연속질의 처리 시스템을 위한 트리거의 슬라이딩 윈도우 지원

이근주[○] 진성일
충남대학교 컴퓨터공학과
{eggnzoo[○], sijin}@cnu.ac.kr

Supporting Sliding Windows of Trigger for Continuous Query Processing System

Keun-joo Lee[○] S.I. Jin
Department of Computer Engineering, Chung-nam National University

요 약

데이터 스트림(data stream)을 처리하기 위해서는 기본적으로 질의 대상이 되는 슬라이딩 윈도우에 대한 지원과 이에 대한 연속질의를 수행할 수 있어야 한다. 기존의 관계형 DBMS는 성능 문제로 인하여 데이터 스트림 처리에 한계가 있었으나 고성능 메인메모리 DBMS의 등장으로 빈번히 발생하는 스트림에 대한 충분한 질의 처리 능력을 갖추게 되었다. 본 논문에서는 메인메모리 DBMS기반에서의 데이터 스트림에 대한 연속질의 처리를 위해서 새로운 접근방법을 제공한다. 즉, 고성능 메인 메모리 DBMS의 높은 삽입과 갱신 성능을 전제로 트리거를 통한 슬라이딩 윈도우의 지원방법을 제시하고, 윈도우에 대한 연속질의는 응용에서 지원하되 효율적인 질의처리를 위해 저장프로시저를 적용한다. 이러한 메커니즘의 연속질의 처리 시스템은 CQL에서 정의한 세 가지 윈도우 유형을 모두 지원할 수 있다.

1. 서 론

기존의 응용 프로그램에서는 데이터 원본들이 주로 디스크나 테이프에 데이터가 들어있다는 관점에서 이들에 대한 순차접근이나 임의 접근이 가능하다고 보았다. 하지만 최근에 이르러 많은 응용 분야에 있어서 이러한 관점이 더 이상 유효하지 않게 되었다. 데이터는 연속적으로 생성되고, 시간에 따라 변화하며, 데이터의 양이 매우 많아 디스크에 저장된 형태로 모델링 되기 어려운 특성을 갖게 되었다[1,2]. 즉, 튜플의 입력단위가 하나의 완전한 테이블 형태가 아닌 연속적인 데이터 스트림의 형태를 갖는 것이다[1,2].

이러한 데이터 스트림은 네트워크 라인을 통해 입력이 되고 들어오는 순서대로 정렬되어, 연속적이며 실시간의 성질을 갖게 된다. 따라서 데이터 아이템의 도착 순서를 변경하는 것과 입력 스트림 데이터 전체를 모두 저장하여 한꺼번에 처리하는 것은 불가능하며, 도착하는 순서대로 지속적으로 처리를 해주어야 한다.

이런 연속적이고 빠르게 발생하는 스트림 데이터를 기존의 DBMS를 이용하여 처리하는 것은 공간 효율면이나 질의에 대한 응답시간 면에서 매우 비효율적이었다[1,2]. 또한 스트림 데이터를 생성하는 응용분야의 특성상 기존의 DBMS에서 지원하는 질의보다는 연속질의(continuous query)[8,9]가 적합하다. 이러한 문제점으로 인해서 유비쿼터스 환경에서의 데이터 스트림을 처리하기 위한 모델이 제시되었고[1], 이러한 분야에 대한 새로운 많은 이슈들이 발생하게 되었다[2-6].

그러나, 높은 삽입과 갱신성능을 갖춘 고성능 RDBMS가 있다면, 슬라이딩 윈도우에 대한 질의 수행을 지원함으로서 데이터 스트림에 대하여 충분한 질의 처리 성능을 기대할 수 있을 것이다. 최근 메인메모리 DBMS의 등장으로 인하여 이러한 고성능 RDBMS기반의 데이터 스트림 처리에 대한 몇몇 연구가 진행되고 있다[7,14]. 데이터 스트림을 처리하기 위해서는 기본적으로 질의 대상이 되는 슬라이딩 윈도우에 대한 지원이 필요하며, 이에 대한 연속질의를 수행할 수 있어야 한다. 따라서, 이러한 사항들을 메인메모리 DBMS에서 만족시킨다면 데이터 스트림에 대한 연속질의 지원이 가능하다.

본 논문에서는 고성능 메인메모리 DBMS의 높은 삽입과 갱신 성능을 전제로 트리거를 통한 슬라이딩 윈도우의 지원방법을 제시하고, 이에 대한 연속질의 지원은 응용에서 구현하되 저장 프로시저를 적용하여 효율적인 질의처리가 가능하도록 한다.

본 논문의 구성은 다음과 같다. 먼저 제 2장에서는 관련된 연구들을 기술하고, 제 3장에서는 본 논문에서 제안하는 연속질의 처리 시스템의 구조와 특징들을 살펴본다. 마지막으로 제 4장에서는 결론 및 향후 연구 내용을 기술한다.

2. 관련 연구

2.1 데이터 스트림 관리 시스템 (DSMS)

유비쿼터스 환경에서처럼 고속으로 대용량의 데이터들이 매우 빈번하게 입력 혹은 출력이 된다면 이를 기존의 전통적인 방법으로 처리하기엔 어느 정도 이상은 역 부족이었다. 이러한 문제점으로 인해서 유비쿼터스 환경의 데이터 스트림을 처리하기 위한 모델이 제시되었다[1]. 이 데이터 스트림 모델은 데이터 스트림이 다음과 같이 전형적인 저장 관계 모델과 다르다는 것을 전제로 한다[2].

- 스트림상의 데이터 요소들은 온라인상으로 도착된다.
- 시스템은 그 안에서 데이터 요소가 처리되는 순서에 구속받지 않는다.
- 데이터 스트림은 잠재적으로 크기에 구속 받지 않는다.
- 일단 데이터 스트림으로부터 한 요소가 처리되면, 그것은 버려지거나 또는 보관된다. 데이터 스트림의 크기와 관련된 작은 메모리에 저장되지 않는 한, 그것은 쉽게 검색 될 수 없다.

따라서 테이블 전체를 입력해서 전체의 결과를 한번에 출력하는 기존의 전통적인 데이터베이스에서의 처리방식은 비효율적이며 적용하기에 어려움이 따른다.

2.2 연속질의 언어

2.2.1 연속질의와 CQL

연속질의[8,9]는 기존 DBMS기반의 스탠딩질의(standing query)와는 많은 차이점이 있다. 스탠딩 질의는 현재 저장된 데이터에 대한 일회성의 질의(one-time query)인 반면 연속질의는 일정 기간 동안 지속적으로 사용될 수 있는 질의를 뜻한

다. 연속질의는 고정된 데이터 집합(Static stored set)에 전달되는 질의와는 달리, 윈도우(window)와 실행 간격(execution interval)이 함께 시스템에 주어진다. 윈도우와 실행 간격은 데이터 스트림과 함께 등장한 연산자이다. 기존 데이터베이스 환경에서의 질의는 한정된 테이블의 데이터에 대해서 이루어지므로 결과도 유한하다. 그러나 데이터 스트림 환경에서는 무한한 데이터 스트림이 질의 대상이 되므로, 질의대상을 제한하기 위하여 데이터 스트림 환경에서 제시된 것이 윈도우[10]이다. 윈도우는 사용자가 관심 있는 '최근 t시간까지 도착한 데이터' 또는 '최근에 도착한 n개의 데이터'등으로 정의된다.

스탠포드의 Stream 프로젝트에서는 SQL과 유사한 형태의 연속질의를 위한 언어인 CQL(Continuous Query Language)을 정의하고 있다[11,12]. CQL은 슬라이딩 윈도우에 대한 연속질의를 표현한 질의 언어이다. 슬라이딩 윈도우에 대한 표현은 SQL-99의 OLAP Function에 정의된 WINDOW절의 문법을 이용하여 표현하고 있다[13]. CQL은 SQL과 유사성이 높으며, 슬라이딩 윈도우에 대하여 지원하고 있다.

CQL에서는 슬라이딩 윈도우를 크게 세 종류로 구분한다. 첫째는 시간기반(time-based) 슬라이딩 윈도우로, 새로 삽입된 데이터에 대하여 일정 시간 범위를 윈도우로 지정한다. 두 번째는 튜플기반(tuple-based)슬라이딩 윈도우로, 최근 삽입된 데이터의 순서대로 일정수의 튜플들을 윈도우로 지정한다. 세 번째는 분할(partitioned) 슬라이딩 윈도우로, SQL의 GROUP BY구문과 동일한 방법으로 분할한 각 그룹에 대하여 튜플기반 슬라이딩 윈도우처럼 일정 수의 최신 삽입된 튜플들을 모아 윈도우로 지정한다. 이와 같은 세 가지형태의 슬라이딩 윈도우에 대하여 CQL의 표현을 살펴보기로 한다.

2.2.2 선형 고속도로 시스템

CQL예제를 소개하기 위해서 단순화된 선형 고속도로 시스템을 설명하고자 한다[11]. 이 시스템은 고속도로의 차량 소통을 원활하게 하기 위해 도로의 교통 상황에 따라 실시간으로 요금을 결정하는 시스템이다. 교통 상황을 파악하기 위해 고속도로를 이용하는 모든 차량은 위치와 속도를 중앙 서버에 전송하는 센서를 가지고 있다. 고속도로에 정체 구역이 발생하면 차량의 위치를 파악하여 정체 구간을 이용중인 차량에 금액을 부과하게 된다.

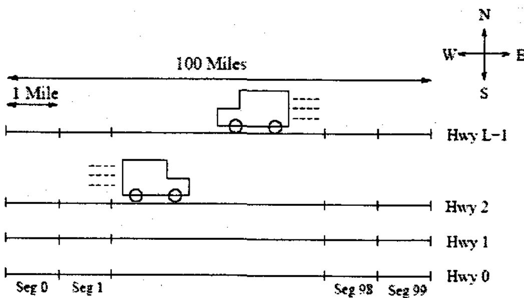

그림 1. 선형 고속도로 시스템

[그림 1]에서 보는 것처럼 고속도로는 동서로 100마일 길이의 직선 도로로 양방향으로 통행할 수 있으며, 총 L개의 고속도로가 존재한다. 각 고속도로는 1마일 단위로 100개의 영역으로 나뉘며, 각 영역에 입구와 출구가 존재하여 금액을 부과한다. 각 차량은 30초 이내의 시간 간격으로 차량 아이디(vehicleID), 고속도로(highway), 영역(segmentNo), 방향(direction), 이동속도(speed)의 정보를 중앙 서버에 보고하고, 중앙서버는 데이터가 입력된 타임스탬프와 함께 SegmentSpeed 스트림에 삽입한다.

2.2.3 시간기반 슬라이딩 윈도우

```
<Query 1>
  SELECT segmentNo, direction, highway
  FROM SegmentSpeed [Range 5 Minutes]
  GROUP BY segmentNo, direction, highway
  HAVING AVG(speed) < 40;
```

Query 1은 시간기반 슬라이딩 윈도우를 지정한 CQL의 예이다. 이것은 교통이 혼잡한 구간을 검색하는 질의로, 지난 5분간의 데이터에 대하여 통행차량들의 평균 속도가 시속 40마일 이하인 고속도로의 영역과 방향을 검색한다.

슬라이딩 윈도우의 범위는 [Range <시간 간격>]구문을 통해 지정한다. 이 때 윈도우의 범위는 질의가 수행되는 시점으로부터 지정된 시간 간격만큼 이전까지의 범위가 된다.

특별한 경우로는 [Range Unbounded]가 있다. 이는 윈도우의 범위를 제한하지 않는다는 의미로, 전체 데이터가 대상이 된다. [NOW]는 질의를 수행하는 시점의 데이터들만 윈도우의 대상으로 삼는 것으로 현재 삽입된 데이터들만 대상이 된다.

2.2.4 튜플기반 슬라이딩 윈도우

```
<Query 2>
  SELECT segmentNo, direction, highway
  FROM SegmentSpeed [Rows 1000]
  GROUP BY segmentNo, direction, highway
  HAVING AVG(speed) < 40;
```

Query 2는 튜플기반 슬라이딩 윈도우에 대한 질의로, 최신차량 이동 정보 1000개에 대하여 Query 1과 동일한 검색을 수행한다. Query 1에서 지난 5분간의 시간을 윈도우로 지정한 반면 Query 2에선 [Rows <튜플 수>]로 슬라이딩 윈도우를 지정하고 있다.

튜플기반 슬라이딩 윈도우에서의 특별한 경우로는 [Rows Unbounded]가 있다. 이는 윈도우의 범위를 제한하지 않는다는 의미로서, 전체 데이터들이 대상이 된다. 따라서 사실상 [Range Unbounded]와 동일한 영역의 윈도우를 갖는다.

2.2.5 분할 슬라이딩 윈도우

```
<Query 3>
  SELECT distinct L.vehicleID,
         L.segmentNo, L.direction, L.highway
  FROM SegmentSpeed [Range 30 Seconds] as A,
  SegmentSpeed [Partition by vehicleID Rows 1] as L
  WHERE A.vehicleID = L.vehicleID;
```

Query 3은 현재 고속도로를 이용중인 차량들의 위치를 검색하는 질의이다. 각 차량은 30초 이내의 간격으로 위치를 보고하므로, [Range 30 Seconds]윈도우는 현재 고속도로를 이용중인 차량들의 정보이며 일부 차량은 두 번 이상의 위치 정보가 포함되어 있다. 따라서 각 차량에 대하여 최종 정보만 다시 선택해야 한다.

이를 위해 Query 3에서는 분할 슬라이딩 윈도우를 사용하고 있다. 분할 슬라이딩 윈도우 [Partition by vehicleID Rows 1]은 차량 아이디로 그룹을 나눈 다음 각 그룹에 대하여 최근 1개 데이터를 선택한다. 따라서 차량별 최종 보고가 선택된다.

분할 슬라이딩 윈도우는 [Partition by <그룹 컬럼> Rows <튜플수>]로 지정할 수 있다. 이는 <그룹 컬럼>에 대하여 GROUP BY에서처럼 그룹을 나눈 다음, 각 그룹에서 <튜플수>만큼의 최근 데이터를 선택하는 것을 의미한다. 이 때 각 그룹에서 선택된 데이터가 하나의 분할 슬라이딩 윈도우가 된다.

2.2.6 연속질의의 반복주기 지정

Query 1, Query 2, Query 3의 질의들은 모두 데이터가 삽입될 때마다 수행하는 질의이다. 이러한 질의에 SLIDE <시간 간격>구문을 통해서 일정한 주기로 수행되도록 지정할 수 있다. 예를 들어 Query 1의 질의를 1분마다 수행하도록 지정하면 Query 4의 질의와 같이 표현할 수 있다.

```
<Query 4>
  SELECT segmentNo, direction, highway
  FROM SegmentSpeed [Range 5 Minutes Slide 1 Minutes]
  GROUP BY segmentNo, direction highway
  HAVING AVG(speed) < 40;
```

Query 4는 지난 5분간의 데이터에 대하여 통행 차량들의 평균 속도가 40 이하인 고속도로의 구역과 방향을 1분마다 주기적으로 검색한다. Query 1과 같은 시간기반 슬라이딩 윈도우뿐만 아니라 튜플기반의 슬라이딩 윈도우와 분할 슬라이딩 윈도우에서도 역시 이와 같이 반복주기를 지정할 수 있다.

3. 트리거와 저장프로시저를 이용한 연속질의 처리 시스템

3.1 기반 시스템 구조

본 논문에서 제안하는 데이터 스트림 처리를 위한 전체적인 구조는 [그림 3]과 같다. 여러 가지 타입의 센서 데이터들은 Event Manager를 통해 필터링과 조율, 일반화된 후 가공된 서비스 데이터를 생성하는 작업을 수행하게 되며, 이러한 스트림은 DBMS에 테이블 형태로 연속적으로 저장된다. 이때 전달되는 스트림은 그 유형에 따라 타임스탬프나 시퀀스 번호가 추가되어야 한다. 이것은 트리거를 사용하여 슬라이딩 윈도우를 지원하기 위한 정보로 사용된다.

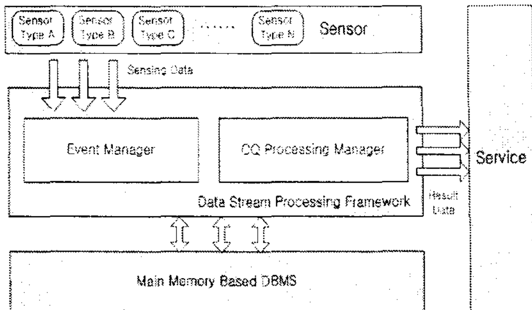

그림 3. 데이터 스트림 처리 구조

데이터에 타임스탬프나 시퀀스번호가 부여되는 시점은 크게 두 경우로 나누어 생각할 수 있다. 하나는 응용에서 부여되는 경우이고, 다른 하나는 데이터가 DBMS에 도착한 후 부여되는 경우이다. 본 시스템에서는 타임스탬프는 응용에서 부여하고 (자바의 TimeStamp클래스 사용), 시퀀스번호는 DBMS의 번호자동증가 기능(Kairos의 AUTO_INCREMENT 예약어)을 사용한다.

DBMS 내부적으로 트리거를 사용하여 연속적으로 삽입되는 스트림에 대한 윈도우를 유지하게 된다. 따라서 CQ Processing Manager에서는 이러한 윈도우에 대한 연속질의를 지원한다. 이때 질의는 저장 프로시저 형태로 미리 DBMS에 정의되며, CQ Processing Manager는 정의된 프로시저를 주어진 간격에 따라 호출하여 결과를 얻는다.

반복주기가 지정된 연속질의 시 시간기반 슬라이딩 윈도우에서의 트리거 로직은 별도의 저장프로시저 형태로 유지될 필요가 있다. 이것은 연속질의에 대해 유효한 결과를 얻기 위한 것으로서 상세설명은 해당 절에서 논한다.

3.2 트리거를 이용한 슬라이딩 윈도우 지원

트리거(trigger)는 지정된 DML(INSERT, DELETE, UPDATE 또는 이 작업들의 결합)으로 실행되는 특수형태의 저장 프로시저이다. 트리거가 주로 사용되는 경우는 두 가지이다. 하나는 업무규칙이 확실하게 지켜주도록 보장하기 위해서 사용되는 경우이고, 다른 하나는 데이터가 수정될 때 어떤 작업을 수행하기 위해서 사용되는 경우이다. SQL에서의 트리거는 3GL콜백함수나 훅된(hooked)인터럽트 벡터와 동일하다.

대부분의 저장 프로시저 프로그래밍에 대한사항들이 트리거에 동일하게 적용된다. 트리거에서 저장 프로시저를 호출할 수 있기 때문에 저장 프로시저가 할 수 있는 것들은 실제로 트리거에서도 거의 다할 수 있다. 트리거가 정상적으로 할 수 없는 한 가지가 있는데, 그것은 바로 결과를 반환하는 것이다. 대부분의 프런트엔드에서는 트리거가 만든 결과 집합을 처리할 수 있는 방법이 없기 때문에 트리거의 결과 집합을 볼 수 없는 것이다. 또한, 문법이 엄격한 일부 DBMS에서는 트리거에서의 SELECT 연산을 허용하지 않는다. 따라서 트리거를 통한 연속질의 지원은 DBMS의 특성에 따라 고려되어야 한다.

[그림 4]는 트리거를 사용하여 슬라이딩 윈도우를 지원하는 방법을 보여준다. 여기에서 워킹 테이블은 최초 데이터 스트림이 삽입되는 테이블을 말한다. 윈도우 테이블은 워킹 테이블의 데이터 중에서 관심대상이 되는 현재 시점으로부터 과거 어느 시점 혹은 몇 개까지의 데이터만을 유지하는 테이블을 말한다.

워킹 테이블에는 어떠한 인덱스도 구성하지 않고 데이터삽입이 발생하면, 트리거(1)에 의해 윈도우 테이블에 동일 데이터를 삽입하게 된다. 윈도우 테이블은 TS컬럼(혹은 SQ컬럼)에 인덱스가 구성되어 있고 데이터 삽입이 발생하면, 트리거(2)에 의해서 지정된 윈도우 구간을 벗어난 레코드들을 삭제한다. 이때 사용되는 DELETE구문의 WHERE절은 일정 범위를 나타내는 Range Query의 형태이지만, 데이터 삽입시마다 수행되는 구문이므로 튜플기반이나 분할 윈도우에서는 Point Query이고, 시간 기반윈도우에서도 데이터 삽입이 일정할수록 그 범위는 Point Query에 가깝다. 따라서 인덱스 유형이 클러스터되지 않은 인덱스라 할지라도 성능저하 가능성이 적다.

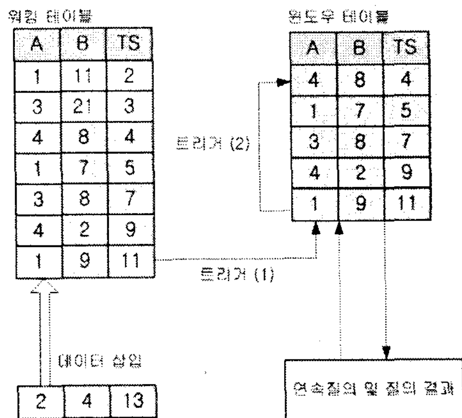

그림 4. 트리거를 이용한 슬라이딩 윈도우 지원

워킹 테이블은 삽입되는 전체 데이터를 유지하므로 백업DB로서의 역할도 하고, 실제 연속질의는 윈도우 테이블에 대하여 수행된다. 이미 트리거를 통해 윈도우 구간을 유지하고 있으므로 질의 수행 은 간결해 진다. 또한 슬라이딩 윈도우 인덱스와 같은 별도의 인덱스 없이도 분할 슬라이딩 윈도우에 대한 지원이 가능하다. 앞서 살펴본 CQL에서 윈도우 구간 설정 시 [Range Unbounded]와 [Rows Unbounded]는 윈도우의 범위

를 제한하지 않는다는 의미라고 하였는데, 이에 대한 질의는 워킹 테이블을 통해 지원될 수 있을 것이다.

다음은 관련연구에서 소개된 선형 고속도로 시스템의 CQL을 기준으로 세 가지 윈도우 유형에 대한 각각의 지원방식을 살펴본다.

3.2.1 시간기반 슬라이딩 윈도우 지원

트리거에는 'new'와 'old'라는 두 논리적 테이블이 존재한다. 이들 두 테이블은 삽입된 행을 저장하고, 삭제된 행을 저장하는 임시적, 논리적 공간이다. 임시적이란 말은 해당 트랜잭션이 끝나면 없어진다는 말이고, 논리적이라는 말은 실제로 디스크 상에 존재하는 테이블은 아니라는 말이다. 한 테이블에 삽입을 하면 새로 삽입된 행은 'new'라는 가상의 테이블에 그 트랜잭션이 완전히 끝날 때까지 남아있게 된다. 마찬가지로 삭제를 하게 되면 삭제된 행은 'old'라는 가상의 테이블에 남아있게 된다. update를 하게 되었을 때는 update될 내용이 insert되고, 이전에 있던 내용이 delete되는 식으로 동작하므로 'new'와 'old'테이블에 관련된 모든 행들이 남게 된다.

```
<TRIGGER 1>
CREATE OR REPLACE TRIGGER trg_SegmentSpeed
AFTER INSERT ON segmentspeed
FOR EACH ROW
DECLARE                          /* SQ시 정의 안함 */
w_TS NUMERIC(20) := :new.TS;     /*       "       */
BEGIN
    INSERT INTO segmentspeed_wdw (
    vehicleID, highway, segment, direction, speed,
    TS    /* 또는 SQ  */
    ) VALUES (
    :new.vehicleID, :new.highway, :new.segment, :new.direction,
    :new.speed,
    :w_TS    /* 또는 :new.SQ  */
    );
END;
```

TRIGGER 1은 앞서 설명한 선형고속도로 시스템을 기준으로 시간기반 슬라이딩 윈도우 지원을 위해 워킹 테이블에서 삽입 이벤트가 발생하였을 때 동작하도록 작성한 트리거이다. 여기에서 워킹 테이블은 SegmentSpeed이고 윈도우 테이블은 Segmentspeed_Wdw가 된다. 워킹 테이블의 트리거는 삽입된 정보를 'new'테이블에서 가져와 이것을 다시 윈도우 테이블인 Segmentspeed_Wdw테이블에 삽입을 수행한다.

```
<TRIGGER 2>
CREATE OR REPLACE TRIGGER Trg_Segmentspeed_Wdw
AFTER INSERT ON segmentspeed_wdw
FOR EACH ROW
DECLARE
w_TS NUMERIC(20) := :new.TS - 300000;
BEGIN
DELETE FROM Segmentspeed_Wdw
WHERE :w_TS >= TS;
END;
```

TRIGGER 1로부터 윈도우 테이블인 Segmentspeed_Wdw테이블에 삽입이 발생하면 TRIGGER 2에서는 주어진 시간만큼의 윈도우 구간을 유지할 수 있도록 DELETE구문을 수행한다. 'new'테이블에서 얻은 TS는 삽입이 발생하는 시점을 기준으로 현재 시간을 나타내는 타임스탬프 값이다. 여기에서는 타임스탬프 값을 mili-second단위로 하였으므로, 300000은 5분(5*60*1000)을 말한다. 현재 시간으로부터 5분을 뺀 것보다 작은 구간을 삭제하므로써 최근 5분간의 윈도우를 유지하게 되고 이것은 Query 1에서의 윈도우 구간과 동일하다.

3.2.2 튜플기반 슬라이딩 윈도우 지원

튜플기반 슬라이딩 윈도우에서 워킹 테이블에 정의 되는 트리거는 TRIGGER 1과 동일하다. 다만 현재 삽입된 레코드부터 주어진 수만큼의 레코드만을 유지해야 되므로 TRIGGER 1의 주석과 같이 타임스탬프 값 대신 시퀀스번호를 사용한다.

```
<TRIGGER 3>
CREATE OR REPLACE TRIGGER Trg_Segmentspeed_Wdw
AFTER INSERT ON segmentspeed_wdw
FOR EACH ROW
DECLARE
w_SQ INT := :new.SQ - 1000;
BEGIN
DELETE FROM Segmentspeed_Wdw
WHERE :w_SQ >= SQ;
END;
```

TRIGGER 3은 윈도우 테이블에 정의되는 트리거로서 Segmentspeed_Wdw테이블에 삽입이 발생하면 주어진 윈도우 구간을 유지할 수 있도록 DELETE연산을 수행한다. 'new'테이블에서 얻은 SQ컬럼 정보는 트리거 수행시점에 삽입된 데이터에 대한 시퀀스 번호이다.

시퀀스 번호는 데이터가 삽입될 때마다 순차적으로 1씩 증가하여 주어지며, new.SQ로부터 1000을 뺀 것보다 같거나 작은 구간을 삭제함으로써 최신차량 이동 정보 1000개에 대한 Query 2의 윈도우와 동일한 윈도우 구간을 유지한다.

3.2.3 분할 슬라이딩 윈도우 지원 (1)

워킹 테이블에 정의되는 분할 슬라이딩 윈도우에서의 트리거는 시간과 튜플기반의 TRIGGER 1과 다른 형태를 취한다.

```
<TRIGGER 4>
CREATE OR REPLACE TRIGGER trg_SegmentSpeed
AFTER INSERT ON segmentspeed FOR EACH ROW
DECLARE
w_SQ2 INT; w_temp INT;
BEGIN
    SELECT (MAX(SQ2)+1) into :w_temp
    FROM Segmentspeed_Wdw
    WHERE vehicleID = :new.vehicleID;
    if(w_temp IS NULL) then
        w_SQ2 := 1;
    else
        w_SQ2 := w_temp;
    end if;
    INSERT INTO segmentspeed_wdw (
    vehicleID, highway, segment, direction, speed,
    SQ, SQ2 ) VALUES (            /* SQ를 TS로 */
    :new.vehicleID, :new.highway, :new.segment,
    :new.direction, :new.speed,
    :new.SQ, :w_SQ2);                  /* :new.SQ를 :new.TS로
*/
END;
```

TRIGGER 4는 차량 아이디인 vehicleID로 그룹을 나눴을 경우 워킹 테이블에 정의되는 트리거의 예를 보여준다. 윈도우 테이블에서는 주어진 컬럼으로 그룹을 나눈 다음, 각 그룹에 대하여 주어진 수의 데이터를 유지해야 하기 때문에 SQ2컬럼과 같은 별도의 시퀀스 번호가 필요하다. 먼저 TRIGGER 4는 삽입이벤트가 발생하면 'new'테이블로부터 방금 삽입된 데이터를 가져온다. 그리고 이 중에서 vehicleID에 해당되는 값과 같다는 WHERE절 조건으로 윈도우 테이블에서 SQ2컬럼의 최대값을 가져와 1을 더해 변수 w_temp에 저장한다. 조건을 만족하는 SQ2값이 없다면 w_temp는 널이 되고, w_SQ2의 값은 1이 설정된다. w_temp가 널이 아니면 이 값은 w_SQ2의 값이 되고, 이것은 워킹 테이블에 삽입된 'new'테이블의 데이터들과 함께 윈도우 테이블인 Segmentspeed_Wdw테이블에 삽입된다. 이렇게 되면 윈도우 테이블의 데이터들은 정해진 각 그룹별로 순차적인 일련번호인 SQ2가 부여된다. 즉, 각 분할 윈도우별로 일련번호가 부여되는 것이다.

```
<TRIGGER 5>
CREATE OR REPLACE TRIGGER Trg_Segmentspeed_Wdw
AFTER INSERT ON segmentspeed_wdw
FOR EACH ROW
DECLARE
w_SQ2 INT;
w_vehicleID VARCHAR(100) := :new.vehicleID;
BEGIN
w_SQ2 := :new.SQ2 - 1;
DELETE FROM Segmentspeed_Wdw
WHERE :w_SQ2 >= SQ2
AND vehicleID = :w_vehicleID;
END;
```

TRIGGER 5는 vehicleID로 그룹을 나눴을 경우 윈도우 테이블에 정의되는 트리거의 예이다. TRIGGER 4에 정의된 형식에 따라 윈도우 테이블인 Segmentspeed_Wdw테이블에 삽입이 발생하면 'new'테이블로부터 현재 삽입된 데이터 중 SQ2와 vehicleID컬럼의 값을 가져와 각각의 변수에 할당한다.

윈도우 테이블의 트리거는 각 그룹에 대해 정해진 튜플 수 만큼의 최근 데이터를 선택하게 되는데, 정해진 튜플 수가 1이라면 TRIGGER 5와 같이 방금 삽입된 vehicleID와 같은 데이터에 대해서 w_SQ2에서 1을 뺀 것보다 같거나 작은 값을 WHERE절 조건으로 하는 DELETE구문을 수행하게 된다. 이렇게 되면 CQL예제의 Query 3과 같이 차량 아이디로 그룹을 나눈 다음 각 그룹에 대하여 최근 1개 데이터를 선택하는 [Partition by vehicleID Rows 1]과 동일한 분할 윈도우를 유지하게 된다.

분할 윈도우 테이블의 SQ2컬럼에는 클러스터된 인덱스를 정의해서는 안된다. 앞서 설명한 것처럼 DBMS에서 클러스터된 인덱스를 지원한다면 시간이나 튜플기반의 윈도우에선 유용하게 사용될 수 있다. 하지만 분할 윈도우는 각 그룹단위별로 시퀀스 번호를 부여하기 때문에 그 순서가 비순차적이다. 이러한 SQ2컬럼에 클러스터된 인덱스가 부여된다면 데이터페이지 전체를 재 정렬하는 오버헤드가 발생할 수 있다.

3.2.4 분할 슬라이딩 윈도우 지원 (2)

TRIGGER 4에서는 각 분할 윈도우별로 일련번호를 부여하기 위해서 윈도우 테이블로부터 주어진 분할에 해당하는 SQ2컬럼의 MAX값을 가져온다. 이러한 질의는 주어진 범위에서 최대값을 계산하는 동안 지연을 초래할 수 있다. 또한, Range Query형태이기 때문에 클러스터안된 인덱스는 도움이 안되며, 중간에 새로운 데이터가 삽입될 수 있으므로 클러스터된 인덱스는 지원될지라도 사용해선 안 된다.

표 1. 분할정보 테이블

vehicleID	SQ2
싼타페03	4
체어맨01	3
무쏘01	7
소나타01	2
그랜저01	6
아반떼01	0
싼타페01	5

이런 문제를 해결하기 위해 분할정보 테이블을 유지할 필요가 있다. [표 1]은 vehicleID로 그룹을 나눴을 경우의 분할정보 테이블의 예이다. 분할정보 테이블은 분할 윈도우 정의 시 생성된다. 분할정보 테이블은 두개의 컬럼 즉, 분할기준이 되는 컬럼과 SQ2컬럼으로 구성되고, 분할기준이 되는 컬럼에는 인덱스(클러스터 되지 않은)가 정의된다. 이 테이블에서 유지되는 SQ2컬럼의 정보는 윈도우 테이블에 주어진 분할 당 가장 큰 SQ2컬럼 값이 된다. 따라서 TRIGGER 4의 빗금 친 부분은 다음의 TRIGGER 5와 같이 변경한다.

먼저 분할정보 테이블인 Segmentspeed_PrtInfo테이블로부터 vehicleID에 해당되는 값과 같다는 WHERE절 조건으로 SQ2컬럼의 값을 가져와 1을 더해 변수 w_temp에 할당한다. vehicleID당 SQ2의 값은 하나만 유지되므로 이러한 질의는 Point Query가 되고 인덱스를 통해 빠른 검색이 가능할 것이다. 조건을 만족하는 SQ2값이 없다면 w_temp는 널이 된다. 이것은 그룹이 존재하지 않는 새로운 데이터가 삽입되는 경우이므로 INSERT구문을 통해 분할정보 테이블에 SQ2의 값을 1로 하여 삽입을 수행하고 w_SQ2의 값은 1이 할당된다. w_temp가 널이 아니면 그룹이 존재하는 새로운 데이터가 삽입되는 경우이므로 해당 그룹에 대하여 SQ2컬럼의 값을 w_temp로 하는 UPDATE구문을 수행하고, 이 w_temp는 w_SQ2의 값이 된다. w_SQ2는 워킹 테이블에 삽입된 'new'테이블의 데이터들과 함께 윈도우 테이블인 Segmentspeed_Wdw테이블에 삽입된다. 이렇게 되면 TRIGGER 4와 마찬가지로 윈도우 테이블의 데이터들은 정해진 각 그룹별로 순차적인 일련번호인 SQ2가 부여된다.

```
<TRIGGER 5>
............
   SELECT SQ2+1 into :w_temp
   FROM Segmentspeed_PrtInfo
   WHERE vehicleID = :new.vehicleID;
   if(w_temp IS NULL) then
      INSERT INTO Segmentspeed_PrtInfo (
      vehicleID, SQ2 ) VALUES (:new.vehicleID, 1);
      w_SQ2 := 1;
   else
      UPDATE Segmentspeed_PrtInfo
      SET SQ2 = w_temp
      WHERE vehicleID = :new.vehicleID;
      w_SQ2 := w_temp;
   end if;
............
```

3.3 저장프로시저를 이용한 연속질의 지원

3.3.1 반복주기가 지정 안 된 연속질의의 지원

슬라이딩 윈도우의 지원은 DBMS에서 제공되는 트리거 기능을 사용하는 반면, 연속질의는 응용에서 지원하되 DBMS의 저장프로시저를 적용한다. 따라서 모든 연속질의는 저장 프로시저 형태로 미리 DBMS에 정의 되어야 하고, 응용에서는 이 프로시저의 연속적인 호출을 통해 질의에 대한 결과를 얻을 수 있다. 반복주기가 지정이 되지 않았을 때 연속질의의 수행은 새로운 데이터가 삽입될 때마다 윈도우 테이블에 대한 질의를 수행한다. 이 때 윈도우 테이블에서 수행되는 트리거 액션은 워킹 테이블에 정의된 트리거의 삽입에 의한 것이고 워킹 테이블에서 수행되는 트리거 액션은 데이터 스트림의 삽입에 의한 것이다. 이와 같은 트리거를 중첩 트리거라고 하며 이러한 과정은 하나의 트랜잭션으로 유지된다. 이때 트랜잭션 수행의 지연으로 인해 연속질의의 수행시점에서 윈도우 테이블이 정확한 데이터를 가지지 못하는 경우가 발생할 수 있다. 이러한 경우에는 연속질의의 수행을 다소의 지연 시간이 지난 후에 처리하여, 윈도우 테이블에 트랜잭션의 처리가 반영된 이후 수행하도록 하면 된다. 이때 데이터 스트림의 삽입과 연속질의의 처리는 병행성(Concurrecy)을 유지해야 한다.

3.3.2 반복주기가 지정된 연속질의의 지원

반복주기가 지정 되었을 때 연속질의의 수행은 주어진 시간 간격에 따라 윈도우 테이블에 대하여 질의를 수행한다. 여기에서 한 가지 고려해야 될 사항이 있다. 튜플과 분할 슬라이딩

윈도우처럼 현 시점으로부터 주어진 수만큼의 데이터를 유지하는 윈도우는 데이터가 삽입되는 시점이나 질의를 수행하는 시점에서 유지 되어야 할 윈도우에는 차이가 없다. 그러나 시간기반 슬라이딩 윈도우의 경우는 그렇지 않다.

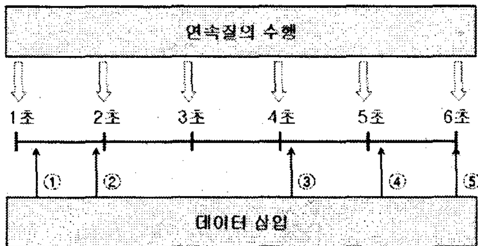

그림 5. 데이터 삽입에 대한 연속질의 수행 시점

[그림 5]는 시간기반 윈도우에서 질의가 수행되는 시간간격이 1초인 경우, 데이터 스트림의 삽입과 질의가 수행되는 시점을 보여주는 예이다. 각 데이터 삽입은 트리거의 이벤트로 작용하여 그 시점에 맞는 윈도우 테이블을 유지하게 된다. 데이터 삽입②와 데이터 삽입③사이에 질의는 세 번 수행되는데, 이때 질의의 대상이 되는 윈도우 테이블은 데이터 삽입②의 이벤트로 작용하는 트리거의 액션에 의한 것이다. 그렇기 때문에 각 질의 시점을 기준으로 수행되는 질의들은 해당 시점에 맞는 윈도우 구간유지를 위해서 별도의 작업을 거쳐야 한다.

```
<PROCEDURE 1>
CREATE PROCEDURE Proc_Segmentspeed_Wdw
(TS IN NUMERIC(20))
AS
w_TS NUMERIC(20) := TS - 300000;
BEGIN
DELETE FROM Segmentspeed_Wdw
WHERE :w_TS >= TS;
END;
```

PROCEDURE 1은 시간기반 윈도우에 대한 질의 수행시 함께 수행되는 저장 프로시저의 예이다. 이것은 TRIGGER 2와 동일한 로직을 갖는 프로시저로서 질의 수행 바로 전에 현재 타임스탬프 값을 매개변수로 하여 먼저 처리된다.

3.3.3 윈도우간의 조인 연산

윈도우간의 조인이 존재하는 연산에서는 일반 관계형 DBMS의 조인에서와 같은 방법으로 각 윈도우에 대해 조인을 수행하면 된다. 다만 반복주기가 지정된 연속질의에서 조인대상이 되는 윈도우중에 시간기반의 윈도우 테이블을 포함 한다면, 3.3.2절에서와 같은 방법으로 질의 수행 시점의 정확한 윈도우가 유지될 수 있도록 미리 정의된 프로시저를 호출해야 한다.

4. 결론 및 향후 연구 방향

데이터 스트림(data stream)을 처리하기 위해서는 기본적으로 질의 대상이 되는 슬라이딩 윈도우에 대한 지원과 이 윈도우에 대해 연속질의를 수행할 수 있어야 한다. 본 논문에서는 고성능 메인메모리 DBMS의 높은 삽입과 갱신 성능을 전제로 트리거를 통한 슬라이딩 윈도우의 지원방법을 제시하였다. 연속질의 역시 트리거에서 지원 가능하나 트랜잭션으로 동작하는 트리거는 그에 따른 부하를 가져올 수 있다. DBMS의 질의 프로세서를 사용하기 위해서 윈도우 지원을 위한 트리거의 사용은 불가피하지만, 연속질의 지원은 응용에서도 가능하다. 따라서, 연속질의는 응용에서 지원하되 효율적인 질의처리를 위해 저장프로시저를 적용하였다. 트리거로 인한 CPU부하는 다중처리 구조로 해소될 수 있을 것이다.

본 논문에서 제안한 메커니즘을 통해 구현된 연속질의 처리 시스템은 별도의 추가적인 인덱스 없이 DBMS의 기본 인덱스 지원만으로 CQL에서 정의한 세 가지 윈도우 유형을 모두 지원하게 되었다. 향후과제는 데이터 스트림을 처리하기 위한 메인메모리 DBMS의 최적화 연구이다. MMDBMS 역시 디스크 기반 DBMS와 같이 데이터 보호를 위해 로그화일을 디스크에 기록한다. 물론 그 방식이 훨씬 단순한 구조를 가지기 때문에 갱신연산 수행 시 디스크 기반 DBMS보다 약 10배가량 빠른 속도를 보인다. 제안된 구조에서는 삽입대상이 되는 워킹 테이블은 상황에 따라 디스크 로깅을 필요로 할 수 있지만, 질의 대상이 되는 윈도우 테이블은 로그에 기록될 필요가 없다. 이와 같은 데이터 스트림 각각의 특성을 고려하여 로그기록 방식을 최적화시킬 필요가 있다. 또한 효율적인 연속질의 지원을 위해서 메인메모리 DBMS에서의 클러스터된 인덱스 지원을 생각할 수 있다. 이것은 트리거에 의한 트랜잭션의 부하를 최소화하고, 테이블간의 조인 성능 향상에도 지대한 영향을 미치게 된다. 이러한 연구들이 향후 MMDBMS에 반영된다면 연속질의 처리 시스템의 질의수행 능력을 향상 시킬 수 있을 것이다.

참고문헌

(1) Lukasz Golab, M. Tamer Ozsu, "Data Stream Management Issues", University of Waterloo, Technical Report CS-2003-08 April 2003.
(2) B. Bobcock, S. Babu, M. Datar, R. Motwani, J. Widom, "Models and Issues in Data Stream Systems," Proc. of Symp. on PODS, 2002.
(3) J. Chen, D. J. DeWitt, F. Tian, Y. Wang, "NiagaraCQ : A scalable Continuous Query System for Internet Database", Proc. ACM SIGMOD Int'l Conf. on Management of Data, pp.379-390, 2002.
(4) S.Madden, M.Shah, J. Hellerstein, V.Raman, "Continuously Adaptive Continuous Queries over Streams", Proc. ACM SIGMOD Int'l Conf. on Management of Data, pp.49-60, 2002.
(5) S. Chandrasekaran, M. J. Franklin, "Streaming Queries Over Streaming Data", Proc. of Int'l Conf. on VLDB, 2002.
(6) D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, S. Zdonik, "Monitoring Streams - A New Class of Data Management Applications", Proc. of Int'l Conf. on VLDB, 2002.
(7) 백승천, 차상균, "트리거를 이용한 연속 질의 지원", 데이터베이스연구회 KDBC (SIGDB-KISS), 2004.
(8) L. Liu, C. Pu, W. Tang. "Continual Queries for Internet Scale Event-Driven Information Delivery", IEEE TKDE, vol.11, no.4, pp.610-628, 1999.
(9) D. Terry, D. Goldberg, and D. Nichols, "Continuous Queries over Append-Only Databases", Proc. ACM SIGMOD, pp. 321-330, 1992.
(10) L. Golab, M. Tamer Özsu, "Processing Sliding Window Multi-Joins in Continuous Queris over Data Streams",. Proc. VLDB, June 2003.
(11) A. Arasu, S. Babu, J. Widom, "The CQL Continuous Query Language: Semantic Foundations and Query Execution", Technical Report 2003-67, Stanford University, 2003.
(12) A. Arasu, S. Babu, J. Widom, "An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations" Technical Report, 2003.
(13) F. Zemke, K. Kulkarni, A. Witkowski, B. Lyle, "Introduction to OLAP Functions", ISO/IEC JTC1/SC32 WG3:YGJ-068 ANSI NCITS H2-99-154r2, 1999.
(14) 김수진, "주기억장치 상주형 DBMS를 활용한 데이터 스트림 관리 시스템 설계", 충남대 대학원, 2005.02.