



VIETNAMESE–GERMAN UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

and

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

DEPARTMENT OF INFORMATICS

Bachelor's Thesis in Computer Science

Data Warehouse and Data Lake in the Cloud for Healthcare Data

Le Dinh Trung Hieu



VIETNAMESE–GERMAN UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

and

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

DEPARTMENT OF INFORMATICS

Bachelor's Thesis in Computer Science

Data Warehouse and Data Lake in the Cloud for Healthcare Data

Author:	Le Dinh Trung Hieu
Supervisor:	Dr. Huynh Trung Hieu
Co-supervisor:	Dr. Tran Hong Ngoc
Submission Date:	19.04.2022

I confirm that this bachelor's thesis in computer science is my own work and I have documented all sources and material used.

Binh Duong, Vietnam, 19.04.2022

Le Dinh Trung Hieu

A handwritten signature in black ink, appearing to read 'Hieu', followed by a long, sweeping horizontal stroke.

This thesis is dedicated to *myself*.

“In god we trust. All others must bring data.”

—— W. Edwards Deming

ACKNOWLEDGMENTS

I would like to thank my supervisors, Dr. Huynh Trung Hieu and Dr. Tran Hong Ngoc, for their patience and support. I would not have been able to finish without their incredibly helpful guidance, inspiration, and enthusiasm during my working on the thesis.

I also want to express my sincere appreciation to my family and friends for their unconditional support during this challenging time when COVID19 changed the way we would study and work.

Thanks to them, this bachelor thesis would forever be a beautiful memory and helpful experience in my life.

Abstract

The healthcare industry is not only one of the largest and most complex sectors in the world economy, but it is also one of the most important aspects of our lives. Nowadays, healthcare organizations are one of the biggest producers of data. With the growth of Data Science and Big Data, the healthcare industry is undergoing tremendous change as it seeks to understand the bigger picture of healthcare and improve patient outcomes. The thesis introduces the concept of modern data storage and stack. The theoretical part focuses on the definition and purpose of data warehouse, data lake, and the ETLT process. In addition, cloud computing and modern data stack are discussed, with the focus on the Google Cloud Platform. In this thesis, we will build a cloud data warehouse based on OMOP CDM and data lake using realistic patient data with the support of the cloud and modern data stack applications. Besides, we conduct research by comparing different models and applications to select the best components for our solutions, which will make a significantly better approach for data and healthcare-related people to understand problems and make decisions.

Acronyms

AI Artificial Intelligence. 18, 26

API Application Programming Interface. 18

AWS Amazon Web Services. 8, 17, 39

BI Business Intelligence. 3, 8

CCPA California Consumer Protection Act. 34

CDM Common Data Model. v, 1, 2, 22, 23

DAG Directed Acyclic Graphs. 27, 28, 39, 40

dbt Data build tool. 18, 29, 46, 47

DL Data Lake. 3

DW Data Warehouse. 3, 4, 29

EHR Electronic Health Record. 1

ELT Extract Load Transform. 3, 11

EMR Electronic Medical Records. 1

ETL Extract Transform Load. 1, 3, 4, 10, 18

ETLT Extract Transform Load Transform. v, 9, 17, 34

GCP Google Cloud Platform. 18, 22, 26, 27, 34, 39, 43

GCS Google Cloud Storage. 20, 39, 40, 43

GDPR General Data Protection Regulation. 10, 12, 34

HDFS Hadoop Distributed File System. 8

HIPAA Health Insurance Portability and Accountability Act. 12, 34

IAM Identity and Access Management. 34, 37

JSON JavaScript Object Notation. 7

LDAP Lightweight Directory Access Protocol. 32

ML Machine Learning. 26

OHDSI Observational Health Data Sciences and Informatics. 22, 24

OLAP Online Analytical Processing. 4, 22, 24

OLTP Online Transaction Processing. 4, 10, 22

OMOP Observational Medical Outcomes Partnership. v, 1, 2, 18, 22, 23

OSS Open-source software. 18

PHI Protected Health Information. 10, 22, 34

PII Personally Identifiable Information. 10, 22, 34

S3 Simple Storage Service. 8, 17, 39

SQL Structured Query Language. 29

SSH Secure Shell. 26

UI User Interface. 28, 29

VM Virtual Machine. 26

VPC Virtual Private Cloud. 38

XML Extensible Markup Language. 7

List of Figures

2.1	Row-wise storage [1]	6
2.2	Column-wise storage [1]	6
2.3	ETL Process[2]	10
2.4	ELT Process[2]	11
2.5	Top Cloud Providers	12
3.1	Proposed Architecture	17
3.2	Overview of all tables in the CDM	23
3.3	Top Cloud Data Warehouses	25
3.4	Cloud provider speed comparison for creating VM and SSH access [3]	26
3.5	Top Cloud Computational Services	27
3.6	DAG Example	28
3.7	Top Popular OSS Data Orchestrator tools	28
3.8	Data build tool [4]	29
3.9	Data Visualizations and Business Intelligence Applications	30
3.10	Compare Star History off Metabase, Redash and Superset	31
3.11	Open-Source Modern BI Tools in Compute Engine	32
3.12	Top Data Catalog - Data discovery, observability and governance applications	32
3.13	Delta Lake and Apache Parquet	33
4.1	Service Account JSON file	37
4.2	Extract DAG	39
4.3	Extract DAG	39
4.4	Created Bucket in GCS	40
4.5	Transform DAG	40
4.6	Creating Cluster and Submitting PySpark Job DAGs	41
4.7	Configuring Transformation Code for Delta Lake	42
4.8	Patient tables in Delta Lake Format	42
4.9	Creating External Tables and Data Quality Check for BigQuery	43
4.10	Load DAG Part 1	44
4.11	Load DAG Part 2	45
4.12	dbt Transformation using SQL	46
4.13	Data Lineage for for table drug_era part 1	47
4.14	Data Lineage for for table drug_era part 2	47
4.15	dbt Failed Run	48
4.16	dbt run Drug_Era table information	48

4.17	Install BigQuery Driver	49
4.18	Install BigQuery Driver fix	49
4.19	Plugins for Metadata Ingestion Sources	49
4.20	Adding Superset as Data Platform request	50
4.21	All Delta Lake Tables in GCS	51
4.22	Delta Lake Files for the CONCEPT_RELATIONSHIP table	52
4.23	Tables in BigQuery	53
4.24	Table Preview	54
4.25	Query Estimation	54
4.26	Data Orchestrator Platform ETL process	55
4.27	SQL Explore and Preview	55
4.28	Superset Charts	56
4.29	Patient Demographic Dashboard	57
4.30	Drug Usage Trend Dashboard	57
4.31	Datahub Homepage	58
4.32	Datahub Homepage Activities Summary	59
4.33	User's activities reports	60
4.34	Searching data or applications in Datahub	61
4.35	Data Lineage of Superset Dashboard and BigQuery	62
4.36	Data Lineage of dbt and BigQuery	62
4.37	SQL query with the help of dbt built-in function	63
4.38	Data Lineage for table drug_era part 1	63
4.39	Data Lineage for table drug_era part 2	64
4.40	dbt Auto-generated Documentation	64
5.1	Great Expectations Docs Example	66
5.2	Great Expectations deployment in GCP [5]	67
5.3	Streaming data with Dataflow pipeline[6]	67

List of Tables

2.1	Denormalization Table	4
2.2	Denormalization Table	5
2.3	Normalization Allergies Table	5
2.4	Normalization Encounter Table	5
2.5	Normalization Patient Table	5
2.6	On-Premises and Cloud Computing Comparision	14
2.7	On-Premises and Cloud Computing Comparision	15
3.1	Top OSS Applications Data Council Survey 2021 [7]	19
3.2	Bus Matrix for Synthea Data	20
3.3	Data dictionary for Synthea Data	21
3.4	Data dictionary for Transformation of Person table	24

Contents

Acknowledgments	iv
Abstract	v
Acronyms	vi
List of Figures	viii
List of Tables	x
1 Introduction	1
1.1 Motivation	1
1.2 Thesis Structure	2
2 Literature Review	3
2.1 Data Warehouse	3
2.1.1 Introduction	3
2.1.2 Characteristics	4
2.2 Data Lake	7
2.2.1 Introduction	7
2.2.2 Characteristic	9
2.3 ETL and ELT	10
2.3.1 Traditional ETL approach	10
2.3.2 The Rise of ELT	11
2.4 Cloud Computing	12
2.4.1 Introduction	12
2.4.2 Benefits	13
2.5 Modern Data Stack	16
3 Solution Architecture	17
3.1 Proposed Architecture	17
3.2 Research on Hybridized ETLT approach	18
3.3 Propose Model for Data Lake	20
3.4 Related Work for Data Warehouse	22
3.5 Data Warehouses Comparison	24
3.6 Cloud Compute Services Comparison	26
3.7 Data Processing Services Comparison	26

3.8	Cloud Object Storage Comparison	27
3.9	Data Orchestrators Comparison	27
3.10	Data Transformation Selection	29
3.11	Data Visualization Comparison	29
3.12	Data Catalogs Comparison	32
3.13	Data Storage Format Selection	33
3.14	Data Privacy and Security	34
4	Implementation and Integration	36
4.1	Prerequisite Knowledge	36
4.2	Setup	37
4.3	Extract	38
4.4	Transform	40
4.5	Load	43
4.6	Second Transform	46
4.7	Integration and Present Insights	48
4.8	Results	50
4.8.1	Data Lake Storage in GCS	50
4.8.2	Data Warehouse in BigQuery	52
4.8.3	Data Orchestrator platform with Apache Airflow	54
4.8.4	Data Visualizations platform with Apache Superset	55
4.8.5	Data Catalog platform with DataHub	58
4.8.6	Second Transformation with dbt Cloud	63
5	Conclusion	65
5.1	Results	65
5.2	Future Work	65
5.3	Discussion and Conclusion	68
	Bibliography	69

1 Introduction

1.1 Motivation

One of the most important aspects of our era is medical and healthcare, which not only impacts the entire global population nowadays but also plays a significant role in the development of humanity in the future. Therefore, improving healthcare systems and decisions is growing to become one of the greatest problems. Today, the medical industry is one the biggest procedural of data. For example, an EHR system at a large hospital can create millions of medicals invoices, healthcare encounters, and clinical treatments. Although organizations are good at collecting data, the implementation of organizing and learning from it is still very limited, especially for traditional healthcare and life sciences institutions. When the complexity, volume, variety, and different data quality of healthcare continue to grow exponentially, it raises the first problem: complicated challenges in building the data ecosystem around it. For example, performing statistical analysis and research on medical data requires complex ETL and management of the raw data to be modeled and transformed into a data application optimized for such studies. Each step of this process currently requires using and discovering different new technologies, and each technology adds a new level of complexity to the process, which reduces efficiency and limits collaboration. Therefore, setting up proper data workflows and an effective data stack are essential tasks if we want to gain information from original data. The second problem is that healthcare and medical data collected from EMR usually exist in different formats with complex forms and structures, making it hard to link them together. In addition, they are often stored in transactional databases that are built to support business and clinical practices and are not designed to support data science and analysis effectively.

This thesis proposes a data ecosystem that includes a data warehouse, data lake, and data applications built around Modern Data Stack and Google Cloud Platform, which optimize and reduce the complexity of the data analysis, storage, transformation, and management process. It also conducts research to compare modern data applications available in the market and choose the best combination to reduce the time thinking about selecting components of the data system and try to focus on what is value-driven. Finally, the data pipeline transforms the synthetic patient data into OMOP CDM, which addresses the second problem. CDM allows for the systematic analysis of disparate observational databases [8]. By transforming the data into a standard medical model or format, the system can allow the user to collaborate on research and generate insights from the data despite coming from different sources. Overall, through this system, healthcare organizations can take advantage of modern advanced technology to make use of information efficiently and greatly improve data platforms and data-driven culture.

1.2 Thesis Structure

The thesis is organized into six chapters:

- **Chapter 1**, the one you are reading, which gives you the introduction to our thesis, including the motivations behind it and the structure of the document.
- **Chapter 2** introduces you to some definitions of how the data is processed and organized in our world today. It also includes a brief introduction to cloud computing and modern data stack.
- **Chapter 3** presents what our project's proposed software and solutions architect looks like. It contains the research on our proposal data model and compares applications and services to implement them in our thesis. Finally, it describes how data security and privacy are guaranteed in our project since healthcare data is one the most sensitive data nowadays with many data regulations around it.
- **Chapter 4** describes how we implement our ideas and integrate our software solution's components, including prerequisite knowledge that you need before implementing our solution, ingesting and storing collected data from different sources, transforming data using large-scale processing systems that can handle massive amounts of data, loading the data from the data lake to the data warehouse, transforming the data in the data warehouse into OMOP CDM, integrating components, and finally presenting insights and results to the viewers.
- **Chapter 5** will summarize all the core results and present a few notes on how the data solution of this topic can go further in the future.

2 Literature Review

As the amount of data generated increases every day, specialized storage systems for processing large amounts of data were developed [9]. In the last decade, The most notable principles for processing and storing large amounts of data are Data Warehouse (DW) and Data Lake (DL). More and more companies have implemented these principles as the amount of data they work with is exponential growth. The data warehouse is suitable for structured data analytics stored in a relational database. On the other hand, data lake solutions support semi and unstructured big data. Besides, many other data solutions come up nowadays to solve specific problems, such as Data Mesh and Data Fabric. In this chapter, we will go through some definitions and characteristics as well as a historical perspective of data warehouse and data lake (Section 2.1). Moreover, we introduce essential information about how the data flow within the data pipeline evolves with ETL and ELT (Section 2.2). Finally, the two most remarkable shifts in the data field recently are the emergence of the cloud-computing and the modern data stack. This chapter will also introduce more details about cloud computing including its evolution, benefits compared with traditional on-premise, and the rise of the modern data stack.

2.1 Data Warehouse

2.1.1 Introduction

One of the original purposes of the data warehouse is to revolutionize the way businesses perform analysis and make strategic decisions [10]. Therefore, it usually comes along with the term Business Intelligence (BI), which is not a new concept. One of the earliest notable statements on this term belongs to H.P.Luhn. He noticed the need for a business intelligence system that could support decision-making in business as early as 1958 [11]. But the technology at this time, according to Luhn, was not enough to create the system presented in his paper [11]. Thirty-year later, Devlin and Murphy, who also came from IBM, demonstrated a follow-up with an idea of a system that can query and request the information when needed. They developed the term Business Data Warehouse as *"The BDW is the single logical storehouse of all the information used to report on the business."* [12]. Today, despite going through many evolutions, there are still many different opinions around data warehouse. Two notables school of concepts around DW are Inmon and Kimball. According to Inmon, who invented the data warehouse term in 1990 [13], a data warehouse is a subject-oriented, consistent, integrated, time-varying, non-volatile collection of data to support the management's decision-making process. On the other hand, Kimball's definition of a data warehouse is a large database that collects data from a variety of sources or copies

of transaction data specifically structured for access, query, and analysis [14]. Overall, a data warehouse is a system with a collection of techniques, tools, and methods that retrieves and consolidates data from various sources into a single consistent database. It can handle complex and increasing amounts of information to support business intelligence, analytic activities and improve decision-making processes [15].

2.1.2 Characteristics

Some common characteristics of data warehouse nowadays are:

- Store in relational databases.
- Online analytical processing (OLAP) instead of online transaction processing (OLTP).
- Processing using ETL (Extract-Transform-Load).
- Contain data mining and multidimensional capabilities to analyze data from different methods and perspectives.
- Scale-able. Can be combined with various applications that are responsible for gathering and delivering a vast amount of data [16].
- Star-schema or snow-flake schema.

Data warehouse nowadays is stored in relational databases. According to Edgar Codd (1970): "A relational database is a digital database based on relational model of data." [17]. This model organizes data in one or more tables (or "*relations*") of columns and rows, making it easier to understand and analyze.

One of the most crucial characteristics of DW is using OLAP instead of OLTP. We include some detailed comparisons between the mechanisms of the OLAP and OLTP in table 2.1.

OLAP	OLTP
Optimized for complex analytics in ad-hoc reports	Optimized for operation transactions in daily applications
Suitable to read, write and aggregate	Suitable for insert, update, and delete
Denormalization	Normalization
Column-oriented	Row-oriented

Table 2.1: Denormalization Table

The first crucial mechanism of OLAP is Denormalization versus Normalization. Denormalization in OLAP is trying to increase performance by reducing the number of joins between tables (as joins can be slow). It will require less effort to do the analysis but will not ensure the data integrity as we can see there are duplicates in table 2.2 from the patient allergies example.

ID	Encounter	Patient Name	Year	Allergies
1	Encounter 1	A	2020	Bee venom, grass pollen, tree pollen
2	Encounter 2	A	2021	bee venom, fish, nut

Table 2.2: Denormalization Table

On the other hand, Normalization in OLDP is about trying to increase data integrity by reducing the number of copies, eliminating redundant data, and ensuring dependencies [18]. With Normalization, we will have less inconsistent and duplicated data but more tables, as can be seen in tables 2.3, 2.4, and 2.5.

Allergies ID	Encounter ID	Allergies
1	Encounter 1	Bee venom
2	Encounter 1	Grass pollen
3	Encounter 1	Tree pollen
4	Encounter 2	Bee venom
5	Encounter 2	Fish
6	Encounter 2	Nut

Table 2.3: Normalization Allergies Table

Encounter ID	Encounter Name	Patient ID	Year
1	Encounter 1	1	2020
2	Encounter 2	1	2021

Table 2.4: Normalization Encounter Table

Patient ID	Patient Name
1	A

Table 2.5: Normalization Patient Table

The second mechanism of OLAP is column-oriented versus row-oriented. Column-oriented databases usually have to deal with fewer transactions and a higher volume of data, while row-oriented databases are able to handle a large number of transactions but a low volume of data. The key difference between both data stores is how they are physically stored on disk: *“Row-oriented databases store the whole row in the same block while columnar databases store columns in subsequent blocks.”* [19]. Figure 2.1 shows how records are stored in a row-oriented database provided by Amazon Redshift, where each row is stored in a sequential disk block.

Block size larger or smaller than the size of records will also lead to inefficient use of disk

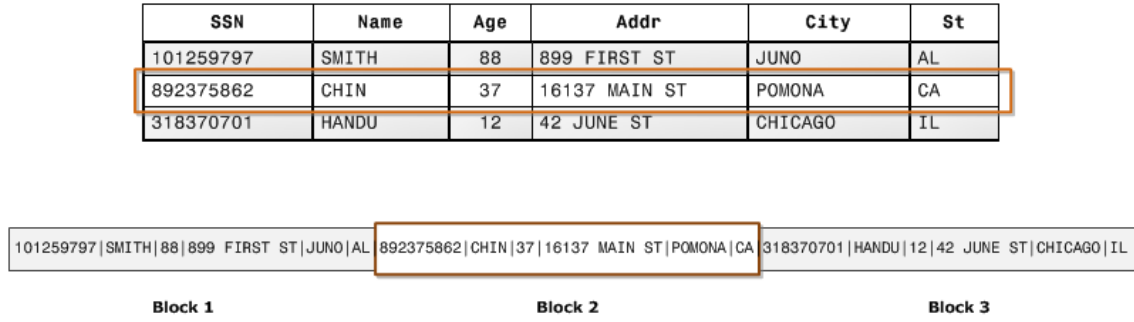


Figure 2.1: Row-wise storage [1]

space. Figure 2.2 shows how with columnar database storage, the values for each column are stored in single block.

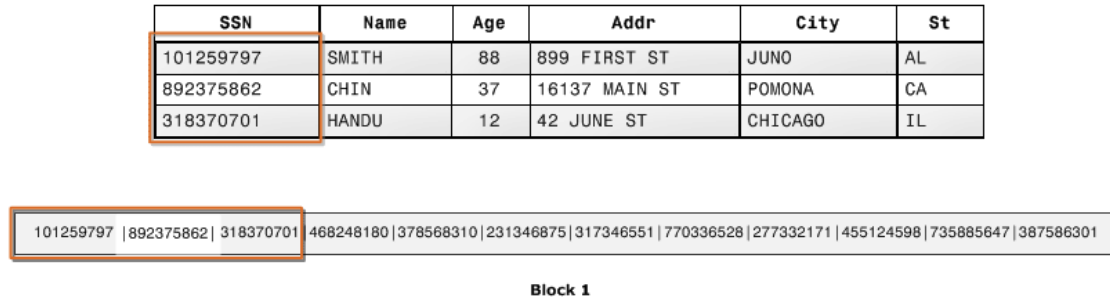


Figure 2.2: Column-wise storage [1]

These approaches work well in different situations. For example, in the hospital, most invoices transactions are required inserting all values for the entire records, so the row-oriented approach is more suitable for OLTP databases. On the other hand, if we want to do some analytic queries, for example, "What is the average age of female patients?". If we have a row-wise database, we have to read all the data in each row, which mean we will have to go through all the database. On the other hand, we only need to read columns gender and age with a column-wise database. An additional example of how significant the differences in performance column stores are provided by John Schulz. When he worked with a commercial database containing twenty million records using typical MySQL, a row-oriented database, it took 30-40 minutes, while using MariaDB AX (a column store variant of MySQL) took between 0.1 seconds and eight seconds [20].

Data warehouses are usually provided after a three-phase process: Extract, transform, load, which will be presented in more detail in section 2.3. This process creates three core components of the data warehouse structure: source system, staging area, and data warehouse. There are two different main methodologies for choosing a data warehouse process architecture. There is the top-down approach of modeling in third normal form with snowflake schema by Bill Inmon [21], and the bottom-up approach of dimensional modeling with star schema by Ralph Kimball [21], known as the data mart bus architecture. Although

we do not go into detail about each methodology, we will give a comparison between the star schema and snowflake schema. Star schema is produced by dimensional modeling and involves facts and dimensions tables to describe the data [14], [22], [23].

The star schema is named after its appearance in logical data model: when drawn with the fact table in the center, it resembles a star or an asterisk. Fact tables consist of the measurements, numerical values, metrics, or facts of a business process, while the dimensions are descriptive and used for grouping and labeling. The snowflake schema, on the other hand, is a variation of the star schema. When we apply normalization to a dimensional table, the result is called a snowflake schema. The Star schema is in a more de-normalized form, uses fewer constraints, so the analytical query execution time is more optimized. In almost all cases, especially in modern cloud data warehouse, the queries speed of a Star schema is better than snowflake [24]. Overall, The star schema has more benefits when compared to the Snowflake. It has a simple query structure, making it easier to read and follow, while Snowflake requires a more complex query structure and is harder to understand. Besides that, because the snowflake schema has a complicated structure, it requires more joins and tends to have less data redundancy. The only thing that makes the Snowflake come before the star schema is the ability to maintain. The fewer redundancies, the fewer places the maintenance needs to take place [25].

Finally, we move on to some benefits of the data warehouse. According to [26], the data warehouse makes it easier to understand and query, especially with star schema and a simple data model. Because it removes duplicate values or simplified complicated column names, it helps the data team to use it faster without the effort to clean and transform data. Since the data warehouse ingests data from all sources into a single place, everyone generates insights from the same data, with no more retrieving data from different sources or varying answers to the same question. Because the data warehouse is separated from the transactions data system, it reduces query stress on the system [27]. To summarize, a well-functioning data warehouse can turn out to be a worthless solution for organizations in all industries.

2.2 Data Lake

2.2.1 Introduction

Data warehousing has been a mature field over the years and has been in the market for a long time. Although it is still the best way to go for organizations, according to Fabio Braga [28], many factors drove the evolution of the data warehouse as it is no longer enough:

- Larger diversity of unstructured data types (text, XML, JSON, logs, sensor data, images, voices), which do not fit into the tabular format of a relational database
- Unprecedented amount of data (social platform, IoT, machine learning). Cost much time to build a data pipeline transforming and loading data into the DW in the correct form.

- The rise of Big Data technologies like HDFS and Spark makes it possible to process faster and store at a much lower cost than normal databases.
- Sometimes data model is too complex for data exploration activities needed by new roles like a data scientist. Using SQL to perform analytics is also not for everyone and will require additional time for training.

Data lake is a relatively new concept created during the exploration of the volume of the data to address these problems. It came along with an internal project from Google on a distributed file system and computed framework called Hadoop to address these problems [29]. The system was able to store any type of data in a distributed file system and use distributed computing framework to execute jobs using compute cluster. The ability to make analyses on top of any type of raw data without predefined schema opens many doors for data science and machine learning engineers. Several years later, it became very popular since it was the first successful technology to solve Big Data problems. However, Hadoop also has its disadvantages. Hadoop did not integrate well with the BI space, and MapReduce jobs were not exactly a good option to work with as they required a very specialized set of skills. The lack of consistent data governance practices made many data lakes unproductive. At this time, Hadoop was no longer the first option when facing Big Data problems. Then a group of Ph.D. students from the University of California saw these drawbacks with Hadoop, and they built a new open-source compute engine that addresses these problems [30]. The engine is Spark, which then becomes part of the Apache Foundation open-source project. It has a much more excellent set of libraries to work with, support for different languages (Scala, Python, SQL), and it is a lot faster than Hadoop due to its in-memory processing capabilities. It has machine learning libraries built-in and the support for structured data streaming scenarios. Nowadays, data lakes are usually built with a distributed data solution by using Spark as a data access layer on top of optimized file formats (Delta Lake, which will be mentioned in 3.13 and Parquet file formats).

Overall, a Data Lake is a clean, organized, single representation for all of our data to exist together in a unified source. Sometimes it works like a regular data source but has a virtual structure and schema on top of it. Having clean, unified places for all sources with a simple process enables us to write simpler queries and work faster [26], especially when we need to work with data from different applications. You will want to create a single home for this data so you can access all of it together and with a single SQL syntax, rather than many different APIs [26]. In the cloud, Data Lakes are storage repositories of multiple sources of raw data in a single location. These are typically stored in cloud c-store data warehouses or in AWS S3 buckets [26]. The data can be in a variety of formats and can be structured, semi-structured, unstructured, or even binary. The applications that we use for business may likely only offer transactional API access to the data. They're not designed for reporting, so unless the data is exported and put into a format you can easily query, you will end up being very limited in what you can pull. If used directly for reporting, these APIs can also become prohibitively expensive. Source data might be from the actual production database, which could affect the application's performance that it is powering. Queries that demand a lot of data, such as aggregations, are not optimally run on transactional databases. Data Lakes are

built to handle faster these types of ad-hoc analytical queries independently of the production environment, while data warehouse requires a lot more effort to build.

2.2.2 Characteristic

Although data lake shares some characteristics with data warehouse, like a single source of truth and optimized for analytics, some common differences between them are:

- Support unstructured data and performing advanced data science and machine learning.
- Using the same hardware for storage and processing, no need for additional storage for a staging area.
- Unifying all data from multiple sources but keeping the data in its original forms.
- cheaper cost for storing low or unproven value data which are not used for analytics.
- Fully query access after loading data into Data Lake. You will have all the power and flexibility of SQL, programming languages, or BI applications versus when they were in original sources and formats.
- Data warehouse uses serial data processing, while the Data lake uses parallel data processing [9].
- Schema on reading when compared to the schema on write of data warehouse. You create the schema only when reading the data. For example, one can load a CSV file and make a query without creating a table or inserting data on the table.
- Simpler data model, although we do not remove the data modeling techniques and ETL/ELT processes to ingest data into data lake. [28]

Some advantages of the data lake can also become important shortcomings for them. Unstructured complex data and simple data modeling can confuse users and makes it harder to understand and maintain. Using Spark to perform small tasks sometimes results in much higher costs and wastes computing power. Overall, the difference between a data lake and a data warehouse is that a data lake requires lower effort to build and can support data analysis for those with the skills to leverage it. A data warehouse takes more time to build and provides less access to raw data, but it is generally the safer place for sensitive data and more productive for the analysis. It can also support people with different skill sets to navigate data outside of data scientists and engineers like data or business analysts. Finally, there is a trend that organizations start to combine data lake and data warehouse to use all of their potential ability in the industry. Data Lake is where you put data that you have yet to prove the value and is a great source of data for a data warehouse. You will be able to model the data on top of the data lake with less effort because the data is cleaner and has fewer errors compared to the original data [26]. Data Warehouse then can provide a cleaner and more structured view of the data. This combination goes along very well with the rise of ETLT, which will be mentioned in the next section (Section 2.3).

2.3 ETL and ELT

2.3.1 Traditional ETL approach

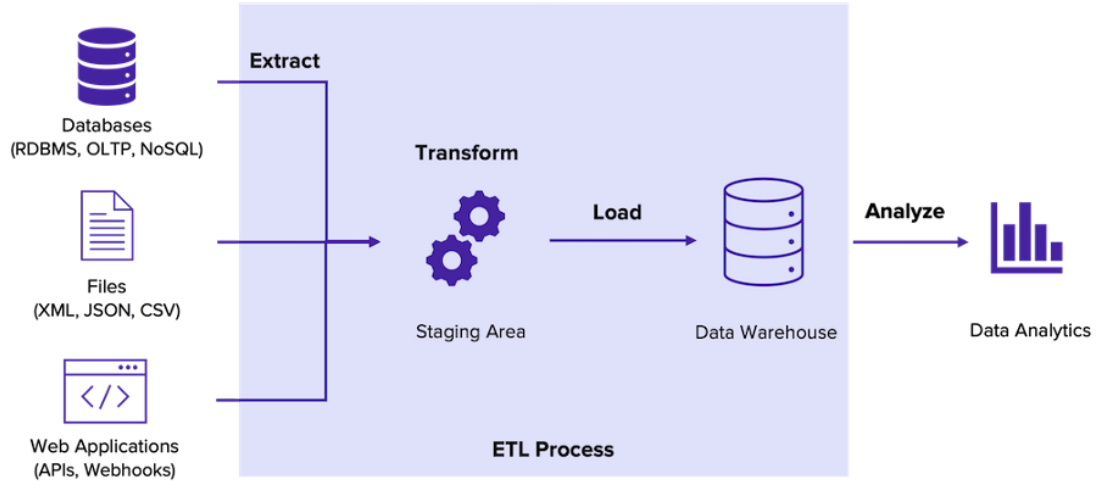


Figure 2.3: ETL Process[2]

Data lake or data warehouse are often the product of Extract-Transform-Load or ETL process [31](Figure 2.3):

- Data extraction involves pulling data from different sources, for example, OLTP databases or APIs, which are not suited for data processing. There are usually two types of extraction: Complete and Incremental extract. Complete mean extracts all the data from data sources, while incremental will only extract the updated data.
- Data transformation processes data by data cleaning and transforming it into a proper storage format/structure for the purposes of querying and analysis [32]. The data cleaning and transforming often include transforming columns or rows to make the data more straightforward to use. Transforming columns includes selecting the column, simplifying names, dropping columns, joining by grouping and aggregating, and clarifying columns by naming convention. Transforming rows include removing null and duplicated data, filtering invalid or incomplete rows, ordering and replacing values.
- Finally, data loading describes the insertion of data into dimensional modeling in the data lake, or a data warehouse [32].

Advantages and Disadvantages of ETL:

- **Advantages:** Reducing the cost for storing a smaller amount of data by filtering only data that you need.
- **Advantages:** Keeping in line with data privacy regulations such as GDPR. They often require removing, masking, or encrypting sensitive data like PHI and PII. We can remove these data before loading it into the data warehouse at the transform step.

- **Disadvantages:** Low flexibility when changing transformation or waiting for it to load into data warehouse [2].
- **Disadvantages:** Additional cost for staging area.
- **Disadvantages:** Can not re-use removed or original data.

2.3.2 The Rise of ELT

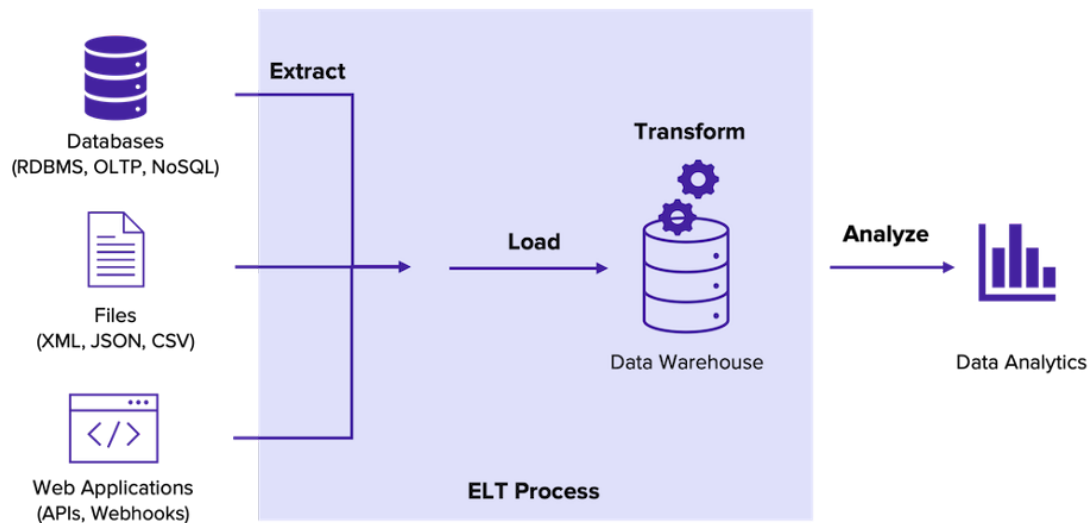


Figure 2.4: ELT Process[2]

ELT, similar to ETL, is Extract-Load-Transform (Figure 2.4). We load the data into the data lake/warehouse and then apply the transformation later. In the past, storing and transforming data in the data warehouse was expensive. Today, with the reducing cost of cloud data warehouse (cheaper storing, processing capability to efficiently manage operations on large data sets) and the growth of Spark (storing and processing data in the same place), some organizations are shifting from ETL to ELT process for performing unstructured data and performing fast, and large-scale data transformations [2].

Advantages and Disadvantages of ELT:

- **Advantage:** Deliver data fast. Having all the original data right in the warehouse is convenient for changing transformation which applied to data from existing sources.
- **Advantage:** More scenarios to integrate: Saving time when the transformation is not well defined at first or when we aren't exactly sure how we want to use the data. Saving the data that can be used later.
- **Advantage:** Reducing cost by removing staging area. Transformation in the data warehouse can be cheaper compared to processing with Spark in ETL.

- **Advantage:** More flexibility for transformation as it is the last step in the process.
- **Disadvantage:** Since we load the data directly to the data warehouse. We may have to take care of data security regulations such as GDPR and HIPAA.

One recent Advantage now is that some people with only SQL skills can do the transformation in the data warehouse with modern tools. One of the most notable examples will be mentioned in 3.10. It also reduces the effort and time to find people that have SQL knowledge compared to the skills needed for the transformation job with ELT.

2.4 Cloud Computing

2.4.1 Introduction

There are different definitions around cloud computing depending on the view of users. From the technical view, according to The National Institute of Standards and Technology: *"Cloud computing is a model for enabling ubiquitous, convenient, on demand network access to a shared pool of configurable computing resources[33]"*. On the other hand, from the business view, they see cloud computing as *"An Internet based service provision provided on a fee bases. where resources can be dynamically rearranged. This promises increased cost efficiency (as a result of a higher degree of resource utilization), as well as higher degrees of performance, stability, scalability, and flexibility [34]"*. Because it has a lot of advantages compared to the traditional on-premises which will be mentioned in Section section 2.4.2, Michael Armbrust predicted that cloud computing would continue growing and become more and more important in the future [35] since 2010. It becomes extremely true nowadays since 3 of the big five [36] are heavily shifted forward and focus on cloud computing: Amazon with Amazon Web Services, Microsoft with Microsoft Azure, and Google with Google Cloud Platform. Besides that, almost all the biggest companies in China also implement their own cloud platform [37].

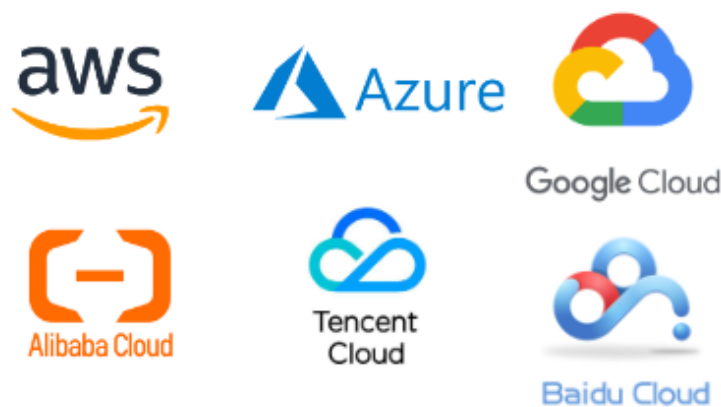


Figure 2.5: Top Cloud Providers

2.4.2 Benefits

In the past, data were relatively small and predictable. They are usually stored and processed data in the enterprise's data center called on-premise. Despite having some advantages in security, a server on-premises has a lot of drawbacks compared to advanced cloud technology. The comparisons between them are summarized in table 2.6 and table 2.7.

Data applications in the cloud inherit all the advantages the cloud has versus on-premises compared to traditional data applications. According to Krishnan [39], implementing data applications in the cloud is a valid option. It removes the effort to manage infrastructure and makes us accessible to the most advanced services that are provided by the best companies in the world. It can scale up and down easily and integrate well with other services to extend the system data science, and big data requires a lot of expensive tools and computational power. It also removes common disadvantages of implementing transitional technologies: a long process of implementing, fear of the unknown, and the need for educating people, which can be minimized by the ease of use and detailed documentation of cloud services [9].

Finally, You don't need to put all the eggs in one basket. You can reduce the reliance on a single vendor by using multi-cloud. You can take advantage of the best services in each cloud provider and combine them with others. It also satisfies regulations for local storage with more data centers and replicates data to be physically present within different places in case of disaster. However, this approach has some negatives. Some providers are trying to lock in consumers, so their services may not be easy to combine with services from others. It is also harder to maintain security and governance as well as the budget for the system.

	On-Premises	Cloud Computing
Time	Need time to deploy	Rapidly provision resources. Almost immediate access
Server	Need to buy equipment: racks of server, adapters, cables, etc.	Rented server, the rent is always more flexible and cheap than buying
Space	Need a location or a room to store equipment	Don't require space for store equipment
Portability	Inconvenient if we have to move our offices. We also have to transport equipment while ensuring that the service is running	Provide efficiently with global access services at the time we need and transfer between locations.
Additional cost	Electrical, maintenance cost, space cost (Rent a room), administration and accounting cost, salaries for information technology personnel, Software and Hardware licensing,... Hard to predict and calculate	No additional cost, simple and automatically calculated budgeting and planning
Continuously	Tasks don't happen continuously: Setting the server up when needed will cost a lot of time. Besides that, If we adjust the processing power high for peak moments, it can be wasted when we don't use all of it at quieter times. Adjusting them again can also cost efforts	We can easily manage cloud services with less cost and effort
Safety	Need to prevent data loss due to equipment failure, power outages, theft, or disaster	No need to worry about physical threats
Security	Need to pay a huge cost for security and potential security breaches as well as careful attention to various details such as security protocols, data encryption, firewall protection, monitoring, and adapting to emerging security threats. A poorly implemented security and protection system can easily lead to lost business	The security of services is guaranteed by the best professionals in the field

Table 2.6: On-Premises and Cloud Computing Comparison

	On-Premises	Cloud Computing
Sensitive Data	If your company manipulates sensitive or confidential data, there is a risk associated with someone else hosting it, and government surveillance	Big Cloud providers have a lot of agreements about regulations which will be mentioned in section 3.14
Latency	High latency if moving away from data centers	Cloud providers own a lot of data centers around the world. The closer the server, the less latency we will experience when using our application.
System	Have to deploy and maintain the system	The services and machines are optimized and managed by the best professionals in the field
Integration	Hard to add a new application to the system	Integrate with other cloud services. Opens possibilities for adding new applications.
Scalability	Requires time and effort to scale both system and hardware	Provides virtually unlimited storage/compute. The cloud does not only enable us to easily scale up but also scale down to fit all sizes of dataset [38]. Scalability is also guaranteed and managed by the provider
Accessibility	Requires additional training for unique applications and systems	Can lower IT barriers to innovation. Reduce the training step as more people are familiar with the services.
Upgrading	Requires manual upgrading which may interrupt the system	The updates are covered by cloud providers

Table 2.7: On-Premises and Cloud Computing Comparision

2.5 Modern Data Stack

The term “data stack” originates from “technology stack”, which is the combinations of different cloud-native technologies and services that are centered around a cloud data warehouse and data lake to combine together and build into a data platform [40]. In the beginning, the term was created for marketing purposes, but with the support of the communities, it has become more and more useful for saving time, money, and effort [41]. These tools usually include:

- Cloud-based data warehouse or data lake as the core components
- Data transformation
- Business intelligence or data visualization platform.
- Data Catalog, Governance and Discovery
- Data orchestration or workflow management
- Data Quality Assurance

The benefits of a Modern Data Stack:

- Lowers the technical barrier to entry for implementing a data platform. These tools can be set up fast and integrated easily with each other without in-depth technical knowledge.
- Consists of technologies with a general standard. Therefore, We can add or swap parts of the stack as our needs to avoid vendor lock-in.
- Wide adoption and Open-source: More people familiar with them, making it easier to co-operate in less time and get more support and resources from the community.
- Hosted in the cloud for scaling without complicated technical on-premise configuration and cost.
- Support modern agile: automated testing, deployment and version control.
- Optimized for much larger data volumes.

For example, a modern data stack core - modern cloud warehouse technology like Snowflake or BigQuery, remove the time and effort for managing the optimization and administration of the database. We do not have to spend time creating indexes to speed up queries or worrying about constraints because the technologies behind the data warehouse handle these problems.

3 Solution Architecture

In this chapter, we present our proposed structure and workflow of the data. First, we provide the proposed models in the data warehouse and data lake for the previous problems. After that, we conduct research comparing different applications between cloud data services and data applications in the modern data stack to select the best elements for our solution. The selected applications must be the most suitable for our solutions as they have to be the best in their field and must be able to integrate with other components. Lastly, we describe how data security and privacy are guaranteed in our project since healthcare data is one the most sensitive data nowadays with many data regulations around it and how cloud computing can address this problem.

3.1 Proposed Architecture

We propose the data platform's architect for the thesis using presented applications in figure 3.1:

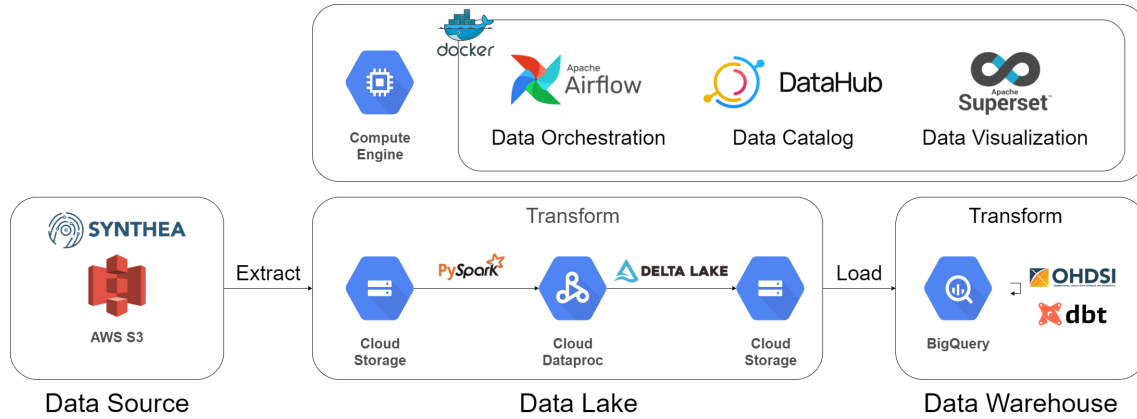


Figure 3.1: Proposed Architecture

First, We will host a machine that installed Apache Superset, DataHub, and Airflow above the docker. We set up Apache Superset to visualize the data, DataHub to present the data catalog and Airflow to control the ETLT processes. We trigger the Airflow to automatically run the ETLT process. The first step is making the data to be extracted from AWS S3. Amazon S3 launched 15 years ago, which is one of the oldest cloud storage services and has become very commonplace to store data in the cloud nowadays[42]. It contains the data provided by Synthea, which is the realistic patient data generated by algorithms but without any sensitive

information and raw vocabulary downloaded from the OMOP websites. This is the first step to moving our data into the GCP cloud platform. The data then now be stored in the raw data lake in GCP Cloud Storage. The purpose of this raw data lake is to represent a single source of truth and to store all kinds of data generated from different sources in raw format. It helps us to make the original data more easily without dealing with different data sources like databases or APIs. It is also usually a good idea to retain the raw data because we can always go back to our raw data lake and change our ETL process or easily add new extraction pipelines. Once the data is moved to the raw data lake bucket, We submit PySpark jobs to Dataproc to create the optimized data lake. This is what we are using for analytics. The data is read, prepared, compressed, and can be partitioned by certain columns to allow for fast query times in Delta Lake format. We are constructing a star schema with fact and dimension tables with bus matrix from Kimball concepts. The Optimized data lake will also be stored in Cloud Storage. After that, we will load the data into the raw data warehouse. We will not have to load or make a copy of our data in the optimized data lake again to store in Bigquery. With the advantages of belonging to the same cloud platform, Bigquery allows us to create an external data table, in which Bigquery will just store the metadata of the optimized data lake and can do the query against it. Airflow DAG runs the data quality check on all raw data warehouse tables once the job execution is completed. After that, we will transform the data into OMOP Common Data Model with the help of dbt. Finally, we create charts and dashboards using SQL queries in Superset, and integrate the data applications with data catalog in DataHub for tracking analytics, components, and data flow through the data system. If we do not want to manage DataHub, Apache Airflow and Apache Superset by ourselves, we can migrate to Acryl Data, Google Cloud Composer or Preset. They are the same products with similar features (unlike Metabase which they limit the features in open-sources version) but fully-hosted managed solutions provided by their founders and creators.

We choose Google Cloud Platform as our core cloud provider for our solution. (GCP) is a collection of cloud computing services offered by Google. Although it has the least market share in the top 3 cloud providers [43], and the fewest services [44], it provides niche advantages in big data and AI, especially when considering the data warehouse and data lake, which are the cores of this project.

Our system also uses the most suitable data applications based on our research and testing, as well as opinions from the community. We select almost the top popular OSS applications presented in table 3.1 according to Data Council [7]. The more popular the applications, the better the product and easier to implement, adopt and collaborate in the future.

3.2 Research on Hybridized ETLT approach

Either ETL or ELT has its own advantages and disadvantages. ETL is better for data quality, data security, and data regulation but scarifies flexibility and the ability to work with structured data. ELT is faster since the data is loaded directly into our data warehouse and transformed the data directly where it is stored. It also brings flexibility to work with

Rank	Project Name	Total Vote
1	dbt	75
2	Apache Airflow	68
3	Apache Superset	55
4	Dagster	54
5	Trino	49
6	Prefect	42
7	Great Expectations	40
8	Apache Spark	37

Table 3.1: Top OSS Applications Data Council Survey 2021 [7]

unstructured data and save data that you may use later. However, it sacrifices data quality, security, and regulation in many cases. Similar to the combination of data lake and data warehouse, with the advances of technologies nowadays as they start to easily integrate with each other, ETLT has become a more and more popular approach to data as it has all the advantages of both original processes [45]:

- **Extract** the raw data from different sources and databases into a raw data lake. You can have a single truth of data with its original format.
- **Transform** data fast and simply. The transformation at this step usually contains changing data formats, data cleansing, and masking/removing sensitive data for compliance purposes, which creates an optimized data lake. This data lake contains data that you want data scientists to perform ML, or you have yet to prove the value of it. Finally, it is a great source for a data warehouse.
- **Load** Load the optimized data into the data warehouse. You can load only the data you need, as stored in the data lake in the storage system is cheaper with the support of column partition.
- **Transform** data more completely within the data warehouse. ETLT allows using the data warehouse faster because this transformation stage only performs transforming on top of pre-light transformations. With the help of some modern tools, more people can do this transformation stage without the skill sets needed in the past [4].

ETLT has the following benefits:

- Satisfies data security and compliance requirements.
- Make use of the advantages of all approaches: data warehouse, data lake, ETL, ELT like offering flexibility in transformation: simply transformation and data model in the data lake, which acts as a source for more complex transformations within the data warehouse later.

- Suit different types of analyses: Data engineers and Data scientists can do their job at the first. transformation stage. Data Analyst and Business Analyst can perform their task in the second stage. They use suitable technologies in the right places while remaining in the same flow, so there is no confusion between them.

3.3 Propose Model for Data Lake

First, the data will be stored in a data lake on GCS. The idea is that we want to have a single source of truth for raw data in our system. In the beginning, we may not be sure how many different ways we are going to use this data, so a flexible schema is needed. Data is stored as object blobs and compressed files and partitioned in folders by some columns.

Because the data already have an original suitable schema, we want a data lake to store all kinds of data generated from different sources in raw format. After that, we will have a data warehouse which optimized for analysis later. So we do not make any changes to the original data schema, except drop some columns and compress them in the optimized data lake.

Although we keep the original schema, in the beginning, we may not be sure in how many different ways we are going to use this data, so a data dictionary is needed. We create a data dictionary for users to have an overview of the data (Table 3.3) as well as an enterprise bus matrix based on Kimball's suggestions [14] for users to know how the table can interact with each other using star schema with fact tables and dimension tables (Table 3.2).

	date	patient	provider	procedures	payer
Clinical Events					
encounters	X	X	X		
procedures	X	X	X	X	
medications	X	X	X		
observations	X	X	X	X	
conditions	X	X	X	X	
Billing/Revenue Events					
payer_transitions	X	X	X	X	X
Operational Events					
devices	X	X	X	X	
supplies	X	X	X	X	

Table 3.2: Bus Matrix for Synthea Data

Table	Column
allergies	START, STOP, PATIENT, ENCOUNTER, CODE, DESCRIPTION
careplans	Id, START, STOP, PATIENT, ENCOUNTER, CODE, DESCRIPTION, REASONCODE, REASONDESCRIPTION
conditions	START, STOP, PATIENT, ENCOUNTER, CODE, DESCRIPTION
devices	START, STOP, PATIENT, ENCOUNTER, CODE, DESCRIPTION, UDI
encounters	Id, START, STOP, PATIENT, ORGANIZATION, PROVIDER, PAYER, ENCOUNTERCLASS, CODE, DEDSCRIPTION, REASONCODE, TOTAL_CLAIM_COST, PAYER_COVERAGE, BASE_ENCOUNTER_COST, REASONDESCRIPTION
imaging_studies	Id, DATE, PATIENT, ENCOUNTER, BODYSITE_CODE, BODYSITE_DESCRIPTION, MODALITY_CODE, MODALITY_DESCRIPTION, SOP_CODE, SOP_DESCRIPTION
immunizations	DATE, PATIENT, ENCOUNTER, CODE, DESCRIPTION, BASE_COST
medications	START, STOP, PATIENT, PAYER, ENCOUNTER, CODE, DESCRIPTION, BASE_COST, PAYER_COVERAGE, DISPENSES, TOTALCOST, REASONCODE, REASONDESCRIPTION
observations	DATE, PATIENT, ENCOUNTER, CODE, DESCRIPTION, VALUE, UNITS, TYPE
organizations	Id, NAME, ADDRESS, CITY, STATE, ZIP, LAT, LON, PHONE, REVENUE, UTILIZATION
patients	Id, BIRTHDATE, DEATHDATE, SSN, DRIVERS, PASSPORT, PREFIX, FIRST, LAST, SUFFIX, MAIDEN, MARITAL, RACE, ETHNICITY, GENDER, BIRTHPLACE, ADDRESS, CITY, STATE, COUNTY, ZIP, LAT, LON, HEALTHCARE_EXPENSES, HEALTHCARE_COVERAGE
payer_transitions	PATIENT, START_YEAR, END_YEAR, PAYER, OWNERSHIP
payers	Id, NAME, ADDRESS, CITY, STATE, HEADQUARTERED, ZIP, PHONE, AMOUNT_COVERED, AMOUNT_UNCOVERED, REVENUE, COVERED_ENCOUNTERS, UNCOVERED_ENCOUNTERS, COVERED_MEDICATIONS, UNCOVERED_MEDICATIONS, COVERED_PROCEDURES, UNCOVERED_PROCEDURES, COVERED_IMMUNIZATIONS, UNCOVERED_IMMUNIZATIONS, UNIQUE_CUSTOMERS, QOLS_AVG, MEMBER_MONTHS
procedures	DATE, PATIENT, ENCOUNTER, CODE, DESCRIPTION, BASE_COST, REASONCODE, REASONDESCRIPTION
providers	Id, ORGANIZATION, NAME, GENDER, SPECIALITY, ADDRESS, CITY, STATE, ZIP, LAT, LON, UTILIZATION
supplies	DATE, PATIENT, ENCOUNTER, CODE, DESCRIPTION, QUANTITY

Table 3.3: Data dictionary for Synthea Data

3.4 Related Work for Data Warehouse

Because our data warehouse is built on top of BigQuery from GCP, one of the most modern and powerful data warehouse platforms nowadays, we can reduce the focus on how to optimize logical data models for analytics. We do not have to spend time creating indexes to speed up queries or worrying about constraints because the technologies behind the data warehouse handle these problems. It also means that we can fully use OMOP CDM or any other data model without worrying about the performance and speed of the queries, although the OMOP CDM is already optimized for data processing and computational analysis to accommodate data sources that vary in size, including databases with up to hundreds of millions of persons and billions of clinical observations [46].

In this thesis, we build the logical data model of our data warehouse completely based on the OMOP CDM (Figure 3.2). The OMOP CDM is built and optimized for a lot of typical observational research purposes as well as consistent design elements[46]. Because OMOP CDM is one of the most widely used and known healthcare common data model in the world today, building the data warehouse on top of it means that it will be easier for us to reuse the process or scale and cooperate with other departments around the world.

According to OHDSI, The CDM is optimized for typical observational research purposes of:

- Discovering, classifying patient populations based on certain healthcare characteristics, interventions, and outcomes.
- Predicting the occurrence of these outcomes in individual patients.
- Estimating the effect of interventions on the population.

The development of the CDM follows many quality design elements. We found these elements helpful for us when considering the data model:

- Design for analysis purpose: The CDM is optimized for OLAP rather than for the purpose of addressing the operational processes (OLTP).
- Data protection and privacy: All data relating to the identity and protection of patients, such as names, birthdays, etc., are limited. Data related to PHI and PII are not included in any tables. Exceptions are only possible when the research requires detailed information for specific studies.
- Storing source codes: Even though all codes are mapped to the Standardized Vocabularies, the model also has tables for storing the original information.
- Technology independence: The CDM can be implemented in any relational database, such as a traditional database like Oracle and SQL Server, or a cloud data warehouse like BigQuery and Redshift.

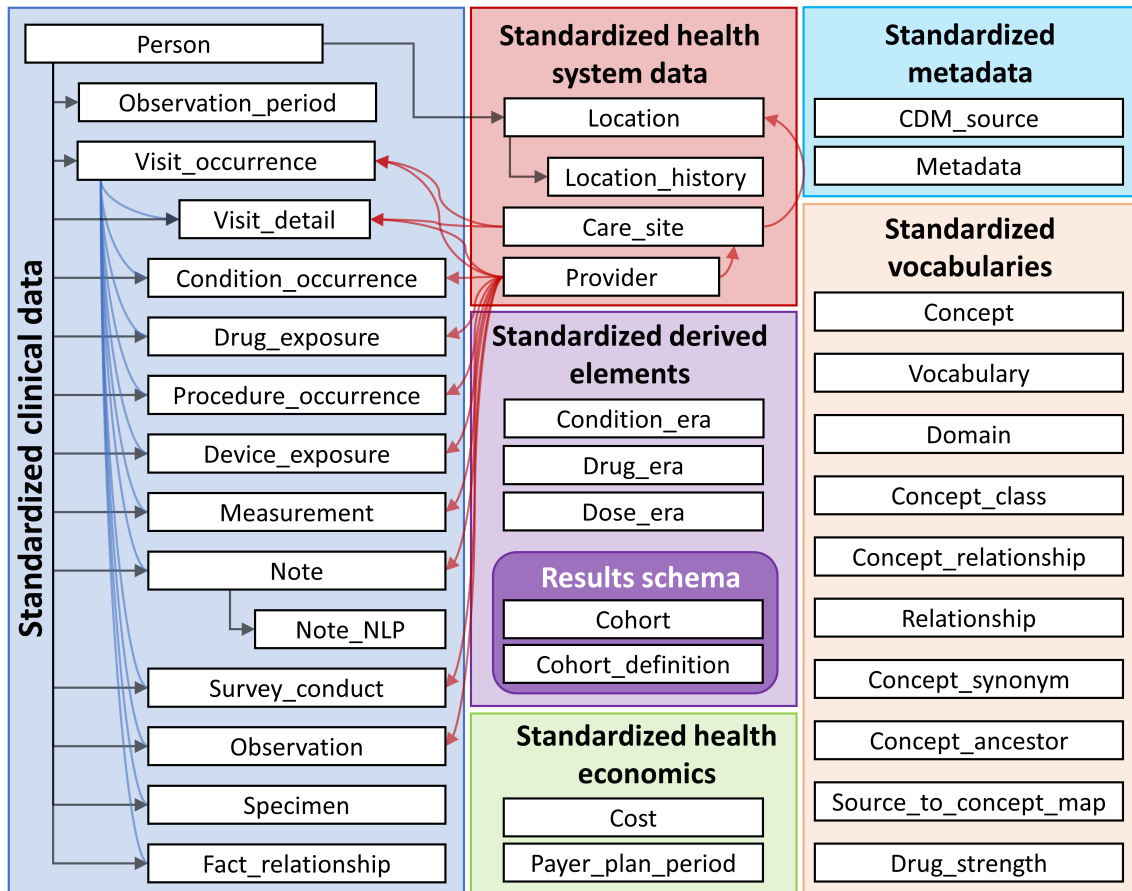


Figure 3.2: Overview of all tables in the CDM

- **Scalability:** The CDM is optimized for data processing and computational analysis to satisfy significant big data sources, including databases with up to hundreds of millions of persons and billions of clinical observations [46].
- **Version compatibility:** All changes between versions are clearly delineated in the Github repository. Switching between versions of the CDM can be easily implemented.

All of the other transformations are based on OMOP Common Data Model Extract-Transform Load Tutorial [47].

Since our data sources do not have enough information to fill in all tables, and for the convenience of data management, we implement only tables in CDM which can be filled with our data. The set of tables includes:

- **Clinical Data Tables:** Person, Condition_occurrence, Observation, Observation_period, Measurement, Procedure_occurrence, Visit_occurrence,
- **Standardized Derived Elements:** Drug_Era, Condition_Era

Person Table			
Field	Required	Source	Transform
person_id	YES		Using ROW_NUMBER() over patient id
gender_concept_id	YES	gender	When gender = 'M' 8507, 'F' then set to 8532
year_of_birth	YES	birthdate	Extract Year from birthdate
month_of_birth	NO	birthdate	Extract month from birthdate
day_of_birth	NO	birthdate	Extract day from birthday
birth_datetime	NO	birthdate	CAST birthdate as TIMESTAMP
race_concept_id	YES	race	'WHITE' as 8527, 'BLACK' as 8516, 'ASIAN' as 8515, otherwise 0
ethnicity_concept_id	YES	ethnicity	When race = 'HISPANIC' set as 38003563, otherwise set as 0
location_id	NO		null as PHI
provider_id	NO		set as 0
care_site_id	NO		null as PHI
gender_source_value	NO	gender	take original gender value
gender_source_concept_id	NO		set as 0
race_source_value	NO	race	take original race value
race_source_concept_id	NO		set as 0
ethnicity_source_value	NO	ethnicity	take original ethnicity race value
ethnicity_concept_id	YES		set as 0
ethnicity_source_concept_id	NO		set as 0

Table 3.4: Data dictionary for Transformation of Person table

- **Vocabulary Tables:** Concept, Vocabulary, Domain, Concept_class, Concept_Relationship, Relationship, Concept_Synonym, Concept_Anccestor, Source_To_Concept_Map, Drug_Strength

We make an example data dictionary table based on the OHDSI document, which describes the process when we transform the data in table 3.4. Details of the requirements for other tables can be discovered in OHDSI documentation.

3.5 Data Warehouses Comparison

Although Data warehouse has existed for a long time, it can not handle the exploration of quantity and complication of data nowadays. Cloud data warehouse, with the benefits of cloud computing, is rising as an efficient solution to address this problem, making use of the most modern technology and innovations provided by some biggest technology companies in the world. Because the data warehouse is one of the core components of this project, it also affects the decision to choose a cloud provider. Currently, when considering a cloud data warehouse, there are four popular services: Amazon Redshift, Microsoft Azure, Google BigQuery, and Snowflake (Figure 3.3).

The first data warehouse to consider is Amazon Redshift. It was officially published in 2012 by Amazon Web Services. Redshift has all the advantages of a modern cloud data warehouse, for example, OLAP structure, column-oriented database, built based on the massive parallel processing for large-scale databases. The detailed structure of Amazon Redshift will not be discussed here. The reasons Redshift was not chosen compared to our option (BigQuery):

- The cost of Redshift is higher according to [48] and [49]. The companies that moved from Redshift to BigQuery: Kabam, Blue Apron, New York Times, Yahoo, Reddit, and



Figure 3.3: Top Cloud Data Warehouses

Buzzfeed all say that they saved 30-70%.

- The speed of Redshift is slower than BigQuery [48] [49]. To achieve the same speed as BigQuery, you have to perform many tweaks, cluster management, and database config [50].
- BigQuery is serverless. You pay only for the storage and queries. Redshift runs in instances. Therefore, It costs every time you run the instances, even when you are not using them.
- It costs time and effort to manage the nodes and CPU. It is required more maintenance, and scaling is more difficult as well as managing downtime and updating.

The second option is Microsoft Azure. Microsoft Azure Synapse Analytics is a distributed enterprise cloud data warehouse of Microsoft responsible for handling big data analytics. Although Microsoft products are very popular nowadays and services in its ecosystem are working well with each other, we can not integrate them with other applications. For example, open-source tools that we use in this thesis are dbt, DataHub, Apache Superset and none of them support any services in the Azure analytic system.

Snowflake is a new rising cloud data warehouse on their servers for a monthly fee and, in exchange for handling all of the maintenance and backups. Today, Snowflake and BigQuery are considered the two best performance cloud data warehouses, but Snowflake has a shortage. It does not belong to any cloud ecosystem, making it harder for organizations to move all the systems into the cloud.

Lastly, we have BigQuery provided by Google Cloud. The heart of Big Query is a query engine built on Google's Dremel project, which is possible to query billions of rows of data in just a few seconds [51]. Secondly, It was server-less, meaning that you do not have to take

care of the infrastructure or worry about the fees charged when not using it. It also belongs to the GCP ecosystem and has many built-in supported AI and ML applications, which make it the best data warehouse option to be implemented in this project.

3.6 Cloud Compute Services Comparison

Compute Engine is the service provided by Google for creating Virtual Machines (VM) and SSH access. We use it to host our modern data stack applications, so it will be available all the time for the users to access. Compute Engine was chosen because it has the quickest performance (Figure 3.4. Google Cloud is more than four times faster than Microsoft Azure in completing creating VM and SSH access processes [3].

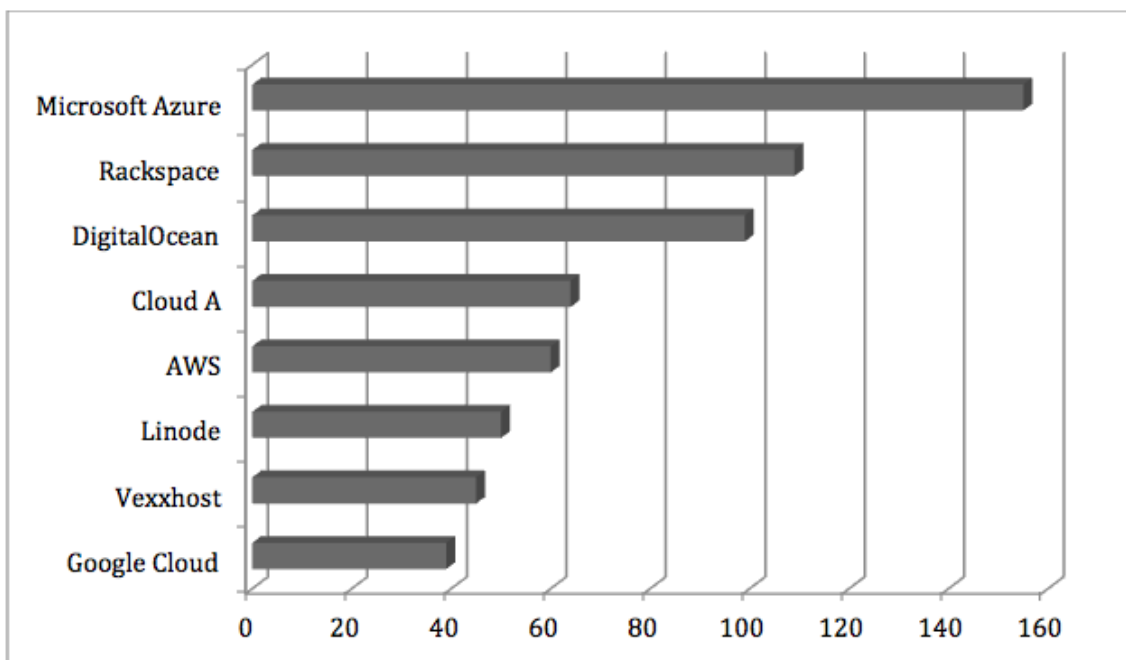


Figure 3.4: Cloud provider speed comparison for creating VM and SSH access [3]

3.7 Data Processing Services Comparison

Dataproc is a fully managed and highly scalable computational service for running Spark and other frameworks. Spark distributes data processing tasks between clusters of computers. Splitting up your data makes it easier to work with very large datasets because each node only works with a small amount of data. Spark has been found to run 100 times faster in memory and ten times faster on disk. It's also been used to sort 100 TB of data 3 times faster than Hadoop MapReduce on one-tenth of the machines [52]. Spark has particularly been found to be faster on machine learning applications, such as Naive Bayes and k-means.



Figure 3.5: Top Cloud Computational Services

We used DataProc to Submit PySpark jobs for transforming data. PySpark is the Python interface to spark. PySpark hosts a DataFrame abstraction, which means you can do operations very similar to pandas DataFrames [53]. PySpark and Spark take care of all the complex parallel computing operations. We prefer Dataproc over AWS EMR, a similar service in AWS because Google Cloud supports new table formats like s Delta Lake and Apache Iceberg in the data lake [54]. It is harder to implement them in AWS EMR as they are not natively supported. There are two services that have the same functions as Dataproc in Azure, which are HDInsight and Azure Databricks. Recently, Azure has been focusing on integrating Azure with Databricks, so HDinsights is not good as AWS EMR and Dataproc. On the other hand, Azure Data Bricks or the original Databricks is overkill for our task and can cost a lot for our budget.

3.8 Cloud Object Storage Comparison

Google Cloud Storage is a Cloud Storage service. Similar to other storage services like AWS S3 or Azure Blob Storage, it allows you to upload files of all types to the cloud. Because we have chosen all the services in the GCP, and there is not a huge difference in storage services between different cloud providers, we also chose Google Cloud storage for storing the data lake.

3.9 Data Orchestrators Comparison

Data Orchestrator tools help data move from one place to another at the correct time, with a specific interval. The idea comes from batch processing. Batch processing is a method of running high-volume, repetitive data jobs at a specific time or runs if a specific condition is met in a specific order without dependencies and user interaction. Batch processing is a less complex system and requires less maintenance than stream processing and also allows companies to process large volumes of data quickly [55]. The core component of Data Orchestrator is Directed Acyclic Graphs (DAG). It represents the set of tasks that make up the workflow, the order, and the dependencies between them. Directed, meaning there is an inherent flow representing the dependencies or order between the execution of components.

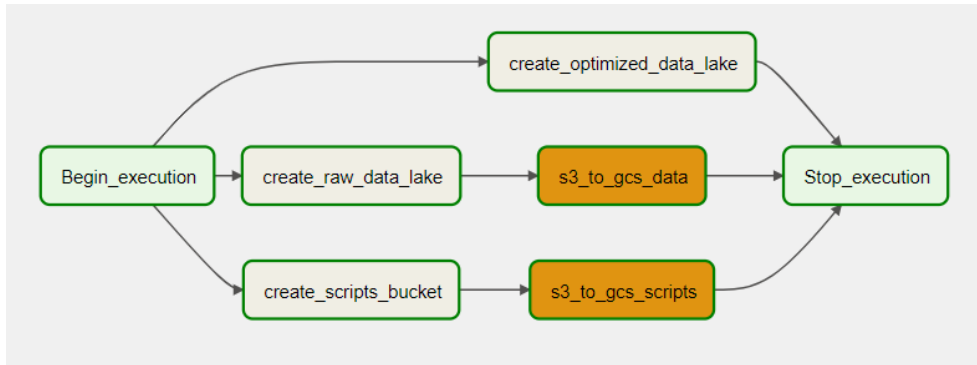


Figure 3.6: DAG Example

Acyclic means it does not contain a loop or repeat. The graph represents the components and the relationships (or dependencies) between them. DAGs are useful for providing a clear, associated order of steps for data processing.



Figure 3.7: Top Popular OSS Data Orchestrator tools

When considering a data orchestrator, there are four available options: Airflow, Luigi, Dagster, and Prefect 3.7. We removed legacy commercial tools like Microsoft SSIS or even modern cloud tools like GCP Data Flow or Azure Data Factory because, although they have some advantages as integrates well with some ecosystem (Microsoft for SISS, Data Factory for Azure, and Data Flow for Google Cloud) and has a visual (GUI) way to create workflows, these benefits also reduce the configurability and flexibility. We can customize more scenarios and integrate better with our platform with open-sources code-based workflows, especially when it comes to version control and deployment. Out of 4 options, Dagster and Prefect are newer compared to Luigi and Prefect, so they have the better UI but fewer sensors, operators, functions, and connectors. On the other hand, Luigi and Airflow have a bigger community for support and more features. Overall, Airflow is the best choice because it is the most mature and has been proven for a long time with dozens of available integrations to others. GCP even integrated Airflow as the workflow orchestration service called "Cloud Composer" to

interact with their platform while they already have their UI workflow management services.

3.10 Data Transformation Selection



Figure 3.8: Data build tool [4]

Data transformation and modeling help package different useful data transformation features into one application. When it comes to data transformation on the modern data stack, there's really only one tool that stands out: dbt (Data build tool). dbt is the most popular open-source transformation that is used by thousands of companies to help analysts and engineers transform data more effectively. It is compatible with many destinations, has built-in features that make it easy to manage data lineage, version control, and documentation which helps the data transformation process organized like a software implementing process directly in the data warehouse or data lake (Figure 3.8). It's a SQL-based data model builder written in Python. We can write SQL queries and use Python advantages like functions, variables, and reusable macros with DAG to manage the dependencies and order of tasks. We can also implement automated tests and auto-generated documents with UI based on the configuration (YAML) files used to describe your models.

3.11 Data Visualization Comparison

We can create dashboards and reports from DWs. Reports and dashboards are the ultimate results of a data system, which can be in the form of tables, charts, texts, or other visualizations and representations forms, depending on which a BI tool is connected to the DW and the purposes of the report. The main goal of them is to help users explore and find insights in their data, and usually provided by "data visualization" or "business intelligence" applications in the modern data stack.

According to [56], we can classify BI tools into three categories (Figure 3.9):

- **Commercial – "Legacy"**. Applications in this category are the oldest and most popular used around the world. They need to be installed in the system or signed as SaaS subscriptions. The key characteristic is that data must be ingested into the platform before analysis happens. Some tools are hard to integrate with other ecosystems or platforms and require learning their own SQL-based analytical languages and data modeling. Some notable examples are PowerBI, Tableau, Qlik, and Domo.

- **Commercial – “Modern”**. Unlike the legacy tools, tools in this category query the data directly in the place where you store your data, unlike the legacy tools. Some notable examples for this category are Looker, Chartio, and Mode.
- **Open source – “Modern”**. The applications are open sources and usually dockerized. Because we have access to the base codes, we can easily customize and integrate with other applications. We can have a host environment for them or choose the available cloud options provided by the authors. There are only three notable examples for this category: Superset, Redash, and Metabase.

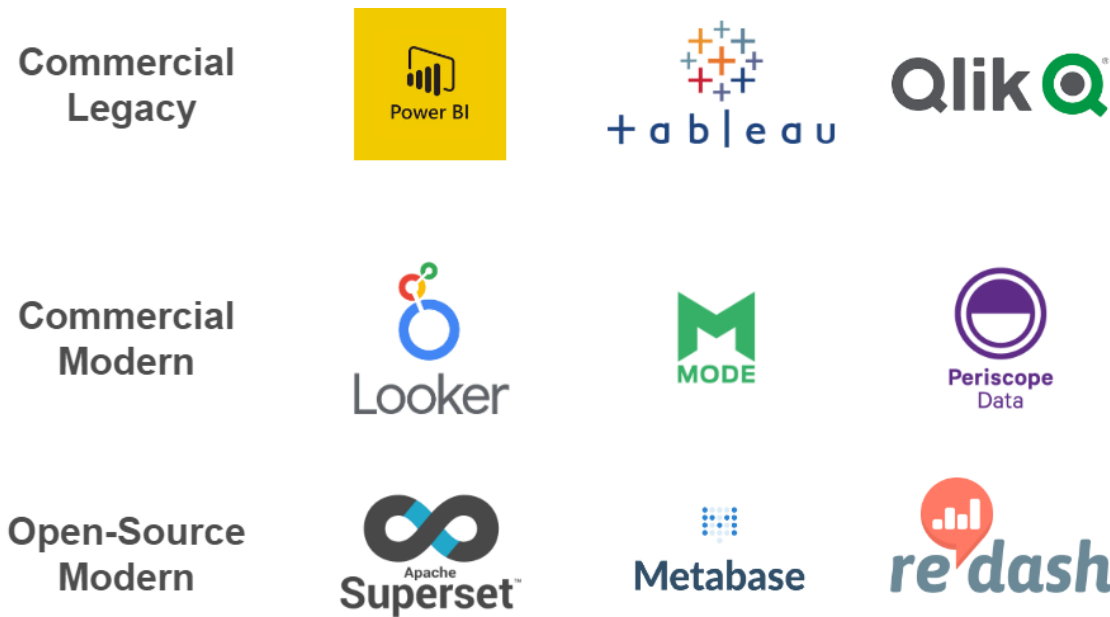


Figure 3.9: Data Visualizations and Business Intelligence Applications

The difference between "Commercial" and "Open-source" is the cost and customization. Commercial applications can become very expensive when scaling up, and the open sources are very easy to customize and integrate with almost the same functions and abilities. But the open-source type also has some shortcomings; the support we received from the community when facing problems can not be compared with the commercial type. Secondly, The difference between "Legacy" and "Modern" is that in the old world of on-premise data warehouse, the cost of running queries and experiments was expensive. So extracting the data to the BI tools can reduce the cost. But with the growth of cloud warehouses, running queries for visualization directly in a data warehouse have become more scalable and efficient, so application is shifted towards it. It is hard to tell which option is more beneficial, but in our case, we choose Open source – “Modern” category applications because they are easier to scale, customize, integrate, and suitable for the budget when scaling up. So we will cover the comparison between three Modern Open-source applications - Metabase, Redash, and Superset.

Firstly, we compare the popularity of the applications using Github stars as the community is contributing mainly to it in figure 3.10. More people familiar with it or the community is bigger will make the implementation and usage more approachable.

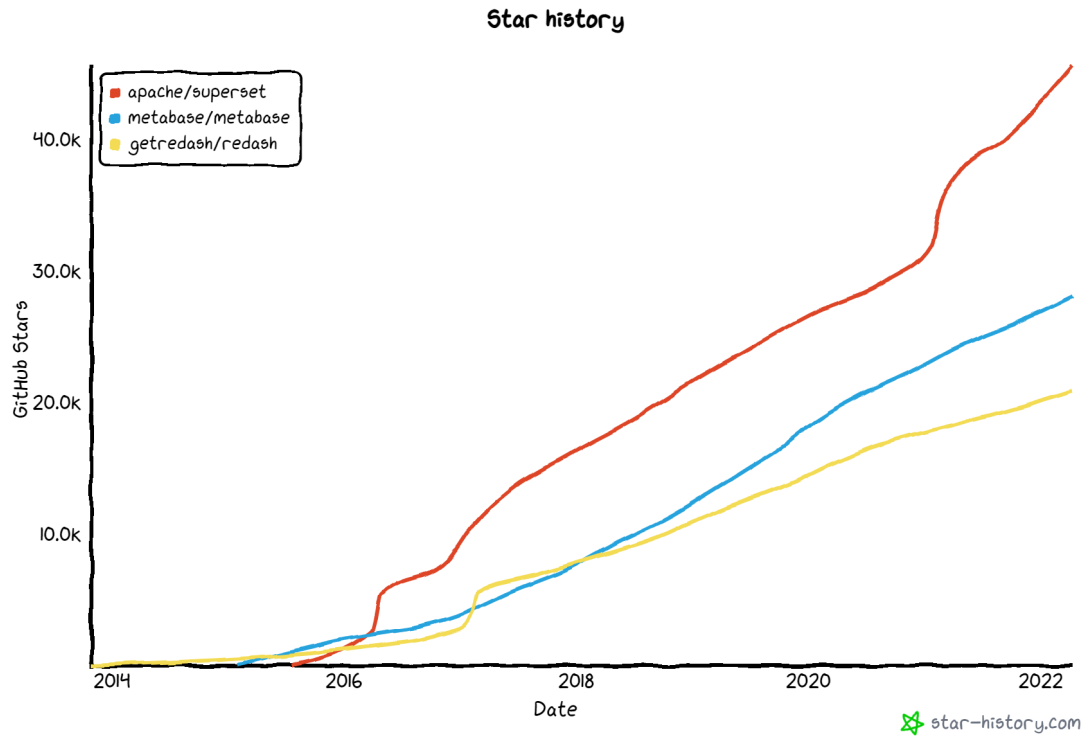


Figure 3.10: Compare Star History off Metabase, Redash and Superset

Although Apache Superset is the latest application to appear, it is rising faster than the other two and has become the application with the most GitHub stars nowadays. After conducting research and implementing all three applications, we propose several reasons behind this trend:

- It was founded by Max Beauchemin, who also is the father of Apache Airflow. Superset started as an internal project at Airbnb and later became an open-source project at Apache, which makes it get much more attention than the other two and gets more users and contributors. Besides that, people believe in Apache as they have standards and reputations for development and management. We can be sure that projects managed by Apache can live for several years more than with normal average open source applications.
- Metabase and Redash have their own problems. Metabase locks some additional features behind their cost version, like Row-Level Permission and Audit Logs, which are available in both Superset and Redash. Since Redash was acquired by Databricks

[57], they are focusing more on the integration with Databricks platform and hence, update less frequently.

- The UI and visualizations of Apache Superset are better for technical people since Metabase is better for business types and, Redash stays between them. People that access Github or want to implement the modern data stack are usually technical people. Business people tend to use their business organization's technology, which is usually PowerBI or Tableau.
- Superset also has more functions and features. For example, Superset supports more options in terms of authentication: Google OAuth, LDAP, OpenID, Database, while Metabase and Redash just support parts of them [58]. Superset has Sankey diagrams, Network visualization, geospatial visualizations, and 40 types of charts compared to 17 of Metabase [59].

<input type="checkbox"/>	Status	Name ↓	Zone	Recommendations	In use by	Internal IP	External IP	Connect
<input type="checkbox"/>	●	superset	asia-southeast1-b			10.148.0.26 (nic0)	None	SSH ▾
<input type="checkbox"/>	●	redash	asia-southeast1-b			10.148.0.24 (nic0)	None	SSH ▾
<input type="checkbox"/>	●	metabase	asia-southeast1-b			10.148.0.21 (nic0)	None	SSH ▾

Figure 3.11: Open-Source Modern BI Tools in Compute Engine

Redash and Superset are based on Python, which is similar to other tools in our thesis: dbt, Airflow, Datahub. The Python-based make them easier to integrate with other applications as well as customizations when needed. Overall, Apache Superset has more advantages than Metabase and Redash, which is the chosen product for our thesis.

3.12 Data Catalogs Comparison



Figure 3.12: Top Data Catalog - Data discovery, observability and governance applications

Data Catalog is a new concept and key to any data platform. It is an essential evolution the modern data stack needs to manage and control the growing complicated data platform. It contains metadata and data management software that use to inventory and organize the

data within the systems, which makes it easier for people to discover and understand what statistics, data, and applications exist in the data platform.

There are four suitable products for us in this category: Atlan, Marquez, Amundsen, and Datahub (Figure 3.12), which was created at LinkedIn. Since Atlan is a commercial product, it is hard to integrate with applications outside their business services. Marquez has the least features and connections. Although Amundsen and Datahub have the same community size as well as features and data sources, DataHub has a more intuitive UI as well as better documentation. However, the most important thing DataHub has over Amundsen is the "push" model. It allows you to push metadata change events from your database (i.e., when there is a new table created, it shows up in the catalog under a second), making it our pick in this category.

3.13 Data Storage Format Selection

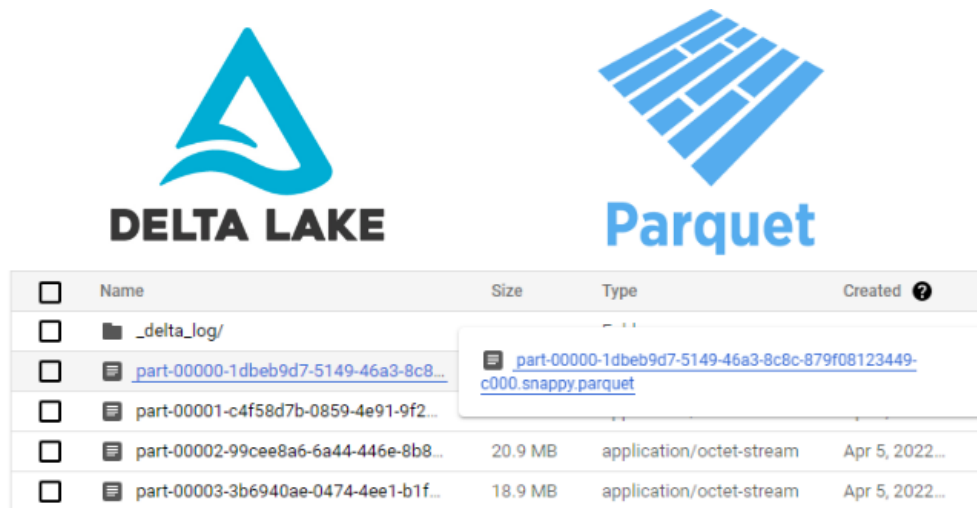


Figure 3.13: Delta Lake and Apache Parquet

One of the most popular file formats in recent years for Data Lake is Apache Parquet. Apache Parquet is an open-source file format that originates from Twitter [60] and Cloudera [61]. The main benefit of Parquet is Column-wise compression for saving storage space and reading and writing files. Delta Lake is built on top Parquet file format to enhance its features (Figure 3.13). It is, in fact, a series of parquet files with an additional log file on top which tracks changes made to the data. It enables some features like support for SQL and python syntax, ACID on top of parquet, time travel with transaction logs, and schema evolution. Vanilla Parquet requires that all files have the same schema. If you add a Parquet file with a schema that doesn't match all the existing files, they can become corrupted and unreadable. [62]. Overall, It has many additional features which can solve a lot of problems, and many people don't think about them at all unless they have encountered them before.

3.14 Data Privacy and Security

Moving data from an on-premises solution to a cloud solution can be a very problematic situation, especially for ensuring data privacy and security process.

The first problem when implementing cloud computing is the regulation of physical presence on a local or national level. Regulation might cover requirements for the location of the physical data center to stay within national or continental boundaries. However, cloud providers are working towards providing more and more data centers at certain geographical locations around the world to fulfilling these requirements. The data processing, data storage as well as the host's location for the virtual machine can happen in one location, country, or region in GCP. For example, in our thesis, all applications in the system are located in Southeast Asia - Singapore.

The second problem is the practice of protecting digital information from outside attacks, for example, unauthorized access or corruption. It covers every aspect of information security, from physical hardware and storage security to logical administrative and access controls of software applications [63]. When cloud computing is implemented, we will not have to worry about these problems because it will be covered by cloud providers with the best professional experts in the field. For example, all Cloud Storage data are automatically stored in an encrypted state [64]. All the services are also always guaranteed up-to-date to protect against new criminal activities.

The third problem is the guard against insider threats and human error, which remains among the leading causes of data breaches today. Cloud providers can also manage these processes for you. GCP Identity and Access Management (IAM) lets us administrate and authorize who can take action on specific resources, giving full control and visibility to manage Google Cloud applications and services centrally. Even applications and services have their own built-in security level. For example, BigQuery can provide fine-grained access to sensitive columns and rows using policy tags, a type-based classification of your data to create access policies for specific types of users [64]. Our modern data stack also has features for this problem. With apache superset, besides account level, we can even have deeper privacy in the platform by row-level security and dashboard, chart, and database.

The next problem is about regulations for special data. For example, Europe's General Data Protection Regulation (GDPR), the California Consumer Protection Act (CCPA), Health Insurance Portability and Accountability Act (HIPAA) also have protection rules against electronic health records. We can address this problem by masking or removing PHI and PII data within the ETLT process as mentioned in chapter 2.3.

Data Observation and other applications from the modern data stack in 2.5 also have the features to enhance the organization's visibility into where its critical data resides, what the data they have, when they collect it, what purposes it's been used for and keeping track of who has access to it. Therefore, your data platform can pass some other requirements in popular regulation tests like the "GDPR test" with flying colors [65].

Finally, all cloud providers are working towards signing different compliance certifications to guarantee their security and privacy. For example, Microsoft Azure has more than 90 compliance certifications, including over 50 specific to global regions and countries, such

as the US, the European Union, Germany, Japan, the United Kingdom, India, and China. More than 35 compliance offerings specific to the needs of key industries, including health, government, finance, education, manufacturing, and media [66]. Overall, they are big old corporations that have a lot of prestige and relationships with other big corporations as well as governments around the world. So believing in them will definitely bring greater benefits in terms of money, time, and effort, as well as effectiveness in terms of security and privacy, than we implement it ourselves.

4 Implementation and Integration

In this chapter, we will go into detail about how to implement the system. The basic installation of the application will not be mentioned here. We only present the overall process and additional steps required for our solution and system's integration. Finally, we will present the final results of our implementation process.

4.1 Prerequisite Knowledge

Before start implementing the solutions and integrating their components, we need to have the prerequisite set up knowledge about the below services and applications as we will not include introduction and basic installation:

- Synthea - 100k synthetic patients records with COVID-19 [67]
- OMOP Common Data Model v5.4 in OHDSI Athena [68]
- Amazon Identity and Access Management (AWS IAM) [69]
- Amazon Simple Storage Service (AWS S3) [70]
- Google Cloud Identity And Access Management (GCP IAM) [71]
- GCP BigQuery [72]
- GCP Cloud Storage (GCS) [73]
- Delta Lake (PySpark and Apache Parquet) [74]
- GCP Dataproc [75]
- GCP Compute Engine [76]
- Docker and Docker Compose [77]
- Apache Airflow [78] or Cloud Composer [79]
- dbt cloud [80]
- Apache Superset [81]
- Linkedin DataHub [82]

After that, we can start to implement and integrate components. The source code of the project is available at the author's Github repository [83].

4.2 Setup

First, we need to create a Google Cloud project. **The Project ID**, for example, *bachelor-thesis-344103*, is very important and will be used several times on the project. Then we create a service account and service account key for authentication. A service account is a special kind of account used by an application or computer workload. To get the permissions that we need to manage service account keys, we must grant our account necessary IAM roles. After that, we can generate a Service Account JSON key which will be used for other applications to connect to the GCP project on the **Service Account** page [84].

The generated key has the following format (figure 4.1), where *private-key* and *private-key-id* are the private parts of the key pair:

```
{
  "type": "service_account",
  "project_id": "project-id",
  "private_key_id": "key-id",
  "private_key": "-----BEGIN_PRIVATE_KEY-----\nprivate-key\n-----END_PRIVATE_KEY-----\n",
  "client_email": "service-account-email",
  "client_id": "client-id",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://accounts.google.com/o/oauth2/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/metadata/service-account-email"
}
```

Figure 4.1: Service Account JSON file

The next part is to set up a **Compute Engine** VM instances for Apache Airflow, Apache Superset, and Datahub. The machine must be in *asia-southeast1 (Singapore)* region for minimizing the latency. Our recommendation for the machine configuration is *e2-standard-4 (4 vCPU, 16 GB memory)*, *Ubuntu 20.04 LTS Operating System*, and *80 GB disk size* to allocate enough hardware resources and meet the requirements for all applications installed in this machine. Before creating the instance, Adding a **dashboard** networking tag in the **VPC Network** to the instance is needed for open port configurations later. To complete the dependency requirements for the applications, we have to install git, docker, jq, and docker-compose. We can run the prepared shell script file *setup.sh* to install them automatically. All applications will be installed and run in docker, which is a tool that allows developers to easily deploy applications in a sandbox (called containers) to run on the host operating system [77].

For deploying the Data Orchestration - Apache Airflow on Docker Compose, we need to fetch Apache Airflow *docker-compose.yaml* file:

```
curl -Lf0 'https://airflow.apache.org/docs/apache-airflow/2.2.5/docker-compose.yaml'
```

Some directories in the container are mounted, which means that the contents are synchronized between our computer and the container. We store the Python-based DAGs and custom plugins in these folders and it will automatically connect with the Airflow system.

- `./dags` - DAG files .

- `./logs` - Logs from task execution and scheduler.
- `./plugins` - Custom plugins.

We have to change the configuration including folder location, port for compatibility between our applications, and turn off loading examples in the *docker-compose.yaml* file to meet our requirements.

```
ports:
  - 8088:8080
export AIRFLOW__CORE__LOAD_EXAMPLES=False
```

All the changes made to the file can be found in the file with the same name in *Airflow* folder. After completing the remaining initializing environment steps (Initialize the Airflow scheduler, database, and other configurations) [78]. We can start all services and start to make our first DAG.

```
docker-compose up
```

4.3 Extract

Now we can ingest the data to GCP with Airflow. We made two versions of Airflow for this step. The local version in the *airflow/dags-local* for transferring files in our computer to the GCP:

- Download and unpack the data
- Upload data and code files to the cloud (AWS S3 and GCP GCS)

and the cloud version in the *airflow/dags* for working completely in the cloud. The local-related process is optional and will not be mentioned here. We assume that everything is available in the cloud and will be processed completely in the cloud for this project.

We created a DAG to automatically pull the data from AWS S3. We move data from it to GCP Cloud Storage. Because we already know the structure and content of the data, this step is just to simulate the extraction process we usually do in the real world, for example, extracting the data from multiple hospitals or clouds. In another terminal, we run *docker-compose ps* to see which containers are up running (there should be 7, matching with the services in the *docker-compose* file). After verifying the docker services and adding the Firewall rule to open port 8088 in **VPC Network** configuration, we can log in to Airflow web UI on *localhost:8088* with default credit: *airflow/airflow*.

There are two connections that need to be set up. A connection to Amazon Web Services *aws_credentials* to pull the data with its credit

```
[AWS]
AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
```

and a connection to Google Cloud Platform for storing data using the JSON file we created before.

The DAG for this task includes (figure 4.2):

- Create buckets for storing data and codes in GCP
- Transfer the data from AWS S3 to GCS.

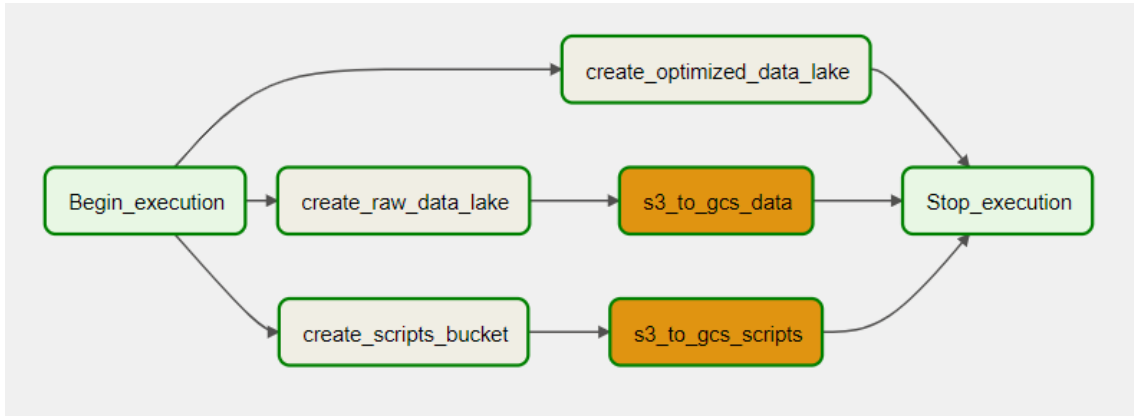


Figure 4.2: Extract DAG

We have successfully established a raw data lake (figure 4.3), which stores data from multiple sources of data in a single source of truth with their original formats.

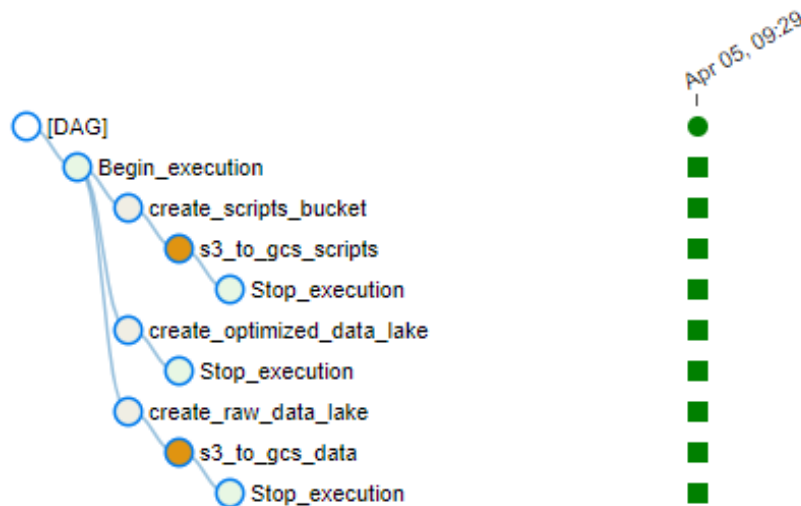


Figure 4.3: Extract DAG

The files now can be seen in the raw_data_lake and scripts_data_lake bucket in GCS (figure 4.4).

<input type="checkbox"/>	raw_data_lake	Apr 5, 2022, 9:29:43 AM	Region	asia-southeas...	Regional	Apr 5, 2022, 9:29:43 AM
<input type="checkbox"/>	scripts_data_lake	Apr 5, 2022, 9:29:44 AM	Region	asia-southeas...	Regional	Apr 5, 2022, 9:29:44 AM

Figure 4.4: Created Bucket in GCS

4.4 Transform

For the first transform step, normally, we have to create the **Datapro** instances and assign the PySpark job to it to do the transformation. But the Airflow can also handle and makes this step automatically. We create a DAG using Google Cloud Dataproc Operators (figure 4.5). The DAG initializes the DataProc Instances using the *setup_transform.sh* file and transforms the data using the *transform_code.py* file, both are stored in the GCS *scripts_datalake* bucket in GCS.

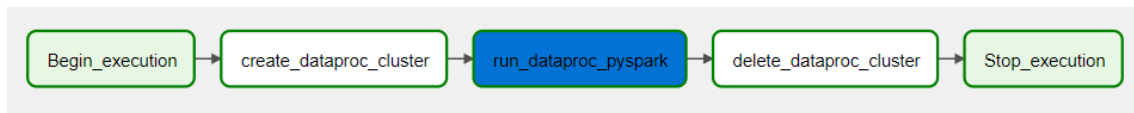


Figure 4.5: Transform DAG

The most important thing is researching and setting up the right variables for the configuration (figure 4.6). For example, Initialization actions file and for creating cluster and Jar files location for submitting PySpark Job, which is not available in any documentation on the websites.

```
create_dataproc_cluster = dataproc_operator.DataprocClusterCreateOperator(
    task_id='create_dataproc_cluster',
    project_id='bachelor-thesis-344103',
    cluster_name='transform-data-lake',
    num_workers=2,
    region='asia-southeast1',
    zone='asia-southeast1-b',
    init_actions_uris=['gs://scripts_data_lake/pip-install.sh'],
    image_version='2.0',
    master_machine_type='n1-standard-4',
    worker_machine_type='n1-standard-4',
    gcp_conn_id = 'gcp_credentials',
    dag = dag
)

run_dataproc_pyspark = dataproc_operator.DataprocSubmitPySparkJobOperator(
    task_id='run_dataproc_pyspark',
    region='asia-southeast1',
    main='gs://scripts_data_lake/transform_code.py',
    cluster_name='transform-data-lake',
    dataproc_jars=['file:///usr/lib/delta/jars/delta-core_2.12-1.0.0.jar'],
    gcp_conn_id = 'gcp_credentials',
    dag = dag
)
```

Figure 4.6: Creating Cluster and Submitting PySpark Job DAGs

The PySpark jobs which are submitted to the DataProc instance include:


1. Create a SparkSession object from your SparkContext (figure 4.7). You can think of SparkContext as your connection to the cluster and SparkSession as your interface with that connection [85]. The SparkSession for Delta Lake is different from regular PySpark SparkSession.

```
conf.set("spark.jars.packages","io.delta:delta-core:1.0.0")
conf.set("spark.sql.extensions","io.delta.sql.DeltaSparkSessionExtension")
conf.set("spark.sql.catalog.spark_catalog","org.apache.spark.sql.delta.catalog.DeltaCatalog")
```


Figure 4.7: Configuring Transformation Code for Delta Lake

2. Read Synthea and OMOP CDM Vocabulary files.
3. Remove duplicated, null, corrupt unused values if needed.
4. Changes format to standard form (date columns).
5. Drop PHI(Protected health information) or PII (Personally identifiable information) information.
6. Compress and Store the data in the Delta Lake format.

The optimized data lake is stored in the *optimized_data_lake* bucket (figure 4.8). Whenever the users want to do the analysis against this data lake, they can run queries against the parquet files and take advantage of the Delta Lake format using Python DataFrame or Spark SQL.

Buckets > optimized_data_lake > patients 

[UPLOAD FILES](#)
[UPLOAD FOLDER](#)
[CREATE FOLDER](#)
[MANAGE HOLDS](#)
[DOWNLOAD](#)
[DELETE](#)

Filter by name prefix only ▼  Filter Filter objects and folders











<input type="checkbox"/>	Name	Size	Type	Created 	Storage class	Last modified
<input type="checkbox"/>	 _delta_log/	—	Folder	—	—	—
<input type="checkbox"/>	 part-00000-77408bee-fd5c-47fb-aa2...	1.2 MB	application/octet-stream	Apr 5, 2022...	Regional	Apr 5, 2022,...
<input type="checkbox"/>	 part-00001-7f0fd04d-6e86-4fab-a77...	1.2 MB	application/octet-stream	Apr 5, 2022...	Regional	Apr 5, 2022,...
<input type="checkbox"/>	 part-00002-a0075d21-c56b-4fc2-bf2...	1.2 MB	application/octet-stream	Apr 5, 2022...	Regional	Apr 5, 2022,...
<input type="checkbox"/>	 part-00003-7f303932-1be9-4af5-8a2...	1.2 MB	application/octet-stream	Apr 5, 2022...	Regional	Apr 5, 2022,...
<input type="checkbox"/>	 part-00004-a001e33e-adbd-4a0d-86...	1.2 MB	application/octet-stream	Apr 5, 2022...	Regional	Apr 5, 2022,...
<input type="checkbox"/>	 part-00005-f1910933-9811-42aa-925...	1.2 MB	application/octet-stream	Apr 5, 2022...	Regional	Apr 5, 2022,...
<input type="checkbox"/>	 part-00006-e962111b-924d-4fed-b4b...	1.2 MB	application/octet-stream	Apr 5, 2022...	Regional	Apr 5, 2022,...
<input type="checkbox"/>	 part-00007-8742d410-ef31-42e7-8f8...	200.2 KB	application/octet-stream	Apr 5, 2022...	Regional	Apr 5, 2022,...

Figure 4.8: Patient tables in Delta Lake Format

4.5 Load

We have to load all the data into the data warehouse. As usual, we have to make two copies of the data when we load it to the data warehouse. But because the project is in the GCP system, we just need to create an external table for **BigQuery** from the **GCS**. **BigQuery** can read the parquet or different format files stored in the GCS and perform queries at high speed, so we don't have to pay a double storage fee. The metadata of the external is available not only in **BigQuery**, but also in different applications such as **dbt** and **DataHub** with tweak configuration. Airflow also handles the automation of this step of us by connecting to **BigQuery** and sending a request with pre-composed queries, which can be found in the *scripts* folder in GCS.

We need to create a task in Airflow for each external table in BigQuery (figure 4.9). After that, we can also add several check operators to check for the data quality of the load process. The load process for OMOP CDM vocabulary is presented in figure 4.10, and the process for Synthe Tables is presented in figure 4.11.

```
create_drug_strength_table = BigQueryCreateExternalTableOperator(  
    task_id="create_drug_strength_table",  
    table_resource={  
        "tableReference": {  
            "projectId": 'bachelor-thesis-344103',  
            "datasetId": optimized_data_warehouse,  
            "tableId": "drug_strength"  
        },  
        "externalDataConfiguration": {  
            "sourceFormat": "PARQUET",  
            "sourceUris": ['gs://optimized_data_lake/DRUG_STRENGTH/*.parquet']  
        },  
    },  
    bigquery_conn_id='gcp_credentials',  
    dag = dag  
)  
  
check_raw_dw_quality = BigQueryCheckOperator(  
    task_id = "check_raw_dw_quality",  
    sql = "SELECT COUNT(*) FROM raw_data_warehouse.allergies",  
    use_legacy_sql=False,  
    location='asia-southeast1',  
    bigquery_conn_id= 'gcp_credentials',  
    dag = dag  
)
```

Figure 4.9: Creating External Tables and Data Quality Check for BigQuery



Figure 4.10: Load DAG Part 1

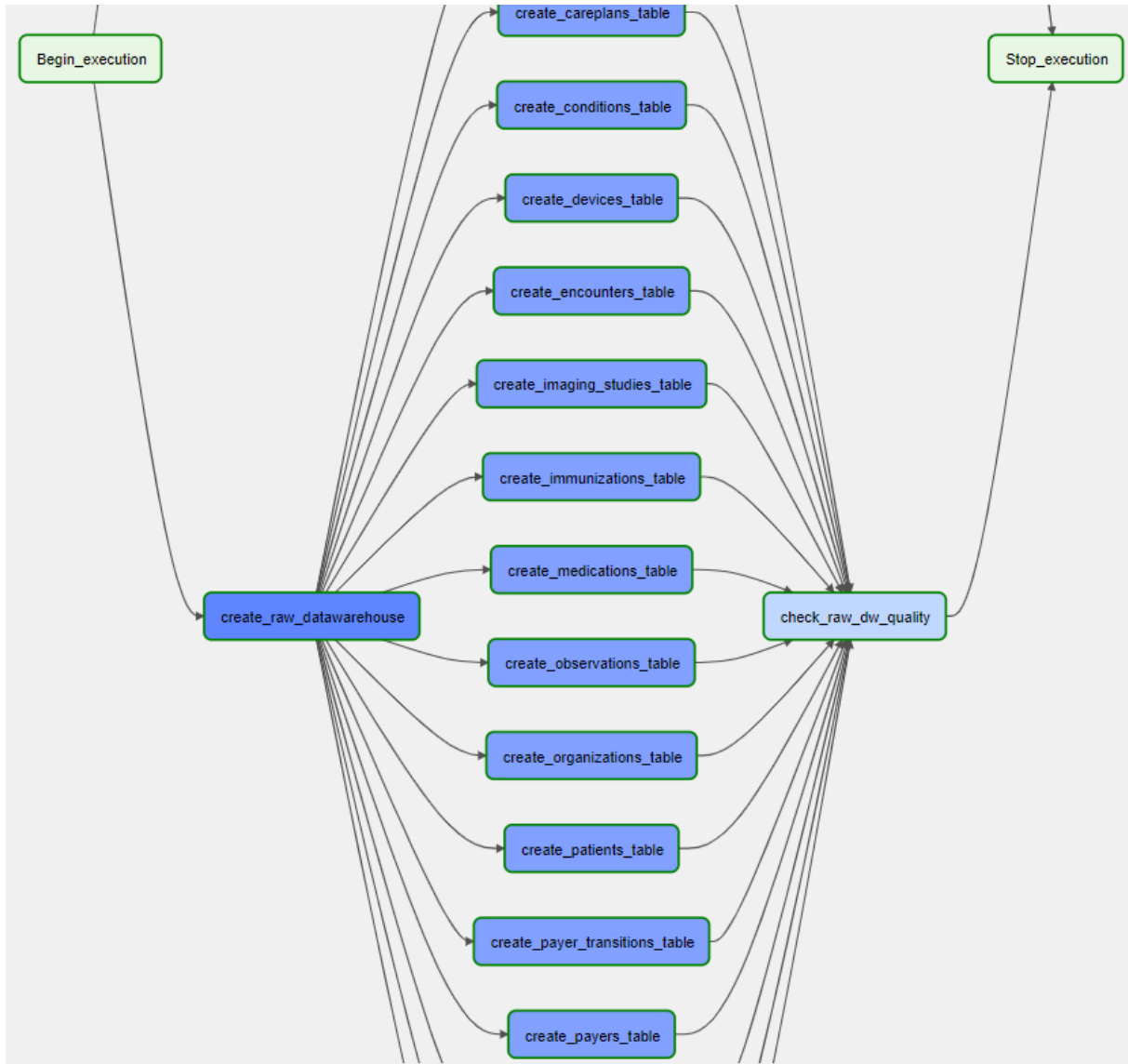


Figure 4.11: Load DAG Part 2

4.6 Second Transform

We use dbt Cloud to transform the data. We need to create and initialize auto-generated essential components for a project, then connect it to Github to make use of features like version-control and establish a connection to our BigQuery database. The configuration files that needed to be added are *dbt_project.yml* for the project and *schema.yml* for the dataset in BigQuery. After that, we can combine SQL and Jinja (Python) Macros to transform the data or add more packages to support our transformation. The queries are based on the ETL OMOP tutorial as mentioned in chapter 3.4 (figure 4.12).

```
{{ config(materialized='table')}}

WITH PreDrugTarget AS(
  SELECT
    d.drug_exposure_id,
    d.person_id,
    c.concept_id AS ingredient_concept_id,
    d.drug_exposure_start_datetime AS drug_exposure_start_datetime,
    d.days_supply AS days_supply,
    COALESCE(
      IFNULL(drug_exposure_end_datetime, NULL),
      IFNULL(date_add(drug_exposure_start_datetime,INTERVAL cast(days_supply as int) DAY), drug_exposure_start_datetime),
      date_add(drug_exposure_start_datetime,INTERVAL 1 DAY)
    ) AS drug_exposure_end_datetime
  FROM {{ref('drug_exposure')}} d
  INNER JOIN {{source('optimized_data_warehouse','concept_ancestor')}} ca ON CAST(ca.descendant_concept_id AS STRING) = CAST(d.drug_concept_id AS STRING)
  INNER JOIN {{source('optimized_data_warehouse','concept')}} c ON CAST(ca.ancestor_concept_id AS STRING)= CAST(c.concept_id AS STRING)
  WHERE c.vocabulary_id = 'RxNorm'
  AND c.concept_class_id = 'Ingredient'
  AND d.drug_concept_id != CAST(0 AS STRING)
  AND coalesce(d.days_supply,0) >= 0
),
SubExposureEndDates AS (
  SELECT person_id, ingredient_concept_id, event_date AS end_datetime
  FROM
  (
    SELECT person_id, ingredient_concept_id, event_date, event_type,
    MAX(start_ordinal) OVER (PARTITION BY person_id, ingredient_concept_id ORDER BY event_date, event_type ROWS unbounded preceding) AS start_ordinal,
    ROW_NUMBER() OVER (PARTITION BY person_id, ingredient_concept_id ORDER BY event_date, event_type) AS overall_ord
    FROM (
      SELECT person_id, ingredient_concept_id, drug_exposure_start_datetime AS event_date,
      -1 AS event_type,
      ROW_NUMBER() OVER (PARTITION BY person_id, ingredient_concept_id ORDER BY drug_exposure_start_datetime) AS start_ordinal
      FROM PreDrugTarget
      UNION ALL
      SELECT person_id, ingredient_concept_id, drug_exposure_end_datetime, 1 AS event_type, NULL
      FROM PreDrugTarget
    ) RAWDATA
  ) e
  WHERE (2 * e.start_ordinal) - e.overall_ord = 0
),
```

Figure 4.12: dbt Transformation using SQL

We can check the auto-generated DAG to see the workflow and dependencies of all related data models (figure 4.38 and 4.39):

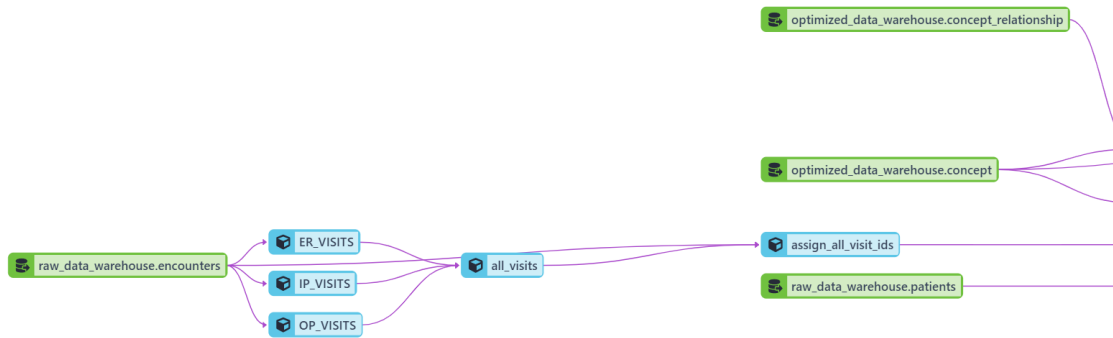


Figure 4.13: Data Lineage for for table drug_era part 1

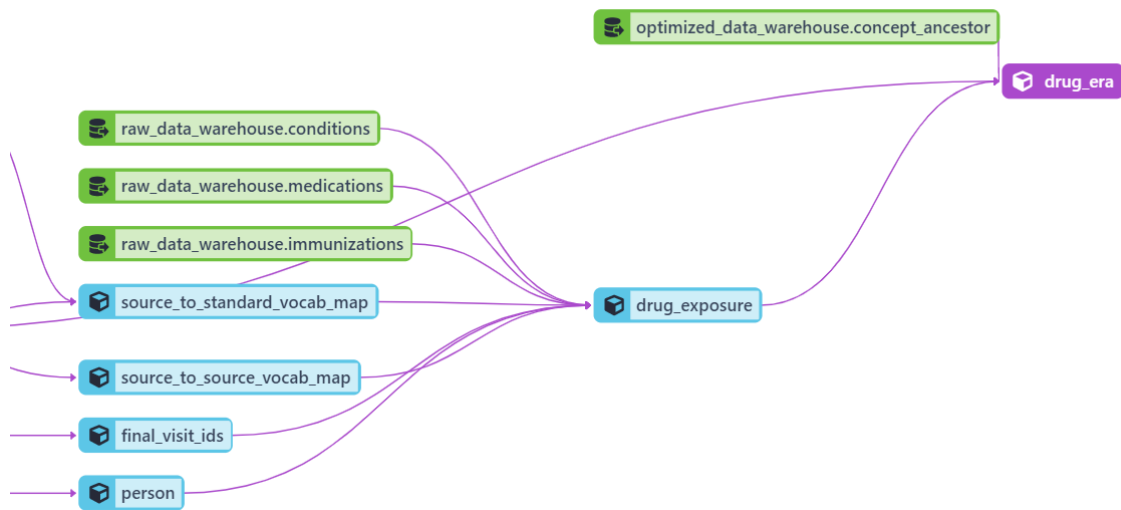


Figure 4.14: Data Lineage for for table drug_era part 2

We can run the data model separately or run all of them together with *dbt run* commands. *dbt* will check the dependencies and run it in order. If one model fails, all transformations related to it in the workflow will be canceled as in figure 4.15.

> ER_VISITS	✓
> IP_VISITS	✓
> OP_VISITS	✓
> person	✓
> source_to_source_vocab_map	✓
> source_to_standard_vocab_map	▲
> all_visits	✓
> observation_period	✓
> assign_all_visit_ids	✓
> final_visit_ids	✓
> condition_occurrence	●
> drug_exposure	●
> measurement	●
> procedure_occurrence	●
> visit_occurrence	✓
> condition_era	●
> drug_era	●

Figure 4.15: dbt Failed Run

After all, our second transformation is complete with the last table required to scan through 9.8 GB of data and transformed with all requirements and dependencies in 46 seconds (figure 4.16).

SYSTEM LOGS
view logs
<div> <div> <div>● Summary</div> <div>○ Details</div> </div> <div> 17:13:36 Found 17 models, 0 tests, 0 snapshots, 0 analyses, 188 macros, 0 operations, 0 seed files, 10 sources, 0 exposures, 0 metrics 17:13:37 Concurrency: 4 threads (target='default') 17:14:23 Finished running 1 table model in 46.73s. 17:14:23 Completed successfully! 17:14:23 Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1 </div> </div>
DETAILS
<div> <div> <div>● Summary</div> <div>○ Details</div> </div> <div> 17:13:37 1 of 1 START table model optimized_data_warehouse.drug_era..... [RUN] 17:14:23 1 of 1 OK created table model optimized_data_warehouse.drug_era..... [CREATE TABLE (415.5k rows, 9.8 GB processed)] in 46.00s </div> </div>

Figure 4.16: dbt run Drug_Era table information

4.7 Integration and Present Insights

We use Apache Superset and LinkedIn Datahub to present data visualization and data catalog. The requirements for these applications are installed with the script in the Setup sections. For this step, we first install Apache Superset following the default local installation guide using Docker Compose [81]. By default, Superset does not include BigQuery connectors, so

we have to add the additional Database Drivers for BigQuery before the first time running it following the author's recommendation (figure 4.17):

```
1 touch ./docker/requirements-local.txt
2 echo "pybigquery" >> ./docker/requirements-local.txt
3 sudo docker-compose build --force-rm
```

Figure 4.17: Install BigQuery Driver

Using the default recommendation for adding BigQuery will cause an error in Superset that can not load the schema for external tables. After researching and testing with different methods, we have found an alternative approach to fix this problem (figure 4.18):

```
1 touch ./docker/requirements-local.txt
2 echo "sqlalchemy-bigquery" > ./docker/requirements-local.txt
3 sudo docker-compose build --force-rm
```

Figure 4.18: Install BigQuery Driver fix

Additional configurations that required for the integration are changing the port 8088 to 8888 and disabling example load in `.env` file and `.env-non-dev` file. After that, we can start the applications with docker-compose.

ports:

- 8888:8088

SUPERSET_LOAD_EXAMPLES=no

```
sudo docker-compose -f docker-compose-non-dev.yml pull
```

```
sudo docker-compose -f docker-compose-non-dev.yml up
```

In the Apache Superset platform available at port 8888 of the instance, We can config the user, role, security for each user to login and perform activities in the platform. After having access, We can write SQL or preview tables in SQL Lab. After create charts using SQL Editor, we can combine charts to make dashboards.

For the DataHub application, after we follow the quick start installation tutorial [82], we need to install additional plugins for our metadata ingestion include (figure 4.19):

```
1 sudo pip install 'acryl-datahub[datahub-rest]'
2 sudo pip install 'acryl-datahub[bigquery]'
3 sudo pip install 'acryl-datahub[superset]'
```

Figure 4.19: Plugins for Metadata Ingestion Sources

For Superset, we need to submit a curl request to add Superset as a Data Platform (figure 4.20).

```
curl 'http://localhost:8080/entities?action=ingest' -X POST --data '{
  "entity":{
    "value":{
      "com.linkedin.metadata.snapshot.DataPlatformSnapshot":{
        "aspects":[{"com.linkedin.dataplatform.DataPlatformInfo":{
          "datasetNameDelimiter":"Superset","name":"superset",
          "displayName":"Superset","type":"OTHERS","logoUrl":
            "https://raw.githubusercontent.com/datahub-project/datahub/master/datahub-web-react/
            src/images/supersetlogo.png"}}],
          "urn":"urn:li:dataPlatform:superset"}]]}'
```

Figure 4.20: Adding Superset as Data Platform request

Upon completion of this step, we should be able to navigate to the DataHub UI at <http://localhost:9002> in our browser. We use GCP Json key file for GCP and configure a custom recipe with the generated access token for Superset. With dbt, we need to generate *dbt manifest file*, *dbt catalog file* and *dbt sources file* in the dbt Cloud for DataHub Ingestion. On the home page, we can see the summary of the data platform, recently viewed and most popular activities.

4.8 Results

4.8.1 Data Lake Storage in GCS

The first outcome of the project is that we have successfully created a Data lake in GCP Cloud Storage. The original CSV format files are approximately 9.5 GB and then compressed into 3 GB Delta Lake files in GCS. We can see all the folders containing delta lake files for each table in figure 4.21. It is suitable for data scientists or users with programming languages to perform advanced analytics and machine learning compared to the raw data.

<input type="checkbox"/>	Name	Size	Type	Created ?	Storage class
<input type="checkbox"/>	CONCEPT/	—	Folder	—	—
<input type="checkbox"/>	CONCEPT_ANCESTOR/	—	Folder	—	—
<input type="checkbox"/>	CONCEPT_CLASS/	—	Folder	—	—
<input type="checkbox"/>	CONCEPT_RELATIONSHIP/	—	Folder	—	—
<input type="checkbox"/>	CONCEPT_SYNONYM/	—	Folder	—	—
<input type="checkbox"/>	DOMAIN/	—	Folder	—	—
<input type="checkbox"/>	DRUG_STRENGTH/	—	Folder	—	—
<input type="checkbox"/>	RELATIONSHIP/	—	Folder	—	—
<input type="checkbox"/>	VOCABULARY/	—	Folder	—	—
<input type="checkbox"/>	allergies/	—	Folder	—	—
<input type="checkbox"/>	careplans/	—	Folder	—	—
<input type="checkbox"/>	conditions/	—	Folder	—	—
<input type="checkbox"/>	devices/	—	Folder	—	—
<input type="checkbox"/>	encounters/	—	Folder	—	—
<input type="checkbox"/>	imaging_studies/	—	Folder	—	—
<input type="checkbox"/>	immunizations/	—	Folder	—	—
<input type="checkbox"/>	medications/	—	Folder	—	—
<input type="checkbox"/>	observations/	—	Folder	—	—
<input type="checkbox"/>	organizations/	—	Folder	—	—
<input type="checkbox"/>	patients/	—	Folder	—	—
<input type="checkbox"/>	payer_transitions/	—	Folder	—	—
<input type="checkbox"/>	payers/	—	Folder	—	—
<input type="checkbox"/>	procedures/	—	Folder	—	—
<input type="checkbox"/>	providers/	—	Folder	—	—
<input type="checkbox"/>	supplies/	—	Folder	—	—

Figure 4.21: All Delta Lake Tables in GCS

Each folder contains delta lakes files for the corresponding table, which is compressed three times smaller into Apache Parquet and delta logs. We can access the Apache Parquet with any programming language like Scala, SQL, or Python to do the analysis without requiring any specific skills. We can use delta logs to support ACID transactions, schema evolution (changing schema of the tables without breaking all files), time travel (switching between versions of the data lake captured at different times) as in figure 4.22.


















<input type="checkbox"/>	Name	Size	Type	Created ?	Storage class
<input type="checkbox"/>	 _delta_log/	—	Folder	—	—
<input type="checkbox"/>	 part-00000-f6f11e1a-b311-4812-8eb...	14 MB	application/octet-stream	Apr 5, 2022...	Regional
<input type="checkbox"/>	 part-00001-c9aeee52-dc07-40f0-af72...	14.2 MB	application/octet-stream	Apr 5, 2022...	Regional
<input type="checkbox"/>	 part-00002-92468be6-8c54-43e9-861...	13.6 MB	application/octet-stream	Apr 5, 2022...	Regional
<input type="checkbox"/>	 part-00003-0a4519d3-974d-49dd-af7...	15.4 MB	application/octet-stream	Apr 5, 2022...	Regional
<input type="checkbox"/>	 part-00004-69e75e64-d88a-4415-a7d...	15.3 MB	application/octet-stream	Apr 5, 2022...	Regional
<input type="checkbox"/>	 part-00005-56c91a77-b36a-4a8f-b1c...	13.6 MB	application/octet-stream	Apr 5, 2022...	Regional
<input type="checkbox"/>	 part-00006-111b63db-de8d-44e7-90d...	18.2 MB	application/octet-stream	Apr 5, 2022...	Regional
<input type="checkbox"/>	 part-00007-77d97dd9-7bbb-4302-942...	19.3 MB	application/octet-stream	Apr 5, 2022...	Regional
<input type="checkbox"/>	 part-00008-c2447c95-f661-4129-b61...	18.4 MB	application/octet-stream	Apr 5, 2022...	Regional
<input type="checkbox"/>	 part-00009-eb8a0269-e59f-4928-957...	16.4 MB	application/octet-stream	Apr 5, 2022...	Regional
<input type="checkbox"/>	 part-00010-66abfbba-37f8-41a0-993c...	18.7 MB	application/octet-stream	Apr 5, 2022...	Regional
<input type="checkbox"/>	 part-00011-c9b8b36d-538c-4143-916...	18.5 MB	application/octet-stream	Apr 5, 2022...	Regional
<input type="checkbox"/>	 part-00012-f4a603c1-6081-4312-b0b...	15.8 MB	application/octet-stream	Apr 5, 2022...	Regional
<input type="checkbox"/>	 part-00013-ca842498-7b71-465a-ae3...	19.3 MB	application/octet-stream	Apr 5, 2022...	Regional
<input type="checkbox"/>	 part-00014-a7a7d609-4417-4d9b-bb3...	18.9 MB	application/octet-stream	Apr 5, 2022...	Regional
<input type="checkbox"/>	 part-00015-a52a02e7-a760-42ab-a48...	12.8 MB	application/octet-stream	Apr 5, 2022...	Regional

Figure 4.22: Delta Lake Files for the CONCEPT_RELATIONSHIP table

4.8.2 Data Warehouse in BigQuery

We have created a cloud data warehouse in BigQuery (figure 4.23). It uses one of the best data warehouse technologies nowadays. We do not have to take care of any configuration or optimizations for the query.

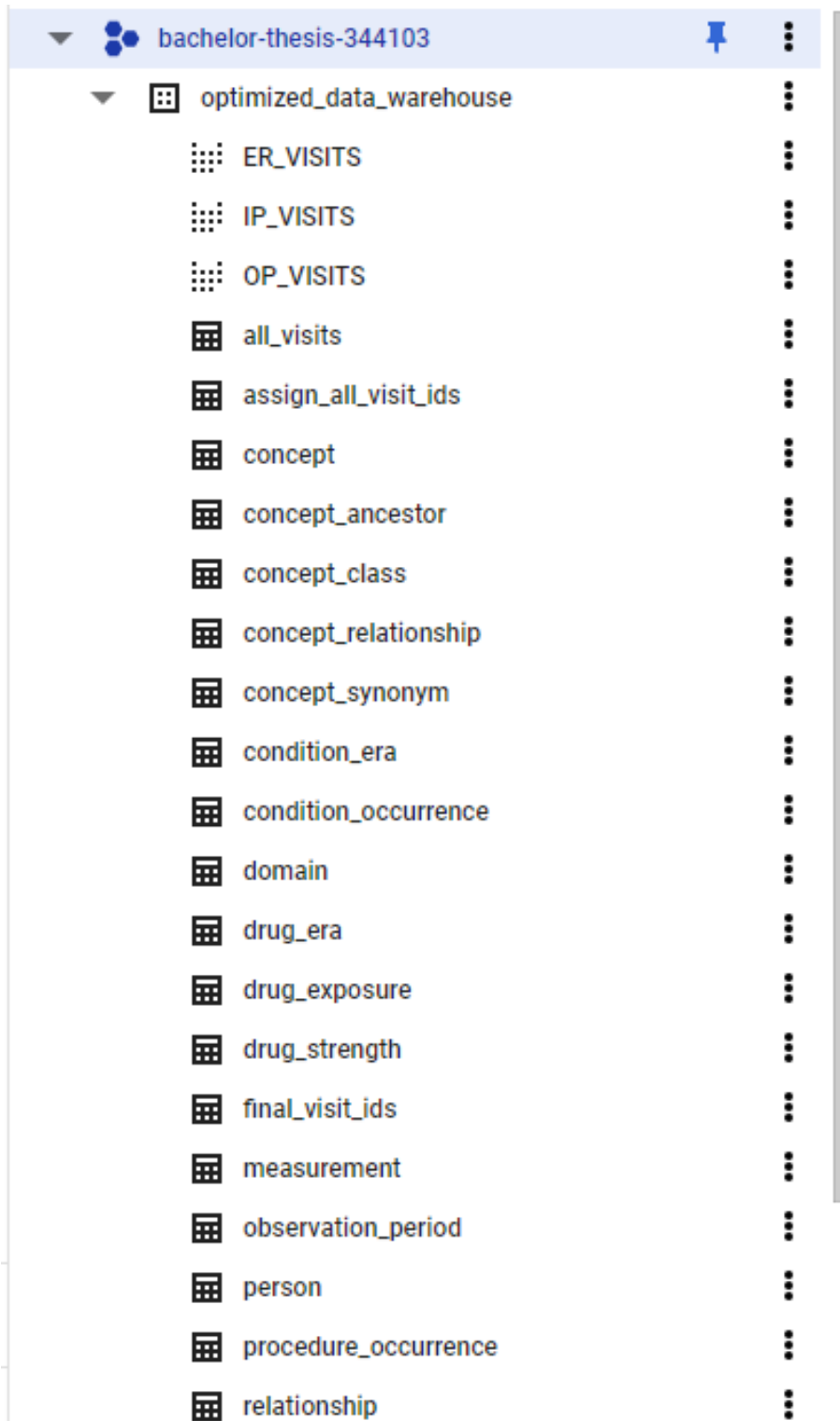


Figure 4.23: Tables in BigQuery

drug_era							
SCHEMA		DETAILS		PREVIEW			
Row	drug_era_id	person_id	drug_concept_id	drug_era_start_date	drug_era_end_date	drug_exposure_count	gap_days
1	57	22	701322	2009-02-07	2009-02-07	1	14282
2	72	27	1521369	2018-10-11	2019-10-06	1	18175
3	91	36	974166	1950-10-16	1972-02-14	22	774
4	125	47	40226742	2011-08-07	2019-06-24	9	18071
5	169	69	1124957	1945-11-24	1945-11-24	1	-8804
6	197	76	1192710	1972-08-10	1972-08-10	1	952
7	201	77	1549786	2019-01-25	2020-04-30	2	18382
8	217	79	40226742	1975-08-17	2020-01-27	50	18288
9	273	104	1149380	1993-08-28	2019-12-18	29	18248
10	361	129	1137529	1989-06-20	2019-12-10	34	18240
11	437	144	1125315	2015-12-13	2016-02-16	2	16817
12	509	162	1125315	2020-03-06	2020-03-06	1	18327
13	513	165	1322184	1997-02-01	2019-06-08	28	18055
14	530	172	1515774	2018-11-25	2019-11-20	1	18220
15	542	175	1125315	2020-03-11	2020-03-11	1	18332
16	546	178	1149380	2020-01-20	2020-01-20	1	18281

RUN **SAVE** **SHARE** **SCHEDULE** **MORE** ✓ This query will process 1.27 GB when run.

DAG	Owner	Runs	Schedule	Last Run	Next Run
extract	hieu	1/3	00 11*	2022-04-05, 02:29:38	2023-01-01, 00:00:00
load	hieu	1/3	00 11*	2022-04-05, 08:58:24	2023-01-01, 00:00:00
transform	hieu	1/3	00 11*	2022-04-05, 07:38:06	2023-01-01, 00:00:00

Figure 4.26: Data Orchestrator Platform ETL process

4.8.4 Data Visualizations platform with Apache Superset

The next result is the Visualizations and Business Intelligence platform with Apache Superset. We can access the platform via the URL: <http://34.142.144.58:8888/> or in localhost port 8888 in our machine. We can access the SQL Editor to preview the data in our data warehouse and explore the results with visualizations as in figure 4.27. For example, we can see that there are 124k patients with more than females.

SQL Query:

```

1 with tt as
2 (
3     SELECT
4         extract(year from t.drug_era_start_date) as year_of_era,
5         t.drug_concept_id,
6         c.concept_name as drug_concept_name
7     FROM
8         optimized_data_warehouse.drug_era t,
9         optimized_data_warehouse.concept c
10    where
11        t.drug_concept_id = c.concept_id
12 ),
13 tt2 as
14 (
15     SELECT
16         tt.drug_concept_name,

```

Results Table (40 rows returned):

drug_concept_name	drug_concept_id	year_of_Era	drug_count	n
simvastatin	1539403	2017	602	1
furosemide	956874	2017	586	2
clopidogrel	1322184	2017	371	3
naproxen	1115008	2017	371	4
hydrochlorothiazide	974166	2017	343	5
amlodipine	1332418	2017	303	6
nitroglycerin	1361711	2017	301	7
metoprolol	1307046	2017	298	8
alendronate	1557272	2017	287	9
epinephrine	1343916	2017	265	10
acetaminophen	1125315	2020	21676	1
enoxanarin	1301025	2020	18131	2

Figure 4.27: SQL Explore and Preview

The platform is one of the BI tools with the most charts available nowadays, as in figure 4.28.

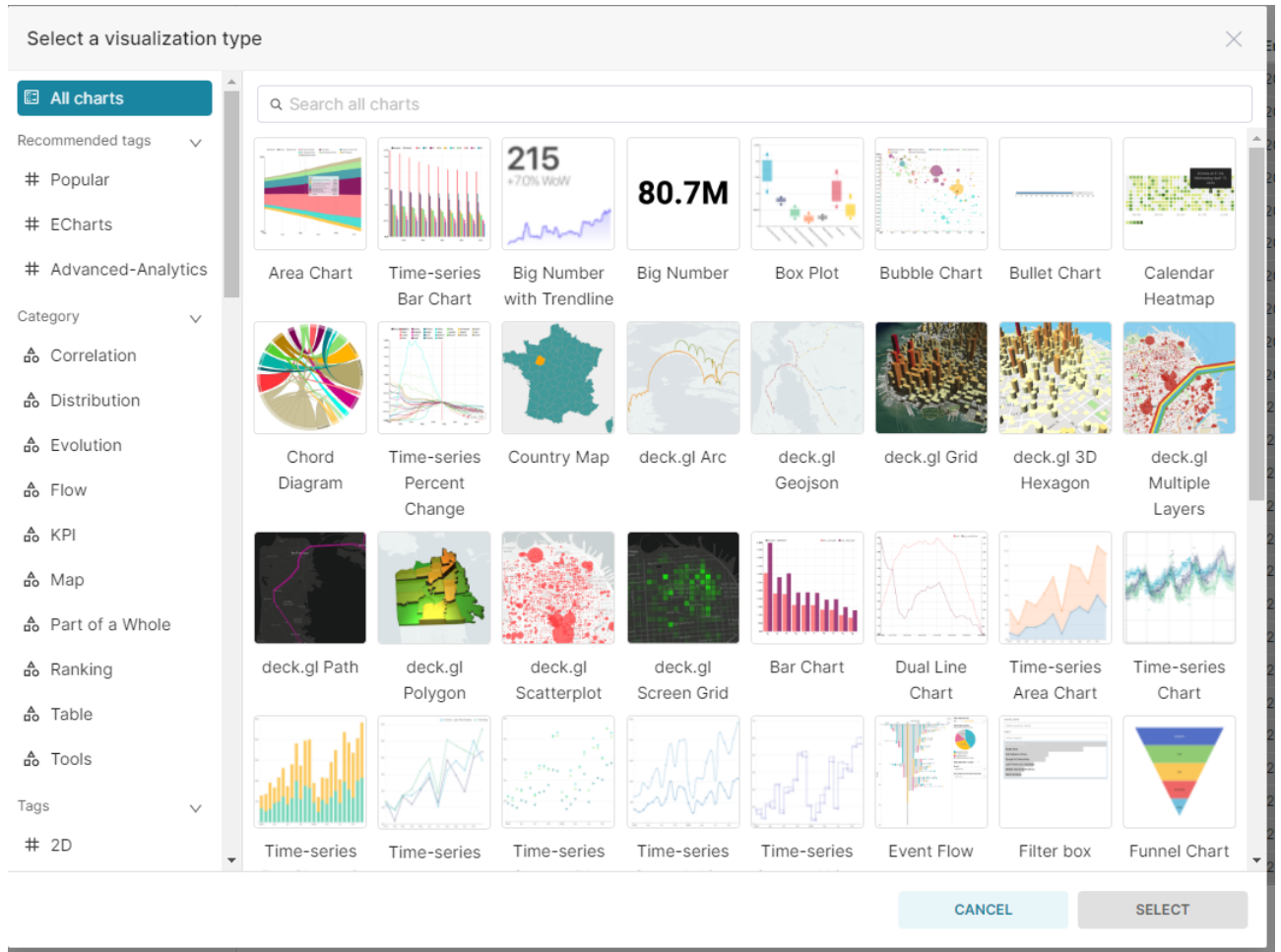


Figure 4.28: Superset Charts

We have created two examples of dashboards. The first one is about the Patient Demographics of our dataset in figure 4.29.

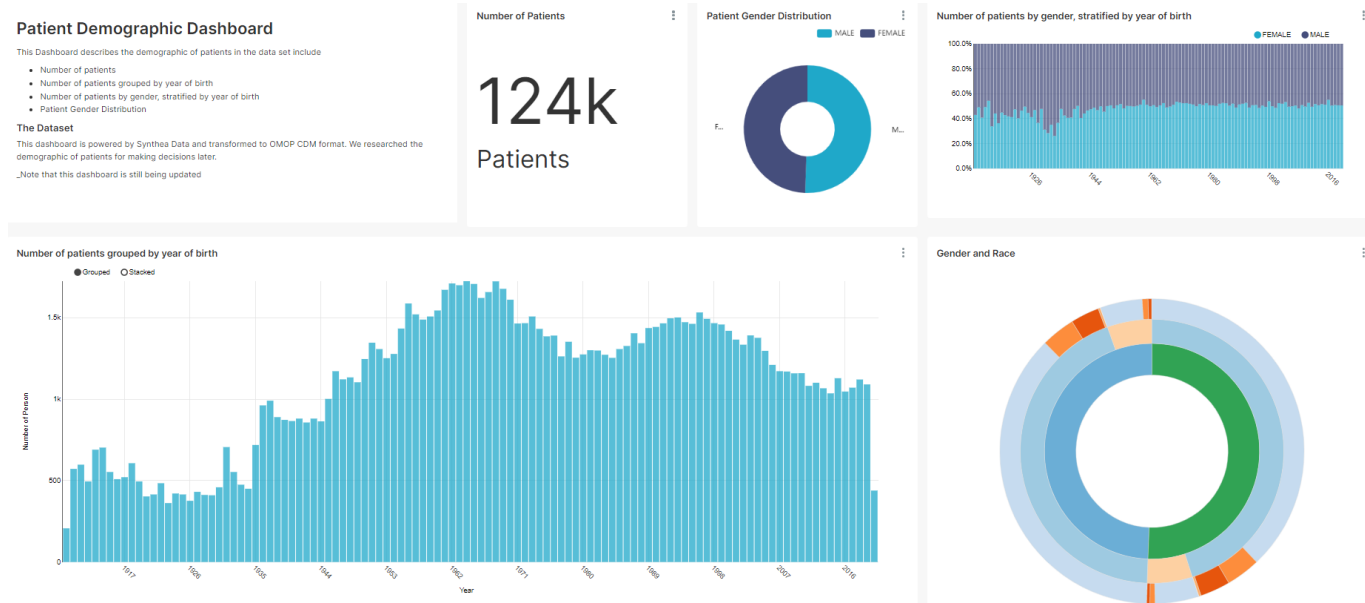


Figure 4.29: Patient Demographic Dashboard

The second dashboard is about Drug Usage Trends of the Patients as shown in figure 4.30. We can see that people are taking more and more drugs these days.

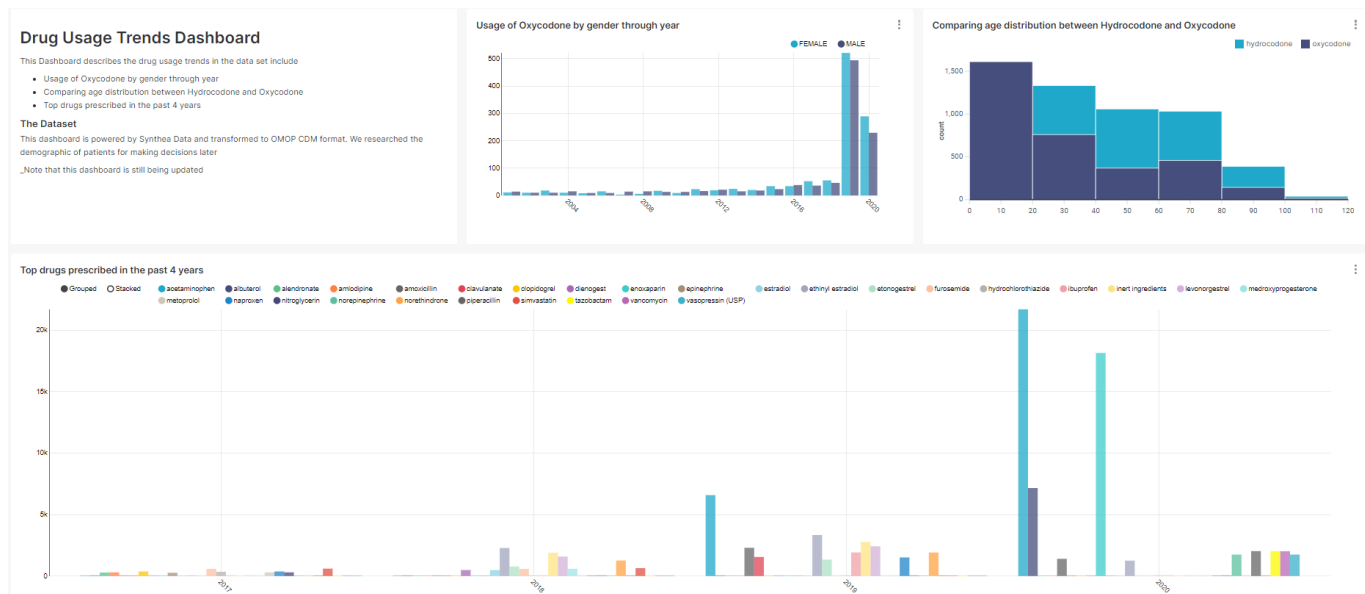


Figure 4.30: Drug Usage Trend Dashboard

4.8.5 Data Catalog platform with DataHub

We can discover data, user activities, analytics, and applications in our system or platform in our Data Catalog Platform based on DataHub. We can access the platform via the URL: <http://34.142.144.58:9002> or in localhost port 9002 in our machine. On the home page, we can see the summary of Metadata and Data Platforms, which are 63 Datasets, 12 Dashboards, 110 Charts, and 122, 38, 27 elements related to Superset, BigQuery, dbt data platforms, respectively, in figure 4.31.

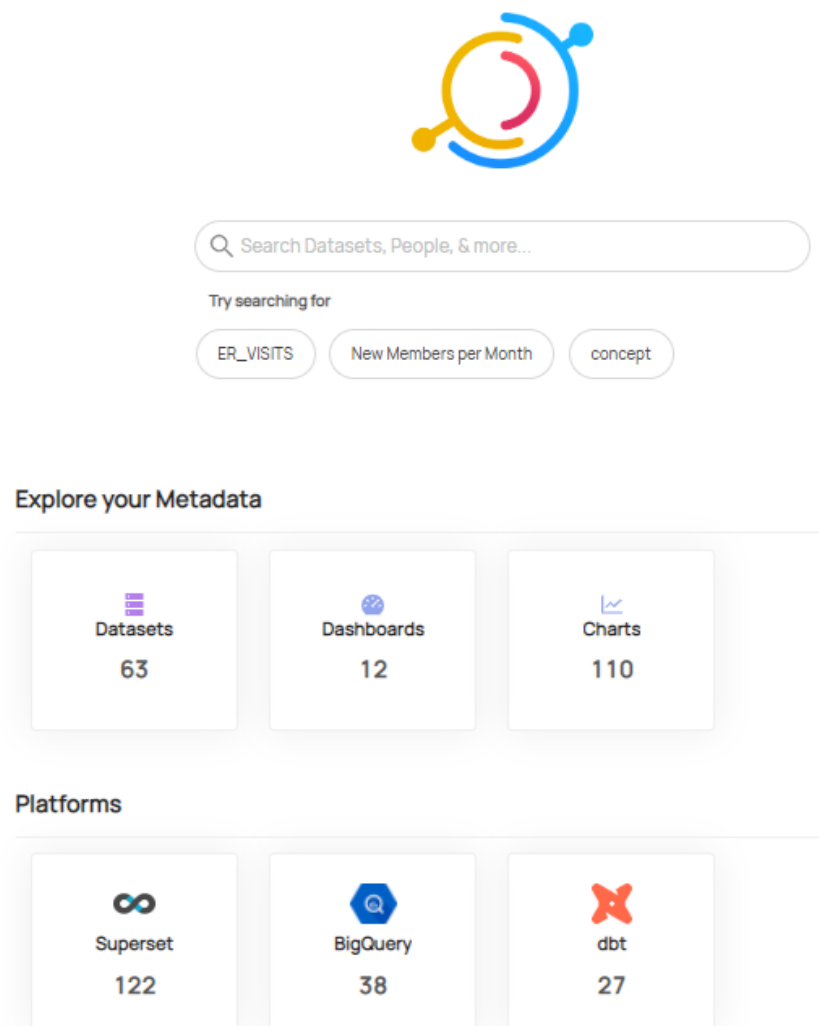


Figure 4.31: Datahub Homepage

We can see the latest recently activities of users or most popular attributes in our platform in figure 4.32.

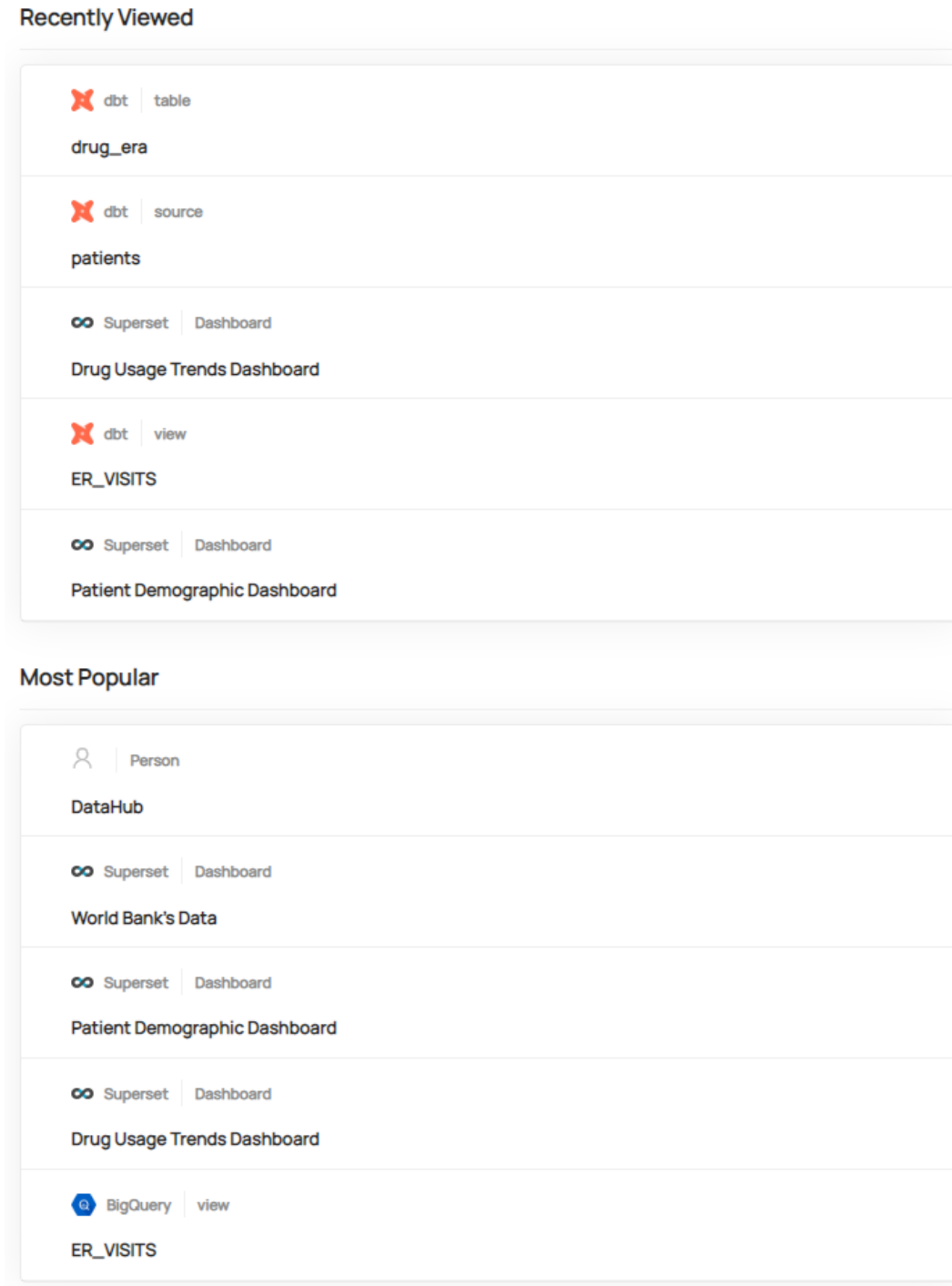


Figure 4.32: Datahub Homepage Activities Summary

We can see the analytics about the activities of users on it. For example, How many Weekly Active Users, number of Searches Last Week, Section Views across Entity Types including schema and documentation, Actions by Entity Type, and finally, Top Viewed Dataset as in figure 4.33.

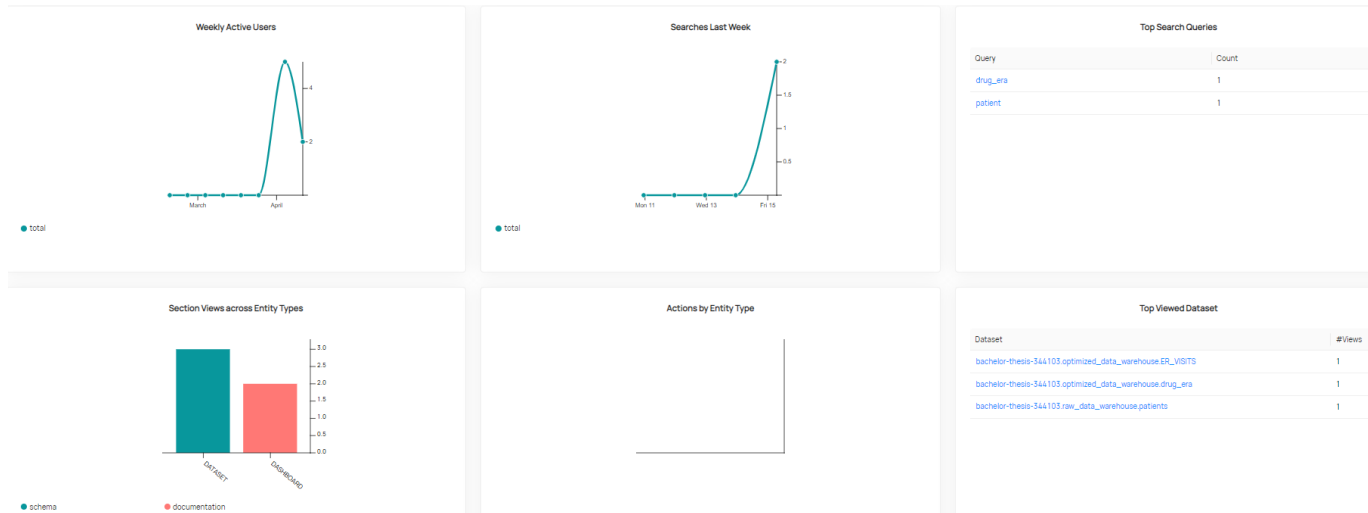


Figure 4.33: User's activities reports

We search for almost anything that exists in our platform, for example, terms, data, applications, etc., through all our data platforms and data sets (figure 4.34).

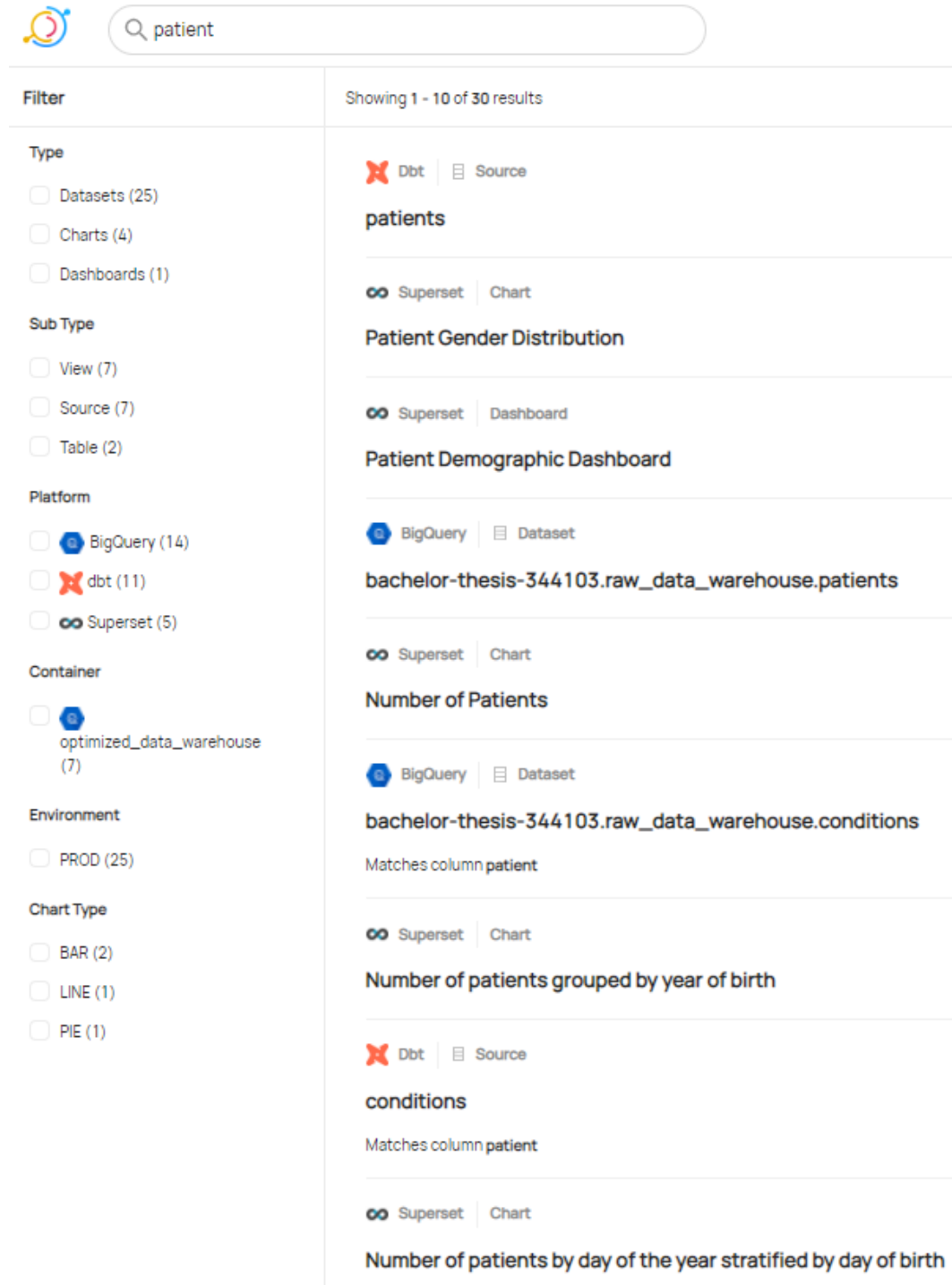


Figure 4.34: Searching data or applications in Datahub

Finally, we can see the DAG data lineage of how the data flow in our data systems like Airflow and dbt. As in figure 4.35, we can see the Drug Usage Dashboard is the combination of 3 charts that are generated from BigQuery queries.



Figure 4.35: Data Lineage of Superset Dashboard and BigQuery

As in figure 4.36, We can see the `drug_era` tables are made from different queries in dbt and BigQuery.

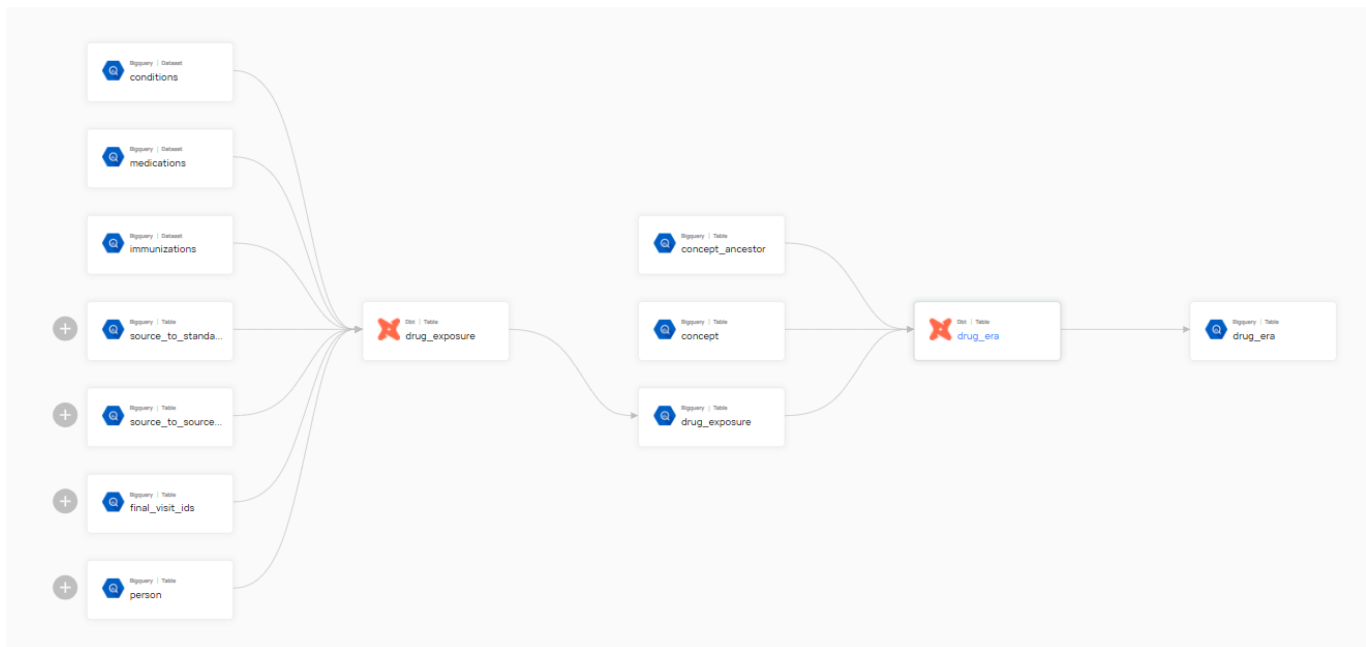
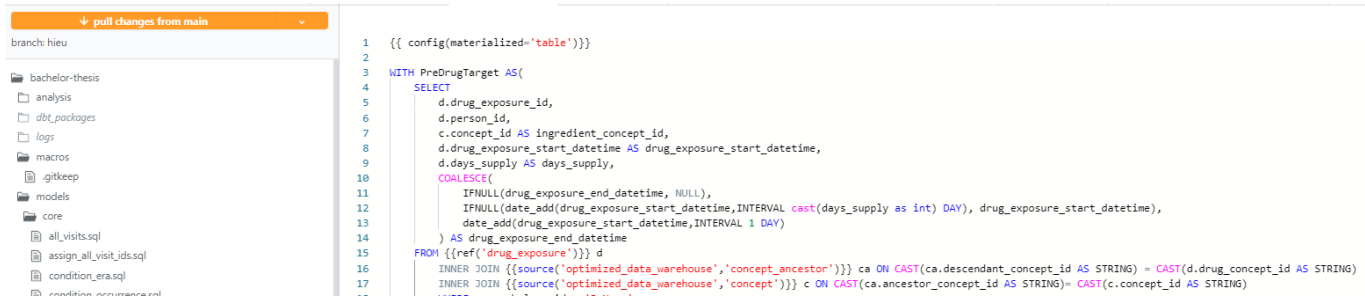


Figure 4.36: Data Lineage of dbt and BigQuery

4.8.6 Second Transformation with dbt Cloud

We can do more transformation with the help of dbt Cloud. It gives us a lot of advantages like: Git workflow and version control, oriented programming (Python), Data Lineage, Testing, and Documentation.

We can combine GitHub, programming, and macro with SQL queries as in figure 4.37



```

1  {{ config(materialized='table')}}
2
3  WITH PreDrugTarget AS(
4    SELECT
5      d.drug_exposure_id,
6      d.person_id,
7      c.concept_id AS ingredient_concept_id,
8      d.drug_exposure_start_datetime AS drug_exposure_start_datetime,
9      d.days_supply AS days_supply,
10     COALESCE(
11       IFNULL(drug_exposure_end_datetime, NULL),
12       IFNULL(date_add(drug_exposure_start_datetime,INTERVAL cast(days_supply as int) DAY), drug_exposure_start_datetime),
13       date_add(drug_exposure_start_datetime,INTERVAL 1 DAY)
14     ) AS drug_exposure_end_datetime
15   FROM {{ref('drug_exposure')}} d
16   INNER JOIN {{source('optimized_data_warehouse','concept_ancestor')}} ca ON CAST(ca.descendant_concept_id AS STRING) = CAST(d.drug_concept_id AS STRING)
17   INNER JOIN {{source('optimized_data_warehouse','concept')}} c ON CAST(ca.ancestor_concept_id AS STRING)= CAST(c.concept_id AS STRING)

```

Figure 4.37: SQL query with the help of dbt built-in function

We can run all the second transformation steps without worrying about dependencies with the help of data lineage as in figure 4.38 and figure 4.39 for table drug_era.

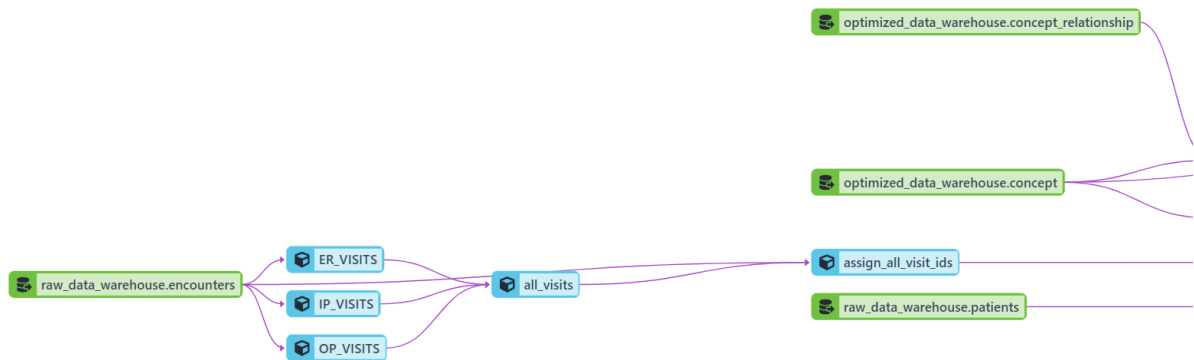


Figure 4.38: Data Lineage for for table drug_era part 1

And finally, we can have auto-generated documentation for our data in dbt (figure 4.40).

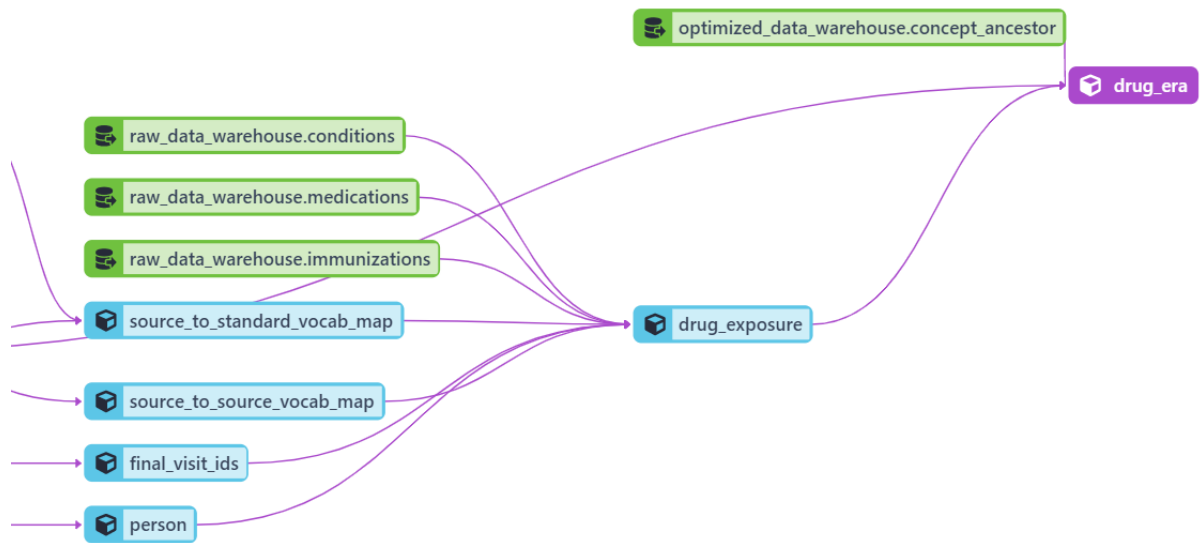


Figure 4.39: Data Lineage for for table drug_era part 2

Search for models...

drug_era table

[Details](#)
[Description](#)
[Columns](#)
[Depends On](#)
[SQL](#)

Details

TAGS	OWNER	TYPE	PACKAGE	RELATION
untagged		table	optimized_data_warehouse	bachelor-thesis-344103.optimized_data_warehouse.drug_era

ROWS

415,459

APPROXIMATE SIZE

23 MB

Description

This model is not currently documented

Columns

COLUMN	TYPE	DESCRIPTION
drug_era_id	INT64	
person_id	INT64	
drug_concept_id	STRING	
drug_era_start_date	DATE	
drug_era_end_date	DATE	
drug_exposure_count	INT64	

Lineage Graph

```

graph TD
    drug_exposure --> optimized_data_warehouse.concept_ancestor
    optimized_data_warehouse.concept_ancestor --> drug_era

```

Figure 4.40: dbt Auto-generated Documentation

5 Conclusion

This chapter summarizes the results and conclusion of the project. In addition, future work on data quality and data streaming is also mentioned.

5.1 Results

As for the result, The first outcome of the project is that we have successfully created a cloud data lake, which contains the compressed delta lake files. It is suitable for data scientists or users with programming languages to perform advanced analytics and machine learning compared to the raw data. We also built a data warehouse with BigQuery and OMOP CDM, which are optimized for analytics and flexible to collaborate with other medical organizations in the future. The second result is we built the data solution around the google cloud platform and modern data stack. It is scale-able for the increased amount of data and reduces the time that we need to manage. The data stack includes data orchestration, data visualizations, and data observation, which provide an automated ETLT data pipeline, a visualization platform, and a catalog application that can be accessed anytime and anywhere.

As for the author, he has acquired basic to advance knowledge in data warehouse, data lake, modern data applications and cloud platforms in general. He has learned about the ability to implement common data models and integrate the components of the data system. Besides, he has a certain understanding of how to write research and present the result to other people.

5.2 Future Work

The first idea for future work is adding data quality applications. Data quality is the measurement of how well a dataset satisfies its intended use. For example, data must be in a certain type, size, or range. It also has to run on a specific schedule or does not contain sensitive information.

There aren't too many data quality products in the market now, but `greate_expectaions` is absolutely the best choice for future deployment. It is the most popular data quality platform and the only application that can integrate well with all other apps in our thesis. With Great Expectations, we create our expectations from the data to catch data issues quickly and ensure data validity and the correctness of data after transformations on other steps. Finally, Great Expectations also creates data documentation and data quality reports from those Expectations so that we can have rich, shared documentation of the data, as in figure 5.1.

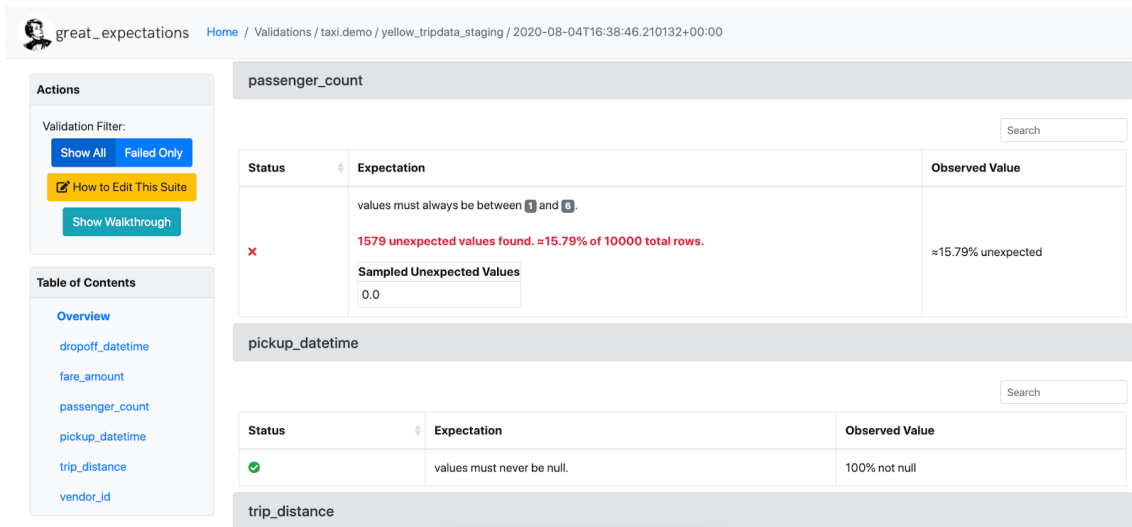


Figure 5.1: Greate Expectations Docs Example

They have published articles on how to integrate Great Expectations (GE) with only Google Cloud Platform (GCP) (not for AWS or Azure) [5] as in figure 5.2, integrate with Airflow and dbt [86], and only two weeks from the time writing this thesis, they announce the integration with Datahub, which is the only combination between open-source data quality and data catalog.

The second ideal is nowadays, patch processing is not enough for some specific scenarios. It is better to have a real-time data pipeline to catch up faster with what happens in real-time, especially in medical observations. Fortunately, the Google Cloud Platform also provides services for streaming data like Pub/Sub and Dataflow. It can be easily integrated with Bigquery, Cloud Storage, machine learning services, or hosted the machine (with our modern data stack) with little effort and simple configuration, as in figure 5.3.

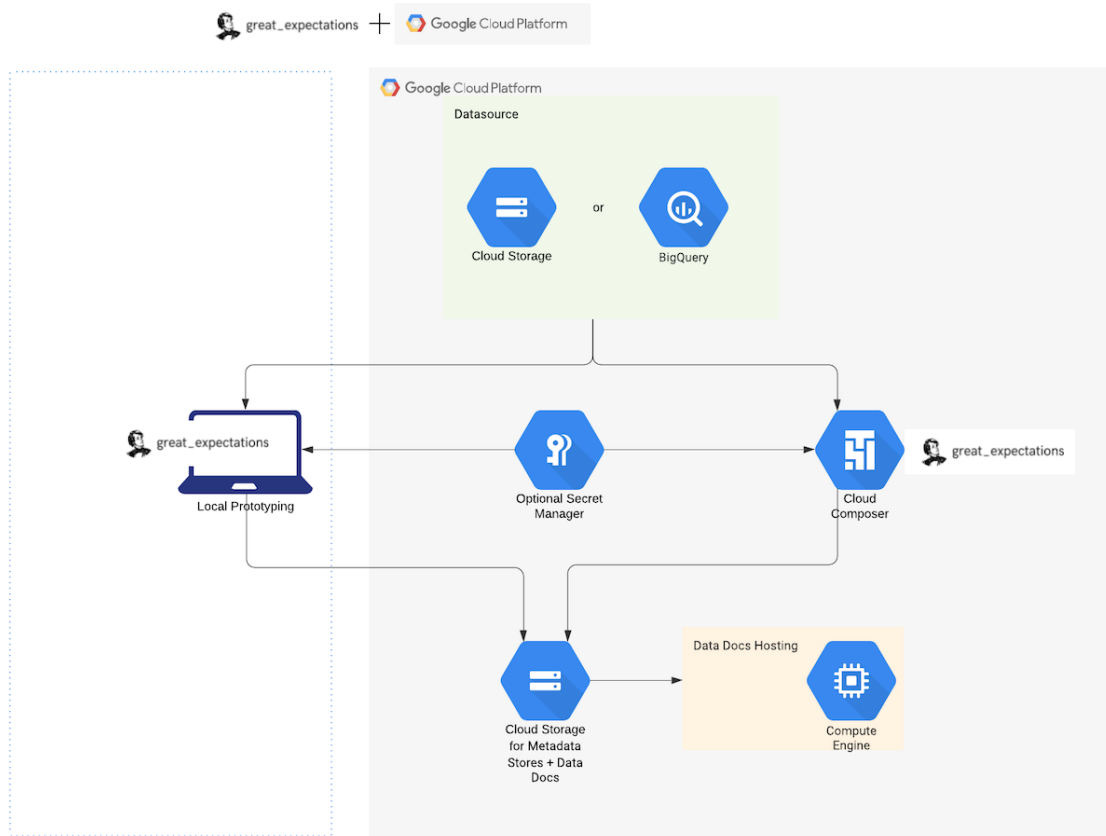


Figure 5.2: Great Expectations deployment in GCP [5]

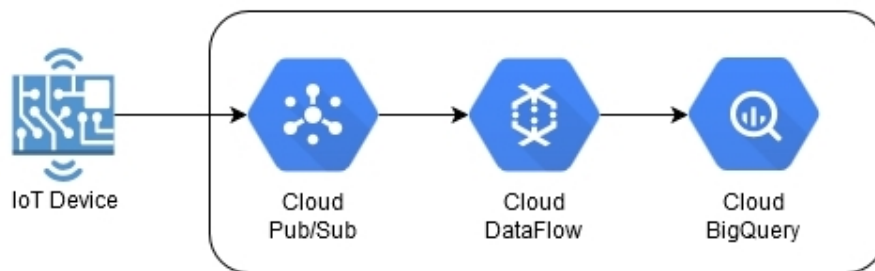


Figure 5.3: Streaming data with Dataflow pipeline[6]

5.3 Discussion and Conclusion

The exponential change in data technology today is making organizations more challenging, especially in traditional institutions like medical and healthcare. They are struggling to implement and manage systems to catch up and take advantage of all-new advanced technologies. This thesis has provided a comprehensive review of cloud data warehouses and data lakes and how data storage and processing are evolving. Besides, the author also introduces the data system and platform of Google Cloud Platform combined with the modern data stack and OMOP CDM in order to get insights of the newest technology in the data industry. Cloud providers can help health and medical take advantage of new technologies while ensuring security and privacy. OMOP CDM helps organizations collaborate with each other. Finally, the modern data stack can help to advance the data system since they are one of the best applications in the field now with the ability of easily integrate and interchange. They are created by the best in the field in all industries compared to specific tools for a project or certain fields like open-source tools to support OMOP CDM use cases provided by OHDSI. For example, Apache Superset and `greate_expectations` are absolutely better than Achilles (Data visualization) and Data Quality Dashboard from OHDSI. They have more contributors, which means better products and more people familiar with them, making them also easier for implementation and adoption. This implementation will hopefully be a general idea for everyone who wants to create a solution for improving the data platform, the data system as well as the data culture for healthcare and medical organizations in the future.

Bibliography

- [1] Amazon Redshift Documentation. Columnar storage. . URL https://docs.aws.amazon.com/redshift/latest/dg/c_columnar_storage_disk_mem_mgmt.html.
- [2] Malsha Ranawaka. Etl vs elt: The difference is in the how). 2021. URL <https://blog.panoply.io/etl-vs-elt-the-difference-is-in-the-how>.
- [3] Kasia Hoffman. Comparing the speed of vm creation and ssh access of cloud providers. 2016.
- [4] Advantages of using dbt(data build tool). 2020. URL <https://www.startdataengineering.com/post/advantages-of-using-dbt-data-build-tool/>.
- [5] Greate Expectations. How to use great expectations with google cloud platform and bigquery. . URL https://docs.greatexpectations.io/docs/deployment_patterns/how_to_use_great_expectations_with_google_cloud_platform_and_bigquery.
- [6] Legorie Rajan PS Ted Hunter, Steven Porter. *Building Google Cloud Platform Solutions*. Packt, 2019.
- [7] Pete Soderling. What are the most popular oss data projects of 2021? 2021. URL <https://petesoder.medium.com/what-are-the-most-popular-oss-data-projects-of-2021-84ef021bb5a2>.
- [8] OHDSI Observational Health Data Sciences and Informatics. Omop common data model. URL <https://www.ohdsi.org/data-standardization/the-common-data-model/>.
- [9] Z. Krpić V. Kluk, D. C. Milić. *Comparison of Data Warehousing and Big Data Principles from an Economic and Technical Standpoint and Their Applicability to Natural Gas Remote Readout Systems*. Preliminary Communication, 2018.
- [10] Paulraj Ponniah. *Data Warehousing Fundamentals: A Comprehensive Guide for IT Professionals*. John Wiley Sons, 2002.
- [11] H.P Luhn. A business intelligence system. *IBM Journal of Research and Development*, pages 314–319, 1958.
- [12] Barry A. Devlin and Paul T. Murphy. An architecture for a business and information system. *IBM systems Journal*, 27(1):60–80, 1988.
- [13] Bill Inmon. *Building the Data Warehouse*. John Wiley Sons, 2005.

- [14] Margy Ross Ralph Kimball. *The Data Warehouse Toolkit*. Kimball Group, 2013.
- [15] Golfarelli and Rizzi. *Data Warehouse Design: Modern Principles and Methodologies*. McGraw Hill Professional, 2009.
- [16] P. Lane. *Oracle Database Data Warehousing Guide*. Redwood: Oracle Corporation., 2005.
- [17] E. F. Codd. A relational model of data for large shared data banks. pages 377—387, 1970.
- [18] Normalization of database. URL <https://www.studytonight.com/dbms/database-normalization.php>.
- [19] Joao Sousa. The data processing holy grail? row vs. columnar databases. 2020. URL <https://dzone.com/articles/the-data-processing-holy-grail-row-vs-columnar-dat>.
- [20] John Schulz. Why columns stores. 2018. URL <https://blog.pythian.com/why-column-stores/>.
- [21] M. Breslin. Data warehousing battle of the giants : Comparing the basics of the kimball and inmon models. 2004.
- [22] Margy Ross Ralph Kimball. *The Kimball Group Reader: Relentlessly Practical Tools for Data Warehousing and Business Intelligence*. Kimball Group, 2010.
- [23] Ralph Kimball Joe Caserta. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. Kimball Group, 2011.
- [24] Managing input data and data sources-denormalizing data. . URL https://cloud.google.com/bigquery/docs/best-practices-performance-input#denormalizing_data.
- [25] Sukhmani Bains. Snowflake vs star schema. 2016.
- [26] Dave Fowler. *Cloud Data Management (4 Stages for Informed Companies)*. The Data School, 2019.
- [27] David Taylor. Data warehousing. URL <https://www.guru99.com/data-warehousing.html>.
- [28] Fabio Braga. Between warehouses, lakes and lakehouses: there is no free lunch. 2021. URL <https://www.linkedin.com/pulse/between-warehouses-lakes-lakehouses-free-lunch-fabio-braga/>.
- [29] EMC Education Services. *Data science and big data analytics: discovering, analyzing, visualizing and presenting data*. Wiley, 2015.

- [30] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 15–28, 2012.
- [31] Michael J Denney, Dustin M Long, Matthew G Armistead, Jamie L Anderson, and Baqiyyah N Conway. Validating the extract, transform, load process used to populate a large clinical research database. *International journal of medical informatics*, 94:271–274, 2016.
- [32] Shirley Zhao. What is etl? (extract, transform, load). 2017. URL <https://www.edq.com/blog/what-is-etl-extract-transform-load/>.
- [33] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. (800-145), 2011.
- [34] Kemper H. Baars, H. Business intelligence in the cloud. *PACIS 2010 Proceeding*, pages 1528–1539, 2010.
- [35] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.
- [36] Top 5 tech giants in the world. 2022. URL <https://www.geeksforgeeks.org/top-5-tech-giants-in-the-world/>.
- [37] Kate Park. Chinese crackdown on tech giants threatens its cloud market growth. 2021. URL <https://techcrunch.com/2021/09/13/chinese-crackdown-on-tech-giants-threatens-its-cloud-market-growth/>.
- [38] A. V. Dastjerdi R. Buyya, R. N. Calheiros. *Big Data - Principles and Paradigms*. Elsevier, 2016.
- [39] Krish Krishnan. *Data warehousing in the age of big data*. Newnes, 2013.
- [40] The Continual Team. The modern data stack ecosystem - fall 2021 edition. 2021.
- [41] Tristan Handy. The modern data stack: Past, present, and future. 2020. URL <https://blog.getdbt.com/future-of-the-modern-data-stack/>.
- [42] Amazon. Amazon s3 - 15 launches that defined 15 years of storage innovation.
- [43] Global cloud services market q1 2021. 2021. URL <https://www.canalys.com/newsroom/global-cloud-market-Q121>.
- [44] Aws vs azure vs google cloud: Choosing the right cloud platform. 2021. URL <https://intellipaas.com/blog/aws-vs-azure-vs-google-cloud/>.
- [45] Abe Dearmer. What is etlt? merging the best of etl and elt into a single etlt data integration strategy. 2020. URL <https://www.integrate.io/blog/what-is-etlt/>.

- [46] OHDSI Observational Health Data Sciences and Informatics. *The Book of OHDSI*. 2021.
- [47] Common data model and extract, transform load tutorial. 2016. URL <https://www.ohdsi.org/common-data-model-and-extract-transform-and-load-tutorial/>.
- [48] Mark Litwintschik. All 1.1 billion taxi rides on redshift. 2016. URL <https://tech.marksblogg.com/all-billion-nyc-taxi-rides-redshift.html>.
- [49] Mark Litwintschik. A billion taxi rides on google’s bigquery. 2016. URL <https://tech.marksblogg.com/billion-nyc-taxi-rides-bigquery.html#benchmarking-bigquery>.
- [50] Donal Tobin. Redshift vs bigquery. 2022. URL <https://www.integrate.io/blog/redshift-vs-bigquery-comprehensive-guide/>.
- [51] Kazunori Sato. An inside look at google bigquery.
- [52] Reynold Xin. Apache spark officially sets a new record in large-scale sorting. URL <https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.htm>.
- [53] Apache Spark. *PySpark Documentation*. URL <https://spark.apache.org/docs/latest/api/python/>.
- [54] Christopher Crosbie Roderick Yao. Getting started with new table formats on dataproc. 2020. URL <https://cloud.google.com/blog/products/data-analytics/getting-started-with-new-table-formats-on-dataproc>.
- [55] talend. Beginner’s guide to batch processing. URL <https://www.talend.com/resources/batch-processing/>.
- [56] Donal Tobin. Bi tools pricing and capabilities – analysis of legacy modern business intelligence tools. 2021.
- [57] Frederic Lardinois. Databricks acquires redash, a visualizations service for data scientists. 2020. URL <https://techcrunch.com/2020/06/24/databricks-acquires-redash-a-visualizations-service-for-data-scientists/>.
- [58] Shalaka Kulkarni. Superset vs metabase vs redash – comparing open source bi tools. 2017. URL <https://hevodata.com/blog/superset-vs-metabase-vs-redash/>.
- [59] Srini Kadamati. Apache superset vs metabase. 2022. URL <https://preset.io/blog/superset-vs-metabase/>.
- [60] Dmitriy Ryaboy. Announcing parquet 1.0: Columnar storage for hadoop. 2013. URL https://blog.twitter.com/engineering/en_us/a/2013/announcing-parquet-10-columnar-storage-for-hadoop.

- [61] Justin Kestelyn. Introducing parquet: Efficient columnar storage for apache hadoop. 2013. URL <https://web.archive.org/web/20130504133255/http://blog.cloudera.com/blog/2013/03/introducing-parquet-columnar-storage-for-apache-hadoop/>.
- [62] Reading delta lakes into dask dataframes. 2021. URL <https://mungingdata.com/dask/read-delta-lake/>.
- [63] Why is data security important? IBM. URL <https://www.ibm.com/topics/data-security>.
- [64] Google Cloud Documentation. Security and privacy considerations. . URL <https://cloud.google.com/storage/docs/gsutil/addlhelp/SecurityandPrivacyConsiderations>.
- [65] Andrew Burt. What do data scientists and data engineers need to know about gdpr? 2018. URL <https://www.infoq.com/articles/effective-data-management-age-GDPR/>.
- [66] Microsoft Azure. Azure compliance. URL <https://azure.microsoft.com/en-us/overview/trusted-cloud/compliance/>.
- [67] SyntheaTM Novel coronavirus (COVID-19) model and synthetic data set. Intelligence-Based Medicine. SyntheaTM novel coronavirus (covid-19) model and synthetic data set. intelligence-based medicine. 2020. URL <https://doi.org/10.1016/j.ibmed.2020.100007>.
- [68] Omop common data model v5.4. URL <https://ohdsi.github.io/CommonDataModel/>.
- [69] Aws identity and access management documentation. . URL <https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html>.
- [70] Amazon simple storage service user guide. . URL <https://docs.aws.amazon.com/AmazonS3/latest/userguide/GetStartedWithS3.html>.
- [71] Google cloud identity and access management documentation. . URL <https://cloud.google.com/iam/docs/>.
- [72] Google cloud bigquery documentation. URL <https://cloud.google.com/bigquery/docs>.
- [73] Cloud storage documentation. URL <https://cloud.google.com/storage/docs>.
- [74] Delta lakes documentation. URL <https://docs.delta.io/latest/delta-intro.html>.
- [75] Google cloud dataproc documentation. . URL <https://cloud.google.com/dataproc/docs>.
- [76] Google cloud compute engine documentation. URL <https://cloud.google.com/compute/docs>.

- [77] Prakhar Srivastav. A docker tutorial for beginners. 2021.
URL https://docker-curriculum.com/?fbclid=IwAR1iGBgLgy8nsp_uEq2ZKbSfvFGJUAibrBA0kf0aY3iVJgy6g6IYb2pnCHo#introduction.
- [78] Apache Airflow. Running airflow in docker. URL <https://airflow.apache.org/docs/apache-airflow/stable/start/docker.html>.
- [79] Google cloud composer documentation. URL <https://cloud.google.com/composer/docs>.
- [80] dbt cloud documentation. URL <https://docs.getdbt.com/docs/dbt-cloud/cloud-overview>.
- [81] Installing superset locally using docker compose. URL <https://superset.apache.org/docs/installation/installing-superset-using-docker-compose/>.
- [82] Datahub quickstart guide. . URL <https://datahubproject.io/docs/quickstart/>.
- [83] Github final bachelor thesis repository. URL <https://github.com/ledinhtrunghieu/bachelor-thesis>.
- [84] Google Cloud. Creating and managing service account keys. URL <https://cloud.google.com/iam/docs/service-accounts>.
- [85] Getting to know pyspark. . URL <https://goodboychan.github.io/python/datacamp/pyspark/2020/08/07/01-Getting-to-know-PySpark.html>.
- [86] Greate Expectations. dbt coalesce 2020: Building a robust data pipeline with dbt, airflow, and great expectations. . URL <https://greatexpectations.io/blog/coalesce2020/>.