Data Ingestion and Storage Strategies for Data Warehouses in the Context of Data Streaming: An Overview of Recent Advances

Alexandre de Queiroz Burle (aqb@cin.ufpe.br)

Federal University of Pernambuco
secgrad@cin.ufpe.br
www.cin.ufpe.br/~graduacao

Recife
2024

Alexandre de Queiroz Burle (aqb@cin.ufpe.br)

Data Ingestion and Storage Strategies for Data Warehouses in the Context of Data Streaming: An Overview of Recent Advances

A B.Sc. Dissertation presented to the Center of Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering.

Concentration Area: *Databases*
Advisor: *Robson do Nascimento Fidalgo (rdnf@cin.ufpe.br)*

Recife
2024

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Burle, Alexandre de Queiroz.
Data ingestion and storage strategies for data warehouses in the context of data streaming: an overview of recent advances / Alexandre de Queiroz Burle. - Recife, 2024.
53 p. : il., tab.

Orientador(a): Robson do Nascimento Fidalgo
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Engenharia da Computação - Bacharelado, 2024.
Inclui referências.

1. Data Warehouse. 2. Data Stream. 3. ETL. 4. Join. I. Fidalgo, Robson do Nascimento. (Orientação). II. Título.

000 CDD (22.ed.)

Alexandre de Queiroz Burle

Data Ingestion and Storage Strategies for Data Warehouses in the Context of Data Streaming: An Overview of Recent Advances

Trabalho de Conclusão de Curso apresentado como pré-requisito para conclusão do Curso de Bacharelado em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco. Defendida e aprovada em 17 de Outubro de 2024 pela seguinte banca examinadora:

Robson do Nascimento Fidalgo
Orientador/CIn-UFPE

Luciano de Andrade Barbosa
Examinador/CIn-UFPE

ACKNOWLEDGEMENTS

First and foremost, I want to thank my family. They have been present through my graduation, helping me in every way I needed to make sure I would make the most out of it. Namely, I would like to thank my mother, Candida, my father, Mario, and my sister, Isabela.

I would also like to thank my graduation colleagues Danilo Vaz, Humberto Lopes, Matheus Teotonio, Rodrigo Duarte, and Uanderson Souza, who ended up becoming great friends of mine. They helped me with group projects and studying for tests and promoted my best moments during my bachelor's degree, even outside the campus.

I want to thank PET Informática for promoting the Petlab program, which was one of the first extracurricular activities that I was able to participate in.

I want to thank the Voxar Labs and everyone involved, where I learned much about research and got my first internship. It was the place that kick-started my life as a developer.

Last, I want to thank Centro de Informática and everyone involved for promoting an excellent learning environment and many opportunities.

ABSTRACT

In the current landscape of data-driven decision-making, data warehouses have proven to be highly valuable tools, especially with the emergence of big data characterized by its volume, velocity, and variety. This study provides a systematic review of data ingestion and storage strategies for data warehouses in the context of data streaming, focusing on the latest advancements and methodologies to address the challenges posed by continuous data streams. Modeling schemas of data warehouses are not the focus of recent studies, as existing schemas seem to satisfy current needs. Instead, the focus has shifted towards optimizing operational processes like ETL (Extract, Transform, Load) and join operations. The review highlights techniques such as parallel processing, in-memory computing, and distributed computing as critical to enhancing data ingestion and storage capabilities. This work synthesizes recent research, providing insights into how modern data warehouses can efficiently process and store streaming data to support real-time analytics and decision-making. The findings offer valuable guidance for developing scalable and efficient data warehousing solutions.

Keywords: Data Warehouse; Data Streaming; ETL; Data Ingestion; Data Storage

RESUMO

No cenário atual de tomada de decisões orientada por dados, os data warehouses têm se mostrado ferramentas altamente valiosas, especialmente com o surgimento do big data, caracterizado por seu volume, velocidade e variedade. Este estudo fornece uma revisão sistemática das estratégias de ingestão e armazenamento de dados para data warehouses no contexto de data streaming, focando nos avanços e metodologias mais recentes para enfrentar os desafios impostos pelos fluxos contínuos de dados. Os esquemas de modelagem dos data warehouses não são o foco dos estudos recentes, pois os esquemas existentes parecem satisfazer as necessidades atuais. Em vez disso, o foco tem se voltado para a otimização de processos operacionais como ETL (Extract, Transform, Load) e operações de join. A revisão destaca técnicas como processamento paralelo, computação em memória e computação distribuída como críticas para melhorar as capacidades de ingestão e armazenamento de dados. Este trabalho sintetiza pesquisas recentes, fornecendo insights sobre como os data warehouses modernos podem processar e armazenar dados em streaming de forma eficiente para apoiar análises e tomadas de decisão em tempo real. As conclusões oferecem orientações valiosas para o desenvolvimento de soluções de data warehousing escaláveis e eficientes.

Palavras-chave: Armazém de Dados; Transmissão de Dados; ETL; Ingestão de Dados; Armazenamento de Dados

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1

INTRODUCTION

In recent years, the importance of data warehouses has grown significantly as organizations increasingly rely on data-driven decision-making to maintain a competitive edge [10]. A data warehouse (DW) serves as a centralized repository for structured data, enabling efficient query processing, reporting, and analysis. This technology supports Business Intelligence (BI) systems by providing a unified and consistent view of an organization's data, facilitating the generation of reports, dashboards, and in-depth analyses [15].

The emergence of big data has further highlighted the necessity of robust data warehousing solutions. Big data is characterized by its volume, variety, and velocity [7], and refers to exceptionally large and complex datasets that exceed the capabilities of traditional data management systems to store, process, and analyze. These characteristics pose unique challenges, requiring technologies like distributed computing and parallel processing. Big data encompasses structured data, often found in data warehouses, and unstructured or semi-structured data, which may reside in data lakes. However, while traditional data warehouses are effective for structured data, they often need help handling the high-velocity data streams and the diverse formats associated with big data. This has led to the development of new strategies and technologies that the performance and scalability of data warehouses in the context of big data [23].

Data streaming, a key component of big data, involves continuously ingesting and processing data. This approach contrasts with the traditional usage of big batches of data processing, which collects and processes data at intervals that may not be suited for critical applications. Data streaming is essential for applications requiring immediate insights, such as fraud detection, real-time analytics, and monitoring systems [16, 24]. Hence, integrating data streaming with data warehousing has driven the evolution of traditional DW to support high-velocity data and the adoption of new architectural paradigms.

Recent advancements in data warehousing focus on addressing the challenges posed by data streaming. Techniques such as parallel processing, in-memory computing, and distributed computing have been employed to enhance data ingestion and storage capabilities. These approaches ensure that data warehouses can efficiently process and store high-velocity data streams, providing timely insights and supporting data-driven decision-making [42, 17].

This work presents a comprehensive review of data ingestion and storage strategies for

data warehouses in the context of data streaming. It aims to provide insights into the latest research and technologies that address the operational challenges of handling continuous data streams. Chapter 2 presents a theoretical foundation covering fundamental concepts and schemas used in data warehousing. Chapter 2.6 reviews related work. Chapter 3 outlines the proposed approach, detailing the research questions, search strategy, and selection process. Chapter 4.5 discusses the results, analyzing the selected studies to answer the research questions. Finally, Chapter 5 concludes the paper, summarizing the findings.

2

THEORETICAL FOUNDATION

2.1 DATA WAREHOUSE

A data warehouse (DW) is a repository that stores data from diverse business sources. It serves as the cornerstone for Business Intelligence (BI) systems, providing a comprehensive and unified view of an organization's data. The DW supports the generation of reports, dashboards, and in-depth data analysis, enabling data-driven decision-making.

The DW typically stores current and historical data, allowing for trend analysis over time. This historical perspective is invaluable for identifying business performance patterns, trends, and anomalies. By integrating data from multiple sources, the DW ensures that users can access consistent and reliable information.

Before data is stored in the DW, it undergoes either Extract, Transform, Load (ETL) or Extract, Load, Transform (ELT) processes (see section 2.3). Figure 1 illustrates the layers of a DW, including the ETL process.

The DW enables users to perform complex ad-hoc queries with minimal latency, providing insights in a timely manner. Its ability to store and manage vast volumes of data, often reaching the petabyte scale, makes the DW an indispensable tool for BI initiatives.

2.2 STAR SCHEMA

The star schema is a widely adopted and practical modeling approach for data warehouses [3]. It organizes multidimensional data into a central fact table surrounded by multiple dimension tables, forming a star-like structure (see Figure 2).

The fact table is the central component of the star schema and stores the quantitative measures or metrics of the analyzed business process. These metrics, typically numeric values such as sales amounts, quantities, or costs, are essential for decision-making. The fact table also contains foreign keys that reference the primary keys of the dimension tables, establishing relationships between facts and their context.

Dimension tables provide the descriptive context for the facts. Each dimension table represents a business entity, such as products, customers, or time, and contains attributes that

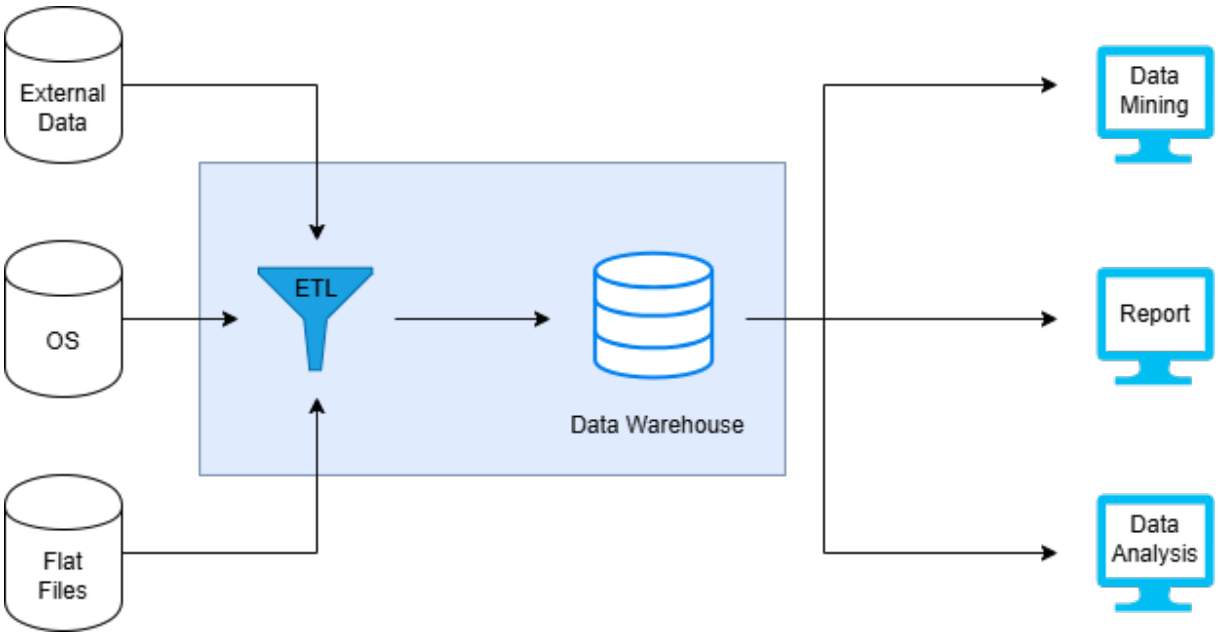

Figure 1: The layers of a data warehouse. The blue rectangle represents the focus of this work: data ingestion and storage.

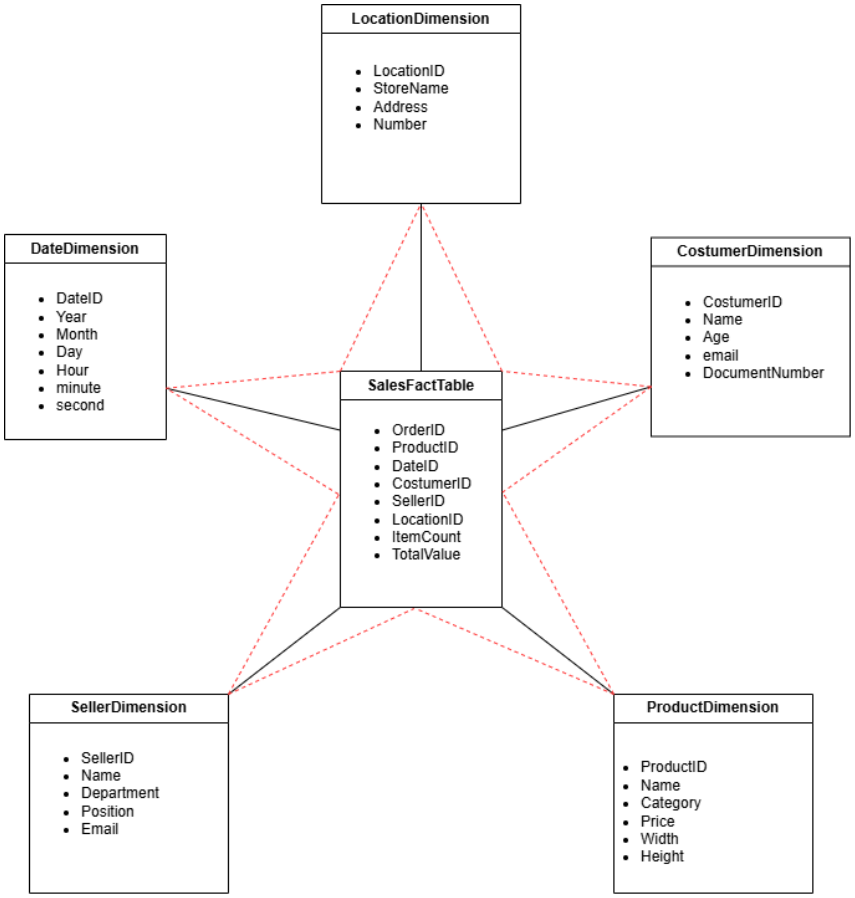

Figure 2: Star schema structure. The black lines represent the relation between fact and dimension tables, and the dashed red line represents the star-shape formed by the schema.

describe these dimensions. For example, a product dimension table may include product name, category, price, and supplier attributes. These attributes assist in understanding and analyzing the facts in the fact table.

A variation of the star schema is the snowflake schema, where dimension tables are normalized into multiple related tables. In this schema, dimension tables can have other dimension tables, which helps minimize data redundancy. While normalization reduces data redundancy and can save storage space, it makes the schema more complex and can slow down query performance.

Another common data warehouse schema is the galaxy schema, where multiple fact tables share dimension tables, further enhancing data modeling capabilities.

The star schema offers several key advantages. Its structure simplifies understanding and querying, making it accessible to non-technical users who benefit from the data warehouse through report generation and business analytics tasks, even without a background in data storage. The star schema is optimized for analytical queries due to the simple, well-defined relationships between tables, resulting in efficient query processing and fast response times. Furthermore, it is highly scalable, easily accommodating large volumes of data and the addition of new dimensions as business needs evolve.

However, the star schema's denormalized structure can lead to data redundancy, with some columns appearing in multiple tables to facilitate faster data retrieval. This redundancy can increase storage requirements and complicate data maintenance, particularly in data streaming environments where data is continuously ingested and updated. Here, the snowflake schema holds an advantage. Its normalized dimensions use less disk memory and allow updates to happen in a single place at the cost of increased complexity in the data warehouse's model.

2.3 ETL PROCESS

ETL (Extract, Transform, Load) is a process in data integration and preparation for data warehouses. The ETL process consists of three main stages.

The first stage, extraction, involves retrieving data from different sources such as operational databases, external systems, flat files, or APIs. The primary challenge during extraction is handling data in diverse formats and structures, ensuring that all necessary information is captured for subsequent processing.

The second stage is the transformation stage, which reshapes and refines the data to ensure data consistency and reliability within the system. This step often includes cleaning the data by removing duplicates or irrelevant entries, normalizing it into standard formats, and integrating data from multiple sources. Additionally, transformations may involve enriching the data with calculated fields or converting it into a format more suitable for the data warehouse schema. The transformation phase is crucial for maintaining data quality, enabling accurate analysis, and ensuring the data meets business requirements.

The third and last stage is the loading stage, where the transformed data is transferred into the data warehouse. This stage can involve loading the data into fact and dimension tables in a star or snowflake schema, depending on the data model used in the DW. The loading step ensures that the data becomes available for querying, reporting, and analysis.

2.4 DATA STREAMING

Data streaming refers to the continuous, real-time processing of data as it is generated. Unlike traditional batch processing, which involves collecting and processing data in intervals, data streaming processes individual pieces of data or small batches as they arrive. In data streaming, data flows continuously from the source to the destination, allowing for instant ingestion, processing, and analysis. However, it is important to clarify that while data streaming often deals with real-time data, the data being streamed does not necessarily need to have been collected in real-time. Streaming frameworks can handle both real-time data—data that is delivered immediately after collection without any significant delay—and historical or delayed data that is streamed for retrospective analysis or other purposes.

Moreover, data streaming architectures support distributed processing, enabling them to handle high-velocity and high-volume data efficiently. These systems ensure that even under heavy loads or network failures, the data stream can be processed without interruptions. This robustness makes data streaming a popular solution in large-scale environments where data must be processed continuously and reliably.

2.5 JOIN OPERATORS

Join operators are fundamental for combining data from multiple tables based on columns, which are typically keys. In data warehouses, where data is spread in dimension tables, joins play a pivotal role in querying data across fact and dimension tables. By enabling the merging of data from various sources, join operations are essential for retrieving comprehensive business insights, thereby supporting analysis and reporting of the data.

Several join operations are utilized in practice, including inner joins, outer joins, and semi-joins, each with a specific use case. While these operations are typically employed in batch processing for static datasets, the real-time demand of modern data systems has led to the development of optimized join methods, such as stream joins, designed for streaming environments where data is continuously ingested.

A semi-stream join is a join operation specifically optimized for streaming data. This operation allows the continuous joining of incoming data streams with either a static dataset or another stream. Unlike traditional join operations, which function in a batch-oriented manner, semi-stream joins operate as data arrives, enabling real-time data integration.

Stream joins are crucial for real-time analytics systems, where new data must be inte-

grated into existing analyses without delay. For example, in retail, a stream join can merge real-time sales transaction data with static product information, allowing immediate sales trends and inventory levels to be monitored. This is essential for operational tasks such as dynamic pricing or stock replenishment.

More specifically, the semi-stream join is utilized in the context of data streams, with the main advantage lying in their ability to perform real-time processing while optimizing memory usage. Since semi-stream joins only return matching rows from the incoming data stream, they require less memory than full join operations, making them suitable for environments handling high-velocity data streams.

2.6 RELATED WORKS

While several studies provide an overview of data streams, they predominantly focus on "big data" rather than specifically addressing data warehouses. Although big data encompasses data warehouses, the primary focus of these studies tends to diverge, emphasizing other types of data storage, such as data lakes, which operate differently and have distinct storage goals.

Siddiqa et al. [41] provide an overview of technologies used to store big data. This work highlights the trend of replacing relational data storage solutions with non-relational alternatives, exploring categories such as key-value, document-oriented, column-oriented, and graph databases, all of which fall under NoSQL storage. The work focuses on the technologies themselves, analyzing aspects such as features, applications of the technology, vendor, and design goal.

In another paper by Siddiqa et al. [40], the focus remains on big data, encompassing the entire data flow within the "big data management process." This process considers the storage aspect and includes network management, security, and classification/prediction. The paper discusses approaches for each stage of big data management and proposes a process flow for these applications. The authors also analyze data management techniques, assessing if they provide availability, scalability, integrity, heterogeneity, resource, optimization, and velocity.

Ashraf et al. [39] also address big data flow, from data source to data storage, emphasizing data processing along the way. The study evaluates when real-time processing is necessary compared to batch or near real-time approaches and discusses existing work in each stage of the data flow. Although data ingestion and storage are covered, the scope does not specifically include data warehouses. The authors state that big data often deals with unstructured data, which cannot be processed using relational databases.

Regarding the overall architecture of big data systems, Pääkkönen et al. [36] propose a reference architecture for big data systems and map the systems of well-known platforms (such as Netflix and LinkedIn) to this architecture. The proposed architecture encompasses the data flow among different system functionalities, including data extraction, analysis, visualization, and storage. The study also highlights the leading technologies used in each system functionality.

Despite addressing data ingestion and storage, the focus is not on improving these functionalities specifically for data warehouses.

Almeida [5] examines time series data processing in real-time within the context of big data. This study explores data processing frameworks and analyzes forecasting and anomaly detection in data streams. Although handling data streams in real-time is relevant to this work, Almeida's paper does not mention data warehouses, concentrating more on data analysis than on ingesting and storing data in a data warehouse. For example, the authors address many forecasting and anomaly detection algorithms.

Three review articles [11, 18, 27, 43] intersect with the scope of this work, but they not include analysis of papers from recent years, nor have the same scope. However, having been selected in the filtering process, these papers will be discussed in Chapter 4.5.

This work differentiates itself from previous studies by emphasizing data warehousing in the context of data streaming, addressing both the ETL (Extract, Transform, Load) process and the storage layer within the data warehousing architecture. This comprehensive approach synthesizes the most recent research in this rapidly evolving field, aiming to equip readers with knowledge of the latest advancements in data warehousing.

3

PROPOSED METHOD

The selection and filtering process for studies of interest in this work was based on a systematic mapping study found in the literature [22]. This work aims to provide an overview of the latest studies on ingesting and storing data in data warehouses within the context of data streaming, addressing the following research questions: How is data modeled in a data warehouse? (RQ1), How is data ingested into a data warehouse? (RQ2), and How is a data warehouse built and used in the context of data streaming? (RQ3).

RQ1 investigates how data is modeled in data warehouses for streaming data applications. RQ2 explores the ETL processes used in these contexts. RQ3 examines the modifications made to data warehouse implementations and data stream applications, referred to as "DW architecture," to handle stream data.

3.1 SEARCH STRATEGY

The studies analyzed in this work were extracted from IEEE Xplore, ScienceDirect, and SpringerLink digital libraries. These libraries were chosen for being commonly used in systematic reviews and mainly because of the access to the studies made possible by the *Centro de Informática* (CIn) VPN (Department of Computer Engineering at the Federal University of Pernambuco). For all three digital libraries, the search string used was "("data warehouse" OR "data warehousing") AND ("stream data" OR "streaming data" OR "data stream" OR "data streaming")". The searches were conducted on July 2nd, 2024, as described follows:

In the IEEE Xplore website[1], the search string was entered in the search bar on the home page. The "Subscribed Content" option was selected, and the "year" range was set to 2015-2024. "Conferences" and "Journals" were chosen, resulting in 59 studies (56 conferences and 3 journals).

Then, in the ScienceDirect website[2], the search string was entered in the search bar on the home page. The options selected were Subscribed journals, English, Computer Science, Review Articles, and Research Articles. The years 2015 to 2024 (inclusive) were chosen, yielding 242

[1] https://ieeexplore.ieee.org/Xplore/home.jsp
[2] https://www.sciencedirect.com/

studies (33 review articles and 209 research articles).

Lastly, in the SpringerLink website[3] using the old search interface of the home page, the search string was entered in the search bar. The following options were selected: Article, English, and Computer Science. The "Include Preview-Only content" option was unchecked, and the "Date Published" filter was set to 2015-2024, resulting in 238 studies.

To organize and track the studies found, Zotero[4] was used. The studies were exported to Zotero using the export options provided by the websites (IEEE Xplore and ScienceDirect) in RIS format, including title and abstract, or by using the Zotero Extension for Google Chrome (SpringerLink). All studies in Zotero were then exported as a CSV file and then loaded into Google Sheets for the subsequent steps of the study filtering process.

3.2 SELECTION PROCESS

This stage of the filtering process aimed to exclude studies that did not address at least one of the research questions and to select studies relevant to the objective of this paper.

3.2.1 Pre-Select Papers From Search

Only the title and abstract of the studies selected in the previous were analyzed at this stage. These fields were compiled into a Google Sheets file and reviewed by the author while applying the exclusion and inclusion criteria. Retracted studies and duplicates were removed during this step.

The main idea at this stage was to filter out studies that did not fit the scope of this work. For instance, some papers might not mention "data warehouse" or "data warehousing" in the title or abstract but discuss closely related concepts such as "ETL" or "semi-stream join". Therefore, papers clearly not within the scope of this work were excluded, while the remaining papers advanced to the next stage of selection. From the 536 studies initially selected, 64 were chosen.

3.2.2 Exclusion Criteria

The following criteria were applied to filter out studies:

- Studies not in the context of data streaming.
- Studies not focused on ETL or data storage or data warehouse architecture (e.g., papers focused on OLAP, Data Mining, or clustering).
- Studies focused on other types of data storage besides data warehouses (e.g., data lakes).

[3] https://link.springer.com/search
[4] https://www.zotero.org/

3.2.3 Inclusion Criteria

The following criteria were applied to select studies for the next stage of the filtering process:

- Studies published from 2015 to 2024 (inclusive).
- Studies written in English.
- Studies where the author has access to the full paper.
- Studies developed in the context of data streaming.

3.2.4 Selected Papers From Search

In this stage, the full text of each study was analyzed to determine if it fit within the scope of this work. The inclusion and exclusion criteria were applied, yielding 25 studies from 64 selected in section 3.2.1.

Figure 3 shows the number of studies at each stage of the paper selection process from the results of the string search.

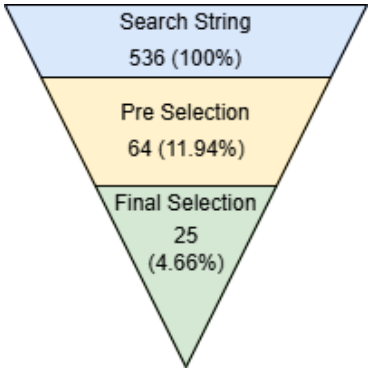

Figure 3: Number of studies at each stage of the paper selection process.

3.2.5 Snowballing and Categorizing

Based on the studies referenced in the papers selected in section 3.2.4, a snowballing process was performed, and 5 more studies were included in this work, yielding a total of 30 studies that will be analyzed. These studies were then categorized into the Data Warehouse Architecture, ETL, Join, and Others categories.

4

RESULTS

The following section will better explain the work done in each selected paper by category, namely: DW architecture, ETL, Join, and Others. After that, the main findings of this work, as well as the answers to the research questions, will be addressed. Tables 1, 2, 3, and 4 summarize the papers by its categories.

Table 1: Summary of papers in the DW Architecture category.

Title	Ref.	Section	Approach	Summary
An end-to-end framework for building data cubes over trajectory data streams	[25]	4.1.1	Snowflake Schema; Prime Number Encoding	Incremental Principal Component Analysis (IPCA) for region-based streaming, snowflake schema, prime number encoding for trajectory data
Cruncher: Distributed in-memory processing for location-based services	[1]	4.1.2	Big Data tool; Cache; batching technique	Distributed spatial data warehouse and streaming system built on Apache Spark with adaptive RDD partitioning for query processing
Data warehouse design approaches from social media: review and comparison	[27]	4.1.3	Literature Review	A Literature Review that analyzes existing solutions for social media and proposes two classes of Data warehouse design: "behavior analysis" and "sentiment analysis in data warehouse schema"
An adaptive and real-time based architecture for financial data integration	[12]	4.1.4	Discretized Streams; Hybrid Ontology; Big Data Tool; RDD	Uses hybrid financial ontology, resilient distributed datasets (RDDs), and real-time discretized streams with Apache Kafka for dealing with real-time financial data.
E-commerce big data computing platform system based on distributed computing logistics information	[14]	4.1.5	Big Data Tool; Metadata Management; Hierarchical Data Management; Distributed Computing	Hierarchical data warehouse system using Hadoop, Hive, and Kafka
On construction of a big data warehouse accessing platform for campus power usages	[8]	4.1.6	Big Data Tool; Distributed Computing	Real-time power monitoring platform using Apache Sqoop and Hive-based data warehouse
A rewrite/merge approach for supporting real-time data warehousing via lightweight data integration	[9]	4.1.7	Separation of Old and New Data; Parallelism; Query rewrite	Addition of a Dynamic DW for stream data while disk data is maintained in a "static" warehouse. Also proposes an intelligent query rewrite.
Research and Design on Architecture for Big Data Platform in Power Grid Dispatching and Control System	[37]	4.1.8	Distributed Computing; Big Data Tool	Unified architecture for data collection, management, service, and algorithm library using HBase, Hive, and Spark for real-time and batch processing

Table 2: Summary of papers in the ETL category.

Title	Ref.	Section	Approach	Summary
Problems and available solutions on the stage of extract, transform, and loading in near real-time data warehousing (a literature study)	[43]	4.2.1	Literature Review	A literature review on the problems and available solutions for ETL in near real-time DW
Integrating Big Data: A Semantic Extract-Transform-Load Frame- work	[7]	4.2.2	Ontology; Semantic	Semantic ETL framework that enhances data integration by introducing semantic technologies in the Transform stage
A big data perspective of current ETL techniques	[18]	4.2.3	Literature Review	Review of current ETL techniques for data streams, highlighting existing challenges, categorizing approaches to these challenges and assessing their efficiency.
Incremental ETL pipeline scheduling for near real-time data warehouses	[38]	4.2.4	Parallelism; Consistency Zones	Incremental ETL pipeline that processes data changes in near real-time by executing jobs in parallel, triggered by incoming queries. Consistency zones are introduced to ensure synchronized execution of tasks.
The challenges of Extract, Transform and Loading (ETL) system implementation for near real-time environment	[2]	4.2.5	Literature Review	Identifies challenges in ETL for near real-time environments, proposing solutions for extraction, transformation, and loading stages.
ETL-aware materialized view selection in semantic data stream warehouses	[28]	4.2.6	Materialized View Selection; Cache	Algorithm for dynamic materialized view selection, optimizing query performance in semantic data stream warehouses.
DOD-ETL: distributed on-demand ETL for near real-time business intelligence	[21]	4.2.7	Cache; Big Data Tool; Parallelism	Distributed on-demand ETL framework using micro-batch processing and in-memory computing with Apache Spark and Kafka.
Privacy-enhancing ETL-processes for biomedical data	[35]	4.2.8	Parallelism; Cell Suppression Algorithm	Integration of data anonymization, encryption, and access control into ETL for biomedical data, ensuring privacy and utility.
From Big Data to business analytics: The case study of churn prediction	[44]	4.2.9	Big Data Tool; Parallelism; Distributed computing	Scalable ETL pipeline using Apache Hadoop, Spark, and machine learning for churn prediction in telecom.
Metadata-Driven Industrial-Grade ETL System	[4]	4.2.10	Metadata management	Metadata-driven ETL system for industrial applications, automating processes with a metadata management framework
Distributed real-time ETL architecture for unstructured big data	[26]	4.2.11	Distributed Computing; Big Data Tool; Parallelism; Cache	Novel ETL architecture using Apache Kafka, Spark, and MongoDB for real-time unstructured big data processing

Table 3: Summary of papers in the Join category.

Title	Ref.	Section	Approach	Summary
A cached-based approach to enrich stream data with master data	[20]	4.3.1	Cache	Cached-based Stream-Disk Join (CSDJ) algorithm
Skewed distributions in semi-stream joins: How much can caching help?	[29]	4.3.2	Cache	Tuple-level caching and load shedding techniques
Optimising Hybridjoin to Process Semi-Stream Data in Near-Real- Time Data Warehousing	[30]	4.3.3	Cache; Parallelism	Addition of a second disk buffer to allow the probing and loading phases to happen in parallel
A memory-optimal many-to-many semi-stream join	[32]	4.3.4	Cache	Semi-Stream Balanced Join (SSBJ) algorithm
Optimizing semi-stream Cachejoin for near-realtime data warehousing	[31]	4.3.5	Cache; Parallelism	Addition of hash table for frequent tables with cache eviction and inequality algorithms
Big Data Velocity Management–From Stream to Warehouse via High Performance Memory Optimized Index Join	[19]	4.3.6	Cache inequality	Proposes the MOIJ (Memory Optimal Index-based Join) algorithm.
An efficient data access approach with queue and stack in optimized hybrid join	[34]	4.3.7	Parallelism; Cache; Old/New Data	Parallel-Hybrid Join (P-HYBRIDJOIN) and Hybrid Join with Queue and Stack (QaS-HYBRIDJOIN) algorithms

Table 4: Summary of papers in the Others category.

Title	Ref.	Section	Approach	Summary
A scalable real-time analytics pipeline and storage architecture for physiological monitoring big data	[6]	4.4.1	Big data tools	Architecture built around Apache Kafka for high-speed.
Challenges and Solutions for Processing Real-Time Big Data Stream: A Systematic Literature Review	[11]	4.4.2	Systematic review	Systematic literature review on challenges and solutions for real-time big data stream processing
The stream data warehouse: Page replacement algorithms and quality of service metrics	[13]	4.4.3	Page replacement	Page replacement algorithms (TRAFF, LATEN, HYBRI) and quality of service metrics
A New Approach Based on ELK Stack for the Analysis and Visualisation of Geo-referenced Sensor Data	[33]	4.4.4	ELK stack	Use of ELK stack for spatial data warehouse and georeferenced sensor data analysis

4.1 DATA WAREHOUSE ARCHITECTURE

4.1.1 An End-to-End Framework for Building Data Cubes over Trajectory Data Streams (2015)

Masciari [25] introduces a framework for analyzing trajectory data streams, emphasizing memory-efficient data representation, warehousing, and querying. The framework addresses the high dimension and redundancy commonly seen in trajectory data, such as GPS logs, which necessitates compression techniques.

The data warehouse employs a snowflake schema optimized for spatial features of trajectory data. It includes trajectory information such as location and time, organizing data by regions that contain trajectory crossings, thus enhancing query capabilities about sub-trajectories. The schema also addresses challenges like the "distinct count" issue when an object remains within the same region across multiple timestamps.

Encoded trajectories using prime number encoding form the basis for building specialized data cubes called Trajectory Cuboids (TRACs). These cuboids are tailored to efficiently query both location and time-based data, continually updating with streaming data to ensure the relevance and timeliness of the information.

4.1.2 Cruncher: Distributed In-Memory Processing for Location-Based Services (2016)

Abdelhamid et al. [1] present Cruncher, a distributed spatial data warehouse and streaming system built on Apache Spark. Cruncher incorporates several optimizations for spatial data handling, including adaptive RDD partitions for query processing and a novel batching technique that dynamically adjusts batch content ordering. It utilizes adaptive data partitioning to manage query and data hotspots. It incorporates a garbage collector and an efficient caching strategy to optimize in-memory data processing and minimize data redundancy.

4.1.3 Data Warehouse Design Approaches from Social Media: Review and Comparison (2017)

Moalla et al. [27] conduct a literature review on data warehouse design approaches for social media, comparing existing methodologies while highlighting their limitations. They address how each study included in their paper approached the data warehouse design and then categorized these approaches into two main types: behavioral analysis and the integration of sentiment analysis.

Behavioral analysis delves into web user activities on social media platforms, focusing on interactions like friendship creation, group formation, and messaging, particularly around

events in areas such as politics or sports. This analysis helps decision-makers understand user behavior patterns.

The integration of sentiment analysis into data warehouse schemas captures emotional and psychological responses to products or events. This approach leverages user-generated text to derive insights into public sentiment, aiding strategic decision-making processes.

4.1.4 An Adaptive and Real-Time Architecture for Financial Data Integration (2019)

Fikri et al. [12] propose an adaptive, real-time architecture for financial data integration to address the latency and semantic heterogeneity issues commonly found in traditional ETL processes. Their solution, called RDD4OLAP (Resilient Distributed DataStream for Online Analytical Processing), incorporates a hybrid financial ontology, resilient distributed datasets (RDDs), and real-time discretized streams.

The hybrid financial ontology standardizes business metadata across heterogeneous information systems, ensuring consistent understanding and mapping of financial data. RDDs enable efficient, fault-tolerant, in-memory data processing across a cluster of machines, providing the necessary resilience and speed for real-time analytics. The real-time discretized stream, facilitated by Apache Kafka, supports continuous data ingestion and processing, ensuring the financial data warehouse is always current.

The proposed architecture includes producers that connect to company information systems and convert data into the hybrid ontology format. Consumers then subscribe to these data streams and process them into RDDs. These RDDs are subsequently transformed and loaded into an in-memory data warehouse, allowing for real-time data integration and querying.

4.1.5 E-commerce Big Data Computing Platform System Based on Distributed Computing Logistics Information (2019)

Hu [14] outlines the architecture and implementation of a big data computing platform tailored for e-commerce. The platform is designed to overcome traditional data warehouse limitations in handling continuous large-scale data by integrating distributed computing technologies and real-time data processing frameworks.

The hierarchical data warehouse system described includes layers for data knowledge and metadata management, using Hadoop and Hive for data storage and computing and Kafka messaging queues. This architecture promotes error identification and resolution through its layer-specific responsibilities while implementing real-time computing engines to manage high-velocity data streams typical in e-commerce environments.

4.1.6 On Construction of a Big Data Warehouse Accessing Platform for Campus Power Usages (2019)

Chang et al. [8] detail the creation of a real-time power monitoring platform for a university campus. The platform's architecture features multiple layers, with the third layer housing the ETL process, data processing, and a search engine. Apache Sqoop is utilized for data ingestion into a Hive-based data warehouse, structured to operate within a four-machine cluster employing a master-slave configuration.

4.1.7 A Rewrite/Merge Approach for Supporting Real-Time Data Warehousing via Lightweight Data Integration (2020)

Ferreira et al. [9] introduced a rewrite/merge approach that divides the data warehouse into two main components: the Static Data Warehouse (S-DW) and the Dynamic Data Warehouse (D-DW).

The S-DW stores most historical data and maintains the indexes, constraints, and materialized views required for efficient query processing. In contrast, the D-DW handles recent data and is optimized for real-time data integration by avoiding indexes and materialized views. Consequently, the D-DW contains significantly less data than the S-DW, allowing quicker data loading and refresh operations.

The proposed architecture includes a Merger Component that rewrites incoming queries into sub-queries. These sub-queries are executed in parallel on both the S-DW and the D-DW. The results from both components are then merged to provide a unified and up-to-date response to the user. This approach enables real-time data integration without substantially impacting query performance.

4.1.8 Research and Design on Architecture for Big Data Platform in Power Grid Dispatching and Control System (2023)

Feng et al. [37] present a big data platform architecture for power grid dispatching and control. The platform architecture comprises four primary components: data collection, data specification, data service, and a general mathematical algorithm library. The data collection component integrates multi-source data from different systems. The data specification component prepares raw data in a synchronization layer for cleaning and fusion. The data service component organizes data into a unified layer, standardizing and labeling it for consistency. The general mathematical algorithm library supports data analysis and mining with advanced algorithms.

The data warehouse architecture of the platform is structured hierarchically to support multiple business topics. The source data layer gathers data from multiple systems. The synchronization layer temporarily stores and prepares raw data. The unified layer processes and

transforms data into standardized datasets. The analysis layer extracts business data for business intelligence and analysis, customizing it for specific needs.

The data flow in the big data platform involves writing real-time data to the synchronization layer (HBase) through messages, which are cleaned and stored in HBase and Hive. The storage design combines distributed and relational databases to handle real-time and batch processing.

4.2 ETL

4.2.1 Problems and available solutions on the stage of extract, transform, and loading in near real-time data warehousing (a literature study) (2015)

Wibowo [43] performed a literature review to identify problems on the ETL stage of a DW, and convey existing solutions. The author divides its review into the three phases of the ETL process (extract, transform, and load). The problems and solutions are shown in table 5.

Stage	Problem	Available Solutions
Extraction	Integrate Multiple - Heterogeneous data source	Change Data Capture + Stream Processor + Data Integration Tools
	Overload data source	Update significance and number of record changed method
		Special format for CDC log table
Transformation	Master data overhead	Master data cache + Database queue
	Need intermediate server to aggregate data	ELT (Extract Load Transform)
Loading	Performance degradation	Staging table (Trickle and Flip)
		Multi stage trickle and flip
	OLAP internal inconsistency	On staging table, OLAP is done outside the data warehouse update period
		Snapshot data
		RTDC (Real Time Data Cache)
		Layer-based view

Table 5: Problems and Solutions in the ETL phase of Near Real-Time Data Warehousing based on the table in [43].

4.2.2 Integrating Big Data: A Semantic Extract-Transform-Load Framework (2015)

Bansal and Kagemann [7] propose a semantic ETL framework that enhances data integration by introducing semantic technologies in the Transform stage. While the Extract and Load phases remain unchanged, the transform stage was modified. It is important to highlight that, in

the transform stage, there is a manual analysis of datasets, their schema, and their purposes. The results of this manual analysis allow the schema to be mapped to an existing ontology specific to the domain in question or, if preferred, the creation of a new ontology. More than one ontology may be needed if dealing with data sources from different domains, in which case alignment rules are needed. The Transform stage is divided into three steps: Data Preparation, Semantic Data Model, and Semantic Data Instances. In the semantic data model stage, the ontology creation, mapping of data fields, and alignment of similar data fields from multiple sources happen. In the data preparation step, the data is cleaned through duplicate removal, data normalization, integrity check, data filtering, and sorting and grouping. The next stage is the Semantic data model stage, where ontology creation, mapping of data fields, and alignment of similar data fields from multiple sources happen. Finally, RDF generation happens in the semantic data instances stage.

4.2.3 A big data perspective of current ETL techniques (2016)

Phanikanth and Sudarsan [18] review current ETL techniques from a big data perspective, highlighting the challenges posed by data streams and the difficulties traditional ETL processes face in meeting these demands. They categorize ETL approaches into Traditional ETL, "Near-Real-Time" ETL, and Real-Time ETL, discussing the analyzed papers' architectural considerations, scalability issues, and performance optimization techniques. Their analysis reveals that most current ETL techniques focus on structured data, with 14 out of 18 studies concentrating on this data type. Additionally, 11 studies adopt a "near-real-time" approach to ETL, while 7 studies use a "batch" approach. No study opted for a real-time strategy.

4.2.4 Incremental ETL pipeline scheduling for near real-time data warehouses (2017)

Qu and Deßloch [38] propose an incremental ETL pipeline that uses parallelism to execute a series of maintenance jobs. Each job handles a batch of delta tuples extracted from source-local transactions. This method ensures that data warehouse tables are brought up-to-date in response to incoming queries. The pipeline operates in a directed acyclic graph structure, where each transformation operator runs in a dedicated thread. As a result, data ingestion can be performed non-terminating, continuously processing new jobs as they arrive. However, this parallelism introduces potential data consistency challenges, particularly when multiple operators concurrently access and update staging tables or slowly changing dimension (SCD) tables.

To address these consistency challenges, the authors introduce two "consistency zones" types designed to maintain data integrity in SCDs and incremental joins. Consistency zones ensure that operations on shared tables occur in a controlled, synchronized manner, preventing anomalies such as stale data or phantom reads during concurrent updates. Consistent zones allow

multiple operators to work in parallel while guaranteeing that each job is processed in the correct sequence, maintaining the overall consistency of the data warehouse.

4.2.5 The challenges of Extract, Transform and Loading (ETL) system implementation for near real-time environment (2017)

Sabtu et al. [2] examine the challenges of implementing ETL systems in near real-time environments, identifying key issues such as latency, data consistency, and system scalability. The authors review existing ETL frameworks and their limitations, categorizing the challenges within the extraction, transformation, and loading stages.

In the extraction stage, the main challenge is capturing change data from streaming sources like network traffic and sensors without overloading the system. Solutions include streaming processors and Change Data Capture (CDC) techniques, which ensure continuous updates without disrupting operations.

For the transformation stage, maintaining data consistency while frequently updating master data poses a challenge. Solutions involve using methods that avoid compromising master data integrity and employing frequent, smaller batch processing to handle continuous data streams efficiently.

The loading stage faces challenges in maintaining optimal performance during OLAP to avoid overlaps and inconsistencies. Proposed solutions include using dynamic mirror technology and staging data separately for analysis and updates to ensure data accuracy.

The paper explores various techniques and solutions in the literature, such as cloud-based infrastructures and real-time data caching.

4.2.6 ETL-aware materialized view selection in semantic data stream warehouses (2018)

Berkani et al. [28] address optimizing query performance in semantic data stream warehouses through ETL-aware materialized view selection. The authors propose a novel algorithm that integrates the ETL process into selecting materialized views, ensuring that these views are up-to-date and relevant to query workloads.

The methodology comprises three main components: "main memory and cache management", dynamic materialized view selection, and a stream ETL process.

The memory and cache management component ensures efficient usage of memory during ETL transformations and materialized view maintenance by using main memory as the primary storage, thus avoiding disk access bottlenecks. Memory is divided into four partitions: buffer of inputs, buffer of streams, view cache, and cache memory.

The dynamic materialized view selection component identifies the most relevant materialized views for the current workload and stores them in the view cache. Views no longer used

are evicted from memory to optimize space and performance.

The stream ETL process begins by extracting instances from RDF data sources into the buffer of inputs. Instances that do not require data from the DW are merged and loaded directly into the DW, while the remaining instances are processed in the buffer of streams. The buffer of streams handles data from both sources and the DW to perform necessary ETL transformations. Finally, ETL operations are executed in the cache memory.

4.2.7 DOD-ETL: distributed on-demand ETL for near real-time business intelligence (2019)

Machado et al. [21] introduce DOD-ETL, a distributed on-demand ETL framework designed to support near real-time business intelligence. DOD-ETL addresses key challenges in ETL processes, including handling multiple heterogeneous data sources, performance degradation, and master data overhead. The framework combines strategies from previous works to achieve high availability, low latency, and horizontal scalability.

DOD-ETL leverages micro-batch processing and in-memory computing, integrating with Apache Spark and Apache Kafka for high-speed data ingestion, transformation, and loading. The framework employs in-memory databases on each processing node, enabling prompt access to master data and ensuring consistent join operations for data with varying arrival rates.

The architecture of DOD-ETL includes a distributed, parallel, and technology-independent pipeline. It utilizes stream processing alongside an in-memory master data cache, a buffer to manage late data with timestamps, and a unified programming model for compatibility with multiple stream processing frameworks. DOD-ETL also incorporates log-based change data capture (CDC) to process data on-demand, minimizing the impact on the source database.

4.2.8 Privacy-enhancing ETL-processes for biomedical data (2019)

Prasser et al. [35] tackle the challenge of implementing privacy-enhancing ETL processes for biomedical data. They propose a solution that integrates data anonymization, encryption, and access control into the ETL pipeline, emphasizing risk assessment to preserve the data's integrity and schematic properties. Their approach used a cell suppression algorithm, which enforces risk thresholds by recursively removing specific attribute values from records. This algorithm treats suppressed values as a unique category, forming equivalence classes that reduce re-identification risk by ensuring all records in a class are indistinguishable. This method maintains data plausibility and correctness without perturbing input data or generating synthetic data.

Additionally, the proposed ETL pipeline operates stream-oriented, where each row of data is treated as an atomic and isolated unit within the data stream. This row-by-row processing enables pipeline parallelism, ensuring efficient and privacy-preserving data handling in real-time ETL environments.

4.2.9 From Big Data to business analytics: The case study of churn prediction (2020)

Zdravevski et al. [44] present a case study on applying big data ETL processes to churn prediction in the telecommunications industry. They demonstrate a scalable ETL pipeline combining traditional ETL tools for small-volume dimensional data and use big data technologies for high-volume transactional data. Spark processes big data on Object Storage Services (OSS) while integrating dimensional data from the data warehouse, ensuring efficient handling of wide fact tables for denormalization or foreign key setup.

The paper introduces Distributed Load Agents to offload the load process to remote machines, minimizing database server impact. The on-demand Hadoop cluster acts as an edge node from the data warehouse perspective.

Three ETL data-flow scenarios are outlined, focusing on data in fact tables. Initial steps involve Spark loading and distributing business and surrogate keys of dimensions across nodes. Subsequent steps read new data from OSS with high parallelism, utilizing its hierarchical storage structure. Processed data is then loaded into HDFS and subsequently into the data warehouse using a distributed load algorithm. The workflow ends with loading collected metadata into the data warehouse before cluster termination.

4.2.10 Metadata-Driven Industrial-Grade ETL System (2020)

Suleykin and Panfilov [4] discuss the design and implementation of a metadata-driven ETL system for industrial applications, particularly in the railway transportation environment. The authors address the complexity and rigidity of traditional ETL systems by leveraging a proprietary metadata management framework (MMF) to automate and streamline ETL processes.

The proposed system optimizes the workflow using a metadata model to define data structures, transformations, and dependencies. The MMF automates typical ETL tasks by generating Airflow Directed Acyclic Graphs (DAGs), which execute ETL jobs using open-source technologies. The MMF provides a single entry point for metadata management, automatic pipeline construction, and synchronization of development environments, reducing errors and the need for manual configuration.

The architecture includes a PostgreSQL-based metadata repository that stores system contours, source structures, and loading steps. Also, the MMF supports distributed data storage.

4.2.11 Distributed real-time ETL architecture for unstructured big data (2022)

Mehmood and Anees [26] present a novel ETL architecture designed for real-time processing of unstructured big data. This architecture addresses the challenge of join operations among sources with different data transfer rates, which can lead to stream data loss. Also, it

utilizes a distributed approach, leveraging Apache Kafka for data ingestion, Apache Spark for real-time processing, and MongoDB for storage.

The ETL architecture consists of three main layers: Data Collection, Processing, and Distributed Data Store layers.

The data collection layer captures heterogeneous stream tuples from multiple sources into several partitions without data loss.

The processing layer performs parallel transformation processes, including user-defined stream filtering, join query processing and other queries. The join window performs stream-disk join operations by loading in-memory distributed database data into a disk-based dataframe. Join queries run over both stream and disk-data dataframes, combining attributes based on a common attribute, with the resulting joined tuples being consumed by various real-time applications such as data lakes, data warehouses, and analytical tools.

The distributed data store layer, built on a NoSQL database, provides load balancing, scalability, and in-memory data access. This architecture uses a distributed message processing system with message queuing middleware to compensate for differences in data source rates, ensuring seamless data integration.

4.3 JOIN

4.3.1 A Cached-based approach to enrich Stream data with master data (2015)

Naeem et al. [20] propose the Cached-based Stream-Disk Join (CSDJ) algorithm to enhance semi-stream joins for one-to-many equijoins in skewed data distributions. CSDJ introduces a cache module that frequently stores portions of the disk-based relation in memory that are frequently accessed, acting as a stream-probing phase to reduce disk I/O and improve performance.

The CSDJ algorithm has two phases: the stream-probing phase and the disk-probing phase. The stream-probing phase quickly matches incoming stream tuples against the cached R tuples in H_R. If no match is found, the disk-probing phase scans the disk-based relation R.

The architecture includes a disk buffer, queue, stream buffer, hash table H_s for stream tuples, and hash table H_R for cached R tuples. This design ensures efficient join operations by alternating between the stream-probing and disk-probing phases, optimizing performance and reducing disk access.

4.3.2 Skewed distributions in semi-stream joins: How much can caching help? (2017)

Naeem et al. [29] investigate the impact of caching and load shedding techniques on the performance of semi-stream joins, mainly when dealing with skewed data distributions, and focusing on one-to-many equijoins. The authors propose two optimization techniques: tuple-level caching and load shedding.

Tuple-level caching involves storing frequently accessed master data tuples in memory to reduce disk I/O and improve join performance. This method ensures optimal memory usage by caching only the most frequent tuples, unlike previous page-level caching approaches. They combine this caching technique with existing semi-stream join algorithms like MESHJOIN and HYBRIDJOIN, creating CMESHJOIN and CHYBRIDJOIN, respectively.

Load shedding aims to increase the rate of join outputs by shedding the most expensive stream tuples to process. This allows the algorithm to identify and discard the most costly tuples, improving output rates.

4.3.3 Optimising Hybridjoin to Process Semi-Stream Data in Near-Real-Time Data Warehousing (2019)

Naeem et al. [30] propose an optimized version of the HYBRIDJOIN algorithm by introducing a second disk buffer. This enhancement allows the probing and loading phases to run in parallel, with one buffer in the probing phase while the other is in the loading phase. This parallelism reduces wait times, as one buffer is always ready for use immediately after the other completes its task, thereby improving overall efficiency by reducing the wait time for a buffer to load.

4.3.4 A memory-optimal many-to-many semi-stream join (2019)

Naeem et al. [32] introduce the Semi-Stream Balanced Join (SSBJ) algorithm, designed to optimize memory usage in many-to-many semi-stream joins. The SSBJ algorithm addresses the challenge of joining a fast-incoming data stream with a larger, slower-changing dataset on disk. It does so by introducing a "cache inequality" criterion for optimal memory allocation, balancing between a cache for frequently accessed master data and a hash table for stream data. This method differs from existing caching approaches, which often assume clustered indices or unique key constraints, making them unsuitable for many-to-many joins.

The SSBJ algorithm features two phases: the cache phase and the disk phase. The stream input first enters the cache phase, where it is joined with the cached master data in H_R. If no match is found, the stream tuple is forwarded to the disk phase, based on MESHJOIN, with added cache migration logic. This alternation between phases ensures efficient join operations.

The architecture includes a disk buffer, switching buffer, stream buffer, and two hash tables: H_s (for stream tuples) and H_R (for cached R tuples). The cache phase quickly matches frequent stream tuples, while the disk phase handles the remaining tuples, optimizing performance and reducing disk access. The dominant components, H_s and H_R, ensure efficient memory usage and cache management.

The authors mention that in further work, the two phases can be parallelized and executed in different threads.

4.3.5 Optimizing semi-stream Cachejoin for near-realtime data warehousing (2019)

Naeem et al. [31] propose two modifications to the existing CACHEJOIN approach for semi-stream join. The first modification is called P-CACHEJOIN (Parallel Cache Join), and the second is OP-CACHEJOIN (Optimized Parallel Cache Join).

P-CACHEJOIN enables the parallel execution of the SP and DP phases of CACHEJOIN. The SP (Stream Probe) phase starts by probing incoming stream records against a cached set of records recurrently accessed. In the DP (Disk Probe) phase, a partition of the data stored in the disk is loaded into memory for joining with the stream records that did not find a match during the SP phase. The modification for P-CACHEJOIN introduces an intermediate buffer, which stores stream records that do not match during the SP phase. This architecture allows the DP phase to proceed in parallel by fetching unmatched records from the intermediate buffer, reducing the idle time between phases and increasing throughput.

OP-CACHEJOIN further optimizes P-CACHEJOIN by addressing the disk I/O bottleneck. It introduces a second disk buffer, which allows the parallel loading of stored data while the DP phase is using the original disk buffer. This setup ensures that the DP phase can switch between buffers with reduced delay.

4.3.6 Big Data Velocity Management–From Stream to Warehouse via High Performance Memory Optimized Index Join (2020)

Naeem et al. [19] present the Memory Optimal Index-based Join (MOIJ) algorithm, designed to optimize memory usage and enhance the performance of semi-stream joins in near-real-time data warehousing. This work is similar to the SSBJ algorithm presented by Naeem et al. [32], with the primary difference being the consideration of indexed master data. MOIJ uses the "cache inequality" approach to determine the optimal memory distribution between a cache for frequently accessed master data and the main hash table for stream data. This algorithm incorporates a tuple-level cache, storing only the most frequent master data tuples, and dynamically adjusts memory allocation based on data distribution, similar to SSBJ.

4.3.7 An Efficient Data Access Approach With Queue and Stack in Optimized Hybrid Join (2021)

Aziz et al. [34] propose two optimized algorithms, Parallel-Hybrid Join (P-HYBRIDJOIN) and Hybrid Join with Queue and Stack (QaS-HYBRIDJOIN), to enhance the efficiency of semi-stream joins in near-real-time data warehousing. The existing HYBRIDJOIN algorithm uses a single buffer to load disk partitions, leading to suboptimal performance due to waiting for the next partition to overwrite the existing one.

P-HYBRIDJOIN addresses this limitation by introducing a second disk buffer (db2) alongside the original buffer (db1). These buffers operate in parallel, with one buffer in the probing phase while the other loads the next partition into memory. This parallel operation eliminates waiting times and significantly improves the service rate. During the probing phase of db1, db2 is loaded into memory, and once db1 completes its probing, db2 becomes available for the join operator. This cycle continues, ensuring that the join operation never waits for data to be loaded into memory.

QaS-HYBRIDJOIN further optimizes the join process by incorporating a queue and stack (QaS) component. This new component replaces the queue (Q) in P-HYBRIDJOIN and is implemented as a double-linked list. QaS allows for simultaneous joining with both the oldest and newest key attributes. Specifically, db1 handles the oldest index attributes from the queue, while db2 handles the newest index attributes from the stack. This configuration potentially increases the number of matches by allowing the join process to utilize both recent and old key attributes effectively.

4.4 OTHER

4.4.1 A Scalable Real-Time Analytics Pipeline and Storage Architecture for Physiological Monitoring Big Data (2018)

Baljak et al. [6] present a scalable, distributed architecture designed to collect, store, and analyze high-volume, high-velocity physiological data from bedside monitors. The proposed system leverages the open-source stream processing software Apache Kafka to ensure high-speed, real-time data handling.

The architecture manages two types of data: high-resolution EKG (electrocardiogram) waveforms and vital signs (e.g., blood pressure). The data flow begins with sensors publishing the data as topics in Kafka. The data is then consumed and written directly into a database (for vital signs) or converted into an intermediary format for fast data collection. Subsequently, the data is integrated into an existing data warehouse (DW), enabling end users to perform queries.

For EKG waveforms, the data is stored in JSON files due to its size, with their locations indexed in the DW to facilitate quick access and analysis. This architecture effectively supports

the real-time integration and querying of large-scale physiological data.

4.4.2 Challenges and Solutions for Processing Real-Time Big Data Stream: A Systematic Literature Review (2020)

Mehmood and Anees [11] conducted a systematic literature review to analyze the challenges and solutions in real-time big data stream processing, particularly for real-time data warehousing (DWH). The authors identified key challenges in this field, including in-memory computing, distributed computing, low latency requirements, effective resource allocation, fast disk I/O operations, distribution of stream engines, and real-time processing of spatiotemporal data streams.

The review highlighted various tools and technologies developed to address these challenges, such as Apache Spark, Yahoo! S4, MapReduce, and Kafka Streams. The authors found that while numerous algorithms exist for real-time join processing of structured data in ETL for real-time DWH, there is less work on unstructured data. The study emphasizes the need for further research in real-time stream processing for unstructured data and suggests developing stream engines that are flexible and adaptable to specific business requirements. Additionally, the authors propose exploring cloud-based DWH solutions and purpose-built ETL tools for improved efficiency and scalability.

4.4.3 The stream data warehouse: Page replacement algorithms and quality of service metrics (2023)

Gorawski et al. [13] focus on the memory paging mechanism of a stream OLAP cube, termed the Cube Iterator. This mechanism manages the continuous flow of data from historical (persistent) and real-time (volatile) sources, providing users with up-to-date analytical results. The authors identify page replacement algorithms as critical for optimizing this data transfer process. They propose three new algorithms that adapt to changing data environments and user requirements. These algorithms consider data retrieval time, consumption speed, and user-defined constraints to balance the needs of data producers (sources) and consumers (users). Additionally, the authors introduce two quality of service metrics: one for consumers, focusing on data delivery speed and continuity, and another for producers, emphasizing database load and query efficiency.

4.4.4 A New Approach Based on ELK Stack for the Analysis and Visualisation of Geo-referenced Sensor Data (2023)

Ngo et al. [33] propose using the ELK (Elasticsearch, Logstash, Kibana) stack as a spatial data warehouse for analyzing georeferenced sensor data, particularly in environmental

applications like the CEBA project. The authors detail a method to implement and query multidimensional models in Elasticsearch, showcasing its ability to handle time series, spatial data, and objects. They introduce the Integration and Aggregation Tool (IAT), a streaming ETL component that integrates diverse sensor data and loads it into Elasticsearch, driven by user configuration. The system architecture is presented as divided into sensors, data processing, storage, and visualization.

4.5 OVERVIEW OF DATA INSIGHTS

Analyzing the most frequent combination of words used in the titles and abstracts of the selected papers (as shown in Table 6) confirms that the selected studies fall within the scope of data ingestion and storage in a data warehouse (DW) in the context of data streaming. This word frequency analysis also reflects the categories used to group the papers. Common and general phrases, such as "of the," "in this," and "in order to," were removed to clean the data about bigram and trigram phrases.

Table 6: Bigram and Trigram Phrase Counts

Bigram Phrase	**Frequency**	**Trigram Phrase**	**Frequency**
big data	28	real-time data warehousing	7
data warehouse	20	real-time stream processing	7
data warehousing	15	data warehouse design	4
real-time data	14	extract transform load	4
stream data	13	railway kpi data	4
master data	13	near real-time data	4
stream processing	12	disk-based master data	4
near real-time	11		
etl system	9		
semi-stream join	9		
real-time stream	9		

The distribution of publication years for the selected studies (shown in Figure 4) indicates that research on handling data streaming in data warehouses is ongoing. It is important to note that the paper selection was performed midway through 2024, so additional papers may still be published this year that will not be part of this work.

4.6 OVERALL APPROACHES USED IN ALL CATEGORIES

The analysis of the selected studies provided insights into the strategies employed to address the challenges of data ingestion and storage in data warehouses within the context of data streaming. Common techniques identified are parallelization, distributed computing, efficient memory usage, adding memory as a trade-off with better time efficiency, and different data

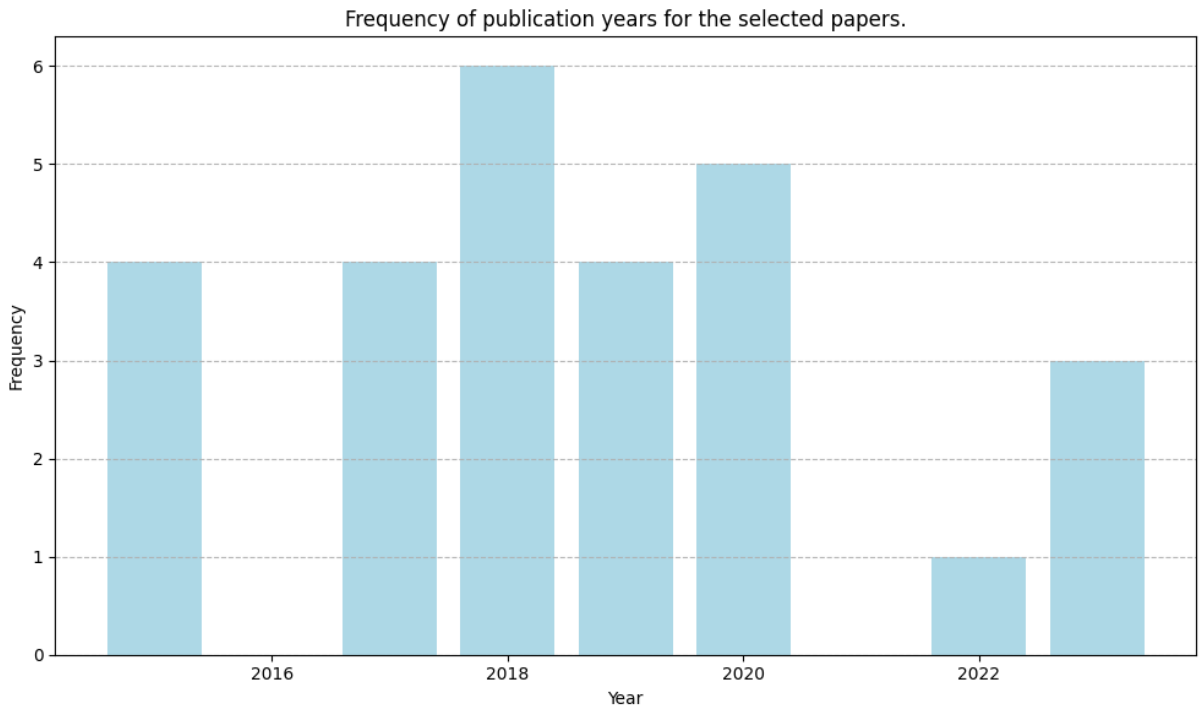

Figure 4: Frequency of publication years for the selected papers.

handling for disk and stream data. These strategies enhance the performance and scalability of data warehouses, enabling them to handle the demands of data streams effectively.

4.7 SPECIFICS OF THE MOST RELEVANT STUDIES

The following sections provide more details of the proposed approach of the most relevant studies.

4.7.1 Data Warehouse Architecture

4.7.1.1 A rewrite/merge approach for supporting real-time data warehousing via lightweight data integration

The authors propose a method to separate streaming data from historical data by introducing a dynamic data warehouse (DW) specifically designed to handle recent stream data (see Figure 5). This dynamic DW stores only the most up-to-date streaming data, while older data is eventually transferred to a static data warehouse, which resides on disk and manages historical data.

Due to its need for frequent updates, the dynamic DW avoids using indexes and materialized views, thereby reducing processing time since it does not require constant updates to these structures. On the other hand, the static DW is responsible for maintaining indexes and materialized views, allowing it to handle historical data efficiently. The dynamic DW is called a

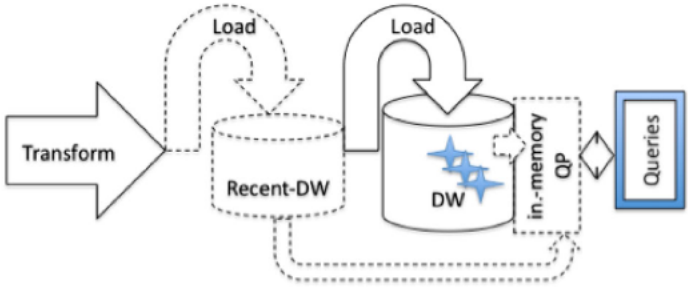

Figure 5: Architecture of dynamic data warehouse with an in-memory query processor [9].

"lighter" warehouse, reflecting its simplified structure and reduced overhead.

Additionally, the system includes an in-memory component represented by the "In-memory QP" situated alongside the static DW. This component processes queries in real-time by executing them in parallel across both the dynamic and static data warehouses. Once both warehouses return their results, the system merges these results using a "Merger" component and presents the final output to the user. This approach ensures that queries benefit from both real-time and historical data without sacrificing performance or data freshness.

4.7.1.2 Cruncher: Distributed In-Memory Processing for Location-Based Services

The authors propose the development of an in-memory data warehouse specifically designed for spatial data (see Figure 6). Built on top of Apache Spark, the system addresses Spark's limitations, which are that it is not inherently optimized for spatial data processing. A key modification introduced by the authors is the reordering of batch data during processing, allowing for more efficient system updates compared to Spark's default sequential stream processing.

Cruncher also dynamically monitors frequently accessed data to optimize partition updates and reduce the processing time required to retrieve data, unlike static partitions. Cruncher uses three components to implement its features: a KD-Tree to store the location of partitions, a high-granularity grid called the data catalog to track the frequency of spatial data usage, and a lineage graph that monitors updates to the system's resilient distributed datasets (RDDs).

Additional features of Cruncher include inter-query optimization, a garbage collector that retains only the most up-to-date spatial data in memory, and data persistence, achieved through the lineage graph which stores the history of data transformations. The system also provides users a graphical user interface (GUI), further enhancing accessibility.

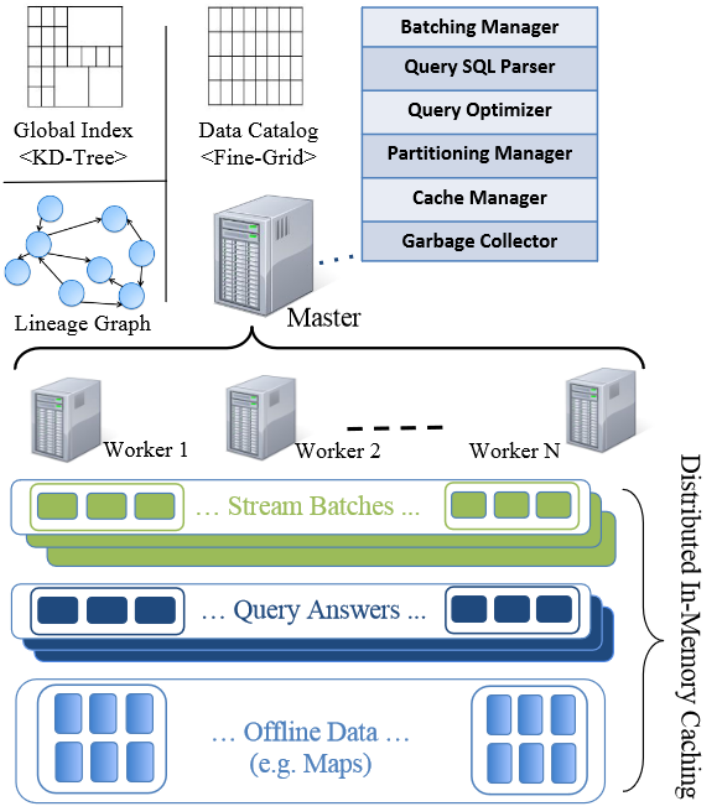

Figure 6: Cruncher architecture [1].

4.7.1.3 *An end to end framework for building data cubes over trajectory data streams*

The authors use the snowflake schema, shown in Figure 7, to model spatial data, and this is an example of how more traditional schema is still being used to model data in the context of data streaming. The schema is a snowflake because the space dimension table contains a region ID, which points to another dimension table.

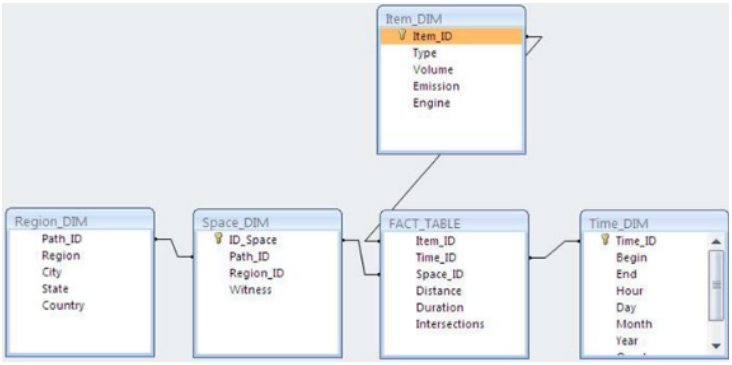

Figure 7: Snowflake schema used to model spatial data in a data warehouse[25].

4.7.2 ETL

4.7.2.1 DOD-ETL: distributed on-demand ETL for near real-time business intelligence

The authors propose a distributed, technology-independent ETL architecture that leverages caching, parallelism, and a real-time data flow, as depicted in the study's flow diagram in Figure 8. The process begins with a data source where a Change Data Capture (CDC) mechanism tracks database operations (CRUD) in real time by monitoring database logs.

Once extracted, the data is processed by the Message Producer, which partitions the data and creates messages using the table keys. These messages are then sent to a message queue, which operates on a publish/subscribe model, directing the messages to the Stream Processor for further processing.

The In-Memory Table Updater maintains distributed in-memory tables, storing the data necessary for the transformation stage. The Data Transformer performs the actual data transformations, retrieving any missing data from the distributed in-memory cache and processing the data partitions in parallel. Finally, the Target Database Updater converts the transformed data into SQL queries and loads it into the target data repository.

It is important to note that the Extract, Transform, and Load (ETL) processes are executed in parallel for each data partition (that was created in the Message Producer component), maximizing the efficiency and speed of the data pipeline. This approach ensures near real-time processing and supports the scalability needed for business intelligence applications.

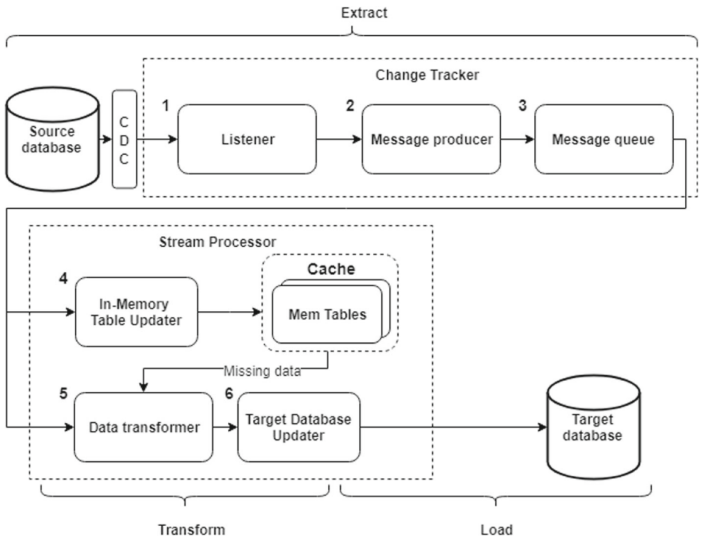

Figure 8: DOD-ETL system architecture for the ETL process[21].

4.7.2.2 *Distributed real-time ETL architecture for unstructured big data*

The authors propose an ETL architecture that utilizes a distributed database before loading data into a real-time data warehouse, as shown in Figure 9. The system's data flow begins at the data collection layer, which gathers data from various sources, potentially in different formats. This data is then passed to the processing layer, where three key transformations occur in parallel: data filtering and transformation, the processing of join queries, and the execution of other queries.

The stream join process occurs in the join window, which is integrated with the distributed database. Here, a tuple from the stream, represented in the stream dataframe, is matched with data from the disk, stored in the disk-data dataframe, via the join query operator. However, this "disk" is not a traditional disk; it is a distributed database, providing advantages such as load distribution, scalability, and in-memory storage. This complexity is managed through the distributed system transparent layer, which abstracts the distributed nature of the underlying database.

Once the data has been processed through this system, it can be loaded into a real-time data warehouse, ensuring that it is readily available for real-time analysis.

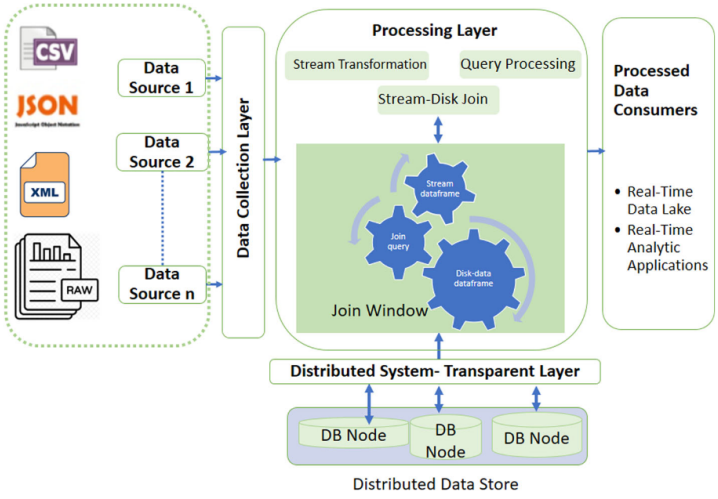

Figure 9: Distributed real-time ETL architecture with the usage of a distributed in-memory database[26].

4.7.3 Join

4.7.3.1 *Optimising HYBRIDJOIN to Process Semi-Stream Data in Near-real-time Data Warehousing*

The authors propose an optimization to the traditional HYBRIDJOIN algorithm by introducing a second buffer (see Figure 10). This addition allows the system to perform the loading and probing phases in parallel, thereby enhancing the efficiency of the join process.

In the join process, disk-based elements are first loaded into a buffer, where they are compared with incoming stream data. This step is referred to as the loading phase. The probing phase follows, during which the system uses the data in the buffer to compare with stream data tuples. The optimization introduced by the authors eliminates idle time between these phases by ensuring that while one buffer is busy loading data, the other is simultaneously being used for probing. This way, the system does not need to wait for the loading phase to complete before starting the probing, as there is always a buffer ready for probing while the other handles loading. This parallel execution significantly reduces the processing delay in semi-stream joins, improving the overall performance in near-real-time data warehousing environments.

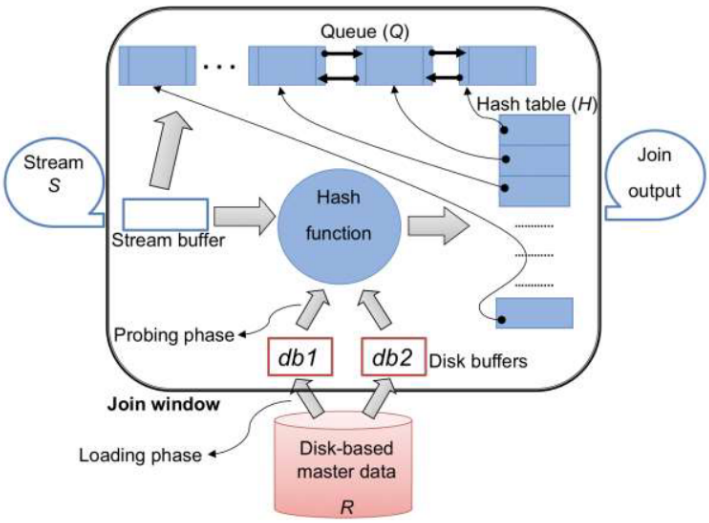

Figure 10: Join architecture with the addition of a second disk buffer[30].

4.7.3.2 A Cached-based Approach to Enrich Stream Data with Master Data

The authors introduce a caching mechanism designed to store the most frequently accessed tuples (as seen in Figure 11), thereby optimizing the join process between stream data and disk-based master data. A frequency monitor is employed to track access patterns and determine when tuples in the cache should be replaced by more frequently accessed tuples from the disk. In this way, if a disk-based tuple becomes more frequently used than one in the cache, it is moved into the cache.

This approach is advantageous because accessing tuples stored on disk is more costly in terms of time and resources compared to accessing them in memory. By storing the most frequently accessed tuples in the cache, the system reduces the need for repeated disk access, which improves the overall efficiency of the join process. Incoming stream data first checks the cache for the required tuples, and only if the desired tuple is not found in the cache does the system query the disk. This layered approach optimizes performance by prioritizing faster cache access while maintaining access to all data on disk when necessary.

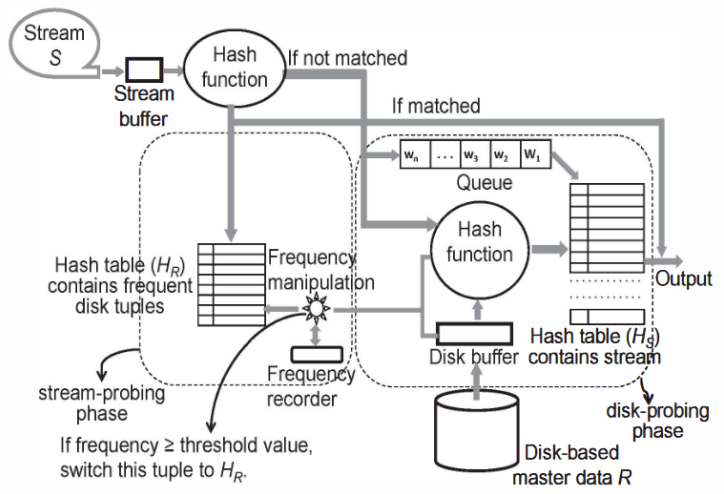

Figure 11: Join architecture using a stream cache[20].

4.7.3.3 *Optimizing Semi-Stream CACHEJOIN for Near-Real Time Data Warehousing*

The authors introduce an intermediate buffer (seen in Figure 12) that stores tuples that have undergone the probe phase in the stream but did not yield a match. Since the intermediate buffer retains tuples from the data stream that have already passed through the stream cache, the cache probing phase can get the next tuple to process it while the intermediate buffer is used to probe the disk in parallel. This concurrent probing is only interrupted when the intermediate buffer if full, necessitating a pause in the stream probe, or when the buffer is empty, resulting in a pause of the disk probe—though, but these scenarios are considered rare.

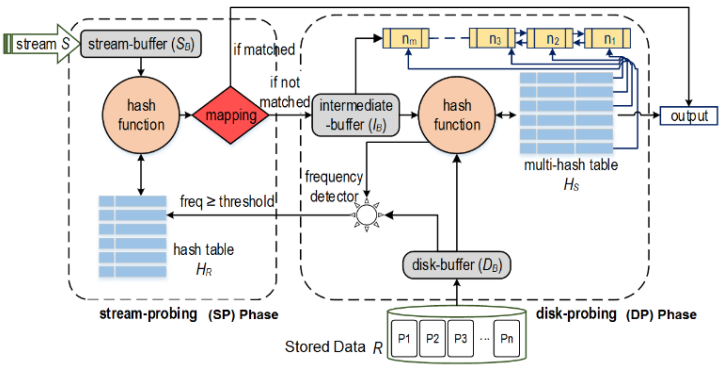

Figure 12: Join architecture using an intermediate buffer in the disk probing phase[31].

Additionally, the author proposes a hybrid approach that integrates the utilization of an auxiliary disk buffer and a stream cache, as discussed previously, alongside the intermediate buffer to store stream tuples (see Figure 13). This design facilitates parallel processing of both stream and disk probing, as well as the loading and probing phases of data within the disk's buffer.

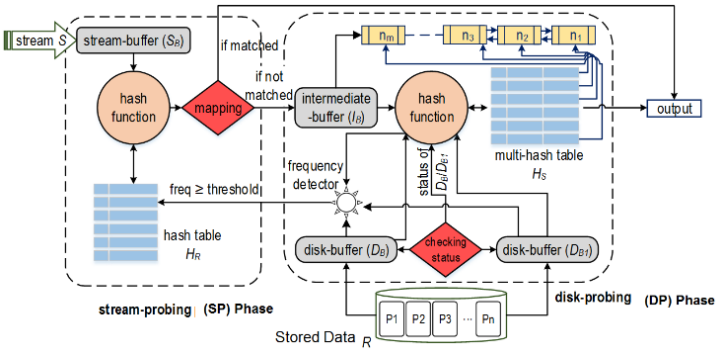

Figure 13: Join architecture using the combination of additional disk buffer, stream cache, and intermediate buffer[31].

4.8 APPROACHES TO OVERCOME CHALLENGES IN DW FOR DATA STREAMS PER CATEGORY

In the domain of data warehouse (DW) architecture, the analyzed studies underscore the importance of both dynamic and static data warehouses, alongside the adoption of big data tools and in-memory processing techniques. One approach involves the separation of DW into static and dynamic components, where the static component stores historical data with indexes and materialized views, while the dynamic component is optimized for real-time integration without indexes [9]. Tools like Apache Kafka and Apache Spark are frequently utilized to manage high-velocity data streams, ensuring efficient processing and real-time analytics [12, 1].

Regarding ETL processes, the studies strongly focus on parallelism, efficient memory management, and distributed architectures as key strategies for handling continuous, large-scale data flows. In this context, parallelism allows multiple ETL tasks to execute simultaneously, reducing overall processing time and latency [21, 44]. Efficient memory management, such as in-memory databases and cache optimization, is crucial in maintaining high performance, particularly in real-time environments where data volumes are immense [26], the usage of cache also helps speed up the transformation process by providing missing data to perform the transformation. Distributed architectures further enhance scalability by distributing the ETL workload across multiple computing nodes, ensuring balanced processing and improved throughput [18, 7].

In the context of join operations, the studies frequently utilize techniques such as cache memory, parallelization, and distributed computing to address performance bottlenecks in data streaming environments. Cache memory plays a pivotal role in reducing disk I/O by storing frequently accessed data, thereby accelerating join processes [32, 29]. Parallelization enables concurrent execution of join operations, significantly improving processing times [30, 34]. Additionally, distributed computing is employed to spread join tasks across multiple nodes, balancing the computational load and enhancing scalability [31, 19].

4.9 RQ1: HOW IS DATA MODELED IN A DATA WAREHOUSE?

The first research question aimed to explore data modeling perspectives in data warehouses within the context of data streaming. The analysis reveals that traditional schemas (such as star and snowflake schema) with fact and dimension table schema are still commonly used, for example, in applications with geospatial data [25]. Instead of proposing new schemas to model the data in a DW, recent studies have shifted towards addressing operational challenges such as dealing with high-volume and high-velocity data of data streaming. This indicates a maturity in data modeling practices where the main concern is optimizing the performance and efficiency of data handling processes such as ETL and joins. Still, regarding the data aspect, studies use metadata management[14][4], ontology[12][7], and semantics [7][28] to model data in data warehouses as a consequence of heterogeneous data from multiple data sources. Ontology is important for describing a project's domain's entities and their relation. Semantics are used to ensure that terms have clear and standardized meanings. For example, two sources may use different terms to reference the same thing, like "customer" and "client"; that's where semantics come in, to map different "terms" to the system's semantics. Lastly, metadata management allows the system to retrieve information about the sourced data. Managing these aspects is important for data consistency across the system.

4.10 RQ2: HOW IS DATA INGESTED INTO A DATA WAREHOUSE?

The second research question investigates the methods employed for data ingestion into data warehouses. The studies suggest that parallel processing, in-memory computing, and distributed computing are key techniques to enhance data ingestion processes.

The works adopted techniques to allow parallelism in the execution of data extraction tasks [21], dataset processing [35][44], data transformation [26], and loading tasks, significantly reducing latency. Parallelism is also applied to make stream joins faster by using disk buffers and executing the loading and probe phases in parallel in these disks[30]. Another approach is to execute the stream and disk probing phases in parallel by probing the disk for the tuples that were not in the cache[31].

In-memory computing leverages the fast access speeds of modern memory systems to perform transformations quickly[28], process in-memory queries[26] and store data (such as tables) for quick access. Also, an in-memory table updater may be used to update a system's in-memory tables[21]. The usage of caches appeared both in works that address stream joins, where caching allows stream data to access a cached table with disk information before going to the disk if needed, and also in ETL where cache is used to provide missing information to the transformation stage [21].

Lastly, distributed computing facilitates the division of tasks across multiple nodes, ensuring load balancing and scalability, which is crucial for real-time data environments. It is

possible to use distributed load agents to perform the load stage of ETL[44]. The extract phase can also take advantage of distributed computing to extract data from multiple sources[26]. Big data tools also make it easier to use distributed computing, like the Hadoop Distributed File System or Kafka, with distributed messaging queues.

Besides Parallelism, cache, and distributed computing, another approach is dealing with stream data and historical data separately [9][34].

4.11 RQ3: HOW IS A DATA WAREHOUSE BUILT AND USED IN THE CONTEXT OF DATA STREAMING?

The third research question focuses on the architectural aspects of building data warehouses and how the data warehouse is used in a system's architecture in the context of data streaming.

The studies emphasize how data warehouses are incorporated into the system and the architecture of the data warehouse itself. Architectures can combine static (deals with old data) and dynamic (deals with the data stream) data warehouses [9] to handle real-time time by reducing indexes and materialized views updates in the disk. Another approach is to maintain the system's statistics to manage partitioning, caching, and query processing [1]. On top of that, the cache can be used to make use of distributed computing to decentralize the workload [14].

Big data tools play an important role in managing data streams, and they are explicitly mentioned in many of the works analyzed, which rely on them to enable real-time data ingestion and storage. Tools such as Apache Kafka, Spark, Hive, Impala, Hadoop, Sqoop, and HDFS are critical for enabling systems to handle high-velocity data, and many rely on these tools to manage the complexities of big data. Apache Kafka builds real-time data pipelines and provides messaging queues, allowing distributed streaming of high-throughput event data across systems. Apache Spark handles large-scale datasets' real-time and batch processing through high-speed, distributed data processing and in-memory computation. Apache Hive enables querying and managing large datasets in distributed storage, while Apache Impala facilitates low-latency SQL queries on data stored in Hadoop. Apache Hadoop serves as a framework for distributed storage and processing of big data using clusters of commodity hardware. Sqoop efficiently transfers bulk data between Hadoop and relational databases, bridging structured and unstructured data. Finally, HDFS (Hadoop Distributed File System) offers a scalable, fault-tolerant file system for storing massive datasets across distributed machines.

5

CONCLUSION

This study provided a comprehensive overview of recent advancements in data ingestion and storage strategies for data warehouses within the context of data streaming. The analysis of selected studies has highlighted various techniques and architectural approaches that address the challenges of handling high-velocity, continuous data streams.

One of the significant findings is the continued reliance on traditional data modeling schemas such as the star and snowflake schemas. Despite the focus of contemporary research shifting towards optimizing operational aspects like data ingestion, these schemas remain foundational due to their effectiveness in organizing and querying a large amount of data. This highlights the maturity of data modeling practices where the emphasis is on enhancing the performance and efficiency of data processing rather than reinventing the schema design.

The insights gained from this study underscore the importance of advanced techniques such as cache/in-memory processing, parallelization, distributed computing, and usage of big data tools. These strategies not only enhance the performance and scalability of data warehouses but also ensure their capability to handle the demands of real-time data environments effectively.

The advancements in data ingestion and storage strategies highlighted in this review show the trend for the ongoing development of efficient and scalable data warehouses. By addressing the operational challenges and leveraging advanced for ingesting high volume and high velocity data, data warehouses can continue providing timely and reliable insights, facilitating informed decision-making in various fields. Future research should continue exploring innovative approaches to optimize these processes further and address emerging data streaming and warehousing challenges.

REFERENCES

[1] A. S. Abdelhamid, M. Tang, A. M. Aly, A. R. Mahmood, T. Qadah, W. G. Aref, & S. Basalamah (2016). Cruncher: Distributed in-memory processing for location-based services. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, 1406–1409. Journal Abbreviation: 2016 IEEE 32nd International Conference on Data Engineering (ICDE).

[2] A. Sabtu, N. F. M. Azmi, N. N. A. Sjarif, S. A. Ismail, O. M. Yusop, H. Sarkan, & S. Chuprat (2017). The challenges of Extract, Transform and Loading (ETL) system implementation for near real-time environment. In *2017 International Conference on Research and Innovation in Information Systems (ICRIIS)*, 1–5. Journal Abbreviation: 2017 International Conference on Research and Innovation in Information Systems (ICRIIS).

[3] A. Silva & C. Antunes (2013). Towards the Integration of Constrained Mining with Star Schemas. In *2013 IEEE 13th International Conference on Data Mining Workshops*, 413–420. Journal Abbreviation: 2013 IEEE 13th International Conference on Data Mining Workshops.

[4] A. Suleykin & P. Panfilov (2020). Metadata-Driven Industrial-Grade ETL System. In *2020 IEEE International Conference on Big Data (Big Data)*, 2433–2442. Journal Abbreviation: 2020 IEEE International Conference on Big Data (Big Data).

[5] Almeida, A., Brás, S., Sargento, S., & Pinto, F. C. (2023). Time series big data: a survey on data stream frameworks, analysis and algorithms. *Journal of Big Data*, 10(1):83.

[6] Baljak, V., Ljubovic, A., Michel, J., Montgomery, M., & Salaway, R. (2018). A scalable realtime analytics pipeline and storage architecture for physiological monitoring big data. *CHASE 2018 Special Issue*, 9-10:275–286.

[7] Bansal, S. K. & Kagemann, S. (2015). Integrating Big Data: A Semantic Extract-Transform-Load Framework. *Computer*, 48(3):42–50.

[8] Chang, C.-H., Jiang, F.-C., Yang, C.-T., & Chou, S.-C. (2019). On construction of a big data warehouse accessing platform for campus power usages. *Journal of Parallel and Distributed Computing*, 133:40–50.

[9] Cuzzocrea, A., Ferreira, N., & Furtado, P. (2020). A rewrite/merge approach for supporting real-time data warehousing via lightweight data integration. *The Journal of Supercomputing*, 76(5):3898–3922.

[10] Dibouliya, A. (2023). Review on: Modern data warehouse how is it accelerating digital transformation. *nternational Journal of Advance Research, Ideas and Innovations in Technology*.

[11] E. Mehmood & T. Anees (2020). Challenges and Solutions for Processing Real-Time Big Data Stream: A Systematic Literature Review. *IEEE Access*, 8:119123–119143.

[12] Fikri, N., Rida, M., Abghour, N., Moussaid, K., & El Omri, A. (2019). An adaptive and real-time based architecture for financial data integration. *Journal of Big Data*, 6(1):97.

[13] Gorawski, M., Pasterak, K., Gorawska, A., & Gorawski, M. (2023). The stream data warehouse: Page replacement algorithms and quality of service metrics. *Future Generation Computer Systems*, 142:212–227.

[14] Hu, J. (2019). E-commerce big data computing platform system based on distributed computing logistics information. *Cluster Computing*, 22(6):13693–13702.

[15] Inmon, W. H. (2005). *Building the Data Warehouse*. John Wiley & Sons, New York, NY, USA, 4th edition.

[16] J. Kreps, N. N. & Rao, J. (2014). Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB'11: 6th International Workshop on Networking Meets Databases*.

[17] K. Grolinger, W. A. Higashino, A. T. & Capretz, M. A. (2014). Data management in cloud environments: Nosql and newsql data stores. *Journal of Cloud Computing: Advances, Systems and Applications*, 2(1):1–24.

[18] K. V. Phanikanth & S. D. Sudarsan (2016). A big data perspective of current ETL techniques. In *2016 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, 330–334. Journal Abbreviation: 2016 International Conference on Advances in Computing and Communication Engineering (ICACCE).

[19] M. A. Naeem, F. Mirza, H. U. Khan, D. Sundaram, N. Jamil, & G. Weber (2020). Big Data Velocity Management–From Stream to Warehouse via High Performance Memory Optimized Index Join. *IEEE Access*, 8:195370–195384.

[20] M. A. Naeem, I. S. Bajwa, & N. Jamil (2015). A Cached-based approach to enrich Stream data with master data. In *2015 Tenth International Conference on Digital Information Management (ICDIM)*, 57–62. Journal Abbreviation: 2015 Tenth International Conference on Digital Information Management (ICDIM).

[21] Machado, G. V., Cunha, , Pereira, A. C. M., & Oliveira, L. B. (2019). DOD-ETL: distributed on-demand ETL for near real-time business intelligence. *Journal of Internet Services and Applications*, 10(1):21.

[22] Maciel, L., Oliveira, A., Rodrigues, R., Santiago, W., Silva, A., Carvalho, G., & Miranda, B. (2022). A systematic mapping study on robotic testing of mobile devices. In *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 475–482.

[23] Marr, B. (2016). *Big Data in Practice*. John Wiley & Sons, Chichester, UK.

[24] Marz, N. & Warren, J. (2015). *Big Data: Principles and best practices of scalable real-time data systems*. Manning Publications, Shelter Island, NY, USA.

[25] Masciari, E. (2015). An end to end framework for building data cubes over trajectory data streams. *Journal of Intelligent Information Systems*, 45(2):131–164.

[26] Mehmood, E. & Anees, T. (2022). Distributed real-time ETL architecture for unstructured big data. *Knowledge and Information Systems*, 64(12):3419–3445.

[27] Moalla, I., Nabli, A., Bouzguenda, L., & Hammami, M. (2017). Data warehouse design approaches from social media: review and comparison. *Social Network Analysis and Mining*, 7(1):5.

[28] N. Berkani, L. Bellatreche, & C. Ordonez (2018). ETL-aware materialized view selection in semantic data stream warehouses. In *2018 12th International Conference on Research Challenges in Information Science (RCIS)*, 1–11. Journal Abbreviation: 2018 12th International Conference on Research Challenges in Information Science (RCIS).

[29] Naeem, M., Dobbie, G., Lutteroth, C., & Weber, G. (2017). Skewed distributions in semi-stream joins: How much can caching help? *Information Systems*, 64:63–74.

[30] Naeem, M. A., Aziz, O., & Jamil, N. (2019a). Optimising hybridjoin to process semi-stream data in near-real-time data warehousing. In *CONF-IRM*, 27.

[31] Naeem, M. A., Mehmood, E., Malik, M. G. A., & Jamil, N. (2019b). Optimizing Semi-Stream CACHEJOIN for Near-Real- Time Data Warehousing:. *Journal of Database Management*, 31(1):20–37.

[32] Naeem, M. A., Weber, G., & Lutteroth, C. (2019c). A memory-optimal many-to-many semi-stream join. *Distributed and Parallel Databases*, 37(4):623–649.

[33] Ngo, T. T. T., Sarramia, D., Kang, M.-A., & Pinet, F. (2023). A New Approach Based on ELK Stack for the Analysis and Visualisation of Geo-referenced Sensor Data. *SN Computer Science*, 4(3):241.

[34] O. Aziz, T. Anees, & E. Mehmood (2021). An Efficient Data Access Approach With Queue and Stack in Optimized Hybrid Join. *IEEE Access*, 9:41261–41274.

[35] Prasser, F., Spengler, H., Bild, R., Eicher, J., & Kuhn, K. A. (2019). Privacy-enhancing ETL-processes for biomedical data. *International Journal of Medical Informatics*, 126:72–81.

[36] Pääkkönen, P. & Pakkala, D. (2015). Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems. *Big Data Research*, 2(4):166–186.

[37] Q. Feng, F. Di, R. Ye, L. Xie, Y. Wang, L. Tao, Y. Huang, D. Li, & C. Feng (2023). Research and Design on Architecture for Big Data Platform in Power Grid Dispatching and Control System. In *2023 IEEE 6th Information Technology,Networking,Electronic and Automation Control Conference (ITNEC)*, 6:887–891. Journal Abbreviation: 2023 IEEE 6th Information Technology,Networking,Electronic and Automation Control Conference (ITNEC).

[38] Qu, W. & Deßloch, S. (2017). Incremental etl pipeline scheduling for near real-time data warehouses. In *Datenbanksysteme für Business, Technologie und Web (BTW 2017)*, 299–308. Gesellschaft für Informatik, Bonn.

[39] S. Ashraf, Y. M. Afify, & R. Ismail (2022). Big Data for Real-Time Processing on Streaming Data: State-of-the-art and Future Challenges. In *2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, 1–8. Journal Abbreviation: 2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME).

[40] Siddiqa, A., Hashem, I. A. T., Yaqoob, I., Marjani, M., Shamshirband, S., Gani, A., & Nasaruddin, F. (2016). A survey of big data management: Taxonomy and state-of-the-art. *Journal of Network and Computer Applications*, 71:151–166.

[41] Siddiqa, A., Karim, A., & Gani, A. (2017). Big data storage technologies: a survey. *Frontiers of Information Technology & Electronic Engineering*, 18(8):1040–1070.

[42] Stonebraker, M. & Çetintemel, U. (2013). "one size fits all": An idea whose time has come and gone. *Communications of the ACM*, 51(12):76–83.

[43] Wibowo, A. (2015). Problems and available solutions on the stage of Extract, Transform, and Loading in near real-time data warehousing (a literature study). In *2015 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, 345–350.

[44] Zdravevski, E., Lameski, P., Apanowicz, C., & Ślzak, D. (2020). From Big Data to business analytics: The case study of churn prediction. *Applied Soft Computing*, 90:106164.