능동 규칙 버퍼링에 관한 연구
A study on Active Rule Buffering

임채명(동원대학), 이혜명(평택공업대학), 신이균(용인송담대학/전자과)
Chae-Myung Lim(D.W.C.), Hye-Myung Lee(P.T.C.),
Yee-Kyun Shin(Y.S.C./Dept. of Electronics)

Key Words : Active Database System(능동 데이터베이스 시스템), Active Rule(능동 규칙), Rule Buffering(규칙 버퍼링)

ABSTRACT : Active database systems integrate active rules into a database system and they provide a uniform mechanism for advanced database features, including integrity constraint enforcement, derived data maintenance, triggers, protection, and others. Whenever a database state is changed, active rule(s) is(are) automatically triggered to guarantee consistency and integrity of data.

However, because conditions in triggered rules must be evaluated to decide whether they are executed or not, and execution results of rules are stored into database, active database systems continuously access database. As a result, performance of overall database systems is degraded. Therefore, their performance depends on optimization of rules, efficient condition evaluation, the number of database access and others.

To reduce frequent database access caused by consecutive trigger of rules, we propose rule buffering and its structure and show its applicable examples.

I. 서론

기존 데이터베이스 시스템은 수동적인 면을 가진다. 즉, 기존의 데이터베이스 시스템에서는 사용자의 질의나 응용 프로그램의 트랜잭션에서 명시적인 처리 요구가 있을 때에만 특정 행위가 수행되며[12], 무결성 제약 조건은 대부분 사용자의 트랜잭션에서 검사하도록 하고 있다. 그러므로 무결성 유지는 프로그래머가 책임을 지고 있으며, 무결성 제약 조건의 변경에 따른 효과적인 대응을 어렵게 하고 있어 데이터베이스의 일관성 유지가 용이하지 않다.[2] 이러한 문제점을 보완하고자 기존의 데이터베이스 시스템은 ASSERT 문이나 DEFINE INTEGRITY 문을 이용하여 무결성 유지를 지원하고 있으나 제한적으로 사용되고 있다.

최근의 데이터베이스가 취급하는 업무의 양이 방대해지고 그 복잡성이 증가하고 있어, 데이터베이스에 저장된 데이터의 일관성과 무결성 유지는 더욱 복잡해지고 어려워지고 있다. 따라서 데이터의 일관성과 무결성 유지를 위한 효율적인 방안이 필요하다. 이를 위해, 능동 데이터베이스 시스템의 연구가 활발하게 진행되어 왔다. 능동 데이터베이스 시스템은 기존 데이터베이스 시스템에서 제공하고 있는 무결성 유지 기능을 확장하여 데이터베이스 시스템에 무결성 제약 조건을 규칙(rule)의 형태로 저장해 두고 데이터베이스의 상태가 변할 때마다 그 규칙이 자동적으로 수행되게 함으로써 데이터의 일관성과 무결성을 유지하도록 한다. 그러므로 능동

데이터베이스 시스템에서는 규칙의 효율적인 관리 유지와 수행을 보장할 필요가 있다.

능동 데이터베이스 시스템은 데이터베이스에서 특정 사건(event)이 발생하면 즉, 데이터베이스의 상태가 변하게 되면 그에 따른 규칙들이 트리거된다. 이 때 트리거된 규칙에서 기술한 제약 조건(condition)을 검사하여 만족하는 경우 규칙에서 기술한 행동(action)을 수행한다. 이러한 기능을 이용하여 능동 데이터베이스 시스템은 데이터베이스의 상태가 항상 일관된 상태를 유지하도록 할 수 있다.

이와 같은 능동 기능은 규칙(rule)으로 표현하며, 규칙은 ECA(event, condition, action) 형태로 나타낸다. 이러한 ECA는 사건(event)의 발생에 따라 조건(condition)을 검사하고, 조건이 만족되면 정해진 행동(action)이 이루어진다.[8, 9]

사건의 발생은 그 사건과 관련된 규칙을 트리거하며, 이와 같은 사건의 발생 요인은 데이터의 삽입(insert), 수정(update), 삭제(delete), 검색(retrieve)등과 같은 일반적인 데이터베이스의 연산이나 일정 시간의 반복(repeated time), 절대 시간(absolute time), 주기(periodic interval)등을 나타내는 시간 연산자 등이 있다.

조건은 규칙의 실행 여부를 결정하는 프레디킷(predicate)이다. 이는 사용자 질의어의 where 절과, 응용 프로그램 프로시져(procedure)에서의 조건 구문 등에 해당하는 부분이다.

행동은 규칙이 트리거되고, 그에 따른 조건이 참인 경우 실행되는 절(clause)이다. 행동은 삽입, 삭제, 수정, 검색 등의 데이터베이스의 연산이나 또는 복귀(rollback), 완료(commit) 등의 데이터베이스 시스템의 명령어, 그리고 응용 프로그램 프로시져 등이 될 수 있다.

이러한 규칙들을 사용하여 능동 데이터베이스 시스템은 무결정 제약이나 트랜잭션의 제어를 효율적으로 수행할 수 있으며, 이는 시스템 상에서 규칙들을 트리거함으로써 이루어진다.

능동 데이터베이스 시스템에서 제공하는 이러한 규칙들을 이용하면 기존의 데이터베이스 시스템들이 반드시 응용 프로그램이나 서브 시스템으로 구현해야 했던 기능들을 보다 효율적으로 구현할 수 있으며, 기존 데이터베이스의 영역을 넘어서는 응용 분야에 대해서도 손쉽게 정의하고 구현하는 일이 가능하다. 예를 들어, 기존의 데이터베이스 시스템에서 서브 시스템으로 만들어야 했던 무결성 제약을 능동 데이터베이스 시스템에서는 규칙만으로 구현할 수 있고, 기존의 데이터베이스 시스템에서 특별한 방법으로 표현해야 했던 트리거 시스템도 능동 데이터베이스 시스템에서는 자체적으로 지원한다. 따라서, 능동 데이터베이스 시스템은 기존의 데이터베이스 시스템에 비해 시스템의 관리, 유지에 있어 오버 헤드가 적은 장점이 있다.[4]

그러나, 능동 데이터베이스 시스템은 연속적으로 트리거되는 규칙들의 수행 여부를 판단(condition evaluation)하거나 데이터베이스에 그 수행 결과를 반영하기 위하여 계속적으로 데이터베이스에 접근하기 때문에 시스템의 성능을 심각하게 저하시키며, 따라서 이러한 문제점을 해결할 수 있는 방법이 필요하다. 이를 해결하기 위해 다음과 같은 방법들을 생각할 수 있다.

첫 번째로, 사용자의 질의를 예측하여 그 결과를 미리 빠른 속도의 기억 장치(주 기억 장치 또는 캐시 메모리)에 보관하는 방법이다. 이 경우 사용자의 질의가 들어오면 빠른 기억장치에 접근하여 원하는 레코드(튜플)를 찾아 출력한다.[5] 이 방법은 사용자가 질의에 대한 빠른 응답을 할 수 있는 장점이 있다. 그러나 사용자가 원하는 결과가 대용량일 경우에는 빠른 기억 장치에 보관하기 어려울 뿐 아니라 사용자의 질의 예측을 위한 메커니즘이 필요하고 정확한 예측이 매우 어렵다.

두 번째 방법은 능동 데이터베이스 시스템 상에서 트리거되는 규칙들과 이와 관련된 데이타를 빠른 기억 장치에 보관하여, 규칙이 데이터베이스에 접근하는 횟수를 줄이는 것이다. 즉,

데이터베이스에서 자주 참조되는 데이터를 빠른 기억장치에 보관시켜 두고 이에 대한 규칙들이 트리거되면 빠른 기억장치에서 실행하고 일정 시점 이후에 이를 데이터베이스에 반영한다. 이는 사용자의 한 트랜잭션이 여러 규칙들을 트리거할 것이고 따라서 여러 번의 데이터베이스 접근을 요구할 것이기 때문에 최대한 두 번(최초에 해당하는 데이터베이스의 튜플을 찾아 기억장치에 가져갈 때와 모든 처리가 끝나고 그 결과를 일정 시점 이후 데이터베이스에 반영할 때)의 데이터베이스 접근을 요구할 것이다. 본 논문에서는 이를 "규칙의 버퍼링(rule buffering)"이라고 정의한다. 규칙은 기억 장치에서 적은 부분만을 차지하며, 주로 참조되는 데이터에 대한 규칙들을 버퍼링하므로 데이터베이스로의 접근 횟수를 크게 줄일 수 있다.

본 논문에서는 규칙의 버퍼링을 능동 데이터베이스의 규칙을 이용하여 정의하고, 그에 따른 제반 구조를 제시하며, 이러한 정의와 구조가 실제 어떻게 적용되는 가에 대한 예제를 보인다.

본 논문의 구성은 2장에서 능동 데이터베이스 시스템에 관한 관련 연구들을 살펴보고, 3장에서 규칙의 버퍼링을 위한 규칙과 버퍼링 구조를 정의하고 그에 따른 예를 살펴보며, 4장에서는 결론과 앞으로의 연구 과제에 대해 논한다.

2. 관련 연구

능동 데이터베이스 시스템의 속도를 빠르게 하기 위해서 데이터베이스로의 접근 횟수를 줄일 필요가 있다.

규칙을 빠른 기억 장치에 일시적으로 보관하여 처리하는 경우에도 규칙이 데이터베이스에 접근하는 횟수는 감소하지 않는다. 이 방법은 단순히 해당하는 규칙을 빠른 기억장치에서 트리거하므로 해당 규칙의 탐색 속도를 증가시킨다. 따라서 데이터베이스로의 접근 횟수룰 줄이기 위해서는 규칙의 수를 줄이거나, 여러 규칙들을 하나의 규칙으로 묶어서 데이터베이스에 적용하는 지연 처리 방식(deferred coupling mode)을 취하는 방법이 있을 수 있다.

Stonebreaker가 제안한 규칙 캐싱(rule caching)은 검색(retrieve action)이 많이 일어나는 튜플(tuple)에 관련된 규칙들을 그 결과와 함께 캐싱(caching)하여 사용자가 그 행동에 대한 결과를 요구했을 경우 빠르게 처리할 수 있도록 한다. 이를 위한 방법으로 캐싱 데몬(caching demon)이라는 응용 프로그램을 사용하여 규칙과 그에 대한 결과들을 캐싱하여 자주 참조되는 데이터를 빠르게 보여줄 수 있다. 따라서 규칙의 캐싱을 통해 디스크 접근 횟수를 줄임으로써 시스템의 속도를 빠르게 한다.[4] 그러나 검색만을 캐싱하므로 부분적으로 시스템의 성능을 향상시킬 뿐이며, 또한 캐싱 데몬을 위한 응용 프로그램이 필요하게 되고 따라서 추가적인 오버 헤드(overhead)가 발생한다.

Starburst가 제안한 규칙 시스템은 Set-Oriented Production Rule을 기반으로 구성되어 있으며, 한 트랜잭션의 처리 중에 규칙에 관련된 모든 튜플이 데이터베이스에 적용된다. 하나의 트랜잭션은 여러 개의 변이(transition)로 구성되고, 연속적인 변이에 따라 변경된 튜플의 내용을 집합 개념으로 조합(transition effect composition)하여 한 트랜잭션에서의 변이 효과를 변이 테이블(transition table)에 보관하고 연속적으로 트리거되는 규칙의 조건을 효율적으로 평가하기 위해 사용한다.[8]

HIPAC의 경우는 효율적인 조건 평가(condition evaluation)를 위해 복수 질의(multiple query)의 최적화와 점진적 평가(incremental evaluation) 그리고 유도된 데이터(derived data)의 구체화 기법을 이용한다.[9]

D. Montesi와 R. Torlone은 갱신 연산의 순수 효과(net effect)를 산정하기 위해 트랜잭션 실행 중에 실시된 갱신들을 저장하는 델타 릴레이션 개념을 이용하였다. 이를 이용하여

처리할 능동 규칙을 감소시키도록 하는 트랜잭션 변환 기법을 제안하였다.[10]

전술한 바와 같이 대부분의 능동 데이터베이스 연구는 효율적인 조건 평가 방법에 대한 방법에 관한 것이며, Stonebreaker의 규칙 캐싱 기법은 단지 검색에 관한 연구이다. 본 논문은 검색뿐만 아니라 데이터베이스의 모든 기본 연산에 대해 규칙의 버퍼링을 적용하는 할 수 있는 방법을 제안하고자 한다.

3. 규칙의 버퍼링을 위한 구조와 버퍼링 단계

3.1 규칙 버퍼링의 구조

검색 연산뿐만 아니라 데이터베이스의 내용을 변경하는 연산에 대해서도 캐싱 또는 버퍼링 작업이 필요하다.

규칙 버퍼링의 목적은 능동 데이터베이스 시스템의 규칙 중에서 데이터베이스의 내용을 변경시키는 규칙들과 해당 데이터베이스의 튜플을 빠른 기억장치에 버퍼링함으로써, 규칙을 효율적으로 처리하여 시스템의 효율을 높이는데 있다.

규칙의 버퍼링 구조는 다음과 같다.

· 버퍼링 영역(buffering area)

버퍼링을 위해서 먼저 규칙 버퍼링을 위한 공간의 확보가 필요하며, 보조 기억장치보다 속도가 빠른 주기억장치나 캐시 메모리를 이용한다.

버퍼링 영역은 먼저 규칙과 그에 대한 데이터들을 저장하는 버퍼 공간(buffer space)과 규칙이 적용된 해당 데이터 즉, 튜플이 버퍼링의 대상인가를 판단할 인덱스로 구성한다. 버퍼 공간에 저장되는 데이터는 튜플 단위이며 모든 애트리뷰트들을 포함한다. 인덱스는 규칙의 버퍼링 대상이 되는 튜플에 관한 정보를 가지고 있으며, 버퍼링 공간과는 별도로 참조 가능해야 한다. 이는 어떠한 규칙이 트리거되었을 경우 그 규칙이 적용되는 튜플이 버퍼링의 대상 인가의 여부를 확인하는데 사용할 수 있어야 하기 때문이다.

3.2 규칙의 버퍼링(rule buffering) 단계

· 1단계(first step)

규칙이 트리거되면, 일단 그 규칙이 적용될 튜플이 버퍼링의 대상 인지의 여부를 버퍼링 영역의 인덱스를 참조하여 판별한다.

· 2단계(second step)

인덱스 참조 결과 그 튜플이 버퍼링의 대상인 것으로 판별되면, 규칙과 해당 튜플들을 버퍼 공간에 저장한다. 만일 대상이 아니면 그 규칙은 그대로 실행한다.

· 3단계(third step)

적당한 시점에 버퍼 공간에 저장된 모든 규칙들을 처리하고 버퍼 공간을 비운다.

이를 위해 유의해야 할 사항은 다음과 같다.

ⓐ 규칙의 버퍼링 역시 규칙을 통해 이루어진다.

ⓑ 이 규칙들은 데이타베이스의 무결성 제약 등 데이터베이스 운영에 영향을 미치는 중요한 제약들(constraints)에 관한 규칙들을 제외한 다른 모든 규칙에 대해 우선적으로 실행되어야 한다.

・4단계(fourth step)

버퍼 공간에서 규칙을 재작성(rule rewrite)한다. 즉, 규칙이 버퍼링되었을 때, 관련된 모든 규칙(한 규칙에 의해 트리거되는 모든 규칙)에 대해, 규칙을 재구성(rule rewrite)하여 하나의 규칙으로 만든다.

규칙을 재작성함으로써 버퍼 공간 안에 있는 데이타의 크기와 양을 줄일 수 있으며, 또한 버퍼 공간에 있는 규칙들을 일정 시점 후 데이타베이스에 적용시켜야 될 경우, 데이타베이스에 적용되는 데이터의 양을 줄일 수 있다.

3.3 규칙의 버퍼링을 위한 새로운 키워드

본 논문에서 제시하는 모든 예제들은 POSTGRES의 Production Rule System 2(PRS2)를 확장하여 만들었으며, 이를 위해 새로이 정의한 키워드(keyword)는 다음과 같다.

① FRONT

검색 규칙에 의해 버퍼링되어 있던 규칙들이 실행될 때, 첫 번째 규칙을 가리키는 키워드이다.

② REAR

버퍼 영역에 규칙을 버퍼링할 때 튜플별로 버퍼링되어 있는 규칙들 중 가장 마지막 규칙을 가리키는 키워드이다.

③ BUFFER

버퍼 공간(buffer space)을 나타내는 키워드이다.

④ INDEX

버퍼링된 튜플의 인덱스를 나타낸다.

⑤ NEW

NEW는 PRS2에 정의되어 있는 키워드이며, 본 논문에서는 NEW 키워드에 새로운 기능을 추가하여 사용한다. 추가된 새로운 기능은 다음과 같다.

만일 새로이 적용되는 규칙이 갱신(update)을 위한 규칙인 경우, NEW는 갱신에 관계하는 애트리뷰트(attribute)들을 제외한 나머지 애트리뷰트들의 값을 널(NULL)로 만든다. 버퍼링된 튜플을 실제의 데이터베이스에 값을 매핑할 때(mapping) 즉 저장할 때, 널 값을 가진 애트리뷰트는 적용하지 않는다.

3.4 규칙 버퍼링의 예

본 논문은 다음과 같은 릴레이션 스킴(relation scheme)을 사용하여 사건(event)의 형태별 즉, 삽입, 갱신, 삭제, 검색 사건이 대해 규칙의 버퍼링이 어떻게 이루어지는가를 살펴본다.

DEPOSIT Scheme =
{branch-name, account-number, customer-name, amount}

한 튜플에 대해 연속하여 발생하는 사건에 있어 가능한 조합은 다음과 같다.

사건의 조합	효 과
insert - insert	error
insert - update	insert(insert_ and update_data)
insert - delete	none
update - insert	error
update - update	update(update_data)
update - delete	delete
delete - insert	update(insert_data)
delete - update	error
delete - delete	error

① 삽입(insert)
삽입 사건은 무조건 버퍼 공간에 삽입한다. 이는 삽입되는 튜플은 잘못된 삽입으로 곧 바로 삭제 또는 갱신 가능성이 있기 때문이다. 규칙의 정의는 다음과 같이 한다.

· buffering rule

```
define rule append_buffer is
on append to DEPOSIT
do instead
{
   append to BUFFER(action = 'a')
         where any(NEW.account-number =
                  INDEX.account-number)
   append to DEPOSIT
         where any(NEW.account-number =
                  INDEX.account-number) = 0
}
```

· rule rewrite

```
define rule replace_append_buffer is
on append to BUFFER(NEW.action = 'a')
do instead
{
  abort where (REAR.action = 'r' or )
```

```
                REAR.action = 'a')
    replace  BUFFER(action = 'r',
            branch-name = NEW.branch-name,
            account-number = NEW.account-number,
            customer-name = NEW.customer-name,
            amount = NEW.amount)
        where (REAR.action = 'd')
  }
```

② 갱신(update)

갱신을 위한 규칙의 정의는 다음과 같다.

· buffering rule

```
  define rule replace_buffer is
  on replace DEPOSIT
  do instead
  {
     append to BUFFER(action = 'i')
            where any(NEW.account-number =
                     INDEX.account-number)
     replace DEPOSIT
            where any(NEW.account-number =
                     INDEX.account-number) = 0
  }
```

· rule rewrite

```
  define rule insert_replace_buffer is
  on append to BUFFER(NEW.action = 'r')
  do instead
  {
   abort where (REAR.action = 'd')
   replace  BUFFER(branch-name =
                    NEW.branch-name,
            account-number = NEW.account-number,
            customer-name = NEW.customer-name,
                  amount = NEW.amount)
       where (REAR.action = 'r')
   append to BUFFER(branch-name =
                    NEW.branch-name,
            account-number = NEW.account-number,
```

```
        customer-name = NEW.customer-name,
              amount = NEW.amount)
    where (REAR.action = 'a')
}
```

③ 삭제(delete)
규칙을 그대로 버퍼링하며, 규칙의 정의는 다음과 같이한다.

· buffering rule

```
define rule delete_buffer is
on delete DEPOSIT
do instead
{
   append to BUFFER(action = 'd')
         where (NEW.account-number =
               INDEX.account-number)
   delete DEPOSIT
         where any(NEW.account-number =
                  INDEX.account-number) = 0
}
```

④ 검색(retrieve)
해당 튜플에 대해서 현재까지 버퍼링된 모든 규칙을 실제 데이터베이스에 적용한다.

3.5 규칙 버퍼링을 위한 질의어의 분할

규칙을 버퍼링할 때, where 구문(clause)에 조건식이 적용되어 있으면 버퍼링할 튜플을 결정할 수 없다. 즉, 튜플을 결정하기 위해서는 조건식을 적용하여 해당하는 튜플을 산출해야 하지만, 질의의 대상이 하나의 튜플이 아니라 여러 개의 튜플인 질의어는 버퍼링이 불가능하다. 그러므로 시스템에 이러한 질의가 들어오면 규칙의 버퍼링이 가능한 질의 형태로 변환시키며, 이는 연산(operation)과 조건식 부분을 두 개의 규칙으로 분할하여 처리한다. 예를 들어, "예금 구좌의 잔액이 1000을 넘지 않는 레코드를 모두 삭제하라"라는 질의는

```
do delete DEPOSIT where amount <= 1000
```

와 같이 표현할 수 있으며, 이에 대한 규칙을 버퍼링하기 위해서는 amount가 1000이 넘지 않는 튜플들을 알아내야만 한다. 즉, amount가 1000이 넘지 않는 튜플들을 알아내서 위해서는 일단 amount <= 1000이라는 조건식을 적용해야만 한다. 이는 결국 위의 질의를 실행시킨다는 것을 의미하며, 따라서 이 질의에 관련된 규칙을 버퍼링할 수 없게 된다. 그러므로 실제 행동(action)에 해당하는 delete 부분과, 조건식인 amount <= 1000 부분을 분할하여 실행시킨다. 이를 위해 질의어는 다음과 같이 분할하여 표현할 수 있다.

```
do retrieve into TEMP (DEPOSIT.ALL)
   where DEPOSIT.amount <= 1000
do delete TEMP
```

여기에서 TEMP는 DEPOSIT 릴레이션에서 amount가 1000이 넘지 않는 튜플들만 모아서 임시로 만든 릴레이션이다.

4. 결론 및 연구 과제

본 논문에서는 규칙들의 연속된 트리거에 따라 접근하는 데이터베이스로의 접근 횟수를 줄이기 위한 규칙의 버퍼링 방안을 제시하고, 이에 대한 적용 예제를 보였다. 이를 위해 버퍼링 영역의 구조를 정의하고 각 사건에 대해 예제를 이용하여 버퍼링을 위한 규칙이 어떻게 정의되는가와 여러 튜플이 질의의 대상이 되는 경우 질의어를 분할하는 방안을 살펴보았다.

앞으로 해결해야 할 연구 과제는 다음과 같다.

첫째, 효율적인 버퍼 영역의 구조 정의에 대한 연구가 필요하다. 버퍼 영역의 구조는 본문에서 기술한 바와 같이 버퍼 공간과 인덱스로 나누어진다.

버퍼 공간에 버퍼링될 각 튜플은 자신이 속한 릴레이션과 같은 저장 구조가 필요하다. 그러나 데이터베이스는 릴레이션을 여러 개 가지고 있다. 따라서 버퍼 공간에 각각의 릴레이션에마다 하나씩 저장 구조를 가질 것인지 아니면 각 릴레이션을 통합하여 하나의 저장 구조만을 가질 것인지에 대한 연구가 필요하다. 직관적인 관점으로 살펴보면 각 릴레이션을 별도로 분리하는 경우는 각 릴레이션마다 하나씩의 인덱스가 필요하므로 추가적인 버퍼 공간이 필요하게 된다. 또한 각 릴레이션을 통합하여 만드는 경우는 버퍼링할 규칙에 대한 데이터를 저장할 때 모든 가능한 애트리뷰트를 알아야 한다. 이를 위해 시스템 카탈로그를 참조해야 하며, 추가적인 디스크로의 접근이 필요하다. 이를 버퍼링 영역에 두는 경우 역시 추가적인 공간이 필요하다.

버퍼 영역의 구조의 또 다른 요소인 인덱스에 대해서는 어떤 튜플을 언제, 어떻게 인덱스에 포함시키고 삭제할 것인가를 결정하는 문제가 있다. 이를 해결하기 위해 DBA가 정해진 기간마다 시스템을 검색하여 조정해 주는 방법이 있을 수 있는데 이는 좋은 방법이 아니다. 그러므로 DBMS 자체 내에서 일정 기준을 사용하여 자동적으로 조정하는 방안이 연구되어야 한다.

두 번째는 로그 레코드(log record)의 처리 문제이다.

버퍼링 영역이 주기억장치 또는 다른 빠른 기억장치에 위치하고 있으므로 시스템 붕괴(system crash) 등의 문제 발생으로 자칫 모든 데이터가 소실될 수 있다. 이러한 문제로부터 소실된 데이터를 복구하기 위해 로그를 이용한다. 그러나 버퍼링 공간에서 규칙은 재작성되므로 로그 레코드에 기록된 내용과 실제 버퍼링 공간의 내용이 서로 상이한 경우가 발생할 수 있다. 또한 버퍼링 영역은 그 처리 속도가 매우 빠르기 때문에, 많은 양의 로그 레코드가 쌓이게 되므로 오버 헤드가 발생한다.

세 번째 문제는 버퍼링되어 있는 규칙들과 데이터를 실제로 데이터베이스에 적용할 것인가 하는 문제이다.

참고문헌

(1) 양우석, 이도현, 김동욱, 김명호, "능동 데이타베이스 시스템의 개발," 한국정보과학회 가을 학술발표 논문집 Vol. 21, No. 2, pp. 217-220, 1994.

(2) 이석호, 데이터베이스 시스템, 정익사, 1995

(3) 임채명, 이혜명, 박동규, "데이터베이스의 상태변이 제어를 통한 일관성 유지," 순천향대학교 산업기술연구논문집 제3권 제1호, pp.71-83, 1997

(4)Widom J., Ceri S., "Introduction to Active Database Systems," *Active Database Systems*, pp.1-41, Morgan Kaufmann publishers, INC., 1996.

(5)Stonebreaker M.,Jhingran A.,, Goh J,, and Potamianos S,, "On Rules, Procedures, Caching and Views in Data Base Systems," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 281-290, 1990.

(6)Widom J,, "The StarBurst Rule System," *Active Database Systems*, pp.87-109, Morgan Kaufmann publishers, INC., 1996.

(7)Simon E., and Kiernan J,, "The A-RDL System," *Active Database Systems*, pp.111-149, Morgan Kaufmann publishers, INC., 1996.

(8)Widom J,, Finkelstein S. J,, "Set-Oriented Production Rules in Relational Database systems," *Proceedings of International Conference on Management of Data, ACM SIGMOD*, pp.259-270, 1990

(9)McCarthy D. R,, Dayal U,, "The Architecture of An Active Data Base Management System," *Proceedings of International Conference on Management of Data, ACM SIGMOD*, pp.215-224, 1989

(10) Danillo Montesi, Riccardo Torlone, "A Transaction Approach to Active Rule Processing," *Proceedings 11th International Conference DATA ENGINEERING*, pp.109-116, 1995

(11)Date C, J., *AN INTRODUCTION TO Database Systems*, Vol. I, Fifth Edition, Addison-Wesley Publishing Company, Inc., 1990

(12)Umeshwar Dayal, Eric Hanson, and Jennifer Widom, "Active Database systems," *MODERN DATABASE SYSTEMS*, pp.434-456, acm PRESS, 1995