Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Bacharelado em Ciência da Computação

Stream and Historical Data Integration using SQL as Standard Language

Jefferson do Nascimento Amará

JUIZ DE FORA

SETEMBRO, 2021

Stream and Historical Data Integration using SQL as Standard Language

Jefferson do Nascimento Amará

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Victor Stroele de Menezes

JUIZ DE FORA

SETEMBRO, 2021

Stream and Historical Data Integration using SQL as Standard Language

Jefferson do Nascimento Amará

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

JUIZ DE FORA

02 DE SETEMBRO, 2021

A Deus.

Resumo

Volume, Velocidade, Variedade, Veracidade e Valor, além de ajudar a definir o significado do termo *Big Data*, passaram a fazer parte da realidade dos sistemas que lidam com *streaming* de dados produzidos por dispositivos IoT. Nesse contexto, a heterogeneidade dos dados se apresenta como um desafio para esses sistemas no que se refere à integração e monitoramento. A complexidade imposta pela heterogeneidade de dados torna difícil integrar os tipos de dados *'streaming x streaming'* e *'streaming* x histórico'. Para uma análise prática, o processo de enriquecimento e contextualização com base em dados históricos e de *streaming* se beneficiaria de abordagens que facilitem a integração de dados, abstraindo das análises os detalhes e o formato dessas fontes primárias. Este trabalho apresenta um *framework* que permite a integração de dados de *streaming* e dados históricos em tempo real, abstraindo dos aspectos sintáticos das consultas do usuário através do uso de SQL como linguagem padrão para consultar fontes de dados heterogêneas. O *framework* foi avaliado por meio de um experimento utilizando bancos de dados relacionais e dados reais produzidos por sensores. Os resultados apontam para a viabilidade da abordagem.

Palavras-chave: Dados Heterogêneos, Integração de Dados, *Streaming* de Dados, SQL.

Abstract

Volume, Velocity, Variety, Veracity, and Value of data have, in addition to help define the meaning of the term Big Data, become part of the reality of systems that deal with data stream produced by IoT devices. In this context, data heterogeneity presents itself as a challenge for these systems concerning integrating and monitoring this data. The complexity imposed by data heterogeneity makes it difficult to integrate 'streaming x streaming' and 'streaming x historical' data types. For practical analysis, the enrichment and contextualization process based on historical and streaming data would benefit from approaches that facilitate data integration, abstracting from the analyses the details and format of these primary sources. This work presents a framework that allows the integration of streaming data and historical data in real-time, abstracting from the user syntactic aspects of queries through the use of SQL as a standard language for querying heterogeneous data sources. The framework was evaluated through an experiment using relational databases and real data produced by sensors. The results point to the feasibility of the approach.

Keywords: Heterogeneous Data, Data Integration, Data Streaming, SQL.

Agradecimentos

Dou graças a Deus por me suster até aqui e por ser a minha força durante as lutas. Agradeço à minha família pela educação e pelos valores cultivados, e por às vezes abrir mão de muitas coisas para me prover o mínimo pra eu chegar até aqui. Em especial, agradeço à minha mãe, a guerreira que nunca mediu esforços pra criar seus filhos, e mesmo com muita luta, me trouxe até aqui. Agradeço à família da minha noiva, Jane, Claudio, Josué e Lucas; por terem me acolhido no seu meio e sempre me apoiado, e a ela, Luanasarah, por ser uma companheira formidável em todos os momentos, me apoiar sempre, me motivar, minha amiga, um presente de Deus pro resto da minha vida. Agradeço aos meus amigos por serem os melhores. Por dividirem os momentos de desafios, lutas e conquistas. Em especial agradeço ao Jurandir, Samuel, Pedro e Wallace, por serem mais que meus amigos, meus irmãos. Ao Belonir e a Tia Edna, por me adotarem como um filho mais novo nessa família, sem eles tudo seria muito mais difícil. Ao meu professor e amigo Rafael Bonfim, por dedicar a mim seu tempo para compartilhar conhecimento, amizade e conselhos em momentos bons e ruins. Agradeço muito ao meu orientador, professor e amigo Victor Stroele, o qual admiro pela excelência como professor e passei a admirar ainda mais como pessoa, sempre me aconselhando, tanto sobre a vida acadêmica profissional e pessoal, e sem o qual este trabalho não seria possível. Agradeço à UFJF, em especial ao DCC, na pessoa dos seus TAES, por serem sempre cordiais e sempre buscar resolver os problemas com um trato bem humanizado e com o mínimo de burocracia possíveis. Aos professores deste departamento, aos quais tenho respeito e admiração, são exemplos pra mim, agradeço por terem me proporcionado um ensino diferenciado, por nunca criar barreiras na comunicação, por motivar e desafiar a ampliar o modo de ver e analisar tudo, e a buscar fazer diferença por onde passo. Agradeço ao CRITT, por ter me proporcionado tantas amizades, desafios e aprendizados ao longo da minha trajetória como bolsista. Agradeço a todos envolvidos nessa vitória, é uma vitória compartilhada com muitos, cada um que de alguma forma me proporcionou chegar até aqui. Tenho

certeza de que saio desse desafio como um ser humano e um profissional melhor do que entrei. Obrigado.

"In God we trust, all others must bring data."

W. Edwards Deming

Contents

List of Figures

List of Tables

Lista de Abreviações

DCC	Departamento de Ciência da Computução
UFJF	Universidade Federal de Juiz de Fora
DBMS	Data base management System
DSMS	Data stream management System
IoT	Internet of Things
IIoT	Industrial Internet of Things
SQL	Structured Query Language
QoS	Quality of Service
CEP	Complex Event Processing
CQL	Categorical Query Language
QODI	Query-driven Ontology-based Data Integration
RDF	Resource Description Framework
SESQ	Semantically Enriched SQL
API	Application Programming Interface
JSON	JavaScript Object Notation
CSV	Comma Separated Value

1 Introduction

The world and its relationship with data are migrating from data islands to a global data space paradigm. If a few years ago the term Big Data was exclusive in the scientific works and not well known in the daily life of civil society, nowadays it is a reality that permeates the routines of people (ABU-SALIH et al., 2021; BARROS, 2020). It is now of paramount importance in decision-making in the most diverse areas of knowledge and the global economy (GHASEMAGHAEI; CALIC, 2020; WANG et al., 2020).

In recent years, organizations have been dedicating themselves to leveraging the intelligent use of the vast amount of data produced (MIKALEF et al., 2020; SHAN et al., 2019). The ability to manipulate efficiently this information and extract knowledge is now seen as a key factor in gaining a competitive advantage (CAVANILLAS; CURRY; WAHLSTER, 2016).

In addition to traditional data sources, which are modeled through persistent relations, applications with transient relations have become increasingly common. Also known as Data Streams, data originated from sources like IoT, sensor networks, mobile applications and social networks add, among others features, volume and heterogeneity to this global space of data (AKANBI; MASINDE, 2020).

These *"new"* sources of data are characterized by being open-ended, flowing at high-speed, and generated by non-stationary distributions in dynamic environments (GAMA, 2012). Also have become ubiquitous, due to the fact that a number of applications generate a huge amount of data at a great velocity. This made it difficult, for example, for existing data mining tools, technologies, methods, and techniques to be applied directly on big data streams due to the its inherent dynamic characteristics (KOLAJO; DARAMOLA; ADEBIYI, 2019).

Kolajo, Daramola and Adebiyi (2019) point to the relevance and increasing importance of researches in data stream field and trends of big data stream tools and technologies, as well as methods and techniques employed in analysing big data streams. They also point to the Data Integration as a key issue in big data stream analysis.

To highlight the emerging relevance of themes related to the big data stream, the graph in Figure 1.1 shows the increase in the number of works involving topics such as *"big data stream analysis", "big data stream technologies", "big data stream analysis tools" or "big data stream processing"*, which were published in the main scientific dissemination platforms from 2014 to 2018.

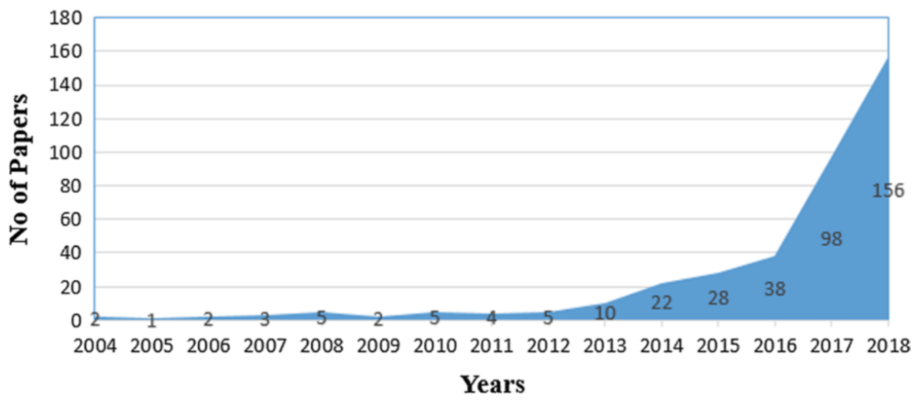


Figure 1.1: Magnitude of change in paper distribution (Kolajo et al. 2019).

Asano et al. (2019), Tatbul (2010)point to the need of integrating data of different origins, whether historical or stream, structured or not. Several works have been developed in order to promote query mechanisms capable of integrating streaming data, addressing aspects of semantic optimization (CAPPUZZO; PAPOTTI; THIRUMURUGANATHAN, 2020; ALKHAMISI; SALEH, 2020), continuous queries with sliding windows (SHEIN; CHRYSANTHIS, 2020), time alignment of queries (TU et al., 2020) and aspects related to scalability (STONEBRAKER; ILYAS, 2018).

In the context of *Industrial IoT (IIoT)* applications for example, Costa et al. (2020) present a solution for real-time integration of data produced by *IoT* devices. In their approach, data from intelligent devices, sensors and robots are extracted, processed, and stored in diverse, independent, and heterogeneous repositories. In that work, however, there is no proposal for the integration of data for monitoring in a unified and simplified way, that is, the complexities and features intrinsic to each data repository have their treatments delegated to the solution's consumers. Thus the integration for monitoring these repositories maintains the complexity at the level of user queries according to the characteristics of each repository.

It is worth noting that when it comes to data integration, the solutions that have had an impact are those that can be explained and easily comprehended by a human (MILLER, 2018). Furthermore, the effort involved in querying this data can create barriers for consumers (WANG; HAAS; MELIOU, 2018). A central problem is a semantic gap between the way users express their queries and the different ways that data is represented internally (FREITAS; CURRY, 2014).

1.1 Objectives

With this context in mind, the objective of this work is to allow queries that integrate stream and historical data to be performed more easily, that is, in a more standardized way, regardless of the origin, format, model or heterogeneity of the data. For this purpose, this work followed four main steps: (i) review of related literature; (ii) definition of an architecture for executing SQL queries for data integration, and selection from heterogeneous sources; (iii) implementation of abstraction classes for the internal characteristics of data sources; and (iv) evaluation of results.

1.2 Research Questions

The following research questions were derived: **Q1)** Can the framework be used as a tool for joining stream and historical data described by heterogeneous data formats and models?; **Q2)** Is the solution extensible, that is, is it possible to add other data sources (stream or historical) in a simplified way?

1.3 Organization

The literature review includes articles related to streaming data integration. The architecture was proposed to enable the selection of data from heterogeneous sources and the execution of queries in standard SQL language. The architecture was developed based on the Apache Calcite[1] technology, and stream sensor and historical weather data were used to evaluate the proposal.

[1]https://calcite.apache.org/

This work is organized as follows: Chapter 2 presents a theoretical foundation about the related themes. Chapter 3 provides an overview of some related work; Chapter 4 describes the approaches and methods, including an overview of the proposal's architecture and infrastructure; In Chapter 5 we present a feasibility study; Finally, Chapter 6 presents conclusion and future directions.

2 Theoretical Foundation

2.1 Introduction

2.1.1 *Internet of Things*

The Internet of Things (IoT) is a well-known paradigm that defines a dynamic environment of interrelated computing devices with different components for seamless connectivity and data transfer (STOYANOVA et al., 2020).

IoT is implemented in most of the areas in day-to-day applications which covers lifestyle, retail, city, building, transportation, agriculture, healthcare, environment, and energy (LEE; LEE, 2015). Some of the applications are smart homes, smart cities, smart energy, and smart industry, etc (SMYS, 2020).

According to Zanella et al. (2014), with so many data sources from such heterogeneous fields of application, it becomes a challenge to find solutions that meet the requirements of all scenarios, from aspects related to the acquisition infrastructure, passing through storage paradigms and the process of acquiring useful knowledge from this volume of data.

The IoT scenario is the era of streaming data that are usually represented in different structures or even semi/non-structures. As such, capturing and/or transferring those heterogeneous data in different formats (e.g., .csv, .txt, .html, .xml and so on) into a unified form, which is suitable for analysis, is a challenging task (CHEN et al., 2013). Another issue revolves around how to integrate IoT streaming data from multiple sources on the fly in real-time, as big data query and indexing (TU et al., 2020).

In the present work, the context of *IoT* is justified, since the primary source of the data used in the research is precisely distributed environmental sensors.

2.1.2 *Data Streaming*

A data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items (GOLAB; ÖZSU, 2003). According to Gama and Gaber (2007) data stream is an ordered sequence of instances that can be read only once or a small number of times using limited computing and storage capabilities. With these definitions in mind, Data Streaming can be defined as the process of transmitting a continuous flow of data (also known as streams), typically fed into stream processing software to derive valuable insights (TIBCO, 2019).

Data streams are usually generated by external sources or other applications and are sent to a Data Stream Management System (DSMS). Typically, DSMS do not have direct access or control over the data sources (JIANG; CHAKRAVARTHY, 2009).

It is a new paradigm useful because of new sources of data generating scenarios which include ubiquity of location services, mobile devices, and sensor pervasiveness (YI et al., 2014). Examples of such applications include financial applications, network monitoring, security, telecommunications data management, web applications, manufacturing, sensor networks, and others.

In these applications it is not feasible to load the arriving data into a traditional data base management system (DBMS) and traditional DBMS are not designed to directly support the continuous queries required by these applications (BABCOCK et al., 2002).

The fundamental assumption of this paradigm is that the potential value of data lies in its freshness and with stream computing organisations can analyse and respond in real-time to rapidly changing data (KOLAJO; DARAMOLA; ADEBIYI, 2019).

Also according to Kolajo, Daramola and Adebiyi (2019), the essence of big data streaming analytics is the need to analyse and respond to real-time streaming data from diverse sources, using continuous queries so that it is possible to continuously perform analysis on the fly within the stream. In this scenario, data are analysed as soon as they arrive in a stream to produce result as opposed to what obtains in batch computing where data are first stored before they are analysed.

Jiang and Chakravarthy (2009) also highlight some characteristics about data stream itself and **Data Stream Applications**, which is organized as follow:

- Continuous processing of newly arrived data is necessary.
- Many applications can tolerate approximate results as long as other critical (e.g., response time) requirements are satisfied.
- Many applications have very specific Quality of Service (QoS) requirements.
- Usage of available resources (e.g., cpu cycles, memory) to maximize their impact on QoS metrics is critical for stream-based applications.
- Finally, Complex Event Processing (CEP), rule processing, and notification are other important requirements of many stream-based applications that detect events or conditions and have to fire rules/triggers/actions in a timely manner when abnormal or user-defined events are detected.

Golab and Özsu (2003) list the **Data Stream characteristics** as follow:

- **The arriving**: The data elements arrive in a continuous way, keeping a relative order to the sequence itself.
- **The notion of time**: The idea of time is incorporated, be in the underlying data model or in the sequence.
- **The data origin**: The data source is associated with the data stream (e.g. a temperature sensor), that is to say, the data origin is unmodifiable by any data processor.
- **The input from the data stream**: It is unpredictable in terms of rate or volume of data.
- **The Data Model**: Each data stream could have its own underlying data model or not (e.g. it could be semi-structured such as an XML document, or not structured like a text file).
- **Data Reliability**: The data inside of the data stream are processed like they are, they are not free of errors because depend on the data source and the transited path.

2.1.3 *Batch vs Stream Data Processing*

It is possible to group Big Data Processing into two major groups according to the state of art of this domain: Batch Data Processing and Stream Data Processing (SAKR et al., 2015).

Batch processing is a well known paradigm which involves operating over a large, static dataset and returning the result at a later time when the computation is complete. Thus, the state of data is maintained for the duration of the calculations during the processing (GURUSAMY; KANNAN; NANDHINI, 2017). This is typically ideal for non-time sensitive work.

According to Gurusamy, Kannan and Nandhini (2017) the datasets in batch processing are bounded(batch datasets represent a finite collection of data), persistent(data is almost always backed by some type of permanent storage), large(batch operations are often the only option for processing extremely large sets of data).

While batch processing is a good fit for certain types of data and computation, other workloads require more real-time processing (VENKATESH et al., 2019). Stream Data Processing is required for these needs, most of the data generated in a real-time data stream need real-time data analysis. In addition, the output must be generated with low-latency and any incoming data must be reflected in the newly generated output within seconds (KOLAJO; DARAMOLA; ADEBIYI, 2019).

Stream data processing is characterized by the constant input of new data or updates thereof, with an infinite amount of data or the size not known in advance, and with input data processing taking place in a few seconds or milliseconds given the need for real-time analysis. Historical data processing, on the other hand, is characterized by a normally known and finite amount of data in which data entry occurs in data chunks. Its processing takes place in multiple rounds and with a considerably longer processing time, when compared to streaming processing. Table 2.1 summarizes some of the differences about the main dimensions considered in between batch and stream data processing.

Also according to Gurusamy, Kannan and Nandhini (2017), the datasets in stream processing are considered "unbounded". This has a few important implications:

- The total dataset is only defined as the amount of data that has entered the system

Table 2.1: Comparing Batch and Streaming processing (KOLAJO; DARAMOLA; ADEBIYI, 2019)

Dimension	Batch	Streaming
Input	Data chunks	Stream of new data or updates
Data Size	Known and finite	Infinite or unknown in advance
Hardware	Multiple CPUs	Typical single limited amount of memory
Storage	Store	Not store or store non-trivial portion in memory
Processing	Processed in multiple rounds	A single or few passes over data
Time	Much longer	A few seconds or even milliseconds
Applications	Widely adopted in almost every domain	Web mining, traffic monitoring, sensor networks

so far.

- The working dataset is perhaps more relevant and is limited to a single item at a time.
- Processing is event-based and does not ”end” until explicitly stopped. Results are immediately available and will be continually updated as new data arrives.

For dealing with them both, stream and batch processing, one of the most common system architecture is called Lambda Architecture (KIRAN et al., 2015). It combines two different processing layers namely batch and speed layers, each providing specific views of data while ensuring robustness, fast and scalable data processing (YOUSFI; RHANOUI; CHIADMI, 2021).

Figure 2.1 shows a Lambda Architecture’s schema. It caters as three layers (1) Batch processing for pre-computing large amounts of data sets (2) Speed or real-time computing to minimize latency by doing real time calculations as the data arrives and (3) a layer to respond to queries, interfacing to query and provide the results of the calculations (KIRAN et al., 2015).

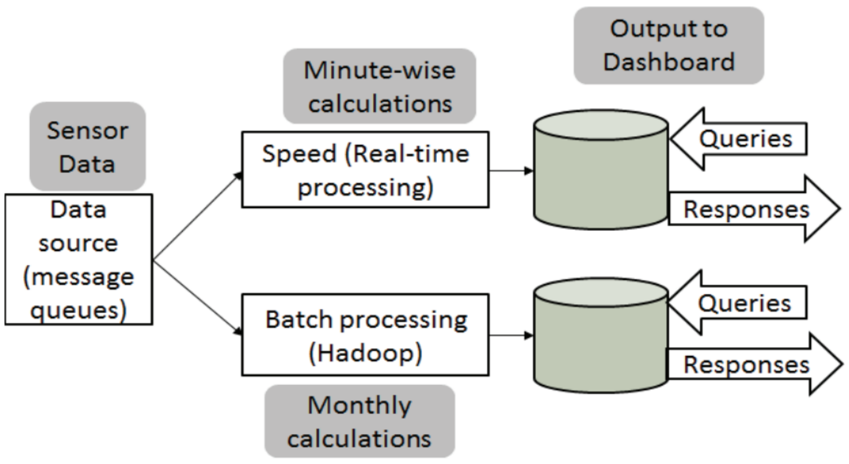


Figure 2.1: Basic lambda architecture for speed and batch processing. Kiran et al. (2015)

In the present proposal this architecture is adopted once is designed to process large volume of historical data, as well as rapid incoming data streams.

3 Related Work

Considering the research areas on which this paper draws, we have selected articles addressing data integration in streaming applications and historical data, and query mechanisms for real-time monitoring, to understand important aspects of the theme and limitations of existing solutions.

Asano et al. (2019) introduce Dejima, a framework focused on system aspects for data integration and control of update propagation of multiple databases. According to the authors, their solution combines two previous approaches to data integration; the first based on the global data schema, in which the data is integrated among a few databases using a single global schema, and the other based on the concept of 'peer' where the propagation of updates is cascaded through the peer networks. The authors do not present structural aspects of the queries.

In the research of Brown, Spivak and Wisnesky (2019), the focus is on ensuring data integrity during the data migration process, presenting CQL (Categorical Query Language) as an intuitive language to allow the movement and integration of data with complex schemas. They also point to the need for tools to combine heterogeneous datasets.

Tian, Sequeda and Miranker (2013) present QODI (Query-driven Ontology-based Data Integration), an algorithm for dynamic mapping and query reformulation. They demonstrate QODI as a solution for data integration in heterogeneous distributed database systems. The queries are performed by ontology users through queries in the SPARQL language and translated to their destination databases. Although QODI is designed to integrate RDF (Resource Description Framework) data, its main motivation is the integration of relational data.

To support the query process with context enrichment, Cavallo et al. (2018) present a semantic labeling module for performing queries in RDF statements with a query engine that combines SPARQL and SQL queries. They introduce the syntax of the query language with context enrichment SESQL (Semantically Enriched SQL).

Considering the works mentioned above, their gaps and limitations, as well as the problems identified by the authors themselves, the approach proposed in this present work presents itself as a feasible solution to the highlighted points. The main contributions of this work are:

1. Allow the integration of data from distributed repositories;
2. Allow the integration of data between heterogeneous data sets, structured or not, relational or not;
3. Promote the execution of real-time queries in streaming and historical data;
4. Provide the abstraction of syntactic aspects of the queries at the model level of the datasets, providing the possibility of queries through the use of the SQL standard.

4 Material and Methods

This work proposes a framework to facilitate the integration of streaming data produced by sensors with historical data. In this framework, the SQL language is used as the standard language, since it is the most used language for database queries (TOMAN, 2017). We assume some sensors produce a stream of data that needs to be ingested and analyzed together with historical data. This integration allows for a better understanding of the data and the detection of new knowledge (TOMIC et al., 2015; COLLARANA et al., 2017; LI et al., 2020), since the value of data increases when it can be linked and fused with other data (ANALYTICS, 2016).

Structured, semi-structured, and unstructured data, such as sensor data and any type of logs, business events, and user activities, are produced in large volume and must be processed by data streaming tools. We assume that these tools are configured in such a way as to have enough computational power to process and capture data streams.

Upon receiving data from the streaming tools, data are analyzed and stored for future analysis. Data that may not be attractive for analysis today may be important further, so it is necessary not to discard data that could generate relevant information in the future. Once ingested and stored, data needs to be analyzed continuously. Monitoring tools are used to gain insights at a very high speed through near real-time analytic dashboards.

We believe that the use of the SQL language allows users to access data from different sources, transparently, regardless of the data model of that source (files, relational or non-relational databases, etc.). Thus, monitoring tools, external APIs (Application Programming Interfaces), and users, can have easier access to different data sources through the use of this framework.

Figure 4.1 provides an overview of the framework components, defined to support continuous monitoring of data streaming, enabling integration with historical data repositories. The components are described below.

The **Data sources** component is responsible for monitoring the data produced by

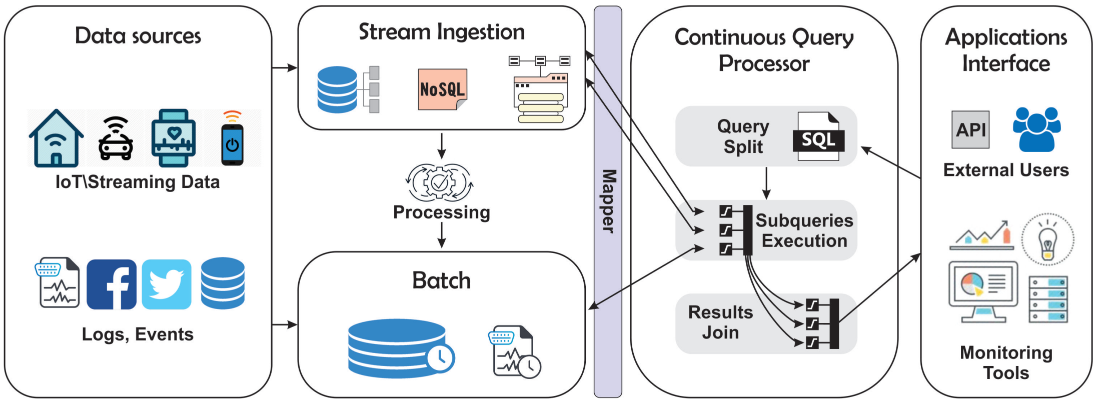


Figure 4.1: Framework Architecture

different devices, which can be sensors, IoT devices, logs, social networks, among others. In this component, two types of applications are expected: applications dedicated to ingesting raw stream data, focusing on high throughput and low latency; and applications aimed at scheduled data ingestion, in which data extraction and processing routines are performed periodically.

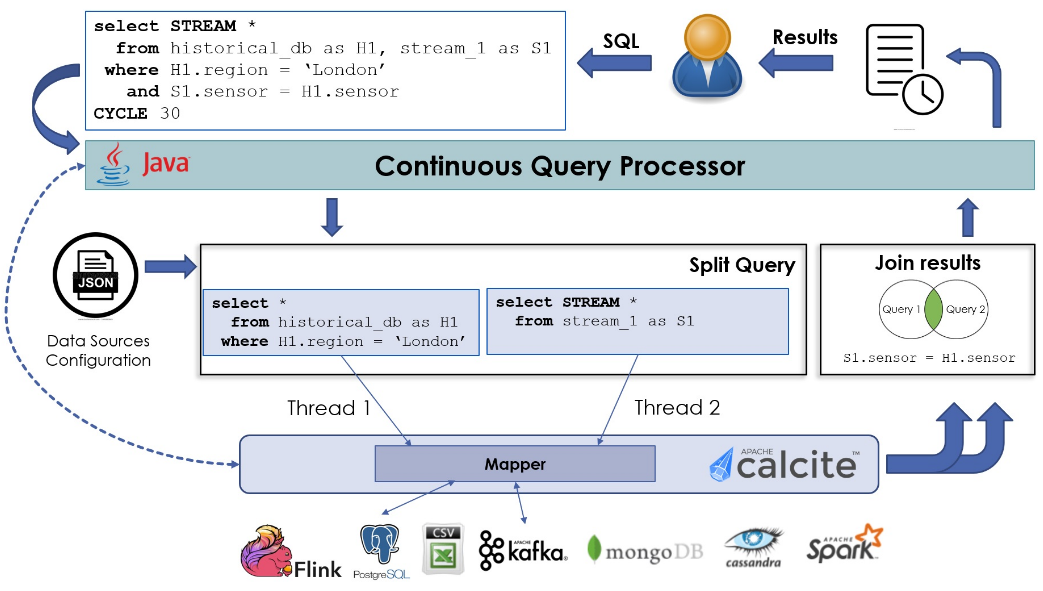
In the **Stream Ingestion** component, streaming data processing tools are configured to support applications such as Flink, Kafka, Spark, Storm, etc. These tools have several operators, such as varied windowing, join of streams, and pattern detection, being able to process and manipulate streaming data in repositories with diversified data models, respecting the defined windows for streaming processing (GAROFALAKIS; GEHRKE; RASTOGI, 2016). Windowing is the technique of executing aggregates over streams, being classified as Tumbling windows (no overlap) and Sliding windows (with overlap). Data processed in the windows are stored for further analysis.

The data processed by the Stream Ingestion component and the data ingested by schedule are stored in the **Batch** component. Dedicated repositories for storing historical data are also defined in this component. We designed these components based on the Lambda Architecture, which, in general, has three layers, represented in our framework by the components: Stream Ingestion (Speed), Batch, and Continuous Query Processor (Serving) (KIRAN et al., 2015).

In the component **Continuous Query Processor** the core of the solution is defined. SQL queries submitted to these components are pre-processed to identify the data repositories involved in the query. If it is identified that the query must be executed in more than one repository with different data models, a set of subqueries is generated. The *Mapper* receives these subqueries, transforms them to the target repository's query language, and submits them for execution. Subqueries run in parallel to optimize data fetching. The combination of the subqueries' results is done considering the criteria for creating the subqueries, respecting the filters originally defined in the main query.

In the **Application Interface** component, a set of applications can be used to consume both historical and stream data. These applications submit SQL queries and receive the query result in a tabular format, standard SQL.

In this paper, we focus on the development of the *Continuous Query Processor* component. This component was developed using the Java language and is an extension of Apache Calcite. The component has three main functions: monitor the user's interaction with the system; query validation considering the data sources configuration; and orchestrate the services communication, considering the multiple threads approach. Figure 4.2 presents a view of this component, as well as the technologies used in its development.



```
select STREAM *
  from historical_db as H1, stream_1 as S1
 where H1.region = 'London'
   and S1.sensor = H1.sensor
CYCLE 30
```

```
select *
  from historical_db as H1
 where H1.region = 'London'
```

```
select STREAM *
  from stream_1 as S1
```

```
S1.sensor = H1.sensor
```

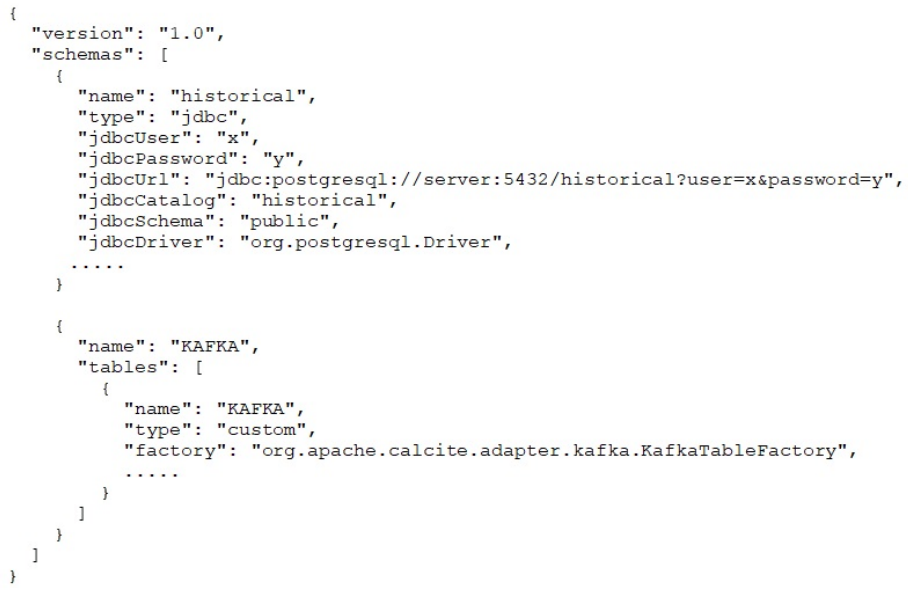
Figure 4.2: Implementation Solution

We use JSON files to configure the data sources, following the model defined in Apache Calcite, which was used as a query solve in *Mapper*. In JSON, a schema was configured for each of the three repositories used in this study: PostgreSQL, CSV, and Kafka. Thus, it was possible to identify the data sources involved in SQL queries. Part of this file can be seen in Figure 4.3, and the file used in this work is presented at *Appendix*

A. As we are using Apache Calcite, our framework is restricted to implemented adapters by it[2].

The SQL query submitted by the user is processed and, based on JSON, the subqueries are created in order to guarantee that all filters and relations refer to the same schema. Filters were characterized between specific filters and intersection filters. Specific filters are those with single-schema relations (eg, *H1.region = 'London'*). Intersection Filters, on the other hand, are those that have relations from more than one schema and, therefore, they can only be applied after executing the subqueries. In Figure 4.2 two subqueries are created, and the intersection filter *S1.sensor = H1.sensor* is not considered in this first step. A Thread is created for each subquery so that execution occurs in parallel in the component process. The mapping of each subquery is the responsibility of the Mapper, which was implemented using the conversion models defined in Apache Calcite.

Based on the same JSON files used to identify the schemas, Apache Calcite identifies the subquery's language, makes the necessary conversions, and executes the query in the appropriate repositories. The STREAM schema subqueries' are executed on the data contained in the stream window.



```
{
  "version": "1.0",
  "schemas": [
    {
      "name": "historical",
      "type": "jdbc",
      "jdbcUser": "x",
      "jdbcPassword": "y",
      "jdbcUrl": "jdbc:postgresql://server:5432/historical?user=x&password=y",
      "jdbcCatalog": "historical",
      "jdbcSchema": "public",
      "jdbcDriver": "org.postgresql.Driver",
     .....
    }

    {
      "name": "KAFKA",
      "tables": [
        {
          "name": "KAFKA",
          "type": "custom",
          "factory": "org.apache.calcite.adapter.kafka.KafkaTableFactory",
          .....
        }
      ]
    }
  ]
}
```

Figure 4.3: Part from JSON file with configuration of two schemas: Relational database (PostgreSQL) and data streaming (Kafka).

[2]https://calcite.apache.org/docs/adapter.html

Upon receiving the result of the execution of each subquery, the Continuous Query Processor component joins the results and applies the intersection filters, which were not applied due to the separation of the query in its subqueries. In the current stage of implementation, the framework does not have implementations of all join operators, only the "inner join" operator was implemented.

As it is a Continuous Query Processor, the query's result is delivered to the user/application/monitoring tool and the same query is executed again considering the execution cycle length defined in the submission of the query. In Figure 4.2, the user has defined an SQL query "**select STREAM * ... CYCLE 30**". This query is executed every 30 minutes. As data stream are limitless, this cycle remains until the user stops the execution.

5 Feasibility Study

In this section, we conduct a feasibility study, presenting real use case scenarios in which the architecture can be used to join streaming data produced by sensors in a Home Environment with historical temperature data.

A feasibility study attempts to characterize a technology to ensure that it actually does what it claims to do and is worth developing (SANTOS, 2016). As this is the first effort to implement the solution and no interface has been implemented, it has become prohibitive to apply more traditional assessment methods, such as case studies. Instead, we use the Goal-Question-Metric (GQM) methodology (CALDIERA; ROMBACH, 1994).

5.1 Datasets description

One of the datasets used in this evaluation is a sensor dataset with data collected from three residences. The layout plan of two of them is presented in Figure 5.1, with the location of each sensor identified in the floor plan.

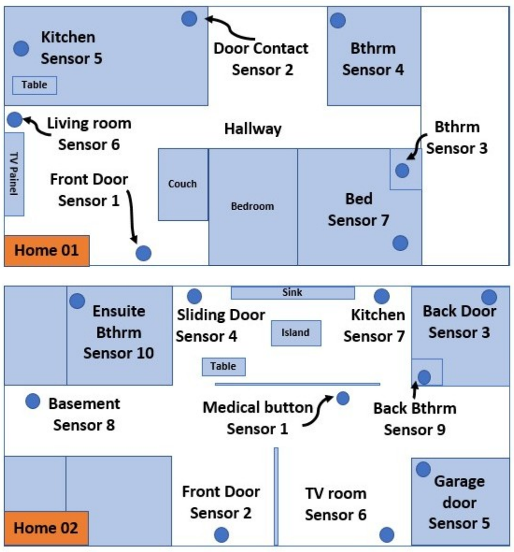


Figure 5.1: Layouts of home 01 and home 02.

The sensor data has the following features: *resident number*, *sensor number*, *message*, *day*, *month*, *year*, *hour*, *minute* and *milli-seconds*. Data were collected from these three homes over a year. Home 01 has 7 sensors and the data were monitored from 05/15/2018 to 05/31/2019, representing 381 days of data collection. Home 02 has 10 sensors and monitoring occurred from 08/09/2018 to 08/31/2019, a total of 352 days. Home 03 has 8 sensors whose data were received from 10/23/2017 to 9/30/2018, or 342 days. Note that each sensor message has an associated timestamp and that there are time gaps between consecutive sensor messages, sometimes on the order of minutes or even hours. While we could process sensor messages in actual time, the processing would require over a year of actual time. To evaluate our framework, we processed the sensor data as a continuous stream using Kafka.

Table 5.1 shows the messages sent by sensors during the monitoring period from Home 01; the other houses follow the same message pattern. We can see that it is possible to detect the action taken by the person who activated or deactivated the sensor. In general, the sensors are activated or deactivated and send messages related to these two actions.

Table 5.1: Sensors messages of Home 01

Sensor Number	**Message**
1	Front Door Contact Opened
	Front Door Contact Closed
2	Back Door Contact Opened
	Back Door Contact Closed
3	Bedroom Motion Activated
	Bedroom Motion Idle
4	Bathroom Motion Activated
	Bathroom Motion Idle
5	Kitchen Motion Activated
	Kitchen Motion Idle
6	Living Rm Motion Activated
	Living Rm Motion Idle
7	ABS Bed Sensor Occupied
	ABS Bed Sensor Briefly Vacated
	ABS Bed Sensor Vacated

The second dataset is about Historical Climate Data. Data made available by the Canadian government were used[3]. In this repository are available weather data for the period 2004-present. Using this service, collecting historical data about weather, climate data, and related information for numerous locations across Canada is possible. Some

[3]https://climate.weather.gc.ca/index_e.html

available data are temperature, precipitation, degree days, relative humidity, wind speed and direction, monthly summaries, averages, extremes, and climate norms.

We collected data for the household monitoring period (2017-2019) to enable integrating these data using their dates as a filter. A total of 3 years of data for the region of London (Ontario) were collected and stored in a PostgreSQL relational database. Based on these data, we used the framework to join the two datasets and evaluate our solution.

5.2 Joining Stream and Historical Data

Based on the data described above, we want to make queries capable of enriching the data produced by the sensors with historical data. The purpose of this scenario is: **analyze** the framework **with the purpose of** evaluating **with respect to** its usefulness and extensibility **from the point of view of** a researcher **in the context of** joining stream and historical data. We thereby derive the following questions:

Q1) Can the framework be used as a tool for joining stream and historical data described by heterogeneous data formats and models?

Q2) Is the solution extensible, that is, is it possible to add other data sources (stream or historical) in a simplified way?

In many cases, users need to correlate persistent historical data and reference data with a real-time data stream to make smarter system decisions. This type of join requires an input source for the reference/historical data to be defined. Some tools (Flink, Spark) allow the user to implement the join between Stream and Tables. However, each tool works on a language, making the integrated use of the data produced by them difficult.

Following the architectural proposal defined in Section 3, the user configures the JSON file with the schemas referring to each data source, one for the sensor stream and another for the relational database with the weather data. With this configuration, the Continuous Query Processor component is able to identify the data sources involved in the submitted query, create the sub-queries, execute them using Apache Calcite as a mapper, and present the consolidated results, providing users with a SQL-based solution.

Queries to different types of data sources, schemas and representations were submitted to the framework, involving schemas as follow:

- Historical
- Streaming
- Streaming x Streaming
- Streaming x Historical (relational data source)
- Streaming x Historical (csv file)
- Streaming x Historical (document)

For example, let's assume the user is interested in monitoring sensor readings for the Back Door Contact. Also, he wants to check weather information when the sensor detects that the door is open. In other words, he wants to execute an SQL query that brings up information that is distributed in two repositories with different data models. The following query can be submitted to the Continuous Query Processor component to consolidate this information.

Query 1:

```
SELECT STREAM Station_Name, time_lst, temp_c,
       Wind Spd_km_h, weather, dates,
       res, sensor, message
  FROM historical.historical_climate AS h,
       kafka.ambient_sensor AS s
 WHERE h.dates = s.dates
   AND s."sensor" = ‘Sensor 2‘
   AND s."message" LIKE ’%Opened%’;
```

The subqueries below are generated and submitted to Apache Calcite. The specific filters are applied in Subquery 1.1 to reduce the volume of data capture in the stream. On the other hand, the intersection filter (h.date = s.date) and the projections defined in the select clause are applied just when the component joins the historical and the stream results.

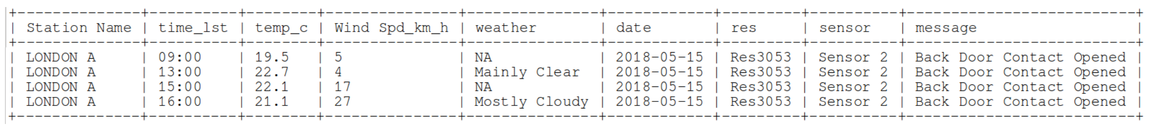
Subquery 1.1:

```
SELECT STREAM *
  FROM kafka.ambient_sensor AS s
 WHERE s."sensor" = ‘Sensor 2‘
   AND s."message" LIKE ’%Opened%’;
```

Subquery 1.2:

```
SELECT *
  FROM historical.historical_climate AS h;
```

Figure 5.2 presents the result of the query returned by the component. In this figure only part of the results is presented. As it is a stream query, the component presents the results separately, considering the data being ingested by Kafka.



```
+--------------+----------+---------+---------------+----------------+------------+---------+----------+---------------------------+
| Station Name | time_lst | temp_c  | Wind Spd_km_h | weather        | date       | res     | sensor   | message                   |
+--------------+----------+---------+---------------+----------------+------------+---------+----------+---------------------------+
| LONDON A     | 09:00    | 19.5    | 5             | NA             | 2018-05-15 | Res3053 | Sensor 2 | Back Door Contact Opened |
| LONDON A     | 13:00    | 22.7    | 4             | Mainly Clear   | 2018-05-15 | Res3053 | Sensor 2 | Back Door Contact Opened |
| LONDON A     | 15:00    | 22.1    | 17            | NA             | 2018-05-15 | Res3053 | Sensor 2 | Back Door Contact Opened |
| LONDON A     | 16:00    | 21.1    | 27            | Mostly Cloudy  | 2018-05-15 | Res3053 | Sensor 2 | Back Door Contact Opened |
+--------------+----------+---------+---------------+----------------+------------+---------+----------+---------------------------+
```

Figure 5.2: One of the results returned by the Continuous Query Processor component.(Query 1)

In addition, other schemas can be configured, allowing users to monitor data (history and stream) from heterogeneous repositories without the need for knowledge of specific languages. This reinforces the utility appeal of the solution since users can consume the data without directly interfacing with storage solutions.

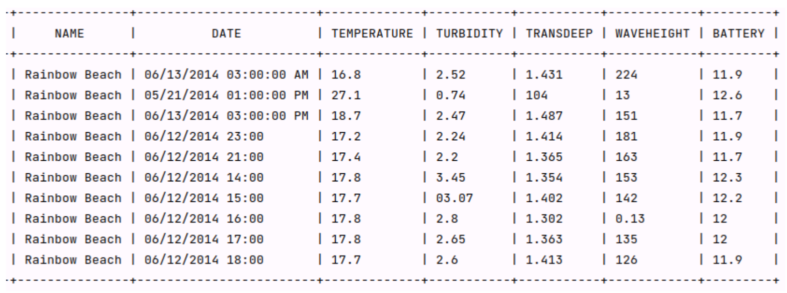
In order to test the above, two relational datasets were added, being one synthetic source data and another containing data from beaches sensors, also in Canada. Some of the avaiable data are temperature, beach name, turbidity and wave height. Another streaming dataset was also added to the context, containing data from sensors with informations about air quality.

The Query 2 presents a simple query on historical dataset only.

Query 2:

```
SELECT name, date, temperature,
         turbidity, transdeep, waveheight, battery
  FROM historical.historical_beach AS hb
 WHERE hb.historical_beach = ’Rainbow␣Beach’;
```

Once the main query only involves a historical dataset, there is no subqueries generated. The Figure 5.3 shows part of the results.



```
+----------------+-------------------------+-------------+-----------+-----------+------------+---------+
|     NAME       |          DATE           | TEMPERATURE | TURBIDITY | TRANSDEEP | WAVEHEIGHT | BATTERY |
+----------------+-------------------------+-------------+-----------+-----------+------------+---------+
| Rainbow Beach | 06/13/2014 03:00:00 AM | 16.8        | 2.52      | 1.431     | 224        | 11.9    |
| Rainbow Beach | 05/21/2014 01:00:00 PM | 27.1        | 0.74      | 104       | 13         | 12.6    |
| Rainbow Beach | 06/13/2014 03:00:00 PM | 18.7        | 2.47      | 1.487     | 151        | 11.7    |
| Rainbow Beach | 06/12/2014 23:00       | 17.2        | 2.24      | 1.414     | 181        | 11.9    |
| Rainbow Beach | 06/12/2014 21:00       | 17.4        | 2.2       | 1.365     | 163        | 11.7    |
| Rainbow Beach | 06/12/2014 14:00       | 17.8        | 3.45      | 1.354     | 153        | 12.3    |
| Rainbow Beach | 06/12/2014 15:00       | 17.7        | 03.07     | 1.402     | 142        | 12.2    |
| Rainbow Beach | 06/12/2014 16:00       | 17.8        | 2.8       | 1.302     | 0.13       | 12      |
| Rainbow Beach | 06/12/2014 17:00       | 17.8        | 2.65      | 1.363     | 135        | 12      |
| Rainbow Beach | 06/12/2014 18:00       | 17.7        | 2.6       | 1.413     | 126        | 11.9    |
+----------------+-------------------------+-------------+-----------+-----------+------------+---------+
```

Figure 5.3: Results returned by the Continuous Query Processor component on historical dataset.(Query 2)

The Query 3 presents a join between an historical(historical_beach) and a stream(stream_air) datasets. Also includes a filter by date in the historical dataset.

Query 3:

```
SELECT STREAM *
  FROM historical.historical_beach AS hb,
       kafka.stream_air AS sa
 WHERE hb.DATE = sa.DATE
   AND hb.DATE = '05-28-2014';
```

The following subqueries are generated.

Subquery 3.1:

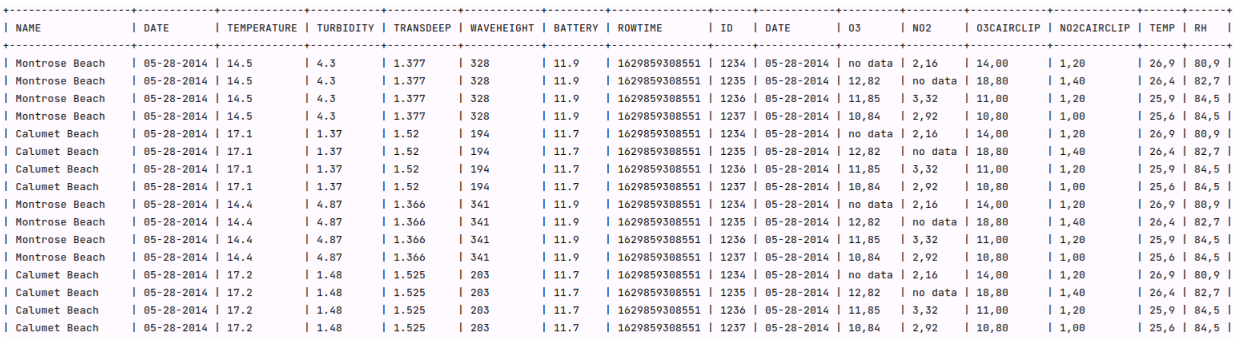
```
SELECT *
  FROM historical.historical_beach AS hb
 WHERE hb.DATE = '05-28-2014';
```

Subquery 3.2:

```
SELECT STREAM*
  FROM kafka.stream_air AS sa;
```

The Figure 5.4 shows part of the results.

The Query 4 involves joins beetween two historical datasets(historical_beach and historical_climate) and one stream dataset (stream_air) and a filter in the stream dataset.



```
+-------------------+------------+-------------+-----------+-----------+------------+---------+---------------+------+------------+---------+---------+------------+-------------+------+------+
| NAME              | DATE       | TEMPERATURE | TURBIDITY | TRANSDEEP | WAVEHEIGHT | BATTERY | ROWTIME       | ID   | DATE       | O3      | NO2     | O3CAIRCLIP | NO2CAIRCLIP | TEMP | RH   |
+-------------------+------------+-------------+-----------+-----------+------------+---------+---------------+------+------------+---------+---------+------------+-------------+------+------+
| Montrose Beach    | 05-28-2014 | 14.5        | 4.3       | 1.377     | 328        | 11.9    | 1629859308551 | 1234 | 05-28-2014 | no data | 2,16    | 14,00      | 1,20        | 26,9 | 80,9 |
| Montrose Beach    | 05-28-2014 | 14.5        | 4.3       | 1.377     | 328        | 11.9    | 1629859308551 | 1235 | 05-28-2014 | 12,82   | no data | 18,80      | 1,40        | 26,4 | 82,7 |
| Montrose Beach    | 05-28-2014 | 14.5        | 4.3       | 1.377     | 328        | 11.9    | 1629859308551 | 1236 | 05-28-2014 | 11,85   | 3,32    | 11,00      | 1,20        | 25,9 | 84,5 |
| Montrose Beach    | 05-28-2014 | 14.5        | 4.3       | 1.377     | 328        | 11.9    | 1629859308551 | 1237 | 05-28-2014 | 10,84   | 2,92    | 10,80      | 1,00        | 25,6 | 84,5 |
| Calumet Beach     | 05-28-2014 | 17.1        | 1.37      | 1.52      | 194        | 11.7    | 1629859308551 | 1234 | 05-28-2014 | no data | 2,16    | 14,00      | 1,20        | 26,9 | 80,9 |
| Calumet Beach     | 05-28-2014 | 17.1        | 1.37      | 1.52      | 194        | 11.7    | 1629859308551 | 1235 | 05-28-2014 | 12,82   | no data | 18,80      | 1,40        | 26,4 | 82,7 |
| Calumet Beach     | 05-28-2014 | 17.1        | 1.37      | 1.52      | 194        | 11.7    | 1629859308551 | 1236 | 05-28-2014 | 11,85   | 3,32    | 11,00      | 1,20        | 25,9 | 84,5 |
| Calumet Beach     | 05-28-2014 | 17.1        | 1.37      | 1.52      | 194        | 11.7    | 1629859308551 | 1237 | 05-28-2014 | 10,84   | 2,92    | 10,80      | 1,00        | 25,6 | 84,5 |
| Montrose Beach    | 05-28-2014 | 14.4        | 4.87      | 1.366     | 341        | 11.9    | 1629859308551 | 1234 | 05-28-2014 | no data | 2,16    | 14,00      | 1,20        | 26,9 | 80,9 |
| Montrose Beach    | 05-28-2014 | 14.4        | 4.87      | 1.366     | 341        | 11.9    | 1629859308551 | 1235 | 05-28-2014 | 12,82   | no data | 18,80      | 1,40        | 26,4 | 82,7 |
| Montrose Beach    | 05-28-2014 | 14.4        | 4.87      | 1.366     | 341        | 11.9    | 1629859308551 | 1236 | 05-28-2014 | 11,85   | 3,32    | 11,00      | 1,20        | 25,9 | 84,5 |
| Montrose Beach    | 05-28-2014 | 14.4        | 4.87      | 1.366     | 341        | 11.9    | 1629859308551 | 1237 | 05-28-2014 | 10,84   | 2,92    | 10,80      | 1,00        | 25,6 | 84,5 |
| Calumet Beach     | 05-28-2014 | 17.2        | 1.48      | 1.525     | 203        | 11.7    | 1629859308551 | 1234 | 05-28-2014 | no data | 2,16    | 14,00      | 1,20        | 26,9 | 80,9 |
| Calumet Beach     | 05-28-2014 | 17.2        | 1.48      | 1.525     | 203        | 11.7    | 1629859308551 | 1235 | 05-28-2014 | 12,82   | no data | 18,80      | 1,40        | 26,4 | 82,7 |
| Calumet Beach     | 05-28-2014 | 17.2        | 1.48      | 1.525     | 203        | 11.7    | 1629859308551 | 1236 | 05-28-2014 | 11,85   | 3,32    | 11,00      | 1,20        | 25,9 | 84,5 |
| Calumet Beach     | 05-28-2014 | 17.2        | 1.48      | 1.525     | 203        | 11.7    | 1629859308551 | 1237 | 05-28-2014 | 10,84   | 2,92    | 10,80      | 1,00        | 25,6 | 84,5 |
```

Figure 5.4: Results returned by the Continuous Query Processor on simple join on date.(Query 3)

Query 4:

```
SELECT STREAM *
  FROM historical.historical_beach AS hb,
       historical.historical_climate AS hc
       kafka.stream_air AS ka,
 WHERE hb.DATE = sa.DATE
   AND hc.DATE = sa.DATE
   AND ka."DATE" = '05-28-2014';
```

And the following subqueries are generated.

Subquery 4.1:

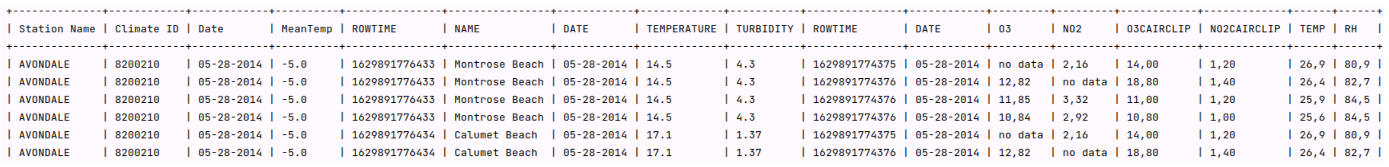
```
SELECT *
  FROM historical.historical_beach AS hb;
```

Subquery 4.2:

```
SELECT STREAM *
  FROM historical.historical_climate AS hc;
```

Subquery 4.3:

```
SELECT STREAM *
  FROM kafka.stream_air AS ka
 WHERE ka."DATE" = '05-28-2014';
```



```
+--------------+------------+------------+----------+---------------+----------------+------------+-------------+-----------+---------------+------------+---------+---------+------------+-------------+------+------+
| Station Name | Climate ID | Date       | MeanTemp | ROWTIME       | NAME           | DATE       | TEMPERATURE | TURBIDITY | ROWTIME       | DATE       | O3      | NO2     | O3CAIRCLIP | NO2CAIRCLIP | TEMP | RH   |
+--------------+------------+------------+----------+---------------+----------------+------------+-------------+-----------+---------------+------------+---------+---------+------------+-------------+------+------+
| AVONDALE     | 8200210    | 05-28-2014 | -5.0     | 1629891776433 | Montrose Beach | 05-28-2014 | 14.5        | 4.3       | 1629891774375 | 05-28-2014 | no data | 2,16    | 14,00      | 1,20        | 26,9 | 80,9 |
| AVONDALE     | 8200210    | 05-28-2014 | -5.0     | 1629891776433 | Montrose Beach | 05-28-2014 | 14.5        | 4.3       | 1629891774376 | 05-28-2014 | 12,82   | no data | 18,80      | 1,40        | 26,4 | 82,7 |
| AVONDALE     | 8200210    | 05-28-2014 | -5.0     | 1629891776433 | Montrose Beach | 05-28-2014 | 14.5        | 4.3       | 1629891774376 | 05-28-2014 | 11,85   | 3,32    | 11,00      | 1,20        | 25,9 | 84,5 |
| AVONDALE     | 8200210    | 05-28-2014 | -5.0     | 1629891776433 | Montrose Beach | 05-28-2014 | 14.5        | 4.3       | 1629891774376 | 05-28-2014 | 10,84   | 2,92    | 10,80      | 1,00        | 25,6 | 84,5 |
| AVONDALE     | 8200210    | 05-28-2014 | -5.0     | 1629891776434 | Calumet Beach  | 05-28-2014 | 17.1        | 1.37      | 1629891774375 | 05-28-2014 | no data | 2,16    | 14,00      | 1,20        | 26,9 | 80,9 |
| AVONDALE     | 8200210    | 05-28-2014 | -5.0     | 1629891776434 | Calumet Beach  | 05-28-2014 | 17.1        | 1.37      | 1629891774376 | 05-28-2014 | 12,82   | no data | 18,80      | 1,40        | 26,4 | 82,7 |
```

Figure 5.5: Results returned by the Continuous Query Processor on multiple join on date.(Query 4)

Finally, although not all SQL operations are implemented in this version, it can be used in scenarios where it is necessary to join data by equality criteria, that is, using an inner join.

With this project, both location and access transparency are provided (COULOURIS et al., 2005), freeing users from the need to understand the underlying storage solution and how data is organized in this distributed environment. In addition, this conceptual design also offers the option of using the SQL language for queries, enabling its use by most database users and by most of the tools used for data monitoring (e.g., dashboards).

This way, given the scenario at hand the elements that compose it, we can answer the questions previously outlined. **(Q1) Can the framework be used as a tool for joining stream and historical data described by heterogeneous data formats and models?** *Partly.* We demonstrate in this scenario that we can query stream and historical data. By automating the process of split the main query and execute the sub-queries in parallel using Apache Calcite as mapper. Via the query endpoints, users can interface with the solution in a transparent manner, leaving to the Continuous Query Processor component the interpretation and proper redirection of incoming queries. However, the current implementation does not support all SQL operations, because of that we are answering this question partly.

(Q2) Is the solution extensible, that is, is it possible to add other data sources (stream or historical) in a simplified way? *Yes.* In the analyzed scenario, we show that by using the Apache Calcite and JSON configuration files, new storage solutions (schemas) can be added, exempting the need to restructure the solution. This way, we show the framework's viability for this scenario, since it can contemplate the goals previously outlined.

6 Final Remarks and Future Works

This work proposed a framework that enables the monitoring of data produced by IoT devices and sensors and its integration with historical data. The proposed solution is based on the SQL language and seeks to facilitate access to and the use of distributed data repositories with different data models. Furthermore, users can use the framework to enrich data produced by IoT devices and sensors by integrating them with historical databases.

The framework was developed as an Apache Calcite extension, using JSON files to configure the data sources. With this, the framework can detect which data sources are involved in the query, create the subqueries and execute them in parallel, with Apache Calcite working as a mapper. Finally, the results of the subqueries are joined, respecting the intersection filters (inter-queries filters).

Real data produced by sensors in assisted home environments and time data were used in the feasibility study conducted to evaluate the proposed solution. The framework integrated this heterogeneous data in a non-intrusive way and allowed the user to access the data by submitting a single query, thus enabling a comprehensive analysis of historical and stream data in a unified system. The results obtained with the evaluation and the answers to the proposed research questions allow us to conclude that the developed framework achieved the objectives of this work.

As future work, once so far the framework does not allow all join operations to be applied on the integrated results from different repositories, we plan to continue with the development of other SQL operations and also incorporate elements for manipulating data stream windows. Although Apache Calcite adapters limit our solution, many initiatives are underway to expand the range of data models supported by Apache Calcite[4]. Given the complexity of streaming processing, we intend to evaluate real-time requirements (i.e., low latency, high throughput, scalability, and fault tolerance). Finally, we plan to measure the overhead generated by the framework for executing the queries since the monitoring

[4]https://calcite.apache.org/docs/powered_by.html

tools are almost real-time.

Bibliography

ABU-SALIH, B. et al. Social big data: An overview and applications. *Social Big Data Analytics: Practices, Techniques, and Applications*, Springer, p. 1–14, 2021.

AKANBI, A.; MASINDE, M. A distributed stream processing middleware framework for real-time analysis of heterogeneous data on big data platform: Case of environmental monitoring. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 20, n. 11, p. 3166, 2020.

ALKHAMISI, A. O.; SALEH, M. Ontology opportunities and challenges: Discussions from semantic data integration perspectives. In: IEEE. *2020 6th Conference on Data Science and Machine Learning Applications (CDMA)*. [S.l.], 2020. p. 134–140.

ANALYTICS, M. The age of analytics: competing in a data-driven world. *McKinsey Global Institute Research*, 2016.

ASANO, Y. et al. Flexible framework for data integration and update propagation: System aspect. In: *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*. [S.l.: s.n.], 2019. p. 1–5.

BABCOCK, B. et al. Models and issues in data stream systems. In: *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. [S.l.: s.n.], 2002. p. 1–16.

BARROS, M. *Book Review: Digital Objects, Digital Subjects: Interdisciplinary perspectives on capitalism, labour and politics in the age of big data*. [S.l.]: SAGE Publications Sage UK: London, England, 2020.

BROWN, K. S.; SPIVAK, D. I.; WISNESKY, R. Categorical data integration for computational science. *Computational Materials Science*, Elsevier, v. 164, p. 127–132, 2019.

CALDIERA, V. R. B.-G.; ROMBACH, H. D. Goal question metric paradigm. *Encyclopedia of software engineering*, v. 1, p. 528–532, 1994.

CAPPUZZO, R.; PAPOTTI, P.; THIRUMURUGANATHAN, S. Creating embeddings of heterogeneous relational datasets for data integration tasks. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. [S.l.: s.n.], 2020. p. 1335–1349.

CAVALLO, G. et al. Contextually-enriched querying of integrated data sources. In: IEEE. *2018 IEEE 34th International Conference on Data Engineering Workshops (ICDEW)*. [S.l.], 2018. p. 9–16.

CAVANILLAS, J. M.; CURRY, E.; WAHLSTER, W. *New horizons for a data-driven economy: a roadmap for usage and exploitation of big data in Europe*. [S.l.]: Springer Nature, 2016.

CHEN, J. et al. Big data challenge: a data management perspective. *Frontiers of computer Science*, Springer, v. 7, n. 2, p. 157–164, 2013.

COLLARANA, D. et al. Semantic data integration for knowledge graph construction at query time. In: IEEE. *2017 IEEE 11th International Conference on Semantic Computing (ICSC)*. [S.l.], 2017. p. 109–116.

COSTA, F. S. et al. Fasten iiot: An open real-time platform for vertical, horizontal and end-to-end integration. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 20, n. 19, p. 5499, 2020.

COULOURIS, G. et al. *Distributed Systems: Concepts and Design*. 5th. ed. [S.l.]: Pearson Education, 2005.

FREITAS, A.; CURRY, E. Natural language queries over heterogeneous linked data graphs: A distributional-compositional semantics approach. In: *Proceedings of the 19th international conference on Intelligent User Interfaces*. [S.l.: s.n.], 2014. p. 279–288.

GAMA, J. A survey on learning from data streams: current and future trends. *Progress in Artificial Intelligence*, Springer, v. 1, n. 1, p. 45–55, 2012.

GAMA, J.; GABER, M. M. *Learning from data streams: processing techniques in sensor networks*. [S.l.]: Springer, 2007.

GAROFALAKIS, M.; GEHRKE, J.; RASTOGI, R. (Ed.). *Data Stream Management*. Springer Berlin Heidelberg, 2016. Disponível em: ⟨https://doi.org/10.1007/978-3-540-28608-0⟩.

GHASEMAGHAEI, M.; CALIC, G. Assessing the impact of big data on firm innovation performance: Big data is not always better data. *Journal of Business Research*, Elsevier, v. 108, p. 147–162, 2020.

GOLAB, L.; ÖZSU, M. T. Issues in data stream management. *ACM Sigmod Record*, ACM New York, NY, USA, v. 32, n. 2, p. 5–14, 2003.

GURUSAMY, V.; KANNAN, S.; NANDHINI, K. The real time big data processing framework: Advantages and limitations. *International Journal of Computer Sciences and Engineering*, v. 5, n. 12, p. 305–312, 2017.

JIANG, Q.; CHAKRAVARTHY, S. *Stream data processing: a quality of service perspective*. [S.l.: s.n.], 2009.

KIRAN, M. et al. Lambda architecture for cost-effective batch and speed big data processing. In: IEEE. *2015 IEEE International Conference on Big Data (Big Data)*. [S.l.], 2015. p. 2785–2792.

KOLAJO, T.; DARAMOLA, O.; ADEBIYI, A. Big data stream analysis: a systematic literature review. *Journal of Big Data*, Springer, v. 6, n. 1, p. 1–30, 2019.

LEE, I.; LEE, K. The internet of things (iot): Applications, investments, and challenges for enterprises. *Business Horizons*, Elsevier, v. 58, n. 4, p. 431–440, 2015.

LI, R. et al. Ontologies-based domain knowledge modeling and heterogeneous sensor data integration for bridge health monitoring systems. *IEEE Transactions on Industrial Informatics*, IEEE, v. 17, n. 1, p. 321–332, 2020.

MIKALEF, P. et al. *Big data and business analytics: A research agenda for realizing business value*. [S.l.]: Elsevier, 2020.

MILLER, R. J. Open data integration. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 11, n. 12, p. 2130–2139, 2018.

SAKR, S. et al. Big data processing systems: state-of-the-art and open challenges. In: IEEE. *2015 International Conference on Cloud Computing (ICCC)*. [S.l.], 2015. p. 1–8.

SANTOS, R. P. dos. *Managing and monitoring software ecosystem to support demand and solution analysis*. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 2016.

SHAN, S. et al. Big data analysis adaptation and enterprises' competitive advantages: the perspective of dynamic capability and resource-based theories. *Technology Analysis & Strategic Management*, Taylor & Francis, v. 31, n. 4, p. 406–420, 2019.

SHEIN, A.; CHRYSANTHIS, P. K. Multi-query optimization of incrementally evaluated sliding-window aggregations. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, 2020.

SMYS, S. A survey on internet of things (iot) based smart systems. *Journal of ISMAC*, v. 2, n. 04, p. 181–189, 2020.

STONEBRAKER, M.; ILYAS, I. F. Data integration: The current status and the way forward. *IEEE Data Eng. Bull.*, v. 41, n. 2, p. 3–9, 2018.

STOYANOVA, M. et al. A survey on the internet of things (iot) forensics: challenges, approaches, and open issues. *IEEE Communications Surveys & Tutorials*, IEEE, v. 22, n. 2, p. 1191–1221, 2020.

TATBUL, N. Streaming data integration: Challenges and opportunities. In: IEEE. *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*. [S.l.], 2010. p. 155–158.

TIAN, A.; SEQUEDA, J. F.; MIRANKER, D. P. Qodi: Query as context in automatic data integration. In: SPRINGER. *International Semantic Web Conference*. [S.l.], 2013. p. 624–639.

TIBCO. *What is Data Streaming?* 2019. Acesso em 19 de agosto de 2021. Disponível em: ⟨https://www.tibco.com/reference-center/what-is-data-streaming⟩.

TOMAN, S. H. The design of a templating language to embed database queries into documents. *Journal of Education College Wasit University*, v. 1, n. 29, p. 512–534, 2017.

TOMIC, D. et al. Experiences with creating a precision dairy farming ontology (dfo) and a knowledge graph for the data integration platform in agriopenlink. *Agrárinformatika/Journal of Agricultural Informatics*, Hungarian Association of Agricultural Informatics, v. 6, n. 4, p. 115–126, 2015.

TU, D. Q. et al. Iot streaming data integration from multiple sources. *Computing*, Springer, v. 102, n. 10, p. 2299–2329, 2020.

VENKATESH, K. et al. Challenges and research disputes and tools in big data analytics. *International Journal of Engineering and Advanced Technology*, v. 6, p. 1949–1952, 2019.

WANG, J. et al. Big data service architecture: a survey. *Journal of Internet Technology*, v. 21, n. 2, p. 393–405, 2020.

WANG, X.; HAAS, L.; MELIOU, A. Explaining data integration. *Data Engineering Bulletin*, v. 41, n. 2, 2018.

YI, X. et al. Building a network highway for big data: architecture and challenges. *Ieee Network*, IEEE, v. 28, n. 4, p. 5–13, 2014.

YOUSFI, S.; RHANOUI, M.; CHIADMI, D. Towards a generic multimodal architecture for batch and streaming big data integration. *arXiv preprint arXiv:2108.04343*, 2021.

ZANELLA, A. et al. Internet of things for smart cities. *IEEE Internet of Things journal*, IEEE, v. 1, n. 1, p. 22–32, 2014.

Appendices

A - JSON Configuration File

```
{
  "version": "1.0",
  "schemas": [
    {
      "name": "historical",
      "type": "jdbc",
      "jdbcUser": "x",
      "jdbcPassword": "y",
      "jdbcUrl":
         ↪ "jdbc:postgresql://server:5432/historical?user=x7password=y",
      "jdbcCatalog": "historical_climate",
      "jdbcSchema": "public",
      "jdbcDriver": "org.postgresql.Driver",
      "materializations": [
                  {
                    "view": "historical_climate",
                    "table": "historical_climate"
                  }
        ]
   },
   {
      "name": "historical2",
      "type": "jdbc",
      "jdbcUser": "x",
      "jdbcPassword": "y",
      "jdbcUrl":
         ↪ "jdbc:postgresql://server:5432/historical?user=x7password=y",
      "jdbcCatalog": "historical_beach",
      "jdbcSchema": "public",
      "jdbcDriver": "org.postgresql.Driver",
      "materializations": [
                  {
                    "view": "historical_beach",
                    "table": "historical_beach"
                  }
        ]
   },
   {
      "name": "KAFKA",
      "tables": [
        {
          "name": "KAFKA",
          "type": "custom",
          "factory":
             ↪ "org.apache.calcite.adapter.kafka.KafkaTableFactory",
          "stream": {
            "stream": true
          },
          "operand": {
            "bootstrap.servers": "server:9092",
            "topic.name": "kafka.ambient_sensor",
        }
      ]
    },
   {
      "name": "KAFKA2",
      "tables": [
        {
          "name": "stream_air",
          "type": "custom",
          "factory":
            ↪ "org.apache.calcite.adapter.kafka.KafkaTableFactory",
          "stream": {
            "stream": true
          },
          "operand": {
            "bootstrap.servers": "server:9092",
            "topic.name": "kafka.stream_air",
        }
      ]
    }
  ]
}
```