Lake Data Warehouse Architecture for Big Data Solutions

Emad Saddad[1]
Climate Change Information Center and Renewable Energy and Expert System
Agricultural Research Center (ARC)
Giza, Egypt

Ali El-Bastawissy[2]
Faculty of Computer Science
MSA University
Giza, Egypt

Hoda M. O. Mokhtar[3]
Faculty of Computers and Artificial Intelligence
Cairo University
Giza, Egypt

Maryam Hazman[4]
Climate Change Information Center and Renewable Energy and Expert System
Agricultural Research Center (ARC), Giza, Egypt

***Abstract*—Traditional Data Warehouse is a multidimensional repository. It is nonvolatile, subject-oriented, integrated, time-variant, and non-operational data. It is gathered from multiple heterogeneous data sources. We need to adapt traditional Data Warehouse architecture to deal with the new challenges imposed by the abundance of data and the current big data characteristics, containing volume, value, variety, validity, volatility, visualization, variability, and venue. The new architecture also needs to handle existing drawbacks, including availability, scalability, and consequently query performance. This paper introduces a novel Data Warehouse architecture, named *Lake Data Warehouse Architecture*, to provide the traditional Data Warehouse with the capabilities to overcome the challenges. *Lake Data Warehouse Architecture* depends on merging the traditional Data Warehouse architecture with big data technologies, like the Hadoop framework and Apache Spark. It provides a hybrid solution in a complementary way. The main advantage of the proposed architecture is that it integrates the current features in traditional Data Warehouses and big data features acquired through integrating the traditional Data Warehouse with Hadoop and Spark ecosystems. Furthermore, it is tailored to handle a tremendous volume of data while maintaining availability, reliability, and scalability.**

Keywords—Traditional data warehouse; big data; semi-structured data; unstructured data; novel data warehouses architecture; Hadoop; spark

I. Introduction

Data warehouse (DW) has many benefits; it enhances Business Intelligence, data quality, and consistency, saves time, and supports historical data analysis and querying [1]. In the last two decades, data warehouses have played a prominent role in helping decision-makers. However, in the age of big data with the massive increase in the data volume and types, there is a great need to apply more adequate architectures and technologies to deal with it.

Therefore, it became crucial to enhance traditional DW to deal with big data in various fields to accommodate this evolution in volume, variety, velocity, and veracity of big data [2], [3]. To achieve this, we propose a new DW architecture called *Lake Data Warehouse Architecture*. *Lake Data Warehouse Architecture* is a hybrid system that preserves the traditional DW features. It adds additional features and capabilities that facilitate working with big data technologies and tools (Hadoop, Data Lake, Delta Lake, and Apache Spark) in a complementary way to support and enhance existing architecture.

Our proposed contribution solve several issues that face integrating data from big data repositories such as:

- Integrating traditional DW technique, Hadoop Framework, and Apache Spark.
- Handling different data types from various sources like structured data (DBs, spreadsheet), semi-structured data (XML files, JSON files), and unstructured data (video, audio, images, emails, word, PowerPoint, pdf files).
- Capturing, storing, managing, and analyzing data volume that cannot be handled by traditional DW.
- Using recent technologies like the Hadoop framework, Data Lake, Delta Lake, and Apache Spark to decrease time spent analyzing data and to decrease storage costs are inexpensive.
- Support all users, especially data scientists, because they need to perform depth data analysis.

The rest of the paper is organized as follows: Section II explains background and preliminaries for traditional Data Warehouses and its limitations, importance of DWs, and big data characteristics and types. Section III overviews the related works to the DW architectures. Section IV presents the proposed DW architecture named Lake Data Warehouse Architecture. Section V describes the case study by applying our contributions called *Lake DW Architecture*. Finally, the conclusion is presented in Section VI.

II. Background and Preliminaries

In this section, we will review the background of related topics, such as traditional DWs, its limitations, and why we need to redevelop it. Moreover, we will discuss various related technologies.

A. *Traditional Data Warehouses(DWs)*

Traditional DWs are integrated, a subject-oriented, nonvolatile, and time-variant data to support decision-makers [4], as presented in Fig. 1. Those four properties differentiate DWs from other data repository systems, such as relational database systems [5].

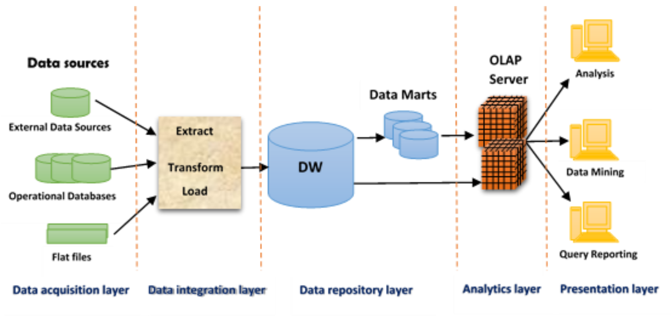


Fig. 1. Traditional Data Warehouses (DWs) architecture

Usually, in primary DW, there are data marts (DMs). Data marts (DMs) are a small DW that contains only a subset of data obtained from a central DW. The content of DMs represents information from a specific domain of interest. Many DWs servers are used to manage data. These servers present multidimensional views of data to a variety of front-end tools.

B. *The limitations of Traditional DW Architecture*

In [6], [7], and [8], the authors reviewed some limitations of DW, such as supporting only structured data, on the contrary, do not support semi-structured data, or unstructured data. In addition to the above, the restriction of handling data volume in terabytes, which does not scale to petabyte size, is widely available. Besides, it is costly as it depends on proprietary hardware and software.

Moreover, traditional DW performs analytic queries, which consequently affects the whole query performance, accessing, and processing of data. The decision-making process also may be affected if: 1) the correct data are not available at a suitable time, and 2) the growth of the business requires new methods for data management other than adapting traditional DW architecture.

C. *The Objectives of the redeveloped traditional DW*

One of our main objectives is to overcome the limitations of traditional DW architecture built on outdated technologies [4]. We initiate an overall architecture that supports the functionalities of the traditional DW with abilities to include [6], [9] :

- Meeting new business requirements.
- Depending on lower-cost infrastructure.
- Handling heterogeneous data and new data structures and formats.
- Managing customer expectations.
- Meeting growth of the business.
- Using advances in new technology and its improvements.
- Handling product existence and status.
- Improving business efficiencies and competitive advantage.
- Decreasing operational and financial risks.
- Evaluating and forecasting trends and behaviors.

D. *The Age of Big data*

Big Data is a data volume that is available in different levels of complication. It is generated at various velocities and levels of uncertainty; hence it is not handled using traditional approaches, traditional technologies, or traditional algorithms [6]. Today, big data is characterized by ten main characteristics namely volume, variety, velocity, veracity, value, validity, variability, visualization, volatility, and venue [3], [10], [2], [11], [12], [13] as follows:

- *Volume*: the huge amount of data generated continuously on an hourly or a daily basis from different sources. Such as terabytes generated per hour for applications like YouTube, Instagram, and Google.
- *Variety*: the types of big data that are ingested from different data sources.
- *Velocity*: the speed at which data is produced. The aspects of data may be batch data or streaming data.
- *Veracity*: the quality of data that is being handled to obtain valuable insights. Such as ambiguous, inconsistent, incomplete, anomaly, uncertain, and biased data.
- *Value*: represents the business value to be derived from big data.
- *Validity*: refers to the correctness of data used to extract outputs in the form of information.
- *Variability*: refers to the inconsistent data flow.
- *Visualization*: refers to the ability to analyze and visual insights as an output of big data analysis.
- *Volatility*: the stored data on how long it is valuable to the consumer.
- *Venue*: refers to a various platform where numerous kinds of data from different sources by several platforms.

In general, data are a set of qualitative values or quantitative variables; Big Data can be categorized into three types [2], [9]:

- *Structured data*. The data has a defined structure or a schema organized either in the form of a relational database or in some other way that is easy to operate. For example, data stored in a relational database (in the form rows and columns), in spreadsheets (such as CSV files), and cleansed data (that have been processed with a lot of cleansing and filtering).

- *Semi-Structured Data*: The data is hard to retrieve, analyze, and store as structured data. It requires a big data software framework (such as Apache Hadoop) to achieve these operations. For example, XML files, JSON files, and BibTex files.
- *Unstructured Data*: The fully unorganized data is difficult to handle, and it requires advanced software and tools to access it. Examples include video, audio, images, emails, word, PowerPoint, pdf files, webpages, location coordinates, and streaming data.

E. Hadoop Framework and Data Lake

Hadoop is an open-source software framework. It allows the execution of the MapReduce processes for data processing. It provides massive storage for all data types, massively parallel processing, and storing data and running applications on clusters to accomplish better computation resources [14]. Hadoop has main components as follows [15], [16], [17]:

- *The Hadoop Distributed File System (HDFS)* is a file system, which manages storage and access to data spread across the different nodes of a Hadoop cluster.
- *YARN* is a Hadoop cluster resource manager used to assign system resources for applications and schedule of the jobs.
- *Map-Reduce* is a processing engine and a programming framework used to manage large-scale batch data in the Hadoop system.
- *Hadoop Common* is a set of services and institutions that provide underlying capabilities needed by the other parts of Hadoop.

Data Lake is a data store that can collect any type of data: structured, semi-structured, or unstructured data, which are stored with one another regardless of structure, format, or types [18], [19]. It is a conceptual idea that is usually implemented with one or more technologies such as Hadoop and NoSQL databases. When querying the Data Lake, only need data will transform that are relevant to business needs [20].

Creating Data Lake depends on Hadoop's technology, which is a component (as the platform) for the data lake. It is the complementary relationship between Data Lake and Hadoop [21], [22], [23]

Data Lake is similar to traditional DW in that they are both repositories for data. However, there are apparent differences in features between them [7]—the schema on reading in Data Lake, but schema on write in DW. The scale of data in Data Lake is enormous, while it is large data volumes in DW. The data sources may be semi-structured data or/and unstructured data, but it is mainly structured data in DW [24], [25], [26], [27].

F. Apache Spark and Delta Lake

Apache Spark is an open-source applied for big data analytics and distributed systems. It provides streaming libraries, SQL, graph analysis, and machine learning. It has two main components Spark streaming, which is used for managing real-time data, and the Spark engine , which directly processes each data chunks by Spark streaming [28], [29], [30].

Delta Lake is an open-source Spark storage layer. It is an extra storage layer that makes reliability to our data lakes built on The Hadoop Distributed File System (HDFS) and cloud storage [31]. Delta Lake provides a series of other features including:

- Joining streaming and batch data processing.
- Giving a scalable metadata approach.
- Providing ACID transactions that guaranteed consistency of the data stored inside the data lake through ensuring that only complete writes are committed.
- Time travel that is allowing one to access and return previous versions of the data.
- Schema evolution as data evolves, Delta allows Spark table to change in the schema and many more while we use Delta.
- Enabling a Data Lake to update data without going through the entire Data Lake repository.

III. Related Works

Several efforts have been conducted to adapt traditional DWs for handling new user requirements and changes in the underlying data sources. Many approaches focus on DWs that deal with a relational database. However, they cannot be appropriated to deal with big data. In [32], some methodologies try to solve the problem by developing the ETL (Extracting, Transforming, and Loading) process. In [33], the authors attempt to update DW Schema to reflect modifications that already took place. As mentioned in [33], [33], [34], [5], [35], [36], the authors use temporal DW and schema versions to update the DWs Structure by keeping more than one DWs version. These works do not depict how the user's needs impact the evolution changes.

Limited approaches have taken care of handling the aspect of big data development. The approach mentioned in [37] describes data schema specification and evolution processing, but it does not depend on DW. The work presented in [38] explained the method for treating the growth of the data sources in the integration area using big data integration ontology. It can handle some changes in data sources, but it does not determine how to answer all requirements.

Several types of research have concentrated on the use of DWs in big data analysis. The authors in [39] display an OLAP method for big data executed with Hadoop. They focus on multidimensional analysis for big data analysis. However, they do not address big data evolution, which applies to the research work proposed in [40] and [26].

Other researchers studied the problem of big data evolution. In [41], the authors discuss a methodology for constructing a system for big data analysis. However, it cannot be applicable in the state of the previously used data is not provided.

The authors in [42] presented the DW approach for Big Data analysis that was implemented using Map-Reduce. This approach handles two types of changes: (1) schema versions in metadata control variations of the fact table and (2) slowly changing dimensions. The approach does not process changes in big data sources that may affect the analysis process and results.

IV. The Proposed Lake Data Warehouse Architecture

Our contribution aims to adapt traditional DW to solve the new challenges by handling semi-structured and unstructured data. In addition to managing the growth of business requirements and treating the two main drawbacks, namely, availability and system performance. Furthermore, it enhances query performance by providing the required data from users at any time.

In this section, we present a novel DW architecture that improves the traditional DW performance to deal with these challenges. Our contribution integrates the traditional DW architecture with big data technologies like Hadoop and Apache Spark.

In the age of big data, We need to improve DW architecture to handle the new challenges imposed by big data. The Hadoop Framework and Apache Spark are a complement to handling the challenges of traditional DW; each has its advantages in different circumstances. In some cases, we need the Hadoop Framework and Apache Spark to process unstructured or semi-structured data (raw data) and large volume datasets (big data). In other words, we still depend on traditional DW for consistent and high-quality data (structured data), and low latency and interactive reports.

In Fig. 2, we explain the proposed Lake DW Architecture. Where Hadoop and Spark do not replace traditional DW and Big Data is going to change traditional DW architecture but not replacing it. Our contribution depends on integrating traditional DW techniques, Hadoop Framework, and Apache Spark into a hybrid solution.

The large amount of big data generated every minute and every hour needs a data lake that can scale to handle this volume. Therefore, we use Hadoop as a data platform for data lakes to provide extensive scalability at an acceptable cost. Besides, Data Lakes can be complemented DW besides the Hadoop framework. Also, Data Lakes can be complemented DW besides the Hadoop framework. Our proposed architecture differentiates itself from all previous work. It is a hybrid environment that upgrades traditional DW with Hadoop environment depending on the Hadoop-based Data Lake because it can extend the use and capabilities of traditional DW, as follows:

- *Collecting or capturing data from structured data sources using ETL Architecture*. DW depends on the traditional ETL process; where extract (E) data from operational databases and then, the data process, clean and transform (T) before loading (L) them into the DW or data marts or virtual data marts. DW is designed to handle and analyze read-heavy workloads. DW needs to define the data model before loading the data. Then, they call a Schema-On-Write approach, as presented in Fig. 2.
- *Collecting or capturing data from Semi-Structured or Unstructured data sources using ELT Architecture*. Big Data requires a different process to collect data where traditional ETL does not work well on semi-Structure or unstructured data. Big Data calls for ELT. The raw data will be stored in its original format. The preprocessing step will not be used until the query or other application acknowledge/ ask for these data. Where Data Lake is different from DW through the processing of data in the ELT order and utilizing the Schema-on-Read approach, as shown in Fig. 2.

Fig. 2 presents a set of essential components of our contribution model as follows:

A. Hadoop-Based Data Lake architecture in Cloud Environment

It is using Hadoop as a staging area for DW by adding Hadoop-based Data Lake that is a storage repository that is used for complementing traditional DW. It is using as a data source that passes only required data to Data Lake and ingesting unlimited amounts of raw data that related to business objectives. As shown in Fig. 2, we explore the Hadoop-based Data Lake architecture has many layers as follows:

1) *Ingesting data layer:* ingests raw data in native format, where the ingested data can be micro-batch, macro-batch, batch, real-time, or hybrid.

2) *Landing data layer:* Different data types (ingested in the previous layer) are stored in native format (without processing). In this layer, users can find the original data versions of their analysis to aid the subsequent handling.

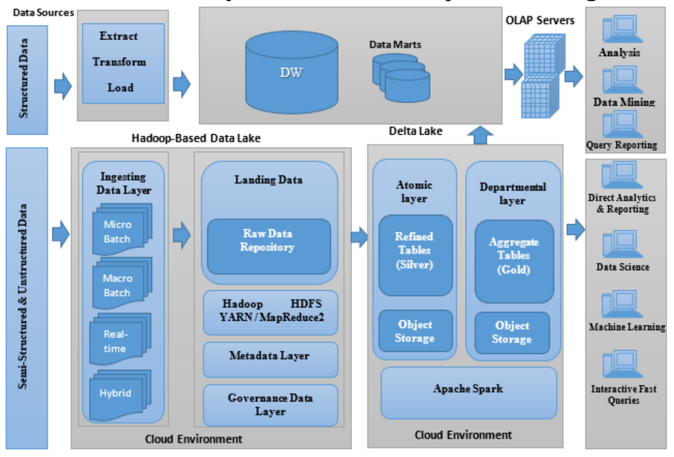


Fig. 2. The proposed Lake Data Warehouse (DW) Architecture

3) *Metadata Layer:* is responsible for making data easy to access and extract values from Data Lake. It helps to make

identifications, data versioning, entity and attributes, distributions, quality.

4) Governance Data Layer: applies to the other layers. It is responsible for authentications, data access, data security, data quality, and data life cycle. It determines the responsibilities of governing the right to access and handling the data.

B. Delta Lake architecture with Apache Spark Cloud Environment

Delta Lake technology is used with Apache Spark to implement our proposed model by creating a cloud data platform for solving Data Lake challenges, such (1) the data quality is low, (2) reading and writing are not guaranteed, (3) insufficient performance with growing volumes of data, and (4) updating records is hard [43]. As presented in Fig. 2, Delta Lake Architecture has two layers:

1) The atomic layer would be a Silver Delta table built using Object Storage, and

2) The Departmental layer would be any number of Gold Delta tables also built on Object Storage. We employ Spark and store all data in Apache Parquet format allowing Delta Lake to leverage the well-organized compression native to Parquet.

Apache Spark is used to read and process huge files and datasets. Spark provides a query engine capable of processing data in huge data files. Some of the most significant Spark jobs in the world run on Petabytes of data. Apache Parquet has the format as a columnar file responsible for optimizations to go faster queries [44]. It is a more efficient file format than JSON files. It is suitable for data processing in the Hadoop. It provides an efficient method to handle complex datasets.

The main difference between our contributions Lake DW Architecture over traditional DW as follows:

- Handling different data types (structured, semi-structured, and unstructured data) from various sources.
- Extracting, storing, managing, and analyzing data volume that cannot handle by traditional DW.
- Integrating between traditional DW technique, Hadoop Framework, and Apache Spark as a hybrid solution. It uses ETL or ELT processes depending on types of data sources.
- Supporting different data types in various sources.
- Determining and analyzing data from Data Lake and Delta Lake, they scale to extreme data volumes.
- Supporting all users, especially Data scientists, because they can do in-depth analysis.

V. Case Study

Our goal of this section is to experiment with our contribution to prove its effectiveness in dealing with big data and analyze it for decision-makers. This case study shows how our proposed model can extract, integrate, and analyze big data that cannot be handled by traditional DW. We use Data Lake to collect and process the Internet of Things (IoT) data. We provide a demo IoT sensor dataset for demonstration purposes. The data simulates heart rate data measured by health tracker devices. Each file consists of five users whose heart rate is measured each hour, 24 hours a day, every day. We store datasets in the data lake as JSON files. We use two data files in JSON format, the first file for readings recorded by the devices in January 2020 (*health_tracker_data_2020_01.json*), and the second file for the readings recorded by the devices for February 2020 (*health_tracker_data_2020_02.json*).

We implemented on Data Lake, Delta Lake, and Apache Spark in Databricks, which provides an integrated platform for working with Apache Spark. When working with Delta Lake, Parquet files can be converted in-place to Delta files. Next, we will convert the Parquet-based data lake table we created previously into a Delta table.

A. Configure Apache Spark

1) Creating the ClustreHelthTracher cluster is computation resources used to run data science and data analytics such as the ETL process, ad-hoc analytics, and streaming analytics.

2) Configuration the Apache Spark, we will need to perform a few configuration operations on the Apache Spark session to get optimal performance. These will include creating a database to store data, as shown in the following Spark SQL script:

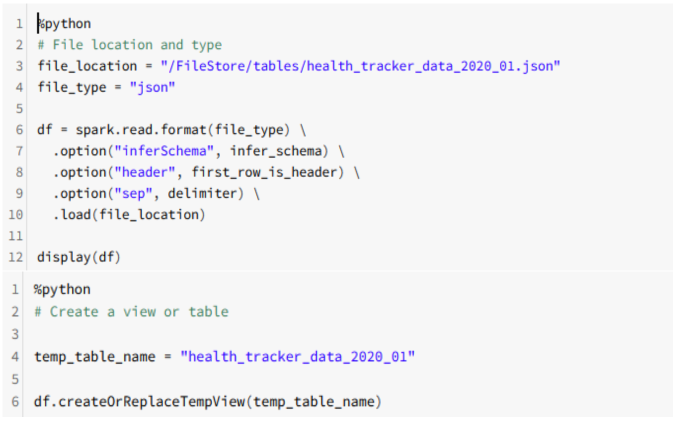
```
CREATE DATABASE IF NOT EXISTS DB_Health_Tracker;
USE DB_Health_Tracker
```

3) In additional to configure the number of shuffle partitions as shown in the following Spark SQL script:

```
SET spark.sql.shuffle.partitions=8
```

B. Importing data from a Data Lake into a Delta Lake by ETL of Apache Spark

We import data from our Data Lake and save it into a Delta Lake as Parquet files. We appended the first month of records and kept in the *health_tracker_data_2020_02.json* file by using the ETL process of Apache Spark. In additional to configure the number of shuffle partitions as presented in the following Python language scripts:



```
%python
# File location and type
file_location = "/FileStore/tables/health_tracker_data_2020_01.json"
file_type = "json"

df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

display(df)
```

```
%python
# Create a view or table

temp_table_name = "health_tracker_data_2020_01"

df.createOrReplaceTempView(temp_table_name)
```

In Fig. 3, we visualize the sensor data over time, as displayed in the following Spark SQL script:

```sql
select * from `health_tracker_data_2020_01`
```

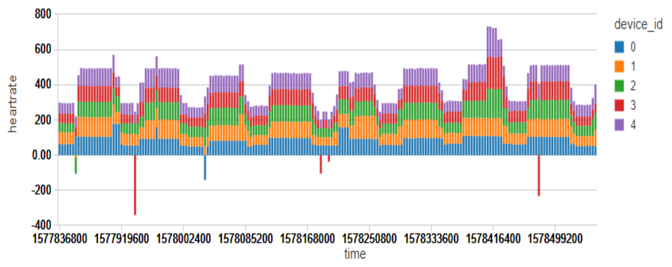


Fig. 3. The sensor data over time.

C. Create a Parquet-based Data Lake Table

We convert the existing Parquet file to The Parquet-based data lake table that will be used to Delta tables. We will be writing files to the root location of the Databricks File System (DBFS) in our cloud object storage. We create the table using the *Create Table As Select* (CTAS) Spark SQL pattern as shown in the following script:

```sql
DROP TABLE IF EXISTS health_tracker_silver;              -- ensures that if we run this again, it won't fail

CREATE TABLE health_tracker_silver
USING PARQUET
PARTITIONED BY (p_device_id)                             -- column used to partition the data
LOCATION "/DB_Health_Tracker/DLRS/healthtracker/silver"       -- location where the parquet files will be saved
AS (
  SELECT name,                                           -- query used to transform the raw data
         heartrate,
         CAST(FROM_UNIXTIME(time) AS TIMESTAMP) AS time,
         CAST(FROM_UNIXTIME(time) AS DATE) AS dte,
         device_id AS p_device_id
  FROM health_tracker_data_2020_01
)
```

Count the records in the ***health_tracker_silver*** table. We expect to have 3720 records: five device measurements, 24 hours a day for 31 days, as shown in the following Spark SQL script:

```sql
SELECT COUNT(*) as NumberOfRecord FROM health_tracker_silver
```

```
▸ (1) Spark Jobs

count(1)
3720
```

D. Convert an Existing Parquet-based Data Lake Table to a Delta table: The Atomic layer of Delta Lake

A Delta table consists of three things: (1) The Delta files containing the data and kept in object storage. (2) A Delta table registered in a central Hive meta-store accessible by all clusters to persist table metadata. (3) The Delta Transaction Log (an ordered record of every transaction) saved with the Delta files in object storage. To Convert an Existing Parquet-based Data Lake Table to a Delta table by using the following steps:

1) Convert the Files to Delta Files: We convert the files in place to Parquet files. The conversion creates a Delta Lake transaction log that tracks the files. Now, the directory is a directory of Delta files, as shown in the following Spark SQL script:

```sql
CONVERT TO DELTA
  parquet.`/DB_Health_Tracker/DLRS/healthtracker/silver`
  PARTITIONED BY (p_device_id double)
```

2) Register the Delta Table: we will register the table in the Metastore. The Spark SQL command will automatically infer the data schema by reading the Delta files' footers, as shown in the following Spark SQL script:

```sql
DROP TABLE IF EXISTS health_tracker_silver;

CREATE TABLE health_tracker_silver
USING DELTA
LOCATION "/DB_Health_Tracker/DLRS/healthtracker/silver"
```

With Delta Lake, the Delta table is ready to use the transaction log stored with the Delta files containing all metadata needed for an immediate query. We count the records in the *health_tracker_silver* table with a Spark SQL query as follows:

```sql
SELECT COUNT(*) FROM health_tracker_silver
```

```
▸ (2) Spark Jobs

count(1)
3720
```

E. Creating an aggregate Delta Table: The Departmental layer of Delta Lake

We create a new Delta table (an aggregate table) from the data in the *health_track_silver* Delta table. We create a *health_tracker_user_analytics* Delta table of summary statistics for each device. We use the Create Table As Select (CTAS) Spark SQL as shown in the following script:

```sql
DROP TABLE IF EXISTS health_tracker_user_analytics;

CREATE TABLE health_tracker_user_analytics
USING DELTA
LOCATION '/DB_Health_Tracker/DLRS/healthtracker/gold/health_tracker_user_analytics'
AS (
  SELECT p_device_id,
         SUM(heartrate) AS sum_heartrate,
         AVG(heartrate) AS avg_heartrate,
         STD(heartrate) AS std_heartrate,
         MIN(heartrate) AS min_heartrate,
         MAX(heartrate) AS max_heartrate

  FROM health_tracker_silver GROUP BY p_device_id
)
```

F. Exploring analysis results

The health_tracker_user_analytics table could be used to define a dashboard for analyzing of the results according to business requirements as provided in Fig. 4 which describes the aggregation results such as maximum, minimum, and average data, as presented in the following Spark SQL script:

```sql
SELECT * FROM health_tracker_user_analytics
```

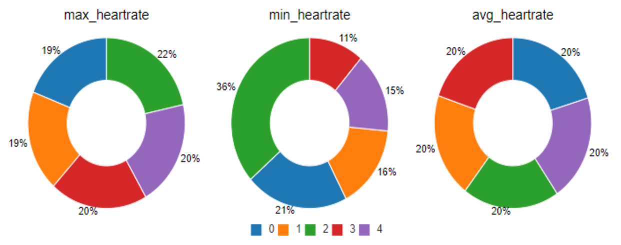


Fig. 4. The analysis of the results according to business requirements.

G. *Appending Files to an Existing Delta Table (Batch data write to Delta Tables)*

We convert the existing Parquet file to The Parquet-based data lake table that will be used to Delta tables. We will be writing files to the root location of the Databricks File System (DBFS) in our cloud object storage. We create the table using the *Create Table As Select* (CTAS) Spark SQL pattern as shown in the following script:

H. *Exploring analysis results*

We can modify existing Delta tables through appending files to an existing directory of Delta files. We append the next month of records, kept in *the health_tracker_data_2020_02* table by using *the INSERT INTO* Spark SQL command as shown in the following script:

```
INSERT INTO health_tracker_silver
SELECT name,
       heartrate,
       CAST(FROM_UNIXTIME(time) AS TIMESTAMP) AS time,
       CAST(FROM_UNIXTIME(time) AS DATE) AS dte,
       device_id as p_device_id
FROM health_tracker_data_2020_02
```

I. *Assessing the Missing Records*

After a batch update of the *health_tracker_silver* table, we counted the number of records in the table. We discovered that some records were missing by *Count the Number of Records* per Device.

```
SELECT p_device_id, COUNT(*) FROM health_tracker_silver GROUP BY p_device_id
```

J. *Assessing the Missing Records*

After a batch update of the health_tracker_silver table, we counted the number of records in the table. We discovered that some records were missing by Count the Number of Records per Device.

TABLE I. THE NUMBER OF RECORDS PER DEVICE

Device ID	Number of Records
0	1440
1	1440
2	1440
3	1440
4	1345

In Table I, device number 4 looks are missing 95 records. We run a query to discover the missing records' timing by displaying the number of records per day. We have no records for device 4 for the last few days of the month, as shown in the following Spark SQL query:

```
SELECT dte as Date, p_device_id as Device_Id,
heartrate as Heart_Rate_Reading
FROM health_tracker_silver
WHERE p_device_id IN (1, 4) and dte > "20-2-2020"
```

In Fig. 5, the absence of records from the last few days of the month shows a phenomenon that may often occur in a production data pipeline: late-arriving data. Delta Lake allows us to process data as it arrives and is prepared to handle the occurrence of late-arriving data.

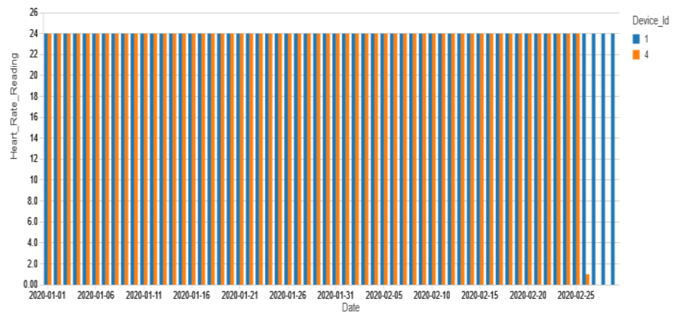


Fig. 5. The absence of records.

K. *Identify Broken Readings in the Table*

In the initial load of data into the *health_tracker_silver* table, we noted that there are broken records in the data. In specific, we made a note of the fact that several negative readings were present even though it is impossible to record a negative heart rate. Let us assess the extent of these broken readings in our table. First, we create a temporary view for the broken readings in the health_tracker_silver table, as shown in the following Spark SQL script:

```
CREATE OR REPLACE TEMPORARY VIEW broken_readings
AS (
  SELECT COUNT(*) as broken_readings_count, dte FROM health_tracker_silver
  WHERE heartrate < 0
  GROUP BY dte
  ORDER BY dte
)
```

Next, we sum the records in the view, as shown in the following Spark SQL query:

```
SELECT SUM(broken_readings_count)
FROM broken_readings

Result: 67
```

VI. CONCLUSION

Many companies use DWs in different areas to help in the decision-making process. Besides, DWs enhance business intelligence, data quality, and consistency, saving time, and storing historical data. In the age of Big Data, the amount of data needed for ingesting and storing is an unprecedented rate. However, the architecture of the traditional DWs cannot manage such large amounts of data. Its new types from the current data sources are autonomous, heterogeneous, scalable, and distributed, which requires modern technology and modern DW architecture to deal with it.

In this paper, we proposed a novel DW architecture called Lake Data Warehouse Architecture. Lake Data Warehouse Architecture is a new DW structure that depends on integrating between traditional DW technique, Hadoop Framework, and Apache Spark as a hybrid solution. It familiarizes a traditional DW. It employs big data technologies and tools (Hadoop, Apache Spark, Data Lake, and Delta Lake) in a complementary way to support and enhance existing architecture. Furthermore, it can improve scalability and reduce the costs of development of traditional DW architecture.

Lake Data Warehouse Architecture has many competencies over traditional DWs Architecture, such as solving several issues that face integrating data from big data repositories, handling different data types from various sources. Like Structured data (DBs, spreadsheet), Semi-structured data (XML files, JSON files), and Unstructured data (video, audio, images, emails, word, PowerPoint, pdf files). Moreover, it captures, stores, manages, and analyzes data volume that cannot be handled by traditional DW. Furthermore, it uses the recent technology such as the Hadoop Platform, Data Lake, Delta Lake, and Apache Spark is inexpensive to minimize the time spent analyzing data and to reduce storage costs.

REFERENCES

[1] F. Silvers, Building and maintaining a data warehouse. CRC Press, 2008.

[2] T. John, Data Lake for Enterprises. Packt Publishing Ltd, 2017.

[3] M. Z. Zgurovsky and Y. P. Zaychenko, Big Data: Conceptual Analysis and Applications. Springer Nature Switzerland AG, 2020.

[4] W. H. Inmon, Building the Data Warehouse, Fourth Edition, vol. 13, no. 401. 2005.

[5] E. Saddad, A. El-Bastawissy, O. Hegazy, and M. Hazman, “Towards an alternative Data Warehouses Architecture,” in 14th International Conference on Hybrid Intelligent Systems (HIS 2014), Kuwait, December 14-16, 2014 (IEEE, Poster), 2014, vol. 6, pp. 48–53.

[6] K. Krishnan, Data Warehousing in the Age of Big Data. Elsevier Inc., 2013.

[7] R. G. Goss and K. Veeramuthu, “Heading towards big data building a better data warehouse for more data, more speed, and more users,” in ASMC 2013 SEMI Advanced Semiconductor Manufacturing Conference, 2013, pp. 220–225.

[8] A. Sebaa, F. Chikh, A. Nouicer, and A. Tari, “Research in Big Data Warehousing using Hadoop,” J. Inf. Syst. Eng. Manag., vol. 2, no. 2, pp. 1–5, 2017.

[9] W. H. Inmon and D. Linstedt, Data Architecture: A Primer for the Data Scientist: Big Data, Data Warehouse and Data Vault. 2014.

[10] C. L. Philip Chen and C. Y. Zhang, Data-intensive applications, challenges, techniques and technologies: A survey on Big Data, vol. 275. Elsevier Inc., 2014.

[11] V. K. A. Arockia Panimalar. S, Varnekha Shree. S, “The 17 V’s of Big Data,” Int. Res. J. Eng. Technol., vol. 4, no. 9, pp. 3–6, 2017.

[12] N. Khan, M. Alsaqer, H. Shah, G. Badsha, A. A. Abbasi, and S. Salehian, “The 10 Vs, issues and challenges of big data,” ACM Int. Conf. Proceeding Ser., no. March, pp. 52–56, 2018.

[13] R. Rialti and G. Marzi, Ambidextrous Organizations in the Big Data Era. Cham: Springer International Publishing, 2019.

[14] A. Khaleel and H. Al-Raweshidy, “Optimization of Computing and Networking Resources of a Hadoop Cluster Based on Software Defined Network,” IEEE Access, vol. 6, no. c, pp. 61351–61365, 2018.

[15] I. B. Lars George, Jan Kunigk, Paul Wilkinson, Architecting Modern Data Platforms: A Guide to Enterprise Hadoop at Scale. O’Reilly Media, 2019.

[16] Z. Yang and X. Guo, “Teaching Hadoop Using Role Play Games,” Decis. Sci. J. Innov. Educ., vol. 18, no. 1, pp. 6–21, 2020.

[17] J. Liu, S. Tang, G. Xu, C. Ma, and M. Lin, “A Novel Configuration Tuning Method Based on Feature Selection for Hadoop MapReduce,” IEEE Access, vol. 8, pp. 63862–63871, 2020.

[18] C. Walker and H. Alrehamy, “Personal Data Lake with Data Gravity Pull,” Proc. - 2015 IEEE 5th Int. Conf. Big Data Cloud Comput. BDCloud 2015, pp. 160–167, 2015.

[19] P. P. Khine and Z. S. Wang, “Data lake: a new ideology in big data era,” ITM Web Conf., vol. 17, no. December, p. 03025, 2018.

[20] A. Panwar and V. Bhatnagar, “Data Lake Architecture: A New Repository for Data Engineer,” Int. J. Organ. Collect. Intell., vol. 10, no. 1, pp. 63–75, 2020.

[21] P. Russom, “Data lakes: purposes, practices, patterns and platforms,” pp. 1–42, 2017.

[22] H. D. Challenges, S. Gupta, and V. Giri, Practical Enterprise Data Lake Insights. Apress, 2018.

[23] A. Gorelik, The enterprise big data lake: Delivering the promise of big data and data science. O’Reilly Media, 2019.

[24] C. Madera and A. Laurent, “The next information architecture evolution: The data lake wave,” 8th Int. Conf. Manag. Digit. Ecosyst. MEDES 2016, pp. 174–180, 2016.

[25] Lei Zhang, “What are the differences between a database, data mart, data warehouse, a data lake and a cube?,” quora, 2016. [Online]. Available: https://www.quora.com/. [Accessed: 15-Feb-2019].

[26] M. Y. Santos, B. Martinho, and C. Costa, “Modelling and implementing big data warehouses for decision support,” J. Manag. Anal., vol. 4, no. 2, pp. 111–129, 2017.

[27] F. Ravat and Y. Zhao, “Data Lakes: Trends and Perspectives,” in Springer Nature Switzerland AG 2019, 2019, vol. 2, no. Umr 5505, pp. 304–313.

[28] A. T. Spark, “Apache Spark,” no. 1, pp. 39–53, 2018.

[29] M. M. Rathore, H. Son, A. Ahmad, A. Paul, and G. Jeon, “Real-Time Big Data Stream Processing Using GPU with Spark Over Hadoop Ecosystem,” Int. J. Parallel Program., vol. 46, no. 3, pp. 630–646, 2018.

[30] T. Mahapatra and C. Prehofer, Graphical Flow-based Spark Programming, vol. 7, no. 1. Springer International Publishing, 2020.

[31] Microsoft Team, “Introduction to Delta Lake - Azure Databricks | Microsoft Docs,” 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/databricks/delta/delta-intro. [Accessed: 26-May-2020].

[32] A. Wojciechowski, “ETL workflow reparation by means of case-based reasoning,” Inf. Syst. Front., vol. 20, no. 1, pp. 21–43, 2018.

[33] F. Bentayeb, C. Favre, and O. Boussaid, “A user-driven data warehouse evolution approach for concurrent personalized analysis needs,” Integr. Comput. Aided. Eng., vol. 15, no. 1, pp. 21–36, 2008.

[34] G. Thakur and A. Gosain, “DWEVOLVE: a requirement based framework for data warehouse evolution,” ACM SIGSOFT Softw. Eng. Notes, vol. 36, no. 6, pp. 1–8, 2011.

[35] W. Ahmed, E. Zimányi, and R. Wrembel, “A logical model for multiversion data warehouses,” in International Conference on Data Warehousing and Knowledge Discovery, 2014, pp. 23–34.

[36] M. Thenmozhi and K. Vivekanandan, “An ontological approach to handle multidimensional schema evolution for data warehouse,” Int. J. Database Manag. Syst., vol. 6, no. 3, p. 33, 2014.

[37] C. Olston et al., “Nova: continuous pig/hadoop workflows,” in Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, 2011, pp. 1081–1090.

[38] S. Nadal, O. Romero, A. Abelló, P. Vassiliadis, and S. Vansummeren, “An integration-oriented ontology to govern evolution in big data ecosystems,” Inf. Syst., vol. 79, pp. 3–19, 2017.

[39] J. Song, C. Guo, Z. Wang, Y. Zhang, G. Yu, and J.-M. Pierson, “HaoLap: A Hadoop based OLAP system for big data,” J. Syst. Softw., vol. 102, pp. 167–181, 2015.

[40] W. Chen, H. Wang, X. Zhang, and Q. Lin, “An optimized distributed OLAP system for big data,” in 2017 2nd IEEE International Conference on Computational Intelligence and Applications (ICCIA), 2017, pp. 36–40.

[41] R. Tardio, A. Mate, and J. Trujillo, “An iterative methodology for big data management, analysis and visualization,” in 2015 IEEE International Conference on Big Data (Big Data), 2015, pp. 545–550.

[42] S. Chen, “Cheetah: a high performance, custom data warehouse on top of MapReduce,” Proc. VLDB Endow., vol. 3, no. 1–2, pp. 1459–1468, 2010.

[43] K. Akepanidtaworn, “Is ‘Delta Lake’ Replacing ‘Data Lakes’? - Dev Dream Team - Medium,” 2020. [Online]. Available: https://medium.com/dev-dream-team/is-delta-lake-replacing-data-lakes-49f1caddc646. [Accessed: 26-May-2020].

[44] Databricks Team, “Parquet files — Databricks Documentation,” 04/03, 2020. [Online]. Available: https://docs.databricks.com/data/data-sources/read-parquet.html#parquet-files. [Accessed: 25-May-2020]