一种支持流数据的查询语言①

涂金德 (南京航空航天大学 信息科学与技术学院 江苏 南京 210016;
温州职业技术学院 计算机系 浙江 温州 325035)

摘 要: 流数据的查询应用十分广泛，而标准 SQL 语言不支持这类查询功能，因此有必要对标准 SQL 语言进行扩展，以满足流数据的查询应用需求。支持流数据的查询语言 StreamSQL 在标准 SQL 语言的基础上增加了对流数据对象的处理机制，通过引入滑动窗口的概念，以支持流数据与关系表的相互转换操作，同时提供用户自定义函数功能，弥补了 SQL 在流数据处理方面的不足。

关键词: 流数据；连续查询；SQL；滑动窗口；关系数据库

StreamSQL: A Query Language for Stream Data

TU Jin-De
(College of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China; Computer Department, Wenzhou Vocational and Technical College, Wenzhou 325035, China)

Abstract: Stream data query is widely used, but the standard SQL language doesn't support it. It is hence necessary to expand the SQL in semantics. A processing mechanism of stream data object is added to the query language for stream data(StreamSQL). StreamSQL supports transformation between stream data and relations by introducing the concept of sliding windows. In addition, it offers the user-defined function to make up for the disadvantage of the standard SQL in stream data processing.

Keywords: stream data; continuous query; SQL; sliding windows; relation database

近年来，对实时数据流处理的研究引起了越来越多人的关注。流数据具有大数据量、数据变化频繁、需要快速响应、查询次数有限等特点，使得流数据的处理需要采用新的方式。在许多应用领域，诸如网络流量监测、网上拍卖、道路交通检测、传感器网络数据库[1,2]等，都需要对实时数据流进行处理。由于这些应用都要求在数据流上进行连续多次的查询，而传统的查询语言只能对存储在永久介质上的数据做单次查询，因此传统的数据查询语言 SQL 已经不能满足应用的需求，必须扩展 SQL 以适应新的应用。

国外很多著名的大学和研究机构都在开展流数据管理系统(Data Stream Management System)的研究，有的已经开发出了原型系统，例如加州大学伯克利分校的 Telegraph 系统、麻省理工大学的 Aurora 系统 、斯坦福大学的 STREAM 系统等。这些原型系统都定义了各自的查询语言。由于这些原型系统是针对不同的应用，因此他们的查询语言有各自的局限性。最早使用数据流查询的是 Tapestry 系统，它定义了基于 SQL 的查询语言 TQL，它实际上是对流快照的多次单次查询的结果进行合并(UNION)。Telegraph 系统定义的查询语言 TelegraphCQ[3]只能单独处理数据流，不能适用于同时涉及到数据流和关系表的应用场合。Aurora 系统定义的查询语言是一种程序性的查询语言，与基于关系的查询语言相比较为复杂[4]。STREAM 系统是一个通用的流数据管理系统，它的查询语言 CQL[5]能同时处理数据流和关系，但是在功能扩充方面没有提供支持。

根据查询规范，数据流查询语言可以分为三类：

① 收稿时间:2009-06-08

基于关系的查询语言、基于对象的查询语言和程序性查询语言[6]。总体来说，针对数据流的查询语言尚处于发展中，还没有一个完善的标准。为此，文中提出一种支持流数据的查询语言 StreamSQL。StreamSQL 是一种基于关系的查询语言，是对 SQL 语言的扩展。StreamSQL 在标准 SQL 语言的基础上增加了对流数据对象的处理机制，引入了滑动窗口的概念，以支持数据流与关系表的相互转换操作，并且支持用户自定义函数功能，弥补了上述各种查询语言在功能扩展方面的不足。

1　具有数据流处理能力的SQL扩展

数据流指的是由一系列连续且有序元组组成的序列[7]。按照固定的次序，这些元组只能被读取一次或者几次。数据流中的每一个元组，都有一个时间戳，数据流元素可表示为(s,t)。其中 s 是关系数据库表中的元组，t 为时间戳，表示元组的生成时间，一般由数据流源生成[8]。对于一个给定的时间戳 $t \in T$，数据流中可能有一个或者多个具有相同的时间戳 t 的元组，也可能没有，但是具有相同时间戳的元组数目是有限的。除了由数据流源(如某个传感器采集的)生成的数据流之外，数据流也可以由对关系表的查询产生。时间戳 t 是一个逻辑时间，不一定是时钟时间，它仅仅表示元组生成的先后顺序。

标准 SQL 处理对象是传统关系数据库中的关系，它不能直接应用于数据流的处理。首先数据流处理涉及一些与时间或次序有关的查询，标准 SQL 在表达能力上明显不足。其次，标准 SQL 中的一些操作，如 SUM、AVG 等，其查询操作是阻塞的。所谓一个查询操作是阻塞的，就是说操作输出第一个结果元组，当且仅当所有的输入都完成。而数据流是一个无限的连续的元组流，连续查询必须按要求返回结果，而不是等待所有输入完成，所以只有非阻塞的操作才适合处理数据流。非阻塞是在检测到输入的最后一个元组之前输出结果元组的操作[9]。为了使传统的数据库查询语言 SQL 能够处理流数据，必须对之作如下功能扩展：

(1) 将数据流转换为关系，再由 SQL 处理；

(2) 将一些阻塞的操作非阻塞化，使其能处理连续的数据流。

要做到上述功能扩展，首先定义数据流和关系之间的相互转换操作，然后定义用户自定义函数，用来重写阻塞的聚集函数和实现一些其它的功能。

2　数据流与关系的相互转换

StreamSQL 是基于关系和数据流这两种数据类型的，对这两种数据类型有四种操作。

(1) 数据流到关系的操作：数据流作为输入，关系作为输出。

数据流到关系的操作是基于滑动窗口的。滑动窗口的概念是在 SQL-99 中提出的，其中定义了多种滑动窗口(基于元组的窗口，基于时间的窗口，固定大小的窗口，基于属性的窗口)，我们的扩展中支持基于时间的窗口和基于元组的窗口。

基于时间的窗口以一个时间间隔 ϕ 作为参数作用在数据流 S 上[10]，具体形式如下：

CREATE RELATION s[RANGE ϕ] AS relation_name

若当前时间为 T，该操作是将数据流中时间戳介于 $T-\phi$ 和 T 之间的元组删去时间戳后，生成一个名为 relation_name 的关系。形式上可以定义为：

$$\text{relation\_name} = \{s \mid (s,t) \in S \wedge t \in [T-\phi, T]\}$$

另外需要说明两种边界情况，即 $\phi=0$ 和 $\phi=\infty$。当 $\phi=0$ 时，导出的新关系中只包含数据流中时间戳为 T 的元组，即 $\text{relation\_name}=\{s \mid (s,t) \in S \wedge t=T\}$；当 $\phi=\infty$ 时，新关系中包含数据流中所有时间戳小于等于 T 的元组，即 $\text{relation\_name} = \{s \mid (s,t) \in S \wedge t \leq T\}$。

基于元组的窗口以一个非负整数作为参数作用在数据流 S 上[11]，具体形式如下：

CREATE RELATION s[ROW I] AS relation_name

若当前时间为 T，该操作是取数据流中的当前时间戳最大的 I 个元组删去时间戳后，生成一个名为 relation_name 的关系。之前提到，数据流中可能存在相同时间戳的有限数目元组。基于元组的窗口，生成的关系是不确定的。有一种边界情况 $I=\infty$，等同于 $\phi=\infty$ 的情况，即 $\text{relation\_name} = \{s \mid (s,t) \in S \wedge t=T\}$。

(2) 关系到数据流的操作：关系作为输入，数据流作为输出。

前面提到过，数据流还可以由对关系表的查询产生。这就是关系到数据流的映射。具体形式如下：

CREATE STREAM stream_name AS {sql_ expression}

其中，stream_name 表示生成数据流的名称，sql_expression 表示 select-from-where 表达式。

(3) 关系到关系的操作：关系作为输入，关系作为输出。

使用标准 SQL 操作即可。

(4) 数据流到数据流的操作：数据流作为输入，数据流作为输出。

数据流到数据流的操作可以由(1)、(2)操作间接实现。

下面用例子说明具体使用：

设有原始数据流 Sensors。它表示传感器产生的数据流，它采集光强度、温度等一系列信息，以如下模式产生数据流：

Sensors(id, location, light, temperature, timestamp)

其中 id、location、light、temperature、timestamp 分别为传感器编号、传感器所处的位置、光强度数据、温度数据、时间戳 。

再设有原始数据流 Events，它表示当传感器检测到一些特殊事件，譬如附近有人活动，它就会以如下模式的数据流返回给用户：

Events(type, location, timestamp)

其中 type、location、timestamp 分别为事件类型、事件发生位置、时间戳 。

应用一：以 12 秒为间隔读取位置 2 的温度数据，并以数据流的形式传给下一级的应用。

实现：

```
/* 将 12 秒内传感器产生的数据流 Sensor 创建为关系 sensor_relation */
CREATE RELATION Sensor[RANGE 12]
  AS sensor_relation;
/* 从关系 sensor_relation 中读取位置 2 的温度创建为数据流*/
CREATE STREAM sensor_2_temp AS
  {SELECT temperature FROM sensor_relation
    WHERE location=2};
```

应用二：事件触发查询。当传感器检测到某位置有人活动的时候，该位置半径 8 米内的传感器以每秒一条数据的速度报告事件发生所处位置 5 秒内的平均光强度和平均温度。

实现：

```
/* 将 5 秒内传感器产生的数据流 Events 创建为关系 events_relation */
CREATE RELATION Events[RANGE 5s]
  AS events_relation;
/* 将 5 秒内传感器产生的数据流 Sensor 创建为关系 sensor_relation */
CREATE RELATION Sensor[RANGE 5]
  AS sensor_relation;
/* 读取平均光强度和平均温度创建为数据流*/
CREATE STREAM result AS
  {SELECT events_ralation.location, avg (light),
    avg(temperature)
    FROM evnents_relations, sensor_relation
    WHERE events_relation.type='human_dect'
AND
    disc(events_raltaion.location,sensor_relation)<=8};
```

disc( )是自定义函数，下面的内容会涉及它的定义。

3 自定义函数

为了将阻塞操作非阻塞化，使得 SQL 能够处理流数据，需要将阻塞的操作用自定义函数的方式重写。在 SQL-3 标准中，支持形如 INITIALIZE- ITERATE-TERMINATE 的自定义聚集函数(UDA)[12]。这里就采用与它相同的形式定义用户自定义函数(UDF)。但不同的是，文中的 INITIALIZE，ITERATE 和 TERMINATE 三个模块是用 SQL 来描述的，而不是用过程性程序语言(如 C 语言)描述。以下是重写的 AVG 函数：

```
FUNCTION my_avg(Next int) : float
{
  TABLE total(sum int, cnt int);
  INITIALIZE: {
    INSERT INTO total VALUES (Next, 1);}
  ITERATE: {
    UPDATE total SET sum=sum+Next,
cnt=cnt+1;
    INSERT INTO RETURN SELECT sum/cnt
      FROM total WHERE cnt % 500 = 0;}
  TERMINATE : {}
}
```

首先定义函数名为 my_avg，返回值为浮点型实数；然后定义了一个关系数据表 total，分别记录到目前为止处理元组的数目以及总和。在 INITIALIZE 模块中，做初始化工作，将第一个元组插入 total 表中，并将处理元组数置 1。具体工作主要是在 ITERATE 模块中完成。当连续的元组接连到达，不断更新 toatal 表，使处理元组数达到一定数目(本例为 500)时，返

回结果，这样就把阻塞函数 AVG 非阻塞化了。

自定义函数除了可以改写阻塞操作，还可以扩展新的操作，实现新的功能，如上述应用中的 disc()函数。下面用我们的方法来实现它，其中 sensor_loc 是关系数据表。

```
FUNCTION disc(location1 loc, location2 loc): float
{
  TABLE coordinate1(loc_id int, x float, y float),
  coordinate2(loc_id int, x float, y float);
  INITIALIZE: {}
  ITERATE: {
    INSERT INTO coordinate1 SELECT *
      FROM sensor_loc Where loc_id=
location1;
  INSERT INTO coordinate2 SELECT *
    FROM sensor_loc Where loc_id=
location2;}
  TERMINATE :{
  INSERT INTO RETURN SELECT sqt ((c1.x-
c2.x)*(c1.x-c2.x)+(c1.y-c2.y)*(c1.y-c2.y))
    FROM coordinate1 c1.coordinate2 c2;}
}
```

首先定义函数名为 disc，返回值为实数；然后分别定义了关系数据表 coordinate1 和 coordinate2。ITERATE 模块完成这二个数据表中传感器位置数据的插入，TERMINATE 模块完成二个传感器位置间距离的计算并返回该值。

4 结论

StreamSQL 对标准 SQL 进行了扩展，使其能够处理连续的流数据，并支持用户自定义函数，可以灵活扩展其功能。目前已使用词法分析工具 lex 和语法分析工具 yacc 实现了 StreamSQL 的编译器。在实际应用中，流数据常包含空间属性，如随着时间变化的车辆位置信息，以后有必要进一步扩展 StreamSQL 以支持与空间属性相关的查询。

参考文献

1 Stream Query Repository: Network Traffic Management. http://www-db.stanford.edu/stream/sqr/netmon.html

2 Stream project page for Linear Road Benchmark. http://www-db.stanford.edu/stream/cql-benchmark.html

3 Chandrasekaran S, Cooper O, Deshpande A, Franklin M, Hellestein M, Hong W, Krishnamurthy S, Madden S, Raman V, Reiss F, Shah M. Telegraph CQ: Continuous Dataflow Processing for An Uncertain World. Proc. of The First Innovative Data Systems Resear- ch, Asilomar, USA: 2003.269－280.

4 Abadi D, Carney D, Cetintemel U, Cherniack M, Convey C, Lee S, Stonebraker M, Tatbul N, Zdonik S. Aurora: A New Model and Architecture for Data Stream Management. VLDB Journal, August 2003, (12)2:120－139.

5 Arasu A, Babcock B, Babu S, Cieslewicz J, Datar M, Ito K, Motwani R, Srivastava U, Widom J. STREAM: The Stanford Data Stream Management System. http://dbpubs.stanford.edu/pub/2004-20

6 Chandrasekaran S, Franklin MJ. Streaming Queries over Streaming Data. VLDB Conference, Hong Kong: August 2002.

7 Madden SR, Shah MA, Hellerstein JM, Raman V. Continuously Adaptive Continuous Queries Over Streams. Franklin MJ, Moon B, Ailamaki A, eds. Proc. of the SIGMOD Conference. New York: ACM Press, 2002.49－60.

8 Abadi D, Carney D, et al. Aurora:A New Model And Architecture for Data Stream Management. The International Journal on Very Large Data Bases, 2003, 12(2):120－139.

9 Mozafari B, Thakkar H, Zaniolo C. Verifying and Mining Frequent Patterns from Large Windows Over Data Streams. International Conference on Data Engineering (ICDE), 2008.

10 Bai YJ, Thakkar H, Luo C, Wang HX, Zaniolo C. A Data Stream Language and System Designed for Power and Extensibility. CIKM, 2006. 337－346.

11 Tucker PA, Maier D, Sheard T, Stephens P. Using Punctuation Schemes to Characterize Strategies for Querying over Data Streams. IEEE TKDE. 2007, 19(9):1227－1240.

12 Brettlecker G, Schuldt H, Schek HJ. Towards Reliable Data Stream Processing with OSIRIS-SE. Proc. of BTW Conf., Karlsruhe, Germany: March 2005.405－414.