Play2Win: A Windowing Playground for Continuous Queries

Mauro Famà
INSA Lyon
Lyon, France
mauro.fama@insa-lyon.fr

Alessandro Ferri
TU Darmstadt
Darmstadt, Germany
alessandro.ferri@tu-darmstadt.de

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

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

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

ABSTRACT

Continuous Queries (CQs) are designed to operate over infinite data streams; the paradigm gained prominence with the rise of Stream Processing (SPs). Central to CQs are *window operators* as they enforce bounded computation by partitioning streams into finite subsets. Although several window operators exist —e.g., slide-by-tuple, session-window, and frames—commercial systems largely adopt a few due to implementation complexity, theoretical opacity, and input-dependent non-determinism. This demonstration shows Play2Win, an interactive playground that empowers users to explore and compare various windowing strategies under a unified system semantics. Our platform offers three key contributions: (I) a real-time environment for experimenting with different window operators; (II) a graph-based representation of the window state that eases direct comparison; and (III) a compositional framework for rapid prototyping of novel windowing mechanisms. The demonstration explore multiple datasets across different scenarios, fostering a deeper understanding of window operators for querying streams.

PVLDB Reference Format:
Mauro Famà, Alessandro Ferri, Samuele Langhi, Riccardo Tommasini, and Angela Bonifati. Play2Win: A Windowing Playground for Continuous Queries. PVLDB, 18(12): 5407 - 5410, 2025.
doi:10.14778/3750601.3750683

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/semlanghi/playtowin.

1 INTRODUCTION

Continuous queries (CQ) monitor streaming data sources over time, remaining active until explicitly terminated [4]. Introduced to query infinite data streams, CQs have recently gained popularity, given the emergence of Stream Processing (SPs), as a principled way to handle real-time data in various domains [1, 4].

CQs fundamental notion is *continuous semantics*, i.e., an infinite output is generated from an infinite input [6]. Several ways to achieve continuous semantics exist, but the most popular is the adoption of a *window operator*, which segments infinite data streams into bounded subsets for processing. Although several types of window operators exist [8], the majority of SPSs rely on a core subset of well-established windowing strategies, including *time-based* (hopping, sliding) and *session-based* windows, which can be efficiently implemented in a distributed setting [1].

One major obstacle to adopting advanced windowing techniques is that their theoretical definitions are often inaccessible to non-experts due to their mathematical complexity. Moreover, their implementations usually require a custom state design that each system's operational semantics can influence. Finally, non-deterministic window behaviour strictly depends on the input data, requiring non-trivial experimentation to understand the best configuration for a given use case. All these challenges make the adoption of advanced windowing slow in commercial SPS, leaving the cost of their implementation to their users.

In this demonstration, we advocate for enabling users to explore and compare different windowing mechanisms *within a consistent system semantic*. This would eliminate ambiguities introduced by system-specific behaviours and enable an *accurate comparison of different strategies*. More specifically, we propose an *interactive windowing playground*, providing an intuitive environment to experiment with various window types, observe their behaviour in real-time, and gain insights into their practical implications within stream processing workflows. In summary, this demonstration presents the following contribution: (I) We present an interactive playground for visualizing and experimenting with different windowing strategies, making their semantics more accessible to users. (II) We give a unified view of the window state based on a graph. Such a view enables direct comparison of window types regardless of the different implementations of their state. (III) We enable fast deployment of continuous queries with alternative window operators, allowing our users to experiment with a few use cases (i.e., electric grid, NYC taxi) and benchmarks, i.e., (Yahoo!, Nexkmarks and Linear Road). **Paper Outline**. Section 2 provides the system overview: it introduces the architecture and implementation of our interactive windowing playground. It presents our observable streaming pipeline and gives a primer on window operations. Section 3 presents the demonstration scenario, detailing how to choose window operators, size their parameters, and fast prototype custom window operators by composing existing ones.

Proceedings of the VLDB Endowment, Vol. 18, No. 12 ISSN 2150-8097.
doi:10.14778/3750601.3750683

2 SYSTEM OVERVIEW

This section introduces Play2Win. First, it presents minimal background knowledge, gives an overview of Play2Win architecture, and finally provides a primer on the included window operators.

2.1 Preliminaries

Continuous Queries, as Play2Win supports them, operate over data-agnostic input streams. Nevertheless, we limit our demonstration scenario to relational data.

Definition 2.1. A data *stream* is an ordered, infinite sequence of records (ω, τ), where ω represents a tuple in Ω, and τ is a timestamp in the time domain $\mathcal{T}$, which we assume to be the natural numbers.

For query syntax and semantics, we rely on the well-known Continuous Query Language (CQL) [2]. CQL semantics, as depicted in Figure 1, are based on the three families of operators: Stream-To-Relation (S2R), Relation-to-Relation (R2R), and Relation-to-Stream (R2S). S2R operators are responsible for dealing with the unboundedness of the input data; R2R represents the CQ core logic, while R2S allows the result to be streamed back. We focus on the S2R family and, in particular, window operators:

Definition 2.2. A window operator is a function $W : \mathcal{T} \rightarrow I$ mapping timestamps from $\mathcal{T}$ to intervals $I = \mathcal{T} \times \mathcal{T}$.

We give a primer on the window functions in Section 2.3; here, we focus on its abstract semantics. By applying the window function to a relational stream, we obtain a Time-Varying Relation. TVR is a view that can be materialized at any helpful timestamp t.

Definition 2.3. A time-varying relation TVR: $\mathcal{T} \rightarrow R$ is a function that for any time instant $\tau \in \mathcal{T}$ returns a finite bag of tuples R(τ).

To identify at what timestamps the TVR is materialised – we call this set $ET \subset \mathcal{T}$ as in evaluation time instants highlighted in grey in Figure 1 (b) – we need to understand the system reporting semantics. In practice, this is done automatically by the stream processor. Still, we can characterise the set also using the SECRET model [5], consisting of four primitives: *Scope* defines the time interval over which a query is evaluated. *Content* maps these window intervals to the stream elements they contain; *Tick* specifies what triggers system actions. Finally, *Report* determines when a window's content is ready for query evaluation.

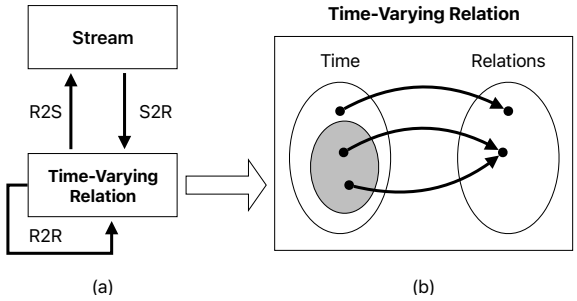

Figure 1: (a) CQL Operator Families [2] and (b) the Time-Varying Relation visualised. The set ET is in grey.

2.2 Architecture

Figure 2 illustrates Play2Win's architecture, our interactive windowing playground. The front (top) is divided into five areas, i.e., from left to right: Input, Window and Query, State, and Output.

The **Input View** (1) represents the input stream within a table, where the most recent records are incrementally appended. We chose this visualization to simplify comprehension for users unfamiliar with streaming data, as they can see it as an ever-growing append-only table [3]. In the context of this paper, the system supports one stream as input with a single schema. Users can also check the record attributes and values, as the system supports multiple scenarios, each with its input stream and schema.

The **Window View** (2) is the control centre of the demo. The system provides an editor menu for defining window policies, spanning multiple options, e.g., hopping, sliding, session windows [1], or Frames [8]. Once selected, each policy can be further defined by specifying the related parameters. For instance, hopping windows have *size* (α), and *slide* β, session windows have the timeout γ.

In the **Query View** (3), users input queries in SQL-like. They can select queries from a given use case or write a new one. A placeholder keyword leaves room for various window definitions (see Listing 1). Users can select multiple window operators, each deployed as an independent query. Table 1 shows a subset of window operators and their parameters.

```
SELECT *
FROM <window> ON Electric_Grid
WHERE consB >= 0;
```

Listing 1: SQL Continuous Query with window Placeholder.

The **State View** (4) shows the *state* of each window operator. Indeed, window operators are stateful operators, and a proper state design is essential for the operator's correctness and efficiency (throughput and latency). However, different window operations may have very different state architectures. For instance, time-based hopping windows are usually implemented with a HashMap whose keys are the window intervals, and values are the window content. Conversely, frames are implemented as lists, given their non-deterministic nature.

Definition 2.4. A *Window State Graph* (WSG) is a directed graph WSG $= (V, E)$, i.e., $V = V_I^W \cup V_\Omega$ and $E = E_W \cup E_{I^2}$, where

- V_I^W is the set of nodes representing window intervals, as determined by a window function W;
- V_Ω is the set of nodes representing the tuples in the data stream.

Table 1: Windowing operators summary. (Dom)ains: $\mathbb{Q}$ rationals; $\mathbb{N}$ naturals, $\mathcal{T}$ time; f is sum/count/avg/max/min.

Window	Params. (Dom)	Gaps (if)	Det.	State
Hopping	$\alpha(\mathcal{T})$,β $(\mathcal{T})$	×	✓	Map
Session	γ $(\mathcal{T})$	×	×	List
SBTW	$\alpha(\mathcal{T})$,$\theta(\mathbb{N})$	✓	×	Queue
Aggregate	f,col(Ω),$\leq \mid \geq$,$\iota(\mathbb{Q})$	× (≥) ✓ (≤)	×	Custom
Threshold	col(Ω),$\leq \mid \geq$,$\iota(\mathbb{Q})$	✓	×	List
Delta	col(Ω),$\leq \mid \geq$,$\iota(\mathbb{Q})$	× (≤) ✓ (≥)	×	List

- $E_W \subseteq V_\Omega \times V_I^W$ is a set of edges indicating which tuples belong to which window intervals (i.e., *membership edges*);
- $E_{I^2} \subseteq V_I^W \times V_I^W$ is a set of edges representing dependencies or transitions between window intervals (i.e., *transition edges*).

Play2Win leverages such graph-based state representation (cf. Definition 2.4) and visualisation to make different window operators comparable. Records (blue/squares) are connected to the corresponding window instances via membership edges; window instances (circles/colour depends on the operator) are connected via transition edges (labelled according to their temporal relations, e.g., Overlaps/After). Records in red do not belong to any window instance (for those operators that allow gaps).

The **Output View** (5) shows the result of the deployed queries. We have a different tab showing the results for each query, named according to the window operator. As we follow CQL, we must choose an R2S operator. For simplicity, we limit our view to the RStream operator [2], represented in grey in Figure 2. In addition to the query result, the Output View also shows a tabular view of the window state. By selecting record nodes, we trigger a comparative tab, showing each selected record which window instances they belong to for each defined policy. The **Backend** of Play2Win is designed using a new version of RSP4J [7] (Polyflow). This allows implementing an *observable* continuous query execution. Indeed, Polyflow's query model maps one-to-one to CQL. In particular, Polyflow explicitly includes the programming abstractions for DataStream (cf Definition 2.1) and *Time-Varying Object* (cf Definition 2.3), allowing to lift their control to the user via Play2Win UI. Moreover, for querying, Polyflow contains the corresponding classes for CQL operator families. We could indeed easily implement several stat-of-the-art window operators while maintaining the same high-level API.

Figure 2 shows the design of our observable CQ execution pipeline. A user-defined query consists of an S2R operator, a set of R2R operators organised in a Directed Acyclic Graph (DAG) (the R2S for outputting the result is fixed to the RStream). Such operators are registered upon click, organised into *Tasks* (6), and globally coordinated by the Polyflow library. Applying the S2R to the input stream instantiates a Time-Varying Relation (7) that is accessible to the R2R for canonical query execution and to the UI for observing the content of the windows. In such cases, the TVR is materialised according to the set of evaluation time instant ET, which is the result of the reporting condition [5]. On the UI, we materialise TVR into

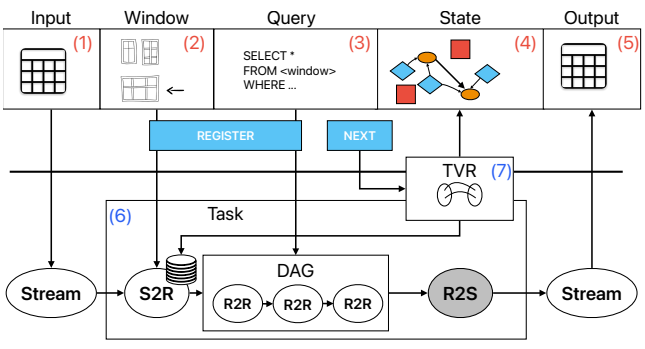

Figure 2: Play2Win Architecture and Flow.

the window state at every incoming event (When clicking the next button). Notably, the window state is mapped to our graph-based representation for better inspection via visualisation.

2.3 Window Operators Primer

We included windowing operators from [8], e.g., deterministic fixed-size time windows, data-driven windows, and session. Below, we present the selected window operators' features (cf Table 1)

Time-Based Hopping Windows (TW) divide the stream into fixed-size time segments. TW require three parameters: an initial timestamp τ, the window size α, and the slide β. We can define the set of sliding windows overlapping at τ as the collection of windows with time intervals: $[t_{close} - \alpha, t_{close})$ where $t_{close} = c_{sup} + k \cdot \beta \wedge k \in \mathbb{N}_0$, with $0 \leq k < \frac{\alpha}{\beta}$, and $c_{sup} = \lceil \tau \div \beta \rceil \cdot \beta$.

Slide-by-Tuple Windows (SBTW) are characterised by a duration parameter α like TW, but the slide $\theta \in \mathbb{N}$ determines the number of tuples, after which a new window starts. Because SBTW slides when a new tuple arrives, it is impossible to determine the stream segmentation before the computation starts.

Session Windows (SW) capture periods of activity in a data stream. A session starts with the arrival of a tuple and remains open as long as subsequent tuples arrive within a given timeout γ. SWs are non-deterministic and do not have a fixed duration since they expand until new tuples arrive before the timeout period.

Frames are non-deterministic data-driven windows where segmentation depends on tuple attributes rather than timestamps and a condition predicate. When the condition is met, the current frame ends, and a new one starts. This demo includes: (i) *Threshold* : the value of a specified attribute (col) of subsequent tuples is above (or below) a given threshold ($\iota \in \mathbb{Q}$). (ii) *Delta*: the difference between the attribute's first and last tuple value is above (or below) a given threshold ($\iota \in \mathbb{Q}$). (iii) *Aggregate*: the result of an aggregation over the tuple's attribute is above (or below) a given threshold ($\iota \in \mathbb{Q}$). Where $\mathbb{Q}$ is the set of rational numbers.

3 DEMONSTRATION SCENARIOS

Our demonstration will include several datasets and queries from different application domains, i.e., the DEBS challenge 2015[1] (NYC Taxi), the data from the benchmarks Nexmark[2] (auctions) and Linear Road[3] (transportation), and an example of electric grid monitoring. We develop two scenarios: selecting the window operator and setting the operator parameters. Below, we explore such scenarios using an *electric grid monitoring* using Figure 3 for reference.

Choosing the Window Operator. This scenario demonstrates how to select an appropriate window operator for electricity consumption monitoring (Figure 3, (1)). We will present two cases needing different window operators: a camping site with an energy-based billing model and an apartment with a time-based tariff system. Users will interact with Play2Win, selecting multiple windows and deploying the query (cf Listing 1) with different S2R.

Figure 3 shows that a Frame is the most appropriate choice for analyzing the camping site data rather than a time-based window; electricity is sold in 15 kWh increments, and a new 15 kWh block

[1] https://chriswhong.com/open-data/foil_nyc_taxi/
[2] https://github.com/nexmark/nexmark
[3] https://www.cs.brandeis.edu/~linearroad/

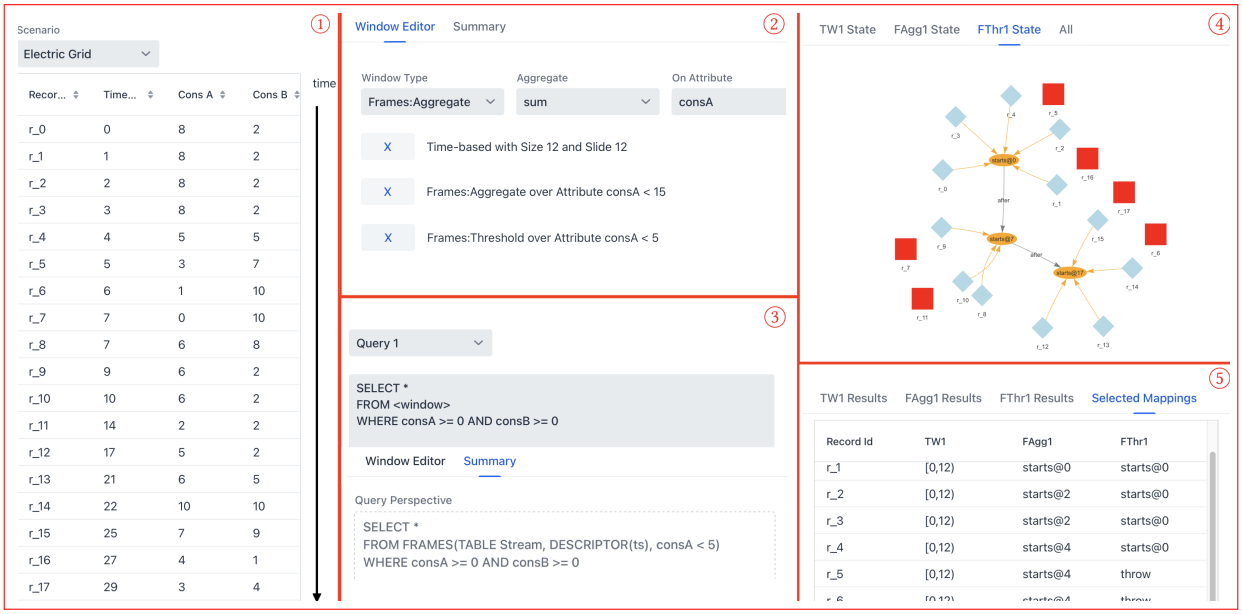

Figure 3: The user interface of the Play2Win system.

is supplied when the previous one is exhausted. Since time is not a factor, we need a window operator considering excess consumption. The users will see what Frames to consider, e.g., Threshold, Delta, or Aggregate. The first two are inadequate as they are too fine-grained (a single tuple or max/min). Finally, the Aggregate is selected for the case (Figure 3, ②). Similarly, we interact with the users to select the appropriate window operator for the apartment billing case. Here, electricity is charged on two 12-hour tariff periods, and a time-based window is best. Hence, either a hopping, a sliding, or a session window. We opt for the first (Figure 3, ②) as we are neither interested in inactivity nor tuple-level sliding. Again, we can deploy different examples and show the differences.

Sizing the Window Parameters. Once the appropriate window operator is selected, the user must configure its parameters to align with the billing requirements. Play2Win provides options for setting window sizes and aggregation functions, which the user will adjust for the camping site and apartment scenarios.

The user configures the aggregate frame function and threshold for the camping site. Albeit simple in the example, we will show what happens when the Aggregate is misconfigured, e.g., the wrong threshold or wrong function. Indeed, Play2Win provides different aggregation functions, and the user can compare their effects.

For the apartment case, our users can observe that a hopping window with overlapping time segments would lead to incorrect billing by counting energy in multiple windows (Figure 3, ④ and ⑤). Thus, we need to set the slide equal to the width to obtain a Tumbling Window of 12 hours. This will ensure that each window corresponds to a billing phase. Play2Win allows the user to experiment with different sizes; for instance, we will try out a 24-hour window to merge the two tariff periods, preventing correct differentiation between day and night rates. A 6-hour window would introduce unnecessary complexity by creating unaligned segments.

4 CONCLUSION

In this demonstration, we presented *Play2Win*, an interactive playground that exposes the semantics of window operators through a unified system, real-time visualization, and graph-based state inspection. Play2Win lowers the barrier to understanding, comparing, and prototyping continuous query behaviour involving various window operators. Ultimately, Play2Win fosters a deeper understanding of windowing behaviour using the graph-based state representation, which helps users make informed decisions when deploying continuous queries.

REFERENCES

[1] Tyler Akidau, Slava Chernyak, and Reuven Lax. 2018. *Streaming systems: the what, where, when, and how of large-scale data processing.* " O'Reilly Media, Inc.".

[2] Arvind Arasu, Shivnath Babu, and Jennifer Widom. 2006. The CQL continuous query language: semantic foundations and query execution. *VLDB J.* 15, 2 (2006), 121–142. https://doi.org/10.1007/S00778-004-0147-Z

[3] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. 2002. Models and Issues in Data Stream Systems. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, Lucian Popa, Serge Abiteboul, and Phokion G. Kolaitis (Eds.). ACM, 1–16. https://doi.org/10.1145/543613.543615

[4] 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 Chile*, P. Barceló, N. Sánchez-Pi, A. Meliou, and S. Sudarshan (Eds.). ACM.

[5] Nihal Dindar, N. Tatbul, RJ. Miller, L M. Haas, and I. Botan. 2013. Modeling the execution semantics of stream processing engines with SECRET. *VLDB J.* (2013).

[6] Douglas B. Terry, David Goldberg, David A. Nichols, and Brian M. Oki. 1992. Continuous Queries over Append-Only Databases. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 2-5, 1992*, Michael Stonebraker (Ed.). ACM Press.

[7] Riccardo Tommasini, Pieter Bonte, Femke Ongenae, and Emanuele Della Valle. 2021. *RSP4J: An API for RDF Stream Processing.* 565–581. https://doi.org/10.1007/978-3-030-77385-4_34

[8] Juliane Verwiebe, Philipp M. Grulich, Jonas Traub, and Volker Markl. 2023. Survey of window types for aggregation in stream processing systems. *VLDB J.* (2023).