 따라 다섯 가지 상태로 나누어 평가한 후 이를 다시 New, Existing, conflicting의 세 가지 상태로 결정한다. 이 중 New 상태(⑤)는 Hash 테이블 상에 존재하지 않는 항목일 경우가 되며, Existing은 충돌집합 상에서(③)와 충돌이 발생하지 않은 상태(①) 두 가지를 모두 고려한다. Conflicting 역시 충돌집합(④)과 충돌이 존재하지 않는 상태(②) 모두를 고려해야 한다. 여기서 주목할 점은 해쉬키를 결정하는 방법에 따라 충돌의 가능성을 최소화 할 수 있다는 점이다. 다음의 그림 7은 충돌을 최소화하기 위한 해쉬 키 생성 알고리즘이다.

그림 7에서 볼 수 있는 바와 같이 해쉬 키는 항목에 참여하는 모든 단일 항목들의 값을 곱한 값과 모든 단일 항목의 값을 더한 값을 합산해서 생성한 후 이를 다시 주어진 상수를 이용하여 나머지 연산을 취한다. 이 때 상수의 크기 값은 $2^{31}-1$값을 갖는다. 이는 이 값이 소수이기 때문에 이 값에 의해 나누어진 나머지의 중복을 최소화 할 수 있으며 최대 15개의 네자리 수 항목들

```
Function Get_From_Hash ( location number) return classified_buf
is
begin
    hash_key_value := Make_Hash_Key( cnt );
    if hash_table.exists(hash_key_value) then
       if hash_table(hash_key_value) > 0 then
         select item_1, ... , item_n into itm_1, ... , itm_n  from cand_n
         where item_id = hash_table(hash_key_value);
         compare selected item with item_buf(location);
         if compared result is true then                                    ------   --------  ----- ①
             classifier.id := hash_table(hash_key_value);     classifier.tag := existing;
         else                                                               ------   --------  ----- ②
             add hash_table(hash_key_value) to conflict set;
             classifier.id := new_conflict_set_id;              classifier.tag := conflicting;
          end if;
       else
          gathering conflicted item_id into gathered_id;
          candidate set database scan for gathered item_ids
          compare each gathered_item with cnt th item;
          if existing matched item id then                                  ---------- ---------- ③
             classifier.id := matched_item_id;          classifier.tag := existing;
          else                                                              ---------- ---------- ④
             classifier.id := hash_table(hash_key_value);
             classifier.tag := conflicting;
          end if;
       end if;
    else                                                                    ---------- ---------- -- ⑤
       calssifier.id := 0;              classifier.tag := new;
    end if;
end;
```

그림 6 Get_From_Hash 함수 구조

```
Function Make_Hash_Key( Item_Pos Number ) Return Number
is
    Multiplex       number;
    Summ            number;
    Result          number(11);
begin
    Multiplex := Buf_n_1(Item_Pos) × Buf_n_2(Item_Pos) ×, ..., × Buf_n_n(Item_Pos);
    Summ :=  Buf_n_1(Item_Pos) + Buf_n_2(Item_Pos) +, ..., + Buf_n_n(Item_Pos);
    select mod((Multplex + Summ), 2147483647 ) into Result from dual;
    return Result;
end;
```

그림 7 해쉬 키 생성 알고리즘

의 값이 곱해져야 하기 때문에 충분히 큰 수를 지원할 수 있도록 하기 위해서 선택된 값이다.

앞에서 해쉬 테이블 상에서 충돌이 일어났을 경우가 발생했을 때 이에 대응하는 방법으로서 충돌집합을 운영하는 알고리즘을 제시하였다. 이는 충돌에 대응한 성능을 최대화하기 위한 조치로서 다음의 그림 8에서 제시하고 있는 시나리오에 따라 동작한다.

그림 8은 2_항목집합에 대한 해쉬 테이블 및 충돌집합 운영 시나리오이다. 그림 8에서 볼 수 있는 바와 같이 해쉬 테이블에 있는 항목 식별자 중 음수를 갖는 식별자는 충돌집합의 식별자로 사용되며 그렇지 않은 식별자들은 충돌이 발생하지 않은 식별자들이다. 이들 구조들은 모두 메인 메모리 상에서 운영함으로써 디스크의 접근을 최소화하여 성능의 증가를 꾀하고 있다.

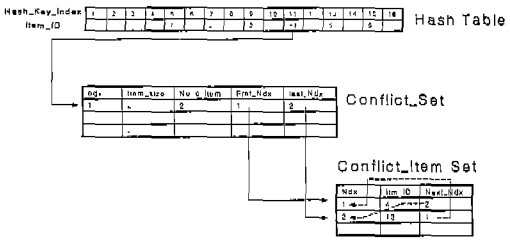

그림 8 해쉬를 이용한 후보 항목 식별 시나리오

6. 실험 및 성능 평가

이 논문에서 트리거를 이용한 점진적 연관규칙 탐사 기법의 성능 평가를 위한 데이타 집합은 다음과 같은

특성을 갖고 있다.

전체 트랜잭션 수	전체 항목수	트랜잭션 당 평균 항목수	빈발항목 집합			
			패턴의 총 수	패턴의 평균길이	신뢰도	신뢰도 분산
20,000	2,000	15	4,000	8	75%	10%

이와 같은 데이타를 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

6.1 제안 알고리즘의 성능 특성

이 논문에서 제시하는 트리거와 저장 프로시저를 이용한 점진적 후보항목 갱신 기법은 안전한 데이타베이스를 구축한 상태에서 탐사하는 것이 아니라 트랜잭션 데이타의 삽입이 발생할 때마다 동적으로 재구성을 해야 한다. 따라서 빈 트랜잭션 데이타베이스에서부터 시작해서 내장 프로시저를 이용한 연관규칙 탐사를 수행한 데이타베이스와 동일한 형태의 데이타베이스가 형성될 때까지 연속적으로 데이타를 삽입하면서 실험을 수행하였다. 이 때 트랜잭션 항목의 크기에 따라 실제 데이타가 삽입되면서 후보항목을 생성하는 트리거가 완전히 수행될 때까지의 시간에 대한 평균치를 측정한 결과가 다음의 그림 9에 나타나 있다. 그림에서 보는 바와 같이 트랜잭션의 크기가 10을 초과하면서부터 연산 부하가 현격히 증가함을 발견할 수 있다.

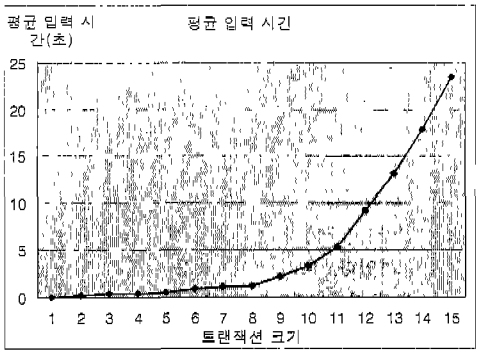

그림 9 항목당 평균 입력 시간

다음의 그림 10은 트랜잭션의 발생 건수에 따른 삽입에 걸린 시간을 측정한 것으로서 트랜잭션의 수가 증가할수록 삽입시간에 대한 편차가 커지기는 하지만 대체로 증가하는 추세를 보이고 있음을 확인할 수 있다. 그림 10에서 보이는 시간 편차의 증가는 트랜잭션 수의 증가에 따른 데이타베이스 크기의 증가로 인해 발생하는 I/O 부하의 증가와 기타 다른 프로세스에 의해 받는 영향들이 복합적으로 나타난 것이라 해석할 수 있으며 아울러 트랜잭션의 항목수가 10을 초과할 때 발생하는 연산부하의 가중에 의해 나타나는 현상으로 풀이할 수 있다.

아울러 그림 11은 트랜잭션 삽입에 따라 나타나는 평균 트랜잭션 크기를 나타낸다. 그림 10과 그림 11을 같이 놓고 비교 해 보면 그림 11의 트랜잭션 크기 증감에 따라 그림 10의 트랜잭션당 평균 입력시간이 민감하게 반응함을 알 수 있다. 이는 데이타베이스 내 데이타량의 증가와 트랜잭션 크기가 시스템 성능에 지대한 영향을 미치는 것을 명료하게 보여주는 것이다.

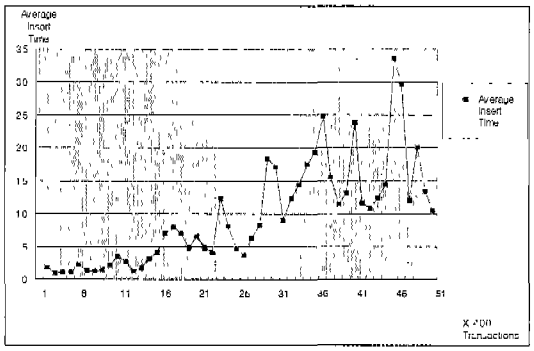

그림 10 트랜잭션 당 평균 입력 시간

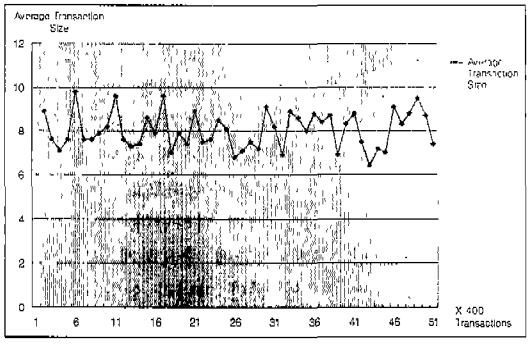

그림 11 평균 트랜잭션 크기의 분포도

6.2 저장 프로시저 방법 비교

이 절에서는 [2]에서 제시한 저장 프로시져를 이용한 연관규칙 탐사 방법과 이 논문에서 제시하는 방법 사이

의 성능 특성에 대한 비교 분석을 수행한다. 이 논문에서 제시하는 방법은 기존의 연관규칙 탐사 기법들이 채택하고 있는 특정 시점에 전체 데이타베이스를 대상으로 연관규칙 탐사를 수행하는 방법과는 기본적으로 접근 형태가 다르기 때문에 기존 기법들과 정량적 비교를 하는 것은 타당하지 않다. 그러나 이 논문에서 제시하고 있는 방법이 나타내는 성능 특성과 기존 방법이 나타내는 성능 특성 사이의 차이점에 대한 분석은 이 논문에서 제시하고 있는 방법의 타당성 검증측면에서 필요한 사항이라 할 수 있다. 따라서 이 절에서는 이 논문에서 제시하는 기법과 기존 방법 사이의 동작 특성에 대한 분석을 통해 기존 방법과의 차이점을 제시한다.

비교를 위한 기본 요소로 데이타 셋은 동일한 데이타 셋을 사용하였으며 최소 지지도는 1%로 설정 하였다. 지지도를 1%로 설정한 이유는 이 논문에서 제시하는 방법이 후보항목을 관리하는데 있어 지지도 적용 없이 모든 가능한 후보 항목들을 처리하기 때문에 가능한 적은 지지도를 설정하는 것이 동일한 조건을 형성할 수 있기 때문이다. 이와 같은 조건 하에서 실험 결과가 다음의 그림 12에 나타나 있다. 그림 12는 연관규칙 탐사 과정에서 후보 항목 생성에 드는 비용을 시간적으로 측정한 것이다.

그림에서 볼 수 있는 바와 같이 기존의 방법은 연산 수행 시간이 매우 높게 나타나며 트랜잭션의 양이 증가함에 따라 소모하는 연산 시간이 선형적인 증가를 나타난다. 이는 기존의 방법이 연관규칙의 탐사 시점에 데이타베이스 전체에 대한 스캔 연산을 반복적으로 수행해야 하기 때문에 나타나는 결과이며 연산 시간이 선형적으로 증가하는 이유는 증가하는 트랜잭션의 양이 일정량씩 증가하도록 실험 환경을 구성하였기 때문으로 풀이할 수 있다. 반면 이 논문에서 제시하는 방법은 후보항목 갱신 과정에서 데이타베이스 전체를 대상으로 스캔 연산을 하지 않고 삽입된 트랜잭션에 한해서 대응하는 항목 단위의 스캔 연산만을 수행하므로 수행결과에 대한 성능 특성이 매우 상이하게 나타난다. 특히 이 논문에서 제시하는 방법에서 성능에 주요하게 영향을 미치는 요인은 트랜잭션의 양 뿐만 아니라 삽입되는 트랜잭션의 크기(트랜잭션 내에 포함된 단일 항목의 수)도 매우 중요한 영향을 주고 있음을 확인할 수 있다.

그림 12는 [2]에서 제시한 저장 프로시져를 이용한 방법과 이 논문에서 제시한 방법 사이의 성능 특성을 비교하였다. 그림 12의 실험 결과에서 중요한 차이점은 [2]의 방법에 대한 실험 결과는 전체 수행 시간을 측정하여 기록 한 것이지만 이 논문에서 제시하고 있는 방법은 평균 수행 시간을 기록 한 것이라는 점이다. 이는 이 논문에서 제시하고 있는 방법이 트랜잭션 삽입이 발생할 때 마다 삽입되는 트랜잭션에 대응해서 연산을 수행하기 때문에 이에 대한 전체 수행 시간을 정량적으로 평가하는 것에 대해 의미를 부여하기 힘들기 때문이다. 이와 같이 이 절에서 수행하는 기존 방법과의 비교는 두 방법 사이의 우열을 가르기 위한 비교가 아니라 서로 어떻게 다른 동작 특성을 나타내는지를 실험적으로 고찰하기 위한 것이다. 실험 결과에서 확인할 수 있는 바와 같이 두 방법 사이에는 정량적으로 비교할 수 있는 요소가 극히 제한적 관점에서만 의미를 가질 수 있음을 확인할 수 있다.

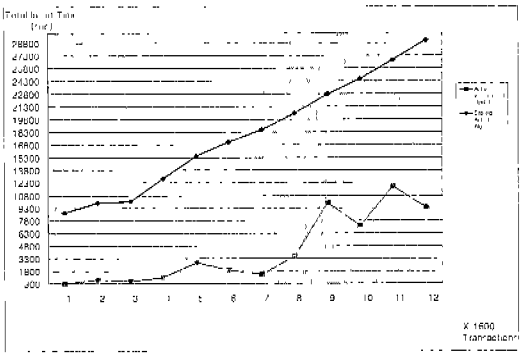

그림 12 후보항목 생성 비용 비교

7. 결 론

이 논문에서는 연관규칙 탐사를 수행하는 데 있어 데이타베이스 트리거를 이용하는 새로운 기법을 제안하고 이의 구현 및 실험을 통해 제안 모델의 특성을 분석하였다. 이 논문에서 제안한 모델은 트랜잭션의 삽입과 동시에 실시간 적으로 후보항목의 지지도를 갱신할 수 있도록 하는 모델이다. 이를 위해 이 논문에서는 트랜잭션에 대응한 트리거를 정의하였으며 아울러 점진적 후보항목 갱신연산을 수행하는 프로시저 부분은 저장 프로시저를 이용하였다. 따라서 이 논문에서 제안하는 트리거와 점진적 갱신 기법을 이용하는 후보항목 갱신 기법은 [2] 및 [4]에서 제시하고 있는 저장 프로시저를 이용한 연관규칙 탐사 기법의 장점을 계승하면서 새로 발생하는 트랜잭션에 대응하여 후보항목 및 항목의 지지도를 실시간으로 유지할 수 있게 하였다. 아울러 이 논문에서의 제안하는 방법과 [2]의 방법에 대한 비교 실험을 통해 이 논문에서 제안하는 방법의 성능 특성을 좀더 객관적으로 평가할 수 있었다.

특히 이 논문에서 제안하고 있는 기법은 한 트랜잭션 내에서 잠재적으로 발생할 수 있는 대규모 후보 항목들을 모두 고려해야하는 매우 높은 연산 부하를 갖는다. 이와 같이 대규모연산이 빈번히 발생하는 문제를 해결하기 위해 메모리 상에 해쉬와 충돌집합을 유지함으로써 디스크 연산을 최소화 할 수 있는 점진적 갱신 기법을 개발하였다. 또한 후보 항목 갱신 알고리즘은 트리거에 의해 기동됨으로써 트랜잭션 발생에 대해 자동으로 빈발계수를 갖는 후보 항목집합을 생성, 유지할 수 있게 된다. 따라서 사용자의 개입을 최소로 하면서도 실시간적으로 연관규칙을 탐사할 수 있도록 한다.

또한 이 논문에서 제안한 기법은 6장 실험에서 고찰한 바와 같이 최대 35초 내에 트랜잭션에 대응한 모든 연산들을 완료 할 수 있으며 적용 가능한 최대 크기의 트랜잭션인 15항목 트랜잭션에 대해서도 평균 23초 정도의 시간 안에 처리할 수 있다. 따라서 트랜잭션이 빈번히 발생하는 동적 환경 하에서 중소규모의 트랜잭션들에 대한 실시간 후보항목집합을 유지하게 함으로써 시간대별 또는 항목의 특수성을 반영한 패턴 분석을 수행할 수 있게 하였다.

한편 이 논문의 기본 목적이 트리거와 마이닝의 결합 가능성을 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] 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.

[4] 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.

[5] 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.

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

[7] Jennifer Widom, Stefano Ceri, "Chapter 1 : Introduction to Active Database Systems," Active Database Systems(Triggers and Rules For Advanced Database Processing), Morgan Kaufmann Publishers, 1996.

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

[9] 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.

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

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

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

[13] Spyros Potamianos, Michael Stonebraker, "Chapter 2 : The POSTGRES Rules System," Active Database Systems (Triggers and Rules For Advanced Database Processing), Morgan Kaufmann Publishers, 1996.

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

[15] 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.

[16] C. Janiolo, S. Ceri, C. Faloutsos, R. T. Snodgrass,

V. S Subrahmanian, R. Zicari, "Chapter 4 : Design Principles for Active Rules," Advanced Database Systems, Morgan Kaufmann Publishers, 1997.
[17] Y. J Lee, S. B. Seo, K. H. Ryu, "Discovering Temporal Relation Rules from Temporal Interval Data," to be submitted in KISS. 2001.
[18] Oracle 8i Development Guide : PL/SQL, Oracle Press, 2000.

황 정 희

1991년 충북대학교 전산통계학과 졸업(학사). 2001년 8월 충북대학교 대학원 전자계산학과 졸업(석사). 2001년 9월 ~ 현재 충북대학교 대학원 전자계산학과 재학중(박사). 관심분야는 능동 데이타베이스, 시공간 데이타베이스, 데이타마이닝

신 예 호

1996년 군산대학교 컴퓨터과학과 졸업(학사). 1998년 충북대학교 대학원 전자계산학과 졸업(석사). 1998년 3월 충북대학교 대학원 전자계산학과 입학(박사과정). 2000년 8월 ~ 현재 충북대학교 대학원 전자계산학과 수료. 관심분야는 능동 데이타베이스, 시간 데이타베이스, 공간 데이타베이스, 데이타 마이닝

류 근 호

1976년 숭실대학교 전산학과(공학사). 1980년 연세대학교 산업대학원 전산전공(공학석사). 1988년 연세대학교 대학원 전산전공(공학박사). 1976년 ~ 1986년 육군군수 지원사 전산실(ROTC 장교), 한국전자통신연구원(연구원), 한국방송통신대 전산학과(조교수) 근무. 1989년 ~ 1991년 Univ. of Arizona Research Staff(TempIS 연구원, Temporal DB). 1986년 ~ 현재 충북대학교 전기전자 컴퓨터공학부 교수. 관심분야는 시간 데이타베이스, 시공간 데이타베이스, Temporal GIS, 객체 및 지식기반 시스템, 지식기반 정보검색 시스템, 데이타마이닝, 데이타베이스 보안 및 Bio-Informatics