UNIVERSITAT D'ALACANT
UNIVERSIDAD DE ALICANTE
Facultat de Ciències Econòmiques i Empresarials
Facultad de Ciencias Económicas y Empresariales

MASTER IN ECONOMICS WITH DATA SCIENCE

ACADEMIC YEAR 2025 - 2026

FROM QUALITY DIMENSIONS TO VALIDATION RULES: DESIGNING AN INTEGRATED DATA QUALITY SYSTEM FOR A MANAGEMENT CONTROL PIPELINE

NEUS SIGNES PEDRO

ROSA MARÍA BOTELLA ORTS
DEPARTAMENTO DE FUNDAMENTOS DEL ANÁLISIS ECONÓMICO

San Vicente del Raspeig, 09/2026

Abstract

Reporting systems for management control depend critically on the quality of the data that feed them. In departments that operate without a unified data model or systematic validation mechanisms, errors typically become visible only once they have reached a final report and may already have affected decisions. This study designs and academically justifies an automated data pipeline with an integrated data quality validation system, intended to detect and report records that violate the implemented validation rules before the data reach the reporting layer. Consistent with a design science orientation, the work develops an artefact rather than an empirical study: an ELTL architecture that preserves an independently retained raw layer for traceability, a dimensional star schema at atomic grain for the reporting layer, a daily batch execution cycle, and a catalogue of six declarative validation rules—currency, uniqueness, completeness, format conformance, referential integrity, and value range—each grounded in a data quality concept established in the literature and operationalized as a deterministic pass/fail check. The rules are executed in a fixed fail-fast sequence and complemented by a retention, notification, and persistent logging mechanism. The resulting behaviour is illustrated using a simulated dataset with deliberately introduced violations. The contribution lies not in the specific catalogue, which is context-specific, but in the documented procedure leading from business rules known a priori to an ordered set of validations integrated into a defined data architecture, together with an explicit statement of its scope and limitations.

Keywords: data quality; validation rules; data pipeline; ELTL architecture; dimensional modelling; management control reporting

Resumen

Los sistemas de reporting para el control de gestión dependen fundamentalmente de la calidad de los datos que los alimentan. En los departamentos que operan sin un modelo de datos unificado ni mecanismos de validación sistemáticos, los errores suelen hacerse visibles únicamente una vez que han llegado a un informe final y pueden haber afectado ya a la toma de decisiones. Este estudio diseña y justifica académicamente un pipeline de datos automatizado con un sistema integrado de validación de la calidad de los datos, cuyo objetivo es detectar y notificar los registros que incumplen las reglas de validación implementadas antes de que los datos lleguen a la capa de reporting. En consonancia con una orientación basada en la ciencia del diseño (Design Science), el trabajo desarrolla un artefacto en lugar de realizar un estudio empírico: una arquitectura ELTL que conserva una capa raw independiente para garantizar la trazabilidad, un esquema dimensional en estrella con granularidad atómica para la capa de reporting, un ciclo de ejecución diario por lotes y un catálogo de seis reglas declarativas de validación —actualidad, unicidad, completitud, conformidad del formato, integridad referencial y rango de valores—, cada una de ellas fundamentada en un concepto de calidad de los datos establecido en la literatura y operacionalizada como una comprobación determinista de tipo pasa/no pasa. Las reglas se ejecutan siguiendo una secuencia fija de tipo fail-fast y se complementan con un mecanismo de retención, notificación y registro persistente. El comportamiento resultante se ilustra mediante un conjunto de datos simulado en el que se han introducido deliberadamente diversas infracciones. La contribución del trabajo no reside en el catálogo específico de reglas, que es dependiente del contexto, sino en el procedimiento documentado que permite pasar de unas reglas de negocio conocidas a priori a un conjunto ordenado de validaciones integradas en una arquitectura de datos definida, junto con una delimitación explícita de su alcance y sus limitaciones.

Palabras clave: calidad de datos; reglas de validación; pipeline de datos; arquitectura ELTL; modelización dimensional; reporting de control de gestión

Contents

1 Introduction

In today's business environments, decision-making increasingly depends on reporting and business intelligence (BI) systems that consolidate data from multiple sources. However, the reliability and usefulness of these systems depend critically on the quality of the data that feed them: incomplete, inaccurate or non-traceable data can lead to distorted indicators, incorrect decisions and a loss of trust in the system itself. Consequences of low data quality are not limited to the operational domain: Redman (1998) shows that the impacts extend to all three levels of the organization. At an operational level, resources are consumed detecting and correcting errors; in service organizations, this cost can absorb between 40% and 60% of total expenses, according to Redman (1998)'s working estimates. At a tactical level, decisions become compromised because they cannot be better than the data they are based on, and the possibility that an important decision is made using entirely correct data is extremely low. At a strategic level, Redman (1998) argues that the absence of relevant, complete, accurate and timely data can severely hinder the formulation of sound strategy. Once a strategy is being implemented, late or inaccurate reporting can also make its execution and timely adjustment substantially more difficult.

This risk intensifies in organizations where data analysis is performed in a fragmented way and without centralized tools. Davenport (2006) warns that, in traditional companies, analytical functions operate across separate departments, each with their own tools and data sources, which generates multiple versions of the same key metrics. Davenport (2006), citing previous research, notes that between 20% and 40% of spreadsheets contain errors. This increases the opportunities for errors to be introduced; in a reporting-oriented environment such as the one considered here, such errors may propagate to final reports without being detected beforehand.

This work starts from a specific situation observed during the professional internship period: a management control department[1] that operated without a unified data model or any systematic validation mechanism, where reports were built from individual spreadsheets and errors were detected only after they had already affected a visible result. This configuration reproduces, at departmental level, the fragmentation that Davenport (2006) describes at organizational level, and highlights the key issue that motivates this work: the point in the data life cycle at which an error becomes visible is the end of the process, when it has already propagated to a reporting output and may therefore affect subsequent decisions.

[1] The concept of management control as used in this study is defined in Section 2.4.3.

In a situation of this kind, an immediate response would be to adopt one of the available data quality tools. However, the literature reviewed in Chapter 2 identifies relevant limitations to this approach. Ehrlinger and Wöß (2022) find that the investigated tools do not uniformly implement metrics across the principal data quality dimensions considered in their survey and that, in general-purpose data quality tools, continuous monitoring is typically offered as a premium feature in professional versions. Although dedicated open-source monitoring tools do exist, these alternatives tend to provide a narrower set of predefined profiling and measurement functionalities. From a complementary perspective, Fadlallah et al. (2023), in their review of context-aware Big Data quality assessment solutions, show that the approaches examined capture only partial representations of the context in which data are used. These findings motivate the approach adopted in this work: rather than assuming that a general-purpose tool provides a complete out-of-the-box solution and adapting it afterwards, the validation system is designed around the department's context and business rules from the outset.

The goal of this work is therefore to design and academically justify an automated data pipeline with an integrated data quality validation system, grounded in the main frameworks identified in the literature and capable of detecting and reporting records that violate the implemented validation rules before they reach management monitoring and control reports. The study also aims to position this design in relation to the alternative approaches available in the literature and to define its scope and limitations.

The methodological approach adopted to achieve this objective should be made explicit. This work does not constitute an empirical study based on real data, but rather the design of an artefact (a data pipeline architecture with an integrated validation system) and the academic justification of its design decisions through the relevant literature and the explicit requirements of the application context. This methodological orientation is consistent with design science research in information systems. Hevner et al. (2004) characterize design science as a problem-solving paradigm in which knowledge and understanding of a problem domain are developed through the construction and application of purposeful artefacts intended to address relevant organizational problems. Peffers et al. (2007) subsequently formalize this perspective through the Design Science Research Methodology (DSRM), which comprises six activities: problem identification and motivation, definition of the objectives for a solution, design and development, demonstration, evaluation, and communication. The present study is aligned particularly with the problem-identification, objective-definition, design-and-development, and demonstration activities of this framework, but it is not presented as a complete implementation of

the DSRM, since evaluation is not developed as an independent empirical phase in an organizational setting.

The internship period provided an opportunity to observe the operation of a data-processing process lacking systematic validation mechanisms; it is this deficiency, rather than the process itself, that the proposed design addresses. The observed situation therefore provides the problem from which the work starts and defines the requirements that the design must satisfy, but it is not itself the object of study: the system is designed in generic terms, while context-specific implementation choices are stated explicitly and distinguished from methodological principles directly supported by the literature. The resulting behaviour is illustrated using a simulated dataset with deliberately introduced errors, for demonstrative purposes: these illustrations show what the system does and why, but do not measure its performance on real data or allow its detection rate, false-positive rate, or computational cost to be estimated. The scope of the work is therefore the design and its theoretical foundation; empirical evaluation falls outside this scope and is explicitly addressed among the limitations discussed in Chapter 5.

This approach clarifies the contribution that the work seeks to make. It does not lie in the specific catalogue of rules that results, which is context-specific and not directly reusable in another setting, but rather in the procedure through which it was derived: the sequence of decisions that leads from a set of business rules known a priori to an ordered catalogue of validations integrated into a specific data architecture. These decisions are grounded in the relevant literature where established methodological principles are available and explicitly adapted to the requirements of the application context where the literature does not prescribe a specific implementation, with the resulting trade-offs stated as part of the design.

To achieve this objective, the remainder of the work is organized into four chapters. Chapter 2 establishes the theoretical framework: it reviews the concept of data quality and its dimensions, the methodologies for data quality assessment and improvement, and how these frameworks are translated into data pipelines and reporting environments. Chapter 3 designs the pipeline architecture and data model—an architecture with a loading layer preceding transformation and a dimensional star schema—and academically justifies each of the adopted design decisions. Chapter 4 reviews the approaches available in the literature for error detection, presents the proposed catalogue of validation rules and the associated alert mechanism, and illustrates its operation using simulated data. Finally, Chapter 5 discusses the limitations of the design, including the organizational conditions that its operation presupposes, and outlines directions for future development.

2 Theoretical Framework

2.1 The Data Quality Concept

The field of data quality has evolved significantly since the early academic works of the 1990s, becoming a discipline in its own right within information systems. Despite this evolution, its core definition has remained quite stable. Wang and Strong (1996) define data quality in terms of fitness for use by data consumers, while Ehrlinger and Wöß (2022) note that *fitness for use* remains the most frequently used characterization of data quality. This conception, rooted in the general product quality literature, places the end user at the center of quality assessment, extending earlier approaches that focused primarily on characteristics of data production and storage rather than explicitly capturing the requirements of data consumers.

This consumer-oriented perspective has an important implication: data quality is not an absolute property of the data, but is instead contingent upon the context in which the data are used. The same data can be of high quality for one task, and inadequate for another one. As Wang and Strong (1996) argue, a dimension such as completeness cannot be evaluated in the abstract, but only in relation to the task that the consumer needs to perform. This broadened the prevailing conception of data quality beyond the strong emphasis traditionally placed on accuracy, by explicitly incorporating the context of use and the perspective of the data consumer.

The limitations of an approach focused primarily on accuracy are illustrated by empirical evidence from the period. Wang and Strong (1996) document cases such as that of a major New York bank whose credit risk database had a completeness rate of only 60%, or that of a manufacturing company that could not aggregate data relating to the same customer because its system assigned multiple different identifiers to the same customer. These examples illustrate why accuracy alone does not provide a complete characterization of data quality: the first problem concerns completeness, while the second affects the ability to identify and retrieve all data associated with the same real-world entity.

In order to complete this perspective focused on the consumer, Wang et al. (1995) propose understanding information systems as data manufacturing systems. Under this analogy, raw data are equivalent to raw materials, data processing corresponds to the industrial transformation process, and output data corresponds to the final product. This metaphor is not merely illustrative: it provides a basis for transferring concepts and practices from product quality management to data quality. Wang et al. (1995) operationalize

this analogy by adapting principles from the ISO 9000 family of standards to the analysis and management of data quality.

Taken together, these two approaches establish the conceptual basis upon which the present study is built. The view of quality as fitness for use justifies the fact that the validation system designed in this study is not aimed at ensuring an abstract notion of data quality, but rather the specific level of quality required for the management control reports that constitute the final output of the pipeline. In the present study, the data manufacturing metaphor is used to interpret the pipeline as a production chain composed of distinct stages at which quality controls can be introduced systematically.

2.2 Data Quality Dimensions

To operationalize the concept of data quality and make it measurable, the literature has identified a set of dimensions that represent its different aspects. Wang and Strong (1996) define a data quality dimension as a set of quality attributes that represents a single aspect or construct of quality. Identifying these dimensions constitutes a necessary prerequisite for the systematic measurement, analysis, and improvement of data quality.

The framework proposed by Wang and Strong (1996), developed from an empirical study with real data consumers, organizes the dimensions into four hierarchical categories: intrinsic data quality, contextual data quality, representational data quality and accessibility data quality. The main contribution of this framework is showing that data quality is a multidimensional concept which extends far beyond just accuracy, and that the dimensions considered relevant depend, at least in part, on the perspective from which quality is assessed.

2.2.1 Intrinsic Data Quality

Intrinsic data quality gathers those dimensions that refer to data quality in and of themselves, independently of the context of use. It includes accuracy, objectivity, believability and reputation. Of these, accuracy has historically received disproportionate attention in both research and practical data quality improvement efforts. Batini et al. (2009) distinguish between two types of accuracy: syntactic accuracy, which measures whether a value conforms to the set of admissible values defined by its domain, and semantic accuracy, which measures whether the value corresponds to the real-world value it is intended to represent. In practice, semantic accuracy is considerably more difficult to operationalize than syntactic accuracy. Batini et al. (2009) note that the methodolo-

gies they review primarily operationalize syntactic accuracy, while specific measurement methods for semantic accuracy are generally absent. This difficulty is also reflected in contemporary tools: Ehrlinger and Wöß (2022) show that implemented accuracy metrics may require an external reference or "gold standard" dataset, which is often unavailable.

A notable feature of the framework proposed by Wang and Strong (1996) is the inclusion of believability and reputation alongside accuracy and objectivity within intrinsic data quality. This reflects that data consumers evaluate not only whether data are correct, but also whether they regard the data and their source as trustworthy and reputable.

2.2.2 Contextual Data Quality

Contextual data quality encompasses those dimensions whose assessment depends on the context of the task that the consumer needs to perform. It includes value-added, relevancy, timeliness, completeness and appropriate amount of data. The defining feature is that the same data may exhibit high contextual quality for one task and low contextual quality for another (Wang & Strong, 1996).

Completeness deserves special attention because of its practical relevance. Batini et al. (2009) define it as the degree to which a dataset contains all the values it should contain in order to adequately represent the set of real-world objects it describes. In the context of relational databases, completeness is closely linked to the meaning of null values: a value may be missing because it does not exist, because it exists but it is unknown, or because it is not known if it exists. Each case has different implications for the assessment of data quality (Batini et al., 2009).

Timeliness, on the other hand, concerns whether data are sufficiently current for the task at hand. Batini et al. (2009) show that the terminology surrounding time-related data quality dimensions is not standardized: across the literature, terms such as timeliness and currency are sometimes used for overlapping concepts, while some definitions characterize timeliness through the age of the data and their volatility. The authors therefore conclude that there is no general agreement on the abstract definition of time-related dimensions.

2.2.3 Representational Data Quality

Representational data quality refers to how data are presented to data consumers. It includes interpretability, ease of understanding, representational consistency and concise representation. Wang and Strong (1996) emphasize that this category highlights the role of information systems in the presentation of data: it is not enough for data to be

intrinsically correct and contextually relevant if they are not presented in a way that enables the consumer to interpret and understand them.

Representational consistency in Wang and Strong (1996)'s framework should not be confused with the consistency dimension commonly used in database-oriented data quality literature. Batini et al. (2009) discuss the latter in detail and define it as the absence of violations of semantic rules defined over a dataset. In the relational field, these rules take the form of integrity constraints, which may be either intra-relational (for example, a person's age must be between 0 and 120 years) or inter-relational (for example, the year of a film in the awards table must match the year recorded in the films table).

2.2.4 Accessibility Data Quality

Accessibility data quality gathers the dimensions related to data availability and access security. It includes accessibility and access security. Wang and Strong (1996) emphasize that, together with representational data quality, this category highlights the role of information systems in determining whether data can effectively be used by data consumers.

It should be noted that the review of dimensions presented in this section establishes the conceptual vocabulary of the work, but does not by itself determine the content of the validation catalogue. As discussed in Section 2.4.4, Ehrlinger and Wöß (2022) note that there is no standardized consensus on data quality dimensions and metrics, while their empirical analysis of existing tools reveals an uneven implementation of the metrics proposed in the academic literature. These findings motivate the approach adopted in Chapter 4, where the validation catalogue is grounded in concrete, operationally measurable properties rather than being derived mechanically from the four-category framework. This also explains the inclusion of uniqueness, which is not among the dimensions identified by Wang and Strong (1996) but is broadly implemented in the tools examined by Ehrlinger and Wöß (2022).

2.3 Methodologies for Data Quality Assessment and Improvement

The identification of data quality dimensions is a necessary but not sufficient condition for managing data quality efficiently. It is also necessary to have methodologies that systematically guide the assessment and improvement process. Batini et al. (2009) define a data quality methodology as a set of guidelines and techniques that, based on information

on the application context, establishes a rational process for assessing and improving data quality.

2.3.1 Phases of a Data Quality Methodology

In the most general case, Batini et al. (2009) organize the sequence of activities of a data quality methodology into three phases. The first one is the state reconstruction, which consists of gathering contextual information about organizational processes, data collections and existing quality problems. The second one is the assessment, which measures the level of data quality across the relevant dimensions and compares it against reference values in order to establish a diagnosis. The third one is improvement, which involves selecting and applying strategies and techniques to achieve new quality objectives. The state reconstruction phase may be omitted when the necessary contextual information is already available from previous analyses.

Within the assessment phase, Batini et al. (2009) distinguish between objective measurements, based on quantitative metrics, and subjective measurements, based on qualitative assessments provided by data administrators and data consumers. The methodologies reviewed by Batini et al. (2009) use these two forms of assessment in different combinations depending on their scope and application context.

2.3.2 Improvement Strategies: Data-Driven and Process-Driven Approaches

One of the most relevant contributions of Batini et al. (2009) to the present study is the distinction between two general strategies of improvement. Data-driven strategies improve quality by directly modifying data values: for example, by updating outdated values, standardizing non-standard formats, or detecting and correcting errors. Process-driven strategies, by contrast, intervene in the processes that create or modify data rather than directly modifying individual data values. They include process control, which introduces verification checkpoints whenever new data are created, datasets are updated, or new datasets are accessed, thereby adopting a reactive strategy triggered by data modification events in order to prevent data degradation and error propagation, and process redesign, which modifies the process itself in order to remove the underlying causes of poor data quality.

The comparison between these two strategies shows an important asymmetry. Batini et al. (2009), summarizing earlier comparisons in the literature, report that data-driven strategies tend to be cost-efficient in the short term but expensive in the long term,

whereas process-driven techniques generally perform better in the long term because they address the root causes of quality problems. Process redesign, however, can be particularly expensive in the short term.

2.3.3 Types of Data Quality Methodologies

Based on the comparative analysis of thirteen existing data quality methodologies, Batini et al. (2009) propose a classification into four categories. Complete methodologies cover both the assessment and improvement phase, and address both technical and economic aspects; they are particularly useful as comprehensive frameworks for large-scale data quality programmes in organizations that manage critical data. Audit methodologies focus on the assessment phase, providing only limited support for improvement; they represent the largest category and tend to provide more precise assessments, but are less action-oriented. Operational methodologies focus on the technical dimensions of data quality assessment and improvement while excluding economic considerations; because they are typically tailored to specific application contexts, they tend to achieve greater efficiency at the expense of generalizability. Finally, economic methodologies focus on the economic dimension of data quality, considering both the costs of assessment and improvement initiatives and the costs associated with poor data quality, and can complement methodologies from the other categories.

2.3.4 Application to the Present Study

The methodological framework of Batini et al. (2009) situates the system designed in this study within a well-defined academic context. The proposed data pipeline adopts a process-driven strategy, specifically the process control approach: rather than detecting and correcting errors once they have already reached the final reports, the system introduces automated validation checkpoints at the transformation stage of the pipeline, triggered as the data are processed and transformed. In the sense of Batini et al. (2009), these controls are intended to prevent data quality problems from propagating through the information system. This does not amount to process redesign, nor does the proposed system itself eliminate the root causes of poor data quality. Rather, it controls error propagation within the pipeline, while correction of the underlying problem remains the responsibility of the corresponding source process. It should also be clarified that process control constitutes, in the authors' own terms, a reactive strategy, insofar as it is triggered by data modification events; the distinction from subsequent correction at the reporting

level therefore does not lie in the reactive nature of the mechanism, but in the point in the data lifecycle at which it is activated. The scope of this study does not include a formal economic assessment of either the costs associated with poor data quality or the costs of quality assessment and improvement interventions, aspects addressed by the economic methodologies identified by Batini et al. (2009) and left as a potential avenue for future work.

2.4 Data Quality in Pipelines and Reporting Environments

2.4.1 From Generic Information Systems to Modern Data Pipelines

The classic works reviewed in the previous sections (Wang and Strong (1996), Wang et al. (1995) and Batini et al. (2009)) address data quality in the context of information systems in a generic sense, without specifying a particular system architecture. The three-step methodology identified by Batini et al. (2009) (state reconstruction, assessment, improvement) is still valid as a conceptual framework. However, it does not explain how these steps can be mapped onto a modern data pipeline, with its distinct stages of integration, transformation, modelling and report generation. By comparing several data quality methodologies, Ehrlinger and Wöß (2022) identify four recurring core activities: state reconstruction, data quality measurement or assessment, data improvement, and continuous monitoring. Not all methodologies include all four activities; in particular, the survey by Batini et al. (2009) does not explicitly include continuous monitoring. This distinction is relevant to the present study because it makes quality assessment an ongoing activity rather than limiting it to a single assessment-and-improvement cycle.

Taleb et al. (2021) translate this perspective into a contemporary Big Data lifecycle model. Building on existing Big Data system architectures, the authors organize the lifecycle into four macro-phases: data generation, data acquisition, data storage, and data analysis. Data acquisition is further divided into data collection, data transmission, and data pre-processing, while data analysis encompasses data processing, analytics, and visualization. On this basis, the authors identify seven points in the lifecycle at which data quality is relevant: data generation, collection, transmission, pre-processing, storage, processing and analytics, and visualization. Particular emphasis is placed on the pre-processing stage for data-driven quality improvement, since activities such as cleansing, filtering, and normalization should be performed before data processing and, whenever possible, as early as possible in the lifecycle. This emphasis on early intervention does not exclude subsequent quality controls, as the proposed framework extends data quality

assessment and monitoring throughout the lifecycle.

This model establishes a useful correspondence with the three functional areas that Ehrlinger and Wöß (2022) use to organize their review of data quality tools: data profiling, data quality measurement, and data quality monitoring. Ehrlinger and Wöß (2022) describe data profiling as the analysis of a dataset to collect metadata such as missing or distinct values, attribute data types, and recurring patterns before data quality measurement or monitoring is performed. In methodological terms, this activity corresponds to the data-oriented component of the state reconstruction phase described by Batini et al. (2009), while data quality measurement is closely aligned with the assessment phase. Continuous monitoring, however, has no direct equivalent in Batini et al. (2009)'s three-phase sequence and is treated by Ehrlinger and Wöß (2022) as an additional ongoing activity intended to maintain knowledge of data quality over time.

This three-part framework, adapted to the context of the present study, provides the conceptual organization of the validation system described in Chapters 3 and 4. The measurement function is operationalized through the execution of the validation rules defined in the catalogue, while monitoring is represented by the periodic execution of these controls together with the storage of their results over time. Data profiling, by contrast, is not incorporated as a distinct stage of the proposed design, since the relevant attributes and business rules are known a priori in the departmental context and the validation rules are defined directly on that basis. This is a context-specific design choice rather than a claim that data profiling is unnecessary in data quality systems more generally.

2.4.2 The Role of Context in Data Quality Assessment

The three-stage framework (profiling, measurement and monitoring) established in the previous section describes how data quality assessment is organized throughout the data pipeline, but it does not explicitly explain what makes the same data valid in one context and unsuitable in another. This is the role of the concept of context, which was already recognized in the classical literature reviewed in Section 2.2, particularly by Wang and Strong (1996) through their category of contextual data quality dimensions. Their framework relates data quality to the requirements of the task at hand, but it does not provide the broader operational decomposition of context developed in later context-aware data quality research.

In their scoping review of data quality assessment in Big Data environments, Fadlallah et al. (2023) provide a more precise definition of context. According to the authors, the

context of a data item may include its metadata (such as data type and format), the organizational policies that apply to it, the domain-specific business rules governing its use, and the system resources available to process it. The authors also note, more broadly, that context is closely tied to the specific stage of the data lifecycle in which an item resides, since quality requirements can vary substantially between ingestion, processing, and analysis. This latter idea is relevant for the present study. For example, the credibility of a data item may be of secondary importance during data ingestion but become critical when generating a report to support decision-making.

Based on their review of twenty-seven data quality solutions published between 2005 and 2021, Fadlallah et al. (2023) reach a conclusion that is especially relevant in the present study: none simultaneously achieves context awareness and satisfies the requirements associated with handling Big Data. The authors point out that each solution captures only a partial view of context, without reaching a comprehensive notion that integrates all of its constituent elements.

2.4.3 The Specific Challenges of Reporting and Management Control

Before examining the data quality implications of reporting, it is useful to clarify how management control is understood in this study. Management control refers broadly to the mechanisms through which managers seek to align organizational actions and decisions with organizational objectives; management control systems therefore encompass performance measurement, evaluation, and other mechanisms used to support that alignment (Merchant & Van der Stede, 2017). In the present study, the reporting layer is understood as an informational component supporting management control rather than as a management control system in itself.

When the final goal of a pipeline is the generation of reports for management control, data quality becomes particularly critical because the consequences of an error are not confined to a single analytical task but are propagated directly into the decision-making process. Although the reviewed literature does not examine management-control reporting as a specific application domain, it provides relevant evidence about the limitations of existing data quality solutions in this type of setting. Ehrlinger and Wöß (2022) show that automated data quality monitoring—the repeated measurement and comparison of data quality over time—is generally treated as a premium functionality in general-purpose data quality tools and is therefore typically available only in professional versions. Dedicated open-source monitoring tools do exist, notably Apache Griffin and MobyDQ, but the authors note that these solutions lack predefined data quality functions and data profiling

capabilities. Consequently, continuous data quality monitoring cannot be assumed to be available out of the box when relying on a general-purpose data quality solution, which reinforces the need to make the required validation and monitoring mechanisms explicit in the design of the pipeline.

This finding can be interpreted alongside the fragmentation of context discussed in Section 2.4.2. Just as no single tool reliably covers the full range of data quality dimensions, none of the solutions reviewed by Fadlallah et al. (2023) provides a comprehensive representation of context. Taken together, these findings identify two complementary limitations, which together make it difficult for a generic data quality tool or framework to be effectively adapted to a specific organizational use case without substantial additional specification and customization.

A further challenge is conceptual in nature. Sebastian-Coleman (2013), as cited in Ehrlinger and Wöß (2022), observes that even data quality practitioners are often unable to specify precisely how dimensions such as completeness or accuracy should be measured within their particular organizational context. This reinforces the need to define explicitly what constitutes an acceptable value and how each relevant business rule is to be operationalized before validation can be automated.

Taleb et al. (2021) provide a quality-control mechanism directly relevant to this scenario. Their framework's quality control component introduces explicit verification checkpoints at which data quality scores are evaluated against predefined requirements before the data are allowed to proceed to the processing and analytics stages.

2.4.4 From Theory to Practice: Recent Data Quality Tools and Frameworks

The systematic review conducted by Ehrlinger and Wöß (2022), which identified 667 data quality tools and evaluated 13 of them in detail (8 commercial and 5 open source) against a catalogue of 43 requirements, provides detailed empirical evidence on the current state of practical implementation. The results reveal a clear pattern: basic column-level profiling (such as row counts, null values, and distinct values) is well supported by virtually all tools, whereas dependency discovery is comprehensively supported by only two tools, while advanced multi-column profiling is also unevenly supported: duplicate detection is common, but correlation analysis, association-rule mining and clustering receive little or no full support. With regard to the measurement of data quality dimensions, no tool provides a comprehensive set of metrics covering the four classical dimensions of accuracy, completeness, consistency, and timeliness. The implementations that do exist

exhibit at least one of the following limitations: they operate only at the attribute level without aggregation capabilities, require a reference ("gold standard") dataset that is often unavailable, or contain implementation flaws. Accuracy is implemented by only a single tool and only when an external reference dataset is available, whereas completeness and uniqueness are the two most broadly implemented aspects, precisely because both share the practical advantage that they can be calculated without necessarily requiring a "gold standard".

From this observation, Ehrlinger and Wöß (2022) draw a conclusion that is methodologically relevant to the present study. Rather than starting from the abstract data quality dimensions proposed in the academic literature, which neither enjoy universal consensus nor are always measurable in practice, they argue that it is more effective to focus data quality assessment on directly measurable aspects, such as missing values and duplicate records.

Taleb et al. (2021) provide an example of how this translation can be carried out systematically through the Data Quality Profile (DQP), which is created when the Big Data Quality Project is defined and progressively enriched through a sequence of states from DQP 0 to DQP 8. The optimized rules are then applied to the complete dataset through the framework's Big Data Pre-Processing component (component 9), after which continuous quality monitoring produces the quality reports recorded as DQP 10. The DQP and its successive updates are stored in a repository throughout the process, enabling reuse and adaptation for datasets belonging to the same domain. A particularly valuable aspect of their approach is the explicit distinction between data quality dimensions and metrics, which are used for measurement, and data quality rules, which are used to drive corrective actions. Each rule is linked to one or more specific preprocessing functions. For example, for the completeness dimension, these functions may involve replacing missing values with the mean, mode, or median, or alternatively removing entire rows or columns. This design (comprising measurement, comparison against a tolerance threshold, rule generation, validation of the generated rules on a sample before applying them to the full dataset, and continuous monitoring of the results) provides a more concrete operationalization of the classical data quality dimensions than that offered by most of the tools analyzed by Ehrlinger and Wöß (2022).

2.4.5 Towards a Validation System Integrated Into the Data Pipeline

The preceding sections establish the methodological basis for the validation system developed in Chapters 3 and 4. Taken together, the reviewed literature suggests that

the design must make three elements explicit: the context-specific requirements that determine what constitutes acceptable data, the validation controls through which those requirements are operationalized, and a monitoring mechanism that preserves the results of those controls over time. The proposed system therefore does not attempt to reproduce any single existing tool or framework. Instead, it combines these principles within the pipeline architecture developed in Chapter 3: validation is integrated into the transformation stage before data reach the reporting layer, while monitoring is implemented through the periodic execution of the controls and the persistent recording of their results.

3 System Design: Architecture and Data Model

3.1 Pipeline Architecture: A Load-First Approach

This section presents a generic model of the data pipeline architecture proposed for the management control department. The objective is not to reproduce the specific pipeline implemented during the professional placement, but rather to design and provide an academic justification for an architecture that addresses the same underlying problem: a department that previously lacked both a unified data model and a systematic mechanism for data validation.

This work takes as its conceptual starting points the lifecycle perspective developed by Taleb et al. (2021) and the profiling, measurement, and monitoring framework used by Ehrlinger and Wöß (2022), and implements these ideas within the integration pattern described below. The proposed pipeline is organized into four integration stages: data extraction from the source systems, an initial loading of the data in raw form, a transformation stage that applies data cleansing, business rules, and aggregation logic, and a final loading stage that delivers the resulting data to the reporting layer.

The ordering of these stages is a deliberate design decision rather than an arbitrary one. Rucco et al. (2025) formalize two hybrid integration patterns that explicitly reflect this choice: ETLT (Extract, Transform, Load, Transform), in which an initial quality-oriented transformation is performed before loading and a second transformation subsequently applies business-oriented logic, and ELTL (Extract, Load, Transform, Load), in which minimally processed data are first loaded into a raw layer, then transformed and loaded a second time into a structure intended for downstream consumption. The pipeline designed in this study follows the ELTL pattern: data are extracted with only minimal preprocessing, loaded into a raw layer intended to preserve source fidelity, and

transformed afterwards before being loaded again into a structure suitable for reporting and downstream consumption.

It should be noted that this formalization is available as a preprint and, in the version cited here, no peer-reviewed publication is identified. It is used in this study because it provides a systematic vocabulary and an explicit characterization of the hybrid integration pattern adopted here. The separation between an upstream data-processing environment and a downstream structure designed for business consumption is also compatible with the distinction drawn by Kimball and Ross (2013) between the ETL back room and the presentation area, although Kimball and Ross (2013) do not use the ELTL terminology.

This design choice is justified by three main considerations. First, data lineage and traceability. Rucco et al. (2025) identify raw-data preservation as a key advantage of the ELTL pattern, since an immutable raw zone can provide a complete audit trail and support the reconstruction of previous data states. The architecture proposed in this study adopts this principle explicitly: each raw load is retained independently rather than overwritten by subsequent executions, so that the historical sequence of source states remains available over time. In the context of management control, this is particularly valuable because, whenever a reported figure is questioned, the corresponding historical raw load can be retrieved to verify the source-oriented data from which that figure was derived. Second, reusability. Because the preserved raw loads remain independent of any specific downstream transformation, different teams or reporting processes can apply different transformation logic to the same source-oriented data without repeating the extraction process, thereby reducing duplicated ingestion work and supporting consistent reuse across outputs (Rucco et al., 2025). Third, schema-evolution flexibility. Since historical raw loads are preserved independently of downstream structures, the reporting schema and transformation logic can be modified without altering the original data previously received from the source systems (Rucco et al., 2025). In a management control environment, where business rules and reporting requirements may evolve over time, this separation makes it possible to revise downstream transformations while retaining access to the original historical loads on which earlier outputs were based.

Within this ELTL architecture, the two loading stages serve distinct purposes. The first loading stage stores a minimally processed representation of the source data, with the objective of preserving their fidelity and providing a traceable record of what has been received from the source systems. The second loading stage, by contrast, delivers a cleansed and structured version of the data that is suitable for downstream consumption. The design of this second layer, including the data organization adopted to support

reporting, is described in detail in Section 3.2.

Finally, the architecture presented in Figure 1 represents the ELTL pattern adopted as the conceptual basis of the proposed pipeline. At this stage, the model specifies the general sequence of extraction, raw loading, transformation, and downstream loading, but it does not yet determine where data validation should be introduced or how detected violations should be handled. These design decisions are developed in the following sections and incorporated into the final system architecture presented in Chapter 4.

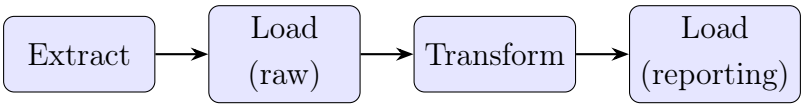

Figure 1: *ELTL pipeline architecture adopted as the conceptual basis of the proposed system*

Source: Own elaboration based on Rucco et al. (2025).

3.2 Data Model Justification: Star Schema Over Normalized Design

Once the pipeline architecture has been justified in Section 3.1, attention turns to the organization of the data within the second loading stage, namely the layer that ultimately feeds the reporting applications. This is far more than a minor implementation detail: the way in which data are structured in this layer strongly influences how easily business users can understand and query the information, as well as how readily the system can accommodate evolving reporting requirements. This section justifies the adoption of a dimensional data model—specifically, a star schema—as the organizing principle for this layer.

Kimball and Ross (2013) present this design decision as a choice between two fundamentally different modelling philosophies. Normalized models, typically implemented in Third Normal Form (3NF), divide data into multiple discrete, non-redundant entities. Although these structures are sometimes referred to in the industry as Entity–Relationship (ER) models, Kimball and Ross (2013) avoid this terminology because both normalized and dimensional models can be represented through entity–relationship diagrams. This approach is well suited to operational systems, where each transaction affects the database in a single, well-defined location. Dimensional models, by contrast, organize data around two types of tables: fact tables, which typically store the numerical measures generated by business process events, and dimension tables, which store the descriptive context

(the "who, what, where, when, why, and how") associated with those events (Kimball & Ross, 2013). Both are relational modelling approaches, but they differ substantially in their degree of normalization and in how data are organized for operational processing or analytical consumption.

For a reporting-oriented environment such as the one considered in this study, a normalized design presents two significant drawbacks. First, normalized schemas tend to fragment data across a large number of tables connected through intricate join paths, a structure that Kimball and Ross (2013) compare to the Los Angeles freeway system. Business users, who are typically not database specialists, are generally unable to navigate this level of complexity without assistance, undermining one of the fundamental objectives of any reporting system: enabling users to access and interpret information without relying entirely on technical intermediaries. Second, this same structural complexity often has a negative impact on query performance. Unpredictable ad hoc queries involving multiple joins are more difficult for database management systems to optimize, which can result in slow response times precisely for the broad, exploratory analyses that reporting users are most likely to perform (Kimball & Ross, 2013).

A star schema addresses both of these issues simultaneously. By organizing data around a central fact table surrounded by denormalized dimension tables, the resulting structure is, by design, simple, symmetrical, and immediately recognizable to business users, as it typically reflects the way they already think about their business processes (Kimball & Ross, 2013). This simplicity also improves query performance. Because the schema contains relatively few tables and predictable join paths, database management systems can execute queries more efficiently by first filtering the indexed dimension tables and then resolving the fact table in a single step (Kimball & Ross, 2013). In other words, the dimensional model addresses the two simultaneous requirements that Kimball and Ross (2013) identify for presenting analytical data: understandability for business users and fast query performance.

An additional advantage of the dimensional approach lies in its ability to accommodate change over time. Kimball and Ross (2013) describe dimensional models as gracefully extensible: new facts can be added when they are consistent with the existing fact-table grain, new dimensions can be incorporated provided that they do not alter that grain, and additional descriptive attributes can be added to existing dimensions without requiring changes to existing BI queries or applications. Applying this reasoning to the context of this study, this property is particularly relevant in the management control environment addressed here, where reporting requirements are rarely static: the set of

metrics considered relevant, as well as the level of detail at which they need to be analyzed, is likely to evolve as the department's information needs mature. A model that required substantial redesign every time a new reporting requirement emerged would be difficult to maintain in practice.

It should be noted that adopting a dimensional model for the reporting layer does not preclude the use of normalized structures elsewhere in the pipeline. As discussed in Section 3.1, the ELTL pattern separates the raw data preserved in the first loading stage from the structured data delivered through the second. Kimball and Ross (2013) explicitly allow optional normalization to support ETL processing and note that normalized structures may be appropriate within the ETL staging area, while discouraging normalized or snowflaked designs in the presentation area intended for business users. The design adopted in this study follows this separation: normalized structures may be used upstream where they support data processing, whereas the layer exposed to reporting applications is deliberately organized according to a dimensional model.

Applied to the context of this study, this design choice leads to a reporting layer conceptually organized around the department's relevant business processes, with fact tables storing the quantitative measures associated with them, such as the number of active projects, the volume of recorded defects and tests, or confirmed budget amounts. These fact tables are linked to dimension tables that provide the descriptive context required to filter, group, and label the measurements, such as a project dimension and a time dimension, with descriptive attributes including project type or reporting period. The structural principle underlying this organization—a central fact table surrounded by dimension tables that provide its descriptive context—is illustrated in Section 3.4 through a simplified simulated example.

3.3 Update Frequency and Additional Design Decisions

Once the pipeline architecture (Section 3.1) and the dimensional data model (Section 3.2) have been justified, three additional design decisions remain to be addressed: how frequently the pipeline should be executed, the level of granularity at which data should be represented within it, and the key strategy used to identify dimension members. These decisions have implications both for the practical operation of the data quality validation system described in Chapter 4 and for the flexibility and maintainability of the resulting reporting layer.

3.3.1 Update Frequency: A Daily Batch Cycle

The pipeline proposed in this study operates on a daily batch cycle: source data are extracted, loaded, transformed, and loaded again once per day, rather than being processed continuously or in real time. This choice is not merely a default inherited from traditional data warehousing practices; rather, it follows directly from the nature of the reporting process that the pipeline supports and from the operational costs associated with more demanding alternatives.

Kimball and Ross (2013) treat real-time fact tables as a distinct design pattern within dimensional modelling and note that supporting updates more frequently than a traditional overnight batch process typically requires dedicated mechanisms. These include, for example, a hot partition held in memory, on which aggregations and indexes are deliberately avoided, or deferred update strategies that allow ongoing queries to complete before newly arrived data are incorporated. These mechanisms illustrate the trade-off involved in low-latency delivery: while the stable portion of a fact table may rely extensively on indexes and aggregates to support query performance, the real-time partition is typically kept lightweight in order to accommodate frequent updates. Applying this reasoning to the present study, a management control department whose reports are consulted periodically, rather than used to support second-by-second operational decisions, has relatively little to gain from the additional architectural complexity required for real-time updates, while still bearing their full implementation and maintenance costs.

The tool survey conducted by Ehrlinger and Wöß (2022) is relevant not for determining the specific frequency adopted here, but for showing how periodic execution forms part of automated data quality monitoring. Their requirements catalogue explicitly includes the scheduling of data quality metrics or profiling tasks at user-defined intervals, together with the storage, retrieval, comparison, and visualization of the resulting measurements over time. In the present study, however, the choice of a daily frequency is a context-specific design decision driven primarily by the reporting cadence and the absence of a requirement for low-latency data, rather than a direct consequence of the limitations of existing tools.

Finally, the adoption of a periodic execution cycle is compatible with the monitoring concept used by Taleb et al. (2021). In their framework, continuous quality monitoring constitutes an ongoing process that closes the quality-enforcement loop and can trigger the adaptation or update of the Data Quality Profile when a quality failure is detected. The authors do not prescribe a specific execution interval or equate continuous monitoring with real-time or streaming processing. The framework is therefore compatible with repeated

periodic monitoring, while the specific daily frequency adopted in this study is determined by the reporting requirements of the application context rather than by the framework itself.

3.3.2 Grain: Atomic-Level Fact Tables

A second design decision concerns the grain of the fact tables in the reporting layer, that is, the precise level of detail represented by a single row in a fact table. Kimball and Ross (2013) describe defining the grain as the pivotal step in a dimensional design, arguing that it should be established before selecting either dimensions or facts, since every candidate dimension and measure must be consistent with it. They distinguish atomic grain, the lowest level of detail at which a business process is captured, from aggregated or summarized grains, and argue strongly in favour of the former. Atomic data can support unpredictable analytical queries, whereas pre-aggregated data may prevent users from answering common business questions that were not anticipated when the aggregates were defined.

This study adopts atomic grain as the guiding design principle for the reporting layer. Kimball and Ross (2013) warn that dimensional models populated solely with summarized data are prone to what they describe as analytic brick walls: users are unable to drill down into details that were never captured, and developers have greater difficulty accommodating new dimensions, attributes, or facts once the data have been prematurely aggregated. In the present study, the atomic grain must therefore be declared separately for each business process represented in the reporting layer. For example, a fact table describing defects may represent one individual defect per row, while a fact table describing tests may represent one individual test execution per row; project-related measurements would similarly require their own explicitly defined grain. Preserving the lowest relevant level of detail within each fact table maintains the possibility of analyzing the data across dimensions or at levels that may not have been anticipated when the model was initially designed. This is consistent with the requirement discussed in Section 3.2 that the reporting layer should accommodate evolving reporting needs without unnecessary redesign.

3.3.3 Surrogate Keys

A final design decision concerns the choice of primary keys for the dimension tables. Kimball and Ross (2013) recommend that dimension tables should not use the natural

key from the operational system as their primary key. Instead, they advocate the use of a dedicated surrogate key: a simple, sequentially assigned integer with no inherent business meaning. Two of the reasons identified by the authors are particularly relevant to the present study. First, natural keys may change, be reused over time, or differ across source systems. A surrogate key allows the data warehouse to take ownership of the identity of each dimension member, rather than relying on whatever key conventions exist upstream. Second, a natural key cannot uniquely identify a dimension row when multiple versions of the same real-world entity must coexist over time, because by definition the natural key remains unchanged across those different versions.

This work adopts surrogate keys for all dimension tables in the reporting layer, in accordance with this recommendation. The simulated example in Section 3.4 implements them as sequential integers with no business meaning, while retaining the natural key as a descriptive attribute of the dimension. The example does not, however, incorporate historical tracking of dimensional attributes, as this falls outside the scope of the illustration. Nevertheless, adopting surrogate keys from the outset preserves the appropriate key structure for a future implementation of historical tracking, although additional attributes and processing logic would still be required to implement such functionality.

3.4 Illustration With Simulated Data

To illustrate how the dimensional modelling decisions justified in the preceding sections translate into practice, a simulated dataset has been constructed that reproduces the basic structure of the proposed model. It should be emphasized that this dataset serves a purely illustrative purpose: it is not intended to reproduce the department's business process, but rather to demonstrate the structural properties of the dimensional model and some of the structural validations that can be performed on it. For this reason, it adopts a generic sales domain instead of the business processes described in Section 3.2, since the structural properties illustrated here are independent of the specific business process being modelled.

The dataset reproduces a minimal star schema consisting of a fact table (`fact_results`), with 25 sales records, and two dimension tables: `dim_product`, with 6 products, and `dim_time`, with 10 dates. Each row in the fact table represents an individual sale rather than an aggregate total by product or period, thereby applying the atomic grain criterion justified in Section 3.3.2. The numerical measures—quantity and total amount—are stored in the fact table, whereas the descriptive attributes used to filter and group the

information—such as product name and category, or day, month, and year—are stored in the dimension tables, in accordance with the separation between facts and descriptive context justified in Section 3.2.

This organization determines which quality checks can be performed on the dataset. The most characteristic one follows directly from the schema: because the fact table references the dimensions through the attributes `id_product` and `id_time`, it is possible to verify that every value present in these attributes has a corresponding entry in the dimension table, a fact-to-dimension referential integrity check that arises specifically from the separation between fact and dimension tables. The simulated dataset can also be used to illustrate record-level checks that do not depend specifically on the star schema, such as verifying that a critical attribute contains no missing values or that a numerical amount remains within its admissible range.

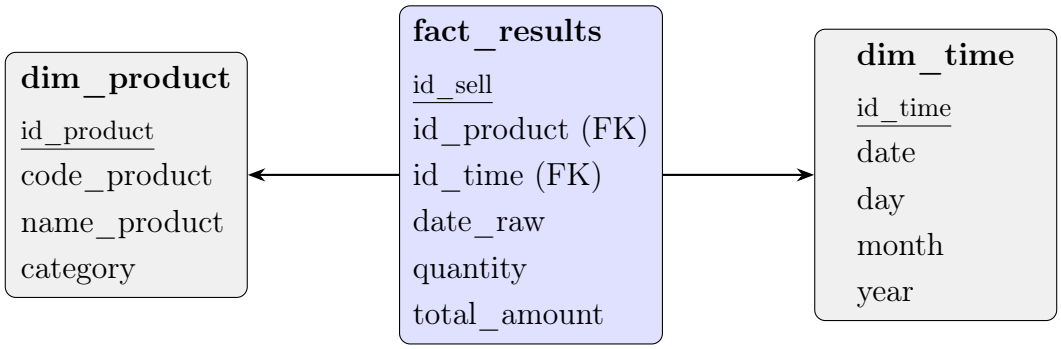

Figure 2: *Star schema of the simulated dataset: one fact table surrounded by two dimension tables. Arrows indicate the foreign key relationships on which the referential integrity rule operates.*

Source: Own elaboration.

The auxiliary attribute `date_raw` is retained in the simulated fact dataset solely to support the validation examples developed in Chapter 4; it represents the value received from the source before its resolution against the time dimension and should not be interpreted as a required attribute of the final reporting schema.

The systematic design of a complete catalogue of validation rules, the methodological discussion of the alternative approaches for defining them, and their application to this same dataset, extended with deliberately introduced errors, are presented in Chapter 4. The code used to generate the simulated dataset is provided in Appendix A.

4 Data Quality Validation System: Rule Design and Alerting Mechanism

Once the general pipeline architecture and the dimensional modelling approach have been justified in Chapter 3, this chapter turns to the systematic design of the data quality validation system itself. Section 4.1 examines the framework proposed by Taleb et al. (2021), with particular attention to the distinction between quality evaluation, quality control, corrective rule discovery, rule validation, optimization, and monitoring. Section 4.2 reviews alternative approaches to the detection of data quality problems and anomalies found in the literature. Building on these two perspectives, Section 4.3 presents the catalogue of validation rules adopted in this study and justifies its design in relation to the alternative approaches discussed previously. Section 4.4 describes the alerting and record-handling mechanism used when validation rules are violated, consistently with the design decisions established in Section 4.3. Finally, Section 4.5 illustrates the resulting validation system through a simulated example that extends the dataset introduced in Section 3.4.

4.1 Quality Rule Design: The Data Quality Profile Framework

This section expands upon the discussion introduced in Section 2.4.4 by examining in greater depth the theoretical framework proposed by Taleb et al. (2021) for structuring a catalogue of data quality rules.

Taleb et al. (2021) introduce the Data Quality Profile (DQP) as the central artefact of their framework: a structure that jointly captures data quality requirements, the target data attributes and quality dimensions, the resulting quality scores, and the quality rules generated to correct any mismatch between the latter two. The Data Quality Evaluation Scheme (DQES), which forms part of the DQP, specifies the attributes to be evaluated, the corresponding data quality dimensions and metrics, and, after quantitative evaluation, the resulting quality scores. It does not by itself define the preprocessing actions used to improve inadequate quality. Although exploratory profiling can generate preliminary quality-rule proposals earlier in the lifecycle, the formal rule-discovery component is activated when measured quality scores fail to satisfy the defined quality requirements.

The rule generation process described by Taleb et al. (2021) begins when the quality control component detects that a measured quality score fails to satisfy its corresponding requirement. At this point, the rule discovery component analyses the resulting error

report and generates a rule composed of one or more Pre-Processing Functions (PPFs), grouped under a Pre-Processing Activity (PPA) selected from a repository of activities, such as data cleaning or data enrichment. The authors illustrate this process with the example of an attribute exhibiting a 50% missing value rate, compared with a required threshold of 20% or less. The resulting rule is applied to all attributes flagged for preprocessing and may be instantiated through several alternative PPFs, such as replacing missing values with the mean, the mode, or the median, rather than through a single predetermined correction. This example illustrates that, within the framework proposed by Taleb et al. (2021), a given data quality deficiency is not necessarily associated with a single corrective action, but rather with a set of alternative preprocessing strategies from which an appropriate one must be selected.

Once a set of candidate rules has been discovered, Taleb et al. (2021) describe a validation stage in which the rules are applied to a sample dataset, producing a preprocessed sample. The quality scores of this preprocessed sample are then re-evaluated and compared with those obtained for the original sample. Only the rules that satisfy the validation criterion proceed as valid candidates for subsequent optimization and application, while failed rules may be discarded, adapted, or retained for future re-evaluation.

A further stage, rule optimization, addresses the fact that a collection of individually valid rules cannot, in general, be applied exactly as generated without prior reorganization. Taleb et al. (2021) identify several undesirable consequences that may arise if this step is omitted, including redundant rules, rules that become ineffective because of an inappropriate execution order, and multiple rules that independently target the same quality dimension and attribute under the same quality requirement. Of particular relevance to the present study is the authors' observation that rules removing attributes or records should be assigned higher execution priority than other rules, thereby avoiding the application of other rules or preprocessing operations to data that will subsequently be discarded. The optimization schemes proposed by Taleb et al. (2021) operate along four main axes: execution priority, elimination of redundant rules, grouping of rules by attribute or quality dimension, and the removal or adaptation of invalid rules.

Taken together, these processes form part of the broader DQP lifecycle defined by Taleb et al. (2021). The DQP is initially created when the data quality project is defined (DQP 0) and is progressively enriched through data sampling and profiling (DQP 1), exploratory data quality profiling (DQP 2), the assignment of attributes to quality dimensions and metrics (DQP 3), quantitative quality evaluation (DQP 4), quality control against predefined requirements (DQP 5), rule discovery (DQP 6), rule validation

(DQP 7), and rule optimization (DQP 8). The optimized rules are then applied to the complete dataset through component 9 of the framework, Big Data Pre-Processing, after which continuous quality monitoring produces the quality report recorded as DQP 10. This progressive structure formalizes a principle already introduced in Section 2.4.1: data quality management is not a one-off assessment but an iterative process in which measurement, corrective action, and monitoring constitute distinct yet sequentially connected stages.

For the purposes of the present study, it is important to distinguish the corrective quality rules defined by Taleb et al. (2021), which specify preprocessing actions, from the declarative validation checks developed in Section 4.3. The framework is therefore used as a conceptual reference rather than reproduced as a complete procedural model; Section 4.3 specifies which of its principles are incorporated into the proposed design and how they are adapted to the different role played by validation rules in this study.

4.2 Alternative Approaches to Error and Anomaly Detection

The previous section presented the framework proposed by Taleb et al. (2021) for organizing data quality assessment and corrective actions, but it did not yet address the mechanisms through which violations or unusual observations can be identified in the data. For the purposes of this study, the approaches reviewed in the literature are organized into three broad families: declarative or rule-based approaches, statistical approaches, and machine learning-based approaches. These categories are used here as an analytical structure rather than as a single taxonomy proposed by any one of the reviewed sources. This section characterizes the three alternatives without yet assessing their suitability for the present context; that comparison is reserved for Section 4.3.

4.2.1 Declarative Approaches: Dependency-Based Detection

The declarative approach considered in this study is grounded in relational dependency theory (integrity constraints), which originated largely in database schema design and normalization. Fan (2008) notes that, in recent years, interest has re-emerged in applying this theory to data quality improvement, with inconsistencies being captured as violations of defined dependencies. The author argues that classical dependencies, particularly functional dependencies (FDs) and inclusion dependencies (INDs), have limited expressive power and are therefore insufficient to capture many real-world inconsistencies.

To address this limitation, Fan (2008) presents conditional functional dependencies

(CFDs), which extend traditional functional dependencies by adding a pattern of values that restricts their application to a subset of tuples. The author also presents matching dependencies (MDs), an extension of functional dependencies that can operate across multiple relations and incorporate domain-specific similarity and matching operators to support object identification and deduplication. This additional expressive power comes at a computational cost. Whereas collections of classical FDs and INDs are always satisfiable, the consistency problem is NP-complete for CFDs and becomes undecidable when CFDs and conditional inclusion dependencies are considered together. Finally, Fan (2008) describes three strategies for dealing with detected inconsistencies: data repairing, consistent query answering, and condensed representations of all possible repairs.

4.2.2 Statistical and Machine Learning-Based Approaches

Chandola et al. (2009) define the fundamental principle underlying statistical approaches to anomaly detection: normal instances occur in high-probability regions of a stochastic model, whereas anomalies occur in low-probability regions of the same model. The authors distinguish between parametric techniques, which assume a particular form for the underlying distribution and estimate its parameters from the data, and non-parametric techniques, which do not generally require the underlying distribution to be specified a priori. Their main advantage is that, when the underlying distributional assumptions are satisfied, they provide a statistically grounded solution with an associated confidence interval. Their main limitation, however, is that these assumptions often fail to hold for real-world, high-dimensional data, as Chandola et al. (2009) point out.

With regard to machine learning-based approaches, Chandola et al. (2009) first review classification-based techniques, which learn a classifier (either multiclass or one-class) capable of distinguishing between normal and anomalous instances. The authors note that multiclass techniques depend on the availability of accurate labels for the different normal classes, which is often impractical in real-world settings.

Second, they review nearest-neighbour-based techniques, which assume that normal instances occur within dense neighbourhoods, whereas anomalies are located far from their nearest neighbours; and clustering-based techniques, which, depending on the specific variant, assume either that normal instances belong to a cluster while anomalies do not belong to any cluster, or that anomalies belong to small or sparsely populated clusters. Nearest-neighbour techniques are generally unsupervised and do not require an explicit model of the data-generating distribution, but their testing phase can be computationally expensive and their performance depends critically on the choice of an appropriate dis-

tance or similarity measure. Clustering-based techniques are also primarily unsupervised, although semi-supervised variants exist. Their performance depends on the ability of the underlying clustering algorithm to capture the structure of normal observations, and their computational cost varies with the clustering method employed.

It should be emphasized that anomaly detection and deterministic data quality validation address related but non-equivalent problems. In the terminology of Chandola et al. (2009), an anomaly is an observation whose behaviour differs from an expected pattern; such an observation is not necessarily erroneous. Conversely, a value may violate an explicit business rule even when it is not statistically unusual. This distinction becomes relevant when selecting the validation paradigm for the present study in Section 4.3.

4.2.3 Empirical Evidence from Data Quality Tools

Beyond the theoretical characterization of these alternative paradigms, recent literature provides empirical evidence on the concrete error-detection functionalities implemented in data quality tools. Papastergios et al. (2026) examine the source code of seven open-source tools (dbt Core, Deequ, Evidently, Great Expectations, Griffin, MobyDQ, and Soda Core) and identify twenty-five low-level functionalities, that is, concrete data verification checks, which they subsequently map to six selected dimensions of the ISO/IEC 25012 standard.

The authors group these functionalities into five categories: conformance checks, subdivided according to the level of granularity at which they operate, from individual values to entire tables; distribution-based checks; volume and cardinality checks; correlation checks; and machine learning-oriented checks.

This last category explicitly encompasses anomaly detection. Papastergios et al. (2026) find that only one of the seven tools examined implemented this functionality, and that the implementation relied on a broad range of heuristic detection strategies rather than ML-based methods, with support for both batch and streaming data. It is important to note that the authors explicitly caution that they were unable to verify the concrete implementation of this functionality in other tools that advertise it as an advanced paid feature, since their study was limited to examining publicly available source code. Within the same category, however, they do document both statistical and machine learning-based variants for data drift detection.

4.3 Design of the Validation Rule Catalogue

Sections 4.1 and 4.2 have reviewed, respectively, the framework proposed by Taleb et al. (2021) for structuring data quality evaluation, control, corrective rule generation and monitoring, and the three families of detection approaches used as the analytical structure of this study: declarative, statistical, and machine learning-based approaches. This section draws on these two reviews to justify the specific design adopted in this study. First, it delineates which elements of the framework proposed by Taleb et al. (2021) are incorporated into the design. Second, it justifies the choice of a declarative approach over statistical and machine-learning-based alternatives. Third, it presents the proposed rule catalogue and its connection to the data quality dimensions reviewed in Chapter 2. Finally, it justifies the proposed execution order within the pipeline architecture established in Chapter 3.

The literature reviewed in Sections 4.1 and 4.2 is used here as a design basis rather than as a prescriptive specification of the proposed system. Where the reviewed frameworks provide explicit methodological principles that are applicable to the present context, these are incorporated directly into the design. Where the literature identifies relevant design considerations but does not prescribe a specific implementation choice, the corresponding decision is adapted to the requirements of the application context and its implications and trade-offs are stated explicitly.

Before discussing the rationale for the adopted paradigm, it is useful to clarify which aspects of the framework proposed by Taleb et al. (2021), reviewed in Section 4.1, are incorporated into the design of this study and which are not. In the formal rule-discovery path of their framework, corrective quality rules are generated when a measured quality score fails to satisfy the corresponding requirement, although exploratory profiling may generate preliminary rule proposals earlier in the lifecycle. The resulting corrective rules are subsequently validated on a sample before being applied to the full dataset. In the context of this study, by contrast, the proposed validation rules are not derived from a measured degree of quality but from business requirements known a priori and expressed as deterministic pass/fail conditions: for example, an amount either satisfies the requirement of being non-negative or violates it. The quality problems targeted by the catalogue therefore do not require automatic discovery from the data, since the corresponding conditions follow directly from the business rules governing the reporting process.

This distinction implies that the framework of Taleb et al. (2021) is incorporated here as a conceptual foundation rather than as a procedure to be reproduced. In particular, the separation between quality evaluation, quality control, and corrective action provides a

useful basis for distinguishing the detection of a violation from the response subsequently taken to it. The validation rules proposed in this study therefore do not constitute direct equivalents of Taleb et al. (2021)'s corrective quality rules: they operate as declarative checks, while retention and notification are handled separately once a violation has been detected. Neither automatic rule discovery nor sample-based validation of corrective rules is incorporated, since the validation catalogue is explicitly defined during the design phase. The optimization principles proposed by Taleb et al. (2021), particularly execution priority, are considered later as a reference for organizing the validation process, although their direct applicability must be adapted to the different role played by rules in the present system.

The choice of a declarative approach does not rest on a general superiority over statistical or machine learning techniques, which address different forms of anomaly detection under different assumptions. Chandola et al. (2009) show that statistical techniques require an appropriate statistical model of normal behaviour, while supervised classification-based approaches depend on labelled training data; other approaches, such as nearest-neighbour and clustering methods, can operate without anomaly labels but instead rely on assumptions about neighbourhood structure, clustering behaviour, or suitable distance measures. In the present study, however, the target quality problems are explicit business-rule violations known a priori. Conditions such as an amount being non-negative or a foreign key having a corresponding dimension member can be represented directly as deterministic pass/fail constraints, without estimating normal behaviour from the observed data. Statistical and machine learning-based anomaly detection could therefore provide a complementary mechanism for identifying unexpected deviations not covered by the catalogue, but they are not required to detect the specific violations targeted by the proposed system.

This methodological argument can be complemented by the empirical evidence reported by Papastergios et al. (2026). In the seven open-source tools examined by the authors, low-level conformance checks such as range or set constraints, regular-expression checks, row-level comparisons, and matching between tables are broadly represented, whereas general anomaly-detection functionality is identified in only one tool and is implemented through heuristic rather than machine learning-based strategies. This evidence does not imply that statistical or machine learning methods are generally unsuitable for data quality assessment, but it does show that deterministic conformance checks constitute a well-established practical mechanism for implementing explicit data validation requirements.

The specific composition of the catalogue also reflects a criterion of direct measurability. Ehrlinger and Wöß (2022) argue that, rather than starting from the abstract quality dimensions proposed in the academic literature, which neither enjoy universal consensus nor are always measurable in practice, it is more effective to focus quality assessment on directly measurable aspects, such as missing values and duplicate records. The authors further observe that completeness and uniqueness are the two most broadly implemented aspects, precisely because they can be calculated without requiring an external reference dataset, whereas accuracy is the least supported dimension and is implemented only when such an external reference is available. Applied to the present study, this criterion motivates selecting checks that can be evaluated without an external "gold standard" dataset. The six proposed rules rely instead on the business constraints defined for the pipeline, the relational structure of the data, and, where relevant, execution metadata such as the current processing date. This distinction is particularly relevant because the design does not assume the availability of a validated external dataset against which incoming records could be compared.

4.3.1 Proposed Rule Catalogue

Building on the declarative approach justified above, the proposed catalogue comprises six validation rules, each of which operationalizes a data quality concept reviewed in Chapter 2 as a deterministic check adapted to the present pipeline. Following Batini et al. (2009), completeness is understood as the degree to which a data collection includes the data required to describe the corresponding set of real-world objects. In the present system, this dimension is operationalized as the absence of null values in attributes defined as mandatory for reporting, such as `total_amount`. Format conformance is grounded in the concept of syntactic accuracy described by Batini et al. (2009), which concerns whether a stored value conforms to its admissible definition domain; here, it is operationalized by verifying that `date_raw` follows the required date pattern. Consistency is defined by Batini et al. (2009) in terms of violations of semantic rules over the data, including both inter-relational and intra-relational integrity constraints. On this basis, referential integrity verifies that every foreign-key value in the fact table has a corresponding entry in the associated dimension table, while the value-range rule verifies that a numerical attribute remains within the admissible interval defined by the corresponding business requirement. Uniqueness is implemented in this study as exact duplicate-tuple detection at the level of the fact table. This implementation should be distinguished from the attribute-level uniqueness metric documented by Ehrlinger and Wöß (2022); what supports the proposed

check is the detection of exact duplicate tuples, which the authors include separately in their catalogue of profiling requirements. In a real deployment, duplicate detection must therefore be defined consistently with the declared business grain and the identifiers available for each business event, since two records with identical descriptive and numerical values may still correspond to distinct legitimate events. Finally, currency refers to how up to date the data are with respect to the real-world state they represent (Batini et al., 2009). In this study, it is operationalized at load level by requiring the load date to match the pipeline execution date, consistently with the daily execution cycle justified in Section 3.3.1.

Table 1 summarizes the proposed catalogue.

Rule	**Theoretical grounding**	**Example**
Completeness	Completeness dimension (Wang & Strong (1996); Batini et al. (2009))	Absence of null values in `total_amount`
Format conformance	Syntactic accuracy (Batini et al., 2009)	Valid date format
Referential integrity	Consistency: inter-relational integrity constraint (Batini et al., 2009)	`id_product` present in `dim_product`
Value range	Consistency: intra-relational integrity constraint (Batini et al., 2009)	`total_amount` not negative
Uniqueness / duplicates	Exact duplicate detection as a directly measurable check (Ehrlinger & Wöß, 2022)	Absence of duplicate records
Currency	Currency (Batini et al., 2009)	Load date matches the execution date

Table 1: *Proposed catalogue of validation rules*

Source: Own elaboration based on Wang & Strong (1996), Batini et al. (2009), and Ehrlinger & Wöß (2022).

The two consistency-related rules also illustrate the relationship between the proposed catalogue and the declarative paradigm discussed in Section 4.2.1. Fan (2008) shows how data dependencies, also referred to as integrity constraints, can be used declaratively to specify the semantics of relational data and to identify inconsistencies when those

constraints are violated. Batini et al. (2009), in turn, distinguish between intra-relational and inter-relational integrity constraints in their treatment of consistency. Accordingly, the referential-integrity rule is implemented here as an inter-relational constraint, since it verifies that each foreign-key value in the fact table is matched by a valid key in the corresponding dimension table, whereas the value-range rule constitutes an intra-relational constraint because it verifies a domain restriction within the same relation. This distinction also clarifies their relationship with the data model: the referential-integrity rule is directly enabled by the star-schema structure justified in Section 3.2, while the value-range rule depends only on the business requirement defining the admissible domain of the corresponding attribute.

4.3.2 Proposed Execution Order

The catalogue of six rules presented in Section 4.3.1 does not, by itself, determine the order in which these rules should be executed within the transformation stage of the pipeline architecture established in Chapter 3. Taleb et al. (2021) identify rule prioritization as one of the relevant aspects of quality-rule optimization. In particular, their framework assigns higher execution priority to corrective rules that remove attributes or records, so that subsequent preprocessing operations are not unnecessarily applied to data that will ultimately be discarded. This principle establishes that execution order is a relevant design consideration, but it does not directly determine the sequence adopted in the present study. As discussed in Section 4.1, the rules proposed here are declarative validation checks rather than the corrective preprocessing rules defined by Taleb et al. (2021), and none of them automatically modifies or deletes the data. Their prioritization criterion is therefore used as a methodological reference, while the specific execution sequence is adapted to the characteristics and requirements of the proposed validation system.

The system adopts a fixed sequential fail-fast strategy. The currency rule is evaluated first at load level; once this condition has been satisfied, the remaining rules are applied sequentially at record level. When a record violates one of these rules, it is flagged and retained at the transformation stage, as described in Section 4.4, and is not evaluated by the remaining rules during the same execution. The purpose of this strategy is not to provide an exhaustive diagnosis of every possible violation affecting a record in a single cycle, but to prevent data already known to be unsuitable for reporting from undergoing unnecessary downstream validation and from reaching the reporting layer. The fail-fast strategy therefore constitutes a context-specific implementation decision rather than a

procedure prescribed by the reviewed literature.

Within this strategy, the first two positions have a direct operational justification. Currency, which concerns how up to date the data are with respect to the real-world state they represent (Batini et al., 2009), is evaluated first because, in the present system, it is operationalized at load level rather than at individual-record level. If the load does not correspond to the expected processing date, the complete load is retained and a new load must be requested; applying record-level checks to it during that execution would therefore provide no benefit for determining whether those records may proceed to reporting. The exact-duplicate detection rule is evaluated second. Ehrlinger and Wöß (2022) explicitly include exact duplicate tuple detection among the profiling requirements identified in their survey. In the present implementation, exact duplicate copies are retained before subsequent record-level checks are executed. Since these copies contain the same values as records already present in the dataset, evaluating them independently would duplicate the same validation work. This placement is consistent with the efficiency rationale underlying the prioritization principle discussed by Taleb et al. (2021), although its implementation here is adapted to a retention-based rather than an automatic-correction mechanism.

The remaining four rules operate on different properties of the records and do not form a general chain of logical dependencies. One dependency is nevertheless explicit: completeness must precede a value-range check when both operate on the same attribute, since the latter presupposes that a value is present. In the simulated implementation, this applies to `total_amount`. By contrast, format conformance is evaluated on `date_raw`, referential integrity on the foreign keys linking the fact and dimension tables, and value range on `total_amount`; failure of one of these checks does not generally prevent the others from being evaluated. Their relative position is therefore fixed to provide a deterministic and reproducible processing sequence rather than because the literature establishes a strict dependency among them. Under the fail-fast strategy, changing the relative order of these independent checks would affect which violation is reported first when a record violates more than one rule, but not the decision to retain that record once any violation has been detected.

Table 2 summarizes the resulting order.

A consequence of the fail-fast strategy is that a record violating more than one rule is associated only with the first violation encountered during a given execution. Once that violation has been corrected at the source and the record re-enters the pipeline, a subsequent violation may therefore be detected in a later cycle. This represents an explicit trade-off in the proposed design: the strategy simplifies the validation flow and

Order	Rule	Justification
1	Currency	Load-level gate. If the load does not correspond to the expected processing date, the complete load is retained and record-level validation is not performed during that execution.
2	Uniqueness / duplicates	Exact duplicate copies are retained before subsequent checks, avoiding repeated evaluation of identical information.
3	Completeness	Missing mandatory values are detected before checks that require those values to be present, most directly the value-range check applied to `total_amount`.
4	Format conformance	Evaluates the syntactic conformity of `date_raw`. Its position relative to the remaining independent checks forms part of the fixed deterministic sequence rather than a strict logical dependency.
5	Referential integrity	Evaluates the validity of the foreign-key relationships defined by the star schema. Its relative position is an implementation choice within the fixed fail-fast sequence.
6	Value range	Evaluates whether `total_amount` lies within its admissible domain. It follows completeness because this check presupposes that the amount is present.

Table 2: *Proposed execution order for the validation rule catalogue*

Source: Own elaboration based on Batini et al. (2009), Ehrlinger & Wöß (2022), and Taleb et al. (2021), as discussed in Sections 4.3.1 and 4.3.2.

avoids further processing of records already classified as unsuitable for reporting, at the cost of providing a less exhaustive diagnosis within a single execution. An alternative implementation could evaluate all applicable rules and accumulate multiple violations for each record; this possibility is considered as a potential extension in Chapter 5.

Finally, the proposed execution sequence should not be interpreted as a ranking of the business importance or severity of the six rules. The fitness-for-use criterion introduced

in Section 2.1 explains why each rule is included in the catalogue and why its violation is sufficient to prevent the affected data from reaching the reporting layer, but it does not determine the relative order among otherwise independent checks. If left undetected, a missing or out-of-range amount may compromise numerical aggregation; a duplicate record may overstate the corresponding indicator; a non-conforming raw date cannot be reliably interpreted for temporal processing; an unresolved foreign key prevents reliable attribution of a measure to its descriptive dimension; and an outdated load may produce internally consistent but no longer current information. In each case, the relevant issue is that the data fail to satisfy a requirement necessary for their intended use in management control reporting.

4.4 Alert Mechanism and Handling of Detected Records

The design decisions established in Section 4.3 determine which conditions are evaluated and the sequence in which they are applied, but they do not by themselves prescribe how a detected violation should be handled. This section therefore defines the response adopted in the proposed system, which consists of four elements: the treatment applied to the affected record, the notification generated, the accumulated log of detected violations, and the return of corrected records to the pipeline cycle.

4.4.1 Record Handling: Flagging and Retention

None of the six rules in the catalogue applies an automatic correction to the detected value or removes the record. When a violation is detected, the system flags the affected record and retains it at the transformation stage, so that it is not included in the second load and therefore does not reach the reporting layer until the violation has been reviewed and corrected. The currency rule constitutes the exception anticipated in Section 4.3.2: since all records in a load share the same load date, its violation affects not an individual record but the entire load, which is therefore retained as a whole. This behaviour is conceptually consistent with the quality-control function described by Taleb et al. (2021), in which measured quality results are evaluated against defined requirements and unsatisfied requirements trigger further quality actions rather than being treated as acceptable results. The specific retention mechanism adopted here, however, is a context-specific implementation decision and should not be interpreted as a mechanism prescribed by Taleb et al. (2021). It is also consistent with the process-control concept described by Batini et al. (2009), which introduces checkpoints within data-production processes to monitor

quality during process execution. Applied to the present pipeline, the transformation stage functions as such a checkpoint. The same design can be interpreted through the data-manufacturing metaphor introduced by Wang et al. (1995): in the present system, data that violate a validation rule are prevented from advancing to the reporting layer until the corresponding issue has been addressed.

The decision to retain the record rather than allow it to proceed with a warning flag follows directly from the objective of the proposed system. Allowing a record that has already been identified as violating a validation rule to enter the reporting layer would not satisfy the objective established in Chapter 1 of preventing detected violations from reaching management-control reports. Retention therefore ensures that only records that have passed the implemented validation catalogue are delivered to the reporting layer. This choice is consistent with the situation described in Chapter 1 and Section 2.4.3, in which quality problems became apparent only when a business user reviewed the final report and identified an inconsistency.

4.4.2 Notification

All rules in the catalogue generate the same type of notification, and no differentiated severity hierarchy is incorporated into the present design. This choice should not be interpreted as a consequence of the binary nature of the validation rules: although each rule produces a pass/fail outcome, the business impact of different violations could still differ. Rather, severity levels are omitted because the present design does not define business-impact thresholds or differentiated response procedures, and every detected violation leads to the same immediate handling decision, namely retention. The notification therefore focuses on identifying the violated rule and the affected element: the corresponding record or records in the case of record-level rules, or the complete load in the case of the currency rule. A future implementation could incorporate differentiated severity levels if the organization defined explicit impact criteria and corresponding escalation procedures.

4.4.3 Accumulated Log

In addition to the one-off notification, the outcome of each validation step is recorded in a centralized log that persists across executions. For each rule, the log records the execution identifier, the number of records evaluated, the number of detected violations, the scope of the check, and, for record-level violations, the identifiers of the affected

records. This persistent history gives substance to the monitoring aspect of the framework adopted in Section 2.4.1. Ehrlinger and Wöß (2022) characterize automated data quality monitoring through capabilities including the periodic scheduling of quality tasks and the storage, retrieval, comparison, and visualization of their results over time. Without persistent execution results, the daily operation of the pipeline would produce a succession of isolated checks rather than a comparable history. As argued in Section 3.3.1, this monitoring loop is closed once per execution cycle, which is sufficient for the purpose described here without requiring real-time operation. The usefulness of this history is illustrated in Section 4.5, where a load retained due to a violation of the currency rule and its subsequent retry are recorded as separate executions.

It should be noted that this log does not constitute a Data Quality Profile in the sense defined by Taleb et al. (2021), for the reasons outlined in Section 4.3: it does not incorporate quantitative quality scores for individual attributes nor feed into any mechanism for the automatic generation of rules. Its function is instead documentary and monitoring-oriented, while extending it towards a structure of this type remains a possible avenue for future work.

4.4.4 Return of Retained Records to the Pipeline Cycle

Retention does not constitute a permanent state, but rather a temporary interruption until the detected violation has been addressed. In the proposed design, correction is performed on the source side rather than by modifying the data stored in the pipeline. The first loading layer preserves a minimally processed representation intended to maintain source fidelity and provide a traceable record of what was received, as established in Section 3.1; modifying this representation would therefore weaken the separation between source data and downstream transformation logic. This decision should not be interpreted as implying that correcting an individual value in the source system constitutes process redesign or necessarily removes the root cause of the problem. As clarified in Section 2.3.4, the proposed pipeline controls the propagation of detected quality problems, while remediation of their underlying causes remains the responsibility of the corresponding source process. Once the affected source data have been corrected, the record may re-enter the pipeline in the following load and is re-evaluated against the complete catalogue in the order established in Section 4.3.2, without requiring a separate reintegration mechanism.

A consequence of this design should be made explicit. While a record remains retained, it is not included in the reporting layer. The resulting report therefore contains only records that have passed the implemented validation catalogue, but it may be in-

complete with respect to the transactions received from the source systems. The design thus prioritizes the exclusion of detected rule violations over completeness of the reporting population. This should not be interpreted as a guarantee of absolute correctness, since quality problems outside the scope of the catalogue—most notably semantic inaccuracies—may remain undetected, as discussed in Chapter 5. The omission of retained records is nevertheless traceable through the accumulated monitoring history until the corresponding source-side correction has been made.

4.5 Illustration of the Complete System Using Simulated Data

This section illustrates the combined operation of the validation rule catalogue justified in Section 4.3, the fail-fast execution sequence established in Section 4.3.2, and the record-handling and monitoring mechanisms described in Section 4.4. To this end, the simulated dataset presented in Section 3.4 is extended with the additional attributes, records, and deliberately introduced violations required to exercise the complete six-rule catalogue. As with the previous example, this illustration is purely demonstrative: it is intended to show the behaviour produced by the design decisions established in the preceding sections, rather than to evaluate the system's detection performance, false-positive rate, or computational efficiency on real data.

- **The Dataset.** The fact table contains 29 records. The first 25 preserve the structure and record identifiers of the simulated dataset introduced in Section 3.4, but are modified here by deliberately introducing three missing values in `total_amount`, two non-existent product references, two negative amounts, and two non-conforming values in `date_raw`. Four additional records are then created as exact copies of records already present in the dataset. In this simulation, `id_sell` is treated as a source-event identifier rather than as a database-enforced primary key during the validation stage; the duplicated rows therefore represent the repeated ingestion of the same source event rather than distinct business events sharing the same identifier. Two of these duplicated records also contain a violation already present in the corresponding original record, making the interaction between duplicate detection and subsequent validation rules observable. All records share the same load date, since `load_date` identifies the load rather than an individual business event.

- **First Execution: Out-of-Date Load.** In the first execution, the load arrives with a date earlier than that of the pipeline execution, simulating a situation in which the expected daily update has not occurred. The currency rule therefore fails

and, in accordance with the load-level handling defined in Sections 4.3.2 and 4.4.1, the complete load is retained. None of the remaining five rules is executed: the 29 records are not evaluated individually and none reaches the reporting layer. This execution illustrates the operational rationale for placing the load-level currency check before the record-level rules, since once the load has been classified as unsuitable for the current reporting cycle, further record-level validation cannot change the decision to retain it during that execution.

- **Second Execution: Retry.** Once a new load is requested from the source system, it arrives with the date corresponding to the pipeline execution day. The currency rule is therefore satisfied and the five record-level rules are executed according to the fail-fast sequence established in Section 4.3.2. Table 3 summarizes the resulting number of records evaluated and retained at each step.

Order	Rule	Records evaluated	Violations
1	Currency	29	0
2	Uniqueness	29	4
3	Completeness	25	3
4	Format conformance	22	2
5	Referential integrity	20	2
6	Value range	18	1

Table 3: *Results of the second execution, by rule execution order*

Source: Own elaboration from the simulated executions described in Section 4.5.

The two violations detected by the referential integrity rule correspond to the product dimension; the check against the time dimension detects none, since all foreign keys are resolved correctly. This illustrates that the rule operates over the set of relationships defined by the schema rather than over a single relationship.

The same detected violations also trigger the notification mechanism defined in Section 4.4.2. In the simulated implementation, notifications are represented as console alerts rather than through a specific external delivery technology, since the purpose of the example is to illustrate the information generated by the mechanism rather than its integration with a particular communication service. Each notification identifies the pipeline execution, the violated rule, and the affected records; in the case of the currency rule, whose scope is the load as a whole, the notification identifies the complete load as the affected element. The same validation result is subsequently

stored in the accumulated log described in Section 4.4.3, while the affected records are retained according to the mechanism established in Section 4.4.1.

The number of records evaluated decreases at each step because, under the fail-fast strategy defined in Section 4.3.2, records retained after violating one rule are not subjected to the subsequent checks during the same execution. The retention of these records at the transformation stage follows the handling mechanism described in Section 4.4. The simulated results therefore illustrate two consequences of the execution strategy established in Section 4.3.2.

The first concerns the effect of applying exact-duplicate detection before the remaining record-level checks. The completeness rule detects three violations after the four redundant copies have been retained. If completeness were instead evaluated on the complete 29-record load, it would detect four missing values, because one of the duplicated records reproduces a record that already contains a missing `total_amount`. The example therefore illustrates how resolving exact duplicates first avoids counting and evaluating the same underlying information more than once.

The second concerns the interaction between referential integrity and value range under the fail-fast strategy. The value-range rule detects only one violation even though two negative amounts were deliberately introduced into the dataset. The second negative amount belongs to a record that also contains an invalid product reference and has therefore already been retained by the referential-integrity rule, which precedes value range in the fixed sequence. The example thus illustrates that, when a record violates more than one independent rule, the execution order determines which violation is reported first and recorded for that record during a given execution. This is a direct consequence of the fail-fast design: once one violation has been detected, that record is retained and is not evaluated by subsequent rules during the same cycle.

This behaviour also means that a record containing multiple violations may require more than one execution cycle before all catalogue-defined problems affecting it become visible. If the source-side action consists of removing a genuinely redundant or invalid record, it does not re-enter the pipeline and subsequent checks are no longer applicable to that record. If, instead, the first detected problem is corrected and the record re-enters in a subsequent load, it is evaluated again from the beginning of the catalogue and may then proceed far enough in the sequence for an additional violation to be detected. The mechanism therefore allows further catalogue-defined

violations to become visible across successive cycles, provided that the record is corrected and resubmitted. It should not, however, be interpreted as guaranteeing the detection of every possible data quality problem, since issues outside the scope of the implemented catalogue remain undetectable by the proposed system.

- **Overall Result.** Table 4 summarizes the outcome of the records in each of the two executions.

Execution	Records received	Records retained	Records in reporting
1	29	29	0
2	29	12	17

Table 4: *Record outcome in each execution*

Source: Own elaboration from the simulated executions described in Section 4.5.

In the second execution, twelve records are retained by the validation process and seventeen reach the reporting layer. Four of the retained records correspond to redundant copies, whose resolution consists of preventing the duplicated occurrence from being reproduced in the load. The remaining eight correspond to records that violate at least one of the implemented validation rules and therefore require source-side review or correction before they can proceed to reporting. At least one of these records contains more than one catalogue-defined violation and may consequently require more than one execution cycle for all of those violations to become visible under the fail-fast strategy described above. The resulting reporting dataset therefore contains only records that have passed the implemented validation catalogue, but it is incomplete with respect to the 29 records received in the load. This should not be interpreted as a guarantee of absolute correctness, since data quality problems outside the scope of the catalogue may remain undetected.

Both executions are recorded in the accumulated monitoring log, documenting both the complete retention of the first load due to the currency violation and the successful resolution of that load-level issue in the subsequent retry. With the record-level traceability defined in Section 4.4.3, the same log also records the rules violated and the affected records during the second execution, thereby providing a persistent history across pipeline cycles rather than a set of isolated validation results.

The complete code used to generate the simulated dataset and implement the validation pipeline is provided in Appendix B.

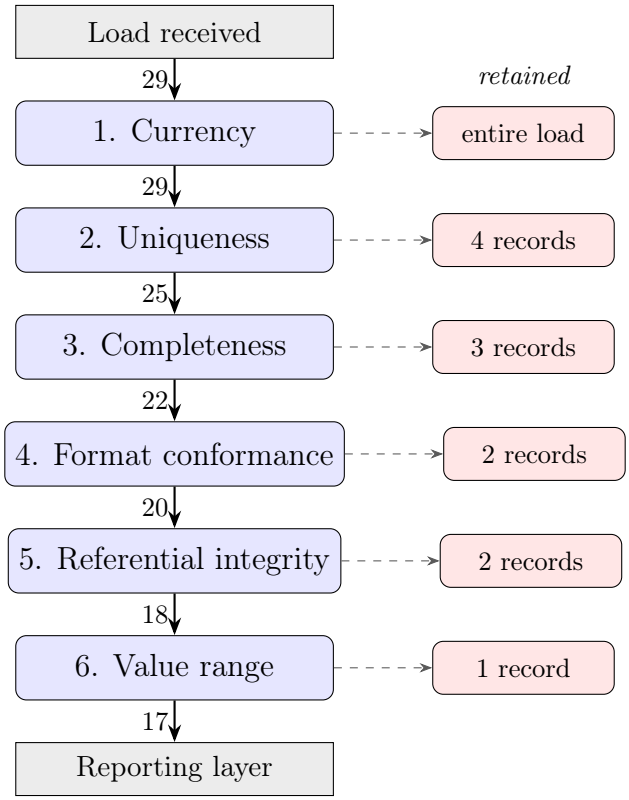

Figure 3: *Execution order of the validation rule catalogue and its retention mechanism. The record-level counts correspond to the second execution reported in Section 4.5, while the currency rule is shown as a load-level gate.*

Source: Own elaboration.

5 Discussion and Conclusions

This work started from the issue identified in Chapter 1: in a management control department without a unified data model or systematic validation mechanism, data quality problems could become visible only at the end of the data-processing chain, after they had already propagated to a reporting output and could therefore affect subsequent decisions. From this starting point, the study has designed and academically justified a data pipeline with an integrated validation system: an ELTL architecture with a source-oriented raw layer that supports traceability (Section 3.1), a star schema for the reporting layer (Section 3.2), a daily execution cycle (Section 3.3.1), a catalogue of six declarative validation rules with a fixed fail-fast execution sequence (Section 4.3), and mechanisms for retention, notification, and accumulated logging (Section 4.4). The common purpose of these decisions is to move the detection of catalogue-defined violations to the transfor-

mation stage, before the affected data reach the reporting layer. In the terminology of Batini et al. (2009), this use of validation checkpoints during data processing is consistent with process control, whose purpose is to monitor quality during process execution and prevent data degradation and error propagation.

The design decisions are also interdependent. The separation between fact and dimension tables creates the relationships on which the referential-integrity rule operates; the daily execution cycle provides the temporal reference used to operationalize the currency rule and defines the frequency of the monitoring loop; and the fact that the targeted problems are explicit business-rule violations known a priori supports the choice of a declarative validation paradigm. Other decisions, however, remain context-specific rather than logically determined by these properties. In particular, the absence of differentiated alert severities reflects the fact that the present design does not define business-impact thresholds or distinct response procedures, not the binary nature of the rules themselves. This combination of structural dependencies and context-specific choices defines both the internal coherence and the scope of the proposed artefact.

The first limitation is methodological. The system has been illustrated using a simulated dataset containing deliberately introduced violations and has not been evaluated on real organizational data. Section 4.5 therefore shows whether the implemented artefact behaves as specified for the constructed example, but it does not provide evidence about production-scale computational cost, the prevalence of the targeted violations, the frequency of inappropriate alerts, or the operational effectiveness of the system in a real reporting process. Claims made in this study consequently concern the conceptual coherence and internal behaviour of the design rather than its empirical performance.

A second limitation is that the system detects, retains, logs, and reports violations but does not itself correct the affected data. As established in Section 4.4.4, correction is performed on the source side so that the raw layer remains a faithful representation of what was received, while the proposed pipeline itself remains a mechanism for controlling error propagation rather than for eliminating the underlying causes of poor data quality. This places an essential part of remediation outside the artefact. In practice, the mechanism therefore presupposes an organizational workflow linking the recipient of the notification with an identifiable owner of the corresponding source process. The present study neither designs nor evaluates such a governance workflow; without it, retained records may remain unresolved and continue to be absent from the reporting layer.

A third limitation is that the validation catalogue is defined a priori and maintained manually. The proposed design incorporates neither a distinct profiling stage (Section

2.4.1) nor an automatic rule-discovery mechanism. Consequently, it can detect only violations represented by the rules that have been explicitly implemented, while unanticipated patterns or newly emerging quality problems may remain outside its scope. The catalogue must also be revised when the applicable business requirements change. This limitation is consistent with the broader measurement challenge discussed by Ehrlinger and Wöß (2022): even when data quality dimensions are conceptually established, translating them into operational metrics or business rules remains context dependent and requires explicit specification. In the present design, that specification problem is therefore not eliminated but located at the catalogue-design and maintenance stage.

A fourth limitation concerns the range of quality problems that deterministic checks can capture. The six-rule catalogue evaluates directly observable conformance, completeness, duplication, temporal currency, and relational-integrity conditions, but it does not assess whether a syntactically valid value corresponds to the true real-world value it is intended to represent. This is the problem of semantic accuracy discussed by Batini et al. (2009), who note that the methodologies addressing semantic accuracy do not provide specific measurement methods. The practical difficulty of accuracy measurement is also reflected in the survey by Ehrlinger and Wöß (2022), where implemented accuracy metrics are limited and may depend on an external reference or "gold standard" dataset that is often unavailable. More broadly, because the proposed catalogue reacts only to conditions explicitly encoded as rules, it is not intended to identify unexpected changes in the statistical behaviour of otherwise rule-compliant data. Anomaly-detection techniques of the type reviewed by Chandola et al. (2009) could complement the catalogue in this respect by identifying observations or patterns that deviate from an expected notion of normal behaviour, although such deviations would not necessarily constitute data errors.

A fifth limitation follows from the retention policy. While a detected violation remains unresolved, the affected record is excluded from the reporting layer, so the reporting population may be incomplete with respect to the records received from the source systems. As clarified in Section 4.4.4, the fact that the remaining records have passed the implemented catalogue does not guarantee their absolute correctness, since quality problems outside the scope of the catalogue may remain undetected. The design also establishes neither a maximum acceptable retention period nor an escalation mechanism for unresolved violations. In a real deployment, these conditions would need to be defined with report users and source-process owners because prolonged retention can materially affect the interpretation and completeness of reported figures.

A sixth limitation arises from the fail-fast execution strategy adopted in Section 4.3.2.

Once a record violates a rule, it is retained and is not evaluated by subsequent rules during the same execution. Consequently, when a record violates more than one rule, only the first violation encountered in the sequence is reported in that cycle, and additional catalogue-defined violations may become visible only after the initially detected problem has been corrected and the record re-enters the pipeline. This design keeps the validation flow simple and avoids applying later checks to records already withheld from reporting, but it provides a less exhaustive diagnosis in a single execution than an alternative that evaluates all applicable rules and accumulates every detected violation. A future implementation could compare both strategies empirically and adopt multiple-violation reporting if diagnostic completeness proves more valuable than the simplicity of the fail-fast flow.

These limitations suggest several directions for future work. Incorporating a preliminary profiling stage could help reveal unexpected distributions, patterns, missingness, or duplication characteristics that are not represented in the current catalogue, consistently with the profiling activities described by Ehrlinger and Wöß (2022). The accumulated log could also be extended with attribute-level data quality metrics and scores, moving it closer to some of the information managed by the Data Quality Profile proposed by Taleb et al. (2021); implementing the complete DQP lifecycle would, however, additionally require quality requirements, evaluation schemes, rule discovery, validation, and optimization mechanisms that are outside the present design. Finally, an economic assessment could evaluate both the costs associated with poor data quality and the costs of assessment and improvement activities, following the cost perspectives reviewed by Batini et al. (2009), thereby providing a basis for prioritizing future data quality investments.

Overall, the study shows how a validation system can be designed without assuming that a general-purpose data quality tool provides a complete out-of-the-box solution, by making explicit the contextual requirements and design decisions that the artefact must implement. Fadlallah et al. (2023) define data context in terms of information about the data itself, organizational policies, domain-specific business rules, and the system resources available to process the data. The present design explicitly incorporates only part of this broader notion of context: metadata and relational structure, organizational reporting requirements, and domain-specific business rules inform the validation catalogue, whereas resource-aware adaptation is not modelled. The design should therefore be described as context-specific rather than comprehensively context-aware. This specificity allows the validation rules to be closely aligned with the intended reporting use, but it also limits portability: applying the same procedure in another organizational setting

would require the relevant context and business rules to be specified again. This contrasts with the direction advocated by Fadlallah et al. (2023), whose proposed context-aware framework seeks generic context properties that allow a solution to operate across different domains and contexts.

The procedure followed to derive the catalogue can be summarized in five design steps. First, the target quality problems are characterized according to whether they can be expressed as explicit business requirements known a priori or instead require abnormal behaviour to be inferred from the observed data. Second, a detection paradigm is selected in light of that characterization, using the declarative, statistical, and machine learning-based alternatives reviewed in Section 4.2. Third, candidate checks are restricted to properties that can be evaluated with the information available to the pipeline, without assuming an external "gold standard" dataset. Fourth, each selected check is related to a data quality concept, metric, or integrity-constraint type reviewed in Chapter 2, while its concrete operational definition is adapted to the business requirements of the application context. Fifth, the checks are organized into a deterministic fail-fast sequence. Taleb et al. (2021) provide the methodological basis for treating execution priority as a relevant design consideration, but the specific sequence adopted here is context-specific: currency operates first as a load-level gate, exact duplicate detection follows to avoid repeated validation of identical copies, completeness precedes value range where both operate on the same attribute, and the relative order of otherwise independent checks is fixed for deterministic execution. The handling of detected violations constitutes a separate design decision: records that fail a rule are retained, notified, and logged rather than automatically corrected or allowed to proceed to the reporting layer.

As stated in Chapter 1, the contribution of this work therefore lies not in the specific six-rule catalogue but in the documented design procedure through which it is constructed: a sequence that moves from business requirements to an integrated validation mechanism within a defined data architecture. The procedure uses the reviewed literature to identify concepts, alternative approaches, and methodological principles, while implementation choices not prescribed by that literature are made explicitly in relation to the application context and their trade-offs are acknowledged. The resulting artefact should therefore be understood as a literature-grounded and context-adapted design rather than as a direct implementation of any single framework.

References

Batini, C., Cappiello, C., Francalanci, C., & Maurino, A. (2009). Methodologies for data quality assessment and improvement. *ACM Computing Surveys*, *41*(3), 16:1–16:52. https://doi.org/10.1145/1541880.1541883

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, *41*(3), 15:1–15:58. https://doi.org/10.1145/1541880.1541882

Davenport, T. H. (2006). Competing on analytics. *Harvard Business Review*, *84*(1), 98–107, 134.

Ehrlinger, L., & Wöß, W. (2022). A survey of data quality measurement and monitoring tools [Article 850611]. *Frontiers in Big Data*, *5*. https://doi.org/10.3389/fdata.2022.850611

Fadlallah, H., Kilany, R., Dhayne, H., El Haddad, R., Haque, R., Taher, Y., & Jaber, A. (2023). Context-aware big data quality assessment: A scoping review [Article 25]. *ACM Journal of Data and Information Quality*, *15*(3), 1–33. https://doi.org/10.1145/3603707

Fan, W. (2008). Dependencies revisited for improving data quality. *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 159–170. https://doi.org/10.1145/1376916.1376940

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, *28*(1), 75–105. https://doi.org/10.2307/25148625

Kimball, R., & Ross, M. (2013). *The data warehouse toolkit: The definitive guide to dimensional modeling* (3rd ed.). John Wiley & Sons.

Merchant, K. A., & Van der Stede, W. A. (2017). *Management control systems: Performance measurement, evaluation and incentives* (4th ed.). Pearson.

Papastergios, V., Ehrlinger, L., & Gounaris, A. (2026). Unfolding data quality dimensions in practice: A survey [Article 1]. *ACM Journal of Data and Information Quality*, *18*(1), 1–37. https://doi.org/10.1145/3786328

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Redman, T. C. (1998). The impact of poor data quality on the typical enterprise. *Communications of the ACM*, *41*(2), 79–82. https://doi.org/10.1145/269012.269025

Rucco, C., Saad, M., & Longo, A. (2025). Formalizing ETLT and ELTL design patterns and proposing enhanced variants: A systematic framework for modern data engineering [Preprint]. https://doi.org/10.48550/arXiv.2511.03393

Taleb, I., Serhani, M. A., Bouhaddioui, C., & Dssouli, R. (2021). Big data quality framework: A holistic approach to continuous quality management [Article 76]. *Journal of Big Data*, *8*(1). https://doi.org/10.1186/s40537-021-00468-0

Wang, R. Y., Storey, V. C., & Firth, C. P. (1995). A framework for analysis of data quality research. *IEEE Transactions on Knowledge and Data Engineering*, *7*(4), 623–640. https://doi.org/10.1109/69.404034

Wang, R. Y., & Strong, D. M. (1996). Beyond accuracy: What data quality means to data consumers. *Journal of Management Information Systems*, *12*(4), 5–33. https://doi.org/10.1080/07421222.1996.11518099

A Simulated Example Code (Section 3.4)

The Python code presented below generates the simulated star schema described in Section 3.4 and illustrates the fact-to-dimension referential integrity checks enabled by its structure. The example reflects the dimensional modelling decisions justified in Sections 3.2 and 3.3; the complete catalogue of validation rules is developed and applied in Chapter 4.

```
import pandas as pd

# 1. Dimension table: products
dim_product = pd.DataFrame({
  'id_product': [1, 2, 3, 4, 5, 6],
  'code_product': ['PRD-001', 'PRD-002', 'PRD-003',
  'PRD-004', 'PRD-005', 'PRD-006'],
  'name_product': ['Product A', 'Product B', 'Product C',
  'Product D', 'Product E', 'Product F'],
  'category': ['Cat1', 'Cat1', 'Cat2', 'Cat2', 'Cat3', 'Cat3']
})

# 2. Dimension table: time
dim_time = pd.DataFrame({
  'id_time': list(range(1, 11)),
  'date': [f'2026-07-{d:02d}' for d in range(1, 11)],
  'day': list(range(1, 11)),
  'month': [7] * 10,
  'year': [2026] * 10
})

# 3. Fact table: simulated sales results
## The 'date_raw' column holds the raw value as received from the
source system, prior to its resolution against dim_time.
fact_results = pd.DataFrame({
  'id_sell': list(range(1, 26)),
  'id_product': [1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1,
  2, 5, 3, 4, 6, 5, 6, 1, 2, 3, 4, 5],
  'id_time': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3,
  4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5],
  'date_raw': [f'2026-07-{d:02d}' for d in
  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
  1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
```

```
      1, 2, 3, 4, 5]],
      'quantity': [10, 5, 8, 3, 12, 7, 9, 4, 6, 11, 2, 15, 8,
      5, 9, 3, 7, 10, 6, 4, 8, 5, 9, 3, 7],
      'total_amount': [100, 50, 85, 30, 120, 70, 90, 45, 60, 110,
      20, 150, 80, 55, 90, 30, 70, 100, 60, 40,
      80, 50, 90, 30, 70]
    })

    # 4. Structural validation
    print("Products resolved:",
    fact_results['id_product'].isin(dim_product['id_product']).all())

    print("Dates resolved:",
    fact_results['id_time'].isin(dim_time['id_time']).all())

```

Listing 1: *Data Validation on a Simulated Star Schema*

B Complete Validation System Code (Section 4.5)

The Python code presented below implements the complete catalogue of six validation rules proposed in Section 4.3, the fail-fast execution sequence justified in Section 4.3.2, and the notification, retention, and accumulated logging mechanisms described in Section 4.4. It extends the simulated dataset of Section 3.4 with the additional attributes, records, and deliberately introduced violations required to exercise all six validation rules, and simulates two consecutive pipeline executions: a first load withheld in its entirety by the currency rule, and a subsequent retry evaluated by the full catalogue. The results are those reported in Section 4.5.

```
    # ================================================================
    # 4.5 Illustration of the complete validation system
    # Extends the star schema of Section 3.4 with the deliberately
    # seeded errors and the six-rule catalogue proposed in Section 4.3.
    # ================================================================
    import pandas as pd

    # ----------------------------------------------------------------
    # 1. Dimension table: products
    # id_product is a surrogate key (Section 3.3.3); the natural key
    # from the source system is kept as a descriptive attribute.
    # ----------------------------------------------------------------
```

```
    dim_product = pd.DataFrame({
      'id_product':   [1, 2, 3, 4, 5, 6],
      'code_product': ['PRD-001', 'PRD-002', 'PRD-003',
      'PRD-004', 'PRD-005', 'PRD-006'],
      'name_product': ['Product A', 'Product B', 'Product C',
      'Product D', 'Product E', 'Product F'],
      'category':     ['Cat1', 'Cat1', 'Cat2', 'Cat2', 'Cat3', 'Cat3']
    })

    # ----------------------------------------------------------------
    # 2. Dimension table: time
    # id_time is likewise a surrogate key; the calendar date is kept
    # as a descriptive attribute alongside its derived components.
    # ----------------------------------------------------------------
    dim_time = pd.DataFrame({
      'id_time': list(range(1, 11)),
      'date':    [f'2026-07-{d:02d}' for d in range(1, 11)],
      'day':     list(range(1, 11)),
      'month':   [7] * 10,
      'year':    [2026] * 10
    })

    # ----------------------------------------------------------------
    # 3. Fact table: 29 records
    # The first 25 reproduce the schema of Section 3.4, into which
    # nine errors are deliberately seeded: 3 null amounts, 2 invalid
    # product references, 2 negative amounts and 2 malformed dates.
    # Records 26-29 are exact duplicates of records 1, 3, 15 and 20.
    # Two of them (3 and 15) also carry a pre-existing error, so that
    # the effect of resolving duplicates first becomes visible.
    # load_date identifies the load, not the individual record.
    # ----------------------------------------------------------------
    dates = ['2026-07-01', '2026-07-02', '2026-07-03', '2026-07-04',
    '2026-07-05', '2026-07-06', '2026-07-07', '2026-07-08',
    '2026-07-09', '2026-07-10', '2026-07-01', '2026-07-02',
    '03/07/2026', '2026-07-04', '2026-07-05', '2026-07-06',
    '2026-7-7',   '2026-07-08', '2026-07-09', '2026-07-10',
    '2026-07-01', '2026-07-02', '2026-07-03', '2026-07-04',
    '2026-07-05']

    base = pd.DataFrame({
      'id_sell':      list(range(1, 26)),
      'id_product':   [1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2, 99,
```

```
      3, 4, 98, 5, 6, 1, 2, 3, 4, 5],
      'id_time':      [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4, 5,
      6, 7, 8, 9, 10, 1, 2, 3, 4, 5],
      'date_raw':     dates,   # raw value as received from the source
      'quantity':     [10, 5, 8, 3, 12, 7, 9, 4, 6, 11, 2, 15, 8,
      5, 9, 3, 7, 10, 6, 4, 8, 5, 9, 3, 7],
      'total_amount': [100, 50, None, 30, 120, 70, 90, None, -60, 110,
      20, 150, 80, None, 90, 30, 70, -100, 60, 40,
      80, 50, 90, 30, 70]
    })

    duplicates = base[base['id_sell'].isin([1, 3, 15, 20])].copy()
    fact_results = pd.concat([base, duplicates], ignore_index=True)

    # ================================================================
    # Validation rules of the catalogue proposed in Section 4.3
    # Each rule receives the records still under evaluation and returns
    # those that violate it. No rule modifies or deletes data: retention
    # is handled by the pipeline that chains them (Section 4.4).
    # ================================================================

    # Rule 1: Currency
    # Operates on the load as a whole: all records share the same load
    # date, so the rule compares it with the pipeline execution date.
    def check_currency(df, execution_date):
      load_date = df['load_date'].iloc[0]
      return load_date != execution_date

    # Rule 2: Uniqueness
    # A duplicate is a record whose columns all match those of another.
    # keep='first' retains one copy and flags the redundant one.
    def check_uniqueness(df):
      return df[df.duplicated(keep='first')]

    # Rule 3: Completeness
    def check_completeness(df, column):
      return df[df[column].isna()]

    # Rule 4: Format conformance
    # Applied to the raw value as received from the source system,
    # before it is resolved against the surrogate key of the dimension.
    def check_format(df, column, pattern):
      return df[~df[column].str.match(pattern)]
```

```
99
100     # Rule 5: Referential integrity
101     # The fact table references both dimensions, so the rule is
102     # evaluated over every foreign key defined by the star schema.
103     # Records violating more than one reference are counted once.
104     def check_referential_integrity(df_facts, references):
105       invalid = [df_facts[~df_facts[fk].isin(dim[pk])]
106       for fk, dim, pk in references]
107       result = pd.concat(invalid)
108       return result[~result.index.duplicated()]
109
110     # Rule 6: Value range
111     def check_range(df, column, minimum=0):
112       return df[df[column] < minimum]
113
114     # ====================================================================
115     # Notification mechanism
116     # Represents the one-off alert defined in Section 4.4.2.
117     # In a production implementation, this function could be replaced
118     # by an email, messaging-service, or dashboard notification.
119     # ====================================================================
120
121     def emit_notification(run_id, rule, scope, affected_records):
122       if scope == 'load':
123         message = (
124         f"ALERT | Run {run_id} | {rule} | "
125         f"affected element: entire load"
126         )
127       else:
128         message = (
129         f"ALERT | Run {run_id} | {rule} | "
130         f"affected records: {affected_records}"
131         )
132
133       print(message)
134
135
136     # ====================================================================
137     # Validation pipeline
138     # Executes the fixed fail-fast validation sequence defined in
    Section 4.3.2.
139     # Records that violate a rule are retained and are not evaluated by
140     # the subsequent rules during the same execution.
```

```
141     # ================================================================
142
143     def run_pipeline(df, dim_product, dim_time, execution_date, run_id,
        log):
144
145       # Rule 1: Currency, evaluated on the load as a whole
146       if check_currency(df, execution_date):
147
148         emit_notification(
149         run_id=run_id,
150         rule='Currency',
151         scope='load',
152         affected_records='entire load'
153         )
154
155         log.append({
156           'run': run_id,
157           'rule': 'Currency',
158           'evaluated': len(df),
159           'violations': len(df),
160           'scope': 'load',
161           'affected_records': 'entire load'
162         })
163
164         return pd.DataFrame(), log
165
166       log.append({
167         'run': run_id,
168         'rule': 'Currency',
169         'evaluated': len(df),
170         'violations': 0,
171         'scope': 'load',
172         'affected_records': []
173       })
174
175       remaining = df.copy()
176
177       # Rules 2-6, evaluated record by record
178       checks = [
179       ('Uniqueness',
180       lambda d: check_uniqueness(d)),
181
182       ('Completeness',
```

```
183         lambda d: check_completeness(d, 'total_amount')),
184
185         ('Format conformance',
186         lambda d: check_format(
187         d, 'date_raw', r'^\d{4}-\d{2}-\d{2}$'
188         )),
189
190         ('Referential integrity',
191         lambda d: check_referential_integrity(
192         d, [
193         ('id_product', dim_product, 'id_product'),
194         ('id_time', dim_time, 'id_time')
195         ]
196         )),
197
198         ('Value range',
199         lambda d: check_range(d, 'total_amount', 0))
200         ]
201
202         for name, check in checks:
203
204           violations = check(remaining)
205
206           affected_records = violations['id_sell'].tolist()
207
208           if len(violations) > 0:
209           emit_notification(
210           run_id=run_id,
211           rule=name,
212           scope='record',
213           affected_records=affected_records
214           )
215
216           log.append({
217             'run': run_id,
218             'rule': name,
219             'evaluated': len(remaining),
220             'violations': len(violations),
221             'scope': 'record',
222             'affected_records': affected_records
223           })
224
225           remaining = remaining.drop(violations.index)
```

```
226
227         return remaining, log
228
229     # ================================================================
230     # Execution 1: the load arrives with an outdated load date.
231     # The currency rule is violated and the entire load is withheld;
232     # the remaining rules are not executed.
233     # ================================================================
234     log = []
235
236     load_run1 = fact_results.copy()
237     load_run1['load_date'] = '2026-07-03'   # date of the received load
238     execution_date_run1 = '2026-07-06'      # simulated execution date
239
240     reporting_run1, log = run_pipeline(load_run1, dim_product, dim_time,
241     execution_date_run1, run_id=1, log=log)
242
243     print("=== EXECUTION 1 ===")
244     print(f"Records received: {len(load_run1)}")
245     print(f"Records withheld: {len(load_run1) - len(reporting_run1)}")
246     print(f"Records reaching the reporting layer: {len(reporting_run1)}"
    )
247
248     # ================================================================
249     # Execution 2: the load is requested again from the source system
250     # and arrives with the correct load date. The currency rule is
251     # satisfied and the remaining five rules are executed in order.
252     # ================================================================
253     load_run2 = fact_results.copy()
254     load_run2['load_date'] = '2026-07-07'   # corrected load
255     execution_date_run2 = '2026-07-07'      # simulated execution date
256
257     reporting_run2, log = run_pipeline(load_run2, dim_product, dim_time,
258     execution_date_run2, run_id=2, log=log)
259
260     print("=== EXECUTION 2 ===")
261     print(f"Records received: {len(load_run2)}")
262     print(f"Records withheld: {len(load_run2) - len(reporting_run2)}")
263     print(f"Records reaching the reporting layer: {len(reporting_run2)}"
    )
264
265     # ================================================================
266     # Accumulated log of detected violations (Section 4.4)
```

```
    # ====================================================================
    print("=== ACCUMULATED LOG ===")
    print(pd.DataFrame(log))

    # ====================================================================
    # Counterfactual: completeness evaluated on the full load, before
    # duplicates have been resolved (Section 4.3.2)
    # ====================================================================
    print("Completeness on the full load:",
    len(check_completeness(load_run2, 'total_amount')))


```

Listing 2: *Complete Validation Pipeline on a Simulated Star Schema*