IMPROVING THE PERFORMANCE OF REAL-TIME DATA ANALYTICS APPLICATIONS BY OPTIMISING THE DATABASE AGGREGATION

Thennamurage Dona Methma Pabasarie Samaranayake

199359L

MSc in Computer Science

Department of Computer Science and Engineering
Faculty of Engineering

University of Moratuwa
Sri Lanka

June 2022

IMPROVING THE PERFORMANCE OF REAL-TIME DATA ANALYTICS APPLICATIONS BY OPTIMISING THE DATABASE AGGREGATIONS

Thennamurage Dona Methma Pabasarie Samaranayake

199359L

This thesis was submitted in partial fulfilment of the requirements for the Degree of MSc in Computer Science Specialising in Software Architecture

MSc in Computer Science

Department of Computer Science and Engineering
Faculty of Engineering

University of Moratuwa
Sri Lanka
June 2022

DECLARATION

I declare that this is my own work and this has not incorporated without acknowledgement any material previously submitted for Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief, it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. Also, I hereby grant to the University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other media. I retain the right to use this content in whole or partially in future works.

....

T. D. M. P. Samaranayake Date

I certify that the declaration above,
The candidate is true to the best of my knowledge and that this project report is acceptable for evaluation for the final research.

....

Prof. Indika Perera Date

ABSTRACT

Organisations must make the best decision at the appropriate time to obtain a competitive advantage in a fast-changing market. To accomplish so, it's critical to make faster and more efficient judgments based on near-real-time data analysis. When it comes to these real-time streaming data analysis systems, the performance of the database is having a huge impact on such applications as it is required to achieve data availability and continuous processing for a large volume of data without a delay. When it comes to streaming data, data warehousing is more challenging. So, it is required to consider performance improvements in all the steps of the Extraction, Transformation, Load (ETL) process and the database architecture level. Therefore, the proposed approach is to improve the performance of the system by optimising the ETL process (Extraction, Transformation, and Load) and real-time data warehousing. In this approach, the optimised aggregation algorithm is introduced. Apart from that, the hardware, storage schemas, and query optimization of the data warehouse are also considered and this study is evaluating the performance of the centralised architecture for the real-time data warehouse.

ACKNOWLEDGEMENT

My profound gratitude goes to Prof. Indika Perera, my supervisor for the knowledge, supervision, advice, and guidance provided with his expertise, throughout in making the thesis a success. My appreciation goes to my family for the motivation and support provided throughout my life. I also would like to thank my colleagues in the MSc batch and at my workplace for their help and support in managing my research work.

THE TABLE OF CONTENT

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1
INTRODUCTION

Organisations need to use analytics applications to process real-time raw data related to the business efficiently to make better and faster decisions. The performance of the database is having a huge impact on such applications as it is required to achieve data availability and continuous processing without delay.

Real-time data generated a large volume of data from multiple, disparate sources. Therefore, it is required to integrate those data into a single database and a data model. Additionally, attempts to conduct massive, long-running analysis queries in transaction processing databases might induce database isolation level lock conflicts in transaction processing systems. [6]. The data warehouse is the answer to these problems. It is a kind of a central repository that stores information from multiple data sources. Most of the time, those are past and commutative information. After further processing, this warehoused information can be used in future forecasting and drawing insights based on them. Hence it's an important part of business intelligence.

A batch-driven approach is used in traditional data warehouses, and it is mainly following the ETL (extract, transform, load), and the ELT (extract, load, transform) approaches. Also, most of the time it is focussing on the 'pull' data model. So the data persisted in the data warehouse is mainly performed in off-peak times such as nightly or weekly basis. Due to that, real-time data is not available in a typical data warehouse [1]. Apart from that, there can be delays in the operational systems during the extraction process and it will be a major concern from the business point of view as it requires quick access to up-to-date information for important business decisions. Most of the research literature has covered traditional data warehousing and the approaches for improving its performance.

It's more challenging to allow users to access real-time/nearly real-time data acquired from different sources, record those data for future processing, and process those data by analysing the complex structure and the relationships between that data. When it comes to the prediction and decision-making part, we need to analyse both real-time and past data. The data compression will improve performance and decrease the cost.

To improve the performance of a real-time analytics application, the data warehouse implementation should be improved during the ETL process and also in data warehouse architecture. New data is regularly added to a data warehouse during the

ETL process. The statistics retrieved from that data are determined by the data warehouse's function and the type of business it supports.
To improve the performance during the ETL process, it is required to consider the core functions such as joins, and aggregations [7]. When it comes to the warehouse, the database type (relational databases, NoSQL databases) also performs differently.
Therefore, in this study, I would like to come up with more optimised real-time data warehouse implementation with new aggregation algorithms for data summarization and an ETL process for real-time data streaming applications.

1.1 Research Problem

With the high throughput received to the database, the tuning and optimising need to be done for all the ETL, partitioning, warehousing, and data integration steps for a better performance of the application. Also, it is required to choose the best architecture from a number of warehousing architecture patterns [1] such as centralised data warehouses, independent data marts, dependent data marts, homogeneously distributed data warehouses, and heterogeneous distributed data warehouses [2]. These classic data warehouses are built utilising a batch-driven strategy, with data input carried out in batches during off-peak hours [3]. But, in this case, a real-time data warehousing approach should be implemented to accommodate the requirements of the organisation’s system [4]. Therefore, when modelling the deployment architecture for a deployment which is having an analytics application, choosing the best architecture for the database is very important.

1.2 Research Objectives

As discussed under the research problem, there is a requirement to have a real-time data warehouse which can persist real time data as soon as possible ensuring there are no down times in the data warehouse due to data backup. Furthermore, the performance improvement is also a major concern.
So, the object is to improve the performance of streaming analytics applications by optimising the aggregation approach.

1.3 Scope

At the end of this research, a real-time data warehouse system will be proposed and implemented which offers better performance and accuracy. The proposed system provides the capability to generate some events in the streaming application and retrieve the statistics for those events in real-time.

The scope of this research includes designing and implementing a streaming application and its real-time data warehouse. To do that, it is required to select the ETL methodologies for generated data and come up with a new aggregation approach. Furthermore, selecting the real-time data warehouse architecture and evaluating the system performance with proper metrics are also required.

1.4 Outline

The first chapter describes the research challenge that this study will attempt to solve. It also gave the motivation and objectives that needed to be met during the research's completion. The second chapter discusses the additional studies that have been conducted in the fields of Extraction, Transformation and Loading (ETL) process and real-time data warehousing. These two topics were chosen in order to compile current information on existing research and tools in real-time data warehousing systems. The methodology chapter explains the methodologies used during the system's development. When designing and executing the final proposed system, several steps have been outlined. The architecture of the suggested system has been designed and organised in the next chapter. The evaluation criteria are presented in the evaluation chapter.

CHAPTER 2
LITERATURE REVIEW

Available research on streaming analytics and data warehousing were studied to understand the research background and their proposed solutions under these domains. The approaches followed in those researches have given the direction for the methodology proposed through this research.

2.1 ETL Process

Real-time data sources generate a large volume of data. The basic goal of ETL is to eliminate redundant data that isn't needed for decision-making and to speed up report generation. Workloads that access data at random, usually to do a brief search, insert, update, or delete, are known as online transaction processing (OLTP). OLTP processes are usually carried out in parallel by a large number of users who use the database in their business operations. Normally, these data sources that store data in data warehouses can be external or internal. As a result, big data files are necessary to temporarily store heterogeneous data streams before storing them in the data warehouse. After reformatting and rearranging the data streams, the extracted data items are moved to the staging area [8].

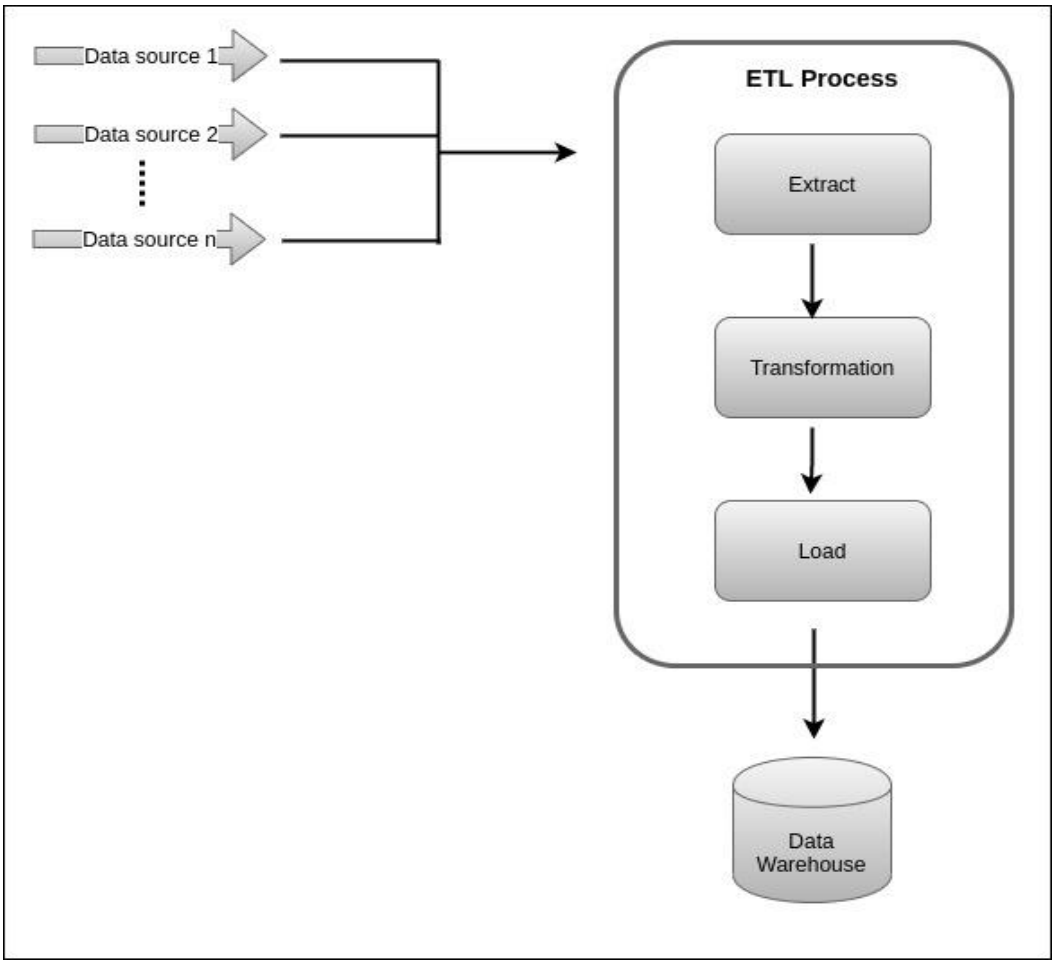

Figure 2.1: The ETL process can be applied for the streams from diverse sources

In the context of big data technologies, [9] discusses a novel way of adapting ETL operations with Big Data called Big Dimensional ETL (BigDimETL), which works with the ETL process and in a Multidimensional structure. The results of this approach's experiments suggest that ETL operation modification can work well, especially with this BigDimETL's Join operation.

In addition, some research on NoSQL ETL was done in [8], with an emphasis on semi-join operations utilising MongoDb for real-time data warehousing. The researchers used this method to complete the analysis, using a join module and various datasets, and evaluating the performance by looking at CPU and memory usage. This led to the conclusion that, regardless of the type of the data stream, stream join processing requires fixed-sized memory and constant CPU time in the

present configuration. Stream processing with big data also necessitates additional disc I/O.

Another study [10] employed Aggregations and Joins for large-scale data processing and found that operations like ETL, data staging for a data warehouse, and database loading performed better. The researchers employed high-performance joins for CDC in this investigation (Changed Data Capture). Furthermore, this strategy reduced resource consumption, allowing for significant cost savings by consolidating hardware.

Furthermore, according to [11], ETL methods are generally used offline [1], and these techniques have latency and reliability issues. [11].

Also, since real-time streaming is generating a high volume of data at a high velocity, volume and velocity management is very important in this research. The study [23] looked at how to manage huge data velocity using an optimised index join. A semi-stream join operator is used to implement the suggested join operation by caching the frequently utilised bits of the master data. The researchers developed a new Memory Optimal Index-based Join (MOIJ) method to test the proposed approach. This allows for many-to-many joins and adjusts to dynamic streaming data.

A distributed processing engine for spatial big-data processing based on batch and streams is discussed in [24]. In this study, the collected spatial information from SNS companies is processed using a dedicated ETL tool and also used in distributed and parallel processing methods considering the velocity and the volume of the data.

Furthermore, according to [26], ETL procedures account for 70% of the warehouse's implementation and maintenance effort.

2.2 Stream Data Warehousing

A data stream warehouse is either a data stream management system that holds historical data or a warehouse system that is updated in near real-time rather than during downtimes. Three types of feasible system designs were described in the study [12], together with their benefits and drawbacks, as well as unresolved concerns in the issues in query languages, optimisations of the performance, and quality of the data streams.

The research paper [13] described a 24/7 real-time data warehouse, and the approach provided here uses replication of the data and temporary tables to transform a typical business database into a real-time data warehouse, allowing continuous data loading and batch and real-time integration. Additionally, they investigated data loading performance utilising in-memory databases like IBM SolidDB and Oracle TimesTen.

A join module and multiple datasets are implemented in a NoSQL data warehouse to store both structured and unstructured data coming from various sources at different I/O rates [8]. Also, performance analysis has been conducted based on CPU and memory utilisation in [8]. In addition, after evaluating semi-stream join processing, the difficulties in joining NoSQL streams were discovered in this study.

In [3], a real-time data warehouse approach was suggested using web services as the prototype. The log capturing and triggering was used in capturing change data from source systems and real-time partitioning is also used in this.

A new semi-stream join called SSJ is introduced in [14] and it is being used to combine a stream with a master data table which is changing slowly that is stored on a disk. Then it was compared to a seminal algorithm named MESHJOIN, which can be employed in this situation. With the SSJ, a cost model of the new algorithm has been introduced and validated with experiments.

In [3], the data has been partitioned real-time and merged into the warehouse, When a user queries a component, this module transmits the query to the data warehouse if the query just needs historical data; if the query requires both historical and instantaneous data, this module rewrites the question to get and integrate data. And views were used to complete the rewriting.

Furthermore, in [21] data warehouse performance focussed techniques such as hardware tuning, physical storage schemas, query optimization techniques, and Additional data structures that improve the efficiency of searching are described. Indexes (join, bitmap, and bitmap join), partitioned tables and materialised views, and are the data structures covered in [21].

The study [22] compares client-server, RAD-UNIFY type DBMS, and improved client-server systems in terms of performance. The study goes through the individual functional components as well as the design rationales. In addition, the study found that the RAD-UNIFY architecture outperforms the client-server architecture in small workloads and that the improved client architecture outperforms the other two in terms of performance and scalability.

The [25] is a systematic overview of the challenges of summarising real-time big data sources in the literature. It has addressed some key research questions in this area, such as which publications are relevant for real-time stream processing studies, which difficulties have been encountered during the development of real stream processing, that methods were reported to deal with the issues introduced at the ETL phase while processing real-time streams for real-time DWH, and what evidence has been reported while addressing various challenges for processing.

Other than the above mentioned real-time approaches, the traditional data warehouses were implemented using a batch processing approach. The batch processing approach is also required in certain use cases when it's required to take decisions based on a certain period of time.

The research paper [29] is having a good comparison between batch based and stream based processing of the big data. In that paper, they have identified big data processing methods into various types such as interactive-based, visual-based, stream-based, batch-based, DAG based and graph-based and focussed on stream-based and the batch-processing areas. As per the study, the batch processing provides efficiency by processing simultaneous jobs for batch wise for a large volume of collected data. The stream processing is done via stream processors and the batch processing is mostly achieved via MapReduce programming model using the divide and conquer method. It has four main steps, input splitting, map phase, shuffle and sort phase and the reduce phase. Apart from that, the researchers have

shared a framework comparison for stream analytics frameworks in these processing approaches by considering Apache Spark, Apache Storm, IBM Infosphere etc. Also a feature comparison has been done considering data, data size, transformation, latency, processing, analysis and the paradigm. The researchers have concluded the study mentioning that the batch processing is more focusing on the volume and the stream processing is more focussed on the velocity. But at the end the method should be picked as per the users requirement.

The study [30] has introduced a micro-batch model for distributed join processing between streaming and stored big data. They have used the the micro-batch concept of distributed stream processing engines (SPEs), such as Apache Spark, and shared a comprehensive solution named DS-join for the distributed processing of the join. The micro-batch model is more fault-tolerant in a distributed setting and executes stream processing as a series of smaller batch processes. By utilising micro-batching, the DS-join lowers the amount of database requests. Additionally, the DS-join parallelises the join procedure, reduces data shuffle, manages a distributed SPE cache, and balances load between the SPE and external database system to optimise the join operation.

Another study is carried out to integrate batch and stream processing using IoT data in [31]. They have more on handling volume and velocity of the data by building a cloud architecture. That architecture consisted of 8 main modules. The data management, stream processing, streams analysis, stream data visualisation, batch processing, batch analytics, batch data visualisation and the data storage are the modules they have implemented in their system. They have identified several challenges when doing that and the main challenge was to identify the semantics of the computations from both streaming and batch data. And also, they have observed a considerable fluctuation of the accuracy when joining the results.

2.3 Summary

This chapter discussed the previous research studies carried out under real-time data warehousing. The chapter is mainly divided into 2 parts which are on ETL process and the stream data warehousing. To summarise, the traditional approach of the data warehousing was more towards the batch processing where the data will be persisted

at a certain period of the day in a batch where the system is having the minimum load. Most of the systems/data warehouses were unavailable during that back up time and the data processing was also done in batch wise. The most widely used approach for this batch data processing is MapReduce. The real-time data persistence approach was done for the streaming processing. In this approach, the challenge with that is associated with the data summarization task and the challenges are associated with the data velocity of the system. Furthermore, the ETL process is also important in the streaming data warehousing and that makes the process more efficient and more accurate.

CHAPTER 3
METHODOLOGY

As discussed in the previous chapters, the most difficult part of the real-time data warehouse is the ETL process from sources. Also, as per previous studies [26], 70% of the efforts in warehouse operations are spent on the ETL process. When it comes to real-time data warehousing this becomes more challenging.

- Row insertions should be the mainstay of continuous data updating activities.
- Updates must be executed concurrently considering the data generated from the streaming application execution, which will likely be requested considerably more frequently as it is a real-time data warehouse.

So, by considering all these factors, the methodology should cover the following key aspects.

1. Persist the recent changes of the streaming application to the database immediately/frequently as possible.
2. Minimising the impact of data update operations on the streaming application performance.
3. Ensuring the no down times in the data warehouse due to data backup.

By ensuring the above key concerns, this research intends to come up with the optimised architecture pattern for the components with the optimised real-time data warehouse to achieve performance improvements in a streaming analytics application.

The system has 3 main components.

- The ETL process module
- The real-time data warehouse
- Data extraction module

3.1 ETL Module

This module consists of two sub-modules.

1. Extraction and Transformation module.
2. Data Loading and Aggregation module.

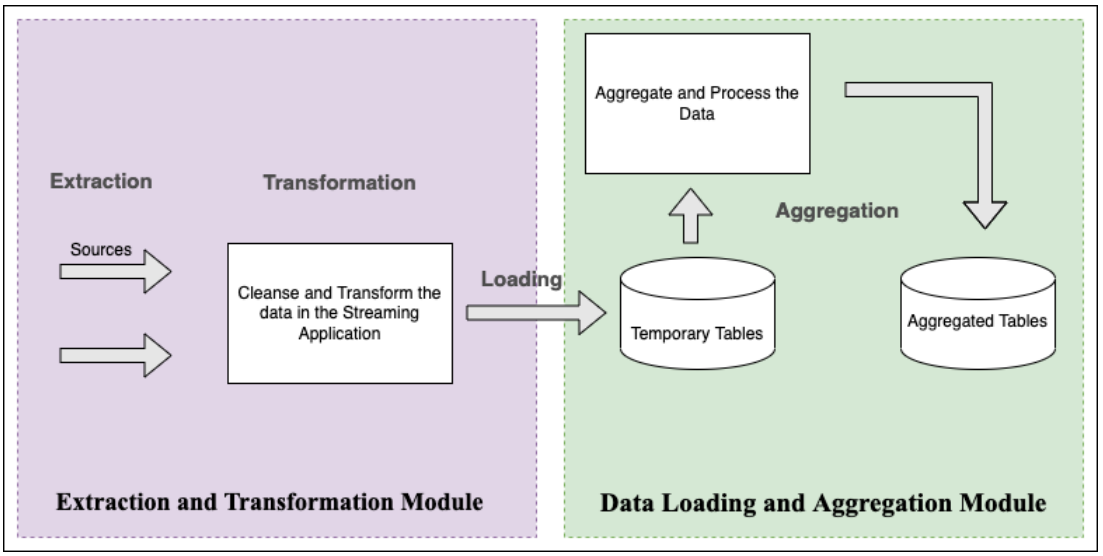

Figure 3.1: Overview of the ETL module

3.1.1 Extraction and Transformation Module (ETL)

The data generated from different sources, in different data formats are captured in this stage and direct to the streaming application. The data is in different formats like JSON events, CSV, Text, XML, and PassThrough messages where these data arrive from different feeds such as client applications, web services, file systems, etc.
This module is implemented using Siddhi, an open-source streaming query language [5].
First, to capture these events, the "*@source*" annotation of the Siddhi language [27] is used. By using "source" the data from external systems are consumed and turn them into events that can be handled by the streams internally. These events are used in further processing and cleansing operations.

Following is an example of the "*@store*" annotation for data extraction from a file system. The following is to extract data from an internal file having sensor data.

```
@source(type = 'file', mode='LINE', dir.uri =
"file:/home/methma/research/etl/fileInput/input",
tailing = "false",action.after.process = "move",
```

```
move.after.process =
"file:/home/methma/research/etl/fileInput/output",
dir.polling.interval = "50",
file.read.wait.timeout='3', timeout='2',
@map(type='csv',header='false',
@attributes(sensorID='0',timestamp='1',temperature='2',
fp = 'trp:file.path')))
define stream  dataStream (sensorID string, timestamp
string, temperature float, fp string);
```

Figure 3.2: Using @store annotation for extracting data from a file

Then the defined streams passed for the further processing and cleansing. For example, null values have handled in the cleansing stage. Assume a sales data scenario. In that, some events arrive with null values for the customerID attribute when processing sales data, “unknown” will be assigned in such cases.

```
@info(name = 'Handling null values store stream')
from salesStream
select ifThenElse(customerCode is NULL, "UNKNOWN",
customerCode) as customerCode, storeCode, PurchaseID,
BillValue, EventTimestamp
insert into cleansedStoreStream;
```

Figure 3.3: Pre-process null values

After pre-processing, the extracted and transformed data is persisted to the data warehouse as quickly as possible to achieve the near real-time statistics of the data. The most important part of the research is the next sub module.
The Siddhi Store annotation used in persisting data.

```
@store(type='rdbms',
jdbc.url="jdbc:mysql://localhost:3306/methma?useSSL=fal
se",
       username="root",
       password="root",
       jdbc.driver.name="com.mysql.jdbc.Driver")
define table TMP_SALES (customerCode string, storeCode
string, storeCode string, PurchaseID string, BillValue
long, EventTimestamp string);
. . .
/* Implement Pre-processing logics*/
. . .
@info(“insert data to temporary tables”)
from CleansedSalesStream
select customerCode, storeCode, storeCode, PurchaseID,
BillValue, EventTimestamp
insert into TMP_SALES;
```

Figure 3.4: Persist data in RTDW

3.1.2 Data Loading and Aggregation Module

The most important logic in this research is implemented in this module. Even though the suggested algorithm can be used in any use-case, the following simple use-case will be used in explaining the method further.

3.1.2.1 Use Case

Suppose there is a requirement to retrieve the stats for a store. For instance, let's consider 4 main tables *STORE, CUSTOMER, and SALES* table definitions as below. For simplification purposes, the data dimension is eliminated. Also, the primary keys are highlighted in bold.

STORE
StoreCode
StoreAddress
StorePhone
StoreEmail
EventTimestamp
…

ITEM
ItemCode
ItemDescription
ItemPrice
ItemStock
EventTimestamp
…

CUSTOMER
CustomerCode
CustomerName
CustomerPhone
CustomerAddress
EventTimestamp
…

SALES
StoreCode
CustomerCode
ItemCode
PurchaseID
BillValue
EventTimestamp

Figure 3.5: Sample table schema for the use-case

Let's say, the end goal of this system is to capture the statistics per store, per customer, per day, per month, and per year.

3.1.2.2 Aggregation Approach

In this proposed aggregation approach, there are two main types of tables in the database. One is the temporary tables and the others are summarised tables. For each summarised table set, there is one temporary table created which will consist of the data for a particular day. For example, let's take the sales table above.

TMP_SALES
StoreCode
CustomerCode
ItemCode
PurchaseID
BillValue
EventTimestamp
...

Figure 3.6: Sample table schema for temporary tables

As the first step, the pre-processed data in the Extraction and Transformation module will be directly persisted to a temporary table created in the real-time data warehouse (RTDW). All the data reached to the database will be *INSERT* without considering the primary key. Also, in this approach, the foreign keys were not considered to improve the performance of the insertion. Also, the *EventTimestamp* is recorded for each table which is very important in aggregations in the next steps. The *EventTimestamp* is recorded in Unix timestamp.

According to [27], several ETL programs make use of the *UPDATE ELSE INSERT* queries to load, which many perceive to be a performance bottleneck. With this suggested approach, each and every appending, removing or updating data operations of streaming applications persist only as new record insertions of the RTDW; reducing row, block, and table locks, as well as other concurrent data access issues. It is a given that INSERT operation is faster than the *DELETE* or *UPDATE*

operations as it is not associated with a database search to identify which row to update. So the simple *INSERT* operation is contributing more to the performance aspect of data loading. Furthermore, according to [27] many of the challenges are involved with the ETL process. ETL tools are used for more than just performance (as one might anticipate), but also for complexity, practicability, and cost. By persisting solely on record insertion operations to allow seamless data integration, and by employing empty or small-sized tables without any type of constraint or corresponding physical file, this technique ensures the fastest and easiest physical and logical support for attaining the performance requirements.

Because the only substantial change in the RTDW's physical and logical structure is the minor adaptation as seen in Figure 3.6, aggregation logic can be implemented in a way that maximises operability. Simple basic SQL instructions can be used to load data with the least amount of complexity. Because the relevant data is freely available, regardless of the ETL tools used, it is not necessary to create complex algorithms for modifying the data area.

On the other hand, keeping a huge amount of data in temporary tables is also a performance bottleneck. Also, it’ll be a challenge in data retrieval as well. Therefore, aggregation has been implemented.

The aggregation for all the temporary tables are executed at the end of each day when the system is having the least load. So, the temporary tables will keep data for only one day in the table.

For the completion of aggregation, DAY, MONTH, and YEAR granularity levels were introduced. A table from each granularity level is created/updated during the aggregation.

The following are aggregation tables for the example use-case.

SALES_Days	SALES_Months	SALES_Years
StoreCode	StoreCode	StoreCode
CustomerCode	CustomerCode	CustomerCode
ItemCode	ItemCode	ItemCode
PurchaseID	PurchaseID	PurchaseID
BillValue	BillValue	BillValue
EventTimestamp	EventTimestamp	EventTimestamp
…	…	…

Figure 3.7: Sample aggregation table schema for the "SALES" table

The GROUP BY is very important in the aggregation. In each aggregation, the records will be summarised according to the *GROUP BY* attributes which we can define as per required summarised statistics.

For example, let's take the *CustomerCode*, *StoreCode* and the *EventTimestamp* as the group by attributes as our end goal is to extract summarised data per store, per customer and per time duration.
The following is the "Days aggregation query" executed in the temporary sales table at the end of each day.

```
SELECT
    StoreCode,
    CustomerCode,
    ItemCode,
    PurchaseID,
    BillValue,
    EventTimestamp
FROM
    TMP_SALES
WHERE
```

```
   (PurchaseTimestamp >= {Starting time of the day} AND
PurchaseTimestamp < {starting time of the next day} )
GROUP BY
   StoreCode,
   CustomerCode,

UNIX_TIMESTAMP(from_unixtime(EventTimestamp/1000,"%Y-%m
-%d"))*1000));
```

Figure 3.8: Aggregation logic for the “Days” granularity tables

Likewise, all the data from the temporary tables will be aggregated to “Days” granularity tables at 00:00:00H each day. Then at the last day of the month at 00:00:00H time, the data from “Days” to “Month” will be aggregated.

In the same manner, at the end of the year, at 00:00:00 time, “Months” to “Years” data aggregation will be executed.

3.1.2.3 Data Purging

The data volume of a table has a huge impact on the performance of that database. It has been noticed during the performance tests as well. Therefore, it’s mandatory to have a data purging mechanism in the proposed system.

The data from the temporary tables will be aggregated at the end of the day. So, after that, that data does not have value. Hence, those can be purged from the temporary tables. To do that, the following DELETE query will be executed in the database after the completion of the aggregation.

```
DELETE
FROM
 {table_name}
WHERE
 EventTimestamp >= {Unix timestamp at 23:59H of the
last aggregated day};
```

Figure 3.9: Sample purging SQL

Likewise, purging is scheduled in other granularity tables after the completion of each aggregation of the “Months” and “Years” tables.

3.2 Real-Time Data Warehouse

Better performance of the data warehouse can be achieved by tuning multiple components of warehouse architecture. Such as

1. Improving the hardware - More memory and CPU allocation
2. Improving the physical storage schemas - Apply indexing to the “*GROUP BY*” attributes.
3. Query optimization - Parallel query execution and optimization, join algorithms, size estimation techniques and cost models, additional data structures such as indexes improving search efficiency, etc.

After studying the traditional data warehouse architecture [19] the proposed system will be evaluated using the ‘centralised architecture’ consisting of one centralised data warehouse which contains integrated data.

In a centralised architecture, the data will be extracted from various sources and after processing by ETL tools the data will be stored in a centralised data warehouse where all the reporting tools, analysis tools, OLAP tools, etc will directly retrieve the required statistical data from that centralised DB.

3.3 Statistics Retrieval

The required statistics are extracted from the relevant aggregation table. For example, let’s say that there is a requirement to extract the net income per store in the previous week.

```
SELECT StoreCode, Sum(BillValue) AS weeklyRevenue
FROM (SELECT StoreCode, BillValue FROM SALES_Days
WHERE PurchaseTimestamp >= {unixTimestamp of the first
date of the week})
```

```
UNION ALL
(SELECT StoreCode, BillValue
       FROM TMP_SALES
WHERE PurchaseTimestamp >= {unixTimestamp of the 0000h
today})
GROUP BY StoreCode
```

Figure 3.10: Retrieve total revenue per store in last week

If the requirement is to retrieve the recent data/statistics it is also possible with the temporary tables.

```
SELECT StoreCode,
          Sum(BillValue) AS revenueToday
FROM TMP_SALES
   WHERE PurchaseTimestamp >= {unix timestamp today at
00:00h}
   GROUP BY StoreCode
```

Figure 3.11: Retrieve daily revenue per store per day

This way, the data warehouse's most recent data will be processed with the help of this solution because this data is kept in the temporary replica tables, which are considered to be of the minimum size possible as those are keeping data of a day. This reduces the costs of executing current data queries in terms of CPU, memory, and I/O. Theoretically, this would allow for the delivery of the most up-to-date decision-making data, while the commercial transaction is in progress.

3.4 Performance Evaluation

The performance of the system is measured under 2 main aspects.

1. Performance evaluation for the JVM factors.

The heap and CPU utilisation was measured in both the streaming application side and the data warehouse comparing the suggested approach and the traditional approach where persisting data in the warehouse in batch wise. To capture the values, scheduled the periodic bash script execution to capture the data.

2. Performance evaluation for response time.

The response time is calculated as an average time using a simple java client where SQL queries can be passed. Following is a code snippet on calculating the average response time.

```
public class DbReasponceMeasure {
    . . .

        if (queryToRun == null) {
            queryToRun =
Constents.DEFAULT_QUERY_TO_EXECUTE;
        }
        if (configs.getProperty("SQL.ITERATIONS") !=
null) {
            iternations =
Integer.parseInt(configs.getProperty("SQL.ITERATIONS"))
;
        }

        for (int index = 0; index < iternations;
index++) {
            try {
                startTime = System.currentTimeMillis();
                statement =
connection.createStatement();
                tempResultset =
statement.executeQuery(queryToRun);
                if (tempResultset.next()) {
                    System.out.println("Query
```

```
Executed");
                }
                statement.close();
                completedTime =
System.currentTimeMillis();
                totalTime += (completedTime -
startTime);
                System.out.println("Time taken to
execute query : " + (completedTime - startTime) +
"ms.");
            } catch (SQLException e) {
                System.out.println("Unable to execute
query.");

. . .

System.out.println("Average time taken to execute query
: " + (totalTime * 1.0 / iternations) + "ms.");

. . .

}
```

Figure 3.12: Code snippet for the average response time calculator

3.5 Summary

The methodology chapter mainly focussed on how the proposed approach for the performance improved real time data warehouse was designed and implemented. It describes more information about sample use-case, how the events were pre processed and aggregated to offer the accurate real time statistics with better performance numbers.

CHAPTER 4
SYSTEM ARCHITECTURE AND IMPLEMENTATION

As discussed in the previous methodology, the proposed system consists of a real-time data warehouse for a data streaming application to achieve better performance and accuracy when retrieving the real-time statistics. There are 2 main components of the system named the ETL module and the RTDW module. The events from several sources will be extracted at the ETL module and complete the required pre-processing and data transformation tasks. After that ETL stage, the processed and transformed data will be loaded to the RTDW where all the aggregations will execute and summarise the data with a minimum impact on the performance of the overall system. In this chapter, the used architecture patterns and the implementation associated with them will be discussed.

4.1 Siddhi.io Stream Processing Architecture

In the proposed solution, "Siddhi.io" has been used in the data extraction and transformation module. Siddhi gives detailed operators for analytical functions, organises the flows of data, generates statistics, and identifies the relationships in real data received from numerous sources. This enables developers to create real-time data collection and analyse the data. Siddhi accepts events from a variety of sources and processes according to the logic implemented in the streaming applications using Siddhi, and the output is published to the event sinks that have been subscribed to. Siddhi can store and receive events from both in-memory tables and remote data stores such as RDBMS, MongoDB, the Hazelcast in-memory grid, and others.

The Siddhi query execution module is used in the extraction and Transformation module in this research. There can be 3 categories of Siddhi query runtime.

1. SingleInputStream Query Runtime
2. JoinInputStream Query Runtime
3. StateInputStream Query Runtime

4.1.1 SingleInputStream Query Runtime Architecture

For filter and window queries, this query runtime is generated. This consumes events from a window or a stream junction or and at the ProcessStreamReceiver, they transform the incoming events to the intended output stream format by deleting all irrelevant attributes coming from the arriving stream.

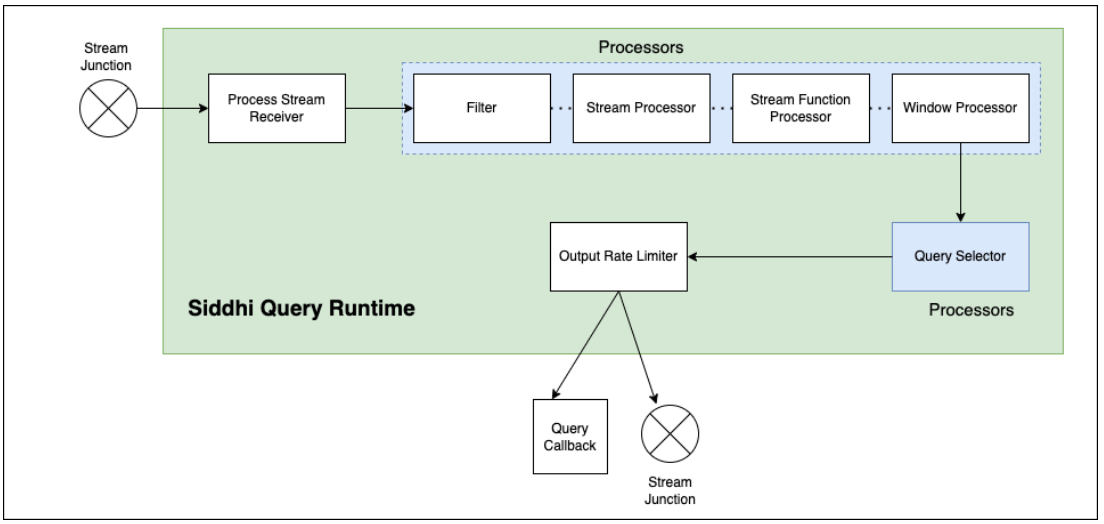

Figure 4.1: SingleInputStream Query Runtime

The transformed events are then pre-processed by passing them via a few Processors. The StreamProcessor, StreamFunctionProcessor, and StreamFunctionProcessor can then be extended. A QuerySelector must always be the last processor in a chain of processors, and each of them can only have one WindowProcessor. A WindowProcessor cannot be found in the query runtime's chain of processors when it receives events from a window.

An ExpressionExecutor that delivers a boolean value is used to implement the FilterProcessor. The Depth First Search technique is used to process expressions, which have a tree structure.

Let’s consider the below condition expression which can be used in data transformation.

```
value >= 200 and ( item == 'ABC' or item == 'DEF' )
```

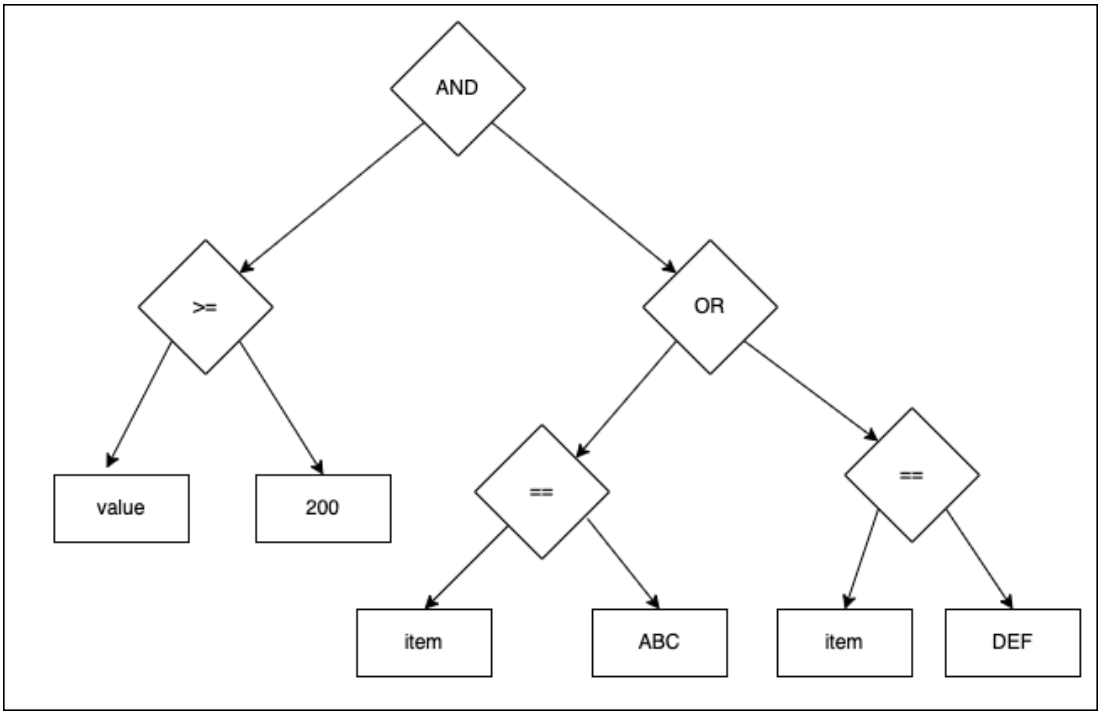

Figure 4.2: Filter expression execution in Siddhi

After processing, the events will be transformed further after reaching the QuerySelector. After that, the event will be reached to the OutputRateLimiter to control before publishing to the query callback or the stream junction.

4.1.2 JoinInputStream Query Runtime Architecture

For join queries, this runtime can be used. This can consume data from 2 separate stream junctions and do further processing. Also, can join against itself after consuming events from one stream junction, or against a table, a window, or an incremental aggregation. The Join operation will be initiated upon event arrival from the stream junction and proceed with further processing as discussed above.

4.1.3 StateInputStream Query Runtime Architecture

The state input stream query runtime is created for pattern and sequence queries. To ingest events from a single or many stream junctions, "ProcessStreamReceivers" are employed. The ProcessStreamReceiver and a set of basic processors are defined for each condition in the query. When one ProcessStreamReceiver receives events, it modifies the states by adding new events that were triggered by prior conditions. It generates a new state and updates it with the incoming event if the applied condition is the query's starting condition. Following that, the data is transmitted to basic processors for filtering and processing.

By using the Siddhi streams, the data extraction and transformation module is implemented and sample implementation code snippets have been shared in the previous methodology chapter.

4.2 Aggregation Algorithm

The algorithm used in aggregation will be summarised in this section as a more descriptive explanation is available in the methodology chapter.

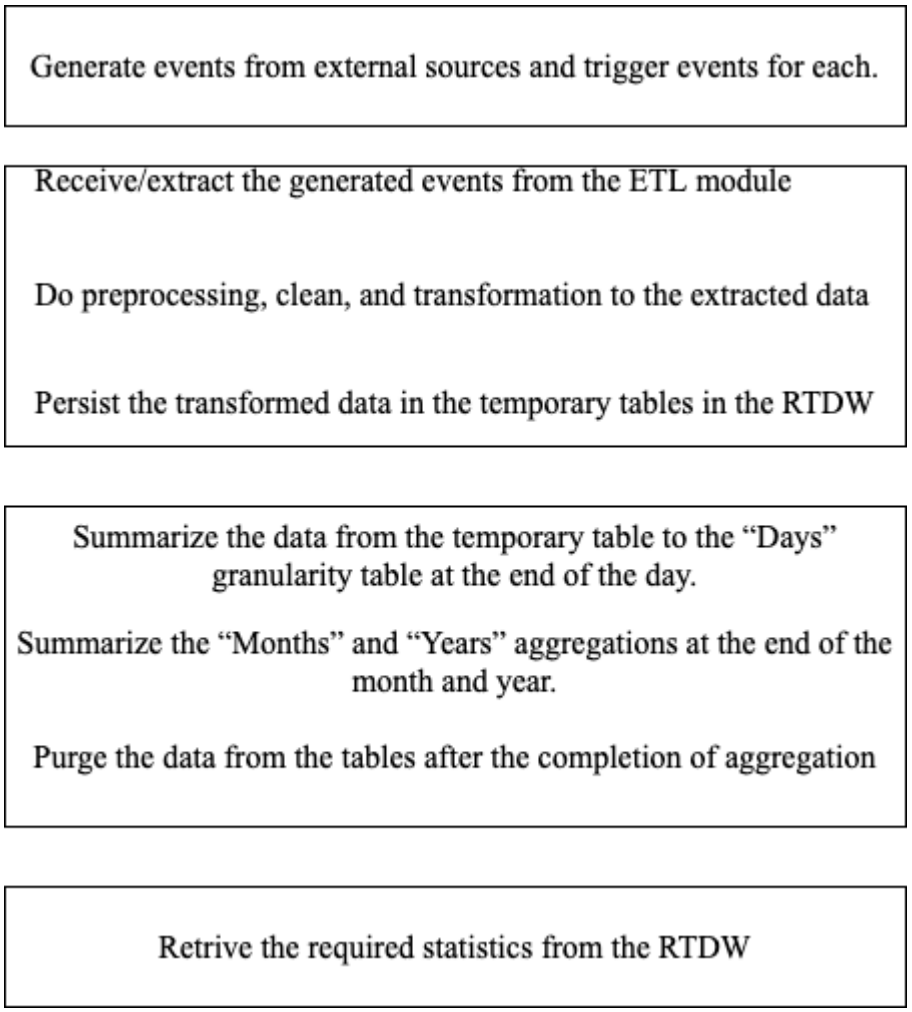

Figure 4.3: Summarised aggregation flow

4.3 Centralised DB Architecture

A centralised database is a database that is kept, managed, and updated in one place. This type of data warehouse is only modified and managed by accessing that centralised location. This location is accessed via an internet connection (such as LAN, WAN, etc). The principal consumers of this centralised database are institutions and organisations.

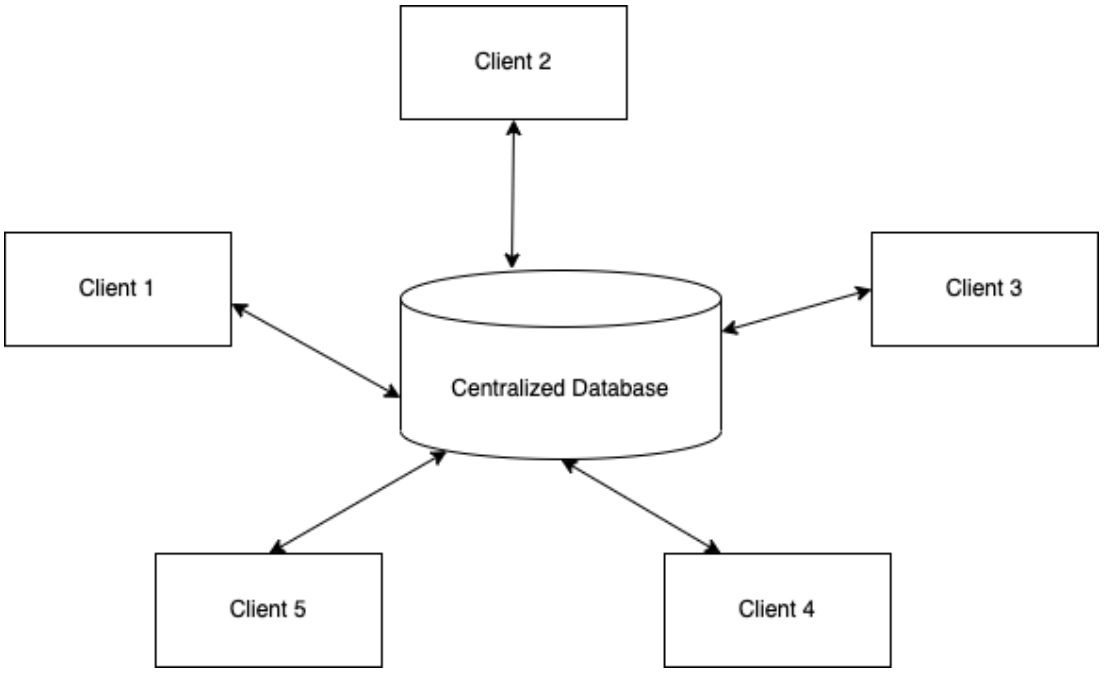

Figure 4.4: Centralised DB architecture

The centralised database has the minimum redundancy as all the data is stored in a centralised location and not scattered across several locations. Also, the increased integrity of data is high as the data persisted at a single physical location which makes it easier to access and ensures the accuracy and consistency of the data. Due to the less maintenance cost, this architecture is cost-effective. A single point of entry makes the system simple and easier to make changes to the data warehouse. The efficiency of the operations in the data warehouse is increased as the data is controlled by a centralised unit and resource utilisation is possible.

So, by considering all these factors, I decided to use a centralised database architecture for this research. I have learned that the distributed databases or no SQL databases will improve the performance further when it comes to a higher volume of data. So the ideal data warehouse would be a clustered data warehouse for more performance improvements.

4.4 Real-World Sales RTDW Schema

The following schema is created in the real-time data warehouse according to the sample use case discussed in the methodology chapter. The GROUP BY fields are highlighted in bold letters.

TMP_STORE
StoreCode
StoreAddress
StorePhone
StoreEmail
EventTimestamp

STORE_Days
StoreCode
StoreAddress
StorePhone
StoreEmail
EventTimestamp

STORE_Months
StoreCode
StoreAddress
StorePhone
StoreEmail
EventTimestamp

STORE_Years
StoreCode
StoreAddress
StorePhone
StoreEmail
EventTimestamp

TMP_ITEM
ItemCode
ItemDescription
ItemPrice
ItemStock
EventTimestamp

ITEM_Days
ItemCode
ItemDescription
ItemPrice
ItemStock
EventTimestamp

ITEM_Months
ItemCode
ItemDescription
ItemPrice
ItemStock
EventTimestamp

ITEM_Years
ItemCode
ItemDescription
ItemPrice
ItemStock
EventTimestamp

TMP_CUSTOMER
CustomerCode
CustomerName
CustomerPhone
CustomerAddress
EventTimestamp

CUSTOMER_Days
CustomerCode
CustomerName
CustomerPhone
CustomerAddress
EventTimestamp

CUSTOMER_Months
CustomerCode
CustomerName
CustomerPhone
CustomerAddress
EventTimestamp

CUSTOMER_Years
CustomerCode
CustomerName
CustomerPhone
CustomerAddress
EventTimestamp

TMP_SALES
StoreCode
CustomerCode
ItemCode
PurchaseID
BillValue
EventTimestamp

SALES_Days
StoreCode
CustomerCode
ItemCode
PurchaseID
BillValue
EventTimestamp

SALES_Months
StoreCode
CustomerCode
ItemCode
PurchaseID
BillValue
EventTimestamp

SALES_Years
StoreCode
CustomerCode
ItemCode
PurchaseID
BillValue
EventTimestamp

Figure 4.5: Sample schema used in the store use case

4.5 Summary

This chapter elaborates more details about the overall system architecture along with the architecture of selected technologies such as Siddhi, the aggregation approach and the centralised database architecture to provide more implementation details on the final system and selected technologies.

CHAPTER 5
EVALUATION

The aim of this chapter is to share the results after the evaluation of the proposed approach. To evaluate the system, the data warehouse schema shown in figure 4.5 for the sample use case is used.

The size of the RTDW is around 7 GB with the 1 year old data as presented in the below figure 5.1.

TABLE_NAME	table_rows	Size in MB
CUSTOMER_Days	85000	423.02
CUSTOMER_Months	138000	724.08
CUSTOMER_Years	832	6.11
ITEM_Days	78056	389.02
ITEM_Months	119476	528.30
ITEM_Years	983	8.85
SALES_Days	121598	646.02
SALES_Months	1085130	2365.05
SALES_Years	378029	879.47
STORE_Days	894	78.02
STORE_Months	6751	192.17
STORE_Years	241	8.05
TMP_CUSTOMER	26245	345.16
TMP_ITEM	2149	10.89
TMP_SALES	4495	27.05
TMP_STORE	23	0.05

Figure 5.1: Table schema of the evaluated system

To implement this, a MySQL community server 8.0.25 on a 4 core, 8 GB RAM and 40 GB root disk machine specifications were used.

A set of 10 queries was chosen to capture the results in decision-making. These inquiries display an example of usual data for decision-making, daily, monthly, quarterly, and annual values for customer product and promotion sales. This group of queries simulates the usual workload in the real world. with a variety of database joins, filtering, grouping, and sorting operations performed on a combination of recent and historical business data.

The performance is evaluated measuring the heap and CPU utilisation of the streaming application and the database. For that, a bash script was scheduled to capture the values using the process id. Apart from that, the response time is measured using a simple java client which calculates the average response time for a particular query.

5.1 Performance evaluation for the JVM factors.

The following bash script is used in capturing CPU and Heap utilisation of components.

```
#!/bin/bash

# Set the threshold according to no of cores in the
server.
trigger=4.00

load=`cat /proc/loadavg | awk '{print $1}'`
response=`echo | awk -v T=$trigger -v L=$load 'BEGIN{if
( L > T){ print "greater"}}'`

# Set used, available memory from physical memory.
mem_used=`free -h | head -2 |tail -1 |awk '{print $3}'`
mem_aval=`free -h | head -2 |tail -1 |awk '{print $7}'`

# Set java process id and other parameters. Set the
JAVA_HOME.
pid=$(pgrep -f /usr/lib/java/jdk1.8.0_131);
count=10
interval=5s

# Set heap usage values and other parameters.

## The heap usage log file will be created in this
location.
tmpdir="/home/methma/Desktop/tmp"
tmpfile="/home/methma/Desktop/tmp/heapusage_full"

## Capture the heap values
currentHeapUsage=$(/usr/lib/java/jdk1.8.0_131/bin/jstat
-gc $(/bin/cat
/home/methma/research/worker/runtime.pid) | tail -n 1 |
/usr/bin/awk '{split($0,a," ");
sum=(a[3]+a[4]+a[6]+a[8]+a[10]+a[12])/1024; print sum"
MB"}' | /usr/bin/cut -d " " -f 1 | /usr/bin/cut -d "."
-f 1)
```

```
totalAllocatedHeap= 4096

# Writing current heap usage value to file.

if [ -d "$tmpdir" ]
then
        echo  "$(date +%F-%T) currentHeap:
$currentHeapUsage totalHeap: $totalAllocatedHeap"
cpu/la: $load memUsed: $mem_used memAval: $mem_aval >>
$tmpfile
else

        mkdir $tmpdir
        echo  "$(date +%F-%T) currentHeap:
$currentHeapUsage totalHeap: $totalAllocatedHeap"
cpu/la: $load memUsed: $mem_used memAval: $mem_aval >>
$tmpfile
fi
```

Figure 5.2: Code snippet of the bash script used in capturing JVM stats

Furthermore, to understand the performance gain of the suggested approach I have compared the performance against the traditional approach without persisting the data immediately. In the tested traditional approach, the generated data was kept in the streaming application and persisted that data in the database at the end of the day. Then, the heap and the CPU consumption was captured in both the streaming application side and the data warehouse end while executing the data insertion and retrieval queries.

5.1.1 Streaming Application

5.1.1.1 Heap utilisation against TPS in the Streaming application

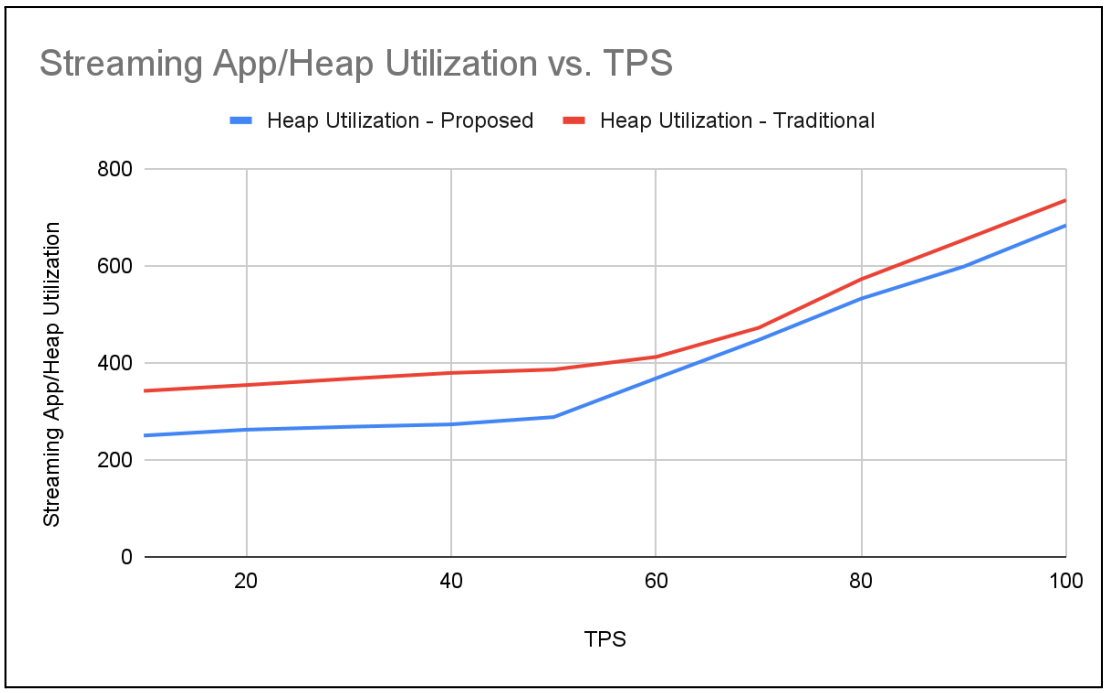

Figure 5.3: Heap utilisation against TPS in the Streaming application

5.1.1.2 CPU utilisation against TPS in the Streaming application.

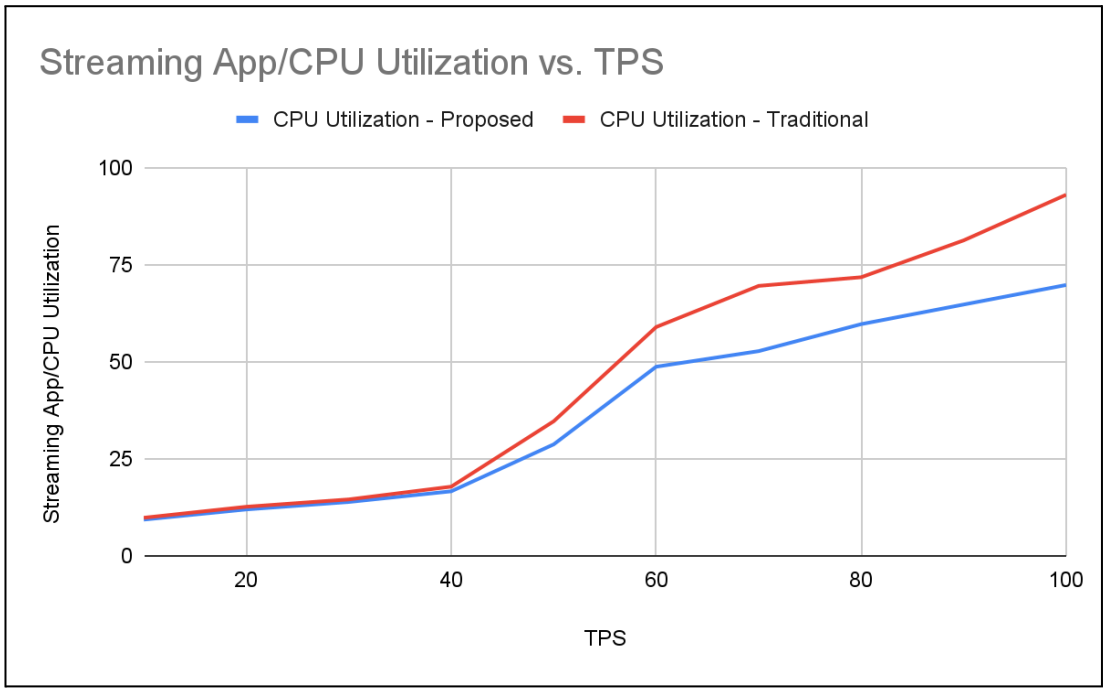

Figure 5.4: CPU utilisation against TPS in the Streaming application.

5.1.2 Data Warehouse

5.1.2.1 Heap utilisation against TPS in the RTDW.

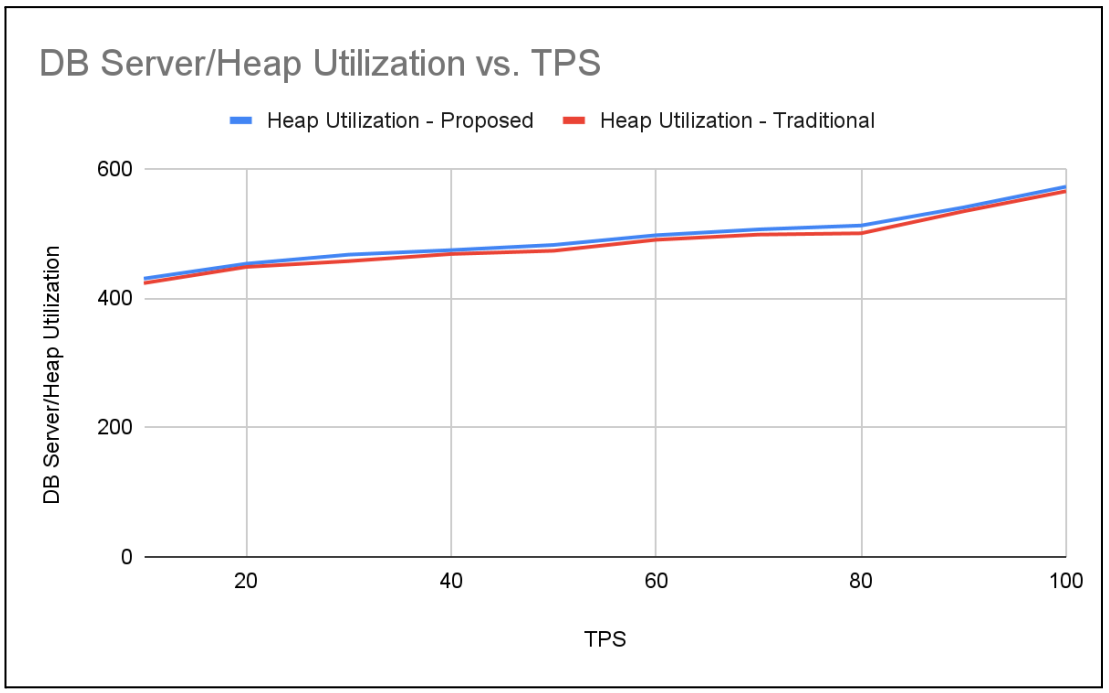

Figure 5.5: Heap utilisation against TPS in the RTDW.

5.1.2.2 CPU utilisation against TPS in the RTDW

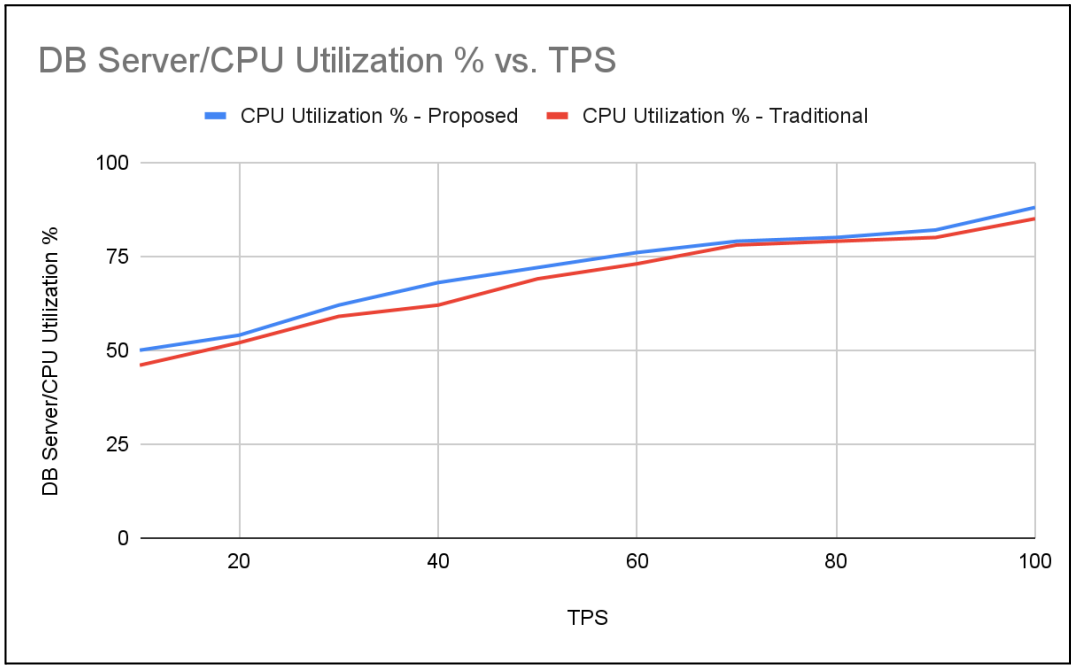

Figure: 5.6 CPU utilisation against TPS in the RTDW

The above performance graph shows a considerable performance improvement of heap and CPU utilisation at the streaming application side in the suggested approach. Also, the heap and CPU utilisation was okay with the suggested approach even though we are persisting data in the data warehouse immediately.

5.2 Performance evaluation for response time.

The average response time was captured using a simple Java client. Following is the average response time in seconds while simultaneous users are retrieving the summarised data from the RTDW. During the data retrieval, the data insertion also occurred in different TPS values as presented in figure 5.7 below.

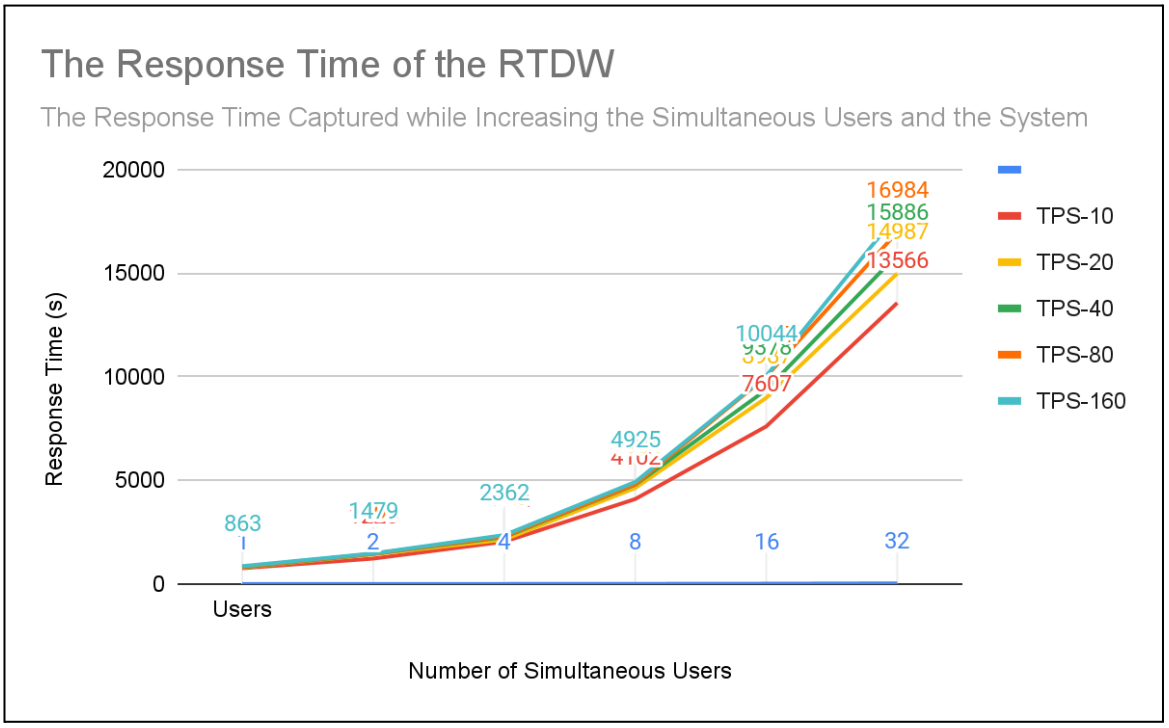

Figure 5.7: Average response time against simultaneous users for different TPS values

For evaluation the simultaneous user count increased from 1, 2, 4, 8, 16 to 32 while generating events in 10, 20, 40, 80, 160 TPS (transactions per second). It can be seen that the response time is considerable response slowness when TPS and the user

count increase. At the same time, the heap and CPU utilisation was increased with the TPS increment. But, no out-of-memory or unresponsiveness was observed.

So to conclude the performance evaluation, the suggested model achieved better performance than the traditional ETL methods such as persisting the data batch wise. Further more, by considering the test results, it can be seen that a greater performance was observed during the lower simultaneous users and the system resources should be scaled up with the increment of the simultaneous users. Hence it can be concluded that we can gain a considerable performance gain with adequate infrastructure resources according to the use case after a proper performance test.

CHAPTER 6
CONCLUSION

Developing a real-time data warehouse (RTDW) which is providing real-time statistics has been an emerging research area for several years with the emerging requirement for real-time statistics in management decisions in organisations. For some business areas, business owners should act immediately after completion of the transactions instead of on a daily or weekly basis.

When implementing the RTDW, performance is a key concern which needs to be considered while thinking of providing real-time data. Therefore, as a solution to overcome these concerns, this paper proposed a system for RTDW.

6.1 Research Contribution

The proposed research allows retrieving the real-time data from the data warehouse while ensuring there are no down times in the data warehouse due to data backup. This is accomplished through data structure replication and an improved approach to data aggregation. This approach has reduced the performance bottlenecks in the data warehouse and the streaming analytics application side as it proved from the test results.

Other than that, this concept offers the benefits of smooth real-time integration without removing any ETL technique presently in use by the business data warehouse. It just employs standard INSERT queries to take advantage of the database's fastest data insertion methods.

This enables it to state that, according to the test, the apparent cost of enabling this suggested RTDW approach is relatively modest and acceptable to its consumers.

6.2 Limitation of the Research Approach

Mainly, the hardware expenses associated with the storage data appear to be the major disadvantage of this suggested method, but when considering the benefits associated with enabling real-time data warehousing, these prices are reasonable.

Other than that, it is a known issue in some databases (eg: MySQL, MS SQL), the table space is not reclaimed after the purging until database optimization [28]. Since this approach involves frequent data purging, this data reclaiming issue can arise. So

as a workaround for this, scheduled database optimizations can be suggested according to the data volume.

6.3 Future Works

In future work, the aggregation module can be improved more with high performing join operations such as mesh join or hybrid join to improve the performance factors more. Other than that, the RTDW can be improved to the NoSQL database which is providing more support for a large volume of data to improve the performance capabilities. It is required to do a proper performance test on this approach in the NoSQL databases to understand exact performance numbers. In addition to that, the database architecture also can improve more with a clustered data warehouse where we can improve the performance power of the database. Apart from all these factors, we need to identify the performance behaviour of this approach when the temporary tables are created in memory. This approach might cause high heap utilisation issues with the large data load. But at the same time, we might be able to achieve a certain performance gain by reducing the data insertion time. Therefore, proper research should be carried out on this.

REFERENCES

[1]. N. Fatima, "Data Warehouse Architecture: Types, Components, & Concepts | Astera", Astera, 2020. [Online]. Available: https://www.astera.com/type/blog/data-warehouse-architecture/. [Accessed: 03- Dec- 2020].

[2]. A. Hajmoosaei, M. Kashfi, and P. Kailasam, "Comparison plan for data warehouse system architectures," The 3rd International Conference on Data Mining and Intelligent Information Technology Applications, Macao, 2011, pp. 290-293.

[3]. M. Obalı, B. Dursun, Z. Erdem and A. K. Görür, "A real time data warehouse approach for data processing," 2013 21st Signal Processing and Communications Applications Conference (SIU), Haspolat, 2013, pp. 1-4, DOI: 10.1109/SIU.2013.6531245.

[4]. Oracle.com, 2020. [Online]. Available: http://www.oracle.com/us/products/middleware/data-integration/realtime-data-warehousing-bp-2167237.pdf. [Accessed: 03- Dec- 2020].

[5]. "siddhi-io/siddhi", GitHub, 2020. [Online]. Available: https://github.com/siddhi-io/siddhi. [Accessed: 03- Dec- 2020].

[6]. "Data warehouse", *En.wikipedia.org*, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Data_warehouse. [Accessed: 27- Feb- 2021]

[7]. D. M. Tank, A. Ganatra, Y. P. Kosta and C. K. Bhensdadia, "Speeding ETL Processing in Data Warehouses Using High-Performance Joins for Changed Data Capture (CDC)," 2010 International Conference on Advances in Recent Technologies in Communication and Computing, Kottayam, India, 2010, pp. 365-368, DOI: 10.1109/ARTCom.2010.63.

[8]. E. Mehmood and T. Anees, "Performance Analysis of Not Only SQL Semi-Stream Join Using MongoDB for Real-Time Data Warehousing," in IEEE Access, vol. 7, pp. 134215-134225, 2019, doi: 10.1109/ACCESS.2019.2941925.

[9]. H. Mallek, F. Ghozzi, O. Teste, and F. Gargouri, ‘‘BigdimETL with NoSQL database,’’ Procedia Comput. Sci., vol. 126, pp. 798–807, 2018.

[10]. D. M. Tank, A. Ganatra, Y. P. Kosta and C. K. Bhensdadia, "Speeding ETL Processing in Data Warehouses Using High-Performance Joins for Changed Data Capture (CDC)," 2010 International Conference on Advances in Recent Technologies in Communication and Computing, Kottayam, India, 2010, pp. 365-368, doi: 10.1109/ARTCom.2010.63.

[11]. M. Asif Naeem, Gillian Dobbie, and G. Webber, "An Event-Based Near Real-Time Data Integration Architecture," in Enterprise Distributed Object Computing Conference Workshops, 2008, pp. 401-404.

[12]. L. Golab and T. Johnson, "Data stream warehousing," 2014 IEEE 30th International Conference on Data Engineering, Chicago, IL, USA, 2014, pp. 1290-1293, doi: 10.1109/ICDE.2014.6816763.

[13]. R. J. Santos, J. Bernardino and M. Vieira, "24/7 Real-Time Data Warehousing: A Tool for Continuous Actionable Knowledge," 2011 IEEE 35th Annual Computer Software and Applications Conference, Munich, Germany, 2011, pp. 279-288, doi: 10.1109/COMPSAC.2011.44.

[14]. M. A. Naeem, "A robust join operator to process streaming data in real-time data warehousing," Eighth International Conference on Digital Information Management (ICDIM 2013), Islamabad, Pakistan, 2013, pp. 119-124, doi: 10.1109/ICDIM.2013.6693964.

[15]. "Siddhi IO File", *Siddhi-io.github.io*, 2021. [Online]. Available: https://siddhi-io.github.io/siddhi-io-file/. [Accessed: 02- Mar- 2021]

[16]. Types of Data Extraction Models | Data Warehouse Information Center", Data Warehouse Information Center, 2021. [Online]. Available: https://datawarehouseinfo.com/data-warehouse-data-extraction-models/. [Accessed: 02- Mar- 2021]

[17]. Tok, W. H., & Bressan, S. (2013). Progressive and Approximate Join Algorithms on Data Streams. Intelligent Systems Reference Library, 157–185. doi:10.1007/978-3-642-28323-9_7

[18]. Purestorage.com, 2021. [Online]. Available: https://www.purestorage.com/au/knowledge/what-is-active-active.html. [Accessed: 02- Mar- 2021]

[19]. Data Warehousing - Architecture - Tutorialspoint", Tutorialspoint.com, 2021. [Online]. Available: https://www.tutorialspoint.com/dwh/dwh_architecture.htm. [Accessed: 02- Mar- 2021]

[20]. Blazic, G., Poscic, P., & Jaksic, D. (2017). Data warehouse architecture classification. 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). doi:10.23919/mipro.2017.7973657

[21]. Wrembel, R. (2012). Data Warehouse Performance: Selected Techniques and Data Structures. Lecture Notes in Business Information Processing, 27–62. doi:10.1007/978-3-642-27358-2_2

[22]. A. Delis and N. Roussopoulos, "Performance comparison of three modern DBMS architectures," in IEEE Transactions on Software Engineering, vol. 19, no. 2, pp. 120-138, Feb. 1993, doi: 10.1109/32.214830.

[23]. M. A. Naeem, F. Mirza, H. U. Khan, D. Sundaram, N. Jamil and G. Weber, "Big Data Velocity Management–From Stream to Warehouse via High Performance Memory Optimized Index Join," in IEEE Access, vol. 8, pp. 195370-195384, 2020, doi: 10.1109/ACCESS.2020.3033464.

[24]. S. Kim and K. Song, "Implementation of a distributed processing engine for spatial big-data processing based on batch and stream," 2017 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Korea (South), 2017, pp. 1196-1198, doi: 10.1109/ICTC.2017.8190896.

[25]. E. Mehmood and T. Anees, "Challenges and Solutions for Processing Real-Time Big Data Stream: A Systematic Literature Review," in IEEE Access, vol. 8, pp. 119123-119143, 2020, doi: 10.1109/ACCESS.2020.3005268.

[26]. Kimball, R., Caserta, J.: The Data Warehouse ETL Toolkit, Wiley Computer Pub., 2004.

[27]. "Query Guide - Siddhi", Siddhi.io, 2022. [Online]. Available: https://siddhi.io/en/v5.1/docs/query-guide/#source. [Accessed: 25- Mar- 2022].

[28]. M. table, S. Deo and L. Martins, "MySQL InnoDB not releasing disk space after deleting data rows from table", *Stack Overflow*, 2022. [Online]. Available: https://stackoverflow.com/questions/1270944/mysql-innodb-not-releasing-disk-space-after-deleting-data-rows-from-table. [Accessed: 28- Mar- 2022].

[29]. S. Benjelloun et al., "Big Data Processing: Batch-based processing and stream-based processing," 2020 Fourth International Conference On Intelligent Computing in Data Sciences (ICDS), 2020, pp. 1-6, doi: 10.1109/ICDS50568.2020.9268684.

[30]. Y. Jeon, K. Lee and H. Kim, "Distributed Join Processing Between Streaming and Stored Big Data Under the Micro-Batch Model," in IEEE Access, vol. 7, pp. 34583-34598, 2019, doi: 10.1109/ACCESS.2019.2904730.

[31]. H. Cao, M. Brown, L. Chen, R. Smith and M. Wachowicz, "Lessons Learned from Integrating Batch and Stream Processing using IoT Data," 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), 2019, pp. 32-34, doi: 10.1109/IOTSMS48152.2019.8939232.

APPENDIX A

CD/DVD