# An Overview of Continuous Querying in (Modern) Data Systems

Angela Bonifati, Riccardo Tommasini

▶ **To cite this version:**

Angela Bonifati, Riccardo Tommasini. An Overview of Continuous Querying in (Modern) Data Systems. 2024, pp.605 - 612. 10.1145/3626246.3654679 . hal-04798334

**HAL Id: hal-04798334**
**https://hal.science/hal-04798334v1**

Submitted on 22 Nov 2024

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# An Overview of Continuous Querying in (Modern) Data Systems

Angela Bonifati
Lyon 1 University, CNRS LIRIS & IUF
France
angela.bonifati@univ-lyon1.fr

Riccardo Tommasini
LIRIS - INSA de Lyon, France
riccardo.tommasini@insa-lyon.fr

## ABSTRACT

Continuous queries, also known as standing or streaming queries, are a class of queries that continuously monitor data sources over time, remaining active until explicitly terminated. This concept was introduced in 1992 by Terry and colleagues to address the need to access data that changes over time. Since then, the domain of continuous queries has seen significant development, research, and application in various data systems, including multiple Database Management Systems (DBMS). Notably, the past five years have marked the rise of Streaming Databases (SDS), and DBMSs that integrate continuous queries with traditional query responses.This growth reflects both industrial and academic interest in the field. Initially, continuous queries were primarily focused on the relational model, with their semantics expressed through extended algebras, calculi, or denotational semantics. However, more recently, there has been a shift towards applying continuous queries in the context of connected data. This includes exploration and abstraction in Graph DBMS, Knowledge Graphs, and Knowledge Evolution, expanding beyond the initial focus on relational models. This tutorial delves into a comprehensive survey of Continuous Queries in past and modern data systems. In addition to the historical perspective, the tutorial will focus on the latest developments in Streaming Databases and the continuous processing of streaming graphs.

## CCS CONCEPTS

• **Information systems** → **Data management systems**; **Database management system engines**; **Online analytical processing engines**; **Query languages for non-relational engines**; **Stream management**.

## KEYWORDS

Continuous Queries; Streaming Databases; Streaming Systems; Streaming Graphs

**ACM Reference Format:**
Angela Bonifati and Riccardo Tommasini. 2024. An Overview of Continuous Querying in (Modern) Data Systems. In *Companion of the 2024 International Conference on Management of Data (SIGMOD-Companion '24), June 9–15, 2024, Santiago, AA, Chile.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3626246.3654679

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only. Request permissions from owner/author(s).
*SIGMOD-Companion '24, June 9–15, 2024, Santiago, AA, Chile*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0422-2/24/06
https://doi.org/10.1145/3626246.3654679

## 1 INTRODUCTION

Continuous queries (CQ), also known as Standing or Streaming Queries, belong to a specific class of queries that perpetually monitors some data sources over time until they are terminated explicitly [46]. Such a class of query was initially introduced by Terry et al. [82] to capture the need for access data that change over time.

Since Terry et al.'s seminal work, continuous queries have been studied [16, 17], prototyped [4, 29–31], and used in various production data systems [14, 20, 26, 53]. In particular, the last five years have witnessed the emergence of Streaming Databases (SDS), e.g., Materialize[1], *RisingWave*[2], as well as traditional Database Management Systems (DBMS) that combine CQs with traditional query answers [24], e.g., Meteor[3] or InvaliDB [90]. Such an increase in offer signals the industrial and academic interest in the topic.

Additionally, initial CQs formulations [68, 82] focused on the relational model, expressing the query semantics in terms of algebras [40], calculi [79], or denotational semantics [13] that extend the existing relational ones [52]. It is not until recently [4], with the rise of graphs as an abstraction for data exploration and management [76], that CQs were studied and applied in the context of connected data, e.g., Graph DBMS [66, 75], Knowledge Graphs [50, 83], and recently to study Knowledge Evolution [72].

This tutorial presents an in-depth literature survey of *Continuous Queries* in existing data systems. In particular, we start the tutorial by presenting the background on Continuous Queries (Section 2), followed by their application and design in preliminary DBMS (Section 3). Then, we move to Stream Systems in the context of Big Data Management (Section 4) and analyse their support for CQs. Finally, we present the recent works on Streaming Databases and continuous processing of streaming graphs (Section 5). We end the paper by positioning this survey in the literature Section 6 and discussing the open challenges in the field ,(Section 7).

## 2 PRELIMINARIES

This section goes through the foundational notions required to discuss Continuous Queries (CQ). Figure 1 presents a high-level view of how a data system for continuous query looks like from the seminal survey on Stream Processing by Cugola et Margara [32]. The figure highlights the paradigm shift introduced by continuous queries vs traditional data-based queries. Indeed, CQs are meant to be issued once and produce results until they are explicitly stopped. Such characteristics make CQs ideal for stream processing applications [38], as in such cases are the data that keep changing while

[1] https://materialize.com
[2] https://www.risingwave-labs.com/
[3] https://www.meteor.com/
[4] it is worth noticing some prominent work on XML was done [56] focusing on streaming algorithms for filtering

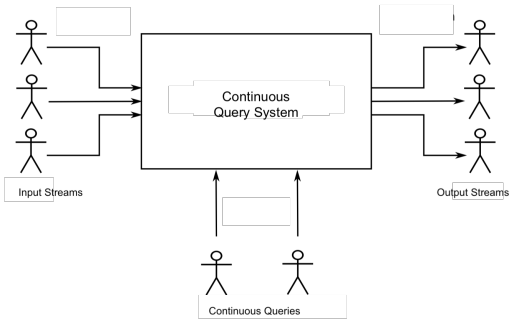


**Figure 1: A Data System for Continuous Querying from [32].**

the analytical task is constant. Thus, the temporal dimension is essential to their characterisation.

To the best of our knowledge, the first appearance dates back to the work of Terry et al. [82]. Terry et al. introduce the notion of *Continuous Semantics* in their seminal work on append-only databases. Such types of databases do not allow deletion and is theoretically meant to retain the whole data history. According to the authors, CQs, unlike standard database queries, are set up once and operate repeatedly on such databases. Such a shift impacts the query semantics, execution, and query optimisation and planning. Most importantly, CQs imply the processing of an infinite input, which yields, in turn, an infinite output. Therefore, *a query's result is the cumulative set of outcomes that would emerge if the query were executed at each moment.* Several proposals followed the one Terry et al. [12, 16, 18, 19, 58, 69] and they all share a few preliminary notions that we indicate below.

*Definition 2.1.* The **Time Domain** $\mathcal{T}$ is an ordered, infinite set of discrete time instants $\tau \in \mathcal{T}$.

In practice, two possible time domains are relevant [9], i.e., *processing time* that represent the time at which the data systems receive data, and *event time*, the time when data are produced in the real world. Intuitively, the former imposes a strictly monotonic time domain, while the latter allows contemporary data.

Assume $\mathcal{T}$ is the ordered time domain, such as the set of natural numbers $\mathbb{N}$. A data stream, (relational) is our second shared abstraction.

*Definition 2.2.* A Data Stream $S$ is a mapping $S : \mathcal{T} \leftarrow 2^R$ that at each instant $\tau \in \mathcal{T}$ returns a finite subset from the set $R$ of tuples with a schema $E$. An additional attribute is designated as the timestamp of tuples and takes increasing values from $\mathcal{T}$.

Such definition allows modelling $S$ as a potentially infinite collection of elements $\langle o, \tau \rangle$, where o represents a data item, like a tuple with a schema $E$, and $\tau \in \mathcal{T}$ is a timestamp. Moreover, it poses the basis for an informal definition of Continuous Query.

*Definition 2.3.* A continuous query $Q^{cont}$ submitted at time instant $\tau$ on data stream S shall provide the results as if the non-continuous query $Q$ was executed for all the possible $\tau \in \mathcal{T}$. Hence, it is evaluated under continuous semantics.

Key to the implementation of continuous query is the window operation. Indeed, although windowless continuous querying is possible, user-defined windowing is currently the most common approach for dealing with data unboundedness. Thus, we introduce the general idea here and will discuss the various notions in detail later. In the literature, several different window operators have been proposed [41, 88]. However, we will limit the scope of this tutorial by focusing on time-based window operators.

*Definition 2.4.* Given a time domain $\mathcal{T}$, windows are defined as functions $W : \mathcal{T} \leftarrow \mathcal{T} \times \mathcal{T}$ where $\mathcal{T}$ is a set of timestamps, e.g., Natural Numbers.

## 3 CONTINUOUS QUERIES AND DBMS

The Database community initially focused on defining query models, highlighting issues of approximate execution [16], and solving algorithmic limitations [36]. Moreover, the research drew the requirements for systems architecture [81] that ultimately pushed the conception *Data Stream Management System* (DSMS) [32].

### 3.1 Continuous Query Languages

The seminal work of Arasu et al.[12] defines the bases of relational continuous querying; the continuous query language (CQL) builds on the relational data model and relational algebra to address append-only ordered data streams. CQL, whose syntax is shown in Listing 1 as an example, assigns abstract semantics to continuous queries via two data types: *Streams*, as per Definition 2.2, and Time-Varying Relations, i.e., a time-aware functional extension of relational data (cf Definition 3.1).

```
1 Select count(P.ID)
2 From Person P, RoomObservation O [Range 15 min]
3 Where P.id = O.id
```

**Listing 1: Example Query using CQL.**

*Definition 3.1.* A time-varying relation R maps each time instant $\tau \in T$ to a finite yet unbounded group of tuples from a set schema with named attributes.

Moreover, CQL introduces three operator classes for writing continuous queries over data streams:

- Stream-to-Relation (S2R) operators, converting a stream into a time-varying relation,
- Relation-to-Relation (R2R) operators, deriving a new time-varying relation from one or more other relations, and
- Relation-to-Stream (R2S) operators generate a stream from a time-varying relation.

S2R operators in CQL are designed to segment a stream S into Windows. A window, denoted W(S), comprises elements selected from a stream by a *window operator*. CQL encompasses time-based,

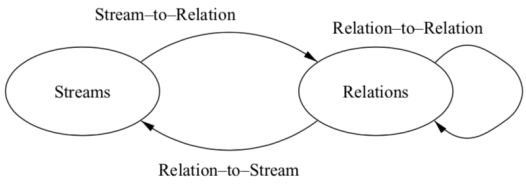


**Figure 2: CQL data abstractions from [12].**

tuple-based, and partitioned window operators. However, the notable aspect of CQL semantics comes from the duality between R2R and R2S. A CQL query does not necessarily have to return to the streaming abstraction. Indeed, the results of a continuous query $Q^{cont}$ at a time $\tau$, which denotes the result of Q once all inputs up to $\tau$ are available, are defined according to two cases:

- When the outermost (topmost) operator in Q is of relation-to-stream type, the result of Q at time $\tau$ is a stream $S$ up to $\tau$, produced by recursively applying the operators comprising Q to streams $S_1, \ldots, S_n$ up to $\tau$ and relations $R_1, \ldots, R_m$ up to $\tau$.
- When the outermost (topmost) operator in Q is of stream-to-relation or relation-to-relation type, the result of Q at time $\tau$ is $R(\tau)$, produced by recursively applying the operators comprising Q to streams $S_1, \ldots, S_n$ up to $\tau$ and relations $R_1, \ldots, R_m$ up to $\tau$.

This approach substantially differs from what was proposed by Babcock and Sellis [16, 69]. Indeed, their definition of continuous semantics has an interpretation closer to sets: the result of $Q^{cont}$ concerns the results $Q_r$ that would be obtained at $\tau_i \in \mathcal{T}$ are the union of the subsets of tuples produced by a series of one-time queries $Q$ on successive stream contents $S(\tau)$, i.e.,

$$\forall \tau_i \in \mathcal{T}, \tau_i \geq \tau_0, Q^{cont} S(\tau_i)) = \bigcup_{\tau_0 \leq \tau \leq \tau_i} Q(S(\tau))$$

The approach adopted by Kramer et al. [58] differs even further, bridging streaming and temporal data. Indeed, they introduce the timeslice operation that generates snapshots from a logical stream. Then, they redefine the notion of *Snapshot Reducibility*, a well-known concept in temporal databases, for query operators over streams. Indeed, Kramer et al. focus on proving that a particular operation, hence query execution, can be evaluated over a finite subset of the data without losing generality. Although operationally similar to windows, timeslice is a global property of the streams, and Snapshot Reducibility can be proved for each operator individually. Revealing interesting properties of streaming algebras and semantics.

*Definition 3.2.* (Snapshot-Reducibility. [58]) A logical stream operator $op_T$ is snapshot reducible to its non-temporal counterpart *op* over multisets if for any point in time $\tau \in \mathcal{T}$ and for all logical input streams $S_1^l, \ldots, S_n^l \in \mathbb{S}^l$, the snapshot at $\tau$ of the results of applying $op_T$ to $S_1^l, \ldots, S_n^l$ is equal to the results of applying *op* to the snapshot $R_1, \ldots, R_n$ of $S_1^l, \ldots, S_n^l$ at time $\tau$.

## 3.2 Queries Optimisation and System Overview

Continuous query optimisation is an essential topic to explore. We will initially focus on theoretical results for continuous queries that led to the definition of optimisation techniques [42]. In particular, it is worth noting the role of rewriting for monotonicity [18], which paves the road to incremental execution.

In their work on characterising continuous queries, Barbara et al. highlighted how the equality of Babcock and Sellis holds only for monotonic continuous queries. Intuitively, a continuous query is non-monotonic when a result that is calculated in the stream at some point may cease to qualify because a new value is added to the database, or more formally: A query is monotonic if for two instances of the data stream $S(\tau_1)$ and $S(\tau_2)$ such that $S(\tau_1) \subseteq S(\tau_2)$ then $Q(S(\tau_1)) \subseteq Q(S(\tau_2))$. Unfortunately, most continuous queries one would like to write are non-monotonic. However, if the stream is append-only, Barbara et al. show that there exists a rewriting that can allow an incremental evaluation.

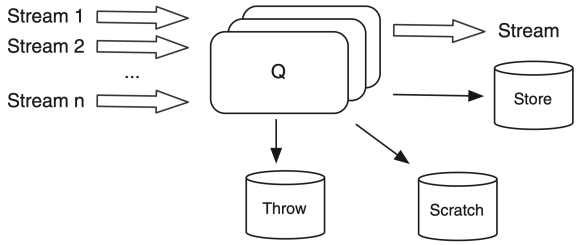


**Figure 3: High Level Architecture of a Data Stream Management System from [32]**

Conversely, the role of planning, especially in relation to adaptive cost models [92], links directly with more non-functional requirements that concern the system level as much as the query language [61]. In these regards, DSMS, as opposed to DBMSs, can handle stream unboundedness and answer the new class of queries meant to last endlessly until explicitly cancelled.

Figure 3 depicts one of the initial architectural proposals for DSMS. The notion of Stream (cd Definition 2.2) denotes both input and main output. *Store* aligns with the CQL's time-varying relation abstraction (cf Definition 3.1. Finally, *Scratch* and *Throw* respectively represent the system's working memory, where intermediate results are saved, and the logical recycle bin is used to throw away unneeded tuples.

Early DSMS prototypes, which are no longer available, like Stanford's STREAM [11], TelegraphCQ [29], NiagaraCQ [30], Auroral/-Borealis [4], and Gigascope [31], were developed to validate the feasibility of continuous query approaches. Indeed, they have similar data models but distinct querying semantics. This era of research introduced critical challenges in systems like sliding window aggregation, fault tolerance, high availability, and load balancing. This initial wave of research, influential from 2004 to 2010, paved the way for commercial stream processing systems like IBM System S, Esper, Oracle CQL/CEP, and TIBCO.

# 4 CONTINUOUS QUERIES AND BIG DATA

The advent of Big Data has witnessed the growing availability of sensor networks, microservices, and cloud-edge infrastructures, which has pushed, in turn, the emergence of data streams as a unifying abstraction [33, 55] that could support the integration of naturally (geo)-distributed and decentralised systems. The adoption of stream shifted the computational paradigm from *data at rest* and post-hoc analyses to *data in motion* and continuous processing [47].

*Streaming Systems* [9] (SPS) emerged to support a large class of applications in which data are generated from multiple sources and are pushed asynchronously to servers responsible for processing them [9]. Notably, aspects like scalability, state management, and out-of-order processing were research priorities for SPS [26].

## 4.1 Declarative Query Interfaces

Streaming Systems exhibit a stack of programming interfaces with different primitives and programming abstractions. Figure 4 depicts such an abstraction, distinguishing between 3 levels, i.e., the architectural level where the Actor exchanged messages to coordinate the computation in real-time; the dataflow level where the streaming computation is expressed in terms of nodes and edges, and the highest declarative layer where Streaming Systems users can specify continuous queries either using SQL-like Dialects or functional DSL (Domain Specific Languages).

It is worth noticing how the Streaming Systems approach continuous query answering from a distributed system perspective. Limiting the computations to simple data transformation that can be expressed using programming primitives and APIs, and substantially avoiding the definition of abstract semantics.

*4.1.1 Dataflow Languages.* The initial query interfaces provided by Streaming Systems were based on the dataflow abstractions, i.e., nodes of the operators to implement procedurally using an Objected-Oriented Turing-complete programming language (Cf Figure 4, mid-level and Figure 5 computational nodes).

Languages designed for dataflow, such as Lustre [43] and Signal [60], were created to simplify the development process of real-time embedded systems. These languages enable developers to articulate a precise and deterministic specification of the system's behaviour as time progresses through discrete logical steps. During each step, the program calculates the value of each data stream based on its inputs and potentially previously computed values. Due to their low level of abstraction, these programs can be thoroughly tested and verified [21]. Hirtzel et al. elaborate on dataflow languages' broader utility for Stream Processing in [48].

In relation to continuous querying, the most prominent of such languages is the *Dataflow Model* [8], derived from FlumeJava [28], which presented a fundamental shift in the approach of stream processing, allowing the user to make a trade-off between correctness, latency, and cost. The *Dataflow Model* introduces *triggers* to provide multiple answers for any given window. Windows and triggers are complementary operations. The former determines *when* data are grouped together for processing using event time; the latter determines *where* the results of groupings are emitted in processing time. In particular, the *Dataflow Model* operates a with two primitives using streams of *(key, value)* pairs: (a) *ParDo* for generic element-wise parallel processing producing zero or more output elements per input. (b) *GroupByKey* for collecting data for a given key before forwarding them for reduction.

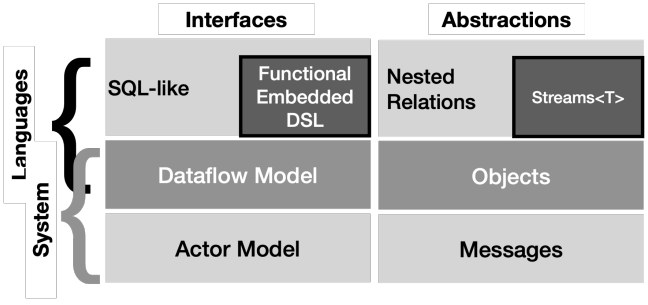


**Figure 4: Streaming System Stack: describing the different abstraction levels in Streaming Systems, inspired by [84].**

Despite its efficiency, dataflow-based approaches for continuous query programming are extremely sophisticated and require low-level programming skills. Thus, new approaches emerged as described in the next section.

*4.1.2 Functional Domain Specific Languages.* A functional DSL for stream processing is a specialized programming language designed to facilitate the development and execution of operations on continuous data streams. Such a DSL incorporates functional programming paradigms [62], emphasizing immutability, stateless functions, and higher-order functions.

Streaming Systems started providing embedded DSLs to facilitate the description of streaming computations [48]. Figure 4 position such DSL at the highest level since they are declarative, but more complex computations still require users to interact with low-level APIs. At the data model level, existing Streaming Systems allow a combination of Object-Oriented Programming and functional operations like map, flatmaps, or aggregates. Lately, such APIs started converging towards a standardised set of operations typical from the DSMS vision, e.g., filter and join. The *Stream and Table Duality Model* [77] explains such data transformations. It includes two notions: the *record* stream and the *changelog stream* (also known as *"table"*). Stream processing operators, divided into *stateless* and *stateful*, describe the transformation of one abstraction to the other. In essence, the model is a blueprint for designing stream-processing applications. Listing 2 shows an example of Flink [27] DSL in java.

```
1 transactions .filter(t -> t.getAmount() > 100)
2  .map(t -> "TID:" + t.getId() + ", Amount:" +  t.get());
```

**Listing 2: Example Flink's Functional.**

*4.1.3 SQL-like Dialects.* One aspect of Streaming Systems is the adoption of SQL-like query interfaces to offer a fully declarative approach to application design [48] instead of the original embedded DSLs (Cf Figure 4, highest level). The initial vision for a Streaming SQL standard dates around 2008: Stonebraker et al. [81] discussed the requirements analysis for an industrial-grade stream processing engine to satisfy. Among other timely proposals, adopting SQL as a language for real-time data management is highly relevant. Similarly, Jain et al. combined and discussed different proposals from the industry that may converge into a so-called standard [54]. Recently, the discussion reopened thanks to a renovated proposal: Bengoli et al. [19] present the opportunities and challenges for a streaming-first SQL standard that builds upon the lessons learned while building Streaming Systems.

In the past, we compared the SQL interfaces of prominent Streaming Systems [85, 86]. Instead, *this tutorial* focuses on the fundamental aspects of the relation between Streaming Systems and Continuous Querying. In particular, Window Operators are probably the most delicate contact between Continuous Querying and Streaming Systems. Having a significant impact on the operational semantics of Streaming Systems [23], window operators have been extensively studied [88], including their richer variants [41, 87]. This digs deeper into the relationship between Windows, and Streaming Systems, always observing existing systems.

## 4.2 Optimisations and Systems Overview

Streaming Systems like MillWheel [7], Storm, Flink [25], and Spark [15], have emerged directly from industrial needs left unsolved by the MapReduce framework. Therefore, as mentioned above, the research agenda behind them addresses open problems of scalability, state management, and out-of-order processing and supports a larger class of data analytics tasks than before. Unlike DSMS, it still lacks an accepted abstract architecture. Figure 5 presents an abstract architecture that highlights some essential aspects related to the deployment and optimisation of SPS. Streaming data are typically accessed by consuming a distributed queue (e.g., Kafka [1] or Pulsar [2]). The output streams are directly pushed to similar systems. Operators, organised in directed acyclic graphs, exchanged intermediate results in parallel, leveraging source-based partitioning. Stateful operations like aggregation and Windows exploit embedded key-value stores (e.g., RocksDB [3]), to persist intermediate results. As described by Figure 4, at the core of these systems, there is some variation of the actor model [45], that used message passing to coordinate parallel and distributed continuous computation [74].

Optimization techniques for Streaming Systems were surveyed by Hirzel et al. [49]. In particular, they focused on static optimisations such as (i) operator reordering [35] , which attempts to anticipate the execution of more selective operators; (ii) redundancy elimination, [39] which tries to remove operators without violating the query correctness; (iii) operator placement [70] , (iv) separation and (v) fusion [51], which respectively attempt to position the operator to reduce the network latency as well as to split or combine operators to leverage parallelisms or distribution better. Notably, optimisation for CQs in Streaming Systems focuses mostly on system aspects and the low-level APIS/DSL. Two notable exceptions are Spark Structured Streaming and Flink (via Apache Calcite [20]), which use volcano-based planning to optimise window-based continuous queries but with customised rules.

# 5 CONTINUOUS QUERIES AND THE MODERN DATA LANDSCAPE

In the contemporary data landscape, continuous queries reshape how we process and interact with real-time data across various sectors. Streaming databases (e.g., Materialize and Risingwave ) emerged as a significant evolution, enabling dynamic query updates as new data streams in, thus facilitating immediate insights. The innovation extends alto to in-database stream processing, which integrates CQs capabilities directly within DBMS systems, allowing for complex analytics and transformations without the need to export data, thereby enhancing efficiency and reducing latency.

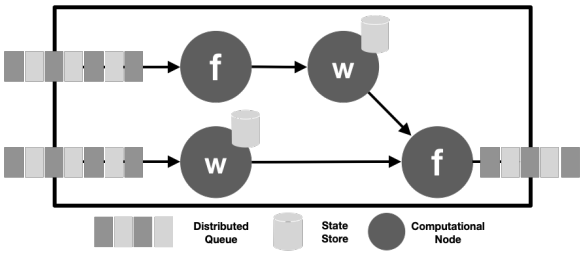


**Figure 5: Abstract Architecture of a Streaming System.**

On the other hand, the central role of graphs in modern data has also impacted the continuous querying area, with specialised systems emerging to handle streaming graphs and graph streams, offering the ability to analyze relationships and patterns dynamically as data evolves, which is vital for applications in network security, social media analysis, and more.

## 5.1 Streaming Databases

Novel solutions like *Materialize*[5], *RisingWave*[6] *HStream*[7], and partially also KSQL DB [53] mark a distinct shift in the landscape of stream processing, setting themselves apart from traditional Big Data streaming systems like Flink or Spark Structured Streaming. These solutions are not merely tools for real-time analysis but comprehensive streaming database management systems with support for continuous queries. Their programming stack is SQL-first, and users can manage data and metadata directly throughout the declarative interface. Last but not least, partial support to constraints is given, with a limited focus on primary and foreign keys.

Moreover, continuous queries also appear within traditional DBMS as additional support for high-velocity workloads. In particular, InvaliDB [90] follows the vision of *Real-Time databases*, offering a push-based query interface on top of a pull-based data store. On the other hand, Winter et al [91] recently proposed an alternative approach for answering queries that require high ingestion throughput. The approach called *continuous view*, relies on a novel maintenance strategy, which splits the work between insertions and queries. The approach was compared to PipelineDB and DBToaster [57], two similar solutions for Incremental View Maintenance on top traditional DBMSs.

## 5.2 Streaming Graphs

Streaming data are also rich in terms of variety [33]: e.g., sensor networks, transaction logs from block-chains, news or social network streams and customer-retail streams, are far more sophisticated and call for more expressive data models and query models and the relational one. NoSQL systems have left an impact on CQs, with Streaming Systems being naively object-oriented and, thus, welcoming towards nested relations and even [5, 73] (Cf Figure 4).

In the Database community, graph streams study dates are extensive [22]. Recently, the investigation has covered the recent property graphs standard [76] with a focus on frameworks and algorithms for complex continuous queries [10, 65, 65, 66, 75]. In particular, given the high expressiveness of the property graph data model, exhibiting multiple edge and node labels as well as sets of properties, continuous property graph queries can tackle highly complex streaming data.

Navigational property graph queries with user-specified constraints in a streaming fashion must adhere to different query semantics, such as simple and arbitrary path semantics [65], while still preserving low latency and high throughput. Subsequently, recursive continuous graph queries as opposed to non-recursive

[5] https://materialize.com
[6] https://www.risingwave-labs.com/
[7] https://hstream.io/

SQL counterparts, require subgraph queries and paths as first-class citizens. Pacaci et al. [66] focus on the above problem by defining a full-fledged continuous query model, based on a streaming path algebra, query formulation and path generation.

As a natural complementary extension of the above works, Rost et al. [75] focus on openCypher, the query language at the core of Neo4j, as a basis for the streaming extension for continuous property graph queries defining its formal semantics precisely. Such an approach has reached the commercial landscape, with solutions like Raphtory [80], Quine [8], and MemGraph [9] emerging to address the problem of continuous queries over graphs. We will discuss complex graph data and query models in streaming and dynamic environments and, focusing on the property graph data model [75] and complex recursive graph continuous queries [65, 66].

In the Semantic Web community, several languages and systems for streaming knowledge graphs have emerged in the last decade to accommodate the need to process heterogeneous data streams on the Web [84]. Dell'Aglio et al. introduced RSP-QL [34], a language that includes new families of operators based on CQL's S2R and R2S, as well as SPARQL 1.1 algebra, allowing continuous SPARQL queries on RDF ( Resource Description Framework) streams. RDF Stream Processing developed under the Stream Reasoning umbrella [33], paving the road for the application and study of Continuous Queries for knowledge graphs and evolution [72]. From RSP-QL emerged additional APIs like RSP4J [83], which tried to generalise the computational approach by borrowing concepts from Streaming Systems and CQL.

## 6 RELATED WORK

This paper focuses on the theoretical and practical impact of continuous queries in the literature, covering 20 years of research.

Directly related to this is a series of tutorials on Stream Processing foundations[10]. The first two tutorials in the cycle focused on the role of declarative languages and SQL, mainly how they translate to the internals of the underlying Streaming Systems. However, this tutorial focuses on continuous queries spanning different types of systems (not just big data) and data models.

"Beyond Analytics: the Evolution of Stream Processing Systems" [26] is a relevant complementary view on the role of stream processing for Big Data. Unlike this tutorial, Carbone et al. focus on analytics rather than continuous querying, drawing a line from the original DSMS vision to modern Streaming Systems. Like us, they discuss the prominence of complex workloads on top of Stream Processing Engines, e.g., graph computations, but they don't focus on high-level querying aspects. Instead, they envision the future of steam processing systems beyond analytical workload.

"*Complex Event Recognition in the Big Data Era*" is also related [37]. The tutorial gives a step-by-step guide on how to realise CER on top of Streaming Systems for BigData. While there is an overlap in the theoretical foundations, being CER a form of continuous querying, Giatrakos et al. discuss how CER state-of-the-art maps on top of Streaming Systems rather than provide a general overview of their querying capabilities.

[8] http://quine.io
[9] https://memgraph.com/
[10] https://streaminglangs.io

## 7 CONCLUSION AND OPEN CHALLENGES

The original DSMS vision, which aimed to manage all aspects of stream data in a completely declarative way [81], has been pushed aside by the advent of the Big Data initiative. In its place, Streaming Systems rose and prospered, satisfying the need for scalable real-time analytics. Lately, Streaming Systems evolved for more sophisticated workloads requiring transactional continuous processing, more complex data models like graphs and documents, and even iterative continuous queries [26]. While the DSMS vision is still unrealised, and the Streaming Systems are evolving towards more sophisticated workloads, a new kind of Streaming Database, with native support to CQs and even more complex data models like RDF or Property graphs, is emerging.

This paper has delineated the extensive scope of related literature, encompassing diverse languages, formalisms, prototypes, production-grade systems, and models, all instrumental in defining the operational semantics of streaming databases. Following such a picture of the state of the art and the state of practice, we propose three main areas that necessitate further exploration within Continuous Queries: 'Linguistic Maturity', 'Query Portability' and 'Data Governance'.

**Linguistic Maturity.** The literature evidences a linguistic maturity adequate for industrial development and standardization. However, a notable gap remains in the absence of a consensus on the fundamental abstractions a database and data systems should offer to support continuous queries. This is particularly critical in a multi-model environment where users may employ various query languages. Establishing a unified set of abstractions for streaming data manipulation is essential to address this complexity.

**Query Portability.** The diversity of existing solutions significantly complicates the porting of query workloads across different systems, which can hinder the adoption of streaming solutions. A primary challenge in this domain is the varied semantics of windowing across languages, leading to distinct operational semantics at the system level [6]. A potential avenue for simplification might be to explore continuous querying independent of windowing concepts, as seen in industrial applications like graph processing in Quine. Nevertheless, a combined theory for continuous semantics is still required to derive compatibility across languages. Several proposals for intermediate representations emerged [40, 44, 59, 63, 64, 78, 79, 89], but none was adopted regardless of the industrial traction.

**Streaming Data Governance.** In Continuous Queries, the research on data provenance in streaming contexts is nascent [67] and is currently limited to why and how-provenance [71] within streaming pipelines framed in functional languages. Furthermore, the issue of ensuring data consistency in the context of continuous queries remains unaddressed. Data cleansing poses a significant challenge, given the stringent latency demands in streaming data. Thus, integrating consistency measures directly into continuous query frameworks might offer a viable pathway forward.

## ACKNOWLEDGMENTS

R. Tommasini is supported by the French Research Agency under grant agreement nr. ANR-22-CE23-0001 Polyflow.
A. Bonifati is supported by the French Research Agency under grant agreement nr. ANR-22-CE92-0025 HyGraph.

# REFERENCES


[1] Apache Kafka — kafka.apache.org. https://kafka.apache.org/. [Accessed 13-04-2024].

[2] Apache Pulsar | Apache Pulsar — pulsar.apache.org. https://pulsar.apache.org/. [Accessed 13-04-2024].

[3] RocksDB | A persistent key-value store — rocksdb.org. https://rocksdb.org/. [Accessed 13-04-2024].

[4] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. B. Zdonik. The design of the borealis stream processing engine. In *Second Biennial Conference on Innovative Data Systems Research, CIDR 2005, Asilomar, CA, USA, January 4-7, 2005, Online Proceedings*, pages 277–289. www.cidrdb.org, 2005.

[5] K. Abakumov. JSQ: distributed querying of JSON stream data. In N. Vassilieva, D. Turdakov, and V. Ivanov, editors, *Proceedings of the Ninth Spring Researchers Colloquium on Databases and Information Systems, Kazan, Russia, May 31, 2013*, volume 1031 of *CEUR Workshop Proceedings*, pages 35–38. CEUR-WS.org, 2013.

[6] L. Affetti, R. Tommasini, A. Margara, G. Cugola, and E. Della Valle. Defining the execution semantics of stream processing engines. *J. Big Data*, 4:12, 2017.

[7] T. Akidau, A. Balikov, K. Bekiroglu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle. Millwheel: Fault-tolerant stream processing at internet scale. *Proc. VLDB Endow.*, 6(11):1033–1044, 2013.

[8] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *PVLDB*, 8(12):1792–1803, 2015.

[9] T. Akidau, S. Chernyak, and R. Lax. *Streaming systems: the what, where, when, and how of large-scale data processing.* " O'Reilly Media, Inc.", 2018.

[10] K. Ammar, S. Sahu, S. Salihoglu, and M. T. Özsu. Optimizing differentially-maintained recursive queries on dynamic graphs. *Proc. VLDB Endow.*, 2022.

[11] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. STREAM: the stanford data stream management system. In M. N. Garofalakis, J. Gehrke, and R. Rastogi, editors, *Data Stream Management - Processing High-Speed Data Streams*, Data-Centric Systems and Applications, pages 317–336. Springer, 2016.

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

[13] A. Arasu and J. Widom. A denotational semantics for continuous queries over streams and relations. *SIGMOD Rec.*, 33(3):6–12, 2004.

[14] M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, A. Ghodsi, I. Stoica, and M. Zaharia. Structured streaming: A declarative api for real-time applications in apache spark. In *Proceedings of the 2018 International Conference on Management of Data*, pages 601–613, 2018.

[15] M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, A. Ghodsi, I. Stoica, and M. Zaharia. Structured streaming: A declarative API for real-time applications in apache spark. In *SIGMOD*, 2018.

[16] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, pages 1–16. ACM, 2002.

[17] S. Babu and J. Widom. Continuous queries over data streams. *SIGMOD Rec.*, 30(3):109–120, 2001.

[18] D. Barbará. The characterization of continuous queries. *Int. J. Cooperative Inf. Syst.*, 8(4):295, 1999.

[19] E. Begoli, T. Akidau, F. Hueske, J. Hyde, K. Knight, and K. L. Knowles. One SQL to rule them all - an efficient and syntactically idiomatic approach to management of streams and tables. In P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1757–1772. ACM, 2019.

[20] E. Begoli, J. Camacho-Rodríguez, J. Hyde, M. J. Mior, and D. Lemire. Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources. In *SIGMOD Conference*, pages 221–230. ACM, 2018.

[21] I. E. Bennour. Formal verification of timed synchronous dataflow graphs using lustre. *J. Log. Algebraic Methods Program.*, 121:100678, 2021.

[22] M. Besta, M. Fischer, V. Kalavri, M. Kapralov, and T. Hoefler. Practice of streaming processing of dynamic graphs: Concepts, models, and systems. *IEEE Trans. Parallel Distributed Syst.*, 34(6):1860–1876, 2023.

[23] I. Botan, R. Derakhshan, N. Dindar, L. M. Haas, R. J. Miller, and N. Tatbul. SECRET: A model for analysis of the execution semantics of stream processing systems. *PVLDB*, 3(1):232–243, 2010.

[24] L. Carafoli, F. Mandreoli, R. Martoglia, and W. Penzo. Streaming tables: Native support to streaming data in dbmss. *IEEE Trans. Syst. Man Cybern. Syst.*, 47(10):2768–2782, 2017.

[25] P. Carbone, S. Ewen, G. Fóra, S. Haridi, S. Richter, and K. Tzoumas. State management in apache flink®: Consistent stateful distributed stream processing. *Proc. VLDB Endow.*, 10(12):1718–1729, 2017.

[26] P. Carbone, M. Fragkoulis, V. Kalavri, and A. Katsifodimos. Beyond analytics: The evolution of stream processing systems. In *SIGMOD*. ACM, 2020.

[27] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[28] C. Chambers, A. Raniwala, F. Perry, S. Adams, R. R. Henry, R. Bradshaw, and N. Weizenbaum. Flumejava: easy, efficient data-parallel pipelines. In B. G. Zorn and A. Aiken, editors, *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2010, Toronto, Ontario, Canada, June 5-10, 2010*, pages 363–375. ACM, 2010.

[29] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, F. Reiss, and M. A. Shah. Telegraphcq: Continuous dataflow processing. In A. Y. Halevy, Z. G. Ives, and A. Doan, editors, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, page 668. ACM, 2003.

[30] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. Niagaracq: A scalable continuous query system for internet databases. In W. Chen, J. F. Naughton, and P. A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pages 379–390. ACM, 2000.

[31] C. D. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. The gigascope stream database. *IEEE Data Eng. Bull.*, 26(1):27–32, 2003.

[32] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, 2012.

[33] E. Della Valle, S. Ceri, F. van Harmelen, and D. Fensel. It's a streaming world! reasoning upon rapidly changing information. *IEEE Intelligent Systems*, 24(6):83–89, 2009.

[34] D. Dell'Aglio, E. Della Valle, J. Calbimonte, and Ó. Corcho. RSP-QL semantics: A unifying query model to explain heterogeneity of RDF stream processing systems. *Int. J. Semantic Web Inf. Syst.*, 10(4):17–44, 2014.

[35] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database systems - the complete book (2. ed.).* Pearson Education, 2009.

[36] M. N. Garofalakis, J. Gehrke, and R. Rastogi, editors. *Data Stream Management - Processing High-Speed Data Streams.* Data-Centric Systems and Applications. Springer, 2016.

[37] N. Giatrakos, E. Alevizos, A. Artikis, A. Deligiannakis, and M. N. Garofalakis. Complex event recognition in the big data era: a survey. *VLDB J.*, 29(1):313–352, 2020.

[38] L. Golab and M. T. Özsu. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, 2003.

[39] M. I. Gordon, W. Thies, M. Karczmarek, J. Lin, A. S. Meli, A. A. Lamb, C. Leger, J. Wong, H. Hoffmann, D. Maze, and S. P. Amarasinghe. A stream compiler for communication-exposed architectures. In K. Gharachorloo and D. A. Wood, editors, *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X), San Jose, California, USA, October 5-9, 2002*, pages 291–303. ACM Press, 2002.

[40] F. Grandi, F. Mandreoli, R. Martoglia, and W. Penzo. Unleashing the power of querying streaming data in a temporal database world: A relational algebra approach. 103:101872, 2022. https://doi.org/10.1016/j.is.2021.101872.

[41] M. Grossniklaus, D. Maier, J. Miller, S. Moorthy, and K. Tufte. Frames: data-driven windows. In A. Gal, M. Weidlich, V. Kalogeraki, and N. Venkasubramanian, editors, *Proceedings of the 10th ACM DEBS '16, Irvine, CA, USA 2016*. ACM, 2016.

[42] S. Guirguis, M. A. Sharaf, P. K. Chrysanthis, and A. Labrinidis. Optimized processing of multiple aggregate continuous queries. In C. Macdonald, I. Ounis, and I. Ruthven, editors, *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, pages 1515–1524. ACM, 2011.

[43] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. *Proc. IEEE*, 79(9):1305–1320, 1991.

[44] S. Herbst, N. Pollner, J. Tenschert, F. Lauterwald, G. Endler, and K. Meyer-Wegener. An algebra for pattern matching, time-aware aggregates and partitions on relational data streams. In F. Eliassen and R. Vitenberg, editors, *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, DEBS '15, Oslo, Norway, June 29 - July 3, 2015*, pages 140–149. ACM, 2015. https://doi.org/10.1145/2675743.2771830.

[45] C. Hewitt. Actor model of computation: scalable robust information systems. *arXiv preprint arXiv:1008.1459*, 2010.

[46] M. Hirzel. Continuous queries. In S. Sakr and A. Y. Zomaya, editors, *Encyclopedia of Big Data Technologies*. Springer, 2019.

[47] M. Hirzel, H. Andrade, B. Gedik, G. Jacques-Silva, R. Khandekar, V. Kumar, M. P. Mendell, H. Nasgaard, S. Schneider, R. Soulé, and K. Wu. IBM streams processing language: Analyzing big data in motion. *IBM J. Res. Dev.*, 57(3/4):7, 2013.

[48] M. Hirzel, G. Baudart, A. Bonifati, E. Della Valle, S. Sakr, and A. Vlachou. Stream processing languages in the big data era. *SIGMOD Record*, 47(2):29–40, 2018.

[49] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm. A catalog of stream processing optimizations. *ACM Comput. Surv.*, 46(4):46:1–46:34, 2013.

[50] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. F. Sequeda, S. Staab, and A. Zimmermann. Knowledge graphs. *ACM Comput. Surv.*, 54(4):71:1–71:37, 2022.

[51] F. Hueske, M. Peters, M. Sax, A. Rheinländer, R. Bergmann, A. Krettek, and K. Tzoumas. Opening the black boxes in data flow optimization. *Proc. VLDB*

*Endow.*, 5(11):1256–1267, 2012.
[52] T. Imielinski and W. L. Jr. The relational model of data and cylindric algebras. *J. Comput. Syst. Sci.*, 28(1):80–102, 1984.
[53] H. Jafarpour and R. Desai. KSQL: streaming SQL engine for apache kafka. In *EDBT*, pages 524–533. OpenProceedings.org, 2019.
[54] N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, U. Çetintemel, M. Cherniack, R. Tibbetts, and S. B. Zdonik. Towards a streaming SQL standard. *Proc. VLDB Endow.*, 1(2):1379–1390, 2008.
[55] M. Kleppmann. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly, 2016.
[56] C. Koch. XML stream processing. In L. Liu and M. T. Özsu, editors, *Encyclopedia of Database Systems, Second Edition*. Springer, 2018.
[57] C. Koch, Y. Ahmad, O. Kennedy, M. Nikolic, A. Nötzli, D. Lupei, and A. Shaikhha. Dbtoaster: higher-order delta processing for dynamic, frequently fresh views. *VLDB J.*, 23(2):253–278, 2014.
[58] J. Krämer and B. Seeger. A temporal foundation for continuous queries over data streams. In J. R. Haritsa and T. M. Vijayaraman, editors, *Advances in Data Management 2005, Proceedings of the Eleventh International Conference on Management of Data, January 6, 7, and 8, 2005, Goa, India*, pages 70–82. Computer Society of India, 2005.
[59] L. Kroll, K. Segeljakt, P. Carbone, C. Schulte, and S. Haridi. Arc: An IR for batch and stream programming. In A. Cheung and K. Nguyen, editors, *Proceedings of the 17th ACM SIGPLAN International Symposium on Database Programming Languages, DBPL 2019, Phoenix, AZ, USA, June 23, 2019*, pages 53–58. ACM, 2019. https://doi.org/10.1145/3315507.3330199.
[60] P. Le Guernic, T. Gautier, M. L. Borgne, and C. L. Maire. Programming real-time applications with SIGNAL. *Proc. IEEE*, 79(9):1321–1336, 1991.
[61] H. Lim and S. Babu. Execution and optimization of continuous windowed aggregation queries. In *Workshops Proceedings of the 30th International Conference on Data Engineering Workshops, ICDE 2014, Chicago, IL, USA, March 31 - April 4, 2014*, pages 303–309. IEEE Computer Society, 2014.
[62] J. MacCormick. Functional programming and streams. *CoRR*, abs/2302.09403, 2023.
[63] M. Meldrum, K. Segeljakt, L. Kroll, P. Carbone, C. Schulte, and S. Haridi. Arcon: Continuous and deep data stream analytics. In *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics, BIRTE 2019, Los Angeles, CA, USA, August 26, 2019*, pages 3:1–3:3. ACM, 2019. https://doi.org/10.1145/3350489.3350492.
[64] C. Miranda, A. Pop, P. Dumont, A. Cohen, and M. Duranton. Erbium: A deterministic, concurrent intermediate representation to map data-flow tasks to scalable, persistent streaming processes. In V. Kathail, R. Tatge, and R. Barua, editors, *Proceedings of the 2010 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, CASES 2010, Scottsdale, AZ, USA, October 24-29, 2010*, pages 11–20. ACM, 2010. https://doi.org/10.1145/1878921.1878924.
[65] A. Pacaci, A. Bonifati, and M. T. Özsu. Regular path query evaluation on streaming graphs. In *SIGMOD*, pages 1415–1430. ACM, 2020.
[66] A. Pacaci, A. Bonifati, and M. T. Özsu. Evaluating complex queries on streaming graphs. In *ICDE*, pages 272–285. IEEE, 2022.
[67] D. Palyvos-Giannas, K. Tzompanaki, M. Papatriantafilou, and V. Gulisano. Erebus: Explaining the outputs of data streaming queries. *Proc. VLDB Endow.*, 16(2):230–242, 2022.
[68] K. Patroumpas and T. K. Sellis. Window specification over data streams. In T. Grust, H. Höpfner, A. Illarramendi, S. Jablonski, M. Mesiti, S. Müller, P. Patranjan, K. Sattler, M. Spiliopoulou, and J. Wijsen, editors, *Current Trends in Database Technology - EDBT 2006, EDBT 2006 Workshops PhD, DataX, IIDB, IIHA, ICSNW, QLQP, PIM, PaRMA, and Reactivity on the Web, Munich, Germany, March 26-31, 2006, Revised Selected Papers*, volume 4254 of *Lecture Notes in Computer Science*, pages 445–464. Springer, 2006.
[69] K. Patroumpas and T. K. Sellis. Maintaining consistent results of continuous queries under diverse window specifications. *Inf. Syst.*, 36(1):42–61, 2011.
[70] P. R. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. I. Seltzer. Network-aware operator placement for stream-processing systems. In L. Liu, A. Reuter, K. Whang, and J. Zhang, editors, *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 49. IEEE Computer Society, 2006.
[71] P. Pintor, R. L. de Carvalho Costa, and J. M. Moreira. Why- and how-provenance in distributed environments. In C. Strauss, A. Cuzzocrea, G. Kotsis, A. M. Tjoa, and I. Khalil, editors, *Database and Expert Systems Applications - 33rd International Conference, DEXA 2022, Vienna, Austria, August 22-24, 2022, Proceedings, Part I*, volume 13426 of *Lecture Notes in Computer Science*, pages 103–115. Springer, 2022.
[72] A. Polleres, R. Pernisch, A. Bonifati, D. Dell'Aglio, D. Dobriy, S. Dumbrava, L. Etcheverry, N. Ferranti, K. Hose, E. Jiménez-Ruiz, M. Lissandrini, A. Scherp, R. Tommasini, and J. Wachs. How does knowledge evolve in open knowledge graphs? *TGDK*, 1(1):11:1–11:59, 2023.
[73] W. Rao, L. Chen, S. Chen, and S. Tarkoma. Evaluating continuous top-k queries over document streams. *World Wide Web*, 17(1):59–83, 2014.
[74] L. Rinaldi, M. Torquati, G. Mencagli, and M. Danelutto. High-throughput stream processing with actors. In E. Castegren, J. D. Koster, and T. C. Schmidt, editors, *AGERE 2020: Proceedings of the 10th ACM SIGPLAN International Workshop on Programming Based on Actors, Agents, and Decentralized Control, Virtual Event, USA, November 17, 2020*, pages 1–10. ACM, 2020.
[75] C. Rost, R. Tommasini, A. Bonifati, E. Della Valle, E. Rahm, K. W. Hare, S. Plantikow, P. Selmer, and H. Voigt. Seraph: Continuous queries on property graph streams. 2024.
[76] S. Sakr, A. Bonifati, H. Voigt, and A. I. et al. The future is big graphs: a community view on graph processing systems. *Commun. ACM*, 2021.
[77] M. J. Sax, G. Wang, M. Weidlich, and J. Freytag. Streams and tables: Two sides of the same coin. In *BIRTE*, pages 1:1–1:10. ACM, 2018.
[78] R. Soulé, M. Hirzel, B. Gedik, and R. Grimm. River: An intermediate language for stream processing. 46(7):891–929, 2016. https://doi.org/10.1002/spe.2338.
[79] R. Soulé, M. Hirzel, R. Grimm, B. Gedik, H. Andrade, V. Kumar, and K.-L. Wu. A universal calculus for stream processing languages. In A. D. Gordon, editor, *Programming Languages and Systems, 19th European Symposium on Programming, ESOP 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6012 of *Lecture Notes in Computer Science*, pages 507–528. Springer, 2010. https://doi.org/10.1007/978-3-642-11957-6_27.
[80] B. A. Steer, N. A. Arnold, C. T. Ba, R. Lambiotte, H. Yousaf, L. Jeub, F. Murariu, S. Kapoor, P. Rico, R. Chan, L. Chan, J. Alford, R. G. Clegg, F. Cuadrado, M. R. Barnes, P. Zhong, J. N. Pougué-Biyong, and A. Alnaimi. Raphtory: The temporal graph engine for rust and python. *CoRR*, abs/2306.16309, 2023.
[81] M. Stonebraker, U. Çetintemel, and S. B. Zdonik. The 8 requirements of real-time stream processing. *SIGMOD Record*, 34(4):42–47, 2005.
[82] D. B. Terry, D. Goldberg, D. A. Nichols, and B. M. Oki. Continuous queries over append-only databases. In M. Stonebraker, editor, *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 2-5, 1992*, pages 321–330. ACM Press, 1992.
[83] R. Tommasini, P. Bonte, F. Ongenae, and E. Della Valle. RSP4J: an API for RDF stream processing. In R. Verborgh, K. Hose, H. Paulheim, P. Champin, M. Maleshkova, Ó. Corcho, P. Ristoski, and M. Alam, editors, *The Semantic Web - 18th International Conference, ESWC 2021, Virtual Event, June 6-10, 2021, Proceedings*, volume 12731 of *Lecture Notes in Computer Science*, pages 565–581. Springer, 2021.
[84] R. Tommasini, P. Bonte, F. Spiga, and E. Della Valle. *Streaming Linked Data: From Vision to Practice*. Springer Nature, 2023.
[85] R. Tommasini, S. Sakr, M. Balduini, and E. Della Valle. An outlook to declarative languages for big streaming data. In *DEBS*, pages 199–202. ACM, 2019.
[86] R. Tommasini, S. Sakr, E. Della Valle, and H. Jafarpour. Declarative languages for big streaming data. In A. Bonifati, Y. Zhou, M. A. V. Salles, A. Böhm, D. Olteanu, G. H. L. Fletcher, A. Khan, and B. Yang, editors, *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*, pages 643–646. OpenProceedings.org, 2020.
[87] J. Traub, P. M. Grulich, A. R. Cuellar, S. Breß, A. Katsifodimos, T. Rabl, and V. Markl. Scotty: General and efficient open-source window aggregation for stream processing systems. *ACM Trans. Database Syst.*, 46(1):1:1–1:46, 2021.
[88] J. Verwiebe, P. M. Grulich, J. Traub, and V. Markl. Survey of window types for aggregation in stream processing systems. *The VLDB Journal*, 2023.
[89] K. Wang, Z. Zuo, J. Thorpe, T. Q. Nguyen, and G. H. Xu. RStream: Marrying relational algebra with streaming for efficient graph mining on A single machine. In A. C. Arpaci-Dusseau and G. Voelker, editors, *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 763–782. USENIX Association, 2018. https://www.usenix.org/conference/osdi18/presentation/wang.
[90] W. Wingerath, F. Gessert, and N. Ritter. Invalidb: Scalable push-based real-time queries on top of pull-based databases (extended). *Proc. VLDB Endow.*, 13(12):3032–3045, 2020.
[91] C. Winter, T. Schmidt, T. Neumann, and A. Kemper. Meet me halfway: Split maintenance of continuous views. *Proc. VLDB Endow.*, 13(11):2620–2633, 2020.
[92] Q. Xie, X. Zhang, Z. Li, and X. Zhou. Optimizing cost of continuous overlapping queries over data streams by filter adaption. *IEEE Trans. Knowl. Data Eng.*, 28(5):1258–1271, 2016.