시점 시퀀스를 이용한 시간지원 집계의 처리
(Processing Temporal Aggregate Functions using a Time Point Sequence)

권 준 호 † 송 병 호 †† 이 석 호 †††
(Joonho Kwon) (Byoungho Song) (Sukho Lee)

요 약 시간에 따라 변화하는 사건들을 저장하는 시간지원 데이타베이스에서 기존의 집계 처리 기법에 시간을 고려하여 처리하도록 확장해야 한다. 기존의 시간지원 집계 처리 기법들은 매번 질의의 대상이 되는 사건들이 다를 때마다 시간 구간을 반복해서 구하고 그 구간마다의 결과를 계산해야 한다는 문제점이 있다. 본 논문에서는 시간지원 데이타베이스에 저장된 사건의 시작 시간과 종료 시간만을 미리 읽어들여서 구성한 시점 시퀀스를 이용하여 시간지원 집계를 처리하는 방법을 제안하였다. 또한 데이타베이스에서 저장된 사건의 삭제나 새로운 사건의 삽입에 따른 시점 시퀀스 갱신의 용이성에 대해서도 언급하였다. 시점 시퀀스는 시간 구간에 대한 정보를 미리 저장하고 있기 때문에, 질의의 대상이 되는 사건들이 다른 시간지원 집계 질의가 계속해서 들어올 때 기존의 방법에 비해 효율적으로 처리할 수 있다.

키워드 : 시점 시퀀스, 시간지원 집계, 시간지원 데이타베이스

Abstract Temporal databases support time-varying events so that conventional aggregate functions are extended to be processed with time for temporal aggregate functions. In the previous approach, it is done repeatedly to find time intervals and is calculated the result of each interval whenever target events are different. This paper proposes a method which processes temporal aggregate function queries using time point sequence. We can make time point sequence storing the start time and the end time of events in temporal databases in advance. It is also needed to update time point sequence due to insertion or deletion of events in temporal databases. Because time point sequence maintains the information of time intervals, it is more efficient than the previous approach when temporal aggregate function queries are continuously requested, which have different target events.

Key words : time point sequence, temporal aggregate, temporal database

1. 서 론

질의 연산자 중에서 모든 직원의 월급을 합계하는 것과 같은 집계(aggregate function)는 릴레이션의 전체 혹은 그 일부를 구성하는 투플에 적용이 되어서 하나의 스칼라 값을 계산해 내는 연산이다. 이러한 집계들은 SQL과 같은 질의 언어에서 매우 중요한 부분이며, 많은 응용 프로그램에서 자주 사용된다. 특히 TPC-D[1]와 같은 질의 성능평가에서는 17개의 질의문 중에서 15개의 질의가 집계를 포함하고 있다. 그러므로 데이타베이스 응용의 성능을 향상시키기 위해서 집계를 효율적으로 처리하는 것이 매우 중요하다.

기존의 데이타베이스 시스템은 현실 세계에서 발생한 사건에 대하여 가장 최근의 상태만을 반영한다. 새로운 데이타 값이 데이타베이스에 반영될 때, 새로운 값으로 기존의 데이타 값을 덮어쓰게 된다. 따라서 기존의 데이타베이스는 항상 가장 최근의 데이타를 저장하고 있다. 따라서 과거의 자료들도 요구하는 응용을 지원하기 위해서, 시간지원 데이타베이스(Temporal Database)가 제안[2]되었다. 시간지원 데이타베이스는 시간에 따라 변화하는 사건(event)이나 정보에 대한 질의와 저장을 지원한다.

† 학생회원 : 서울대학교 전기·컴퓨터공학부
bluerain@db.snu.ac.kr
†† 종신회원 : 상명대학교 소프트웨어학부 교수
bhsong@sangmyung.ac.kr
††† 종신회원 : 서울대학교 전기·컴퓨터공학부 교수
shlee@cse.snu.ac.kr
논문접수 : 2001년 10월 4일
심사완료 : 2003년 3월 18일

해당되는 릴레이션에 대해 하나의 스칼라 값을 계산하는 기존의 집계 또한 시간지원 데이타베이스의 시간 속성을 반영한 시간지원 집계로 확장되어야 한다. 시간지원 집계를 지원하는 여러 가지 시간지원 질의 언어[3]가 제안되었다. 이들 언어에서 시간지원 집계는 사건들의 시간 속성에 의하여 전체의 시간 축을 분할하여 여러 개의 시간 구간들을 만들며, 이 각각의 시간 구간마다 집계의 연산을 수행하여 그 구간마다 결과 값을 계산하여 사용자에게 돌려준다. 시간지원 집계의 처리 과정 중에서 각각의 시간 구간에 따라 해당되는 사건들을 묶고 그 그룹들에 대해서 집계 값을 계산하는 것을 시간지원 그룹화(temporal grouping)라 한다. 또 시간지원 집계를 계산하는 동안 그 대상이 되는 사건들의 그룹이 변화하지 않는 시간 구간을 불변 구간(constant interval)이라 한다. 즉 불변 구간에서 집계의 결과 값은 일정하다.

기존의 시간지원 집계의 처리 방법들은 사용자가 시간지원 집계 질의를 요청하면, 트리와 같은 자료 구조를 이용하여 시간지원 그룹화를 수행하여 불변 구간을 구하고 그 구간마다의 결과를 계산한다. 이 논문에서는 시간지원 그룹화의 기본이 되는 불변 구간에 대한 정보로서, 미리 데이타베이스를 스캔하여 사건들의 시작 시간과 종료 시간만을 정렬하여 유지한 시점 시퀀스(time point sequence)를 제안하고, 시점 시퀀스를 이용한 시간지원 집계 질의의 처리 방법을 설명하였다. 또한 인공적인 데이타를 대상으로 한 실험을 통하여 기존 기법과의 비교와 분석을 하였다. 기존의 방법들은 시간지원 집계 질의의 대상이 되는 사건들이 다를 때마다 매번 불변 구간을 구해야 하지만, 시점 시퀀스는 미리 불변 구간에 대한 정보를 유지하고 있기 때문에 이런 경우에 좀 더 효율적으로 처리할 수 있다.

본 논문의 구성은 다음과 같다. 먼저, 2장에서는 시간지원 데이타베이스와 시간지원 집계에 대해서 알아보고, 기존의 시간지원 집계 처리 기법들을 살펴본다. 3장에서는 시간지원 집계 처리를 위한 시점 시퀀스(time point sequence)를 제안하고, 시점 시퀀스의 생성 방법과 시점 시퀀스를 이용한 시간지원 집계의 처리 방법에 대해 알아본다. 그리고 시점 시퀀스의 갱신에 대해 살펴본다. 4장에서는 인공적으로 생성된 실험 데이타를 이용하여, 시점 시퀀스를 이용한 시간지원 집계의 처리와 기존의 시간지원 집계의 처리를 실험을 통해 비교하고 실험 결과에 대한 분석을 한다. 마지막으로, 5장에서 연구 결과를 정리하고 결론을 맺는다.

2. 관련 연구

2.1 시간지원 집계

기존의 데이타베이스에서 집계 연산은 데이타베이스에서 현재의 상태만을 보여 준다. 예를 들어 그림 1과 같은 집계 질의는 직원 릴레이션의 투플들을 읽어서, 현재 직원들의 월급의 합과 최대값을 출력한다. 집계 연산은 COUNT, SUM, AVG와 같이 누적된 계산으로 결과를 산출하는 누적형 집계(computational aggregate)와 MIN, MAX와 같이 여러 값들 중에서 대표적인 값을 선택하는 선택형 집계(selective aggregate)로 나눌 수 있다.

```
SELECT SUM(Salary), MAX(Salary)
FROM Employee
```

그림 1 집계 질의

일반적으로, 시간지원 데이타베이스에 저장되는 사건들은 두 가지 시간 속성 시작 시간(t_s)과 종료 시간(t_e)을 가지게 된다. 이 시간 속성들은 시간 구간의 하위 경계값과 상위 경계값을 나타낸다. 즉 시작 시간(t_s)과 종료 시간(t_e)을 가지는 사건은 시간 구간 [t_s, t_e)에서 계속해서 유효(valid)한 것이다. 이처럼 시간지원 데이타베이스에서는 각각의 사건이 시간 속성을 포함하고 있으므로, 시간을 반영하여 질의를 처리해야 한다. TSQL2는 SQL-92 질의 언어에 시간지원에 관한 요소들을 추가하여 확장한 질의 언어이다[4]. 시간지원 집계의 처리 과정에서 시간 축을 여러 부분으로 분할한 후 시간지원 그룹화를 하며, 나누어진 각각의 시간 구간에 대해 집계를 수행하여 결과값을 돌려준다.

시간지원 집계 질의와 그 결과의 예는 그림 2와 같다. 그림 2의 (a) 예제 릴레이션은 어떤 회사에 채용된 직원의 월급, 부서를 기록하고 있다. 특히 Start와 End 속성은 직원들이 어느 정도의 월급을 받으면서, 어느 부서에서 근무를 시작하고 끝냈는지를 나타내는 시간들이다. 이 두 속성은 시작 시간과 종료 시간에 각각 대응된다. 예제 릴레이션을 보면 시간에 따라서 부서에 근무하는 직원들의 수와 그 직원들의 월급이 변화함을 알 수 있다. 따라서 이 예제 릴레이션에 그림 2의 (b)와 같은 TSQL2 질의가 요청되었다고 하면, 결과로서 직원들의 월급의 합과 최대값을 시간에 따라 변화하는 값으로 주어져야 할 것이다. 그림 2의 (b) TSQL2 질의에 대한 결과는 그림 2의 (c)와 같다.

질의 처리 과정 중 시간지원 그룹화 과정을 자세히 보면 그림 3과 같다. 그림 2의 (a) 예제 릴레이션의 사건들을 시간 축 위에 그리고, 각각의 사건들에 대한 시작 시간과 종료 시간을 시간 축에 표시한다. 먼저 집계 연산을 위해 사건들의 그룹이 변화하지 않는 불변 구간을 찾는다. [7,8), [8,12), [12,14), [14,18), [18,20), [20,21), [21,31)의 구간이 불변 구간이다. 그런 다음 각각의 불변 구간마다 해당되는 사건들만으로 시간지원 그룹화를 하여 질의에서 요청하고 있는 집계 연산을

Name	Salary	Dept	Start	End
Richard	46K	Accounting	18	31
Karen	45K	Shipping	8	20
Nathan	35K	Marketing	7	12
Nathan	38K	Accounting	14	21

(a) 예제 릴레이션 Employee

```
SELECT SUM(Salary), MAX(Salary)
FROM Employee
```

(b) TSQL2 시간지원 집단 함수 질의의 예

SUM	MAX	Start	End
35K	35K	7	8
80K	45K	8	12
45K	45K	12	14
83K	45K	14	18
129K	46K	18	20
84K	46K	20	21
46K	46K	21	31

(c) 시간지원 집단 함수 질의의 결과

그림 2 예제 릴레이션과 시간지원 집계 질의의 예

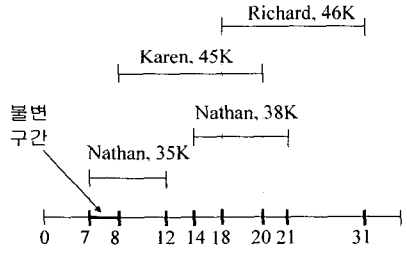


그림 3 그림 2의 예제 릴레이션에서 시간지원 그룹화

수행하여 결과값을 계산한다. 예를 들어 [7,8)의 구간에서는 <Nathan, 35K>가 [18,20)의 구간에서는 <Nathan, 38K>, <Karen, 45K>, <Richard, 46K>가 각각의 구간에서 시간지원 그룹화의 대상이 된다.

2.2 기존 연구

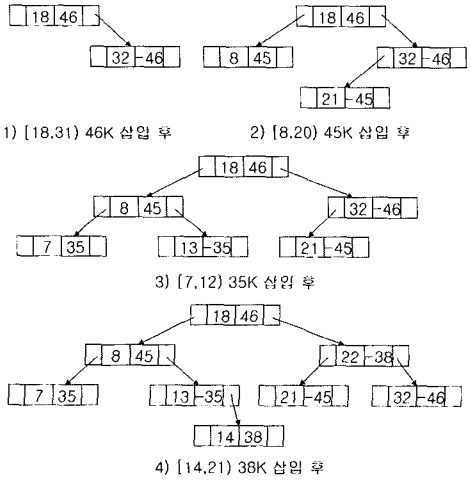
기존의 데이타베이스에서 집계를 계산하기 위한 방법으로 두 단계로 동작하는 Epstein의 알고리즘[5]이 있다. Epstein의 방법에서는 집계 질의의 결과를 저장하기 위해 사건의 개수를 저장하는 count와 집계의 결과값을 저장하는 result의 두 가지 속성을 가진 투플을 미리 할당한다. 그런 다음 주어진 조건을 만족하는 사건을 읽어들여 count와 result의 값을 변경하며, 모든 사건을 다 읽었을 때 최종적으로 결과를 반환한다. Tuma가 기존의 Epstein의 알고리즘에 기반을 두고 확장을 하여, 첫 단계에서 불변 구간을 구하고 두 번째 단계에서 Epstein 의 알고리즘을 사용하는 시간지원 집계의 처리를 구현하였다[6]. 두 번의 데이타베이스 스캔이 필요한 Tuma의 방법과 달리 데이타베이스를 한 번만 스캔하면서, 시간지원 집계를 효율적으로 처리하는 방법으로 세그먼트 트리와 유사한 형태의 트리를 사용한 집계 트리 기법[7]이 제안되었다. 집계 트리는 데이타베이스가 한 번 스캔될 때 형성된다. 모든 사건들을 읽은 후, 트리의 리프 노드에 있는 시간 구간들이 불변 구간을 나타낸다. 해당 불변 구간에 대한 집계의 결과는 루트와 리프 노드, 루트에서 리프 노드에 이르는 중간 노드들을 깊이 우선 탐색(depth first search)을 하면서 집계를 위해 저장된 필드에 해당 집계 연산을 적용하여 얻을 수 있다. 그러나 사건들이 시작 시간과 종료 시간에 대해 정렬되어 있다면, 집계 트리의 모양이 연결 리스트와 비슷한 구조가 되어 성능이 저하된다.

집계 트리보다 효율적인 처리 기법에 대한 연구로 PA-트리[8,9]와 균형화된 트리 기법[10]이 제시되었다. 집계 트리는 노드에 시간 구간을 기록하는데, PA-트리와 균형화된 트리는 노드에 시점(time point)만을 기록한다.

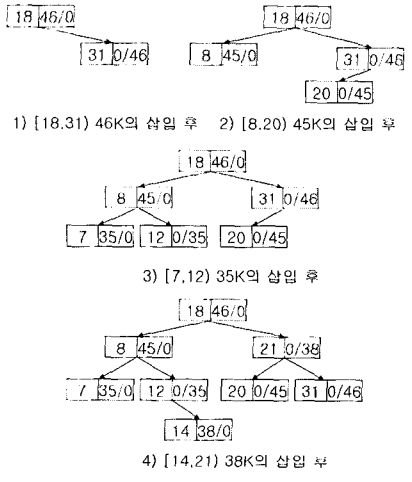
PA-트리와 균형화된 트리의 노드에는 시간 구간의 시작 시간 혹은 종료 시간에 대한 정보를 나타내는 필드와 집계를 위한 필드가 저장되고, 균형화된 트리에는 부가적으로 노드의 색깔을 나타내는 필드가 더 필요하다. PA-트리와 균형화된 트리의 생성은 이진 트리의 생성과 유사하게 노드에 저장된 시점을 키(key) 값으로 하여 수행된다. 이 때, 삽입하고자 하는 시점을 갖는 노드가 이미 트리에 있으면 집계 연산에 따라 집계를 위한 필드만 갱신하면 된다. PA-트리와 균형화된 트리에

서 하나의 사건을 처리하기 위해서는 그 사건의 시작 시점과 종료 시점 중의 한 값과 집계를 위해 필요한 값을 가지는 두개의 노드를 삽입한다. 노드들이 삽입 될 때마다 PA 트리와 균형화된 트리는 각각 AVL 트리와 레드 블랙 트리의 균형화 기법을 사용하여 트리의 편중을 막는다.

그림 4는 그림 2의 (a) 예제 릴레이션에서 SUM 집계 처리를 위한 PA 트리와 균형화된 트리의 생성 예를 보여준다. PA 트리에서는 시작 시점을 삽입 할 때는 집계 처리에 필요한 값의 변화량이 양수가 되고, 종료 시점을 삽입할 때는 변화량이 음수가 되게 한다. 유효 시



(a) PA 트리의 생성 예



(b) 균형화된 트리의 생성 예

그림 4 PA 트리와 균형화된 트리의 생성 예

간이 [18,31)인 구간을 가지는 <Richard, 46K>의 데이타를 표시하기 위해 그림 4의 (a) PA-트리에서는 시점이 18이고 변화량이 46K인 노드와 시점이 32이고 변화량이 46K인 노드 두개가 삽입된다. 그림에는 편의상 K를 생략하였다. 같은 사건에 대한 정보를 기록하기 위해, 그림 4의 (b) 균형화된 트리에서는 두 개의 시점 18과 31이 생성되고, 각 시점마다 변화량을 표시하는 값을 가진다. 시점 18에서의 "46/0"은 시점 18에서 사건이 시작하고 그 값이 46임을 나타내고, 시점 31에서 "0/46"은 사건이 종료하고 그 값은 46임을 나타낸다. 균형화된 트리에서 음영이 있는 노드는 블랙 노드이고, 음영이 없는 노드는 레드 노드이다.

집계의 결과를 얻기 위해서는 트리가 생성된 후, 중위 순회를 하면서 각 노드가 기록한 값들을 읽어서 결과에 반영한다. PA 트리에서 중위 순회를 하면 <7,35>, <8,45>, <13, 35>, <14 ,38> 등의 순서로 노드들을 방문하게 된다. 이 때 SUM의 결과는 시점 7에서부터는 35K, 시점 8에서부터는 35K+45K=80K 시점 13에서부터는 80K 35K-45K가 된다. 균형화된 트리도 SUM의 처리를 위해서 중위 순회를 하면서 시작하는 사건의 값은 더하고 종료하는 사건의 값은 빼면서 원하는 결과를 얻을 수 있다. 균형화된 트리도 중위 순회를 하여 같은 결과를 얻을 수 있다.

누적형 집계의 처리는 위와 같이 PA-트리와 균형화된 트리 둘 다 트리를 생성한 후, 중위 순회를 하면서 불변 구간과 집계의 결과를 구할 수 있다. 선택형 집계의 처리를 위해 PA 트리는 부가적인 자료구조를 이용한다. PA 트리는 시작 시점을 기록하는 노드에 해당 시작 구간의 길이와 집계의 대상이 되는 값에 대한 리스트를 유지한다. 이 리스트를 이용하여 시간 지원 집계를 스카이라인 문제로 변환하여 트리의 후위 순회를 통하여 결과를 구한다. 균형화된 트리 기법은 2개의 히프(heap)를 이용하여 한 사건의 유효한 시간 구간을 기록하고 있고, 트리를 중위 순회할 때 히프를 사용하여 집계의 결과를 구한다.

2.3 기존 기법의 제약

그림 2의 (a) 릴레이션에 대해 그림 5와 같이 해당하는 부서만의 월급의 합계를 요구하는 시간지원 집계 질의가 계속해서 사용자에 의해 요청된다고 하자.

이 질의들에 포함되는 사건들의 집합들이 서로 다르기 때문에, 불변 구간들도 각각 다르게 된다. 따라서 기존의 기법에서는 각각의 질의에 대해서 불변 구간을 구하고 시간지원 그룹화를 하기 위해 매번 트리를 생성하고 트리의 노드들을 순회하여 결과를 돌려주는 과정을

```
TSQL> SELECT SUM(Salary)
        FROM Employee
        WHERE Dept='Shipping';

TSQL> SELECT SUM(Salary)
        FROM Employee
        WHERE Dept='Accounting';

TSQL> SELECT SUM(Salary)
        FROM Employee
        WHERE Dept='Marketing';
```

그림 5 시간지원 집계를 포함하는 연속된 질의들

반복하게 된다. 이것은 먼저 생성된 트리가 이후의 질의에 대해서 불변 구간에 대한 어떠한 정보도 제공하지 않기 때문이다.

3. 시간지원 집계 처리를 위한 시점 시퀀스

3.1 시점 시퀀스의 생성

시점 시퀀스란 시간 축을 따라 정렬된 시점들의 순차로서 불변 구간에 대한 정보를 가지고 있는 리스트 구조이다. 시점 시퀀스에 저장되어 유지될 수 있는 시간 값들은 시간지원 데이타베이스(TDB)에 저장된 데이타의 시작 시간(t_s)이나 종료 시간(t_e)이다. 시점 시퀀스에 저장될 수 있는 시점들의 집합(TS)은 다음과 같이 형식화 할 수 있다.

$$TS=\{\ t \mid \exists e \in TDB((t = e.t_s) \vee (t = e.t_e))\ \}$$

시점 시퀀스는 시점들의 집합(TS)에 포함된 원소만으로 구성되며, 그 시점들을 정렬하여 유지한다. 즉 시점 시퀀스(TPS)에 속한 원소들은 다음과 같은 성질을 만족한다.

$$\forall t_i, t_j \in TPS : i < j \rightarrow t_i < t_j$$
(단 i, j는 시점 시퀀스에서의 위치)

시점 시퀀스는 그림 6의 알고리즘을 사용하여 미리 생성하여 유지한다. 전체 데이타베이스에서 해당되는 릴레이션을 한번 스캔한다. 이 때 사건의 시작 시간과 종료 시간만을 읽어들인다. 읽어 들인 시점이 시점 시퀀스에 이미 존재하고 있으면 그 시점의 발생 빈도 필드를 증가시키고, 이미 존재하고 있지 않던 시점이면 새로이 시점 시퀀스에 추가한다. 여기서 발생 빈도 필드는 시점 시퀀스의 삭제 시에 하나의 사건에 의해서 생성된 것인지, 여러 개의 사건들에 의해서 생성된 것인지 구별하기

```
1   Procedure Create_TPS() Begin
2       Set TPS  empty
3       For each event e in a relation Do
4           If (e.t  t for any time point in TPS) Then
5               t.freq++
6           Else
7               add a new time point into TPS using B+ Tree Insertion
8           End if
9           If (e.t  t for any time point in TPS) Then
10              t.freq++
11          Else
12              add a new time point into TPS using B+ Tree Insertion
13          End if
14      Next
15  End Procedure
```

그림 6 시점 시퀀스의 생성 알고리즘

위해 필요한 것으로 시점 시퀀스의 갱신에 사용된다. 시점 시퀀스의 실제적인 구현은 B+ 트리[12]를 이용하였다. 즉 새로운 시점을 추가할 때 B+ 트리의 삽입 알고리즘을 사용한다. 모든 사건들을 읽은 후에 B+ 트리의 리프 노드에는 시점들이 정렬된 순서로 유지된다.

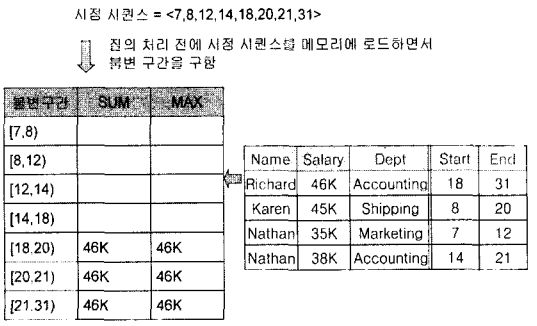
그림 2의 (a) 예제 릴레이션에서 시점 시퀀스를 구해보자. 시퀀스에 저장되는 시점은 데이타의 시작 시간과 종료 시간만 해당되고, 이것이 시점 집합이다. 따라서 그림 2의 (a) 예제 릴레이션에서 사건들을 읽은 순서대로 시점들의 집합을 구성하면 TS = {18,31,8,20,7,12,14, 21}이 된다. 시점 시퀀스는 시점들의 집합의 정렬된 순차이므로 시점 시퀀스 = <7,8,12,14,18,20,21,31>이 된다.

3.2 시점 시퀀스를 이용한 시간지원 집계의 처리

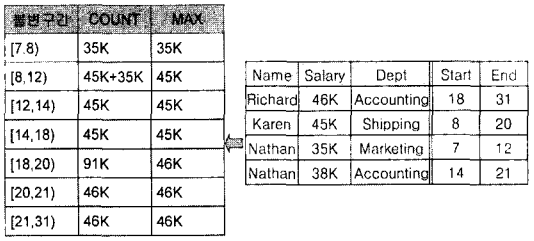
미리 만들어 놓은 시점 시퀀스는 시점들을 시간 축으로 정렬한 것이다. 따라서 이 시점 시퀀스에 있는 시점들을 차례대로 이웃한 2개씩 쌍을 만들면 그 사이의 연속된 시간 구간이 불변 구간이 된다. 시간지원 집계 질의 요청이 오면 미리 만들어서 유지하고 있던 시점 시퀀스에서 불변 구간을 구성한 다음 이를 메모리에 읽어 들여서 유지한다. 그런 다음 시간지원 집계 질의가 수행되어야 하는 릴레이션의 사건들을 읽어 들인다. 이 때 사건의 시작 시간과 종료 시간을 포함하는 모든 불변 구간에 대해서 질의가 요청하는 집계 연산을 수행하면 된다. 모든 사건들을 다 읽어 들인 후 최종적으로 생성된 값이 시간지원 집계의 결과가 된다. 시점 시퀀스를 이용한 집계 처리 알고리즘은 그림 7과 같다.

```
1   Procedure Compute_Temporal_Aggregate() Begin
2       Load TPS into Memory and make constant interval
3       For each event e in a relation do
4           For each interval that contains {e.t , e.t}
5               update aggregate_value of that interval
6           Next
7       Next
8   End Procedure
```

그림 7 시점 시퀀스를 이용한 시간지원 집계 처리 알고리즘



(a) 첫 번째 사건(event)을 읽었을 때



(b) 세 번째 사건(event)까지 읽었을 때

그림 8 시점 시퀀스를 이용한 시간지원 집계의 처리

그림 8은 그림 2의 (a) 예제 릴레이션에 대하여 그림 2의 (b)와 같이 SUM과 MAX 시간지원 집계 질의를 시점 시퀀스를 이용하여 처리하는 과정을 보여주고 있다. 먼저 시점 시퀀스를 메모리에 로드하면서, 정렬되어 저장된 시점들을 읽어서 불변 구간을 만든다. 그런 다음 해당되는 릴레이션의 사건들을 읽는다. 유효 시간이 [18,31)인 첫 번째 사건을 읽어 들이면, 메모리에서 [18,31)이 포함하는 구간인 [18,20)과 [20,21), [21,31)을 찾아서 그림 8의 (a)와 같이 SUM 값과 MAX를 46K로 한다. 두 번째 사건을 읽어 들이면 [8,12)과 [12,14), [14,18), [18,20)의 구간의 값이 변경되고, 세 번째 사건을 읽어 들이면 [7,8), [8,12)의 구간의 값이 변경된다. 두 번째, 세 번째 사건을 읽은 후, [18,20), [8,12)의 구간의 SUM은 91K, 80K가 되고 MAX는 45K, 46K가 된다. 세 번째 사건까지 읽었을 때 중간 결과는 그림 8의 (b)가 된다. 나머지 구간도 사건을 읽을 때마다 같은 방식으로 해당되는 집계를 반영하여 중간 결과들을 갱신한다. 모든 사건들을 다 읽으면 집계에 대한 최종 결과가 된다. COUNT, AVG, MIN의 경우에도 위와 같은 방법으로 처리할 수 있다.

3.3 시점 시퀀스의 갱신

시점 시퀀스는 미리 데이타베이스를 스캔하여 사건의 시작 시간이나 종료 시간만을 정렬하여 유지하고 있다. 따라서 데이타베이스에 새로운 사건이 삽입되거나 기존의 사건이 삭제될 때 시점 시퀀스도 함께 갱신되어야 정확한 불변 구간에 대한 정보가 된다.

시점 시퀀스에 저장되는 시점은 하나의 사건에 의해서 발생한 경우도 있고, 두 개 이상의 사건에 의해 발생할 수도 있다. 두 가지 경우의 구분을 위해서 시점 시퀀스에 저장된 시점들마다 부가적으로 시점을 발생시킨 사건의 개수를 기록하는 발생 빈도 필드가 필요하다. 이 필드는 시간지원 집계를 처리하는 과정에서는 필요하지 않고, 시점 시퀀스를 갱신하는 과정에만 필요하다.

새로운 사건의 삽입이 이루어진 경우, 그 사건의 시작 시간과 종료 시간만을 읽어, B+ 트리의 검색 연산을 이용하여 시점 시퀀스에서 해당되는 시점이 있는지 검사한다. 이미 존재하고 있는 시점인 경우 그 시점의 발생 빈도 필드의 값을 1 증가시키고, 기존에 존재하지 않고 새로 추가해 주어야 되는 시점이면 시점 시퀀스에 B+ 트리의 삽입 연산을 이용하여 그 시점을 추가하고 발생 빈도는 1로 초기화한다. 기존 사건의 삭제가 발생한 경우, 우선 그 사건의 시작 시간과 종료 시간에 해당되는 시점을 시점 시퀀스에서 찾는다. 그런 다음 그 시점들에 해당되는 발생 빈도 필드를 검사하여 발생 빈도가 1인 것은 삭제를 하고, 발생 빈도가 2 이상인 시점은 삭제하지 않고 발생 빈도를 1 감소시키고 시점 시퀀스에는 그대로 유지한다. 시점 시퀀스의 갱신 알고리즘은 그림 9와 같다. 시점 시퀀스의 갱신은 B+ 트리의 연산을 사용하여 이루어진다. 따라서 B+ 트리의 최소 차수(minimum degree)가 t라고 하고 B+ 트리에 N개의 카들이 있다고 할 때, 시점 시퀀스의 갱신을 위해서 $O(\log_t N)$의 디스크 접근이 필요하고, CPU 시간은 $O(t\log_t N)$의 복잡도를 가진다.

예를 들어 그림 10의 (a)와 같이 [14, 26)동안 유효하고, 직원의 이름이 Mike이고 월급이 50K인 사건이 새로 삽입되었다면, 이 데이타로 인해 기존의 [21,31)의 불변 구간이 [21,26), [26,31)의 불변 구간으로 더 나누어진다. 따라서 새로운 불변 구간을 정확히 유지하기 위해서 그림 10의 (b)와 같이 기존의 시점 시퀀스에 새 시점 14와 26을 추가하여 시점들을 새로 정렬하는 시점 시퀀스의 갱신이 필요하다. 이때 시점 26은 시점 시퀀스에 존재하지 않는 시점이므로 B+ 트리 삽입 연산으로 새로 시점 시퀀스에 추가하고, 시점 14는 이미 시점 시퀀스에 존재하고 있기 때문에, 시점 20에 해당되는 발생 빈도 필드를 1 증가시킨다. 그림 10의 (c)와 같이 기존 사건의 삭제가 발생한 경우에도, 삭제된 사건의 시작 시간과 종료 시간에 해당되는 시점을 찾아 시점 시퀀스를

```
1  Procedure Update_TPS(event e, operation oper) Begin
2      // process start time e.ts
3      If(oper is insertion ) Then
4          If any time point t same as e.ts is founded
5              t.freq++
6          Else
7              add new time point into TPS using B+ Tree Insertion
8          End if
9      End if
10     If(oper is deletion) then
11         If (any time point t same as e.ts is founded and t.freq =>2 ) then
12             t.freq--
13         Else
14             delete time point e.ts from TPS using B+ Tree Deletion
15         End if
16     End if
17    // process end time e.te with the same manner of processing start time e.ts
18 End Procedure
```

그림 9 시점 시퀀스의 갱신 알고리즘

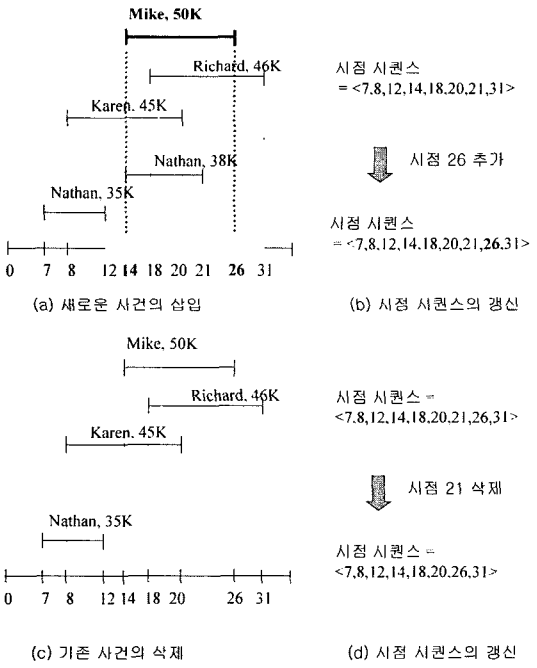


그림 10 사건의 삽입, 삭제에 따른 시점 시퀀스의 갱신

갱신한다. 시점의 발생 빈도 필드가 2 이상이면 발생 빈도만 1 감소시키고, 발생 빈도가 1이였다면 B+ 트리 삭제 연산을 이용하여 시점 시퀀스에서 삭제한다. 시점 14는 발생 빈도가 2이므로 시점 시퀀스에 계속 유지하고, 시점 21은 발생 빈도가 1이므로 삭제된다.

4. 실험 및 성능 평가

이 장에서는, 제안한 시점 시퀀스를 구현하고, 이를 기존의 처리 방법과 비교한다. 기존의 시간지원 집계의 처리 기법 중 AVL 트리의 균형화 기법을 사용한 PA-트리 기법과 레드-블랙 트리의 균형화 기법을 사용한 균형화된 트리 기법을 시점 시퀀스의 처리 방법과 비교한다. SUM 시간지원 집계를 선택하여 인위적으로 생성한 데이타를 가지고 실험을 수행한다.

4.1 실험 환경

실험은 Intel Pentium II MMX DUAL CPU 기계에서 수행하였다. 이 기계는 256 MB의 메모리를 가지고 있으며, 리눅스 커널 2.2.14에서 gcc를 이용하여 프로그램을 작성하였다. 실험의 성능 척도로서 경과된 시간을 측정하였다.

[7]의 논문에서 사용된 것과 같은 방법으로 인위적으로 실험 데이타를 생성하였다. 실험에 사용된 모든 데이타의 시간 구간 범위는 [1, 1000000]에 포함된다. 먼저 데이타의 시작 시간이 균일 분포(uniform distribution)를 이루도록 생성한다. 다음에 데이타의 종료 시간을 생성하였다. 이 때, 데이타의 시간 구간 범위를 단기간과 장기간의 두 부류로 고려하였다. 단기간 데이타는 시간 구간 범위가 1에서 1000 사이의 길이를 가지도록 랜덤하게 생성하였고, 장기간 데이타는 시간 구간 범위가 200000에서 800000 사이의 값을 가지도록 랜덤하게 생성하였다.

실험 과정에서 사용된 파라미터는 시간지원 데이타의 개수와 장기간 데이타의 비율이 된다. 시간지원 데이타의 개수는 10000개부터 시작하여 20000, 40000, 80000, 160000, 320000, 640000개까지 2배씩 증가하여 생성하였다. 각각의 데이타 개수마다 장기간 데이타의 비율이 0%, 10%, 20%, 40%, 80%가 되는 데이타들의 집합을 생성하였다. 각각의 데이타 집합에서, 사건들에 대한 정보는 시작 시간 속성(4 바이트)과 종료 시간 속성(4 바이트), 집계 연산이 적용되는 속성(4 바이트), 집계가 적

용되지 않는 속성(8 바이트)으로 이루어져 총 20 바이트로 저장된다.

4.2 실험 결과의 분석

실험은 기존의 기법 중 PA-트리 기법과 균형화된 트리 기법을 시점 시퀀스 기법과 비교하였다. 시간지원 집계 연산자중 SUM 질의를 처리하는 시간을 비교하였다. 비교를 하는 세 기법이 모두 시점들만 저장하기 때문에 장기간 데이타의 비율에 따른 실행 시간의 차이는 크지 않았다. 따라서 이 논문에서는 장기간 데이타의 비율이 80%일 때의 결과만을 포함했다.

단일 질의를 처리할 때 집계의 대상이 되는 사건의 개수가 N이라고 하자. 하나의 누적형 집계 질의를 처리하기 위해 PA-트리 기법과 균형화된 트리 기법은 기존에 있던 AVL 트리의 균형화 기법과 레드-블랙 트리의 균형화 기법을 이용하여 트리를 생성하고, 이들을 중위 순회하여 결과를 계산하므로 최악의 처리 시간은 $O(NlogN)$이다. 하나의 선택형 집계 질의를 처리하기 위해서 PA-트리는 트리 생성후 후위 순회를 통해 스카이라인 문제를 처리하므로 시간 복잡도는 $O(NlogN)$이고, 균형화 기법은 히프를 사용하므로 역시 시간 복잡도는 $O(NlogN)$이다. 시점 시퀀스는 B+ 트리에 기반을 둔 구조이기 때문에, B+ 트리의 최소 차수(minimum degree)가 t라고 하면 생성에 걸리는 시간은 $O(t\log_t N)$이다. 집계 질의 처리를 하기 위해서는 시점 시퀀스를 메모리에 올린 다음 해당되는 구간을 찾아서 변경을 하게 되므로 $O(t\log_t N)$이 소요된다.

그림 11은 각각의 기법마다 하나의 SUM 시간지원 집계 질의를 처리하는 시간을 비교하였다. 기존의 기법들은 메모리에서 시간지원 집계 질의에 해당되는 트리를 만들고 결과를 구하지만, 시점 시퀀스 기법에는 메모리에서 질의를 처리하는 시간 이외에 시점 시퀀스를 생성하고 디스크에 저장하는 시간이 필요하기 때문에 기존의 PA-트리 기법이나 균형화된 트리 기법보다 시간이 더 걸리는 것이다.

그림 12는 시점 시퀀스를 이용하여 하나의 SUM 시간지원 집계 질의를 처리하는 과정에서 메모리에서 실제 질의를 처리하는 시간과 시점 시퀀스를 생성하고 시점 시퀀스를 화일에 저장하는 시간을 알아본 것이다. 하나의 SUM 질의를 처리하는 과정에서 시점 시퀀스를 생성하는 비용이 메모리에서 질의를 처리하는 시간보다 큰 것을 알 수 있다.

집계의 대상이 되는 질의가 K번 들어오게 되면, PA-트리와 균형화된 트리는 $O(NlogN)$의 걸리는 처리를

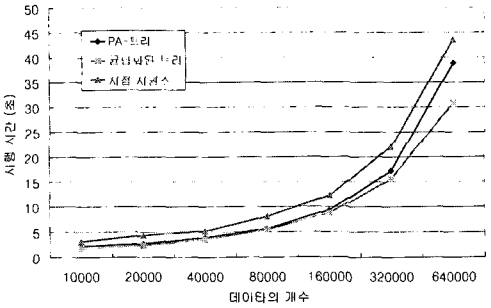


그림 11 1개의 SUM 질의를 처리하는 시간의 비교

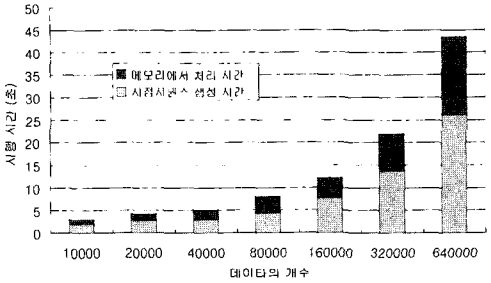


그림 12 시점 시퀀스를 이용한 하나의 SUM 질의 처리 시 시간의 비율

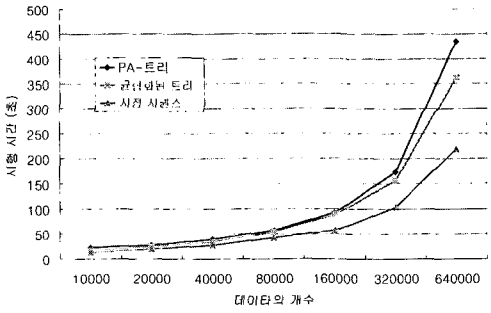


그림 13 10개의 서로 다른 SUM 질의 처리 시간의 비교

K번 동안 반복하게 된다. 시점 시퀀스는 처음에 생성할 때 $O(t\log_t N)$의 시간이 걸리지만 시점 시퀀스의 불변 구간 정보를 이용하여 집계를 구하기 위해 K번 동안 $O(t\log_t N)$의 작업을 반복한다. 그림 13은 서로 다른 사건들을 대상으로 하는 SUM 시간지원 집계 질의가 10번 들어왔을 때, 수행 시간을 비교한 것이다. 실제 수행 시간을 보면 PA-트리와 균형화된 트리는 비슷한 시간이 소요됨을 볼 수 있고, 시점 시퀀스를 이용한 시간지원 집계 질의의 처리가 PA-트리나 균형화된 트리를 이

용한 기법보다 효율적임을 알 수 있다.

5. 결 론

이 논문에서는 시점 시퀀스를 제안하고, 이를 이용하여 시간지원 집계 질의를 처리하는 방법을 제안하였다. 또한 좀더 정확한 불변 구간에 대한 정보를 유지하기 위해 시간지원 데이타베이스에 새로운 사건의 삽입이나 기존 사건의 삭제에 따른 시점 시퀀스의 갱신 방법도 제시하였다.

기존의 시간지원 집계 처리 기법들은 시간지원 집계 질의가 요구되었을 때 불변 구간을 구하기 위해 트리를 생성하고, 트리의 노드들을 순회하여 결과를 계산한다. 그렇지만, 시점 시퀀스를 이용한 방법은 우선 불변 구간에 대한 정보로서 미리 시점 시퀀스를 생성해 놓는다. 시간지원 집계 질의 요청이 있을 때 시점 시퀀스를 메모리에 올리면서 불변 구간에 대한 정보를 얻고, 사건을 읽을 때 해당되는 불변 구간에 대한 값들을 갱신하는 방식으로 집계를 처리한다. 따라서 시점 시퀀스는 대상이 되는 사건들이 다른 시간지원 집계 질의가 계속해서 들어올 때 기존의 방법에 비해 효율적으로 처리할 수 있다.

참 고 문 헌

[1] TPC, TPC Benchmark™ D(Decision Support), Working draft 6.5, Transaction Processing Performance Council, Feb. 1994.

[2] A. Tansel, J. Clifford, S. Gadia, and et al., Temporal Databases: Theory, Design, and Implementation, Benjamin/Cummings, 1993.

[3] R.T. Snodgrass, S. Gomez, and E. Mackenzie. "Aggregates in the temporary query language TQuel," IEEE Transactions on Knowledge and Data Engineering, pp. 826-842, Oct 1993.

[4] R.T. Snodgrass, I. Ahn, G. Ariav, and et al. "The TSQL2 Temporal Query Language," Kluwer Academic Publishers, Boston, 1995.

[5] Epstein, R. "Techniques for Processing of Aggregats in Relational Database Systems," UCB/ERL M7918. Computer Science Department, University of California at Berkeley, Feb 1979.

[6] P. A. Tuma, "Implementing Historical Aggregates in TempIS," Master's Thesis, Wayne State University, Nov 1992.

[7] N. Kline and R.T. Snodgrass, "Computing Temporal Aggregates," In Proc. of the 11th Inter. Conference on Data Engineering, pp. 222-231, Taipei, Taiwan, Mar 1995.

[8] Jong Soo Kim, Sung Tak Kang, and Myoung Ho Kim, "Effective Temporal Aggregation Using Point-Based Trees," In Proc. of 10th International Conference on Database and Expert Systems Applications, Florence, Italy, Aug/Sep 1999.

[9] 강성탁, 김종수, 김명호, "시간지원 데이타베이스에서의 효과적인 시간지원 집계 처리 기법", 정보과학회논문지(B), 제26권, 제12호, pp. 1418-1427. 1999.

[10] Bongki Moon, Ines Fernando Vega Lopez and Vijaykumar Immanuel, "Scalable Algorithms for Large Temporal Aggregation", In Proc. of the 16th International Conference on Data Engineering, pp. 145-154, San Diego, CA, Mar 2000.

[11] Rudolf Bayer, Edward M. McCreight, "Organization and Maintenance of Large Ordered Indices," Acta Informatica 1, pp. 173-189, 1972.

권 준 호

1999년 서울대학교 컴퓨터공학과 학사
2001년 서울대학교 컴퓨터공학과 석사
2001년~현재 서울대학교 전기·컴퓨터공학부 박사과정. 관심분야는 시간지원 데이타베이스, XML 등

송 병 호

1985년 서울대학교 컴퓨터공학과 학사
1987년 서울대학교 컴퓨터공학과 석사
1994년 서울대학교 컴퓨터공학과 박사
현재 상명대학교 소프트웨어학부 부교수
관심분야는 데이타베이스, 멀티미디어 정보검색, 전자문서, 전자정부 및 정보화정책

이 석 호

정보과학회논문지 : 데이타베이스
제 30 권 제 2 호 참조