Tracking Consistency over Data Streams with InkStream

Samuele Langhi
samuele.langhi@univ-lyon1.fr
Lyon 1 University
Lyon, France

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

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

ABSTRACT

Establishing robust frameworks to safeguard data consistency in streaming applications is a strategic imperative. Nevertheless, existing methods cannot deal with the infinite nature of streaming data. On the other hand, Stream Processing Engines, systems that serve as the infrastructure for executing continuous queries efficiently, were never leveraged for data consistency management. Indeed, handling data consistency by enforcing constraints over data streams may compromise the strict performance requirements that streaming applications impose on latency and throughput. In this demonstration, we introduce InkStream, a novel system utilizing provenance-based techniques to track the consistency of streaming data. InkStream enriches each record in the input stream with provenance annotations, encoding its consistency across a set of constraints. These annotations are propagated through query operators, enabling the quantification of the impact of the consistency on the query results. Users can engage with InkStream through interactive visualizations, real-time monitoring of query outputs, and runtime monitoring of consistency metrics. Through InkStream, monitoring data consistency over streams becomes accessible and actionable during runtime, unlike traditional post-hoc approaches.

VLDB Workshop Reference Format:
Samuele Langhi, Angela Bonifati, and Riccardo Tommasini. Tracking Consistency over Data Streams with InkStream. VLDB 2024 Workshop: International Workshop on Quality in Databases (QDB'24).

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/riccardotommasini/inkstreamui.

1 INTRODUCTION

Streaming data, which are characterized by their continuous and rapid generation, demands not only swift processing but also stringent adherence to data management principles [7]. At any point of continuous analysis, we shall ensure that every piece of data is accurate and consistent with reference to a set of constraints.

Data consistency (DC) over streams is foundational to reliable decision-making and operational effectiveness in various high-stakes domains, such as smart-grid management, financial trading, and emergency response systems. Failing to assess DC can lead to flawed insights, operational risks, and strategic missteps. However, handling data consistency in streaming is hard as it involves validating an integrity constraint over the infinite stream while the application and the related continuous queries are executed.

For doing so, traditional approaches require complex stateful computations, e.g., an arbitrary number of joins, that do not apply as is to streaming data. First, the infinite nature of data streams [7] prevents a complete DC validation, and calls for the use of *windows operators* [2] to reduce the query scope to finite stream subsets [1]. Moreover, once the integrity constraints are validated, techniques like repair [6], are applied before the (windowed) query execution permanently altering the input stream, resulting in a loss of the mapping between input and output. This can lead to missed answers [5] or inaccurate analysis.

In this demonstration, we propose InkStream, a system prototype capable of leveraging polynomial-provenance to track the consistency of streaming data across streaming queries. In practice, it involves annotating streaming data with polynomials that describe their consistency and propagating them up to the queries' results.

To derive such polynomial annotations, InkStream combines window operations with a graph-based method that tracks the consistency relation across tuples. Navigating the graph, the polynomials are built. Then annotations are *propagated* through the query operators at runtime following the semantics of provenance semirings algebra [3]. Such a method offers a non-invasive and flexible solution for dealing with DC, as annotations can be exploited to generalize different techniques, e.g., data repair [6].

To exemplify our approach, we consider the real-world consumption monitoring of two electric grids (A and B). Data comes as a relational stream with schema $\langle \texttt{consA}, \texttt{consB}, \texttt{ts} \rangle$, where consA and consB represent the consumption of the two grids at time ts. The monitoring query, written in CQL [1] and illustrated in Listing 1, consumes the Consumption stream and calculates the usage percentages over a sliding window of 5 minutes every 2 minutes.

```
SELECT percent(consA,consB),percent(consB,consA),ts
FROM Consumption [RANGE 5 minutes SLIDE 2 minutes]
WHERE consA >= 0 AND consB >= 0
```

Listing 1: Monitoring the usage percentages by zone over a sliding window of 5 minutes every 2 minutes. Percent is a UDF (x,y) = sum(x)*100/(sum(y)+sum(x)).

The monitoring aims to prevent malfunctions on both grids. To enforce this, Speed Constraints [6] such as SC1 and SC2 are applied on records, to detect inconsistent "spikes" in the consumption [6].

$$\forall r_i, r_j \quad -2 \leq \frac{r_i.\texttt{consA} - r_j.\texttt{consA}}{r_i.\texttt{ts} - r_j.\texttt{ts}} \leq 2 \tag{SC1}$$

$$\forall r_i, r_j \quad -2 \leq \frac{r_i.\texttt{consB} - r_j.\texttt{consB}}{r_i.\texttt{ts} - r_j.\texttt{ts}} \leq 2 \tag{SC2}$$

These spikes should not be repaired, as they may be related to problems in the grids. Moreover, these inconsistent behaviours

Proceedings of the VLDB Endowment. ISSN 2150-8097.

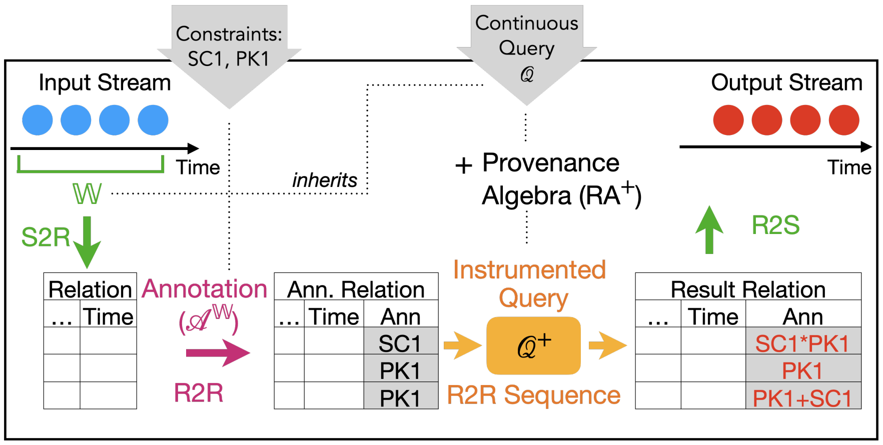


Figure 1: An Overview of our approach.

cannot always be derived from aggregate information like the result percentages in Listing 1. As a result, constraint violations should be encoded in metadata and made available to the end users.

Through InkStream, users can engage in the monitoring data consistency over streams in the following aspects:

- An interactive visualization of consistency annotation of streaming input records wrt the reference window, with the additional monitoring of the underlining, graph-based data structure and its evolution
- The real-time monitoring of query outputs and their consistency, encoded in the form of provenance annotations
- The runtime monitoring of various consistency metrics calculated over such provenance annotations

2 TRACKING STREAMING CONSISTENCY

This section provides an overview of our approach for tracking data consistency over streams, i.e., infinite sequences of time-ordered records. Figure 1 shows a high-level representation. Records are first *annotated* according to their violation of a set of constraints, provided by the user. Such annotations are built over provenance polynomial semirings [3]. Then, they are *propagated* by integrating the semiring positive algebra with CQL operators [1].

Table 1: Stream S, where each row is a record. In red the sudden changes (SC1/SC2), annotated with violations of SC1, SC2.

ts	consA	consB	*id*	$\mathcal{A}^{\mathbb{W}}$
0	8	2	r_0	1
1	8	2	r_1	1
2	8	2	r_2	1
3	8	2	r_3	1
4	5	5	r_4	$SC1_3 \times SC2_3$
5	3	7	r_5	$SC1_3 \times SC2_3$
6	1	10	r_6	$SC1_3 \times SC2_3^2 \times SC2_4 \times SC2_5$
7	0	10	r_7	1
...	...	...	...	

2.1 Consistency Annotation

The annotation process is based on the consistency of the records involved, validated against a set of constraints. The validation results are encoded into a metadata annotation based on Provenance Semirings [3], specifically polynomials.

Intuitively, windows are necessary when the constraint evaluation involves multiple records, e.g., for binary constraints. Thus, the intuition behind our framework is that the constraint validation and the related annotation procedure operates within a window, denoted as $\mathbb{W}$, inherited from the user query, as in Figure 1.

From a formal perspective, validation and annotation are defined through the same function, called $\mathbb{W}$-Relation ($\mathcal{A}^{\mathbb{W}}$). For each record r, its annotation $\mathcal{A}^{\mathbb{W}}(r)$ returns a finite polynomial where each variable V represents a violation generated by r wrt a given constraint validated within a window $\mathbb{W}$.

In the context of the example from Section 1, we consider the constraint labels SC1 and SC2 as variables, although our approach can be generalized for an arbitrary number of constraints. Since speed constraints SC1 and SC2 are binary, we also use subscripts to reference the record with which the violation is caused, e.g.,$SC1_3$ represents a violation of SC1 wrt r_3 (cf. Table 1). Moreover, polynomials can be exploited to represent the violation "degree", by adding the distance ϵ as the exponent of violation labels. Such distance is the difference between the actual and repaired value of a record [6].

Example 1. *Continuing the example from Section 1, Table 1 shows how a stream can be annotated wrt the violation of constraints SC1 and SC2. Record r_4 violates both constraints if combined with r_3, thus*

$$\mathcal{A}^{\mathbb{W}}(r_4) = SC1_3 \times SC2_3$$

The same violations are generated by r_5 and r_6, which violates SC2 also with r_4 and r_6. Moreover, r_6 and r_3 present a high degree of violation, which can be calculated through the following formula

$$\underbrace{r_3.consB + 2(r_6.ts - r_3.ts)}_{repaired\ \texttt{consB}\ of\ r_6} - \underbrace{r_6.consB}_{actual\ r_6} = 2 \longrightarrow SC2_3^2$$

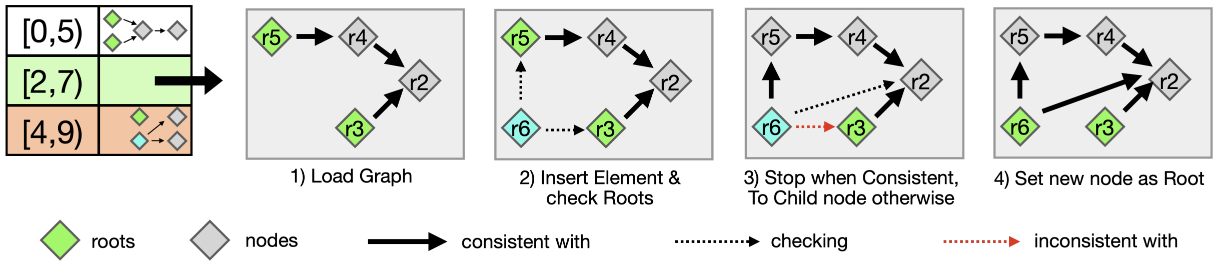


Figure 2: Exploration of the consistency graph for constraint SC1 built over the input stream from Figure 5 on record r6 arrival.

2.2 Consistency Graph

In practice, the annotation phase can be naively implemented through an arbitrary number of join operations. However, this can highly impact performances due to the complexity of the operation. For this reason, we designed an approach based on *consistency (provenance) graphs* to annotate constraint violations efficiently. In the graph, each node corresponds to a record r_k, and each directed edge (r_i, r_k) represents the consistency between the two records wrt a given constraint. Since constraints evaluation follow temporal order, an edge (r_i, r_k) is present if $r_i.ts > r_k.ts$. Consequently, the graph is acyclic and topologically ordered wrt time.

Performance-wise, the advantage of this graph-based approach comes from exploiting a transitive (inc)consistency relation between records when using certain constraints. For instance, for SC1 and SC2, when a record r_x is consistent with record r_y, and r_y is consistent with record r_z, then r_x and r_z are consistent. As a result, a path within the graph indicates a set of consistent records.

The annotation is executed by navigating the graph in a BFS-like approach, represented in Figure 2. When the new element (in blue) arrives, its consistency is evaluated starting from graph roots (in green). If a node is inconsistent with the new record, the latter is annotated with a violation, and the check progresses on child nodes. Otherwise, if transitive consistency is satisfied, the traversal ends.

Compared to a naive join-based annotation over n records, that has a time complexity of $O(n)$, the graph traversal is $O(i)$, with i being the number of constraint violations generated by the annotated record, which is inherently less than n.

Notably, the consistency graph is time-varying in the streaming context and the annotation process is scoped over a given window, as specified in Section 2.1. Thus, a graph is maintained for each window. In particular, when a new element arrives, expired windows are cleared. Then, the earliest window that contains the arrived element and the related graph is selected for annotation (cf. Figure 2). To guarantee completeness of analysis for future records, the new record is inserted in all graphs related to overlapping windows.

Example 2. *If we consider the example from Section 1 and the related query from Listing 1, we annotate wrt to a time-based window sliding window* $\mathbb{W}$ *of size* 5 *and slide* 2. *In this context, the annotation window for* r_6 *is the earliest window that containing it, i.e.,* $[2, 7)$. *For this window, the construction of the consistency graph wrt constraint SC1 starts with* r_2*, which is the first node inserted. Then,* r_3 *is consistent with* r_2*, so it is connected to the related node. On the other hand,* r_4 *is instead inconsistent with* r_3*, but remains consistent with* r_2*. The same approach is adopted at the arrival of* r_5 *and* r_6*. Figure 2 shows the annotation process for the latter: the new node is inserted and checks the consistency wrt* r_5 *and* r_3 *first; while the former is consistent, the latter is not, resulting in the annotation of* r_6 *with* $SC1_3$ *and the further check with* r_2*, which is ultimately connected to the new node.*

2.3 Provenance-based Propagation

Our framework combines CQL continuous semantics with the provenance semirings and the positive relational algebra (RA$^+$) [3]. For instance, join results are annotated with the product $(\cdot)$ of the contributing tuples' polynomials when using the related semiring.

On the one hand, RA$^+$ associates each relational algebra operator with an operator over a generic semiring. In the following, we provide the operator definitions.

SELECTION. Given a selection predicate $P : \Omega \longrightarrow \{0, 1\}$, we define Selection as the application of the $\mathbb{W}$-relations multiplied by the selection predicate. If the prediction is

$$(\sigma^+_P \mathcal{A}^{\mathbb{W}})(r) = \mathcal{A}^{\mathbb{W}}(r) \cdot P(d)$$

PROJECTION. Since we operate under set semantics, when a projection is performed over a record, it may collapse with other projected records. The resulting annotation is the sum of all annotations of the collapsed records. More formally, let A and U be sets of attributes such that $A \subset U$. Given a $\mathbb{W}$-relation $\pi_A \mathcal{A}^{\mathbb{W}}$ defined over a stream schema A, and $\mathcal{A}^{\mathbb{W}}$ defined over record tuples with schema U is

$$(\pi^+_A \mathcal{A}^{\mathbb{W}})(r) = \sum_{r=\pi_A r' \wedge \mathcal{A}^{\mathbb{W}}(r') \neq 0} \mathcal{A}^{\mathbb{W}}(r')$$

UNION. Similar to the projection, the union may cause duplicates to collapse into one, and the annotation is the sum of the collapsed records' annotations. Given $\mathcal{A}_1^{\mathbb{W}}, \mathcal{A}_2^{\mathbb{W}}$:

$$(\mathcal{A}_1^{\mathbb{W}} \cup^+ \mathcal{A}_2^{\mathbb{W}})(r) = \mathcal{A}_1^{\mathbb{W}}(r) + \mathcal{A}_2^{\mathbb{W}}(r)$$

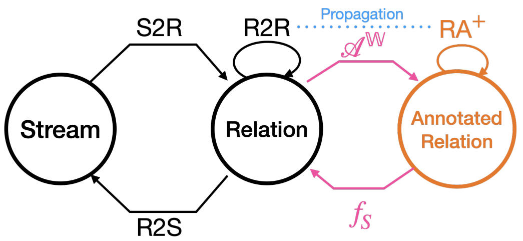


Figure 3: The enhanced CQL continuous semantics for Consistency Annotation and Propagation.

Join. When performing a join, the final annotation is the product of the two input annotations. Given two annotation policies $\mathcal{A}_1^{\mathbb{W}}, \mathcal{A}_2^{\mathbb{W}}$, and records $r_1 = (\omega_1, \tau_1), r_2 = (\omega_2, \tau_2), r = r_1 \bowtie r_2$,

$$(\mathcal{A}_1^{\mathbb{W}_1} \bowtie^+ \mathcal{A}_2^{\mathbb{W}_2})(r) = \mathcal{A}_{res}^{\mathbb{W}_3}(r) = \mathcal{A}_1^{\mathbb{W}_1}(r_1) \cdot \mathcal{A}_2^{\mathbb{W}_2}(r_2)$$

$$\text{with } \mathbb{W}_3 = [min(\mathbb{W}_1.o, \mathbb{W}_2.o), max(\mathbb{W}_1.c, \mathbb{W}_2.c)]$$

Stream processing often concerns aggregate queries (e.g., `percent` from Listing 1). We do not formally define aggregate operators due to space limitations. However, since aggregates can be derived from a projection over the aggregated value, we consider its annotation as the sum of the annotations of the aggregated records.

On the other hand, CQL semantics are based on three operators types: stream-to-relation (S2R) which convert a stream into a relation, relation-to-relation (R2R) which perform data manipulation through classic SQL relational operators, and relation-to-stream (R2S) which unrolls the continuously updated relation into a stream.

Figure 3 shows how the two frameworks are combined. The $\mathbb{W}$-Relation $\mathcal{A}^{\mathbb{W}}$ operates on the result of an S2R operator, returning a time-varying, annotated relation [1]. In this context, CQL's and RA$^+$ operators are seamlessly integrated with each other, since they all stem from the relational model [1, 3]. For each R2R operator manipulating the data, the respective RA$^+$ operator manipulates the annotations. Ultimately, annotations can be used to calculate consistency parameters through a function f_S, returning a relation, which can be unrolled into a stream through an R2S operator.

Example 3. *According to the query from Listing 1, the records processed are first filtered through the* `WHERE` *condition and then aggregated through the* `percent` *function. According to the Positive Relational Algebra (cf. Section 2.3), the* `WHERE` *selection (σ_P) would simply propagate the annotation of those records that satisfy the related condition, turning them to* 0 *in the opposite case, which would imply the removal of the record. On the other hand, the* `percent` *aggregation instead, would sum up all the annotations of the record involved in the aggregation, according to a reference window. Table 2 shows the final results. For instance, the aggregation performed over records in the window $[2, 7)$, i.e., r_2, r_3, r_4, r_5 and r_6, would sum all the related annotations, ending up with the following polynomial, where no violations of r_2 and r_3 can be detected (they are totally consistent)*

$$\underbrace{2(SC1_3 \times SC2_3)}_{r_4 \,\&\, r_5} + \underbrace{SC1_4 \times SC1_5 \times SC2_4 \times SC2_5^2}_{r_6}$$

Resulting annotations can then be used to manage constraint violations in techniques like repair or top-k analysis [4]. Multiple approaches might be adopted. An intuitive approach is to *quantify the degree of consistency*, for instance, by utilizing the polynomial's degree, as demonstrated in [4]. Polynomials with higher degrees might indicate more severe constraint violations due to multiple constraint interactions or annotations. Another similar parameters might be the number of variables within the polynomials, standing as the number of violations from that specific record. A more expressive yet less intuitive approach is polynomial *simplification*, which may reveal underlying consistency patterns.

Example 4. *In Table 2, we reported in the last three columns the three proposed usages of annotations in the context of the example from Section 1. More specifically, the percentages calculated over windows $[0, 5)$ and $[2, 7)$ are both generated from inconsistent records. By deepening the analysis through the calculation of the polynomial degree (Degree) and the number of variables in the polynomial (Variables), we derive that percentages from window $[2, 7)$ carry much more inconsistency than the ones calculated from $[0, 5)$. Additionally, the polynomial annotations enables to detect the co-violation of SC1 and SC2, which signifies healthy system behavior since grid B correctly backs up the malfunction on grid A. The percentages calculated over $[0, 5)$ become in this case entirely consistent. Indeed, the related annotation can be simplified (Simpl) by removing simultaneous violation labels of SC1 and SC2. Thus, $SC1_3 \times SC2_3$ becomes* 1.

3 USE CASES

Stock Analysis. Aside from the electric grid consumption scenario, andother potential use case for our approach is financial analytics. In this context, constraints like SC1 or SC2 can be used to monitor the volatility of a financial asset, i.e., the price fluctuation within a unit of time. Volatility is generally regarded as major risk factor, and needs to be taken into account in multiple task, e.g., Portfolio Diversification. More specifically, traders aims at diversifying their portfolios by investing in multiple *uncorrelated* assets, which are identified through a given parameter, e.g., Pearson's coefficient.

In practice, correlation can be monitored with a continuous query like the one in Listing 2, while annotations can be generated wrt the violation of a Speed Constraint defined over the asset `price`. The query performs a self-join between records of diverse assets, and calculate the `pearson` coefficient over the last 5 minutes for each combination (`GROUP BY`). The coefficients are enhanced through the annotations, highlighting whether the correlated assets are volatile.

```
SELECT S1.name, S2.name, pearson(S1.price,S2.price)
FROM Stock [RANGE 5 Min SLIDE 1 Min] AS S1,
     Stock [RANGE 5 Min SLIDE 1 Min] AS S2
WHERE S1.ts = S2.ts AND S1.name <> S2.name
GROUP BY S1.name, S2.name;
```

Listing 2: A CQL query checking assets correlation.

Movie Review Analysis. With the advent of social media, stream processing solutions are commonly used to address low-latency requirements in sentiment analysis, which is another potential use case for our approach. Specifically, review platforms aim to maintain the integrity of their review systems by filtering out manipulated reviews, identifies through an integrity constraint.

For example, constraints like SC1 or Primary Keys can be used to identify Review Bombing anomalies in user reviews. Review bombing is generally regarded as a major issue in online reviews and it generally consists in sudden influxes of extreme ratings for a particular movie posted from the same specific user. In practice, such phenomena can be monitored by checking the violation of a Primary Key Constraint defined on the `user` and the movie `title`, which imply multiple reviews from the same user on a movie.

Such policy can be used for calculating average stars of the movie in a day, performed through the query from Listing 3, where results annotations highlights signs of manipulation in the average scores.

```
SELECT title, avg(stars)
FROM MovieReviews [RANGE 24 HOURS SLIDE 24 HOURS]
GROUP BY title;
```

Listing 3: A CQL query analysing the movie review stream.

Table 2: The results from Listing 1, with related annotations, reference windows, and consistency metrics.

Window ($\mathbb{W}$)	%consA	%consB	ts	$\mathcal{A}^{\mathbb{W}}$	*Simpl*	*Degrees*	*Variables*
[0,5)	74	26	5	$\underbrace{SC1_3 \times SC2_3}_{r_4}$	1	1	2
[2,7)	47	53	7	$\underbrace{2(SC1_3 \times SC2_3)}_{r_4 \& r_5} + \underbrace{SC1_4 \times SC1_5 \times SC2_4 \times SC2_5^2}_{r_6}$	$SC2_5^2$	2	6
[4,9)	...	...	9	...		...	...

GPS Position Analysis. Another promising use case for our approach is detecting anomalies in GPS data for bike-sharing services. Specifically, bike-sharing services need to ensure the precision of their tracking systems by detecting and filtering out or adjust erroneous GPS data. In this context, constraints like SC1 or SC2 can help identify abnormal bike movements, such as sudden and implausible changes in position over short periods.

In practice, GPS data integrity can be ensured with annotations generated from violations of a Speed Constraint defined over the bike's coordinates x and y. The annotated violations are crucial indicators of potential issues and must be accounted in tasks like ensuring accurate billing, performed through a continuous query from Listing 4. The query calculates the distance each client travels between two subsequent positions, which is then used to determine the client's charge. The results are further enhanced with annotations that highlight any detected anomalies in movement.

```
SELECT clientId, distance(x, y) * 1.5
FROM GPSPosition [RANGE 2 TUPLES SLIDE 1 TUPLE] AS P
GROUP BY clientId;
```

Listing 4: Sn SQL analyzing bike-sharing GPS data stream.

4 INKSTREAM SYSTEM ARCHITECTURE

We developed our system on top of RSP4J [8], a stream processing framework that enables fast prototyping and fine-grained control over stream ingestion. Originally implemented for graphs, we extended its data model over records with attribute-value pairs. The system implements a processing model inspired by CQL [1], making RSP4J natively compatible with our approach (cf. Figure 1).

As presented in Section 2, our method for tracking data consistency over streams consists of two phases, i.e., *annotation* and *propagation*. The two phases are mapped to the system architecture components, represented in Figure 4.

Annotation is implemented through an R2R operator that integrates *Provenance4J*, i.e., the module that implements consistency graphs, including their maintenance and annotation algorithms (cf. Section 2.2). The module inherits the window used by the user-defined continuous query, that in RSP4J is implemented by an S2R operator. Consistent graph traversal results are encoded in a polynomial, whose logic is implemented in the Polyomials4J module.

For the **Propagation** phase, we provided a set of R2R, SQL-like operators capable of natively dealing with polynomial annotations. For each operator, i.e., selection, projection, join, and aggregate, we designed a variant that manipulates annotations according to the positive streaming algebra described in Section 2.

We created a **Streaming Dashboard** tailored for InkStream, enabling users to input streaming queries and desired constraints, monitor the progression of consistency graphs, and visualize their results within a controlled environment.

Once a query and the related constraint are submitted and registered within the InkStream engine, they are converted into a set of streaming operators. The resulting data is streamed and promptly exhibited in the user interface, together with the consistency annotation derived from the propagation phase.

5 DEMONSTRATION PIPELINE

The demonstration pipeline aligns with the approach shown in Figure 1, serving as a bridge between theory and practical application. Designed to cater to both novice and expert users, it operates within the framework of a web interface for tracking consistency across data streams, specifically tailored around the Web User Interface (WUI) as illustrated in 5. The WUI is composed by the following components: **1)** the *Control Panel* provides the button for controlling the ingestion of records, including the possibility of processing the next event, defining query and constraints; **2)** in the *Input Stream Tab*, we represent the input stream in a relational format, indicating through the column "Cursor" the elements processed; **3)** the *Output Stream Tab* enables to see the results of the query, but also view their annotations and consistency metrics; **4)** the *Policy Display* allows to view and modify both the query and constraints used; **5)** finally, the *Graph Display* shows the evolution of the consistency graphs for each defined constraint.

In practice, the demonstration follows the steps described below.

Policy Definition. First, Users must choose a use case scenario that provides the basic query and constraints. For instance, in the case of grid monitoring, the constraints SC1 and SC2, the query is the one from Listing 1, as shown in Figure 5. The demonstration will

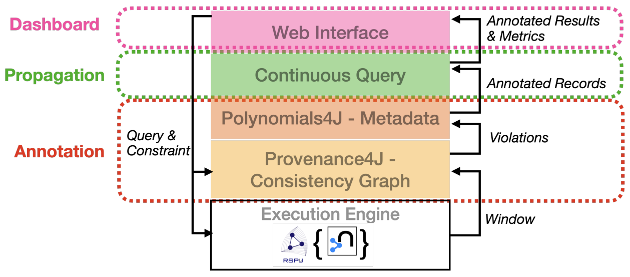
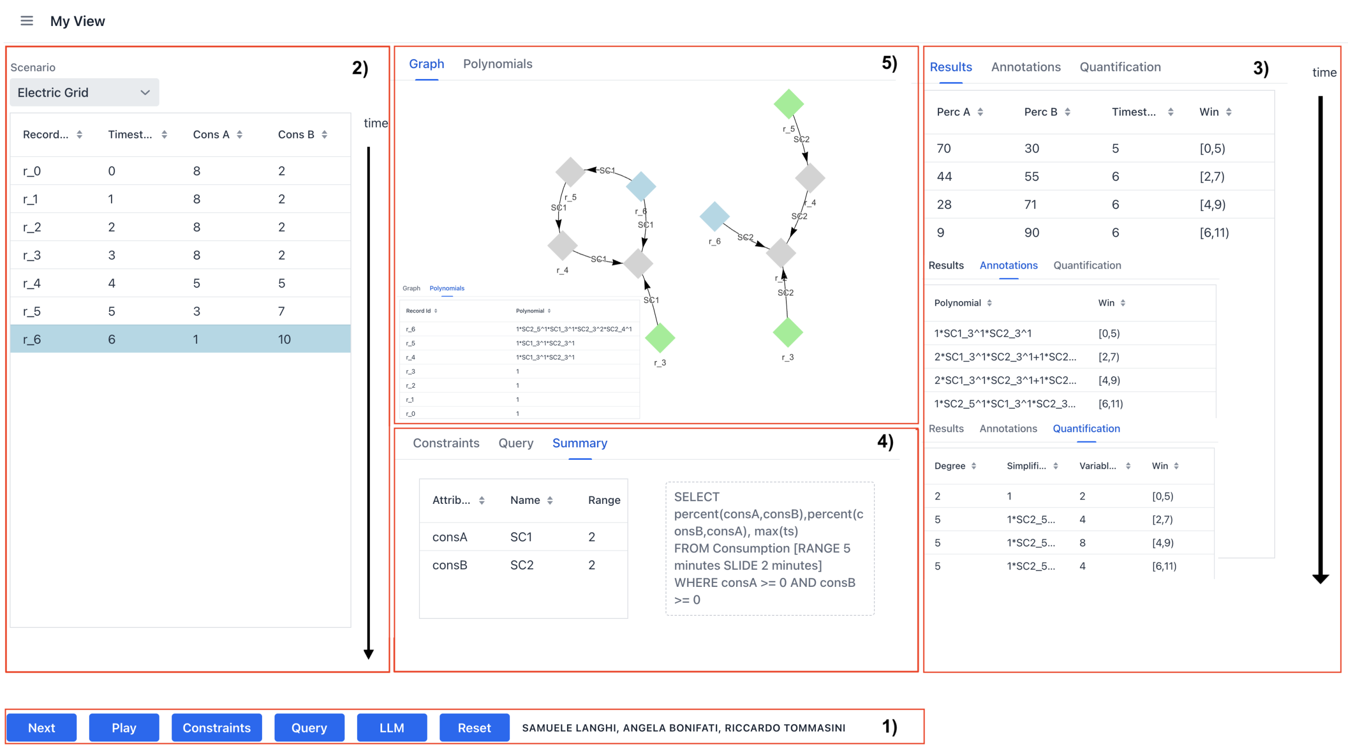


Figure 4: The Architecture of the system.



Scenario: Electric Grid

Record...	Timest...	Cons A	Cons B
r_0	0	8	2
r_1	1	8	2
r_2	2	8	2
r_3	3	8	2
r_4	4	5	5
r_5	5	3	7
r_6	6	1	10

Graph Polynomials

Record Id	Polynomial
r_6	1*SC2_5^1*SC1_3^1*SC2_3^2*SC2_4^1
r_5	1*SC1_3^1*SC2_3^1
r_4	1*SC1_3^1*SC2_3^1
r_3	1
r_2	1
r_1	1
r_0	1

Constraints Query Summary

Attrib...	Name	Range
consA	SC1	2
consB	SC2	2

```
SELECT
percent(consA,consB),percent(c
onsB,consA), max(ts)
FROM Consumption [RANGE 5
minutes SLIDE 2 minutes]
WHERE consA >= 0 AND consB
>= 0
```

Results Annotations Quantification

Perc A	Perc B	Timest...	Win
70	30	5	[0,5)
44	55	6	[2,7)
28	71	6	[4,9)
9	90	6	[6,11)

Results Annotations Quantification

Polynomial	Win
1*SC1_3^1*SC2_3^1	[0,5)
2*SC1_3^1*SC2_3^1+1*SC2...	[2,7)
2*SC1_3^1*SC2_3^1+1*SC2...	[4,9)
1*SC2_5^1*SC1_3^1*SC2_3...	[6,11)

Results Annotations Quantification

Degree	Simplifi...	Variabl...	Win
2	1	2	[0,5)
5	1*SC2_5...	4	[2,7)
5	1*SC2_5...	8	[4,9)
5	1*SC2_5...	4	[6,11)

Figure 5: The InkStream Web Interface.

include other scenarios of different levels of complexity, including Stock Correlation Analysis, Review Bombing, and GPS malfunction Detection. Once the scenario is defined, users can modify the query or the constraints in the Policy Display. For instance, monitoring of the electric grid may cover only those records where the cumulative consumption does not exceed a threshold, i.e., $\texttt{consA} + \texttt{consB} < n$.

Graph & Annotation Visualization. As events are ingested through the corresponding buttons in the control panel, users can visualize the consistency graphs' evolution in the upper part of the WUI, as shown in Figure 5. In particular, the WUI shows a consistency graph for each assigned constraint and the earliest reference window, whose label is reported on the graph's edges. Nodes are instead marked with the record label of the form r_i, where "i" is an incrementally assigned index. The graph put a visual highlight on the consistency of streaming records: disconnected nodes in the graph represent records that are inconsistent with all the others, while paths within the graph represent groups of consistent records. Consequently, the number of different paths within a connected component is also indicative of how inconsistent are the records. Moreover, the earliest added node is shown in blue, and the roots of the graph are shown in green, as in Figure 2. Figure 5 shows the consistency graph for the Electric Grid scenario when r_6 is added, thus reporting in the WUI the graph from Figure 2. In the "Polynomials" tab, users can visualize the polynomial annotations. As shown by their annotations, r_4, r_5, and r_6 are inconsistent wrt both SC1 and SC2, which indicates a sudden increase/decrease in the electricity consumption of both consA and consB.

Result Monitoring. Users can view the query result over the ingested records as the last step of the demonstration. Users can also view the annotations for each result, derived according to the input annotations and the positive algebra [3], shown in Section 2.3. In a separate tab, we provide different approaches for using the annotations to monitor consistency as reported in Section 2.3. For instance, in Figure 5 we reported the results of Table 2, where percentages calculated over window [2,7) in the electric grid scenario are the most inconsistent, being based on r_4, r_5, and r_6.

REFERENCES

[1] Arvind Arasu, Shivnath Babu, and Jennifer Widom. 2006. The CQL continuous query language: semantic foundations and query execution. *VLDB J.* 15, 2 (2006).

[2] 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/PODS 2024, Santiago AA, Chile, June 9-15, 2024*, Pablo Barceló, Nayat Sánchez Pi, Alexandra Meliou, and S. Sudarshan (Eds.). ACM, 605–612. https://doi.org/10.1145/3626246.3654679

[3] Todd J. Green, Gregory Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China.* ACM.

[4] Ousmane Issa, Angela Bonifati, and Farouk Toumani. 2020. Evaluating Top-k Queries with Inconsistency Degrees. *Proc. VLDB Endow.* 13, 11 (2020).

[5] Dimitris Palyvos-Giannas, Katerina Tzompanaki, Marina Papatriantafilou, and Vincenzo Gulisano. 2022. Erebus: Explaining the Outputs of Data Streaming Queries. *Proc. VLDB Endow.* 16, 2 (2022).

[6] Shaoxu Song, Aoqian Zhang, Jianmin Wang, and Philip S. Yu. 2015. SCREEN: Stream Data Cleaning under Speed Constraints. In *Proceedings of the 2015 ACM SIGMOD, Melbourne, Victoria, Australia, May 31 - June 4, 2015.* ACM.

[7] Michael Stonebraker, Ugur Çetintemel, and Stanley B. Zdonik. 2005. The 8 requirements of real-time stream processing. *SIGMOD Rec.* 34, 4 (2005).

[8] Riccardo Tommasini, Pieter Bonte, Femke Ongenae, and Emanuele Della Valle. 2021. RSP4J: An API for RDF Stream Processing. In *The Semantic Web - 18th International Conference, ESWC 2021, June 6-10, 2021, Proceedings.* Springer.