SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Ensuring High Data Quality Standards: A Framework for Single and Cross-Enterprise Platforms

Tesi di Laurea Magistrale in
Computer Engineering - Ingegneria Informatica

Author: **Leonardo Mandruzzato**

Student ID: 969958
Advisor: Prof. Marco Brambilla
Academic Year: 2022-23

Abstract

This Master's Thesis presents a comprehensive framework to enhance data quality in multi-source analytics platforms across various organizational contexts. The study emphasizes the significance of data quality in today's data-driven business landscape, where data accuracy, reliability, completeness, and consistency are crucial for informed decision-making. The framework addresses the "Garbage In, Garbage Out"" (GIGO) issue, underscoring the direct link between input data quality and the reliability of subsequent analysis. It introduces three technological solutions tailored to different data producer-consumer relationships: a *Data Quality Assessment* (DQA) system, which is the atomic unit of the framework and applies to any scenario, *Data Contracts* (DC) for intra-organizational contexts, and a *Push API* (PAPI) for cross-enterprise collaborations. Practical implementations in various real-world contexts demonstrate the effectiveness of these solutions in addressing data quality challenges. The thesis also acknowledges the limitations of the proposed framework. Indeed, it suggests potential areas for future research, including expanding the framework to cover a broader range of collaborative scenarios and complex multi-tenant relationships. This research contributes to data management and analytics by providing a decision-making tool for organizations to select the most suitable technological solutions based on their specific needs, thereby helping them maximize their data asset value.

Keywords: Data Quality, Data Platform, Data Management, Data Governance, Data Contracts, API

Abstract in lingua italiana

Questa Tesi Magistrale presenta un quadro completo per migliorare la qualità dei dati nelle piattaforme analitiche multi-sorgente in differenti contesti organizzativi. Lo studio sottolinea l'importanza della qualità dei dati nell'attuale panorama aziendale guidato dai dati, in cui la loro accuratezza, affidabilità, completezza e coerenza risulta fondamentale per prendere decisioni informate. Il framework affronta il problema del "Garbage In, Garbage Out" (GIGO), sottolineando il legame diretto tra la qualità dei dati in ingresso e l'affidabilità delle analisi successive. Introduce tre soluzioni tecnologiche adatte alle diverse relazioni produttore-consumatore: un sistema di *Data Quality Assessment* (DQA), che è l'unità atomica del framework e si applica a qualsiasi scenario, i *Data Contracts* (DC) per i contesti intra-organizzativi e una *Push API* (PAPI) per le collaborazioni tra aziende. Le implementazioni pratiche in vari contesti del mondo reale dimostrano l'efficacia di queste soluzioni nell'affrontare le sfide della qualità dei dati. La tesi riconosce anche i limiti del framework proposto. Infatti, suggerisce potenziali aree di ricerca future, tra cui l'espansione del framework per coprire una gamma più ampia di scenari collaborativi e relazioni complesse multi-consumatore. Questa ricerca contribuisce alla gestione e all'analisi dei dati fornendo uno strumento decisionale alle organizzazioni per selezionare le soluzioni tecnologiche più adatte in base alle loro esigenze specifiche, aiutandole così a massimizzare il valore del loro patrimonio di dati.

Parole chiave: Qualità dei Dati, Piattaforma Dati, Gestione dei Dati, Governance dei Dati, Contratti sui Dati, API

Contents

Abstract **i**

Abstract in lingua italiana **iii**

Contents **v**

Introduction **1**

1 Background **7**
1.1 Data Engineering Lifecycle 7
1.1.1 Generation 8
1.1.2 Storage 13
1.1.3 Ingestion 16
1.1.4 Transformation 18
1.1.5 Serving Data 20
1.2 Undercurrents 21
1.2.1 Data Management 21
1.3 Data Quality 22
1.3.1 Dimensions 22
1.3.2 Relevance 23

2 Related Works **25**
2.1 Data Contracts 25
2.2 API 27
2.3 DQ Platforms and Tools 28
2.3.1 Monte Carlo 29
2.3.2 DBT Tests 29
2.3.3 Great Expectations 29

3 Problem Statement **31**

3.1 Manifestation 31
3.1.1 Quality Dimensions 32
3.1.2 Schema Compatibility 33
3.2 Root 34
3.3 Proposed Solution 36

4 Framework and Implementation 39
4.1 Data Quality Assessment Solution 40
4.1.1 Context 41
4.1.2 Tooling 42
4.1.3 System and Workflow 42
4.2 Data Contracts Solution 52
4.2.1 Context 54
4.2.2 Tooling 54
4.2.3 System and Workflow 55
4.3 Push API Solution 63
4.3.1 Context 63
4.3.2 Tooling 64
4.3.3 System and Workflow 65

5 Results Discussion 69
5.1 Proactive VS. Reactive 69
5.2 Prevention VS. Detection 70
5.3 Single- VS. Cross-enterprise 70
5.4 Comprehensive Framework 71

6 Conclusions 75
6.1 Limitations 75
6.2 Future Work 76

Bibliography 79

List of Figures 83

List of Tables 85

List of Acronyms 87

Introduction

Data has emerged as one of the most invaluable resources in today's digital era, revolutionizing industries and shaping how organizations operate. The vast amount of data generated by various sources offers immense potential for organizations to gain valuable insights, make informed decisions, and achieve strategic goals. However, to effectively leverage this potential, data must possess high-quality standards. Data Quality (DQ) is crucial in ensuring information accuracy, reliability, completeness, and consistency, enabling organizations to derive meaningful insights and make confident business decisions. In a data architecture—where data serves as the foundation for computing metrics, reporting, and decision-making—the need for high DQ standards becomes even more paramount.

This thesis sets the stage for understanding the importance of data quality and introduces a framework that aims to achieve and maintain it in single and cross-enterprise systems. By addressing the challenges and providing practical solutions, this framework contributes to data management and analytics, ultimately assisting organizations in maximizing the value of their data assets.

Context

In today's increasingly data-driven business landscape, nearly every company recognizes the value of data as a strategic asset. As a result, organizations invest in developing robust data architectures. *Data architecture* refers to the "design of systems to support the evolving data needs of an enterprise, achieved by flexible and reversible decisions reached through a careful evaluation of trade-offs" [18]. It represents a comprehensive infrastructure enabling data collection, storage, processing, analysis, and visualization to derive meaningful insights and support decision-making processes. It serves as a centralized hub where diverse data sources converge, facilitating the transformation of raw data into actionable knowledge.

Three key participants play vital roles within a data platform: data producers, data custodians, and data consumers [29]. Data producers are entities or systems responsible for

generating and supplying data to the platform. They could include various sources such as sensors, databases, applications, or external data providers; software engineers usually sit behind these entities. On the other hand, data custodians provide and manage computing resources for storing and processing data. Finally, data consumers are the stakeholders who utilize the data within the platform to gain insights, drive decision-making, and support business operations. These consumers may span different organizational functions, such as data analysts, business intelligence teams, or executives responsible for strategic planning.

Data quality is a fundamental aspect that underpins the effectiveness and reliability of an analytics platform (even if they do not refer to the exact same thing, this thesis uses the terms “data platform,” “analytics platform,” “data architecture,” and “ETL platform” interchangeably). It refers to the degree to which data meets specific criteria or standards, including accuracy, completeness, consistency, and time-related dimensions such as currency, timeliness, and volatility [2]. High DQ ensures that the information derived from the data platform is trustworthy, enabling confident decision-making and reliable analysis. However, achieving and maintaining data quality can be complex, particularly in environments with multiple participants. Each participant may introduce their sources, formats, and integration processes, potentially leading to inconsistencies, errors, or discrepancies that undermine the overall DQ.

Problem and Motivation

Organizations recognize the “Garbage In, Garbage Out” (GIGO) concept as a fundamental challenge to achieve data quality. Evidence of this problem—not necessarily applied to DQ—exists as early as the 1860s. However, the first time the term “GIGO” was printed on paper seems to be in 1962 by an IBM programmer named George Fuechsel [28]. This concept highlights the critical relationship between the quality of input data and the resulting output or analysis. If data practitioners feed poor-quality data into a system or process, its outcomes and insights will inevitably be flawed or unreliable. The GIGO problem underscores the importance of ensuring high data quality standards to avoid propagating inaccuracies, biases, or misleading conclusions throughout the ETL pipeline.

In the context of a data platform, the GIGO problem further emphasizes the interdependence between data producers and consumers. Producers’ inaccurate, incomplete, and inconsistent data becomes the foundation for subsequent analyses and decision-making. Consequently, the outputs derived from the downstream pipelines, consumed by various stakeholders, including data analysts, business intelligence teams, and executives, will be

tainted by the inherent flaws in the input data.

In today's complex data ecosystems, data producers often need more visibility or concern for the downstream consequences of their work. They have no incentive to take ownership of the data quality they produce outside operational use cases. They primarily focus on their specific tasks or goals without considering the impact of their data on subsequent analysis or decision-making processes. Similarly, data consumers—who utilize the outputs of the analytics platform—often need more knowledge or control over the quality of the data sources upstream. Furthermore, they are often well-trained in working with technologies such as SQL and Python but have yet to develop a set of robust requirements to instruct the upstream workforce. This lack of awareness and communication between data producers and consumers exacerbates the GIGO problem, perpetuating a cycle of poor DQ and compromised insights [21].

Moreover, this problem magnifies when multiple entities are involved, for instance, in a multi-source system. In such scenarios, data may flow from various sources, including external providers, subsidiaries, or partner organizations. Each entity may have its own data production processes, standards, or quality control measures, resulting in significant heterogeneity and variability in the DQ fed into the analytics platform. As the number of entities and data sources increases, maintaining high data quality standards becomes even more daunting, as inconsistencies, discrepancies, or conflicts in the requirements can arise across the ecosystem.

Addressing the problem of maintaining high DQ standards in analytics platforms—particularly in multi-entity architectures—is essential to mitigate the GIGO problem and enable organizations to make reliable, data-driven decisions.

Solution and Contribution

This thesis proposes a comprehensive framework that provides tailored technological solutions based on the specific context and relationships between data producers and consumers to address the outlined challenges and achieve and maintain high data quality standards in data platforms. The framework offers distinct solutions for different scenarios, considering the organizational context.

Firstly, the *Data Quality Assessment* DQA system applies to any scenario as it does not require a specific relationship between data producers and consumers; this is the foundational element of the comprehensive framework, and a data team can set it up with no external contribution. Data engineers and architects configure the platform to

proactively look for existing DQ issues. Indeed—in an ideal world—it would always be possible to determine whether or not each piece of data in the platform is correct and, if not, figure out where it came from and what caused the error [15]. Therefore, this framework aims to detect non-met expectations and notify the engineers and consumers, ensuring they are immediately aware of any data quality concerns and can take appropriate actions. It serves as a foundation for maintaining a baseline level of data quality across the data platform. It is important to note that data practitioners should implement the DQA framework across all the scenarios discussed later in this thesis, as it functions as the atomic unit of the overall framework this work aims to present.

The proposed framework offers more advanced solutions that allow for a shift from a data quality detection approach to a data quality prevention by building on top of the previously presented atomic unit. These solutions address the varying relationships between data producers and consumers in different scenarios, ensuring enhanced DQ practices.

In a single enterprise or organization, where internal data producers—e.g., software engineers—interact with internal data consumers—e.g., data analysts, the framework introduces the concept of data contracts. *data contracts* DC serve as agreements between data producers owning the services and data consumers understanding the business requirements to generate well-modeled and trusted data [21]. These contracts ensure clear communication and understanding between the two parties, facilitating the organization's production and consumption of high-quality data.

In the scenario where data producers from multiple enterprises or organizations interact with the data platform of a single central entity, the framework proposes the implementation of a *Push API* PAPI. This Push API enables producers to send data to the central entity following specified formats and requirements. By standardizing the data ingestion process, the Push API ensures high-quality data ingestion into the analytics platform. This solution streamlines the integration of external data into the central entity's ETL platform, facilitating reliable and efficient data consumption. However, the central entity can pursue this approach only if it has relevant bargaining power vis-à-vis the data sources to convince or force them to push the data regularly according to the pre-defined requirements. Other approaches should be adopted if the central entity is not in this position.

The thesis will present the framework and illustrate its design and practical implementation through diverse use cases, showcasing its efficacy in addressing data quality challenges in various real-world contexts.

Structure of the Thesis

The thesis entails several chapters exploring the proposed framework's different aspects. The "Background" chapter (Chapter 1) sets the foundation by discussing the Data Engineering Lifecycle, Undercurrents, and Data Quality. This chapter briefly presents the state of the art of certain aspects of data engineering and provides a comprehensive understanding of the key concepts and factors influencing DQ practices.

Moving forward, Chapter 2—"Related Works"—explores the theoretical underpinnings and practical tools that underlie the very foundation of our proposed framework. This chapter serves as a bridge, connecting theoretical principles to tangible implementations that culminate in our ultimate objectives.

Chapter 3—the "Problem Statement and Proposed Solution"—emerges as a focal point, delving into the nuanced challenges when the interplay between data producers and consumers needs more optimization. This exploration casts a spotlight on the precise pain points that our framework seeks to address.

With a shift towards practicality—the "Framework and Implementation"—chapter (Chapter 4) focuses on the system design and practical implementation of the technological solutions that will be part of the final framework. It thoroughly examines each technological solution part of the framework, showcasing its functionalities, benefits, and effectiveness in enhancing data quality in the respective scenarios.

The "Result Discussion"—in Chapter 5—presents the final framework as a flowchart while spotlighting the transformative contributions made by our approach. A comprehensive dialogue ensues, engaging with the implications of our work.

Finally, Chapter 6—the "Conclusion"—serves as a comprehensive wrap-up of the thesis. It provides a summary of the key findings. Additionally, it includes a critical discussion that examines the strengths and limitations of the research and suggests potential areas for future work, indicating avenues for further exploration and improvement in the realm of DQ in data platforms.

1 | Background

This chapter serves as an introduction to the technical background of this thesis. It presents the key concepts and undercurrents that form the foundation of the data engineering lifecycle, with a particular focus on data management and governance, leading us to the central theme of data quality (Figure 1.1).

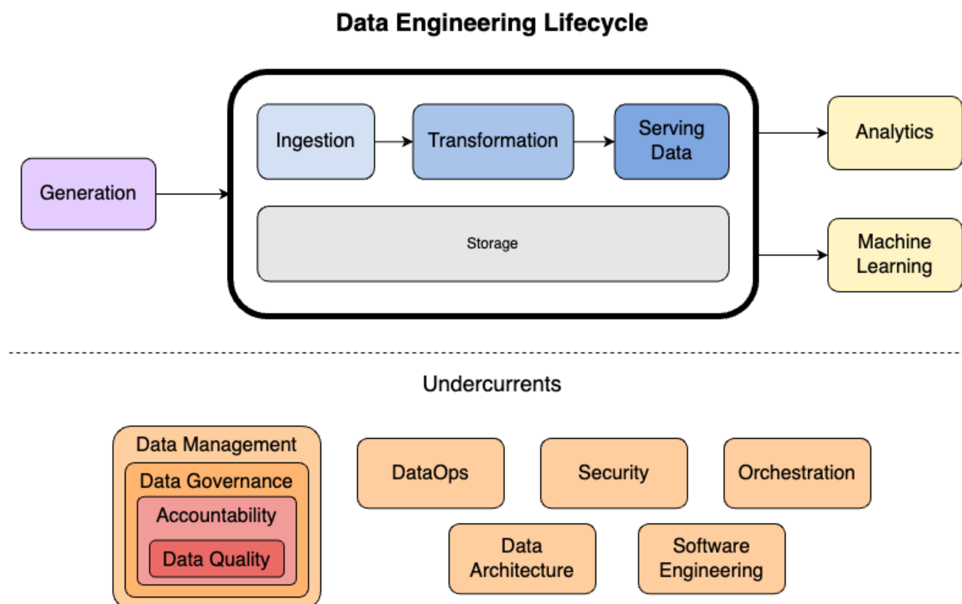

Figure 1.1: Key concepts and undercurrents that form the data engineering lifecycle [18].

1.1. Data Engineering Lifecycle

As Joe Reis and Matt Housley write in the book *Fundamentals of Data Engineering*, "The data engineering lifecycle comprises stages that turn raw data ingredients into a useful end product, ready for consumption by analysts, data scientists, ML engineers, and others" [18]. In other words, the *data engineering lifecycle* encompasses the end-to-end processes involved in managing data within an organization.

Experts break down the whole lifecycle into five stages. It begins with generating data from various sources and the consequent ingestion into the data platform, progresses

through the transformation and culminates in serving data for analysis and decision-making; the storage stage undercuts the entire lifecycle and hence supports all the previously mentioned phases (Figure 1.1). Understanding this lifecycle is essential for comprehending the challenges and considerations associated with data architectures and quality.

1.1.1. Generation

The *generation stage* is the first stage in the data engineering lifecycle. It involves everything to do with the entities feeding data into the platform, referred to as "data sources" or "providers" in this thesis. There are three different perspectives from which to look at these:

1. Semantic perspective - what or who they are;
2. Technological perspective - what form they assume;
3. Practical perspective - what they return.

Semantic Perspective

Semantically, it is possible to divide data providers into two main categories: service and data stakeholders. Data engineers interact with both categories daily (Figure 1.2). *Service stakeholders* are individuals or teams that implement and maintain the source systems. Typically, these groups include software engineers, application developers, and occasionally third-party organizations in charge of hosting or providing particular services. On the other hand, *data stakeholders*—IT teams, data governance groups, or third-party entities—are responsible for owning and controlling access to the data. Though they may overlap, service and data stakeholders are often separate. [18]

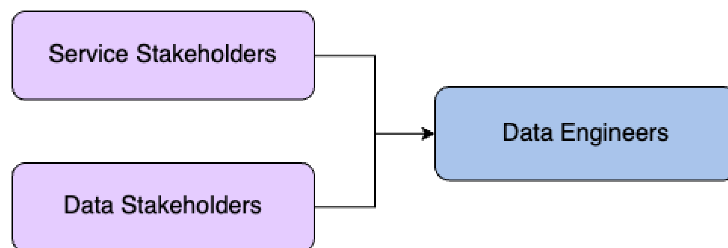

Figure 1.2: Data Engineer's upstream stakeholders.

Technological Perspective

Technologically, sources assume different forms. Some are listed in Table 1.1 and detailed in this subsection.

File or Object Storage Systems

File or object storage systems are fundamental data sources in a data engineering lifecycle. These systems provide a convenient way to store and organize vast amounts of structured and unstructured data. Standard file storage systems include local file servers, network-attached storage (NAS), and cloud-based object storage services like Amazon S3 or Google Cloud Storage. Data engineers often leverage file formats like CSV, JSON, Parquet, Avro, or Excel/Google Spreadsheet to store data in these systems. The advantage of using file or object storage lies in its simplicity and flexibility, making it suitable for small-scale and large-scale data extraction processes. It is possible to set up data pipelines to periodically read data from these storage systems, making it a reliable choice for various data-driven applications.

OLTP Database

Applications produce operational data and store it in Online Transaction Processing (OLTP) databases.

> "Online transaction processing systems are purpose-built to handle large numbers of transactions by large numbers of concurrent users while guaranteeing the integrity of the data" [11].

These databases are specifically optimized to handle high-frequency, real-time transactional data and are typically the primary storage choice during the development of service by software engineers.

The nature of the data stored in OLTP databases can exhibit distinct characteristics based on the employed database management system. Specifically—if the OLTP database follows the relational model—the data is typically structured and organized into well-defined tables with predefined schemas and adheres to a strict data model. This structured approach facilitates efficient querying and enforces data integrity constraints, ensuring consistent and accurate data management.

Conversely, when the underlying OLTP database adopts a NoSQL (Not Only SQL) paradigm, the stored data may diverge from the structured nature observed in relational databases. Instead, NoSQL databases allow for greater flexibility, accommodating semi-structured or unstructured data formats. This relaxed schema approach in NoSQL databases enables data storage without the constraints of predefined schemas, allowing for easy adaptation to evolving data requirements and diverse data types. It is particularly well-suited for scenarios where the data schema is subject to frequent changes or if the data originates from various sources with differing structures.

API

Another approach to sharing data between systems is via Application Programming Interfaces (APIs). APIs define the protocols and conventions that enable seamless interaction and data exchange between diverse systems. They serve as intermediaries, allowing applications to access and utilize each other's functionalities, services, and data. Notable API standards include REST, SOAP, WebSockets, GraphQL, RPC, and gRPC, each serving distinct use cases.

Representational State Transfer (REST) has emerged as the most widely adopted protocol for designing and implementing APIs in the contemporary software development landscape. RESTful APIs leverage the principles of simplicity, scalability, and statelessness to ensure efficient communication and interoperability between systems. This thesis acknowledges the prevalence and significance of REST in modern data integration practices and will consistently consider REST APIs whenever referring to API-related concepts.

It is common to have services producing data and sharing it via API. Usually, the form of the output data depends on the API standard in use. The payloads they transmit are often in semi-structured formats like JSON or XML.

Message Queue and Streaming Platform

In event-driven architectures, message queues and streaming platforms are two relevant concepts. A message queue is a fundamental communication mechanism in distributed systems to enable asynchronous and decoupled data exchange between components or services. It acts as an intermediary buffer that temporarily stores messages until the intended recipients consume them. Messages within a message queue typically follow a first-in-first-out (FIFO) order, ensuring the processing happens in the order of arrival. The underlying data structure of a message queue revolves around messages, which are packets of information containing meaningful data transmitted between system components. These messages are usually semi-structured in formats like JSON or XML, encapsulating relevant details, including payload, metadata, and destination information.

Streaming platforms—also known as data streaming or event streaming platforms—play a pivotal role in managing and processing continuous data streams in real-time. They empower organizations to handle vast amounts of data generated from diverse sources, such as applications, devices, or IoT sensors, while maintaining low-latency data processing capabilities. The core component of a streaming platform is a stream, which represents an unbounded sequence of events ordered chronologically. Each event within the stream

encapsulates specific data related to a particular occurrence, such as a user action, sensor reading, or system event.

Technology	Data Format
File System	Semi- and Structured Data
Object Storage System	Semi- and Structured Data
API	Semi-structured Data
Messages Queues and Streaming Platforms	Semistructured Data
OLTP database	Semi- and Structured Data

Table 1.1: Different types of sources and the data formats they produce.

Practical Perspective

Practically speaking, knowing the data structure produced by the sources and how the latter makes it available to third parties is necessary for a data engineer to design and develop the downstream system properly. This section details the different levels of structure the data can assume by listing their characteristics and the associated pros and cons. Section 1.1.3 will address how a data source provides access to a data asset and how the data architecture ingests it in the storage system.

Structured Data

Structured data refers to information organized in a predefined and rigid format, typically represented in tables with rows and columns. It adheres to a fixed schema, ensuring consistency and uniformity and facilitating efficient storage, retrieval, and analysis processes. The characteristics of structured data include ease of integration with relational databases, enabling seamless querying and reporting, and transparent data lineage and lineage tracking. However, the main drawback of structured data lies in its limited flexibility in accommodating new data sources or unexpected changes, often necessitating time-consuming schema modifications.

Examples of structured data include traditional database records and can encompass Spreadsheets and CSV files. These file formats are considered structured data due to their tabular organization and are used extensively in the business industry, especially regarding cross-enterprise data sharing. However, this prevalence can sometimes lead to data quality issues as Spreadsheets and CSV files lack strict data validation and integrity constraints, making them prone to errors and inconsistencies. Also, by not following

data best practices, an organization might accumulate tech debt and hinder efficient data analysis and decision-making processes. To avoid such pitfalls, implementing robust data governance policies, employing standardized data formats like JSON or XML, and utilizing proper version control mechanisms can significantly enhance DQ, reliability, and overall data management.

Semi-structured Data

Semi-structured data refers to data that does not conform to traditional relational databases' strict tabular structure yet exhibits some structure. It often contains tags, metadata, or hierarchical elements, allowing for some organization and flexibility in representing complex data. Examples of semi-structured data include XML files and JSON documents. The main characteristics of semi-structured data are its flexibility and schema evolution capability, which enables data engineers to adapt to changing data requirements without altering the entire data architecture. However, managing semi-structured data can be challenging due to the lack of standardized schemas, which can lead to potential data quality and integration issues.

Nowadays, semi-structured data production and consumption increased massively. Most of the new applications rely on this format because of its flexibility. Semi-structured is now used everywhere: systems store it in OLTP NoSQL databases, push it in message queues in the form of messages, save it in the form of files in object storage systems, share it in the form of streams in streaming platforms, and return it as an API payload upon a REST API request.

Unstructured Data

Unstructured data refers to information that lacks a predefined data model or organized format. Data does not adhere to a fixed schema in this context, making it challenging to interpret and analyze using traditional database management systems. Examples of unstructured data include free-form text documents, audio files, images, videos, social media posts, and email communications. Due to its inherent complexity, handling unstructured data requires specialized techniques, such as natural language processing (NLP), image recognition, and machine learning algorithms to extract valuable insights. Unstructured data poses significant DQ challenges, including the potential for inaccuracies, redundancy, and incompleteness, necessitating robust data cleaning and preprocessing strategies during the data engineering lifecycle.

1.1.2. Storage

In the second stage of the data engineering lifecycle—referred to as *storage*—the primary objective is establishing a robust and efficient foundation for housing data within the overarching data architecture. This pivotal phase involves various facets that collectively contribute to the seamless management and accessibility of data. Three fundamental tenets deserve attention when discussing storage.

Firstly, it is essential to recognize that the storage layer underpins the entire data engineering lifecycle. Its impact extends to at least three other stages: ingestion, transformation, and serving (Figure 1.1). Consequently, this stage serves as foundational support for most of the other steps in the lifecycle.

Secondly, separating storage and transformation stages in the lifecycle became feasible after Google introduced the principle of separating storage and computing [31]. This principle is paramount in data engineering due to its profound effects on scalability, flexibility, and cost-efficiency. It also mitigates the notorious *scalability dilemma* [27] wherein traditional monolithic architectures necessitate scaling both storage and compute resources together, often leading to overprovisioning and increased operational costs. Moreover, it empowers data engineers to adopt a polyglot approach, selecting specialized tools for each data processing task, optimizing performance, and facilitating the incorporation of new technologies emerging. Therefore, the segregation of storage from computing emerges as a cornerstone in modern data engineering paradigms, enhancing system adaptability, modularity, and overall data processing efficiency.

Finally, it is possible to classify storage components into a three-level taxonomy (Figure 1.3).

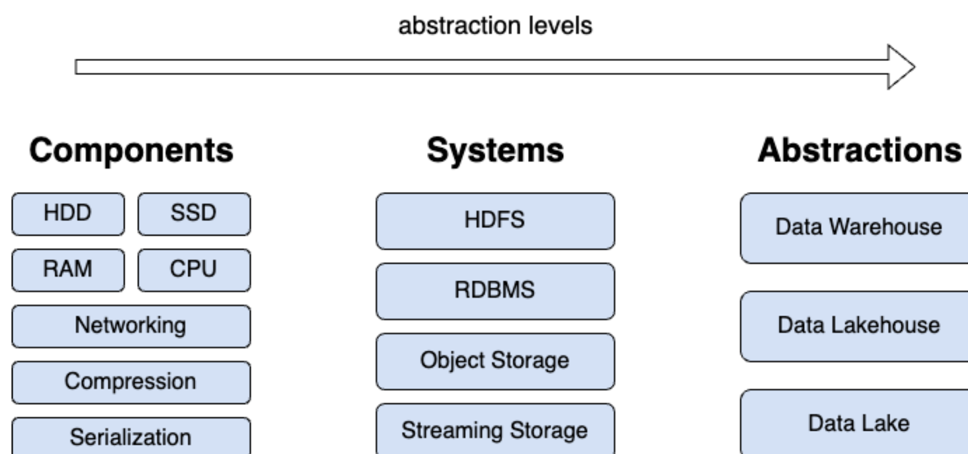

Figure 1.3: Technological components, storage systems, and storage abstractions [18].

At the lower and more concrete level, essential components such as HDD, SSD, RAM, and CPU play pivotal roles in data storage and retrieval processes. Additionally, serialization and compression techniques are crucial to optimizing data representation and minimizing storage footprints. Furthermore, engineers must consider networking aspects to facilitate smooth data transfer across distributed systems. By moving up the abstraction ladder, the second layer encompasses storage systems like HDFS, RDBMS, object storage, and streaming storage, which play a decisive role in accommodating diverse data requirements. At the highest level of abstraction, different architectural paradigms of data storage, namely data lake, data lakehouse, and data warehouse, each offer unique advantages in terms of data organization and accessibility. [18]

This section will focus solely on the abstract layer and provide further information about the architectural paradigms provided as examples.

Data Warehouse

Bill Inmon first formally defined the *data warehouse* concept as "a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management's decision-making process" [12]. The data warehouse paradigm is a well-established data storage and management approach that supports structured and historical data analysis. Characterized by a schema-on-write architecture, data warehouses enforce a predefined schema before data ingestion, ensuring data consistency and facilitating complex querying and reporting. Extract Transform Load (ETL) processes are typically employed to cleanse, transform, and load data from multiple sources into the warehouse. The structured nature of data warehousing enables optimized and efficient execution of analytical queries, making it a preferred choice for Business Intelligence (BI) and decision-making applications. However, the rigidity of schemas may pose challenges in accommodating diverse and rapidly changing data formats, limiting the data warehouse's suitability for handling semi-structured or unstructured data.

The Data Warehouses paradigm—established several decades ago—predominantly saw development within on-premises environments, particularly during their nascent stages. *Cloud data warehouses* [18] have recently surfaced, signifying a progressive evolution of their on-premises predecessors. Presently, the term "data warehouses" primarily denotes infrastructures hosted and functioning in the cloud. This transformation owes its realization to the introduction of Amazon Redshift, which served as the pioneering technology, paving the path for subsequent innovations like Snowflake and BigQuery [10].

Data Lake

The first time the term "data lake" appeared online was in the blog of James Dixon (Pentaho CTO), and he described it with the following words:

> "If you think of a datamart as a store of bottled water—cleansed and packaged and structured for easy consumption—the data lake is a large body of water in a more natural state. The contents of the data lake stream in from a source to fill the lake, and various users of the lake can come to examine, dive in, or take samples" [6].

He describes a new approach that should oppose the datamart paradigm (not described in this thesis) to revolutionize how data practitioners think. The *data lake* paradigm represents a more flexible and scalable approach to data storage, accommodating a vast array of data types, including structured, semi-structured, and unstructured data. By embracing a schema-on-read architecture, data lakes allow data ingestion in its raw form and schema applied during analysis. Flexible schemas grant data engineers and analysts greater freedom to explore and derive insights from diverse datasets without requiring extensive data transformation beforehand. Additionally, data lakes leverage distributed storage systems, enabling seamless horizontal scaling to handle massive volumes of data. However, the schema-on-read approach may challenge maintaining data consistency and quality, necessitating robust metadata management and data governance practices.

Data Lakehouse

The *data lakehouse* paradigm emerges as a hybrid solution, combining the best elements of data warehouse and data lake approaches. Michael Armbrust et al.—the authors of the "LakeHouse" whitepaper—initially considered this pattern "characterized by open direct-access data formats, such as Apache Parquet and ORC, first-class support for machine learning and data science workloads, and state-of-the-art performance" [34]. This innovative approach aims to bridge the gap between structured and unstructured data by introducing an additional layer of organization and governance on top of the data lake. Data ingestion happens in the raw form, similar to data lakes, allowing for schema flexibility and easy data integration from various sources. However, a converged processing model is adopted in the data lakehouse, allowing engineers to apply transformations and schema enforcement during data ingestion or query time. This convergence enables real-time and batch processing capabilities, empowering organizations to handle traditional analytics and more dynamic, event-driven use cases. By blending the strengths of data warehousing and data lakes, the data lakehouse paradigm seeks to deliver a unified plat-

form that addresses the challenges of data silos, data quality, and data consistency often encountered in traditional architectures.

1.1.3. Ingestion

The *ingestion* stage holds significant importance in achieving the objectives of this thesis. It centers around how sources share the produced data with consumers (Figure 1.4). This stage focuses on the interaction between producers and consumers. Hence, it intrinsically impacts the success of the proposed framework, which focuses on single- and multi-source systems and mono- or cross-enterprise architectures.

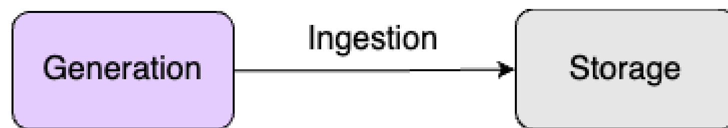

Figure 1.4: Role of the ingestion stage within the whole data engineering lifecycle.

The ingestion stage involves two primary participants: the data source and the data consumer. Data sources can vary in number and affiliation, belonging to the same organization as the consumers or different organizations engaged in collaborative data-sharing agreements. It is important to note that these data providers primarily aim to operate platforms that provide products and services to the organization's end users rather than producing and sharing data specifically for analytics.

In recent times, the utilization of data has transcended its original operational purpose. The emergence of the data-driven approach has highlighted the previously untapped potential of this data, encouraging its exploration for extracting valuable insights. As a result, the realm of data consumers has expanded beyond just services and applications to encompass analytics. This Master's Thesis focuses on analytics as a critical subject. In this context, the analytics platform becomes the primary stakeholder for data consumption. Under these assumptions, the data produced by sources ingestion into a comprehensive data architecture with the ultimate aim of transformation, enabling analytics, or integration into machine learning models.

Effective and transparent communication between the teams responsible for the two components mentioned above—the data sources and the data architecture—emerges as a crucial aspect of the overall data ingestion process. The successful implementation of the proposed framework, which aims to address data quality challenges for both single and cross-enterprise systems, relies on seamless collaboration and effective diplomacy between these entities.

There are different ways data sources share data with an analytics platform. The most common are file sharing, database connection, change data capture, APIs, message queues, and event-streaming platforms.

File Sharing

File sharing is a prevalent data exchange method between data sources and analytics platforms. In this approach, data producers generate files containing structured, semi- or unstructured data and distribute them to the analytics platform. These files could be in various formats, such as CSV, JSON, XML, or Parquet. The analytics platform then ingests these files, processes their contents, and transforms them into insights through analysis. While file sharing offers simplicity and compatibility, it might pose challenges related to real-time data synchronization, quality, and consistency, mainly when dealing with high-frequency updates.

Database Connection

Establishing a direct database connection is widely used for sharing data between sources and analytics platforms. Data sources grant access to their databases, allowing the analytics platform to query and retrieve the required data in real-time or in batches. This approach offers advantages regarding up-to-date information and query flexibility, enabling complex data transformations and aggregations. However, it requires careful management of access permissions and security protocols to ensure data integrity and prevent unauthorized access.

Change Data Capture (CDC)

Change Data Capture CDC is a technique that focuses on capturing and propagating changes made to data sources in near real-time. CDC systems monitor source databases for any modifications and then replicate these changes to the analytics platform, ensuring it always contains the latest updated data. This approach is particularly beneficial for scenarios requiring timely insights and minimizes the load on source systems compared to continuous querying. Implementing CDC demands consideration of data consistency and synchronization mechanisms to maintain accurate information.

API

APIs serve as intermediary interfaces that enable seamless communication between data sources and analytics platforms. Data producers expose APIs that allow the analyt-

ics platform to retrieve specific data or perform certain actions programmatically. This approach offers fine-grained control over data retrieval, supports real-time interactions, and enables selective extraction of relevant information. However, designing robust APIs and managing version changes become vital to ensure compatibility and smooth data exchange.

Message Queues and Event-streaming Platforms

Message queues and event-streaming platforms facilitate data sharing through asynchronous messaging and streaming mechanisms. Data sources publish events or messages onto these platforms, and the analytics platform subscribes to receive and process these events. This approach suits scenarios where real-time processing of events is essential, such as IoT applications or dynamic data feeds. The use of message queues and event-streaming platforms requires attention to data durability, ordering, and message processing guarantees to maintain data integrity and reliability.

1.1.4. Transformation

The *transformation* stage is a crucial component of the data engineering lifecycle. In this stage, raw data from diverse sources undergoes a series of operations to ensure its suitability for downstream applications and, possibly, its alignment with quality standards. This subsection delves into the intricacies of the transformation stage, specifically focusing on its significance, challenges, and methodologies in ensuring high data quality standards in data architectures.

Objectives

The primary objectives of the transformation stage revolve around refining and enhancing raw data to make it consistent, accurate, complete, and compliant with predefined formats, meaning, and quality criteria. These objectives can be summarized as follows:

1. Data Transformation - converting data into a standardized format, aligning with business rules and requirements. This step can involve data type conversions, unit standardization, and normalization.

2. Data Cleansing - identifying and rectifying anomalies, errors, duplicates, and inconsistencies in the raw data. This process involves data deduplication, outlier detection, and imputation [1].

3. Data Integration - integrating data from disparate sources into a unified format,

allowing seamless analysis and interpretation; this often involves mapping different data structures and resolving schema disparities. Often, data integration is associated with the concept of data modeling. It is possible to achieve seamless data integration only with advanced modeling capabilities.

4. Data Enrichment - augmenting raw data with additional information from supplementary sources to enhance its value and context.
5. Data Aggregation - combining granular data into higher-level summaries or aggregates to facilitate meaningful insights. Aggregation methods include grouping, summarization, and roll-up operations.

Data Transformation and Data Quality

Within the data engineering lifecycle context, the transformation stage is the pivotal phase that wields significant influence over data quality, exerting both positive and negative effects. While the primary objective at this juncture often revolves around enhancing DQ, it is essential to acknowledge that if the executed data transformations deviate from expected outcomes, adverse consequences can ensue, thereby substantially compromising data quality. This precarious situation can potentially give rise to what can be called "datastrophes" [17]. Hence, a pivotal juncture for scrutinizing DQ must exist immediately after the execution of a data transformation process.

Compute Engines

The successful execution of data transformation heavily relies on efficient and capable compute engines, the systems responsible for executing the transformation logic on the underlying data. Compute engines ensure timely and accurate data processing, primarily as organizations deal with increasingly large and complex datasets. Two primary types of computing engines are commonly employed.

Firstly, *batch processing engines* for handling transformations on data in predefined batches. These engines are well-suited for scenarios where data can be collected over a period and processed in non-real-time intervals. Technologies like Apache Hadoop's MapReduce and Apache Spark's batch processing module are popular choices, allowing the parallelization of tasks for improved performance and scalability. Batch processing allows to optimize resource-intensive transformations by distributing tasks across a cluster of machines. In addition to these distributed computing frameworks—which might be overkill sometimes—many data engineering tasks leverage the power of programming languages such as Python, often utilizing libraries like Pandas and Polars. These packages provide a

versatile and high-performance platform for data manipulation and transformation while harnessing the computing power of a simple multicore system. Furthermore, SQL-based transformations are commonly employed in batch processing scenarios, leveraging the strengths of relational databases and cloud data warehouses to perform operations like joins, groupings, aggregations, and filtering. Tools like Apache Hive, Google BigQuery, and Snowflake enable the execution of SQL-like queries on large datasets, contributing to the rich landscape of batch-oriented transformation techniques.

Secondly, *stream processing engines*, on the other hand, are designed for real-time or near-real-time data transformations. They are employed when data needs to be processed as it arrives, enabling organizations to react promptly to emerging insights. Technologies like Apache Kafka Streams and Apache Flink enable the processing of data streams in real-time, making them suitable for applications such as fraud detection, real-time analytics, and monitoring of system events.

Both batch and stream processing engines offer their advantages and challenges. While batch processing provides the opportunity for comprehensive transformations, stream processing engines enable immediate responsiveness to dynamic data. The choice between these engines depends on the specific use case, data velocity, and business requirements. In either case, engineers must appropriately provision and configure compute engines to ensure optimal performance, fault tolerance, and scalability.

1.1.5. Serving Data

The *serving stage* is a pivotal juncture within the overarching data engineering lifecycle, facilitating the transformation of raw data into valuable insights and actionable information. In this phase, the primary objective is to ensure seamless and efficient access to high-quality data for analytical and operational purposes. As organizations accumulate vast volumes of data from diverse sources, maintaining data quality standards during the data serving stage becomes increasingly evident.

At its core, the data-serving stage involves providing data to end-users, applications, and systems that rely on accurate, timely, and reliable information. Data engineers play a pivotal role in constructing a robust data architecture that supports the real-time or batch retrieval of data and guarantees its integrity, consistency, and relevance.

1.2. Undercurrents

The domain of data engineering has undergone substantial evolution, marked by rapid maturation. In previous cycles, data engineering primarily centered around the technological aspects; however, contemporary trends have transformed this paradigm. Notably, the persistent abstraction and streamlining of tools and methodologies have broadened its scope. Presently, data engineering extends beyond the confines of mere technological implementations, encompassing a more comprehensive array of essential considerations. This expanded perspective entails integration with conventional enterprise practices, including data management and cost optimization, alongside incorporating novel methodologies like DataOps, thus positioning data engineering strategically up the value chain.

The *Fundamentals of Data Engineering* authors termed these practices "undercurrents" [18]. The most important are security, data management, DataOps, data architecture, orchestration, and software engineering (Figure 1.1). However, this thesis mainly focuses on one of them: data management.

1.2.1. Data Management

Data management is a critical undercurrent in the data engineering lifecycle, encompassing the comprehensive strategies, practices, and processes employed to ensure the effective handling, storage, integration, validation, and maintenance of data assets within a data architecture. It represents a systematic approach to address the challenges associated with the ever-increasing variety, volume, and velocity of data generated by modern data platforms. Data management endeavors to promote the organization, accessibility, and reliability of data, facilitating its transformation into valuable information for decision-making and analytics [7].

Data management also encompasses data governance, quality assurance, security, and lifecycle management, which are vital in preserving data integrity, consistency, and usability across mono- and cross-enterprise systems. A robust data management framework establishes the foundation for ensuring high data quality standards, as it enables data engineers and stakeholders to trust the information derived from the data architecture, fostering better decision-making and enhancing overall business performance.

Data Governance and Accountability

Data Governance is "first and foremost, a data management function to ensure the quality, integrity, security, and usability of the data collected by an organization" [8].

While data management refers to the overall process of acquiring, storing, organizing, processing, and utilizing data systematically and efficiently, on the other hand, data governance is a more focused aspect of data management that deals with establishing policies, processes, and rules governing the usage, access, and handling of data across the organization.

Data governance involves two core concepts: discoverability and accountability [8]. While discoverability focuses on making data available and discoverable by the end users through the usage of metadata, and hence it is not related to data quality, accountability is a crucial data governance aspect that directly impacts it.

Conceptually, "accountability consists of defining governance to comply in a responsible manner with internal and external criteria, ensuring implementation of appropriate actions, explaining and justifying those actions and remedying any failure to act properly" [9]. Put, accountability means assigning an individual to govern a portion of data. A person responsible for a data asset handles several domains, including data quality.

1.3. Data Quality

Even though it is possible to find in books and articles generic definitions of data quality such as "the optimization of data toward the desired state" [18] and "the health of data at any stage in its life cycle" [16], the most shared vision by the literature is data quality as a multidimensional concept [29, 32]; this implies that the quality of a piece of data can be decreed by analyzing one after the other a series of characteristics it possesses that have been termed "dimensions". Therefore, it is possible to derive a DQ definition by focusing on its dimensions.

1.3.1. Dimensions

In the literature, many articles outline the essential dimensions to define data quality [2, 29, 32, 33]. The Data Management Association of the UK (DAMA UK) drafted one of the most comprehensive resources identifying six of them [14]:

1. Completeness is the degree to which records are present. In other words, the proportion of stored data against the virtual is 100% complete;

2. Consistency corresponds to the degree to which data set values do not contradict other values representing the same entity. Consistency identifies the absence of difference when comparing two or more representations of a thing against its definition;

3. Uniqueness is the degree of record duplication; this indicates whether each value is saved only once and that there is only one record for each entity the data represents;
4. Validity corresponds to the degree to which the data is within the expected range and format. Data is valid if it is compliant with its definition in terms of format, syntax, and range;
5. Accuracy represents how data correctly describes the real-world object or event. Accuracy provides information about how well data corresponds to reality;
6. Timeliness identifies the degree to which the data accurately reflects the period they represent and that the data and its values are up to date.

1.3.2. Relevance

In the contemporary era of data-driven decision-making and the proliferation of data-intensive technologies, the relevance of data quality has emerged as a critical concern for organizations across various sectors. The exponential growth in data velocity, volume, and variety, coupled with the need to integrate diverse data sources, has accentuated the importance of maintaining impeccable DQ standards within data architectures.

Akin to the adage “Garbage in, Garbage out,” the quality of insights, predictions, and actionable intelligence derived from data analytics, machine learning, and artificial intelligence systems is heavily contingent upon the quality of underlying data. Organizations prioritizing data quality benefit from enhanced business intelligence, better-informed decision-making, and the ability to identify emerging trends and patterns proactively. High-quality data fuels the efficacy of advanced analytics, empowering data scientists and analysts to extract meaningful correlations and actionable insights.

Ensuring high data quality in data platforms is a prerequisite for yielding trustworthy and credible outcomes and a key determinant of operational efficiency, cost-effectiveness, and regulatory compliance. Poor DQ can lead to erroneous business decisions, flawed customer insights, and compromised organizational performance, ultimately undermining the competitive advantage that data-driven organizations seek to attain.

With the *modern data stack*, data engineers, analysts, and scientists have much freedom; this is fine in the early stages of a company when things go very fast, and flexibility and time-to-action are essential preconditions for the start-up. However, when a company starts to establish itself in the market, and the business takes a definite shape, this freedom and flexibility can become a real threat. Indeed, this approach can affect the quality of data. Many companies have experienced severe repercussions due to inaccuracies in the

data and consequent serious errors in decisions based on the insights provided by analytics or machine learning models [22].

In a company facing an exit event, such as an IPO, the information provided to the regulators and the entities involved plays a pivotal role in determining the outcome. The regulators and entities overseeing the transaction seek reliable insights from the company's data team under consideration and require comprehensive details about the used data platform to get such information. Their assessment revolves around the suitability of the data platform vis-à-vis the company's stage, ensuring that it can yield dependable insights for inclusion in its public financial statements. Unfortunately, this crucial phase often leads to significant delays, and the primary impediment lies in the data architecture itself, which fails to meet the requisite quality standards mandated by regulators.

2 | Related Works

This Master's Thesis seeks to integrate various established theoretical and technological solutions into a cohesive framework resembling a flowchart. By leveraging specific input parameters, the framework systematically poses a set of inquiries to comprehend the optimal technological approach and implementation choices.

Therefore, this brief chapter introduces the theoretical concepts and technological tools that will be part of the final framework; this is a generic introduction to these concepts, as later chapters will present the details.

2.1. Data Contracts

In the book *Data Pipelines Pocket Reference*, the author defines a data contract as "a written agreement between the owner of a source system and the team ingesting data from that system for use in a data pipeline. The contract should state what data is being extracted, via what method (full, incremental), how often, as well as who (person, team) are the contacts for both the source system and the ingestion. Data contracts should be stored in a well-known and easy-to-find location such as a GitHub repo or internal documentation site. If possible, format data contracts in a standardized form so they can be integrated into the development process or queried programmatically" [5].

In summary, a data contract is a formal agreement that specifies how data should be structured and interpreted when exchanged between different software components or systems. It helps ensure consistency and reliability in communication, even when the components or systems are developed independently or by different parties.

There are two main approaches to data contracts. These two approaches depend on the stage at which the check of contract fulfillment takes place. Suppose it happens before the data flows into the data platform (Figure 2.1). In that case, the system can proactively discover data issues beforehand, preventing the propagation of inconsistent data within the analytics platform [25].

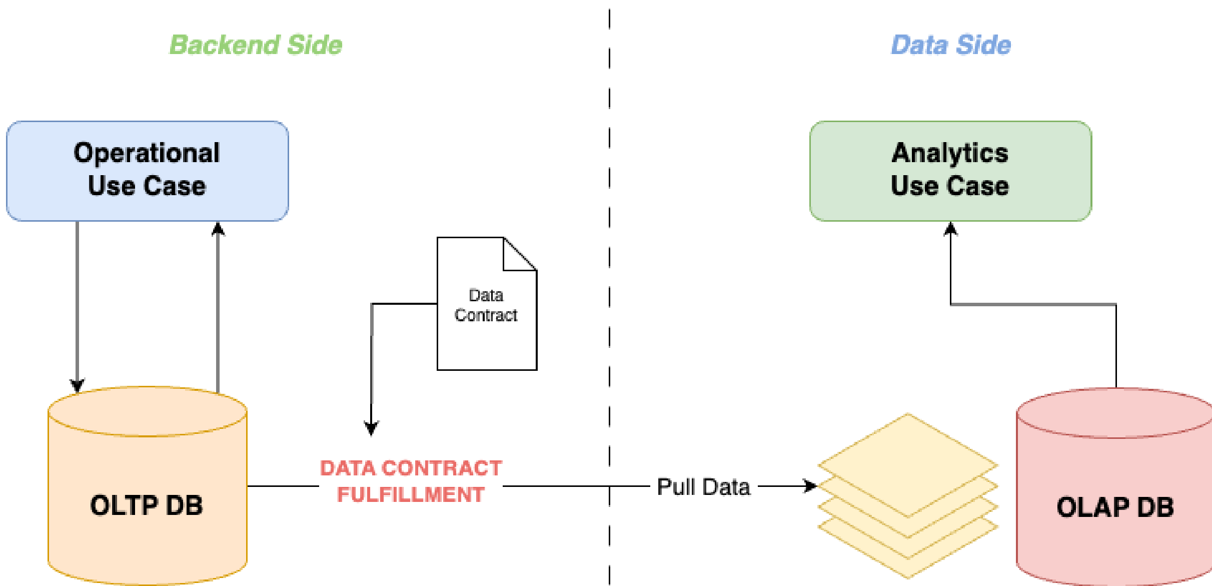

Figure 2.1: Data contracts for Transactional Databases: a prevention-oriented approach that allows a system to spot data issues before ingesting them into the analytics platform.

On the other hand, if the data contract fulfillment takes place within the ETL platform (Figure 2.2), then the system will follow a more detection-oriented approach [24]; this implies a further effort to fix the data issues if needed but still enables automatic detection of compromised data.

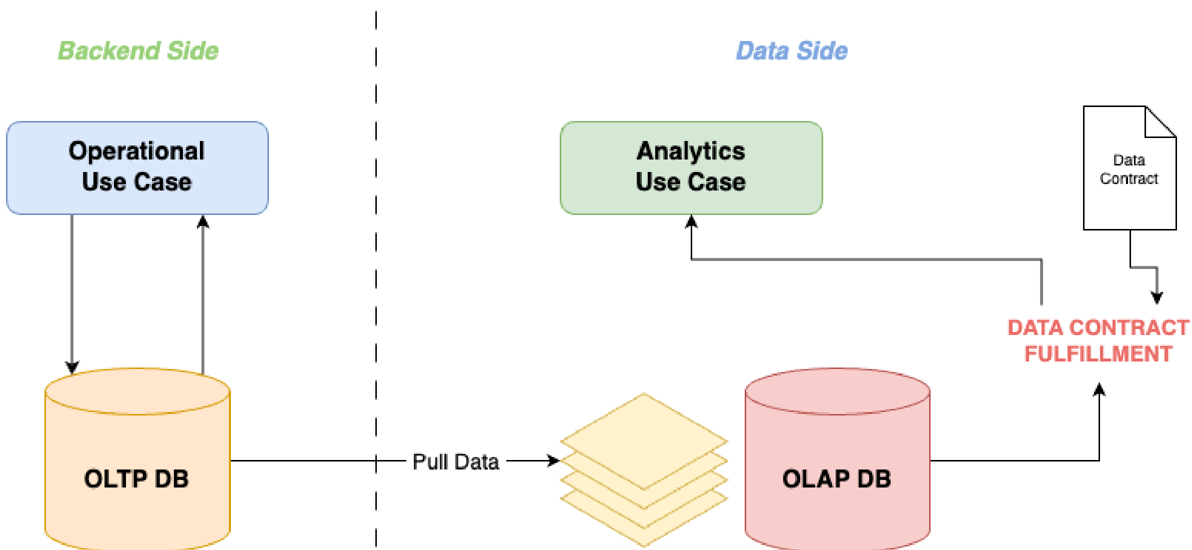

Figure 2.2: Data contracts for data platforms: a detection-oriented approach that allows a system to automatically spot data issues within the ETL processes.

Thus far, the focus has been exclusively on use cases where an OLTP database represents

the upstream provider, and the downstream consumer is a data lakehouse. The scope of this problem can be expanded, though, to any source, not just databases. In such a case, the core idea looks essentially the same.

2.2. API

In modern data architectures, where the proliferation of data sources and the need for seamless data integration is paramount, the REST (Representational State Transfer) API is a pivotal technological tool (see Chapter 1 for further information about API from the data engineering lifecycle perspective). Of particular relevance to the topic at hand is the distinction between the *pull* and *push* approaches in the context of REST APIs [18]. The difference is in who is responsible for starting the cross-entity communication and designing and developing the API.

The *pull approach* (Figure 2.3) involves a consumer—the data platform—requesting data from one or more sources when needed. Therefore, each provider has its API implemented according to their needs. While this approach provides flexibility to the providers, it may result in disparate data quality standards, as each may interpret and provide the data differently. This lack of standardization can lead to inconsistencies in data handling, impeding adequate DQ assurance across the ecosystem.

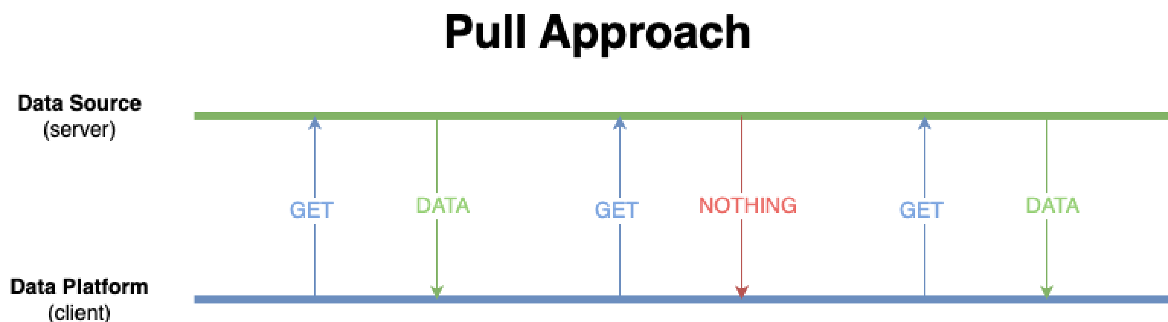

Figure 2.3: API Pull approach: the Data Sources are responsible for designing and implementing each API; the data consumer, on the other hand, is responsible for starting the communication by fetching the data via GET requests.

Conversely, the *push approach* (Figure 2.4) emerges as a promising solution to the challenges posed by data quality standardization. In the push approach, the sources actively send data updates to the consumer in real-time or on a predefined schedule leveraging an API designed and implemented by the consumer. This approach offers a unique advantage in ensuring high DQ standards. By establishing a well-defined protocol, the consumer can

ensure that data is transmitted in a predetermined format and adheres to specified quality standards.

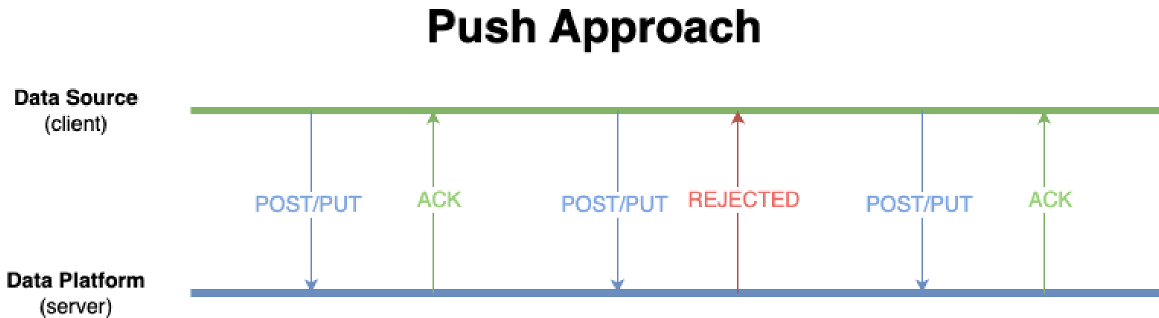

Figure 2.4: API Push approach: The data Consumer is responsible for designing and implementing the API; the data sources, on the other hand, send the data via POST/PUT requests according to a pre-defined timeline.

It is possible to associate the concept of *push-based API* with data contract. The API implementation encapsulates the agreement between the data source and the consumer. It defines the exchanged data's structure, format, semantics, and quality attributes. Through this contract, the consumers communicate data accuracy, completeness, consistency, and timeliness expectations to the sources; this not only streamlines the integration process but also enforces uniform DQ standards throughout the data architecture.

In conclusion, within the context of ensuring high DQ standards in data architecture, the push approach to REST API usage emerges as a potent strategy when intended as a proper data contract. As organizations increasingly rely on data-driven decision-making, implementing such a framework can enhance data reliability, accuracy, and usability across single and cross-enterprise.

2.3. DQ Platforms and Tools

Data quality platforms and tools are pivotal in offering comprehensive solutions to assess, monitor, and enhance DQ across data platforms. By describing some of these tools in detail, one can understand how these solutions are of fundamental importance for the final framework's objective. This section explores some of the prominent data quality platforms and tools, namely Monte Carlo, DBT Tests, and Great Expectations, delving into their features, capabilities, and contributions to maintaining data quality.

2.3.1. Monte Carlo

Monte Carlo (MC)—a data observability platform—addresses the challenge of detecting and mitigating data pipeline issues that could impact data quality. By leveraging statistical analysis and machine learning techniques, Monte Carlo proactively monitors data pipelines and identifies anomalies, data drifts, and DQ deviations. This platform enables data engineers and analysts to gain real-time insights into the health of their data, ensuring timely identification and resolution of potential issues. Monte Carlo's automated data monitoring and alerting capabilities enhance the efficiency of DQ management by reducing the time required to identify and rectify anomalies, ultimately contributing to more reliable data-driven decision-making.

2.3.2. DBT Tests

Data Build Tool (DBT) has emerged as a powerful platform for transforming and modeling data within the modern data stack. One of its key features is the ability to define and execute data tests as part of the transformation process. DBT Tests allow data engineers and analysts to establish DQ rules and validation criteria, ensuring the processed data conforms to expected standards. These tests can encompass various DQ dimensions, such as completeness, accuracy, and consistency. By integrating data quality checks seamlessly into the data transformation workflow, DBT promotes a proactive approach to maintaining data quality. The platform's automated testing capabilities help identify issues early in the data transformation process, minimizing the propagation of errors and inconsistencies downstream.

2.3.3. Great Expectations

Great Expectations offers an open-source framework for defining, managing, and validating data expectations. This tool allows data practitioners to codify their assumptions about data quality in the form of *expectation* definitions. These expectations encompass various aspects of data quality, including schema structure, value range, uniqueness, and more. By specifying these expectations, data teams can automatically validate incoming data against predefined criteria, thus ensuring that the data adheres to the anticipated quality standards. Great Expectations provides a flexible and customizable approach to DQ management, allowing organizations to tailor expectations to their specific use cases and data sources. Its open-source nature fosters collaboration and knowledge sharing among data professionals, contributing to a community-driven effort to elevate DQ practices.

3 | Problem Statement

This chapter elucidates the predicament targeted by the framework proposed in this thesis. It introduces the core issue and delves deeply into its intricacies by probing its underlying causes. Effectively addressing a problem necessitates a comprehensive understanding of its origins, enabling the integration of a system that precludes stakeholders from making inadvertent errors and facilitates the identification of anomalies.

At the heart of this challenge lies the data itself. The assurance of data quality hinges on aligning the data with predetermined expectations. Conversely, deviations from anticipated data characteristics indicate discrepancies that originated further upstream. As such, two distinct vantage points emerge. One view centers on the data, allowing a direct perception of the manifest problem. However, a more profound comprehension requires peering beyond the surface and examining stakeholder behaviors, thereby uncovering the fundamental roots. Frequently, these roots trace back to miscommunications among the entities involved. Given the diverse spectrum of systems, various communication patterns come into play. This diversity occasionally obscures the precise identification of the problem's origin.

3.1. Manifestation

The data-centric vantage point constitutes an essential dimension for comprehending the intricacies of the manifest problem. At its core, the challenge lies in DQ assurance, a foundational tenet in data architecture. Data quality is a linchpin, dictating the efficacy of decision-making processes and downstream analyses. When data conforms to pre-established expectations, its veracity, consistency, and reliability are upheld, culminating in an environment conducive to accurate insights and conclusions.

However, the data landscape is not immune to aberrations. Deviations from anticipated data characteristics indicate anomalies that materialize upstream in the data flow. Such anomalies manifest in inconsistencies, errors, or inaccuracies within the dataset, potentially stemming from data acquisition, transformation, integration, or migration processes.

Detecting these discrepancies is pivotal, as they often signify underlying systemic issues that warrant resolution.

Two main aspects need particular attention: quality dimensions presented in Subsection 1.3.1 and the schema. It is always possible to check the health status of DQ in a system by monitoring these two domains.

3.1.1. Quality Dimensions

In principle, a robust Data Quality Assessment system is paramount in a piece of software—whether it is a production service or a data platform. Such a system must be able to continuously evaluate the various dimensions of data quality, thereby identifying potential inconsistencies. Anomalies in data quality become evident when discrepancies arise across any of these quality dimensions.

The effectiveness of this quality control mechanism hinges on its ability to adapt to the specific requirements of each quality dimension. Sometimes, a singular data record under examination suffices, while other instances necessitate a broader perspective encompassing the entire dataset. For instance, examining the individual record can validate a field marked as `non-nullable` for `null` values. In contrast, confirming the `uniqueness` of a value in a field designated as `unique` demands a comprehensive comparison across all records within the dataset.

Distinct quality dimensions necessitate divergent data access strategies (refer to Table 3.1). This delineation becomes pivotal during the deployment of a Data Quality Assessment system.

Partial-data Dimensions	Full-data Dimensions
Accuracy	
Completeness	Consistency
Timeliness	Uniqueness
Validity	

Table 3.1: Quality dimensions and how a system can monitor them: in some cases, the system needs access to the full dataset; in others, it just needs access to the single record or a batch.

In tandem with a monitoring mechanism, implementing an alerting system assumes significance; this enables timely notifications when the monitoring system identifies data

anomalies. Subsequently, stakeholders are always aware of the DQ health status, allowing them to initiate corrective measures to enhance overall data quality.

Nevertheless, even if an impeccable DQ control service were conceivable, it would address only part of the issue. The genesis of compromised data often traces back to earlier stages in the data lifecycle. While a Data Quality Assessment system excels in detecting data irregularities, it does not possess the capacity to prevent their occurrence. At this juncture, the imperative is to delve deep into the intricate dynamics among the various stakeholders to uncover the underlying sources of the problem (further details at Section 3.2).

Issues can arise from multiple entities. They might originate from within the data team (refer to Subsection 1.1.4) or external stakeholders. The latter might inadvertently introduce bugs in the services or malfunctions in the systems, or they might deliberately alter the semantic interpretation of a field without synchronization with downstream data consumers—such as converting a currency column from Euro to US Dollar. These examples exemplify potential sources impacting DQ dimensions. Another critical concern, also aligned with data quality, is the uncommunicated alteration of a dataset schema, significantly breaking systems under the control of downstream consumers.

3.1.2. Schema Compatibility

To ensure data quality, one of the recurring challenges pertains to *schema compatibility*, a matter that frequently arises due to incongruences between data producers and downstream consumers. It is not uncommon for data producers to introduce modifications to the schema without synchronized communication with those who rely on the data downstream (further details in Section 3.2). These seemingly innocuous alterations can have profound ramifications on the entire data architecture. When producers unilaterally introduce schema changes, downstream consumers can experience a breakdown in their processing pipelines, as the data they receive no longer adheres to their expected structure. This phenomenon can result in data ingestion errors, processing failures, and inconsistencies in analytical outputs. The impact extends beyond technical disruptions; misaligned schema can propagate incorrect insights and flawed decision-making.

Two potential solutions warrant consideration to address the challenges posed by schema compatibility. The first is the concept of "schemaless" architecture, wherein data is stored in a more flexible and adaptable format, allowing for changes in structure without disrupting downstream processes. While offering a degree of resilience to schema modifications, this approach introduces its complexities related to querying and data validation. It might not be as effective as it promises to be. The second solution involves establishing well-

defined data contracts between data producers and consumers. These contracts articulate the data's structure, semantics, and expectations, creating a shared understanding that can mitigate the risks of unsynchronized schema changes (see Section 2.1). This approach promotes consistency and transparency across the data ecosystem by enforcing proper communication channels and requiring explicit agreement before schema alterations.

In general, the solution lies in adding an abstraction layer that allows the decoupling of the producers' and consumers' systems so that modifications upstream do not directly impact the downstream processes. Ultimately, managing schema compatibility is critical in ensuring high DQ standards and the effective functioning of both single and cross-enterprise.

3.2. Root

The second vantage point focuses on the root of the problem: the interactions between the various stakeholders in an information system and how any inefficiency can lead to poor-quality data being produced upstream and consumed downstream.

First, it is necessary to present the roles that the various entities involved may assume. According to the article "Data Quality in Context" [29], these can be of three types (see Figure 3.1):

1. Data Producers - entities or systems responsible for producing and providing the platform with data. They could include various sources such as sensors, databases, or applications that Software Engineers (SWEs) usually set up. In this context, Data Engineers (DEs) consider the data generated by SWEs as *internal data*; typically, this data is in the form of events or logs. Also, external organizations—SaaS platforms or partner businesses—can provide data to the platform; in this case, data engineers consider it as *external data* [18];

2. Data Custodians - entities in charge of providing and managing computing resources for storing and processing data. In the modern data context, there is usually a one-to-one correspondence between data custodians and engineers. Put, "the data engineer is a hub between data producers, such as software engineers [...], and data consumers, such as data analysts, data scientists, and ML engineers" [18]. Even though the *Data Custodian* is a well-defined role, sometimes it is also considered a *consumer*. In some way, an ETL platform consumes data from various upstream producers, and therefore, this thesis also refers to it as a consumer;

3. Data Consumers - entities or components that use and extract insights from the data

available on the platform. These consumers can span a broad spectrum, ranging from data analysts and scientists to various applications and automated algorithms. The primary focus of data consumers is to derive value and knowledge from the data, facilitating informed decision-making processes and generating actionable outcomes. This category encapsulates individuals or systems that interact with the data directly or indirectly, ultimately driving the purpose of the entire data ecosystem.

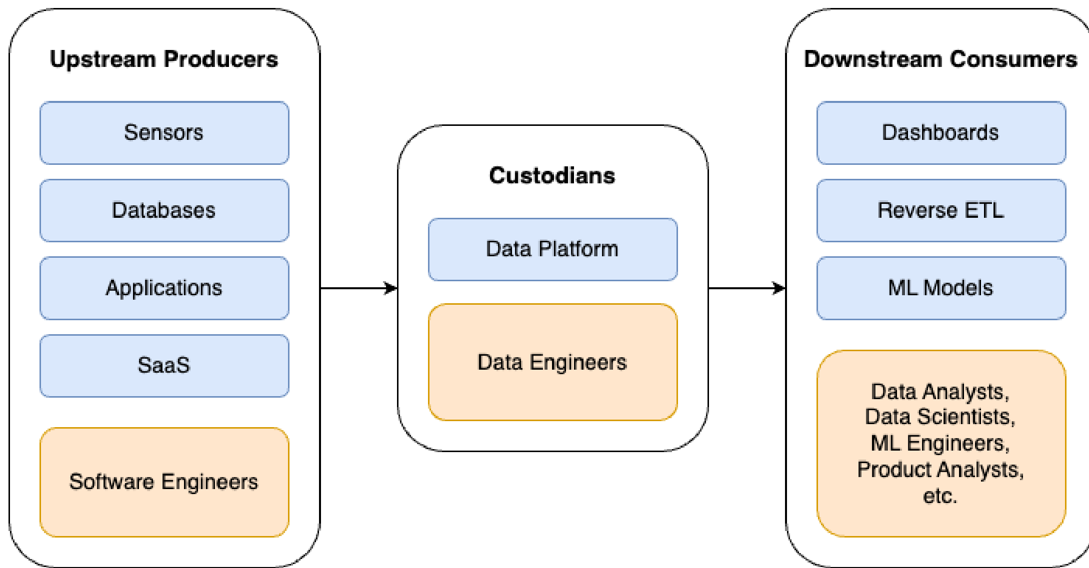

Figure 3.1: Technical Stakeholders of Data Engineering [29].

In contemporary intricate data ecosystems, a prevalent issue exists wherein data producers frequently need adequate awareness or responsibility regarding the potential ramifications of their downstream actions. This deficiency in motivation to assume accountability for the quality of data they generate beyond immediate operational necessities hinders the overall data quality. Their primary focus remains confined to fulfilling specific tasks or objectives, disregarding their data's profound implications on subsequent analytical procedures and decision-making protocols.

Likewise, data consumers, who depend on the outcomes furnished by the analytics platform, encounter a parallel challenge. They find themselves in need of greater comprehension of the data sources upstream. While proficient in utilizing technologies such as SQL and Python, these consumers often lack a comprehensive framework of robust prerequisites to guide the workforce responsible for producing data upstream. This juxtaposition highlights a critical disparity between technological adeptness and formulating comprehensive guidelines for upstream processes. [21]

Indeed, improving the interaction between data producers and consumers is crucial for

addressing DQ issues after they arise and preventing ingesting bad data into a data platform altogether. The concept of prevention underscores the idea that establishing robust communication channels and collaborative workflows between these stakeholders is a proactive measure to avert the introduction of poor-quality data into the ecosystem (Figure 3.2). By fostering a shared responsibility and awareness culture, producers gain a clearer understanding of how downstream participants utilize their data; this heightened awareness naturally leads to a more cautious approach, motivating them to ensure the accuracy, consistency, and adherence to established standards in the data they generate.

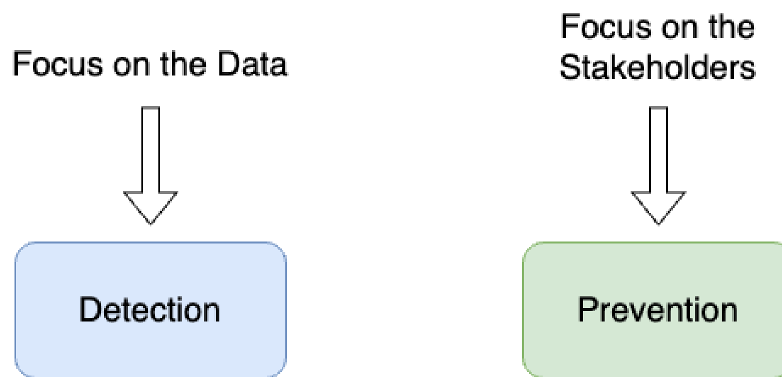

Figure 3.2: Detection VS. Prevention: where to focus the effort to detect and prevent the data quality issues.

Similarly, consumers with a more profound comprehension of upstream data sources can provide more precise specifications and expectations for the required data. This proactive engagement empowers data producers to align their processes with consumer needs, reducing the likelihood of data inconsistencies or misinterpretations. In this context, prevention entails technical alignment and a cultural shift that promotes open communication, knowledge exchange, and mutual respect among stakeholders.

3.3. Proposed Solution

As reiterated throughout prior discussions, the fundamental focus of this thesis revolves around presenting a comprehensive framework aimed at aiding teams or organizations responsible for data platforms in the meticulous selection of optimal technological solutions. These solutions aim to uphold elevated DQ standards within managed datasets. The organization must thoroughly analyze the prevailing circumstances involving various stakeholders to identify the most fitting option.

It is important to note that, given the initial stage of framework development, it is un-

realistic to anticipate its encompassment of the entirety of existing collaborative scenarios—a realm that might boast of practically infinite possibilities. Neither can it prescribe precise approaches for maintaining data quality across every conceivable scenario. This thesis deliberates exclusively upon three archetypal cases representing the most prevalent situations. Subsequent iterations will progressively expand the framework's scope, accommodating additional scenarios and their respective solutions in a horizontal progression (see Section 6.2).

The final framework will be in the form of a flowchart. At every step, it questions the users about the problem they want to solve and their context. Then, it suggests the most suitable solution to adopt to achieve the goal based on the answer provided by the user. Where possible, it will advise implementing prevention mechanisms. However, there are situations where there is no way to apply any form of prevention, and hence, detection is the only viable solution to monitor data quality.

4 | Framework and Implementation

This chapter serves as the cornerstone of the thesis, delving into the intricate details of the framework. It elucidates the array of technological solutions comprising the final contribution (depicted in Figure 4.1). These solutions represent distinct approaches to enhancing an organization's data quality tailored to specific scenarios.

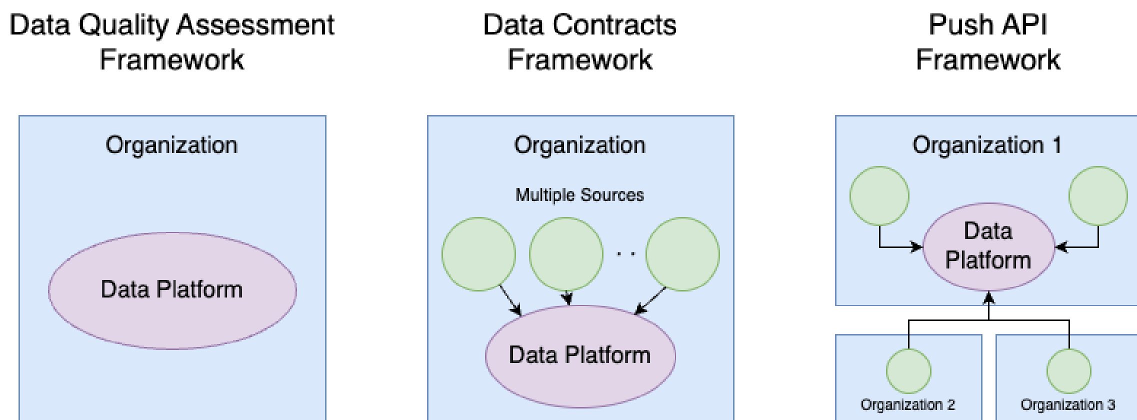

Figure 4.1: Technological solutions involved in the framework and the associated application contexts.

The first scenario revolves around an organization reaching the proper *data maturity* stage, where internal controls are necessary to uphold impeccable data quality standards. In this context, the solution lies in a Data Quality Assessment (DQA) approach (Section 4.1), the elemental unit of the final framework. Its implementation predominantly rests with the data team. Consequently, it is a universally applicable solution, irrespective of interactions with internal software engineers or external data sources. This solution's implementation may correspond to the foundational step toward standardizing data controls and achieving high-quality standards. However, it is essential to note that this approach primarily detects data issues when the data already reached the data platform. Therefore, it does

not include any form of DQ prevention.

On the other hand, the subsequent solutions proposed by the framework—Data Contracts (DCs) (Section 4.2) and Push API (PAPI) (Section 4.3)—adopt a preventive stance. Implementing these technologies empowers the data team to avert the influx of subpar data into the analytics platform. Both approaches are valid options to adopt in multi-source environments but have a subtle distinction. Data contracts usually govern data exchange within the same organization, whereas a Push API finds utility in cross-enterprise systems.

4.1. Data Quality Assessment Solution

When organizations embark on their journey, data often is not the primary focus due to its scarcity. The same applies to data quality. Nevertheless, establishing sound technological infrastructure in the early stages facilitates seamless integration of a proper data architecture later.

Adopting a data-driven approach becomes imperative as data accumulates, marking the organization's path toward data maturity. In nascent data maturity stages, enforcing solutions to prevent quality issues may be excessive due to the modest headcount and informality of interactions. Consequently, the emphasis stops at detecting data issues (Figure 4.2), hence implementing a Data Quality Assessment solution.

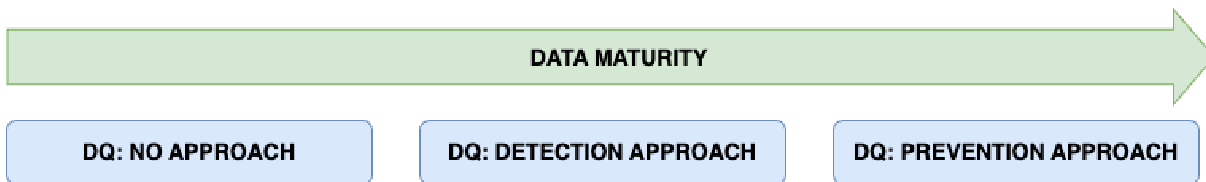

Figure 4.2: Approaches to data quality based on the data maturity stage.

The pivotal question determining the viability of this solution is: "Has the organization transitioned to a more enterprise-oriented data approach?" If affirmative, data quality becomes fundamental, warranting the implementation of a system capable of autonomously identifying data inconsistencies and promptly notifying data owners (Figure 4.3).

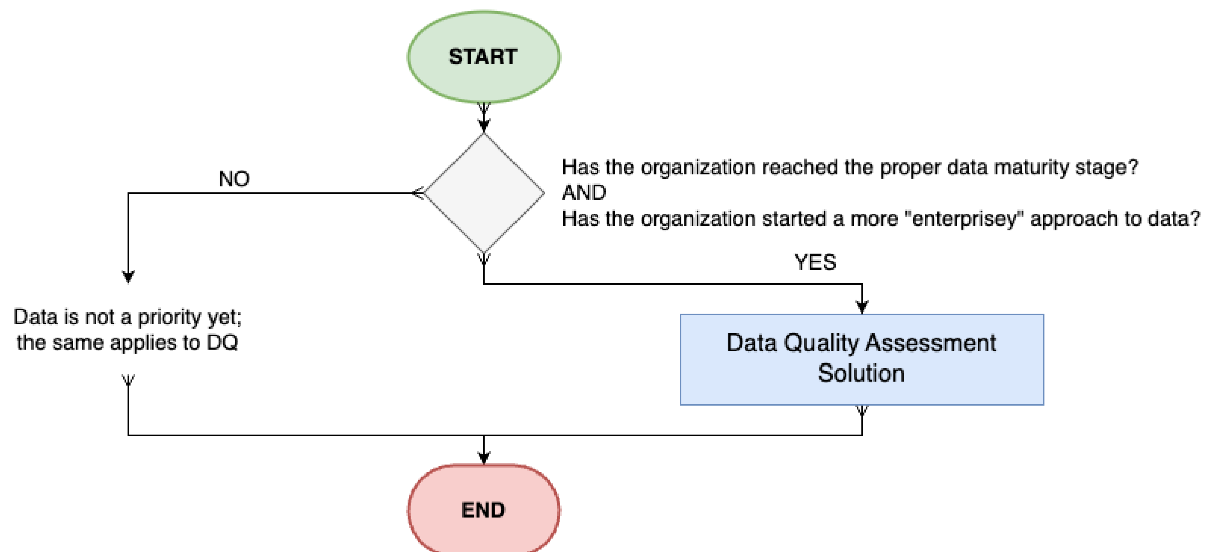

Figure 4.3: First part of the final framework.

4.1.1. Context

The subsequent sections outline a plausible implementation of this technological solution. Specifically, they present the DQA Framework employed at Blockchain.com (BCDC)—a company operating in the crypto and financial sector for over a decade. As the company reached data maturity, concerns regarding data quality emerged. Initial strides happened with in-house mechanisms mainly for consistency checks. A more significant leap occurred later with a data stack modernization, leading to a new data lakehouse platform. The platform's launch in late 2022 prompted the integration of an innovative Data Quality Assessment solution to autonomously identify inconsistencies in ETL-processed data. Refer to the System and Workflow section (4.1.3) for comprehensive details on its operation and implementation.

The challenge posed to the data infrastructure team revolves around the significance of the managed data. Operating within the financial sector subjects the organization to regulatory audits. The financial reconciliation team demanded close alignment between the lakehouse data and production databases. They requested to minimize inconsistencies attributed to ETL processes as much as possible to deliver extremely precise reports upon request.

In crafting this framework, the community of Great Expectations—one of the tools used in development (Section 4.1.2)—provided relevant insights. Inspiration was also derived from Mediaset Group's implementation, as presented in the article "How to monitor Data Lake health status at scale" by Davide Romano [20].

4.1.2. Tooling

A fundamental pillar in establishing a comprehensive DQA Solution is using data quality platforms and tools (Section 2.3). These tools are pivotal in empowering Data Engineers (DEs) to observe data intricacies and proactively tackle the complex challenge of detecting data pipeline issues that could undermine data quality. Specifically, within the context of the BCDC use case, the primary tool of choice is Great Expectations (Section 2.3.3). This decision arises from its open-source nature, remarkable flexibility, and the ability to tailor it according to specific requirements.

Great Expectations allows data practitioners to define and formalize *data assets*, encapsulating their expectations concerning data quality dimensions and schema specifications. Consequently, developers establish a virtual repository where to codify expectations about the data. One can analogously regard *expectation suites* as data contracts confined within the data platform. As expounded in Section 2.1, data contracts encompass two distinct approaches, differing in the phase of contract fulfillment. The second approach entails fulfillment within the data platform (Figure 2.2), permitting DEs to put in place an adept detection mechanism: the Data Quality Assessment framework.

As a technological tool, Great Expectations effectively translates the conceptual essence of data contracts within data platforms into practical implementation. Note that this conceptual framework for detection can also be actualized by harnessing alternative technologies outlined in Section 2.3 (e.g., DBT Tests, Monte Carlo) or opting for an in-house solution.

4.1.3. System and Workflow

The system's implementation detailed in this section harnesses the capabilities of Great Expectations, a Python-based tool encompassing a diverse range of software components. Understanding the tool's core components is indispensable to navigating this implementation. Table 4.1 introduces the fundamental concepts within the GX glossary, each holding significant relevance in understanding the ensuing discourse.

Concept	Definition
Expectation	An Expectation is a verifiable assertion about data. Great Expectations' built-in library includes more than 50 common Expectations.
Custom Expectation	A Custom Expectation is an extension of the Expectation class, developed outside the Great Expectations library.
Expectation Suite	An Expectation Suite is a collection of verifiable assertions about data. Expectation Suites combine multiple Expectations into an overall description of data.
Validation Results	A Validation Result is an object GX generates when validating the data against an Expectation or Expectation Suite.
Data Docs	Human readable documentation generated from Great Expectations metadata detailing Expectations, Validation Results, and others.

Table 4.1: Great Expectations Glossary - `https://docs.greatexpectations.io/docs/glossary`.

The optimal approach to introducing the system and its implementation is to dissect it into phases. Consequently, the system design and workflow unfold across seven distinct stages:

1. Requirements Stage - The engineer overseeing the integration of a detection mechanism for a specific dataset convenes with data producers (e.g., SWEs) and consumers (e.g., product teams) to amass a comprehensive set of requirements and expectations;

2. Experimentation Stage - The engineer delves into the dataset, manually exploring whether some samples of the dataset align with the expectations outlined in the prior stage;

3. Definition Stage - the engineer transitions expectations into code, effectively crafting a robust data contract and subsequently integrating the checks into the ETL pipeline;

4. Inference Stage - The system autonomously deduces Great Expectations' artifacts, initiating from the contract's definition;

5. Validation Stage - During ETL pipeline execution, the system employs Great Expec-

tations to validate expectations, identifying potential data inconsistencies. When the system detects an issue, it promptly notifies the owner;

6. Monitoring Stage - Engineers formulate and consult data quality reports, facilitating the continuous oversight of the organization's data quality trends;
7. Documentation Stage - The system generates lucid documentation from Validation Results, contributing to comprehensible and easily accessible insights.

The ensuing sections delve into the intricacies associated with each of these stages.

Requirements Stage

The *Requirements Stage* (depicted in Figure 4.4) is a pivotal cornerstone ensuring the framework's effectiveness. It involves a collaborative effort, as the team responsible for embedding data quality checks into a specific organization data asset convenes with two essential parties: the data producers, often comprising software engineers managing the underlying production service, and the consumers, encompassing product team members and data analysts engaged in analytical tasks.

In theory, the actual integration of such quality and consistency checks lies within the domain of the data engineering team. Nevertheless, in nascent organizations, the active involvement of an Analytics Engineer (AE) might also contribute to this aspect.

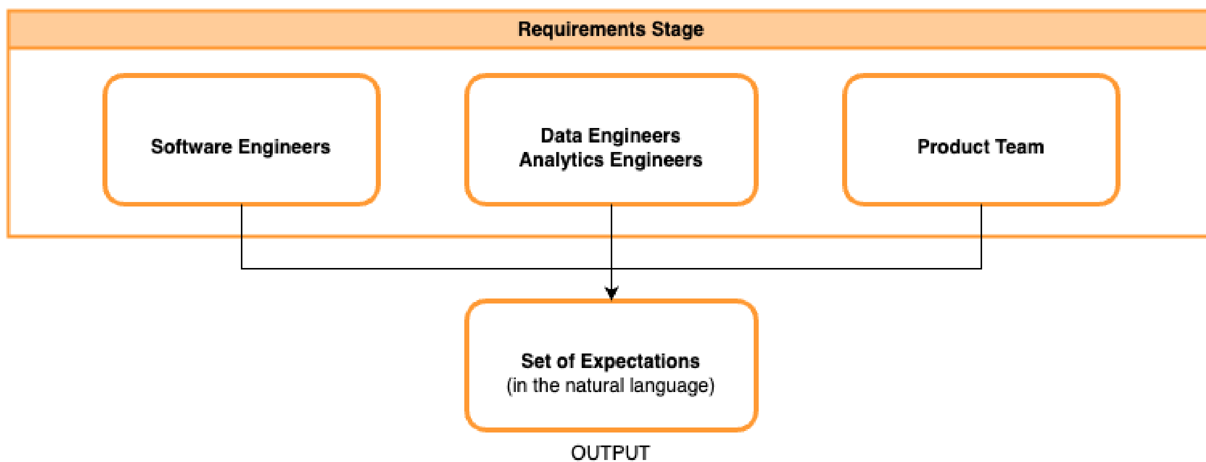

Figure 4.4: Requirements Stage: SWEs, DEs, AEs, and the product team meet to write down a set of expectations they have regarding a data asset.

The outcome of this stage manifests as a documented representation of the comprehensive discussions held. The DE or AE must have a firm grasp of data semantics and expectations. Nevertheless, the initial list of requirements need not be exhaustive; the subsequent

stages accommodate the inclusion of new expectations and checks. Indeed, such evolution is quite commonplace [25]. Over time, the absence of evolving requirements may signify stagnant or less valuable underlying data.

In essence, the requirements stage is the bedrock for establishing shared understanding, ensuring that data producers, DE/AE teams, and product teams collectively articulate expectations concerning specific data assets.

Experimentation Stage

The *Experimentation Stage* (illustrated in Figure 4.6) embodies a crucial phase where the DE/AE responsible for quality checks engages in a hands-on validation process. Their task entails manually assessing a subset of data samples against the expectations established in the prior requirements stage. This stage draws upon the expectations outlined in the previous stage, subjecting them to scrutiny through practical application.

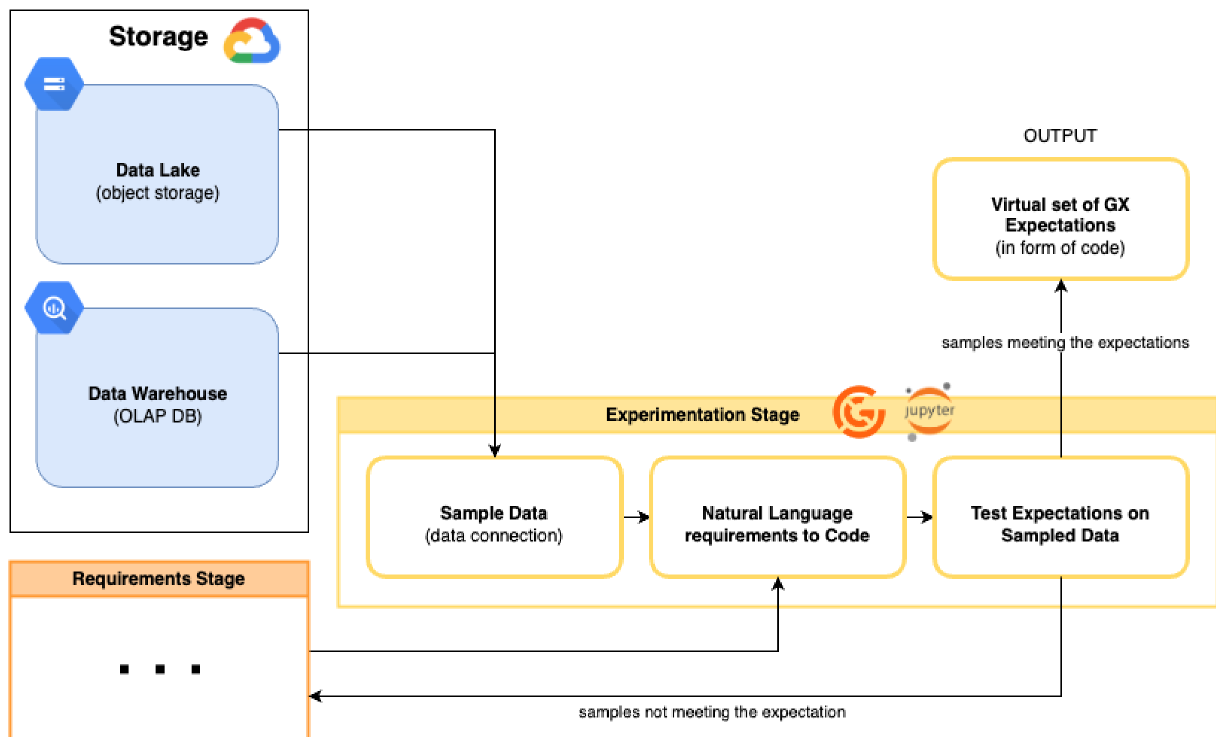

Figure 4.5: Experimentation Stage: connect to the data via GX connectors, infer the possible set of Expectations, and validate them on data samples; eventually re-iterate if data does not meet expectations. The output corresponds to the“virtual” set of Expectations to integrate into the final ETL pipeline.

At Blockchain.com, the data platform team equips AEs with the capability to leverage

Great Expectations within a Jupyter Notebook environment. AEs connect to the pertinent data asset within this setup through GX connectors. The subsequent task involves translating the natural language expectations into executable code; this is possible by selecting relevant expectations from the existing Great Expectations library. Engineers must craft custom expectations tailored to precise needs in more intricate scenarios. The outcome is a series of expectations—Python functions or classes encapsulating the stipulations from stakeholder conversations. These expectations are then employed to assess the subset of previously acquired data samples.

This stage often prompts further refinement of the expectation set. Instances may arise where the data fails to align with initial stakeholder expectations. Consequently, an iterative process of adjusting expectations is embarked upon to harmonize anticipated data behavior with actual observations.

In essence, the experimentation stage empowers DEs/AEs to bridge the conceptual gap between stakeholder-defined expectations and real-world data behavior, fostering an adaptive and iterative approach toward refining expectations.

Definition Stage

The journey progresses to the *Definition Stage* (depicted in Figure 4.6). In this phase, the data practitioner at the helm of quality checks translates the identified expectations into code and seamlessly integrates them into the version-controlled data contract.

At Blockchain.com, the data team meticulously outlines each data asset or table within the codebase. This approach facilitates the ongoing monitoring of table evolution and serves as a centralized repository containing essential information about data assets overseen by the data team. Effectively, the table definition evolves into a data contract, encapsulating details such as location, partitioning, schema, and column types.

During this stage, the responsibility of the DE or DA encompasses enriching the table definition with data quality checks to execute. Additionally, any relevant *custom expectations* are integrated into the code repository, ensuring comprehensive version control.

Essentially, the definition stage consolidates the alignment between stakeholder-defined expectations and concrete code, providing a structured foundation for subsequent validation stages.

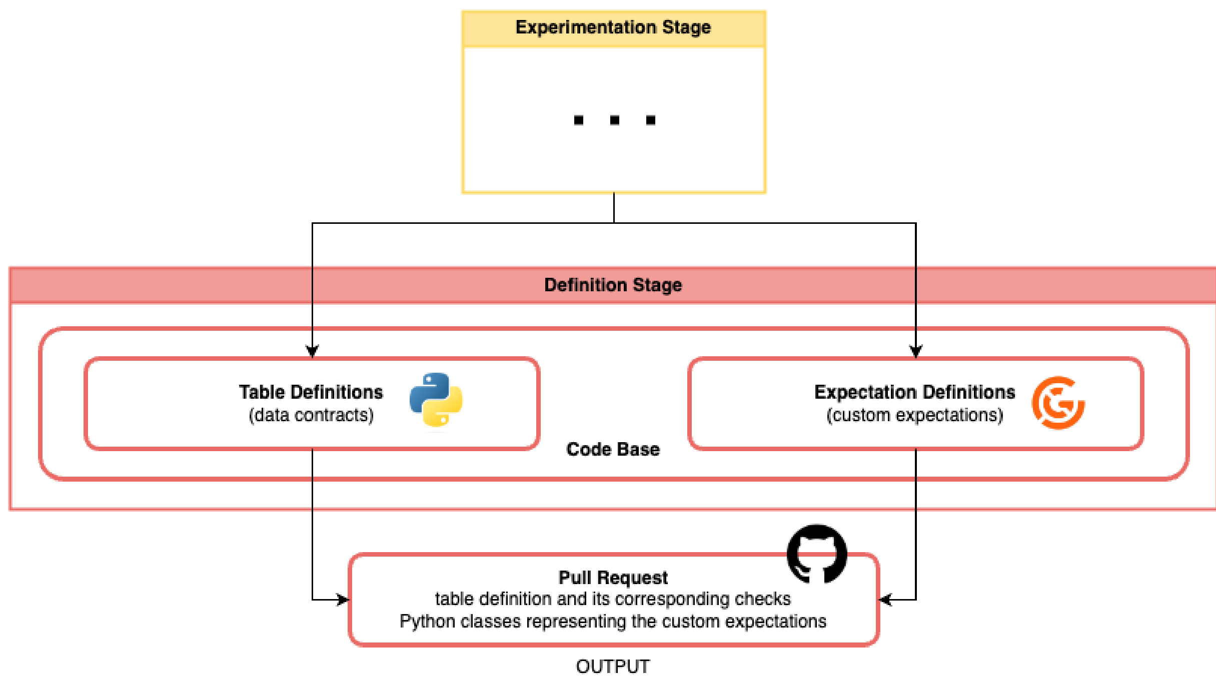

Figure 4.6: Definition Stage: add the set of data expectations and the new custom expectations to the code base.

Inference Stage

The effective execution of Great Expectations hinges upon furnishing it with information about which expectations to execute, their sequence, and the subsequent actions based on their outcomes. This vital information is conveyed through JSON files, encapsulating three primary components:

1. Custom Expectations;
2. Expectation Suites - set of expectations for a particular data asset;
3. Checkpoints - information on the order to validate the expectations and the *Actions* to take based on the results.

The system can deduce these artifacts either manually or programmatically. At BCDC, the data platform team opted for the latter approach, and the *Inference Stage* takes care of it. The framework dynamically infers these artifacts from the outputs generated during the definition stage. Specifically, the table definitions and Python classes embodying custom expectations become pivotal inputs. Once developers conclude the definition stage, they initiate a Pull Request on the repository to merge the new code. Subsequently, an automated Continuous Integration & Continuous Delivery (CI/CD) pipeline is triggered.

This pipeline executes code to derive JSON artifacts from the recently modified Python files, and afterward, it loads them into an object storage system (Figure 4.7).

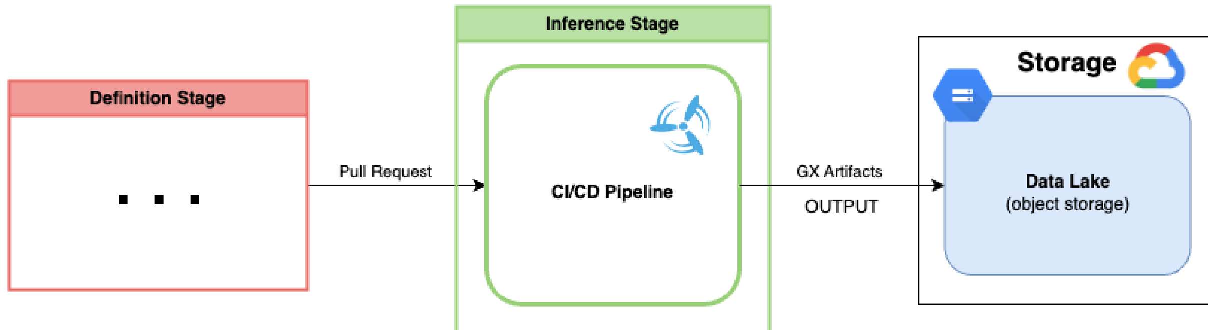

Figure 4.7: Inference Stage: the CI/CD pipeline triggers some logic to infer the GX artifacts starting from the Python files modified at the previous stage.

In essence, the inference stage exemplifies the fusion of development processes and automated mechanisms, translating Python-based custom expectations and table definitions into JSON artifacts that Great Expectations can seamlessly utilize in later stages.

Validation Stage

The heart of data asset quality assessment beats within the *Validation Stage*—the pinnacle of the framework. Even though this phase brings much value by fulfilling the system's core purpose—unearthing data anomalies—it demands meticulous integration with the ETL framework. The integration unfolds in two distinct ways:

1. Deep Integration - This approach involves embedding quality checks within the data pipelines. It entails a transition from the conventional ETL/ELT (Extract, Transform, Load) framework [30] to an ETLV/ELVT (Extract, Transform, Load, and Validate) paradigm. While this approach requires more extensive effort and attention, it provides enhanced pipeline monitoring and streamlined maintenance;

2. Two-Level Integration - The validation phase remains decoupled from the data pipelines. Quality checks occur separately from ETL job execution; thus, they entwin less with the ETL framework. Although more straightforward to implement, this approach sacrifices efficiency and efficacy compared to deep integration.

In the context of Blockchain.com's multi-layered data lakehouse, which predominantly operates in batch processing, engineers decided to opt for deep integration (Solution 1). This integration strategy was pursued for the initial and second data refinement layers. The data lakehouse comprises tiers representing distinct stages of data refinement. The

first layer retains the complete historical dataset to facilitate Slowly Changing Dimensions (SCDs) tracking [13], while the second layer houses the latest version of each record. Notably, these layers employ diverse storage solutions—object storage for the first and a cloud data warehouse for the second.

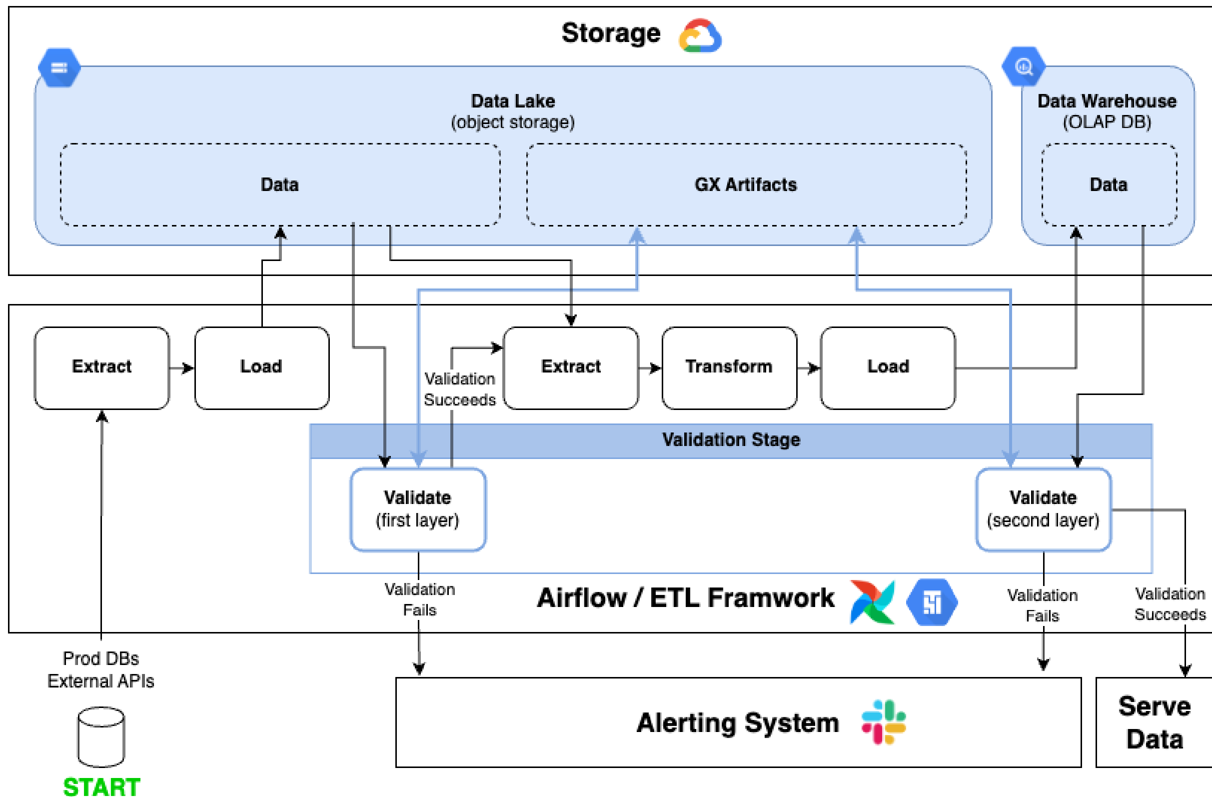

Figure 4.8: Validation Stage: integration of GX validation with the existing ETL/ELT framework.

To summarize, the validation stage finds its place within the ETL/ELT framework. At its endpoints, it gets the data from an external source—e.g., a production DB or an external API—and makes it available to data and product analysts. In between, the pipeline extracts the raw data and loads it—sans transformation—in the object storage; afterward, GX runs the expectation suite on the loaded data, producing the *validation results*, and then pushes them to the object storage system, irrespective of the outcome. These results enable insightful reports on the DQA Framework's performance (refer to "Monitoring Stage") and facilitate documentation for non-technical stakeholders (refer to "Documentation Stage").

Upon validation failure, the pipeline halts and triggers the Alerting System, which notifies the data owner. Conversely, if the system successfully validates the first layer, the pipeline

activates the second layer ETL job's extraction task. The framework then proceeds by extracting the data from the object storage, transforming it according to the requirements, loading it into the Cloud Data Warehouse, and validating the second layer. Again, Great Expectations verifies the data, and alerting mechanisms engage upon validation failure. Successful validation propels data provision to external users.

It is important to note that the solution presented in this section is a brief overview of the framework implemented at Blockchain.com. For instance, from the given description, the framework seems not to prevent the inconsistent data from propagating in the storage layer; the "load" task comes before the "validation." However, more advanced mechanisms are in place that leverage staging temporary tables and development instances of the databases to make the execution of the validation before the loading function possible. However, these sophisticated mechanisms are beyond the scope of this thesis.

Monitoring Stage

The *Monitoring Stage* (Figure 4.9) stands as a specialized phase thoughtfully requested by the Blockchain.com data platform team due to its paramount role in evaluating data quality trends within the data lakehouse.

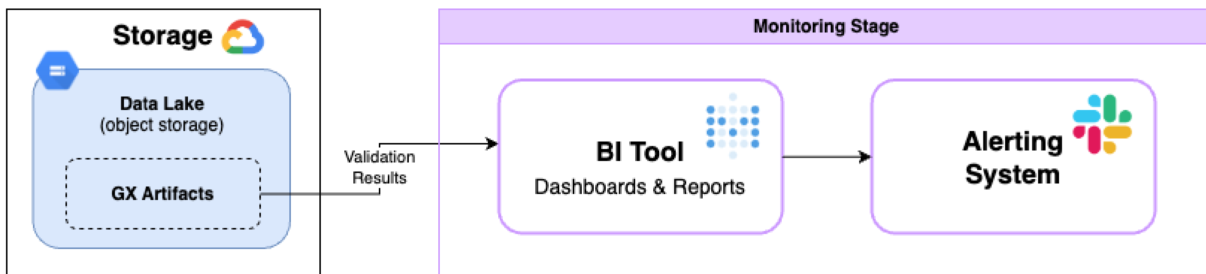

Figure 4.9: Monitoring Stage: implementation of reports based on the Validation Results.

This stage involves formulating comprehensive reports utilizing the Validation Results deposited into the Object Storage system. With each iteration of Expectation Suite validation, Great Expectations generates artifacts housing the validation outcomes. The system stores the results in JSON format, whether affirmative or adverse. Subsequently, these artifacts are the foundation for assembling aggregated and selectively filtered reports. These reports offer insights into data quality Quality trends spanning the entire data platform or specific clusters of Data Assets.

Furthermore, the incorporation of thresholds enriches the reporting landscape. When crossed, they serve as triggers, catalyzing functions, or procedures designed to notify relevant teams about anomalies or unconventional patterns manifesting within the platform.

To wrap up, the monitoring stage embodies a dynamic conduit for data-driven insights, empowering stakeholders to address DQ nuances and deviations proactively.

Documentation Stage

The *Documentation Stage* (Figure 4.10) was an explicit request, mainly from the financial reconciliation team, who recognized the significance of facilitating understanding for non-technical members. This team, encompassing individuals with diverse backgrounds, sometimes finds it challenging to navigate the intricacies of retrieving Validation Results files from an object storage system and comprehending their implications.

In response to this requirement, GX extends a solution—the GX Data Docs. Alongside the automatic generation of JSON files that encapsulate the outcomes of Data Asset batch Validations, the framework can export these results as user-friendly HTML files. These GX Data Docs, once harnessed by the Data Team, can be presented as a comprehensive web-based application. Each Expectation Suite and its associated Validation Results become effortlessly navigable through this avenue.

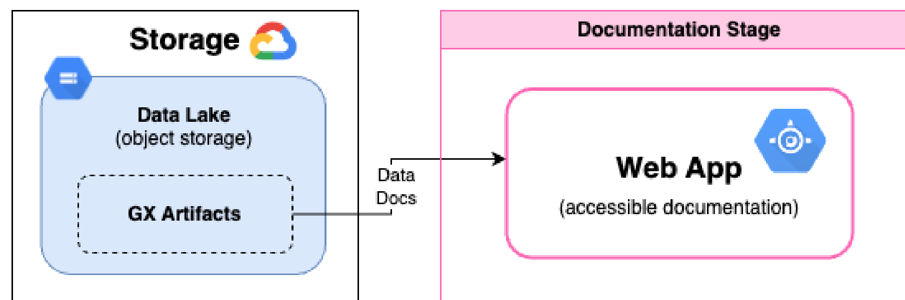

Figure 4.10: Documentation Stage: presentation of the Expectation Suites and the associated Validation Results in a readable way for non-technical people.

By adopting this approach, non-technical stakeholders gain lucid insights into the spectrum of expectations governing specific Data Assets. Furthermore, they access a lucid panorama of the validation outcomes, bridging the comprehension gap between technical intricacies and business implications.

Big Picture

The DQA solution presented in this section encapsulates seven distinctive stages and orchestrates a holistic approach to data quality assurance within the data architecture. This comprehensive journey, illustrated in Figure 4.11, addresses data challenges, catering to diverse organizational needs and stakeholder expectations.

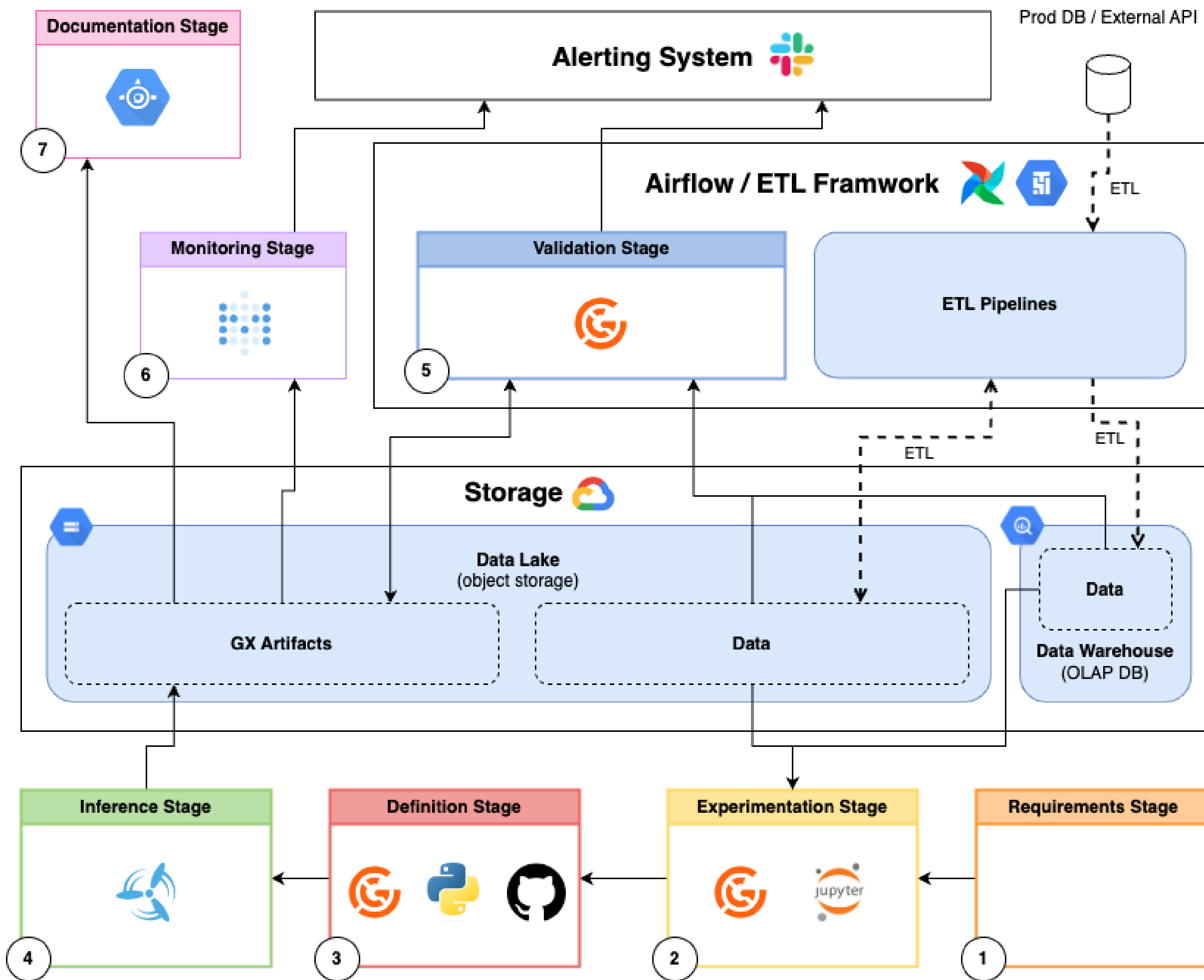

Figure 4.11: The Big Picture: all the framework stages combined.

Collectively, these stages epitomize a journey that transitions from initial expectations to an operationalized system deeply embedded in the ETL architecture. It underscores a paradigm shift towards not just passively identifying problems but also inculcating a proactive stance to detect data quality deviations. This framework delivers a tangible impact, aligning data practices with organizational excellence.

4.2. Data Contracts Solution

The second solution in the final framework involves the application of data contracts. While the previous section hinted at a form of data contracting, this section delves into the standard industry practice of implementing data contracts within an Enterprise Architecture.

Section 2.1 discussed two approaches to integrating data contracts into a system: detection

and prevention approaches. This section focuses on the preventive approach, where data contracts govern data at production before it enters the data platform. When effectively applied, this solution substantially impacts DQ as it allows for the early detection and prevention of data inconsistencies, sometimes even before productionizing the code.

In smaller organizations with limited teams, verbal agreements among stakeholders regarding data format and semantics may suffice. Effective communication can address cases where data producers deviate from these verbal contracts. However, verbal agreements become impractical as organizations grow with multiple teams, and automated data contracts become essential.

This approach shines in multi-source systems, especially within the same organization. In such systems, data contracts relieve the data team from daily interactions with software engineers working on upstream applications due to inconsistent data or schema incompatibility. issues The only drawback is that it demands a substantial effort on the producer side to set up the technology for defining, enforcing, and fulfilling the contracts. This effort may come from a SWE or a DE overseeing infrastructure. In either case, the service owner must embrace this approach and accept the increased infrastructure complexity.

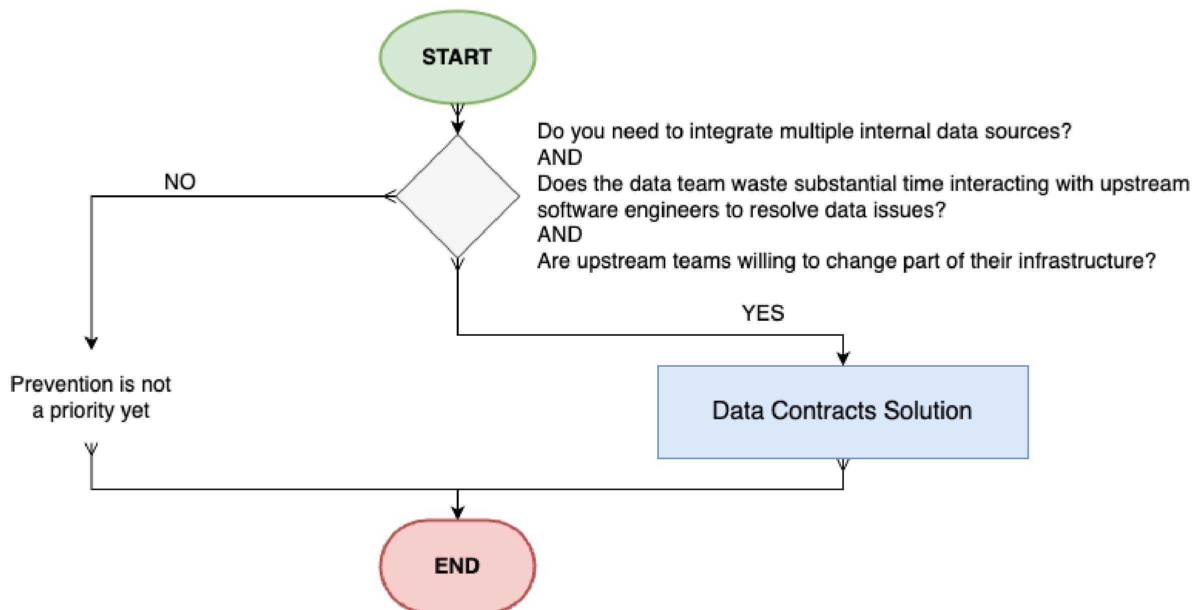

Figure 4.12: Second part of the final framework.

To determine if an organization could benefit from a Data Contracts solution, it is essential to answer two fundamental questions:

1. Does the data team waste substantial time interacting with upstream software engineers to resolve data issues? Does this impact their primary goal to extract valuable

insights from the data?

2. Are upstream teams willing to change part of their infrastructure, specifically the CI/CD pipeline?

If the answer is "yes" to both questions, the DQ of a company will surely benefit from a Data Contract Solution.

4.2.1. Context

This section explores a practical example of implementing a Data Contracts solution to illustrate its intricacies and benefits. Specifically, it examines the system described by Adrian Kreuziger and Chad Sanderson in their article, "An Engineer's Guide to Data Contracts," which details the infrastructure they implemented at Convoy to support DCs [25]. Their solution primarily focuses on identifying schema- and type-breaking changes before deployment.

It is important to note that various ways exist to integrate data contracts into a system, and this section explores one of them only. This thesis provides an overview of the advantages of handling inconsistent data and improving data quality, offering insights into the critical elements of a Data Contracts system without diving into exhaustive details.

4.2.2. Tooling

In terms of tools, the implementation centers around data contracts, which are essentially version-controlled code. Specifying contracts involves utilizing established open-source solutions designed for the serialization and deserialization of structured data. Two prominent projects, Google's Protocol Buffers (Protobuf) and Apache Avro, offer *Interface Definition Language* (IDL) support. These IDLs enable the formulation of event schemas, or contracts, in a language-agnostic manner. These schemas are the foundation for auto-generating code to serialize event payloads before publishing them to the Kafka platform.

The data source is a transactional database, but data extraction does not occur via a traditional DB connection but instead through Change Data Capture, a technology introduced in Section 1.1.3. CDC monitors source database logs for modifications and replicates these changes to the data platform, ensuring it always contains the latest data. Data is extracted as messages, propagated to Kafka, and finally, an ETL job retrieves and stores it in the analytics platform.

Another vital tool to bridge the gap between the theoretical concept of data contracts and their practical implementation is the *schema registry*, a central repository or database to store and organize the contracts. These contracts—or schemas—define the structure and format of data objects, ensuring that all data producers and consumers adhere to a standard set of rules when encoding and decoding data.

A *stream processing framework*—such as Apache Flink or kSQL—is another essential component within the proposed system, which is pivotal in maintaining data integrity and enforcing the data contract. These platforms serve as the real-time processing engine that takes the event data generated and pushed to the Kafka topic by the CDC solution and applies the data contract. It processes each incoming event, ensuring it meets the defined contract specifications.

Even though it is possible to implement a Data Contracts solution by simply combining these four technologies, it is essential to recognize that data contracts represent a cultural shift in engineering rather than merely amalgamating technical tools. This shift entails changes to the organization's vision of data products, aligning with modern approaches like the Data Mesh [3].

In summary, the development requires five key components: a cultural shift in the organization's data approach, an IDL to define contracts, a schema registry to store them, a CDC pipeline for data extraction, and one or more streaming processing jobs to enforce data contracts.

4.2.3. System and Workflow

The best way to present the final system design is to break it down into four different stages:

1. Definition Stage - During this stage, the stakeholders implement the streaming processing jobs and the contract in the form of code;
2. Enforcement Stage - This represents a fully automated process that runs after modifications to the codebase and checks that any changes to the underlying production service or contract will not break the system;
3. Fulfillment Stage - During this phase, the system makes sure that any event produced by the service adheres to the defined contract;
4. Monitoring Stage - This stage may coincide with the detection solution presented in the previous section, which is much more suitable to catch subtle changes in the

data semantics: the DQA framework.

Definition Stage

During the *Definition Stage*, data contract contributors, including software engineers, data engineers, and product owners, collaborate to define or update the data contract. They express this contract in code, utilizing Interface Definition Languages. In the case of Convoy, they employed Protobuf, Google's IDL (example provided by the Algorithm 4.1).

Algorithm 4.1 TransactionLeg Message Schema (Protobuf)

```
1: message TransactionLeg {
2:     required string id = 1;
3:     required string transactionId = 2;
4:     required string ownerId = 3;
5:     required Timestamp createdAt = 4;
6:     required double amount = 5;
7:     required string currency = 6;
8: }
```

Once contributors finalize the contract, they push it to a central codebase accessible to all. This action triggers a CI/CD pipeline that runs tests (see the "Enforcement Stage" for further details) to validate the new or updated contract's correctness. Typically, the repository selected for this function is the same one housing the code for the underlying production service.

During this stage, software engineers also have the additional task of defining the streaming job responsible for processing events to ensure their output aligns with the contract's expectations. Examining best practices when integrating a Change Data Capture solution into a system is essential to understand why a streaming processing job is necessary. In the "Fulfillment Stage," a CDC tool connects to the database logs and generates new events. Directly ingesting these events as they are into the data platform is not ideal because changes in the production service can break downstream pipelines. For instance, if a software engineer modifies the schema of an entity representing a forex transaction by splitting an existing `pair` column into `base` and `quote`, this alteration may cause issues downstream, where SQL queries rely on the `pair` column (example depicted in Figure 4.13). In summary, consuming CDC events directly tightly couples the production and consumer infrastructures [26], violating database encapsulation [19].

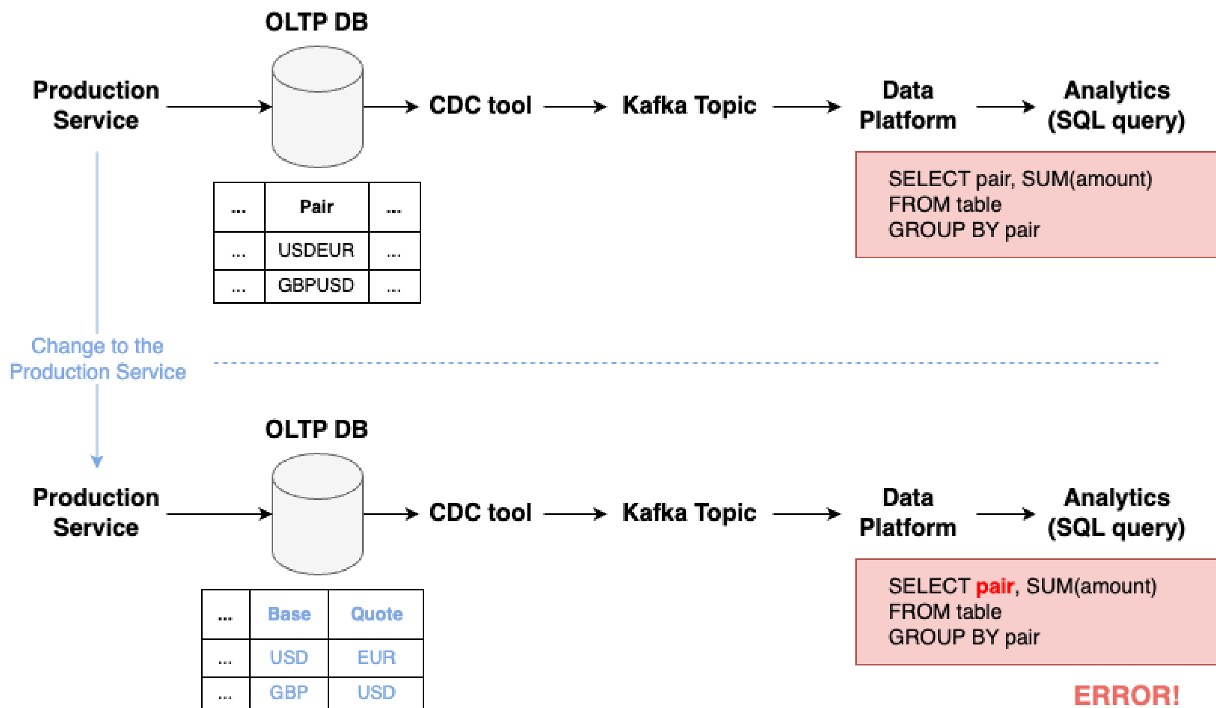

Figure 4.13: Tightly Coupled System: an example of how an upstream change requires a downstream change.

The solution to this problem is introducing an abstraction layer between the producer and consumer [19], leading to a loosely coupled architecture [18]. One of the most common approaches for doing so consists of publishing the CDC events collected from the logs of an OLTP DB into a private Kafka topic, which is only accessible by a streaming processing framework that will take care of transforming the event in such a way it meets the contract specifications. Therefore, during the definition stage, a software engineer must develop a job that processes CDC events to adhere to the data contract schema. In the forex transaction example, this solution ensures that even if the production service changes the internal logic and modifies the schema of the processed entity, the schema of the final event consumed by the ETL platform remains unchanged, preserving the functionality of downstream pipelines (Figure 4.14).

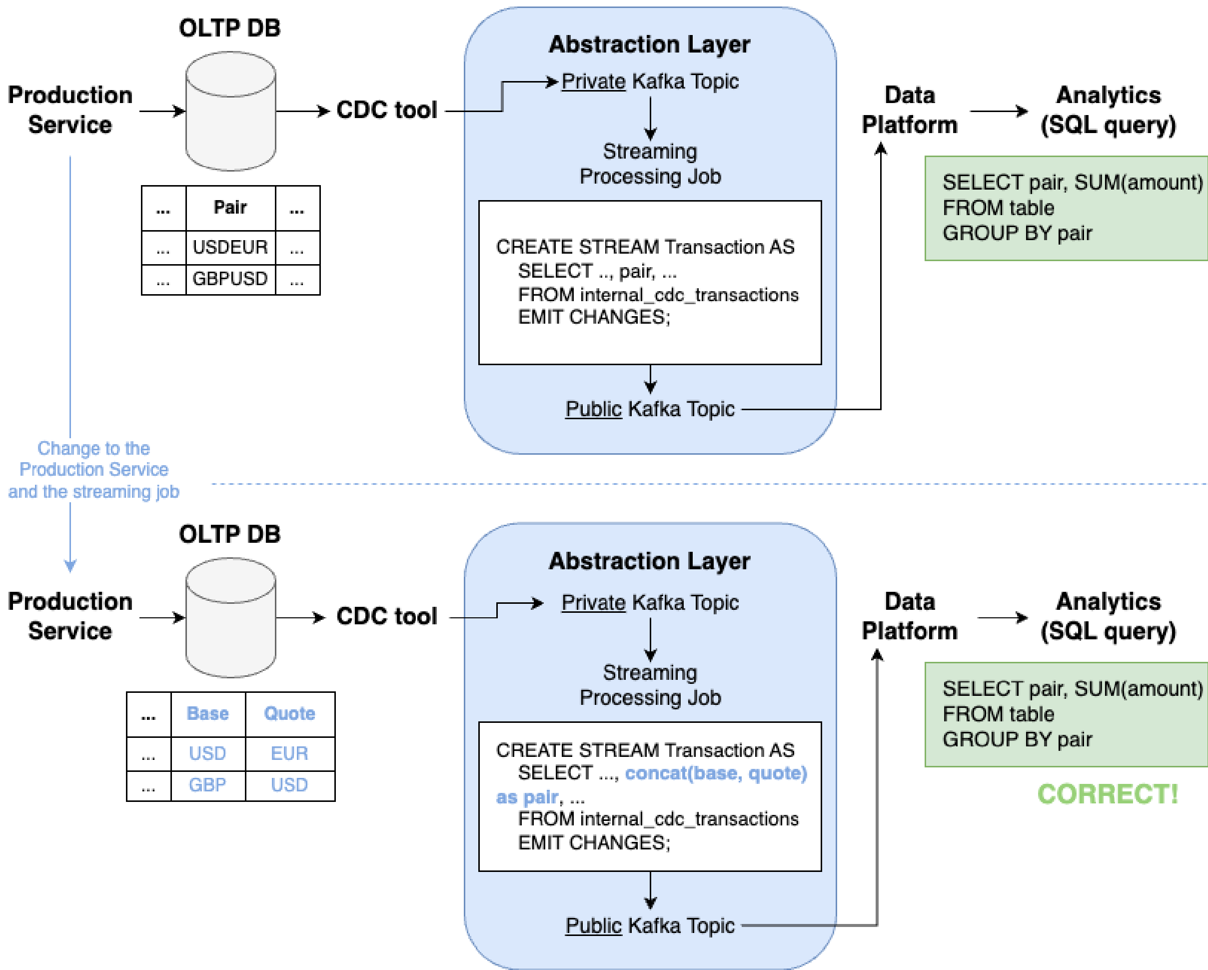

Figure 4.14: Loosely Coupled System: an example of how an upstream change does not require any downstream change thanks to the abstraction layer.

In essence, the definition stage encompasses these steps, as illustrated in Figure 4.15.

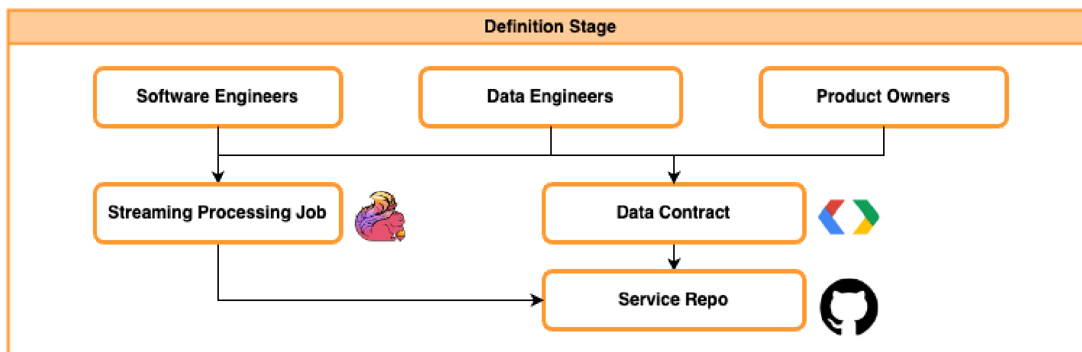

Figure 4.15: Definition Stage.

Enforcement Stage

The *Enforcement Stage* operates each time a Pull Request (PR) merges into the main branch of the codebase, forming an integral part of the CI/CD pipeline. It serves two primary objectives: validating that the updated streaming processing job fulfills the data contract and ensuring the new contract version is compatible with the previous one.

For what concerns the first objective—ensuring the processing job complies with the data contract—the process assumes the existence of a method to create a test database instance with an identical schema to the production database (as illustrated in Figure 4.16).

Since the test database mirrors the production schema, any PR that includes schema modifications will also exist in the test instance. Typically, when software engineers alter a schema, they must also update the processing job because it will likely receive different input events. How they update the job may not align with the contract requirements, potentially resulting in an output event with an inconsistent schema. Hence, an integration test is essential. This Integration Test relies on Docker Compose to instantiate the test database, the CDC pipeline to access the logs, private and public Kafka topics, the streaming processing platform, and a test schema registry instance.

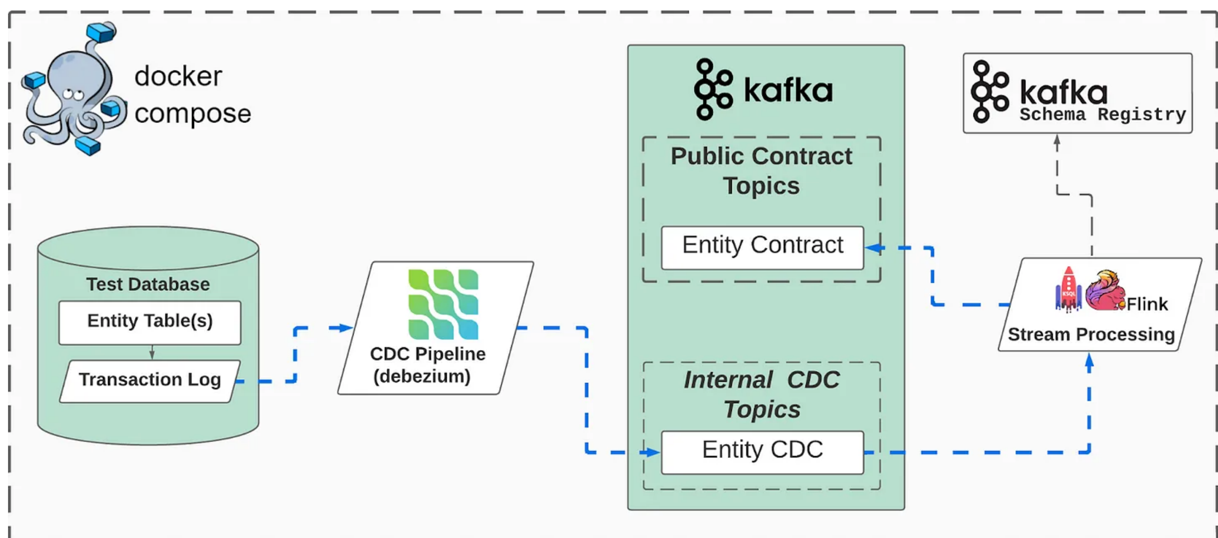

Figure 4.16: Integration tests to check whether the newly defined streaming processing job fulfills the Data Contract [25].

The CDC pipeline extracts an event with the new structure and pushes it to the private Kafka topic. The streaming processing job processes this raw event into an event with a structure that complies with the contract's specifications. At this point, the system can compare the output schema pushed to the test schema registry with the one defined

in the new contract version in the PR to ensure they match. If they are compliant, the CI/CD pipeline can go ahead by checking the new schema's compatibility with the previous version. Otherwise, the CI/CD fails, and the delivery of the new code stops waiting for the developers to fix the streaming job so that it complies with the defined data contract.

Schema compatibility constitutes the second objective, as depicted in Figure 4.17. Even if the new streaming job operates as expected, producing events that precisely match the schema defined in the new data contract, it does not guarantee that the schema defined in the new data contract is backward compatible; this is where the schema registry and its functionalities come into play. Schema compatibility is usually a built-in feature of schema registry tools, requiring the execution of the appropriate function to perform compatibility checks.

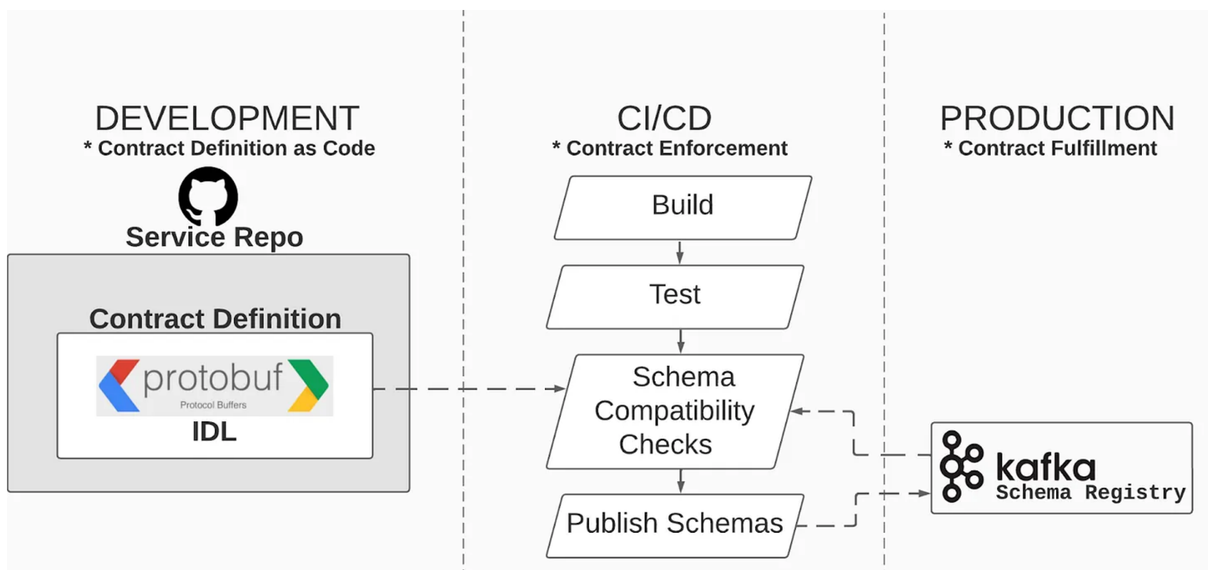

Figure 4.17: Schema Compatibility tests to check whether the changes made to the Data Contract are compatible with the previous versions [25].

Fulfillment Stage

Firstly, clarifying what "fulfilling a contract" means is essential. It refers to the actual performance of the obligations and promises outlined in the contract by the parties involved. Therefore, during this stage, the system ensures that the events produced by the CDC tool, reflecting changes made by the production service to the underlying entity, comply with the defined data contract.

The sequence of events (Figure 4.18) unfolds as follows. An entity event happens on the

user side, and the production service modifies the underlying entity in the database. For instance, an entity event could involve changing the status of an order from `to deliver` to `delivered` by altering the corresponding transactional database row. Subsequently, the CDC connector detects this change from the logs and translates it into a message pushed into a private Kafka topic. At this point, the stream processing framework, possibly an SQL-oriented one such as Apache Flink or kSQL, processes the messages in the private Kafka topic, delivering the output into the public Kafka topic. This public Kafka topic exclusively contains messages adhering to the data contract specifications. Thanks to the enforcement stage, all messages produced by the processing job and pushed into the public topic unquestionably meet the schema agreements, ensuring downstream consumers can access them in the predetermined format without causing pipeline disruptions.

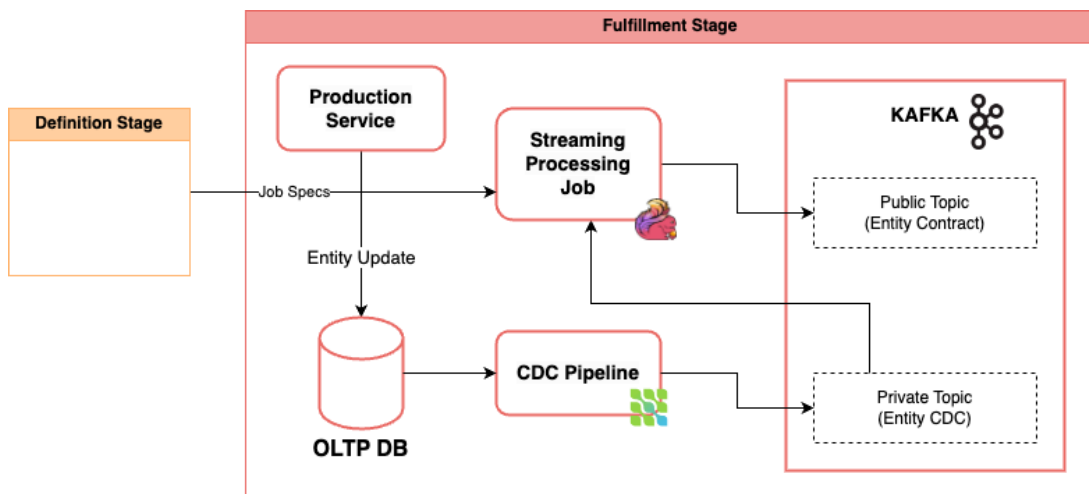

Figure 4.18: Fulfillment Stage.

Monitoring Stage

Despite rigorous testing procedures in software development, the inevitable reality remains that specific bugs manage to elude prevention and manifest themselves in the production environment. These elusive defects frequently hide subtle alterations in behavior, which, regrettably, do not readily trigger alarm bells on the production side. The same holds for data management. There exist certain facets of semantic enforcement that defy reliable control before deployment. While the presented system does make diligent efforts to enforce data integrity by explicitly implementing value constraints and employing statistical analyses in the testing and staging phases, the ultimate safeguard comes in the form of robust monitoring systems. These monitoring systems are indispensable, serving as vigilant sentinels that promptly notify the owners of any shifts in the data semantics, thereby ensuring the ongoing reliability and integrity of the systems.

Consequently, in the context of this Master's Thesis, it is imperative to underscore detection's pivotal role in maintaining data fidelity throughout the software development lifecycle; this further demonstrates how the Data Quality Assessment solution introduced in the previous section must be considered the atomic unit to ensure data quality in an organization.

Big Picture

The Data Contracts solution is a comprehensive framework to ensure data integrity and consistency throughout the software development lifecycle; hence, it prevents data issues from flowing into the data platform.

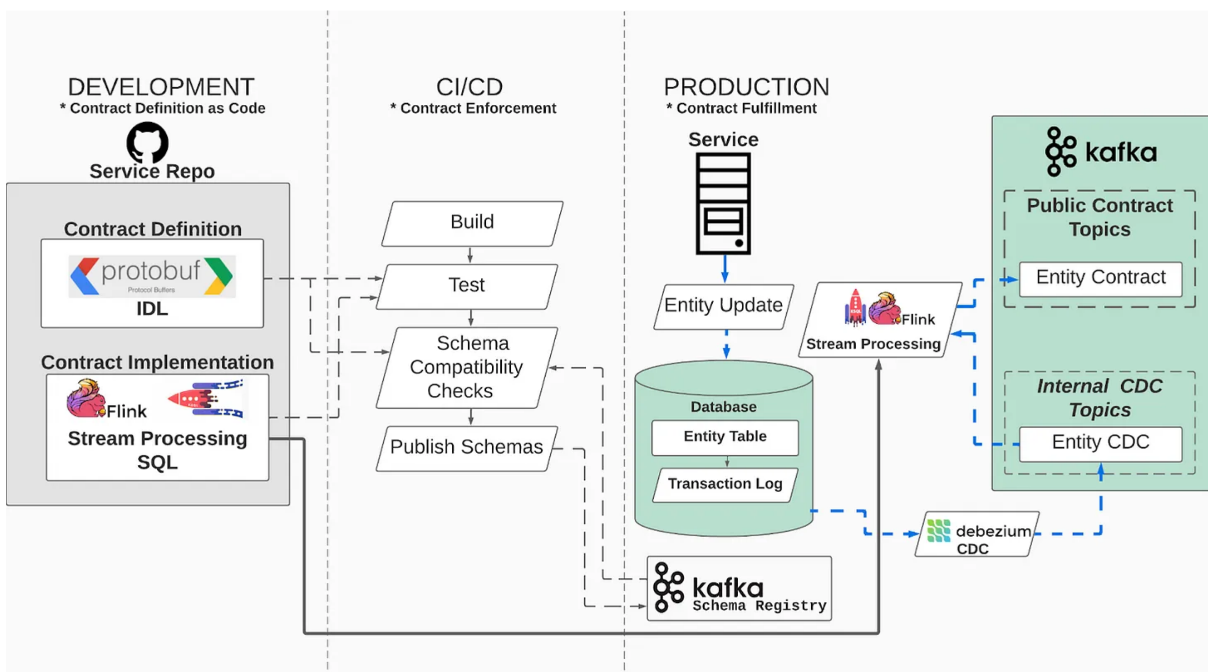

Figure 4.19: The Big Picture.

The Data Contracts solution encompasses four stages, offering a holistic approach to data governance, quality, and consistency. By addressing the data lifecycle from definition to enforcement and fulfillment with robust monitoring mechanisms, this framework provides a comprehensive solution to one of the most critical challenges in modern data-driven organizations.

4.3. Push API Solution

The third technological solution within the comprehensive framework involves a *Push API* approach. Similar to the earlier-discussed DC Approach, this solution addresses scenarios encompassing multiple data sources. Furthermore, it presupposes that these data sources belong to distinct organizations. This solution finds its niche in managing data quality within a subset of cross-enterprise collaboration scenarios.

However, the Push API Solution does not apply to any cross-enterprise collaboration scenarios. It necessitates specific conditions within the relationship between organizations. It mandates that the *central entity*, often the enterprise the data platform team belongs to, wield considerable influence over data sources and external organizations. This influence empowers the central entity to impose and implement a standardized Push API solution. If such influence is lacking, the feasibility of this solution diminishes.

Though catering to specific use cases, the Push API Solution proves remarkably effective when applicable. It offers the central entity the scalability needed to seamlessly integrate numerous intra- and cross-enterprise data sources.

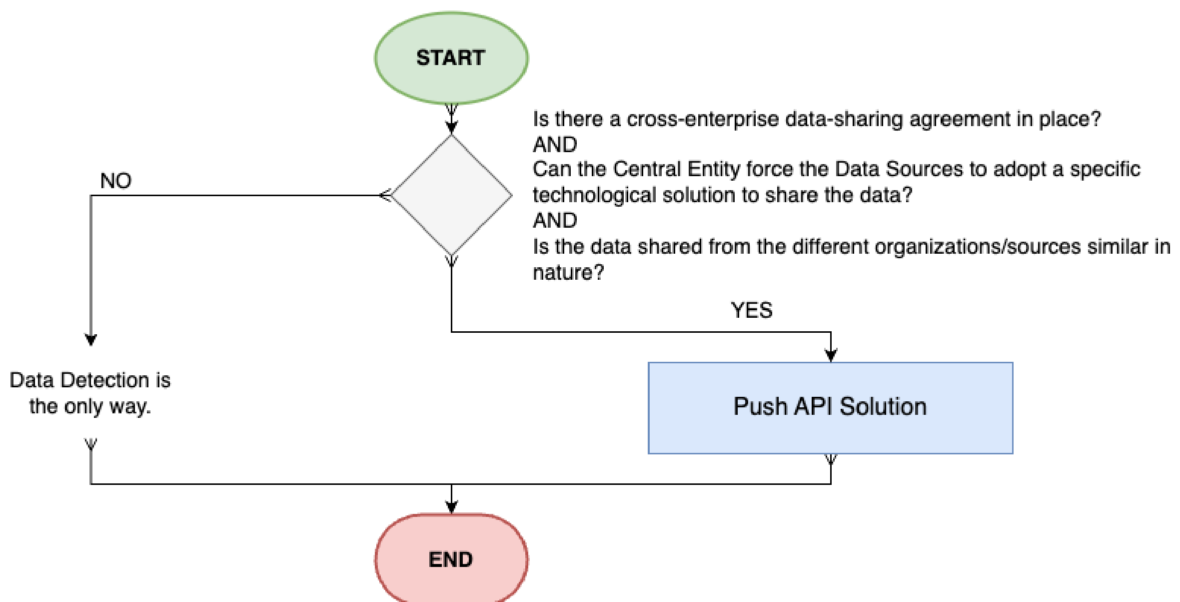

Figure 4.20: Third part of the final framework.

4.3.1. Context

The thesis highlights two application examples of the Push API Solution. One exemplifies a potential solution, while the other showcases an existing operational implementation.

1. The first use case centers around *Jordan's Tourism Analytics*. The Ministry of Tourism and Antiquities (MOTA) in Jordan collaborated with Politecnico di Milano to conceptualize an end-to-end system. This system aggregates data from diverse tourism-related sources, processes insights, and presents them as KPIs and dashboards, offering a comprehensive overview of the nation's tourism landscape.

2. The second case regards how each cash register in Italy shares data with the tax, payments, and customs authority—Agenzia delle Entrate—at the daily closing. In fact, as per the Legislative Decree of 05/08/2015 No. 127, subjects who carry out certain sales transactions (further info on what transactions at D.P.R. 633/1972) and who, through the use of appropriate cash registers, store them, are obliged to transmit telematically to the Agenzia delle Entrate the data relating to the daily receipts every day [4].

4.3.2. Tooling

As the term suggests, the foundational tool for implementing this solution is the API, specifically the Push API, as defined in Section 2.2. This approach operates on a push-based model, and thus, it entails data sources regularly transmitting updates to the consumer, in this case, the central entity. By deploying an API tailored to these needs, the central entity achieves the dual advantage of standardized communication and adherence to high DQ standards.

However, the central entity's involvement in designing the API does not negate the importance of collaboration with data sources. Effectively gathering the requirements is pivotal, as it is the foundation for a successful implementation. With well-converted requirements into code, the API can serve as a gatekeeper, screening data inconsistencies and ensuring only high-quality data flows into the data platform.

One drawback is the demand for data sources to interface with the API; this involves the implementation of connectors, data formatting, jobs scheduling, and automated transmission—a potentially substantial effort. For this reason, data providers often try to convince the central entity to adopt a Pull rather than a Push approach.

Even this time, it is possible to look at the Push API implemented by the central entity as a data contract. When fulfilled, it guarantees alignment with the requirements stipulated during the requirements collection phase, safeguarding data quality in a collaborative ecosystem.

4.3.3. System and Workflow

Limited access to the inner workings of these implementations restricts a comprehensive exploration of their system and workflow. Thus, this section provides a concise overview of two use cases rather than an exhaustive examination.

Jordan's Tourism Analytics

Jordan's Government collaborated with Politecnico di Milano to introduce data-driven decision-making into its tourism-related practices. PoliMI focused on the initial design phase of this technological solution, with the subsequent implementation left to the Ministry of Tourism and Antiquities.

The project required an end-to-end system. It had to establish connections with diverse data sources, some belonging to separate organizations. The system was to centralize data storage and processing, finally serving the data to a visualization system presenting KPIs and insights to end-users.

Stakeholders in this project represent five distinct groups (illustrated in Figure 4.21). Data Collectors and Providers, such as agencies, companies, and ministries, are the data source entities. Notable examples include the Border Police, the Central Bank of Jordan, and the Royal Society for Conservation of Nature (RSCN). The Ministry of Digital Economy and Entrepreneurship (MODE), commissioned by the Prime Minister, acts as an intermediary tasked with collecting provider data and distributing it to various consumers. One of the principal consumers of MODEE is the central entity's data platform, commissioned by the Ministry of Tourism and Antiquities, responsible for serving data to the visualization tool. The visualization tool, designed by Politecnico di Milano's Design Department, comprises metrics and charts presented through specific approaches to end Data Consumers. This category includes agencies, companies, small and medium enterprises (SMEs), and ministries that rely on insightful dashboards to derive actionable insights from concealed data, thus contributing to Jordan's tourism sector and its impact on the national economy.

Even though, during the designing phase, it was somehow ambiguous who between the MODEE and MOTA was responsible for collecting data from the providers, it is extremely interesting for the sake of this thesis how the entity in charge of it could achieve the goal of gathering information from several different organizations while keeping high data quality standards.

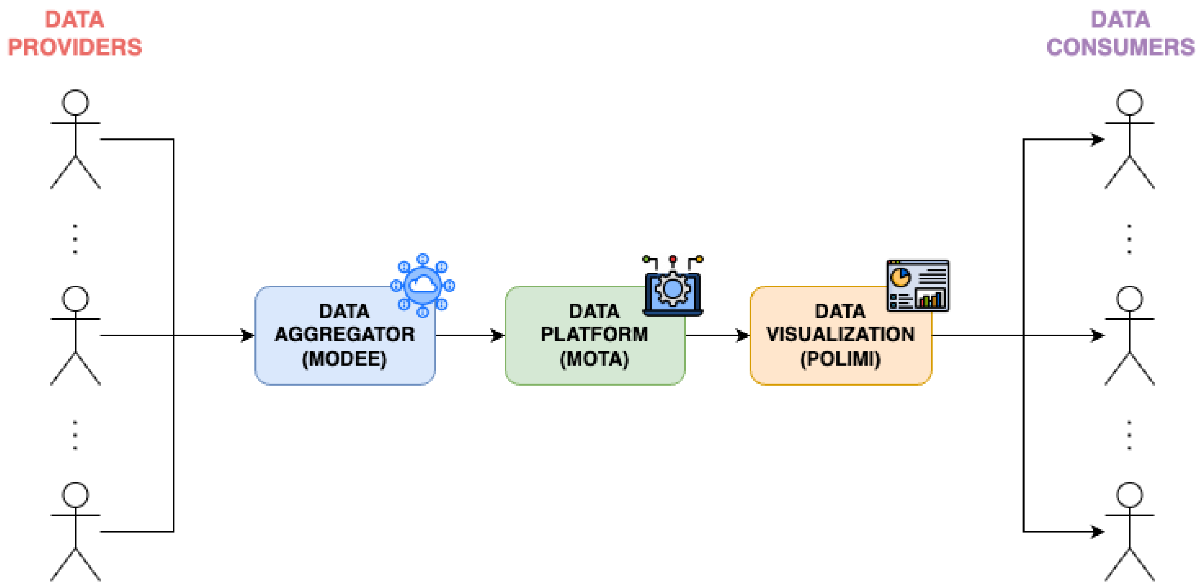

Figure 4.21: Jordan's Tourism Analytics System Architecture.

The Push API approach stands out as an efficient solution. Since the data-requesting entities—MODEE or MOTA—are all ministries closely linked to the Government, they might potentially influence data sources; this makes the Push API solution a promising key to the puzzle. Collaborating with data providers to define data types and exchange schedules would enable the implementation of an internal API. The data sources could then use this API to transmit data, allowing the central entity to ensure incoming data adheres to the requirements established during the collection phase.

Agenzia delle Entrate and Daily Recipts

The use case involving Agenzia delle Entrate offers a more tangible example. An operational Push API Framework is in place to ensure that all cash registers send information in the correct formats at the end of each day.

The common thread that links this practical case with the more theoretical case above is the role that the central entity, in this case, the Agenzia delle Entrate, has vis-à-vis sources. Even in this case, the latter is a governmental organization. Hence, it potentially can force all cash register manufacturers to develop firmware that formats daily data as specified by the API and submits it via a REST POST request according to the schedule defined by the law decree.

A PAPI approach is well-suited here due to both the central entity's authority and the underlying system characteristics:

1. Uniform data format - The data produced by all the sources closely resembles each other and is very easy to format. A few manufacturers can develop software that integrates with the API provided by the Agenzia delle Entrate because the underlying data remains consistent. This code reusability makes the effort worthwhile. In situations with varying data structures, this approach may be less suitable, requiring software extensions for each new data format;

2. Data Quality Priority - Given the financial nature of the data, data quality is paramount. The Agenzia delle Entrate might gain significant leverage over Data Providers by emphasizing the importance of data integrity. They can underscore how data quality safeguards citizens' financial interests, with the Push API approach being the sole means to ensure it.

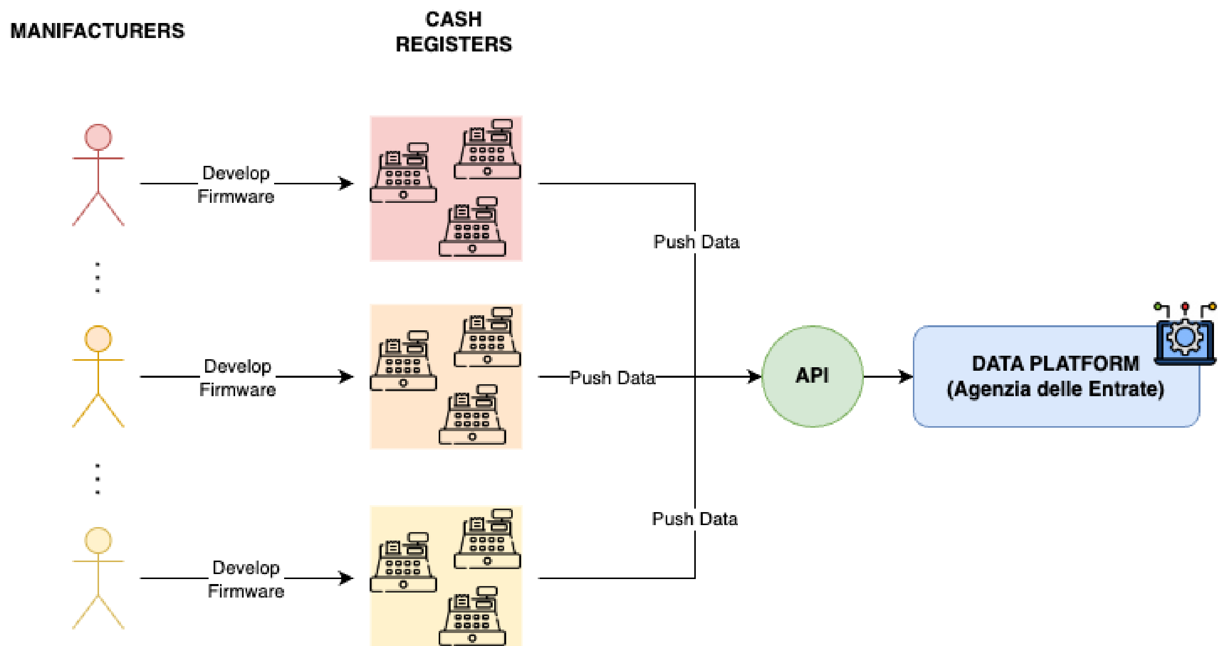

Figure 4.22: Agenzia delle Entrate System Architecture.

The use case of the Agenzia delle Entrate underscores the adaptability of the Push API solution in scenarios where data formats remain consistent and data integrity is paramount.

5 | Results Discussion

The preceding chapter systematically elucidated a framework that addresses the multifaceted challenges of upholding data quality standards within the complex milieu of data architecture. This chapter serves as the crucible wherein the disparate elements of our framework coalesce into a cohesive whole, forging a unified vision that harmonizes the three distinct technological solutions articulated earlier. In essence, the primary aim of this chapter is to consolidate these three integral facets into a comprehensive and comprehensible framework, providing a lucid perspective on the contribution it imparts to the domain of data quality management.

To do so, it is first necessary to introduce three essential concepts often mentioned by the previous chapters, which require a more profound detailing to fully understand the framework's contribution to the data quality landscape.

5.1. Proactive VS. Reactive

The organization's most effective way to move towards high data quality standards is through a cultural shift in looking for data inconsistencies. The most common approach organizations adopt is *reactive*; this means they wait for data analysts to start working on new projects and find issues with the information they contain. When they encounter problems or find out the data does not respond to their business questions, they usually react by contacting engineers who previously worked on the underlying pipelines. Often, these engineers have already moved on to other projects and usually do not have time to address their requests; additionally, it might be hard for them to get back to the details of those pipelines, especially when not adequately documented [23].

This reactive approach can impede data teams in extracting valuable insights from raw data. A *proactive* mindset is essential to enhance efficiency. Automating data checks and discovering issues is critical to proactively improving data quality. The framework introduced in the previous chapters offers three technological solutions that guide organizations toward proactive data issue discovery. Its contribution demonstrates how organizations

can elevate data quality by adopting a proactive mindset, paramount to achieving this objective.

5.2. Prevention VS. Detection

In addition to the proactive VS. reactive contrast, exploring the juxtaposition between prevention and detection is also interesting, as it is fundamental in data quality. The choice between these approaches depends on the organization's and the problem's specific context.

Detection mechanisms, primarily used for checking data semantics, are crucial and should always be in place. While organizations can set up detection systems to react to issues as they arise, a proactive approach, facilitated by an automatic system like the Data Quality Assessment framework, is far more efficient. Proactive detection is a cornerstone of the comprehensive framework discussed throughout this thesis. Organizations are encouraged to implement proactive detection as soon as they reach an appropriate stage of data maturity.

Larger organizations with a diverse range of products may require more than focusing on detection only. Detection identifies issues, but someone must resolve them. When issues reside downstream of the ETL jobs, they are straightforward to solve; however, if the problems live upstream in the data engineering lifecycle, they might involve data engineers and software engineers responsible for the service. Such discussions and the consequent resolution can be time-consuming. Even if they require a significant initial effort to build, *prevention mechanisms* would help in these cases offering one pivotal advantage: they can prevent code changes that would disrupt underlying data pipelines, especially those involving schema and column type alterations. As previously discussed, a downstream detection system remains the only viable and effective option for semantic issues.

5.3. Single- VS. Cross-enterprise

The final distinction worth mentioning pertains to single and cross-enterprise platforms. In most organizations, data is primarily extracted from internal production databases, keeping data sources within a *single enterprise*. In such cases, the best approach to ensure high data quality involves a combination of preventive measures, focusing on schema and type issues upstream—e.g., the DC Solution—and downstream detection mechanisms for semantic issues—e.g., the Data Quality Assessment system.

However, there are scenarios where an entity—the central entity—needs to extract data from various external organizations, creating a *cross-enterprise data collaboration*. In these cases, particularly when the central entity can influence the external sources to align with its systems, one solution to address DQ-related problems consists of a prevention mechanism via API and, more precisely, through an API relying on a push-based approach. Additionally to the role and influence of the central entity, the nature of the underlying data must be considered. Indeed, suppose the structure of the data shared by the external providers varies from one to another. Implementing a REST API to integrate all such different formats might be very complex in that case.

Conversely, if the underlying data shares a typical structure over all data sources, it is much easier to develop a single API that works identically for all the sources and checks for consistency with the specifications established during the requirements collection phase. Nevertheless, an organization can regroup the data sources based on the nature and structure of the data they provide and then develop one Push API solution per group. Multiple implementations are possible.

In practice, scenarios can be a blend of single-enterprise and cross-enterprise contexts. Organizations often ingest data from internal production databases and external sources—e.g., a CRM. This hybrid case is very possible in the real world. This hybrid situation highlights how particular cases might need the implementation of both solutions—DC and PAPI—as each approach addresses specific data quality challenges.

One last thing it is worth mentioning is that operating within a single-enterprise context does not necessarily equate to utilizing a singular data source. A data platform within a single enterprise may in fact ingest data from multiple internal sources. Thus, a crucial distinction must be made between the number of organizations participating in a data-sharing agreement and the number of data sources contributing to an analytics platform.

5.4. Comprehensive Framework

Figure 5.1 illustrates the final framework, presenting a unified flowchart that guides users based on responses to specific questions. Each *decision symbol* corresponds to a set of questions, and the final decision is derived by combining answers logically via an `AND` operation.

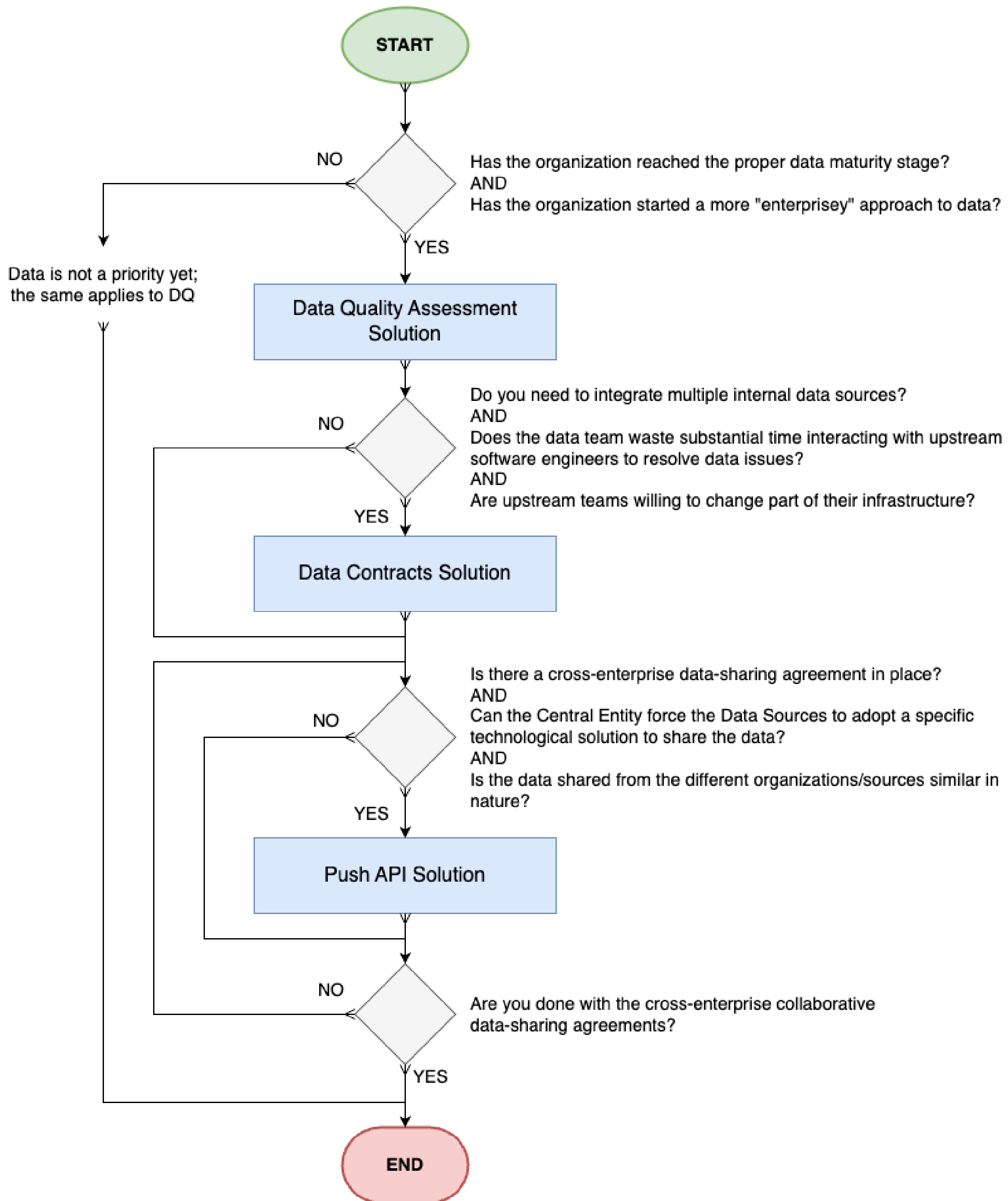

Figure 5.1: Comprehensive Framework in the form of a flowchart.

Analyzing the framework top-down makes it possible to notice that the Data Quality Assessment solution is the catalyst of the flowchart: if implemented, it triggers a series of events, including the eventual implementation of other tech solutions to address DQ issues; on the other side, not implementing it intrinsically means not having reached the correct data maturity stage, and hence, it is a stalemate.

By moving forward in the flowchart, a user encounters the Data Contract solution, which can be implemented—at most once—as not; this comes intrinsically with the fact that it is mainly adopted to prevent schema and type inconsistencies in single-enterprise scenarios. When integrated, it complements the work of the DQA framework, reducing data quality risks.

Finally, the last decision symbol introduces a loop for scenarios involving multiple collaboration agreements where the participating organizations manage different data in format and semantic meaning. In such cases, multiple Push API systems can be developed to accommodate varying data formats, effectively addressing the intricacies of cross-enterprise data integration.

6 | Conclusions

This thesis presents a comprehensive framework for enhancing data quality in multi-source analytics platforms across various organizational contexts. The framework, rooted in a proactive approach and blending prevention and detection techniques, offers organizations a decision-making tool in the form of a flowchart for selecting appropriate technological solutions tailored to their specific requirements.

The initial section of this thesis underscores the significance of data quality and the challenges posed by its maintenance in multi-source analytics platforms. It elucidates the concept of DQ while emphasizing the roles of data producers, custodians, and consumers in safeguarding it. Additionally, it highlights the fundamental issue of "Garbage In, Garbage Out," underlining the direct connection between input data quality and the dependability of subsequent analysis.

Subsequently, this thesis proposes a comprehensive framework to bolster DQ by customizing technological solutions to the unique dynamics between data producers and consumers. The framework introduces three technological solutions: a Data Quality Assessment solution, which is considered the atomic unit and applies to any scenario, Data Contracts tailored for intra-organizational contexts, and a Push API designed for cross-enterprise collaborations. Real-world implementations of the framework underscore its efficacy in addressing DQ challenges.

6.1. Limitations

Despite its contributions, the framework presented herein bears certain limitations. Firstly, it relies on three technological solutions that may only encompass part of the spectrum of scenarios within complex data ecosystems. While these solutions effectively address common data quality challenges, intricate or unique scenarios may need more research.

Secondly, the framework presupposes influence and control over data sources, which may only sometimes hold, particularly in cross-enterprise settings. This assumption may restrict the framework's applicability when the central entity lacks substantial control over

external data sources, impacting the overall framework's versatility.

The framework focuses on multi-source situations, leaving unexplored terrain in multi-tenant environments. This emphasis may curtail its relevance in scenarios characterized by complex N-to-M relationships between sources and consumers. Extending the framework to encompass multi-tenant scenarios and developing strategies for data quality maintenance in these settings could enhance its practicality and comprehensiveness.

Lastly, the framework accentuates a proactive approach to data quality management, which can demand significant effort and resources for implementation, particularly in establishing the requisite technology for defining, enforcing, and fulfilling data contracts.

6.2. Future Work

The framework proposed in this thesis offers a potent decision-making tool for guiding users in selecting appropriate technological solutions to uphold data quality in multi-source analytics platforms. While the current framework yields valuable insights and solutions, several promising avenues for future research and expansion can amplify its scope and relevance.

Two primary avenues for enhancing the framework are evident. Firstly, extending its coverage to encompass a more comprehensive array of collaborative scenarios is a crucial dimension for improvement. The existing framework presents only three technological solutions, thus addressing only a fraction of the diverse collaboration dynamics that entities may encounter. By incorporating additional decision pathways and corresponding technological solutions, the framework could better cater to intricate interactions, ensuring the maintenance of data quality standards across diverse partnership models.

Secondly, it might be worth broadening the current focus on multi-source situations to include multi-tenant environments. The extension of the framework to incorporate multi-consumer scenarios, particularly those characterized by complex N-to-M relationships, holds significant promise. Investigating how to preserve data quality standards in such intricate settings represents a compelling avenue for future research. By designing decision branches tailored to multi-tenant interactions and the associated data quality challenges, the framework could aid organizations in navigating the complexities of sharing and maintaining high-quality data among diverse consumers.

In addition to these principal directions, the "data contract" concept warrants special attention. Ultimately, all proposed solutions trace back to implementing contracts governing data exchange terms between specific entities. Exploring various forms of implementing

such data contracts can yield valuable insights and results.

Bibliography

[1] W. Badr. 6 different ways to compensate for missing values in a dataset (data imputation with examples). https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779, Jan 2019.

[2] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino. Methodologies for data quality assessment and improvement. *ACM computing surveys (CSUR)*, 41(3):1–52, 2009.

[3] Z. Dehghani. *Data Mesh*. O'Reilly Media, Inc., Mar 2022.

[4] P. della Repubblica Italiana. Trasmissione telematica dei dati dei corrispettivi. *Gazzetta Ufficiale*, 190(2), Aug 2015.

[5] J. Densmore. *Data Pipelines Pocket Reference*. O'Reilly Media, Inc., Feb 2021.

[6] J. Dixon. Pentaho, hadoop, and data lakes. https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/#comments, Oct 2010.

[7] S. Earley, D. Henderson, and D. M. Association. *Dama-Dmbok : Data Management Body of Knowledge*. Bradley Beach New Jersey: Technics Publications, second ed. edition, 2017.

[8] E. Eryurek, U. Gilad, V. Lakshmanan, A. Kibunguchy-Grant, and J. Ashdown. *Data Governance: The Definitive Guide*. O'Reilly Media, Inc., Mar. 2021.

[9] M. Felici, T. Koulouris, and S. Pearson. Accountability for data governance in cloud ecosystems. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, volume 2, pages 327–332, 2013.

[10] T. Handy. The modern data stack: Past, present, and future. https://www.getdbt.com/blog/future-of-the-modern-data-stack/, Dec 2020.

[11] M. Heller. What is oltp? the backbone of ecommerce. https://www.infoworld.com/article/3650769/what-is-oltp-the-backbone-of-ecommerce.html, Feb 2022.

[12] W. Inmon. *Building the Data Warehouse*. Wiley, Oct 2005.

[13] R. Kimball. Slowly changing dimensions. *DM Review*, 18(9), Sept 2008.

[14] D. M. A. U. Kingdom. The six primary dimensions for data quality assessment. https://www.sbctc.edu/resources/documents/colleges-staff/commissions-councils/dgc/data-quality-deminsions.pdf, Oct 2013.

[15] A. Maydanchik. *Data Quality Assessment*. Technics Publications, LLC, August 2007.

[16] B. Moses, L. Gavish, and M. Vorwerck. *Data Quality Fundamentals*. O'Reilly Media, Inc., Sept 2022.

[17] A. Petrella. Datastrophes: Il buono, il brutto, il cattivo. https://andy-petrella.medium.com/datastrophes-il-buono-il-brutto-il-cattivo-c0bad03e8dcb, Mar 2021.

[18] J. Reis and M. Housley. *Foundamentals of Data Engineering*. O'Reilly Media, Inc., June 2022.

[19] C. Riccomini. Kafka change data capture breaks database encapsulation. https://cnr.sh/essays/kafka-change-data-capture-breaks-database-encapsulation, Nov 2018.

[20] D. Romano. How to monitor data lake health status at scale. https://towardsdatascience.com/how-to-monitor-data-lake-health-status-at-scale-d0eb058c85aa, Oct 2022.

[21] C. Sanderson. The rise of data contracts. https://dataproducts.substack.com/p/the-rise-of-data-contracts, August 2022.

[22] C. Sanderson. How scale kills data teams. https://dataproducts.substack.com/p/how-scale-kills-data-teams, May 2023.

[23] C. Sanderson and M. carlo Data. Is modern data warehouse architecture. https://www.montecarlodata.com/blog-is-the-modern-data-warehouse-broken/, Apr 2022.

[24] C. Sanderson and D. Dicker. Data contracts for the warehouse. https://dataproducts.substack.com/p/data-contracts-for-the-warehouse, Jan 2023.

[25] C. Sanderson and A. Kreuziger. An engineer's guide to data contracts - pt. 1. https://dataproducts.substack.com/p/an-engineers-guide-to-data-contracts, Oct 2022.

[26] K. Sandoval. The difference between tight and loose coupling. https://nordicapis.com/the-difference-between-tight-coupling-and-loose-coupling/, Aug 2020.

[27] J. Y. Shi. The scalability dilemma and the case for decoupling. https://www.hpcwire.com/2016/03/30/scalability-dilemma-and-case-for-decoupling/, Mar 2016.

[28] R. Stenson. Is this the first time anyone printed, 'garbage in, garbage out'? https://www.atlasobscura.com/articles/is-this-the-first-time-anyone-printed-garbage-in-garbage-out, March 2016.

[29] D. M. Strong, Y. W. Lee, and R. Y. Wang. Data quality in context. *Communications of the ACM*, 40(5):103–110, 1997.

[30] D. M. Tayade. Comparative study of etl and e-lt in data warehousing. *Int. Res. J. Eng. Technol*, 6:2803–2807, June 2019.

[31] T. Tereshko. Separation of storage and compute in bigquery. https://cloud.google.com/blog/products/bigquery/separation-of-storage-and-compute-in-bigquery, Nov 2017.

[32] Y. Wand and R. Y. Wang. Anchoring data quality dimensions in ontological foundations. *Commun. ACM*, 39(11):86–95, nov 1996. ISSN 0001-0782.

[33] R. Y. Wang and D. M. Strong. Beyond accuracy: What data quality means to data consumers. *Journal of Management Information Systems*, 12(4):5–33, 1996.

[34] M. A. Zaharia, A. Ghodsi, R. Xin, and M. Armbrust. Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics. In *Conference on Innovative Data Systems Research*, 2021.

List of Figures

1.1 Key concepts and undercurrents that form the data engineering lifecycle . 7
1.2 Data Engineer's upstream stakeholders 8
1.3 Technological components, storage systems, and storage abstractions . . . 13
1.4 Role of the ingestion stage within the whole data engineering lifecycle . . . 16

2.1 Data contracts for Transactional Databases 26
2.2 Data contracts for data platforms 26
2.3 API Pull approach 27
2.4 API Push approach 28

3.1 Technical Stakeholders of Data Engineering 35
3.2 Detection VS. Prevention 36

4.1 Technological solutions involved in the framework 39
4.2 Approaches to data quality based on the data maturity stage 40
4.3 First part of the final framework 41
4.4 DQA Solution - the Requirements Stage 44
4.5 DQA Solution - the Experimentation Stage 45
4.6 DQA Solution - the Definition Stage 47
4.7 DQA Solution - the Inference Stage 48
4.8 DQA Solution - the Validation Stage 49
4.9 DQA Solution - the Monitoring Stage 50
4.10 DQA Solution - the Documentation Stage 51
4.11 DQA Solution - the Big Picture 52
4.12 Second part of the final framework 53
4.13 Tightly Coupled System 57
4.14 Loosely Coupled System 58
4.15 DC Framework - Definition Stage 58
4.16 DC Framework Integration tests 59
4.17 DC Framework Schema Compatibility tests 60
4.18 DC Framework - Fulfillment Stage 61

4.19 DC Framework - the Big Picture 62
4.20 Third part of the final framework 63
4.21 Jordan's Tourism Analytics System Architecture 66
4.22 Agenzia delle Entrate System Architecture 67

5.1 Comprehensive Final Framework 72

List of Tables

1.1 Different types of sources and the data formats they produce 11

3.1 Quality dimensions and how a system can ensure them 32

4.1 Great Expectations Glossary 43

List of Acronyms

AE Analytics Engineer. 44

BCDC Blockchain.com. 41

CDC Change Data Capture. 17

CI/CD Continuous Integration & Continuous Delivery. 47

DC Data Contract. 4

DE Data Engineer. 34

DQ Data Quality. 1

DQA Data Quality Assessment. 3

ETL Extract Transform Load. 14

GX Great Expectations. 29

IDL Interface Definition Language. 54

MODE Ministry of Digital Economy and Entrepreneurship. 65

MOTA Ministry of Tourism and Antiquities. 64

OLTP Online Transaction Processing. 9

PAPI Push API. 4

SWE Software Engineer. 34