

Interoperability over meteorological and spatial big data warehouse

Pavlos Baritakis

Thesis submitted in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Prof. *Dimitris Plexousakis*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS).

UNIVERSITY OF CRETE
COMPUTER SCIENCE DEPARTMENT

Interoperability over meteorological and spatial big data warehouse

Thesis submitted by
Pavlos Baritakis
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author:



Pavlos Baritakis

Committee approvals:

**DIMITRIOS
PLEXOUSAKIS**

Digitally signed by DIMITRIOS
PLEXOUSAKIS
Date: 2021.06.15 18:57:02 +03'00'

Dimitris Plexousakis
Professor, Thesis Supervisor

IOANNIS TZITZIKAS

Digitally signed by IOANNIS
TZITZIKAS
Date: 2021.06.14 19:29:44 +03'00'

Yiannis Tzitzikas
Associate Professor., Committee Member

**KONSTANTINOS
MAGOUTIS**

Digitally signed by
KONSTANTINOS MAGOUTIS
Date: 2021.06.14 23:36:22 +03'00'

Kostas Magoutis
Associate Professor., Committee Member

Departmental approval:

**Polyvios
Pratikakis**

Digitally signed by Polyvios
Pratikakis
Date: 2021.06.16 12:40:45
+03'00'

Polyvios Pratikakis
Assistant Professor, Director of Graduate Studies

Heraklion, March 2021

Abstract

Nowadays with climate change being a major issue in Scientific Community and to citizens too, the scientific community struggles to find causes, results and make predictions about future impacts to our lives.

Having Internet Of Things, like Meteorological Sensors, like an arrow in the quiver, helps the Scientific Community collect important meteorological and spatial data in a manner of using this kind of data to build statistical and mathematical models to guide them to more accurate results, plus faster than the existing models.

The frequent collection of the meteorological and spatial data from heterogeneous sources of information, drives to huge portions of data that have to be stored and managed efficiently in a sense of being useful to users by converting raw data format into knowledge.

The solution to the efficient storage and management of these big portions of data was given firstly by building a data warehouse with a collection of different databases, regarding the sources of information and secondly by using a Knowledge Base Layer over the data warehouse. With this approach, we create interoperability over the data warehouse.

The approach of the thesis is a Web-Based Management Information System that uses NoSQL databases to build the Storage Layer so as the Knowledge Base Representation Layer. The Apache Cassandra DB is used as the Storage Layer and the Knowledge Base Layer implemented with the usage of Neo4j Graph DB. The combination of these two NoSQL Databases leads to a dynamic M.I.S. Web-Based Application that can handle the load of data from sensors. The Web App can be used easily from novice to more advanced users to gather and manage the data, create statistics, views and execute dynamic queries to the database warehouse to have results on demand.

Περίληψη

Στις μέρες μας με την κλιματική αλλαγή να αποτελεί μείζων ζήτημα για την Επιστημονική Κοινότητα αλλά και για τους απλούς πολίτες επίσης, η Επιστημονική Κοινότητα αγωνίζεται για να βρει αιτίες, αποτελέσματα και προσπαθεί να κάνει προβλέψεις σχετικά με τις μελλοντικές συνέπειες που θα επιφέρει στις ζωές μας.

Έχοντας το Δίκτυο των Πραγμάτων (IoT), όπως τους αισθητήρες Μετεωρολογικών δεδομένων, σαν βέλος στη φαρέτρα, βοηθά την Επιστημονική Κοινότητα να συλλέγει σημαντικές πληροφορίες σχετικά με τα Μετεωρολογικά και Χωρικά δεδομένα ώστε με τη χρήση των δεδομένων αυτών να υλοποιούν στατιστικά και μαθηματικά μοντέλα, τα οποία και θα τους οδηγήσουν σε ακριβέστερα αποτελέσματα, με ταχύτερο ρυθμό από τα ήδη υπάρχοντα.

Η συχνότητα συλλογής των Μετεωρολογικών και Χωρικών δεδομένων από διαφορετικές πηγές πληροφόρησης, οδηγεί σε μεγάλα τμήματα δεδομένων, τα οποία πρέπει να αποθηκευτούν και να είναι διαχειρίσιμα αποδοτικά, με την έννοια του να μπορούν να είναι χρήσιμα προς τους χρήστες μετατρέποντας ακατέργαστα δεδομένα σε γνώση.

Η λύση για την αποδοτική αποθήκευση και διαχείριση αυτών των μεγάλου όγκου δεδομένων δόθηκε αρχικά με την κατασκευή μιας «Αποθήκης Δεδομένων» με συλλογή από διαφορετικές βάσεις δεδομένων, σχετικές με τις πηγές πληροφόρησης από όπου προέρχονται και δευτερευόντως με τη χρήση επιπέδου «Γνωσιακής Αναπαράστασης», πάνω από το επίπεδο της «Αποθήκης Δεδομένων».

Σκοπός της εργασίας αυτής είναι η υλοποίηση μιας Διαδικτυακής Εφαρμογής Διαχειριστικού Πληροφοριακού Συστήματος, το οποίο χρησιμοποιεί Μη-Σχεσιακές NoSQL βάσεις δεδομένων για την κατασκευή του Επιπέδου Αποθήκευσης, όπως επίσης και του επιπέδου «Γνωσιακής Αναπαράστασης». Η βάση δεδομένων «Apache Cassandra DB» χρησιμοποιήθηκε για το επίπεδο αποθήκευσης και το επίπεδο Γνωσιακής Αναπαράστασης υλοποιήθηκε με τη χρήση γράφων δεδομένων και πιο συγκεκριμένα με «Neo4j Graph DB». Ο συνδυασμός

αυτών των δύο Μη-Σχεσιακών βάσεων δεδομένων οδήγησε σε ένα δυναμικό Διαχειριστικό Πληροφοριακό Σύστημα το οποίο είναι ικανό να χειρίζεται τον όγκο δεδομένων από τους αισθητήρες. Η διαδικτυακή εφαρμογή μπορεί να χρησιμοποιηθεί από αρχάριους ως και έμπειρους χρήστες για να συλλέγουν και να διαχειρίζονται τα δεδομένα, να δημιουργούν στατιστικά και πίνακες δεδομένων, όπως και να εκτελούν δυναμικές επερωτήσεις στην «Αποθήκη Δεδομένων» κατά απαίτηση.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον καθηγητή και επόπτη μου κ. Δημήτριο Πλεξουσάκη, Διευθυντή Έρευνας και πρεσβευτή του Ινστιτούτου Πληροφορικής, ΙΤΕ, για την εμπιστοσύνη που μου έχει δείξει αλλά και την στήριξή του όλα αυτά τα χρόνια σαν καθηγητής, αλλά και σαν άνθρωπος πάνω απο όλα. Οι γνώσεις και το όραμά του όπως και η συμβουλευτική καθοδήγηση του μου έδωσαν τα απαραίτητα εφόδια όλο αυτό τον καιρό να επιζητώ πάντα το καλύτερο τόσο στις σπουδές μου όσο και στη μετέπειτα εργασιακή μου σταδιοδρομία.

Στο σημείο αυτό θα ήθελα επίσης να ευχαριστήσω τους κ. Απόστολο Τραγανίτη και κ. Ιωάννη Τζιτζικα για την συμβουλευτική καθοδήγηση στις μεταπτυχιακές σπουδές μου όπως και για τις καθοριστικές γνώσεις που απέκτησα μέσα από μαθήματά τους.

Ακόμα ευχαριστώ τα μέλη του εργαστηρίου Πληροφοριακών Συστημάτων για την άριστη συνεργασία που έχουμε όλο αυτόν τον καιρό, τις γνώσεις που μου έχουν μεταφέρει αλλά και τις ευχάριστες στιγμές που έχω περάσει στο εργαστήριο αυτό.

Από τα μέλη του εργαστηρίου δε θα μπορούσα να μην ευχαριστήσω ιδιαίτε-
τως την Αργυρώ(Πούλα της καρδιάς μας) και Νίνα για τη χρόνια φιλία που μας συνδέει και για την αμέριστη υποστήριξή τους.

Επίσης από καρδιάς μου θα ήθελα να ευχαριστήσω τους αδερφικούς μου φίλους Γιάννη, Ηλία, Νίκο και τον κουμπάρο μου Νίκο για τη δύναμη που μου δίνουν στο κάθε τι ακόμα και από μακριά από τις χώρες που πλέον βρίσκονται.

Τέλος το μεγαλύτερο ευχαριστώ ανήκει στην οικογένεια μου Δημήτρη, Βάσω, Κέλλυ, Γιάννη και την Ειρήνη έχοντάς τους σαν βάση για το κάθε τι στη ζωή μου, για την κατανόηση, την υπομονή, τη βαθύτατη αγάπη που μου δείχνουν αλλά και τη στήριξη που δείχνουν σε κάθε μου εγχείρημα είτε αυτό αφορά τις σπουδές μου είτε τις επαγγελματικές επιλογές μου. Το ήθος και οι αρχές των γονιών μου Δημήτρη και Βάσω, όπως και η διαπαιδαγώγησή που έλαβα με βοήθησαν στη μετέπειτα πορεία της ζωής μου.

Σας ευχαριστώ όλους έναν προς έναν!

στους γονείς μου,
Δημήτρη και Βάσω

Contents

1	Introduction	1
2	Related Work	4
3	Meteorological and Spatial Data format	6
4	System Architecture	10
4.1	Data Warehouse with NoSQL Apache Cassandra DB	10
4.1.1	Apache Cassandra	11
4.2	Implementing a Data Integration Layer with NoSQL Neo4j Graph DB	14
4.2.1	Neo4j	15
5	Data Managment	20
5.1	CQLSH Shell and COPY FROM Command	20
5.2	Java Process Handling for the import	23
5.3	Scalability	25
5.4	Performance Issues	25
6	Ensuring Interoperability	27
6.0.1	Incremental Addition of Data Source	35
6.0.2	Lesson Learned	35

7	M.I.S. Web App Overview	37
7.1	M.I.S. Web App Capabilities	37
7.1.1	Data Table (Access: admin, user)	38
7.1.2	Query OWDB (Access: admin, user)	40
7.1.3	Line Chart (Access: admin, user)	42
7.1.4	3D Plot (Access: admin, user)	43
7.1.5	Import File (Access: admin)	44
7.1.6	Execute CQL (Access: admin)	45
7.1.7	Settings (Access: admin)	46
7.1.7.1	Mediator Table	46
7.1.7.2	Export of Column Family (Table) in JSON File	47
7.1.7.3	Users Table	48
8	Conclusion and Future Work	49
	Bibliography	51

Chapter 1

Introduction

Nowadays with climate change being a major issue in Scientific Community and to citizens too, the scientific community struggles to find causes, results and make predictions about future impacts to our lives.

With the invention of Internet of Things Sensors, the collection of data that Scientist can use for profit has been much simpler job than before. In our case the collection of Meteorological and Spatial measurements are the data that has high importance for monitoring and analysis that can lead to valuable results about the climate change. The historic observation of these data is what adds value and not the data by themselves. So collecting and observing Meteorological and Spatial Time Series data is the case of study but yet an easy job.

The frequency of the collection of raw Meteorological and Spatial data comes with the high cost of storing, retrieving and managing-handling the big amount of data that increases over time. A Meteorological Sensor keeps sending data in a daily base to Lab Clusters every certain period of time, that can be every minute, every five or ten minutes or hourly, creating an amount of time series data that is an issue that matters and needs a solution, in a manner of turning the data into useful pieces of information.

Gathering big data from multiple different sensors that are located to different areas around the Crete Island makes the problem of storage and

performance much more complicated. Combining all the sources of information into one Management Information System Web Based App drives us to the need of the Interoperability over these Meteorological and Spatial data.

Related work can be found in Kanishk Chaturv 'InterSensor Service: Establishing Interoperability over Heterogeneous Sensor Observations and Platforms for Smart Cities' [10], Ramar, Kaladevi 'Ontological based interoperability and integration framework for heterogeneous weather systems' [11], M. G. Kibria, S. Ali, M. A. Jarwar and I. Chong, 'A framework to support data interoperability in web objects based IoT environments' [12] and B. Ahlgren, M. Hidell and E. C. -. Ngai, 'Internet of Things for Smart Cities: Interoperability and Open Data' [13] papers. The previous approaches have developed solutions through ontology algorithms, services and frameworks that apply interoperability on data from heterogenous sources of information. The above solutions are from the semantic point of view approach.

The Contribution of our work differs from the previous solutions due to the holistic approach to the interoperability on heterogenous sources of information that offers solution on both knowledge base and the storage of time series big data. As a result we developed a Management Informational System (M.I.S.) Web Application that can be used from novice to expert users due to the simplicity of the system, for data retrieval over the heterogenous data sources of information. The M.I.S. Web App offers usages for data representations, dynamic queries for data retrieval, analytics and statistics of data and also the management of data.

In this thesis we used the NoSQL Apache Cassandra DB [1, 2] as the database warehouse (Storage Layer), in which all the data will be stored. The Apache Cassandra DB is used due to fast writes, handling of massive datasets, high fault tolerance, easy administration and the closely query approach with Cassandra Query Language (CQL) to SQL Developers, plus the fact that distributed systems (Clusters) are used in the Labs that the data is going to be stored.

Having data from different sources makes it difficult to query over different databases and get the right result. The solution in this thesis is the Knowledge Base Representation layer over the data warehouse, which provides us the Interoperability over the Meteorological and Spatial Data. The technology we used to build the Interoperability is the NoSQL Neo4j Graph DB [3, 4]. Neo4j Graph DB is a database that connects everything into graph, with nodes and relationships making very easy to understand every graph model. The high performance in storage and processing, the scalability, the reliability and the ease of use of Cypher query language were the key features to be the intermediate layer that connects the Client with the data warehouse, avoiding an overhead in query performance.

The combination of Apache Cassandra DB for Storage Layer as well as Neo4j Graph DB as the Knowledge Base Representation Layer in this thesis, lead to a high-performance ecosystem to store and query big data over different sources of information, with the Web Based Management Information System making easy to manage this kind of data from novice to advanced users.

Interoperability is achieved through the mapping of nodes, of Neo4j, with certain attributes, to certain Column Families (Tables) and their Columns of Apache Cassandra. This abstract approach works smoothly for time series data but is not limited on this kind of data. This approach can work in any kind of data with the right database design in Storage Layer and with the accurate mapping with the Knowledge Base Layer. The mapping approach in this thesis came from the Global Schema (GAV) [9] approach, with the difference that we used the Knowledge Base Layer as Mediator instead of using a Mediator Table. This approach guided to a much more efficient way to apply mapping through the sources of information.

Chapter 2

Related Work

Kanishk Chaturvedi, et al. in 2018 [10], has developed an open source Java Based implementation of Intersensor Service. His Application supports RDBSM databases, so as NoSQL database, such as MongoDB. Data produced by sensors apart from stored in databases, are stored also in CSV and Excel sheets format. The approach in this paper is the interoperability as a service in heterogenous sensor observations in smart cities.

Ramar Kaladevi et al. in 2016 [11], developed a framework for interoperability over weather sensor data by a novel ontology merging algorithm based on semantic relations. It can be used for knowledge sharing and information retrieval.

M. G. Kibria et al. in 2017 [12], developed a framework for interoperability IoT sensor data of smart cities. This framework processes data from semantic and non-semantic data sources, from heterogeneous sources of information and supports different type formats (CSV, Databases, XML, etc.). The outcome is the export them in RDF/Triple store format so that can create knowledge and analytics to end users.

B. Ahlgren, et al. in 2016 [13], has developed a Project called GreenIoT. This is an application that offers interoperability from sensors big data over heterogeneous sources of information. It's purpose is to offer to citizens and public authorities of Uppsala in Sweden and also worldwide, open accessible data from these sources for daily societal challenges, such as air quality, meteorological metrics and more.

The above approaches have developed frameworks and services that can be used on existing heterogeneous sources of information and they can support the Knowledge Base Layer. These solutions are used in real world in smart cities. What differs our approach from the previous is that we have developed from the scratch a Web Based application that can totally support the Weather Time Series data interoperability from heterogeneous sources of information. Our approach is an architecture with its own Knowledge Based Layer and Storage Layer that are scalable and easily managed to offer great performance as well as stable usage. Using latest NoSQL technologies combined, Apache Cassandra DB for storage and Neo4j Graph DB for the knowledge layer we solve the problem of storing and retrieving big data in distributed systems in an effective way.

Chapter 3

Meteorological and Spatial Data format

The data files that we used are in .CSV and .DAT format, that include measurements from meteorological sensors. These sensors keep tracking data, at the end of each day a file is created with all day long data included. Each file is generated from a different source (sensors) of information. Each source gives data daily, weekly, monthly, yearly and even a file can be created for a certain period of time.

The Weather Stations of Department of Mathematics and Applied Mathematics and Hellenic Republic Decentralized Administration of Crete were the sources of information, from which we took samples of data.

The sample file from Department of Mathematics and Applied Mathematics was a tab separated .DAT file. The file consists of forty three (43) columns and one thousand two hundred and thirteen (1.213) rows. The data is from a certain period of time and it keeps data tracking per ten minutes for meteorological and spatial measurements with the minimum, maximum and average values of each attribute.

The Meteorological attributes are (only attribute names described below, min, max, average column names excluded):

1. Date (dd/mm/yyyy)
2. Time (hh:mm)
3. Wind Speed at 20m (m/s)
4. Wind Speed at 28.5m (m/s)
5. Wind Speed at 30m (m/s)
6. Average Wind Direction at 20m (degrees)
7. Average Wind Direction at 28.5m (degrees)
8. Temperature (Celsius degree)
9. Rain Duration
10. Atmospheric pressure at the altitude of the station
11. Relative Humidity

The sample files from Hellenic Republic Decentralized Administration of Crete were two files from two different sources of information. Both files were in tab separated .CSV files. The first source of information was from the source of Tympaki Town and the second from the source of Doxaro Village Weather Stations. The generated file from Tympaki sensors consist of eleven (11) columns and nine thousand the two hundred fifty six (9.256) rows. The generated file from Doxaro sensors consist of eleven (11) columns and five thousand and twenty one (5.021) rows. Both files have similar Meteorological attributes and their average values per hour.

The Meteorological attributes are:

1. Date (yyyy-mm-dd)
2. Time (hh:mm:ss)
3. Barometric Pressure

4. Relative Humidity
5. WIND SPEED (m/s)
6. WIND DIRECTION (degrees)
7. Temperature (Celsius degree)
8. Precipitation (mm)
9. Pyranometer 0 2000
10. Hourly Eto
11. Rain Duration

The meteorological files are in NetCDF [14] climate common data format. NetCDF format is 'self-describing', this means that there is a header which describes the layout of the rest of the file, as name/value attributes. NetCDF is commonly used in climatology, meteorology and oceanography applications.

Data in NetCDF format is:

- Self-Describing: A NetCDF file includes information about the data it contains.
- Portable: A NetCDF file can be accessed by computers with different ways of storing integers, characters, and floating-point numbers.
- Scalable: Small subsets of large datasets in various formats may be accessed efficiently through NetCDF interfaces, even from remote servers.
- Appendable: Data may be appended to a properly structured NetCDF file without copying the dataset or redefining its structure.
- Sharable: One writer and multiple readers may simultaneously access the same NetCDF file.
- Archivable: Access to all earlier forms of NetCDF data will be supported by current and future versions of the software.

Geospatial Data has use two main types of geospatial data formats. The Vector Data and Raster Data [15]. In our case of study, the files contain neither Vector nor Raster Data explicitly. The Geospatial information is produced from the data sources(meteorological sensors) that are in different Geographical points around Crete Island.

Chapter 4

System Architecture

In this section we are going to tear down step-by-step the Layers that we described earlier and make a clear view of the usage of each technology.

As a first step, we are going to examine the Storage Layer with NoSQL Apache Cassandra DB. In second step we will describe the technic that we used to build the Knowledge Base Representation Layer with NoSQL Neo4j Graph DB.

4.1 Data Warehouse with NoSQL Apache Cassandra DB

From the analysis of data files format of previous section, we observed that are wide-column tables, that consist of time series data with the attributes(columns) Date and Time being the indexes of each record with measurements written from the sensors.

Requirements in storage and performance in this kind of data drove us to the NoSQL solution Apache Cassandra DB [5].

4.1.1 Apache Cassandra

A short briefing on Apache Cassandra DB is that we have a NoSQL Database that provides no single point of failure by not having a Master-Slave Architecture. Having this as an aspect, we can have a scalable Cluster by adding any type of machine-server at any moment, that each one of them can work properly and equally in a Apache Cassandra DB Cluster. A feature that makes Apache Cassandra DB stand out from the rest NoSQL Databases is called Tunable Consistency Model, by providing to a developer the choice of having performance over accuracy or vice versa, compared to other distributed systems databases according to CAP-Theorem (also known as Brewer's theorem) [6]. Apache Cassandra DB is distributed over several machines that cooperates with each other, these machines are called nodes. It arranges each node in a ring format and assigns data to each one. Every node contains the replication of data and in case of failure the replication changes as it needs to.

Apache Cassandra DB stores data in Keyspaces is the outer container of data and the basic attributes of Keyspaces are:

1. **Replication Factor:** number of machines in the cluster that will receive copies of the same data.
2. **Replica placement strategy:** strategy to place replicas in the ring.
3. **Column Family (Table):** is a container for an ordered collection of rows in the Keyspace container. Each row is a container for an ordered collection of columns. Keyspace can contain at least one Column Family or more.

A picture of Apache Cassandra DB Model:

After this briefing as a next step, we are going to explain how we modeled database by using the Apache Cassandra technical terms.

From the data file analysis, we created a Keyspace that contains all the column families created for each different source of information.

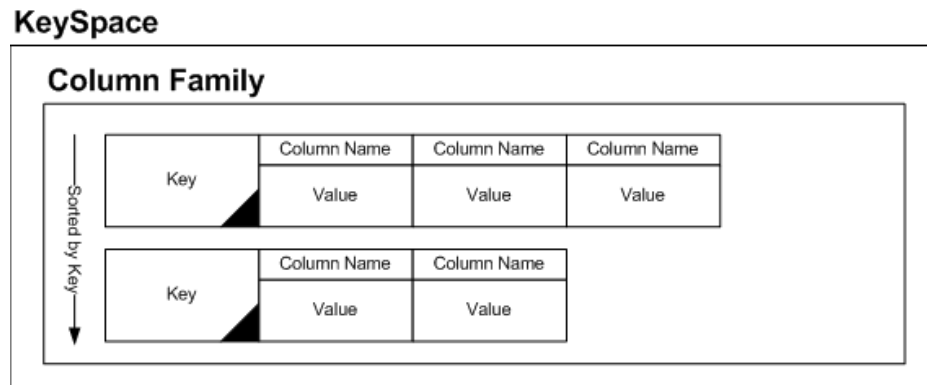


Figure 4.1: Cassandra Model

Design Decision:

1. Keyspace is named as meteo
2. Column Family generated from Doxaro Village file is named as doxaro.
3. Column Family generated from Tympaki Town file is named as tympaki a3 (a3 is from the meteorological sensor that data is collected).
4. Column Family generated from Department of Mathematics and Applied Mathematics file is named as mathuoc.

All the Column families are created from time series data as we described in Meteorological and Spatial Data format section, Date and Time attributes combined creates the Composite Key of Cassandra that makes each record unique in a column family. Composite Key [1] is a special type of Primary Key to represent groups of related rows, called partitions.

A picture of what a Cassandra Table with Composite Key Structure [1]:

CQL Command to create the Keyspace:

```
CREATE KEYSPACE meteo
WITH replication =
```

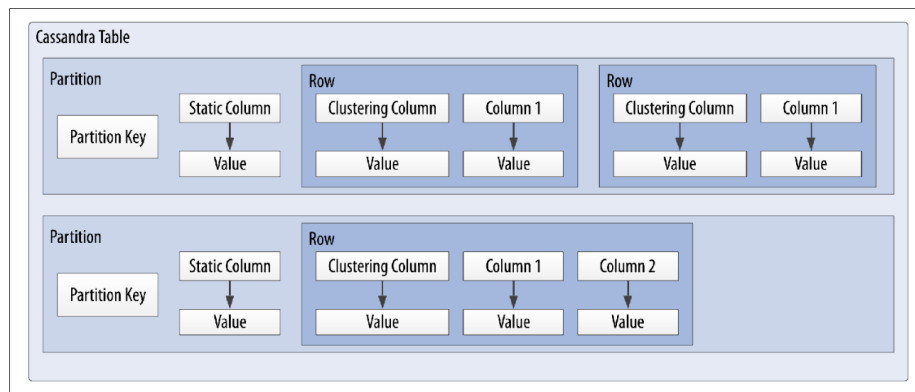


Figure 4.2: Cassandra Partitions

```
'class': 'SimpleStrategy',
'replication_factor': 1
;
```

The replication factor attribute was set to 1, just because at first stage we used a single node machine. The class of SimpleStrategy [1] is used because we wanted a simple replication factor for the Cluster. After choosing the usage of meteo as a Keyspace we create the Tables inside.

For our case we will see a human readable example for Column Family of doxaro and how is created.

CQL Command to create Column Family (Table) [1]:

```
CREATE TABLE doxaro(
Date                               date,
Time                               text,
WIND_SPEED                         decimal,
Relative_Humidity                  decimal,
Temperature                        decimal,
Barometric_Pressure                decimal,
```

```

Pyranometer_0_2000          decimal,
Precipitation                decimal,
WIND_DIRECTION              decimal,
Hourly_ETo                  decimal,
Rain_Duration                decimal,
PRIMARY KEY(Date, Time) );

```

Date attribute is the Partition Key and Time attribute the Clustering Column. Clustering Column is used to sort the data in a partition.

An example of how you can query in CQL like an SQL query is:

```
SELECT * FROM meteo.doxaro limit 10;
```

date	time	barometric_pressure	hourly_eto	precipitation	pyranometer_0_2000	rain_duration	relative_humidity	temperature	wind_direction	wind_speed
2019-04-23	0:00	991	0	0	0	0	64.4	10.3	104	1
2019-04-23	0:30	991.2	0	0	0	0	65	10.1	86	1.2
2019-04-23	0:00	990.9	0.13	0	479	0	44.5	16.8	2	12.1
2019-04-23	0:30	990.9	0.21	0	706.3	0	34.9	18.4	104	0.7
2019-04-23	1:00	990.7	0.21	0	708.5	0	37.2	19.3	127	0.8
2019-04-23	1:30	990.6	0.25	0	815.6	0	29.5	20.6	178	0.5
2019-04-23	12:00	990.9	0.25	0	930.6	0	26.7	21.2	196	1.1
2019-04-23	12:30	990.9	0.27	0	852.8	0	23.2	20.9	222	1.5
2019-04-23	13:00	991.1	0.27	0	990	0	21.4	22	353	4.7
2019-04-23	13:30	991.3	0.32	0	1005.5	0	23.1	21.3	351	0.4

Figure 4.3: Cassandra Query Result

(Apache Cassandra version used for the thesis 3.11.4)

4.2 Implementing a Data Integration Layer with NoSQL Neo4j Graph DB

In Data Warehouse section we talked about data files, tables, columns and how we store them efficiently. In this section we are going to provide an approach of how we connected each source of information that is stored in different tables in Apache Cassandra DB with each other, in a way of creating Interoperability over the storage layer.

With a closer look at attributes that each data file has, we can easily discover similarities in the context. So how can we build a feature that will fulfill all requirements of a user to get all the result of a Weather attribute from all the Tables from the Data Warehouse?

4.2.1 Neo4j

In the thesis to create the Knowledge Base Representation Layer we used the NoSQL Neo4j Graph DB. The reason of choosing this database over others [7] is because we wanted to avoid adding a Read overhead in our data, in a way to achieve great performance when we execute a query that reads data from the tables from Data Warehouse.

A short brief on Neo4j Graph DB [8] is that we have a NoSQL Database that provides a model that data is connected as Nodes and Relationships and as a result creating a Graph Model. Every Node represents an entity and can have one or more Label for grouping the Nodes. A Node can have from none to many labels. Each Node connect with other Nodes by a directed or bidirectional Relationship. A Relationship can have only one Type. Both Nodes and Relationships have properties that are represented as a name with value and are used to add meanings and qualities to a Node or a Relationship. Neo4j has it's query language that is used to query, traverse and retrieve information from a graph model and it is called Cypher. A key feature of Neo4j is the Index-free adjacency that accelerates read and write performance even if the graph model gets bigger and more complexity is added.

After this briefing and the model that we are going to use in this thesis, we are going to explain the creation of our model and its usage. The graph model that we are going to explain in our example is constructed from the UI of the Web App by an admin user. Here we are going to see what happens in the Backend and how everything is created from the scratch.

The first node is created from default to be the root Node from which

the whole graph is going to expand. The Label of the first node is WEATHERONTOLOGY and its property is name:Weather.

The Cypher command to create this node is:

```
neo4j$ MERGE (:WEATHERONTOLOGY{name:"Weather"});
```

Figure 4.4: Cypher Merge Command

*MERGE is used instead of CREATE to avoid duplicate nodes.

To check the result of this we execute the cypher query:

```
neo4j$ MATCH (n:WEATHERONTOLOGY) RETURN n;
```

Figure 4.5: Cypher to check results

Result:

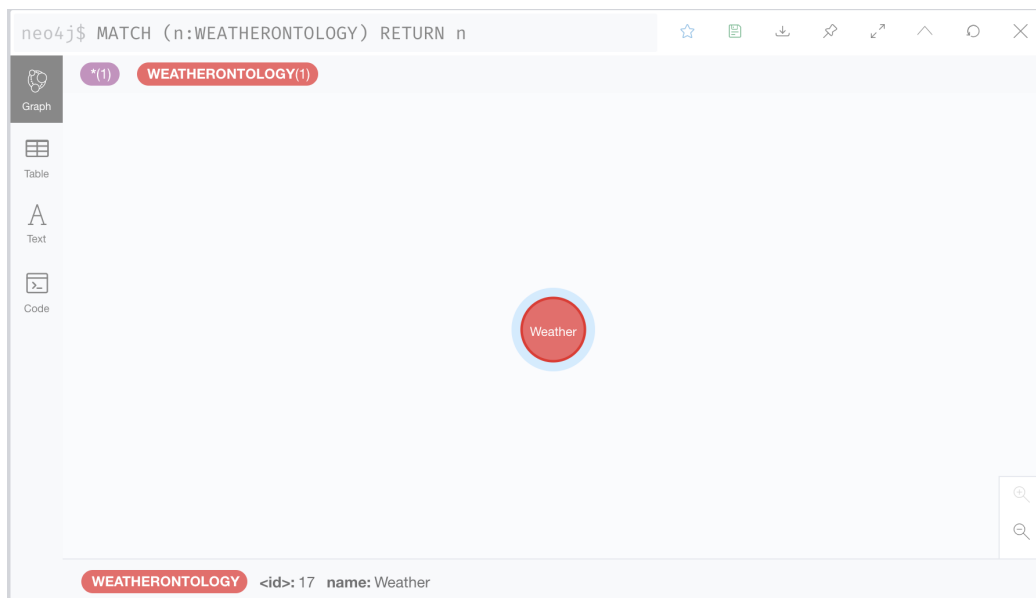


Figure 4.6: Result a node created

From the UI and admin user side now, the admin starts to construct the

graph by adding Weather Attributes, the name of the Table with which it connects and the column name that the Weather Attribute stands for.

As an example lets have a look of what happens when the admin wants to create the weather attribute Temperature and connect with the Table of doxaro and Column Temperature that has the Temperature attribute as we saw in data files.

At first stage the cypher command that are executed are:

```
MERGE(:ATTRIBUTE{name:'temperature'});
MATCH (a:WEATHERONTOLOGY),(b:ATTRIBUTE{name:'temperature'})
CREATE (a)-[:hasAttribute]→(b);
MERGE (:DBTABLE{name:'doxaro', dbtableattributes:'temperature'});
MATCH(a:ATTRIBUTE{name:'temperature'}),(b:DBTABLE{name:'doxaro', dbtableattributes:'temperature'})
CREATE (a)-[:mapTo]→(b);
```

Figure 4.7: Cypher command to create mapping of attributes

This Cypher command creates a node if not exists with the Label : ATTRIBUTE and the property name:'temperature'); Then with the MATCH command we find two nodes that we want to create a directed Relationship with the Type :hasAttribute. After this a Node with Label :DBTABLE is created if not exists with the two properties one is the name of the table and the other the column with the temperature value name:'doxaro', dbtableattributes:'temperature'. As final step we execute the MATCH command to find the node of Temperature Attribute and create a connection with Type :mapTo with the node with :DBTABLE and properties name:'doxaro', dbtableattributes:'temperature'.

To check the result of the query we execute the cypher query:

```
neo4j$ MATCH (n)-[r]→(m) RETURN n, r, m;
```

Figure 4.8: Cypher command to check the mapping results

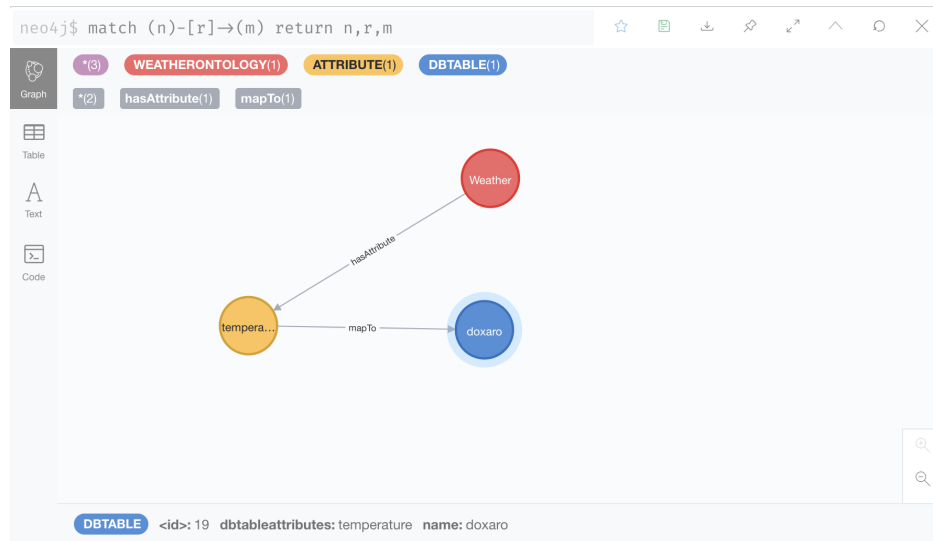


Figure 4.9: Mapping results

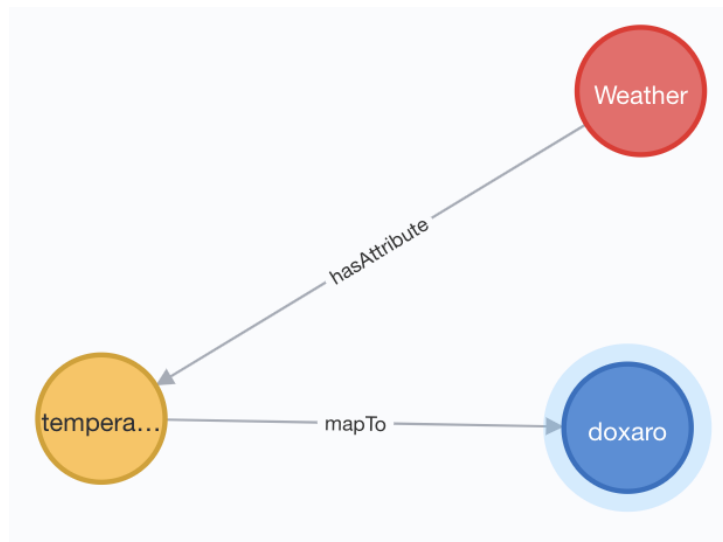


Figure 4.10: Mapping results detailed

By adding the admin user more and more weather attributes and mapping them with the tables will make a graph grow and provide the interoperability that we wanted.

As a proof of concept we created the Weather Attributes of Time Series Data Temperature, Humidity, Date and Time. When a user will ask for the Temperature Attribute values all the data that is mapped with Apache Cassandra DB Tables will be the results.

Having doxaro, tympaki a3 and mathuoc tables the following model is created:

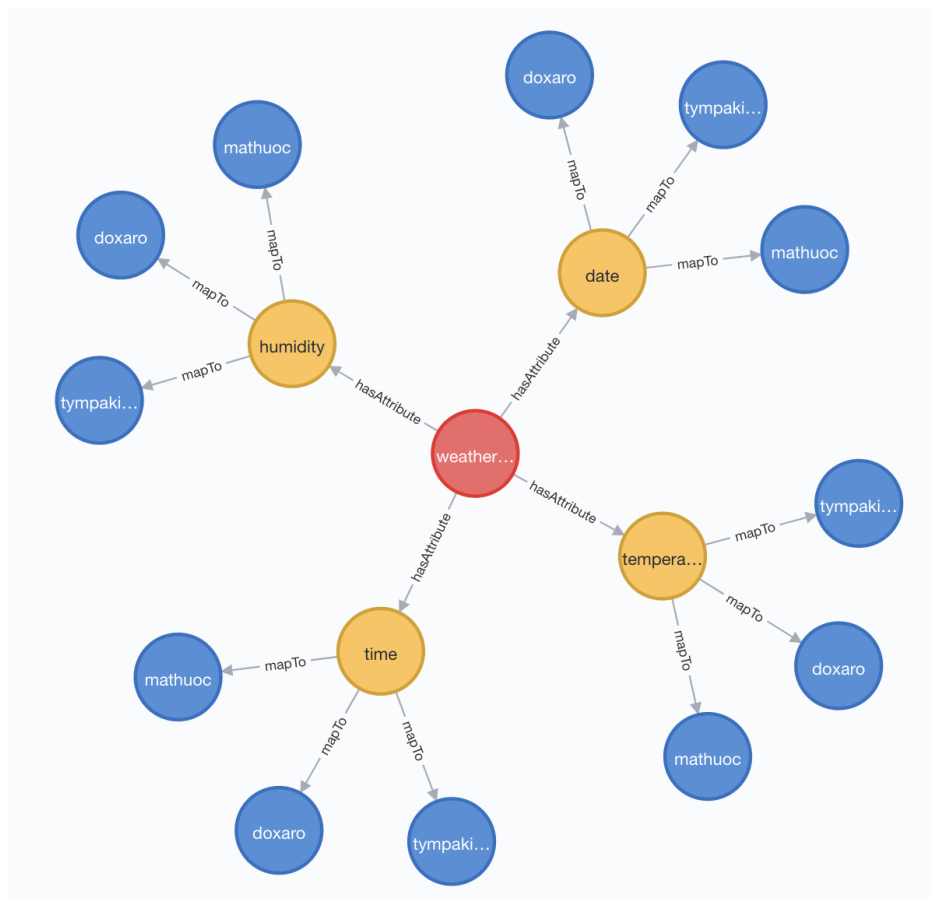


Figure 4.11: Mapping in Neo4j

(Neo4j version used for the thesis 4.2.0)

Chapter 5

Data Management

As we saw in previous section the management of the amount of data from the CSV and DAT files was an issue, so is the process of import, of those files, in the NoSQL Apache Cassandra DB that is the Storage Layer in our Application.

5.1 CQLSH Shell and COPY FROM Command

Apache Cassandra DB has a built-in tool named CQLSH. CQLSH is a built-in tool, that can execute CQL Commands from terminal. The role of this tool is to interact with the database and it is shipped with every Apache Cassandra DB package. The CQLSH tool is stable to run only with each Apache Cassandra DB version that is released with. With this tool apart from executing CQL Commands you can have the full access to make changes in the database environment like add setting, creating keyspaces, formatting the view of terminal shell and much more special commands.

One key special command for us is the COPY FROM command. The COPY FROM command copies data from files as CSV and DAT in our case. The advantage of COPY FROM command is that can achieve great import performance, with huge amount of data from a file as a batch import to database.

Usage of COPY FROM command:

```
COPY <keyspace name>.<table name> [(<columnName>, ...)]
FROM <file name>
WITH <copy option> [AND <copy option> ...]
```

Copy option that we are going to use for the import process in our case is the:

1. **HEADER:** This is a Boolean option (true/false) which specifies whether the first line of the file is the column names or not. Default value of this option is the false
2. **DELIMITER:** This option specifies the delimiter character that separates the columns in the file. The default delimiter character is the ","
3. **DATETIMEFORMAT:** This option specifies the datetime format that reads data from the file. Default format is the %Y-%m-%d %H:%M:%S%z

As a next step after the how the COPY FROM commands works , is to give an example for our example of usages. The example that we are going to have uses the m_doxaro_hourly.csv file, from Doxaro Vilalge sensors and it contains eleven (11) columns and five thousand and twenty one (5.021) rows as we described in the second (2) section.

The command to import the doxaro.dat file is:

```
COPY meteo.doxaro(Date, Time, WIND_SPEED, Relative_Humidity,
                  Temperature, Barometric_Pressure, Pyranometer_0_2000,
                  Precipitation, WIND_DIRECTION, Hourly_ETo, Rain_Duration)
FROM '/ImportFilePath/m_doxaro_hourly.csv'
WITH
HEADER = FALSE AND
DELIMITER = '\t' AND
DATETIMEFORMAT = '%Y-%m-%d';
```

Figure 5.1: COPY FROM Command

What the figure 4.1 command does?

1. Points the keyspace where the data will be imported. `meteo` is our keyspace name.
2. Points the column family(table) where the data will be imported. `doxaro` is our column family(table) name.
3. In the parenthesis we specify the column names of the table, that must be the same column number in the data file because it is exact one-to-one import, where the data is going to be stored. The column names are:
 - (a) Date
 - (b) Time
 - (c) Barometric Preasure
 - (d) Relative Humidity
 - (e) WIND_SPEED
 - (f) WIND_DIRECTION
 - (g) Temperature
 - (h) Precipitation
 - (i) Pyranometer_0_2000
 - (j) Hourly Eto
 - (k) Rain Duration
4. It specifies the path and the file from which is going to read the data for the import. The path and the file is the `/ImportFilePath/m_doxaro_hourly.csv`
5. Lastly are the option that we used for the import
 - (a) `HEADER = FALSE` defines that there is no header row in the file.
 - (b) `DELIMITER = tab` symbol defines that the columns in the file are tab separated.

- (c) DATETIMEFORMAT = '%Y-%m-%d' defines that the data of Date column must be in format like 2019-02-25.

The results of the import of each file are:

Filename	Rows	Columns	Import Time
m <u>doxaro</u> hourly.csv	5.021	11	1 sec.
m <u>tympaki a3</u> hourly.csv	9.256	11	~2 sec.
<u>mathuoc</u> 2EN AK 007 200219 14.00.00 280219 23.50.00.dat	1.213	43	1 sec.
25K.dat	25.000	43	~ 4 sec.
100K.dat	100.000	11	~ 4 sec.

Figure 5.2: Import Results

1. 25K.dat is a file created by generator with random 25K data(rows) and 43 Columns, same as the mathuoc file just to test the import performance with much more data.
2. 100K.dat is a file created by generator with random 100K data(rows) and 11 Columns, same as the doxaro file just to test the import performance with much more data.

The single node machine that was used for the import test had 2,6 GHz 6-Core Intel Core i7 CPU, 16 GB 2400 MHz DDR4, 512 Flash Storage hard disk and MacOS Operating System.

5.2 Java Process Handling for the import

Now we have a clear view of how the CQLSH shell and COPY FROM commands works, we are going to describe how we use this tool with Java in our Application.

The CQLSH is a tool that you can use it through terminal. The way of having its advantages in our Application was to execute it as a System

Process.

1. We need to create a temporary file that it will contain the COPY FROM command as we saw in previous section. CQLSH has the feature that can read a COPY FROM command from a file.
2. We use need to specify the path where Apache Cassandra CQLSH tool is installed.
3. We use Process and Runtime Classes to execute the CQLSH as a process.

Example of what we described:

```
Process p = Runtime.getRuntime().exec(  
//Step 1  
"/CassandraInstallationPath" +  
//Step 2  
"/bin/cqlsh -f " +  
//Step 3  
"/CopyFromTempFilePath/tmpCOPYFROM.txt");
```

Figure 5.3: Run CQLSH as Java Process

From the Application UI the admin user just selects the file that is to be imported in the database and the user gets back the result of execution.

The result from the import of the mathuoc file:

```
Using 11 child processes

Starting copy of meteo.mathuoc with columns [date, time, c1min, c1max,
c1avg, c1sdv, c2min, c2max, c2avg, c2sdv, c3min, c3max, c3avg, c3sdv,
se1min, se1max, se1avg, se1sdv, se2min, se2max, se2avg, se2sdv, se3min,
se3max, se3avg, se3sdv, se4min, se4max, se4avg, se4sdv, se5min, se5max,
se5avg, se5sdv, se6min, se6max, se6avg, se6sdv, se7min, se7max, se7avg,
se7sdv, p1tot].

Processed: 1214 rows; Rate: 1321 rows/s; Avg. rate: 1321 rows/s

Processed: 1214 rows; Rate: 661 rows/s; Avg. rate: 1185 rows/s

1214 rows imported from 1 files in 1.025 seconds (0 skipped).
```

Figure 5.4: Import Result

5.3 Scalability

The Scalability in our data will not be an issue, because with a certain Cluster hardware as well as the configuration of Cassandra, a linear scalability can be achieved, so as the handle of millions or billions of data.

The design approach that we implemented, will lead to steady and accurate data management, due to the usage of Cassandra mechanisms for not having conflicts with any third party developed tools that may lead to data loss or inaccurate results.

5.4 Performance Issues

Performance in database was always an issue and with from what we saw from the previous section we are going to a further any issues that may arise as the data amount increases.

Initialy we have to separate the performance to Write and Read performance.

The Write performance in Cassandra works steady and great even if the data import comes to huge amounts, a problem that is known in writes is the replicas data loss but this can be handle by the configuration of Cassandra

(adding replication factor to a number that guarantees that a batch of nodes in a Cluster have taken the data and replicas are created).

The Read performance is an issue that we have to consider and that is because of multiple factors. Factors that affect the Read performance are the Network Traffic (heavy traffic loads leads to slow read performance), the queries that may not designed as it should (creating queries that does not contain the partition keys is an issue for the read performance), the bloom filters that are used for the read process are not designed to act very fast and finally the limitation that Cassandra has to partition size and the number of values (if a table contains too many columns and values it may lead to very slow performance), so creating small sized partitions is a good practice to overcome this obstacle. But all in all the reads in Cassandra are great compared to other NoSQL Databases and provide steady data availability, it uses cache to store the data that needed often and the common Read performance problems may be solved with the right configuration, query design or hardware update.

Chapter 6

Ensuring Interoperability

In previous sections we saw the data files, the layers analysis and how we implement the import of data in Storage Layer with great performance. In this sections we will do the Interoperability Analysis and how it works from a Client Request to Web App, to Web App Response back to the Client.

The analysis that we are going to do will be as a case scenario of certain data information request from a client to our Application.

The scenario:

'A Client who uses our Web App, needs to find the temperature of weather, that is greater than 10 Celsius degree, in all locations on 21/02/2019 at 15:00 o'clock.'

Client fills the dynamic query over the data Warehouse. The Request to our Web App is to find and return data from all the Column Families (Tables) with the columns Date = '2019-02-21', Time = '15:00' and Temperature > '10'.

Query Over WarehouseDB



Figure 6.1: Query Over Warehouse DB

The full road of Client Request of Date, Time and Temperature attribute to find and get data back:

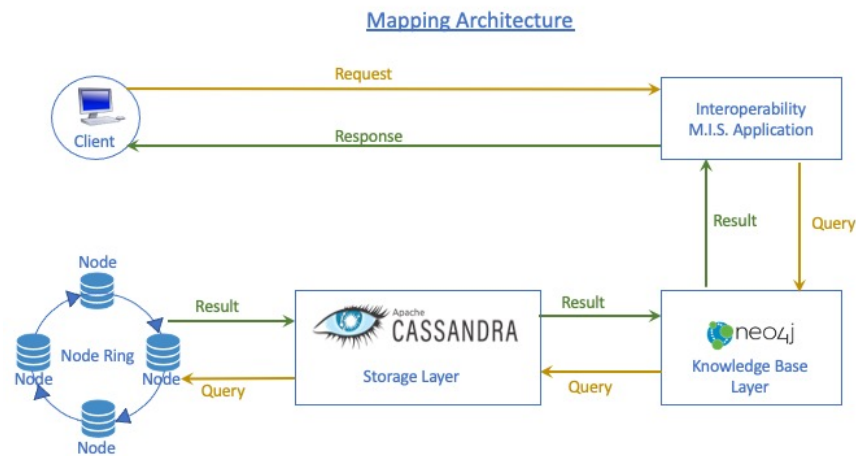


Figure 6.2: Mapping Architecture

At the time the Web App get the Client Request a Java Servlet asks the Neo4j Graph DB (Knowledge Base Layer) whether this attribute has been mapped to a Column Family (Table) and to its column.

In our scenario the Web App asks to find in which column of which Table the attribute 'Temperature' has been mapped.

The result that Neo4j returns is the nodes with the Attributes 'name' and 'dbtableattributes', that indicates the Column Family (Table) and its Column respectively, that is mapped with the Temperature, Date, Time attributes that Client Requested.

The picture below shows the return of three nodes(blue colored) that are mapped with the 'Temperature' Attribute. Similar result will be returned for the 'Date' and 'Time' attributes

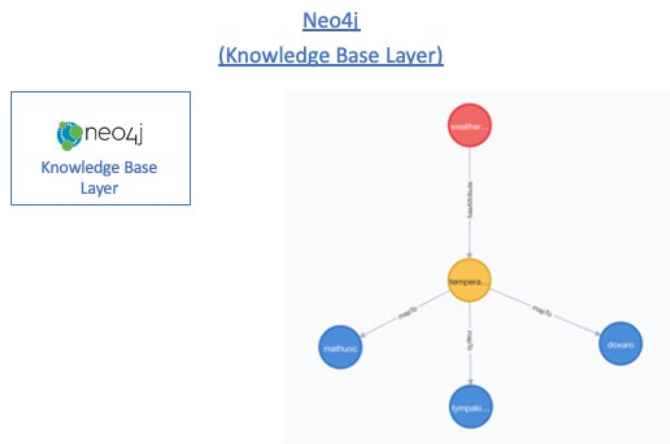


Figure 6.3: Knowledge Base Layer

As a next step the servlet handles the result from Neo4j Graph DB and it sends it to Apache Cassandra, plus the given 'Where Clause' values, querying all the Column Families(Tables) from the result of Neo4j the given 'Where Clause' values.

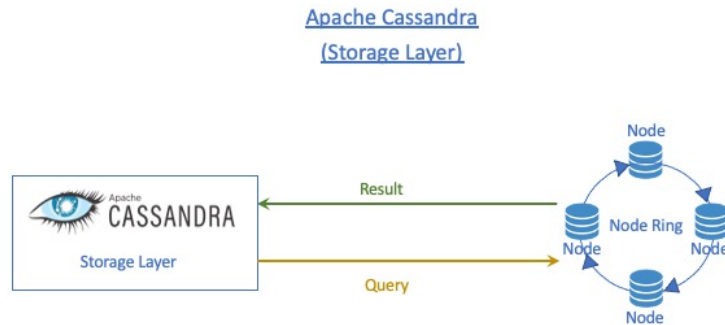


Figure 6.4: Storage Layer

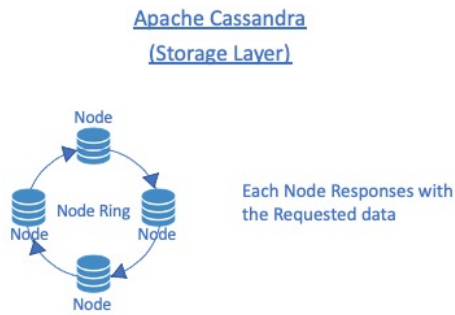


Figure 6.5: Node Response to Query

Apache Cassandra finds the nodes in the 'Ring of Nodes' (Servers in the Cluster) that contain the Requested Result and retrieves the data. The data is sent back to the Client as a Response to the initial Request.

The final result that the Client has from the Web App is shown to the picture below:

Given values:
Date = '21/02/2019' AND Time = '15:00' AND Temperature > 10

Show entries Search:

Table	Date	Time	Temperature
doxaro	2019-02-21	15:00	14.7
mathuoc	2019-02-21	15:00	14.74000
tympaki_a3	2019-02-21	15:00	17.6

Showing 1 to 3 of 3 entries Previous Next

Figure 6.6: Query result to Client UI

A second scenario:

'A Client who uses our Web App, needs to find the temperature of weather, that is greater than 15 Celsius degree and less than 30 Celsius degree, humidity greater than 15 percentage , in all locations from date 21/02/2019 to 25/02/2019, from time 15:00 o'clock to 19:00 o'clock.'

As we show in previous senario the route that it follows will be the same and here we are going to see the parameters for the dynamic query and the results as a proof of concept that Interoperability works in range queries.

From the results of the query we can see that mathuoc does not fulfil the query at all, doxaro has only one record for this query and tympaki_a3 has more records regarding the query request.

Query Over WarehouseDB

AND
Add rule

Date	greater or equal	21/02/2019	Delete
Date	less or equal	25/02/2019	Delete
Time	greater or equal	15:00	Delete
Time	less or equal	19:00	Delete
Temperature	greater	15	Delete
Temperature	less	30	Delete
Humidity	greater	15	Delete

Apply
Reset

Figure 6.7: Query Over Warehouse DB 2

Given values:
 Date >= '21/02/2019' AND Date <= '25/02/2019' AND Time >= '15:00' AND Time <= '19:00' AND Temperature > 15 AND Temperature < 30 AND Humidity > 15

Show 10 entries Search:

Table	Date	Time	Humidity	Temperature
doxaro	2019-02-22	15:00	15.3	46.2
tympaki_a3	2019-02-21	15:00	17.6	61.3
tympaki_a3	2019-02-21	15:30	17.1	63
tympaki_a3	2019-02-21	16:00	17	62.3
tympaki_a3	2019-02-21	16:30	16.6	64.6
tympaki_a3	2019-02-21	17:00	16.3	66.9
tympaki_a3	2019-02-21	17:30	15.2	67.7
tympaki_a3	2019-02-22	15:00	17.1	64.6
tympaki_a3	2019-02-22	15:30	16.3	65.8
tympaki_a3	2019-02-22	16:00	16.3	64.3

Showing 1 to 10 of 17 entries Previous 1 2 Next

Figure 6.8: Query Results to Client UI 2.1

Given values:
 Date >= '21/02/2019' AND Date <= '25/02/2019' AND Time >= '15:00' AND Time <= '19:00' AND Temperature > 15 AND Temperature < 30 AND Humidity > 15

Show 10 entries Search:

Table	Date	Time	Humidity	Temperature
tympaki_a3	2019-02-22	16:30	16.3	62.6
tympaki_a3	2019-02-22	17:00	16.2	67.8
tympaki_a3	2019-02-22	17:30	15.1	70.9
tympaki_a3	2019-02-24	15:00	15.1	67.3
tympaki_a3	2019-02-24	15:30	15.8	63.1
tympaki_a3	2019-02-24	16:00	15.7	62.7
tympaki_a3	2019-02-24	16:30	15.5	63

Showing 11 to 17 of 17 entries Previous 1 2 Next

Figure 6.9: Query Results to Client UI 2.2

A third scenario:

'A Client who uses our Web App, needs to find the temperature and the humidity of weather, that are greater than 15 Celsius degree and 15 percentage, in all locations at date 21/02/2019, from time 18:00 o'clock to 21:00 o'clock.'

Query Over WarehouseDB

The screenshot shows a query builder interface with a dark background. At the top left, there is a blue button labeled 'AND'. At the top right, there is a green button labeled 'Add rule'. Below these are five rows of query conditions, each with a 'Delete' button on the right:

- Row 1: Date (dropdown), equal (dropdown), 21/02/2019 (input), Delete (button)
- Row 2: Time (dropdown), greater or equal (dropdown), 18:00 (input), Delete (button)
- Row 3: Time (dropdown), less or equal (dropdown), 21:00 (input), Delete (button)
- Row 4: Humidity (dropdown), greater (dropdown), 10 (input), Delete (button)
- Row 5: Temperature (dropdown), greater (dropdown), 10 (input), Delete (button)

At the bottom left, there are two buttons: 'Apply' (blue) and 'Reset' (yellow).

Figure 6.10: Query Over Warehouse DB 3

Given values:
 Date = '21/02/2019' AND Time >= '18:00' AND Time <= '21:00' AND Humidity > 10 AND Temperature > 10

Show entries Search:

Table	Date	Time	Humidity	Temperature
doxaro	2019-02-21	18:00	78.8	11.7
doxaro	2019-02-21	18:30	82.8	11.2
mathuoc	2019-02-21	18:00	82.40000	13.37000
mathuoc	2019-02-21	18:30	83.20000	13.12000
mathuoc	2019-02-21	18:20	83.90000	13.00000
mathuoc	2019-02-21	18:30	84.40000	12.96000
mathuoc	2019-02-21	18:40	84.80000	12.91000
mathuoc	2019-02-21	18:50	85.10000	12.81000
mathuoc	2019-02-21	19:00	85.50000	12.80000
mathuoc	2019-02-21	19:30	86.00000	12.66000

Search Table Search Date Search Time Search Humidity Search Temperature

Showing 1 to 10 of 26 entries Previous Next

Figure 6.11: Query Results to Client UI 3.1

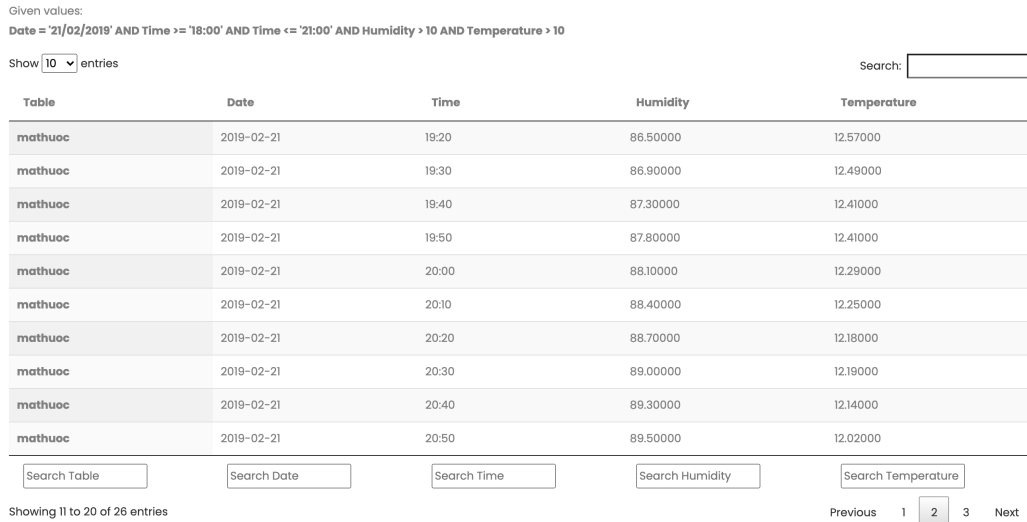


Figure 6.12: Query Results to Client UI 3.2

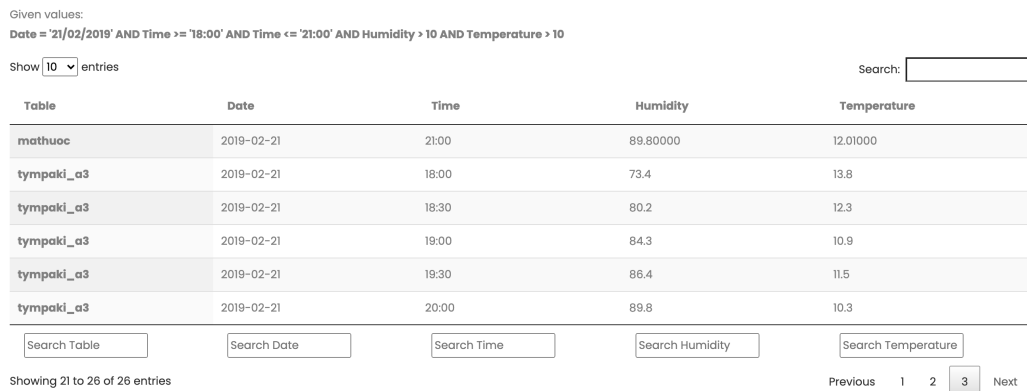


Figure 6.13: Query Results to Client UI 3.3

All the above steps show how the Interoperability between many different data sources can be implemented, by having performance, accuracy and consistency as key parts for the best result.

The performance part comes from the very fast and steady reads that Neo4j offers as well as the read in Cassandra database that is used to store the data. Neo4j is responsible for the fast reads due to the almost zero read time to find the mappings in the graph by using caching and the Shortest Path Algorithm. Cassandra can have great read performance with the right configuration as we saw in previous section.

Accuracy and Consistency comes with the Cassandra data availability and its fault tolerance mechanisms and also the right mapping that the administration has made from the administration panel UI of M.I.S. Web App in Neo4j.

With all the above parts we can ensure the efficient Interoperability over heterogenous sources of information.

6.0.1 Incremental Addition of Data Source

The incremental Addition of Data Source is not an issue that should consider us in our thesis and that comes from that it has an abstract design without any strict rules of mapping or database creation. So even if a new source or sources of information may need to inserted in our M.I.S. Application the only need is the right database design and also the right mapping from an expert user between the Knowledge Base Layer and the Storage Layer.

6.0.2 Lesson Learned

Lessons Learned from this experience is that with modern technologies and technics, we can achieve great results in data management. We can take advantage of hardware that we have nowadays used High-Performance Distributed Systems, using a variety of Databases, Relational and NoSQL by each case of use and finally built a hybrid application, which gives solutions

to problems that years before would be difficult to solve or would be costly enough not having the opportunity to work on and find their potentials.

Chapter 7

M.I.S. Web App Overview

After all, we have a great knowledge of how everything works from previous sections, it is time to have a look at the final product, the Management Information System(M.I.S.) Web App, its Contents and in Use.

7.1 M.I.S. Web App Capabilities

The contest of the Management Information System(M.I.S.) Web App are the following:

1. Data Table
2. Query OWDB(Over Warehouse DB)
3. Line Chart
4. 3D Plot
5. Import File
6. Execute CQL
7. Settings
8. Users

Lets now have a deeper look of what the above contents are. In every section access has been implemented for registered users that has two types, 'admin' and 'user'. In each conctect the access is written next to title.

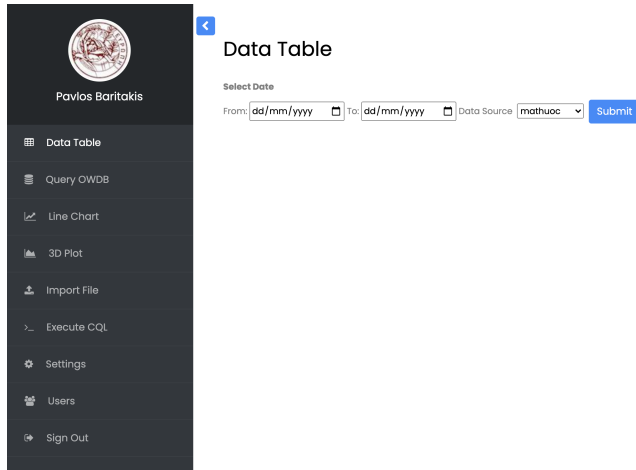


Figure 7.1: Web App User Interface index page

7.1.1 Data Table (Access: admin, user)

Data Table is used for querying a Column Family (Table) by Date and get all the results in a Table format with rows and columns.

The Table has the following features:

1. Total Search of certain data value in all of the columns of the Table
2. Column Search of certain data value
3. Column ordering
4. Paging results
5. Fixed columns (keeps data of certain column/s in position in a horizontal scroll)

User fills a StartDate(From), an EndDate(To) form. and then selects the Column Family(Table) that wants to retrieve data from and submits a request.

The response to the request is a result from the chosen Column Family(Table) formatted in a Table.

Data Table

Select Date
 From: To: Data Source:

Given values:
 From Date: 20/02/2019 To Date: 28/02/2019

Show entries Search:

DATE	TIME	C1AVG	C1MAX	C1MIN	C1SDV	C2AVG	C2MAX	C2M
2019-02-20	14:00	8.19000	10.78000	2.68700	1.39000	8.32000	10.41000	4.560
2019-02-20	14:10	8.25000	10.58000	5.19000	0.96600	8.42000	10.21000	6.123C
2019-02-20	14:20	7.33900	10.47000	3.90800	1.07300	7.55600	9.80000	4.157C
2019-02-20	14:30	7.14100	9.71000	4.11200	0.95900	7.26900	9.90000	4.460
2019-02-20	14:40	6.55800	8.74000	3.70500	1.06800	7.06800	9.20000	4.510C
2019-02-20	14:50	6.79500	9.30000	3.80700	0.99200	7.00100	9.15000	4.107C
2019-02-20	15:00	7.34900	9.71000	4.82500	0.81600	7.47500	9.10000	4.883
2019-02-20	15:10	6.93100	9.20000	3.24700	0.99300	7.12600	9.55000	4.560

Showing 1 to 10 of 1,212 entries Previous **1** 2 3 4 5 ... 122 Next

Figure 7.2: Data Tables Usage

7.1.2 Query OWDB (Access: admin, user)

The Query OWDB works as a dynamic query builder for the interoperability, from which a user can execute complex queries from the UI without the need of knowledge of any query language. User just inputs parameters, operators and values of data to get the wanted results in a Table.

The Table has the following features:

1. Total Search of certain data value in all of the columns of the Table
2. Column Search of certain data value
3. Column ordering
4. Paging results
5. Fixed columns (keeps data of certain column/s in position in a horizontal scroll)

User can build a query by choosing parameters that wants results from, chooses the operator of each parameter should be and finally fills the values of the given parameters that wants to retrieve results regarding the operator. User request is created by the submit of the form with parameters, operators and values.

The response to the request is a result from all the Column Families (Tables) regarding the mapping that has been implemented for interoperability, formatted in a Table.

Query Over WarehouseDB

The screenshot shows a query builder interface with the following components:

- Operator:** AND
- Rules:**
 - Rule 1: Date equal 20/02/2019 (Delete)
 - Rule 2: Date equal 21/02/2019 (Delete)
 - Rule 3: Date equal 22/02/2019 (Delete)
 - Rule 4: Time equal 15:00 (Delete)
- Buttons:** Add rule, Apply, Reset

Figure 7.3: Dynamic Query Builder Over Warehouse DB

Given values:
Date = '20/02/2019' AND Date = '21/02/2019' AND Date = '22/02/2019' AND Time = '15:00'

Show 10 entries Search:

Table	Date	Time
doxaro	2019-02-20	15:00
doxaro	2019-02-21	15:00
doxaro	2019-02-22	15:00
mathuoc	2019-02-20	15:00
mathuoc	2019-02-21	15:00
mathuoc	2019-02-22	15:00
tympaki_a3	2019-02-20	15:00
tympaki_a3	2019-02-21	15:00
tympaki_a3	2019-02-22	15:00

Showing 1 to 9 of 9 entries Previous 1 Next

Figure 7.4: Result of Dynamic Query

7.1.3 Line Chart (Access: admin, user)

The Line Chart works for statistical - analytical purposes. It creates a graphical interface for one or multiple attributes and it shows the results in colored lines in 'x' and 'y' axis. With the mouse over each line user gets the exact info of values at that point of line.

User fills a Date Range (From - To) and a Hour Range (From - To) form, then selects an attribute or multiple attributes that wants to have results for and finally choses the 'x' axis that wants the data to be represented and can be in 'Date' or in 'Time'. A request is created by the form submit.

The response to the request is the results in line chart graphical representation showing color separated attributes lines. The 'y' axis shows the values of each attribute and the 'x' axis the 'Date' or the 'Time' that attribute value is represented. Mouse position on each line shows information of the exact name of the attribute, its value and 'Date' or 'Time' that represents.

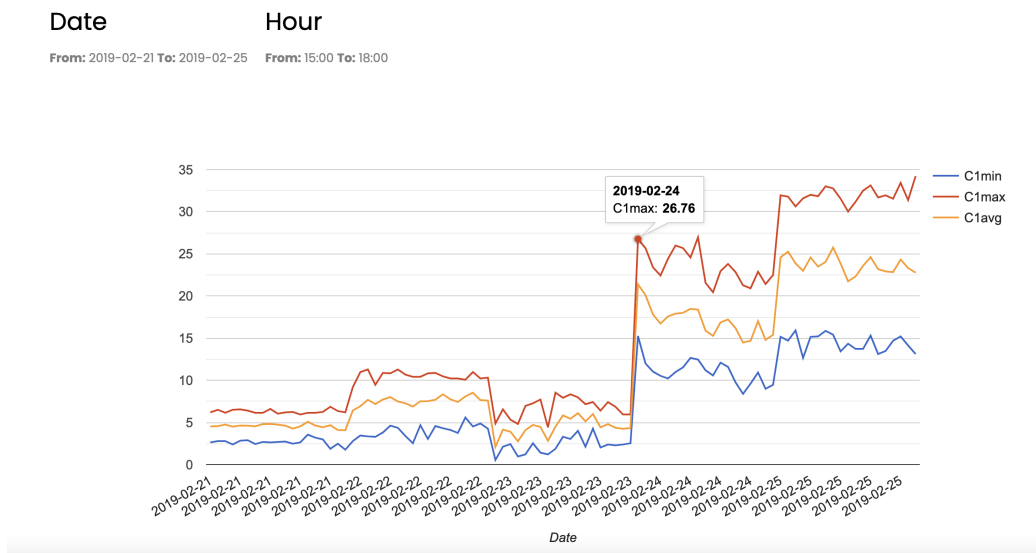


Figure 7.5: Line Chart plot

7.1.4 3D Plot (Access: admin, user)

3D Plot is used to create a 3D Sparce Data graphical interface. Each dot in sparce representation has information of each record in the result. The 'x' axis represents the 'Date', the 'y' axis represents value of data and the 'z' axis represents the 'Time'.

The 3D plot has the following features:

1. Zoom In and Out in 3D Plot
2. Download the plot as .png image
3. Orientation modes of the plot

User adds data for Date Range (From - To) for Hour Range (From - To) and finally selects one attribute from which wants to have results. A request is created by the form submit.

The response from the request is the results in Sparce Data graphical representation.

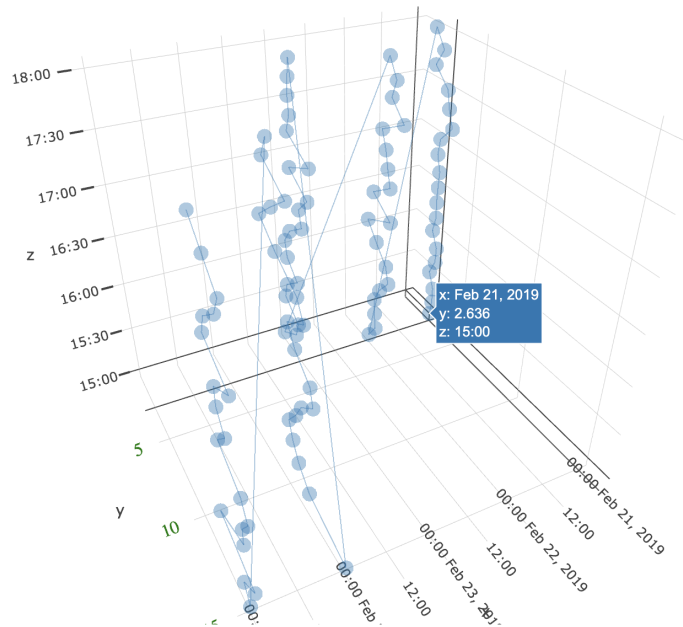


Figure 7.6: 3D Sparce Data plot

7.1.5 Import File (Access: admin)

The Import File has the usage of importing data from the .csv and .dat in the Column Family (Table) that we want as it is described in section 4. The admin choses the file for the import and it submits a request for the import process.

The response to the request is the result of the import in depth analysis.

Import File: import.dat

Result message:

Using 11 child processes

Starting copy of csdtest.mathuoc with columns [date, time, clmin, clmax, clavg, clsdv, c2min, c2max, c2avg, c2sdv, c3min, c3max, c3avg, c3sdv, selmin, selmax, selavg, selsdv, se2min, se2max, se2avg, se2sdv, se3min, se3max, se3avg, se3sdv, se4min, se4max, se4avg, se4sdv, se5min, se5max, se5avg, se5sdv, se6min, se6max, se6avg, se6sdv, se7min, se7max, se7avg, se7sdv, plltot].

Processed: 1214 rows; Rate: 1191 rows/s; Avg. rate: 1191 rows/s

Processed: 1214 rows; Rate: 595 rows/s; Avg. rate: 1078 rows/s

1214 rows imported from 1 files in 1.126 seconds (0 skipped).

Figure 7.7: Import Data File Result in UI

7.1.6 Execute CQL (Access: admin)

With the Execute CQL content we give the opportunity to the admin to write and execute any abstract CQL command that might need to make changes in any Column Family(Table) or Keyspace.

The user writes the CQL command and a request is created by the form submit.

The response from the request is the result of CQL Execution in a Table.

Execute CQL

Write and Execute a CQL Query:

```
SELECT * FROM meteo.mathuoc LIMIT 5
```

> Execute 

Figure 7.8: Terminal to execute CQL Commands in UI

7.1.7 Settings (Access: admin)

In the Setting the admin has the full access of the Web App. In this content admin can control the usage of:

1. Mediator Table
2. Export of Column Family (Table) in JSON File

7.1.7.1 Mediator Table

In Mediator Table the admin can only insert a mapping of attribute with a Column Family (Table) and an attribute of that Table. Deletes and Updates are very sensitive processes for the mapping and it is safer to be executed by an advanced admin user and that is the reason of not allowing this process from the UI to any admin.

Mediator Table

Insert Mapping Attribute

Attribute	Table	TableAttributes
time	doxaro	Time
time	mathuoc	Time
time	tympaki_a3	Time
humidity	doxaro	Relative_Humidity
humidity	mathuoc	SE6avg
humidity	tympaki_a3	Relative_Humidity
temperature	doxaro	temperature
temperature	mathuoc	SE3avg
temperature	tympaki_a3	Temperature
date	doxaro	Date
date	mathuoc	Date
date	tympaki_a3	Date

Figure 7.9: Mediator Table with attribute mapping

7.1.7.2 Export of Column Family (Table) in JSON File

In the export of Table in JSON File an admin user choses the Table that wants its data to be exported in JSON format.

Request is created by choosing the Table submit.

The response to the request is the JSON File.

```
{
  "mathuoc": [
    {
      "SE1sdv": 5.10888,
      "SE2max": 253.91020,
      "C1avg": 2.94000,
      "SE3sdv": 0.05100,
      "SE7min": -0.01000,
      "SE7avg": -0.01000,
      "SE4max": 0.00000,
      "SE5min": 982.06940,
      "SE5avg": 982.20050,
      "SE6max": 89.20000,
      "SE7sdv": 0.00100,
      "Time": "00:00",
      "C3min": 2.10700,
      "C3avg": 2.98800,
      "SE5sdv": 0.11073,
      "C1min": 2.38100,
      "SE1min": 183.89059,
      "SE1avg": 168.98421,
      "SE3min": 12.08000,
      "SE3avg": 12.24000,
      "SE7max": 0.00000,
      "C3max": 3.60000,
      "C1max": 3.45000,
      "C2sdv": 0.24300,
      "SE2sdv": 4.42619,
      "SE1max": 153.50000,
      "SE3max": 12.33000,
      "SE4sdv": 0.00000,
      "SE6min": 88.50000,
      "SE6avg": 88.90000,
      "Pltot": 0.00000,
      "SE5max": 982.84430,
      "SE6sdv": 0.20400,
      "C2min": 2.29200,
      "C2avg": 2.96500,
      "Date": {
        "year": 2019,
        "month": 2,
        "day": 22
      },
      "SE2min": 276.24219,
      "SE2avg": 161.93330,
      "SE4min": 0.00000,
      "SE4avg": 0.00000,
      "C3sdv": 0.25300,
      "C2max": 3.50200,
      "C1sdv": 0.20400
    }
  ],
}
```

Figure 7.10: Export sample of JSON File

7.1.7.3 Users Table

In Users Table an admin can register a new member of the application with the role of simple user or admin user, with full access in features of the Web App.

User Table UI:

Users Table

[Insert](#)


Username	Email	Firstname	Lastname	Role	Edit	Delete
pbaritak	pbaritak@csd.uoc.gr	Pavlos	Baritakis	user	 Edit	 Delete
admin	admin@csd.uoc.gr	Admin	Admin	admin	 Edit	 Delete

Figure 7.11: Users Table UI

Chapter 8

Conclusion and Future Work

As the data from Meteorological sensors continuously increases, the need of solutions to manage this kind of data as well as combine them from heterogeneous sources of information for the creation of Knowledge Base, has been arose.

Our work addresses this problem by using a Layered based Web Application, not only to solve the mapping of attributes of heterogeneous databases but also to store the data efficiently. The usage of modern technologies in distributed systems gave us the opportunity to develop a high performance and scalable Web App that offers an Interoperability over a data warehouse of heterogeneous sources of information.

Our approach uses two Layers of to handle the Knowledge Base and the Storage. The solution to the Knowledge Base came with the use of NoSQL Neo4j Graph DB. Neo4j is used to map attributes between the different sources of information having the role of mediator from the Client to the data warehouse by building dynamic queries to data warehouse regarding the Client requests. Neo4j has high performance reads, despite the graph scalability. The second layer is the Storage Layer. In that layer all the data from heterogeneous sources are stored in Column Families (Tables) in NoSQL database the Apache Cassandra. This database offers us high performance writes and can handle huge amounts of imports to a Cluster in parallel.

Apache Cassandra storage system works great even if the data that has to handle will increase rapidly.

Our future work is to develop an extension of this Web App that will support the translation of RDF Triple Stores to Neo4j Graph Database, that will drive to a semantically more accurate mapping approach by importing ontologies with relations and attributes to a Knowledge Graph.

Bibliography

- [1] Cassandra: The Definitive Guide, 3rd Edition by Jeff Carpenter, Eben Hewitt Released April 2020 Publisher(s): O'Reilly Media, Inc. ISBN: 9781098115166

- [2] <https://www.datastax.com/dev/academy>

- [3] <https://neo4j.com/graph-databases-book/>

- [4] <https://neo4j.com/developer/get-started/>

- [5] Hendawi, Abdeltawab & Gupta, Jayant & Liu, Jiayi & Teredesai, Ankur & Ramakrishnan, Naveen & Shah, Mohak & El-Sappagh, Shaker & Kwak, Kyung & Ali, Mohamed. (2019). Benchmarking large-scale data management for Internet of Things. *The Journal of Supercomputing*. 75. 10.1007/s11227-019-02984-6.

- [6] Gilbert, Seth & Lynch, Nancy. (2012). Perspectives on the CAP Theorem. *IEEE Computer*. 45. 30-36. 10.1109/MC.2011.389.

- [7] Pandey, Santosh & Joshi, Erika & Maharjan, Manish & Karki, Nissan. (2018). A research on Architectural and Performance Comparison of Relational Database, NoSQL and Graph database (Neo4j). 10.13140/RG.2.2.33279.25762/1.
- [8] <https://neo4j.com/docs/getting-started/current/graphdb-concepts/#graphdb-nodes>
- [9] Lenzerini, Maurizio. (2002). Data Integration: A Theoretical Perspective. Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. 233-246. 10.1145/543613.543644.
- [10] K. Chaturvedi and T. H. Kolbe, InterSensor Service: Establishing Interoperability over Heterogeneous Sensor Observations and Platforms for Smart Cities, 2018 IEEE International Smart Cities Conference (ISC2), Kansas City, MO, USA, 2018, pp. 1-8, doi: 10.1109/ISC2.2018.8656984.
- [11] Ramar, Kaladevi & Mohan, Geetha & Narayanasamy, P.. (2016). Ontological based interoperability and integration framework for heterogeneous weather systems. 39. 185-192.
- [12] M. G. Kibria, S. Ali, M. A. Jarwar & I. Chong, A framework to support data interoperability in web objects based IoT environments, 2017 International Conference on Information and Communication Technology Convergence (ICTC), 2017, pp. 29-31, doi: 10.1109/ICTC.2017.8190935.
- [13] B. Ahlgren, M. Hidell and E. C. -. Ngai, "Internet of Things for Smart Cities: Interoperability and Open Data," in IEEE Internet Computing, vol. 20, no. 6, pp. 52-56, Nov.-Dec. 2016, doi: 10.1109/MIC.2016.124.

[14] <https://www.unidata.ucar.edu/software/netcdf/>

[15] <https://www.omnisci.com/learn/geospatial>