Revisiting Formal Ordering in Data Stream Querying

Loïc Petit
Orange Labs
University of Grenoble
LIG Laboratory, France
loic.petit@orange.com

Cyril Labbé
University of Grenoble
LIG Laboratory, France
Cyril.Labbe@imag.fr

Claudia Lucia Roncancio
University of Grenoble
LIG Laboratory, France
Claudia.Roncancio@imag.fr

ABSTRACT

The use of stream based applications is in expansion in many contexts and easy and efficient data stream management is crucial for such applications. That is why numerous solutions for stream query processing have been proposed by the scientific community. Several query processors exist and offer heterogeneous querying capabilities. This paper reports a formal work on the operators behind such query processing solutions. It points out the semantic heterogeneity of some important operators and how this leads to some kind of semantic ambiguity which may affect the application semantics. This paper revisits the definition of the main operators used for stream query processing and proposes definitions which are semantically unambiguous. The main issue is the positional order of data items in a stream and its propagation across the operators. The proposed formalization deepens the understanding of stream queries and facilitates the comparison of the semantics implemented by existing systems. This paper also presents the prototype implementing our formal proposal.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Algebraic approaches to semantics*

General Terms

Theory, Languages

Keywords

data stream, positional, temporal, order, algebra

1. INTRODUCTION

Stream based applications are used in many contexts, among them financial and sensor based environments. Stream data management has deserved the attention of the academic and industrial communities. Several projects propose different forms of declarative data stream processing (see section 2). The power of expression of such solutions differ and are difficult to compare. Moreover, the semantics of queries supported by different systems is, in some cases, ambiguous and a query executed by two stream processing systems may lead to different results. One important factor leading to such a situation is data ordering in streams. Both, temporal and positional orders are meaningful and the way they are handled during stream query processing is important and has an impact on the result. Formal operators, as defined up to now, are (or may be) semantically ambiguous in regard to the positional order.

This paper clarifies and formalizes main operators involved in stream queries and the propagation of tuple order along them. It provides a unified model to express unambiguous stream queries. It goes beyond previous work in the clarification of the semantics of tuple position for important operators such as window and joins of which the complexity has been underestimated.

This paper is organized as follows: Section 2 introduces related work and motivates our proposal by showing the positional order ambiguity problem. Section 3 present the core definitions of the formal model we propose for stream querying. Sections 4 and 5 presents the different operator definitions and how data orders are managed through them. Section 6 presents the implementation of our proposal. Section 7 gives our conclusions and research perspectives.

2. QUERYING STREAMS AND MODELS

Section 2.1 presents related work on stream data management and then Section 2.2 focuses on issues about data ordering in streams, which motivated our work.

2.1 Related work

The early years. Data streams were explored early in the SEQ model [20] where a stream is considered as a set of records (or tuples) with a positional order (without time model). This formalism has been used by severals [11, 5]. At first [23], continuous queries were considered as instantaneous queries executed periodically. Database relations were already used as relations varying over time. Later on, continuous queries over full streams were considered [14]. Firstly only full or instantaneous windows were used in these models, then more complex windows (sliding, fixed, tumbling) were introduced [21] following the two major criteria: time-based and count-based windows.

Network analysis and sensor monitoring motivated new models for data stream management [1, 17, 25, 8]. Sliding windows, aggregation and joins have been defined. However, semantics of these operators remain mainly driven by implementation leading to divergent interpretation and restrictions on the expressiveness of the model.

The two-fold approach. STREAM [2] and its semantic model [3] distinguishes two major concepts: on the one hand, **streams** as infinite sets of tuples with a common schema having a timestamp, on the other hand, **relations** as functions that map time to a finite set of tuples with a common schema. Given this, windows are stream to relation operators, streamers are relation to stream operators and relational algebra [7] operators remain relation(s) to relation operators. Stream to stream operations exist only as a composition of other operators. This proposal and the associated CQL language [4] have been used ever since in many stream projects [24, 13].

Rethinking formal models. Recently, the core basis of the aforementioned models has been proven to be semantically ambiguous. Therefore, standardization and clarification of the core semantics of some operators have been proposed [15]. The concept of **batch** as a set of tuples having the same timestamp, has been introduced. A new wave of formalization arises subsequently to fill the lack of mathematical models. Some works contribute to more complete formalization of windows [18, 6, 19] as they are the aspect of stream processing which has led to the most different interpretations.

2.2 Temporal & positional order

Data stream processing involves temporal and positional orders in streams. **The temporal order** is induced by the timestamp of tuples in a stream. As several data items may have the same timestamp (they are *simulatenous*) this order is not strict. **The positional** is a strict order induced by the "place" of data in the stream. Although the temporal order is specified by the timestamps the positional order must be transmitted through the query processing.

Need to explicit positional order. Let us consider temperature monitoring in a product warehouse. A temporal relation R(ID, SEC) informs that the product ID is currently in sector SEC. Temperature sensors feed a stream S(SEC, TEMP, T) indicating that at time T, the temperature of sector SEC is TEMP. Let's consider the following instances at time T = 60.

R

ID	SEC
1	2
2	23
3	23
4	12

S

SEC	TEMP	T
12	12	21
2	11	32
2	14	48
12	13	54

The user wants to get every 60 seconds the temperature of the products in the monitored sectors. At time T=60, the resulting stream is one of the following:

ID	TEMP	T
1	11	32
1	14	48
4	12	21
4	13	54

ID	TEMP	T
4	12	21
1	11	32
1	14	48
4	13	54

The first resulting stream, is obtained by iterating over R and then joining with a window on S. Therefore IDs are grouped in the resulting stream. The second resulting stream, would be obtained by iterating over the window on S, and then joining with R. Tuples are ordered by the original timestamp in the resulting stream.

The difference in data ordering in the stream has several impacts. Among them (1) if windows of limited size are created on the resulting stream then the selected tuples will not be the same [15]; (2) load-shedding policies will be impacted [22]; (3) the cost of evaluating some aggregation operators on such streams will differ.

The possible ambiguity of the positional order in streams may change the semantics of the result. The same problem arises in stream joins and unions: what will be the resulting order in the final stream? We state that this ordering may be altered in the stream processing especially during relational operations. Those were often considered as directly inherited from the relational algebra. We shall see that this assumption is wrong for properties like the join symmetry.

As a matter of fact, a good comprehension of querying mechanisms is a key point for the development of ubiquitous systems. A clearly defined model will enhance the comprehension of requirements and allow a better reusability, coupling and development of existing supports.

3. THE BASIS OF THE ALGEBRA

This section introduces the foundations of our algebra. The following presents the definitions concerning data representation. The operators are introduced in later sections.

3.1 Tuples and identifiers

Before going any further, let us recall the basic principles of strict formalization of tuples in the relational model.

DEFINITION 1 (TUPLE). *A tuple is a partial function from a finite subset of attribute names to atomic values. The domain of this function is called the schema of the tuple.*

As any partial function it is possible to represent a tuple as a set of couples of Attribute×Value (with a unique constraint on the attribute). We will often use this representation to manipulate tuples.

Now let us add a physical identifier to tuples. This particular attribute is intended to identify tuples even when the values of the other attributes are duplicated. It also allows the definition of a positional order in a set of tuples. We now consider the definition of physical identifier, tuple-set and the position of a tuple.

DEFINITION 2 (PHYSICAL IDENTIFIER). *The physical identifier of a tuple s is an element of the identifier space $\mathbb{I}$ isomorph to $\mathbb{N}$ and totally ordered. The name of this attribute will be denoted by φ and $s(\varphi)$ designates the value of the physical identifier of s.*

DEFINITION 3 (TUPLE-SET). *A tuple-set is a countable set of tuples sharing the same schema and including a physical identifier φ. The physical identifier of a tuple is unique in the tuple-set and induces a strict order of the tuples in the tuple-set. The common schema of a tuple-set TS is designated by $Attr(TS)$.*

Data stream management also requires timestamps to be associated to each tuple inside a stream. We adopt continuous time to allow general data management as it allows to make no hypothesis on the distance between two consecutive timestamps.

Definition 4 (Timestamp). *The time-space $\mathcal{T}$ is a field isomorphic to $\mathbb{R}$. The time-space is naturally and totally ordered. A timestamp t is an element of $\mathcal{T}$.*

As a recall, in a time-based system, tuples are grouped by timestamp. Therefore, when there was multiple sources that could send simultaneous tuples, only one tuple was considered. On the other hand, in a tuple-based system, tuples were never grouped by timestamp. However, in a time-based window, multiple evaluations were performed for the same timestamp. In order to support both system, the notion of batch [15], has been introduced. A stream is neither a series of tuples, nor a series of timestamps, but a series of batches.

Definition 5 (Batch). *A batch is a tuple-set which contains simultaneous tuples only. Given a timestamp, batches form a partition of the set of all tuples having this timestamp. A tuple is part of one single batch, but two simultaneous tuples may belong to two different batches.*

Batch identifiers *are elements of the ordered set $\mathcal{T} \times \mathbb{N}$.*

3.2 Streams & Relations

As shown in [2, 10] relations and streams are two different concepts that have to be defined separately.

Definition 6 (Temporal Relation). *A temporal relation R is a step function that maps a batch identifier $(t, i) \in \mathcal{T} \times \mathbb{N}$ to a tuple-set $R(t, i)$.*

In the following, R designates a *temporal relation*. For simplicity, we will use the term *relation* for *temporal relation* (def. 6). The tuple-set $R(t, i)$ is called *instantaneous relation at batch* (t, i).

Definition 7 (Stream content). *A stream content S is a possibly infinite tuple-set with a schema containing one more special attribute t for a timestamp.*

The physical identifier rules the positional order in streams, and this plays central role in data ordering. Note that the notion of stream is incomplete for now as the stream needs also to be partitioned into batches (see 8,9).

An element of S is a n-tuple s including a value $s(t) \in \mathcal{T}$.

As a stream is partitioned into batches, the full definition of a stream needs to consider the function that identifies the batch given a tuple.

Definition 8 (Batch indicator). *A batch indicator is a function that gives the batch identifier of a given tuple taken from a stream content S. $\mathcal{B}_S : S \mapsto \mathcal{T} \times \mathbb{N}$*

Now we can define a stream as a composition of a content and this indicator.

Definition 9 (Stream). *A stream is a couple $(S, \mathcal{B}_S)$ composed of its content and a batch indicator on those tuples.*

In the following, the notation of the stream content S may be used to denote the proper stream $(S, \mathcal{B}_S)$. The timestamp t_0 will now designates the start timestamp for query evaluation or for temporal relations.

3.3 Batches and positions on streams

We will now assume that, the total order induced by the batches is coherent with the positional order. That is to say: if a tuple s precedes s' in the positional order then $\mathcal{B}_S(s) \leq \mathcal{B}_S(s')$. However, the positional order is strict, e.g. there are no tuples that have the same position on a stream as opposed to the temporal order where two tuples may have the same timestamp. We claim that this property must be verified to have a coherent algebra.

Hypothesis 1 (Order Coherence). *The temporal order and the positional order are coherent: Let S be a stream, then*

$$\forall s \in S, s' \in S, \quad s(\varphi) < s'(\varphi) \Rightarrow \mathcal{B}_S(s) \leq \mathcal{B}_S(s')$$

In SQuAl, the algebra of Aurora, this assumption is not made. To insure the consistency of the model, they require an operator *Order*. In practice, the implementation has to fulfil this property. Interesting techniques [16] have been made to ensure the consistency of the positional and timestamp orders without breaking the workflow. As a matter of fact, Oracle [24], *advises* for performance reasons, to fulfil this property in order to avoid complex treatments that would be otherwise required to obtain consistent results.

Therefore, data sources must ensure that for each new tuple, the physical identifier must increase, and the timestamp must be at least the same as the last one.

4. RELATIONAL OPERATORS

As introduced previously, our algebra adopts the two-fold approach of data stream management based on streams and relations. More precisely, this work uses temporal relations (see definition 6) which are fairly different from the classic relations in Codd's algebra [7]. This section discusses how Codd's operators are inherited and proposes adapted definitions of the operators to work with temporal relations and especially positional ordering. Such operators are necessary to provide clear semantics when handling streams and relations together.

The following sections discuss cartesian product, set oriented operators and joins. For space reasons, the other operators (unary operators, set difference) will not be detailed as they do not raise ordering issues and their definitions are directly inherited from Codd's algebra.

4.1 Cartesian product

Let us consider now the cartesian product between two temporal relations producing a temporal relation. The main issue in the definition of this operator is the order among tuples and, more precisely, the physical identifier of the resulting tuples.

Recall that the physical identifier allows us to differentiate between tuples having the same values and to order tuples of a tuple-set. Multiple semantics can be used to create the physical identifier of the tuples produced by the cartesian product. To avoid implicit heterogeneity that may impact the final result of a query, we propose to make explicit the creation of the physical identifier in cartesian products. A function $\Phi_{t,i}^{R_1 \times R_2}$ is introduced for this purpose. The cartesian product for temporal relations is defined as follows.

Definition 10 (Cartesian product). *Let R_1 and R_2, be two temporal relations such that $Attr(R_1) \cap Attr(R_2) = \{\varphi\}$,* *let (t,i) be a batch identifier,*

let $\Phi_{t,i}^{R_1 \times R_2} : \mathbb{N}\times\mathbb{N} \rightarrow \mathbb{N}$ be an injective function computing the physical identifier of the new tuple given the two physical identifiers of the tuples from R_1 and R_2.

The temporal relational cartesian product of R_1 by R_2 at batch (t,i) is defined as: $(R_1 \times R_2)(t,i) =$

$$\bigcup_{\substack{r \in R_1(t,i) \\ s \in R_2(t,i)}} \left\{ \begin{array}{c} r[Attr(R_1)\backslash\varphi] \cup s[Attr(R_2)\backslash\varphi] \cup \\ (\varphi, \Phi_{t,i}^{R_1 \times R_2}(r(\varphi), s(\varphi)) \end{array} \right\}$$

where $r[a_1, ..., a_n]$ denotes the restriction of tuple r to attributes $a_1, ..., a_n$.

The major difference with respect to the classic cartesian product is the function $\Phi_{t,i}^{R_1 \times R_2}$. This function rules the order (induced by the physical identifier) of tuples in the instantaneous relation $(R_1 \times R_2)(t,i)$ without affecting the composition of the tuples. The choice of function $\Phi_{t,i}^{R_1 \times R_2}$ is nevertheless important because the order's semantic is meaningful for some operators. This function must be injective in order to validate the unique constraint over the physical identifiers on the final tuple-set.

The function used by default in our algebra is:

$$\Phi_{t,i}^{R_1 \times R_2}(\varphi_1, \varphi_2) = \varphi_1 * [\max_{r \in R_2(t,i)}(r(\varphi)) + 1] + \varphi_2$$

Although the value of the physical identifier is not relevant by itself, the induced order is important. The preceding function ensures the following property

$$\Phi_{t,i}^{R_1 \times R_2}(a,b) < \Phi_{t,i}^{R_1 \times R_2}(c,d) \Leftrightarrow a < c \vee (a = c \wedge b < d)$$

that describes the strict lexicographic order (which is total) by giving priority to the left side.

There is no direct criteria to argue that the choice made for $\Phi_{t,i}^{R_1 \times R_2}$ is a good one or not. Our choice has been guided by multiple aspects. Firstly, the function provides a total order commonly established over $\mathbb{N}^2$. Secondly, this order reflects the behaviour of the usual nested loop algorithm: iterate on R_1 and for each tuple iterate on R_2, which is easily understandable.

The characteristics of $\Phi_{t,i}^{R_1 \times R_2}$ may impact the properties of the cartesian product. We highlight the following important fact.

Property 1 (Asymmetrical product). *The cartesian product is not symmetric in the general case.*

The example presented in section 2.1 illustrate this property. The cartesian product is performed on R and RS[1] which is the correct window over S with attributes renaming (in order to make the cartesian product valid). The difference between $R \times RS$ and $RS \times R$ can be seen on table 1.

As already stated, in the first case, the IDs are grouped together and in the second case, the measure and time are grouped together (and therefore ordered by time).

4.2 Relational union

The definition of the union for temporal relations is a little more complicated because the resulting tuples require a meaningful physical identifier. As seen for the cartesian product, a problem of semantics arises in the resulting order.

[1]$RS = \rho_{SEC2/SEC}\sigma_{temp>10}S[\text{RANGE } 60s \text{ SLIDE } 60s]$

	$(R \times RS)(t,i)$					$(RS \times R)(t,i)$				
φ	ID	S	S2	M	T	ID	S	S2	M	T
0	1	2	12	12	21	1	2	12	12	21
1	1	2	2	11	32	2	23	12	12	21
2	1	2	2	14	48	3	23	12	12	21
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
14	4	12	2	14	48	3	23	12	13	54
15	4	12	12	13	54	4	12	12	13	54

Table 1: Illustration of product asymmetry

Definition 11 (Set union). *Let R_1 and R_2 be two temporal relations with the same schema.*

Let $\Phi_{t,i}^{R_1 \cup R_2} : \{0,1\} \times \mathbb{N} \rightarrow \mathbb{N}$ be a function that will ensure a coherent order of φ at each instant (t,i)

The set union is defined as: $R_1 \cup R_2 : t, i \mapsto$

$$\begin{array}{l} \bigcup_{r \in R_1(t,i)} \left\{ r[Attr(R_1)\backslash\varphi] \cup (\varphi, \Phi_{t,i}^{R_1 \cup R_2}(0, r(\varphi)) \right\} \\ \bigcup_{s \in R_2(t,i)} \left\{ s[Attr(R_2)\backslash\varphi] \cup (\varphi, \Phi_{t,i}^{R_1 \cup R_2}(1, s(\varphi)) \right\} \end{array}$$

Several semantics can be considered for $\Phi_{t,i}^{R_1 \cup R_2}$. Note that using $\Phi_{t,i}^{R_1 \cup R_2}(i, \varphi) = \varphi$ would lead to a symmetric union, but without unicity of φ values in the result. This may happen for tuples of R_1 and R_2 having the same value for φ but no exact match for all the other attributes. Such a function can not be used because a total order inferred by φ is required. However, if we can ensure that both relations do not share identifiers, then, this order may be used.

For a general case, we will consider the following function

$$\Phi_{t,i}^{R_1 \cup R_2}(j, \varphi) = j * [\max_{r \in R_1(t,i)}(r(\varphi)) + 1] + \varphi = \Phi_{t,i}^{R_2 \times R_1}(j, \varphi)$$

Here, as for the cartesian product, the value of φ itself is not relevant but the induced order is important. The chosen function gives priority to the left hand side relation (their tuples are placed first). The symmetric property does not hold for the union.

Such ordering problems have not yet been clarified by the literature and are important because they impact the result of queries involving position in streams.

4.3 Joins

Given the preceding definitions, the formalization of joins is now straightforward. The correspondence between joins and cartesian products for temporal relations is the same as in relational algebra. Therefore, the ordering directly depends on the cartesian product.

We have now explored the core relation-to-relation operators. We have seen that such formalization work allowed us to reveal existing semantic discrepancies in data management which were hardly perceivable but which were nevertheless significant. With our model, we can target semantic problems and be aware of them while using the operators.

5. STREAM OPERATORS

This section goes thought main operators to manipulate streams and shows their ordering properties. Section 5.1 presents the *streamers* which create streams from relations. Windows operators, allowing the creation of relations from streams, are quickly addressed in Section 5.2.

5.1 Streamers

Streamers create a stream from a relation. Streams can be generated by monitoring changes in relations or periodically. Streamers have already been defined in other algebra like in STREAM [2]. For instance, $ISTREAM$ sends, for each relation update, the newly inserted tuples.

Streamers are in charge of *stamping* tuples of the streams. This stamping is both positional and temporal and provides coherent orders. We define the stamping function as the function that adds a timestamp and a position to a tuple of a relation.

Definition 12 (Streamer stamping). *Let R be a relation, let (t, i) be a batch identifier, let $\Phi^S_{t,i} : \mathbb{I} \to \mathbb{I}$ be a strictly increasing function that ensures:*

$$\forall (t', i') \leq (t, i), \forall w \in \mathbb{I}, \Phi^S_{t',i'}(w) \leq \Phi^S_{t,i}(w).$$

The function that stamps tuples from $R(t, i)$ is defined by: $\forall s \in R(t, i), \forall t_s \in T,\ \Psi_{t,i}(s, t_s) =$

$$\{('t', t_s), (\varphi, \Phi^S_{t,i}(\varphi))\} \bigcup_{a \in Attr(R) \setminus \{\varphi, 't'\}} \{(a, s(a))\}$$

Then, if we want to send the tuple $s \in R(t, i)$ to the stream at batch (t_s, i_s), we must send the tuple $\Psi_{t,i}(s, t_s)$.

This definition ensures that the stamped tuples that will form a stream verifying the following properties:

- each tuple has a timestamp t_s, which is the moment when it has been stamped,
- each tuple has a physical identifier (position) greater than the identifiers of the previous stamped tuples,
- the positional order of $R(t, i)$ is preserved in the final stream.

If a timestamp was defined in the original tuples, it will be replaced in the stamping process. It is possible to conserve the value of the original timestamp by using an operator $\rho_{t_{original}/t}$ before the streamer that will stamp the tuples.

The actual expression of $\Phi^S_{t,i}$ is not important. Its purpose is to insure a coherent positional order derived from the physical identifier of the instantaneous relation. Stamping using the creation time is crucial. In sensor applications, the question of which timestamp has to be used for tuples containing aggregated measures (5min range for instance) has been discussed many times [5, 9, 12]. Four usual choices have been identified: *min* or *max* of the timestamps of the involved tuples, *user-defined* or the moment of tuple creation[2]. Several strong arguments can be given to this last choice. Firstly, this choice is coherent with the time when the tuple has been computed. Secondly, the order is preserved whereas order violations could be introduced by taking another value. Thirdly, this choice does not make any hypothesis on the schema of R and allows time-stamping of tuples without loosing the original timestamp.

5.2 Windows

Window operators can be used to create relations from streams. These relations can then be used to create a stream of aggregate values (min, max,...) and also to define joins between streams. A window may be temporal (e.g. data from the last 10 minutes), or positional (e.g. the 10th last data of a stream) or cross domain (e.g. the 10th last data every 10 minutes). For space reasons, we will not detail the definitions. Although, we have the following property: considering $[W]$ is a window operator,

$$s \in S[W] \Rightarrow s \in S$$

Therefore, the order inside the window relation is kept exactly identical to the input stream.

It may also be the case for partitioned window. This last operator is usually defined with a set union of multiple windows over substreams. The order of the union set can be defined with $\Phi^{R_1 \cup R_2}_{t,i}(i, \varphi) = \varphi$ to keep the original ordering (which is valid as the substreams forms a partition of the main stream). It has to be mentioned that the function $\Phi^{R_1 \cup R_2}_{t,i}$ defined for union in 4.2 can not be used here as the result will be ordered by partitions (which may be non-deterministic).

We have now detailed the main stream operations. Other operators are a composition of the previously presented operators, therefore, the ordering is clearly defined.

6. IMPLEMENTATION

An open-source prototype[3] of our proposal has been developed in Java/OSGi. This is basically a proof of concept for the operators and allows to experiment query rewriting. All Φ functions defined in this paper have been implemented. Here after, some specificities of the implementation that ensures the orders presented in this paper:

- Tuples have a physical identifier. For tuple-sets iterations, tuples are ordered using the physical identifier.
- The identifier for tuples issued by a cartesian product is $\Phi^{R_1 \times R_2}_{t,i}(\varphi_1, \varphi_2)$. Such identifiers are computed as a class $Couple(\varphi_1, \varphi_2)$ in lexicographic order. The value of such an identifier is obtained by the evaluation of the following expression $\varphi_1 \ll (\textbf{sizeof}\ \varphi_2) \mid \varphi_2$.
- Two implementations are provided for the identifiers of tuples issued by set union as proposed in Sections 4.2 and 5.2. One implementation is based on tuple by tuple iteration with new identifiers whereas the other preserves original identifiers as necessary for the partitioned window.
- For the streamers, tuples in the stream have a new identifier. A counter initialized to 0, is increased one by one for each new "streamed" tuple. The operator iterates over the tuple-set to "stream" and the original ordering ensures the correct behaviour.
- For all the other operators, the output tuples keep their original physical identifier.

7. CONCLUSION AND FUTURE WORK

The relational part of a stream algebra has mainly be considered in the literature as a direct application of the relational algebra. We showed, that using traditional definition of stream querying the positional ordering in a stream may be non-deterministic. Thus we introduced the use of a physical identifiers that forces the definition of the final ordering of tuples produced by each operators. This leads to an explicit and unambiguous definition of stream queries as well as new properties for operator manipulation.

This clarification is important because those core properties are used in several optimization approaches in relational

[2]Greater or equal to the *max*

[3]http://astral.ligforge.imag.fr

DBMS. To the best of our knowledge, such formalization and results have never been presented.

Future research is planned to extend both, the theoretical and the practical work. On the theoretical side the research agenda includes further work on operator's properties, on query equivalence and query rewriting. More globally efforts are required on the combination of several optimization techniques such as aggregation, logical and physical optimization to build efficient and rich data stream query processing.

Acknowledgment: Many thanks to Solveig Albrand and François-Gaël Ottogalli for their help with this paper. This research is supported by the Transcap and ALAP projects.

8. REFERENCES

[1] D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.

[2] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and et al. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin*, 26(March 2003):19–26, 2003.

[3] A. Arasu, S. Babu, and J. Widom. An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations. Technical Report 2002-57, Stanford University, 2002.

[4] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, jun 2006.

[5] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. Technical Report 2002-19, Stanford University, Mar. 2002.

[6] I. Botan, R. Derakhshan, N. Dindar, L. Haas, R. J. Miller, and N. Tatbul. SECRET: a model for analysis of the execution semantics of stream processing systems. *Proc. VLDB Endow.*, 3:232–243, sept 2010.

[7] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), 1970.

[8] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: a stream database for network applications. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 647–651, 2003.

[9] L. Golab and M. T. Özsu. Update-pattern-aware modeling and processing of continuous queries. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 658–669, New York, NY, USA, 2005. ACM.

[10] Y. Gripay, F. Laforest, and J.-M. Petit. A simple (yet powerful) algebra for pervasive environments. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 359–370, New York, NY, USA, 2010. ACM.

[11] L. Gürgen, C. Roncancio, and C. Labbé. Data Management Solutions in Networked Sensing Systems. *Wireless Sensor Network Technologies for the Information Explosion Era*, 278:111–137, 2010.

[12] M. Hammad, W. Aref, M. Franklin, M. Mokbel, and A. Elmagarmid. Efficient execution of sliding window queries over data streams. Technical report, 2003.

[13] S. Inc. SQLstream. `http://www.sqlstream.com`, 2010.

[14] H. V. Jagadish, I. S. Mumick, and A. Silberschatz. View maintenance issues for the chronicle data model (extended abstract). In *Proceedings of the fourteenth ACM symposium on Principles of database systems*, PODS '95, pages 113–124, 1995.

[15] N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, U. Çetintemel, M. Cherniack, R. Tibbetts, and S. Zdonik. Towards a streaming SQL standard. *Proc. VLDB Endow.*, 1:1379–1390, August 2008.

[16] S. Krishnamurthy, M. J. Franklin, J. Davis, D. Farina, P. Golovko, A. Li, and N. Thombre. Continuous analytics over discontinuous streams. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 1081–1092, 2010.

[17] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation service for Ad-Hoc sensor networks. In *Proceedings of the 5th symposium on Operating systems design and implementation*, OSDI '02, pages 131–146, 2002.

[18] K. Patroumpas and T. Sellis. Window specification over data streams. In *Current Trends in Database Technology – EDBT 2006*, volume 4254 of *Lecture Notes in Computer Science*, pages 445–464. 2006.

[19] L. Petit, C. Labbé, and C. L. Roncancio. An algebric window model for data stream management. In *Proceedings of the Ninth ACM International Workshop on Data Engineering for Wireless and Mobile Access*, MobiDE '10, pages 17–24, 2010.

[20] P. Seshadri, M. Livny, and R. Ramakrishnan. SEQ: A Model for Sequence Databases. In *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, pages 232–239, Washington, DC, USA, 1995. IEEE Computer Society.

[21] M. Sullivan. Tribeca: A stream database manager for network traffic analysis. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pages 594–, 1996.

[22] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. In *Proceedings of the 29th international conference on Very large data bases*, VLDB '2003, pages 309–320, 2003.

[23] D. Terry, D. Goldberg, D. Nichols, and B. Oki. Continuous queries over append-only databases. In *Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, SIGMOD '92, pages 321–330, 1992.

[24] A. Witkowski, S. Bellamkonda, H.-G. Li, V. Liang, L. Sheng, W. Smith, S. Subramanian, J. Terry, and T.-F. Yu. Continuous queries in oracle. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 1173–1184, 2007.

[25] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31:9–18, September 2002.