



UNIVERSITÀ
di **VERONA**

UNIVERSITY OF VERONA

DEPARTMENT OF COMPUTER SCIENCE

PHD IN COMPUTER SCIENCE

From Log Files to Prediction-Oriented Multidimensional Analysis:

conceptual modelling of complex information
to support advanced data analytics

Advisor:

Prof. Carlo Combi

Coordinator:

Prof. Umberto Castellani

Candidate:

Evelina Rakhmetova

Abstract

Keywords: conceptual modelling; event logs; JSON; UML; multidimensional data modelling; temporal data; temporal window; data warehousing; OLAP; SQL; Python; predictive analytics.

Contemporary information systems generate large volumes of data whose structures, semantics, and temporal characteristics are heterogeneous and often only implicitly defined. Event logs capture detailed evidence of operational and business-process execution, but their diversity of formats, vocabularies, nested structures, and execution contexts hinders consistent interpretation, validation, and integration. In parallel, multidimensional analytical environments provide mature support for descriptive analysis, yet their conceptual models typically treat time as a single, uniform role and therefore do not make explicit the distinct temporal assumptions required when preparing data for prediction-oriented tasks.

This thesis investigates how conceptual modelling can be extended to make structural, semantic, and temporal assumptions explicit in these two complementary settings. Following a design-science perspective, the work defines, formalises, implements, and evaluates conceptual artefacts through case-based and implementation-oriented evidence.

The first contribution is a UML-based conceptual modelling approach for heterogeneous system event logs. It introduces log-specific modelling constructs, sup-

ports nested objects and semantic dependencies, aligns terminology with the Elastic Common Schema while enabling domain-specific extensions, and specifies mappings between UML models and JSON log structures. A supporting modelling tool demonstrates bidirectional transformations (forward generation and reverse engineering) together with schema-oriented validation.

The second contribution is a prediction-oriented framework for multidimensional analysis that distinguishes three temporal roles: an Observation Window for feature construction, a Waiting Window for outcome latency, and a Prediction Window for target definition. The framework is formalised via Prediction-Oriented OLAP operators and a Prediction-Oriented Dimensional Fact Model that adds explicit temporal bindings and predictive dependencies.

Python- and SQL-based implementations demonstrate how these temporal semantics can be operationalised in the construction of Analytical Base Tables. The evaluation provides evidence of implementability, reproducibility, and enforcement of the specified temporal constraints in the examined configurations, while not aiming to compare predictive algorithms or claim superior predictive accuracy.

Contents

Abstract	1
1 Thesis Overview	7
1.1 Context	8
1.2 Structure of the Thesis	10
2 Conceptual Modelling of Log Data	12
2.1 Introduction	12
2.2 Background and Related Work	16
2.2.1 Background	16
2.2.2 Related Work	18
2.3 Motivation and Problem Statement	22
2.3.1 Motivating Example from Real-World Log Data	22
2.3.2 Problem Statement	30
2.4 The Proposed Modelling Approach	33
2.4.1 Metamodel for Log Files Modelling	33
2.4.2 Conceptual Model and Key Features Definition	35
2.4.3 Implementation on Industrial Case Studies	44
2.5 Tool Support for Conceptual Modelling of Log Data	46
2.5.1 Design and Functional Scope of the Modelling Tool	46

2.5.2	Examples of Log File Modelling Scenarios	51
2.6	Conclusion	54
3	A Prediction-Oriented Framework for Multidimensional Data Analysis	57
3.1	Introduction	57
3.2	Background and Related Work	62
3.2.1	Foundational Concepts in Multidimensional Data Modelling .	62
3.2.2	Data Modelling of Temporal Data for Analytics	70
3.2.3	prediction-oriented and Temporal Analytics	75
3.2.4	Process Mining and Predictive Monitoring of Business Processes	80
3.3	Motivation and Problem Statement	82
3.3.1	Use Case: prediction-oriented Credit Risk Analysis	82
3.3.2	Limitations of Traditional Multidimensional Models	85
3.4	Fundamental Concepts for Developing a Framework for Multidimensional Analysis	98
3.4.1	The Enhanced Fact Model	98
3.4.1.1	Standard Dimensional Fact Model Definitions	99
3.4.1.2	Proposed Symbolic Notation for Prediction-Oriented Dimensional Fact Model Representation	101
3.4.2	Modelling Inter-Schema Relationships: The Fact Constellation	104
3.4.2.1	Linking Fact Schemes: From Strict Overlap to General Relationships	105
3.4.2.2	The Fact Constellation	106
3.4.2.3	Managing Summarizability and Aggregation Scope .	111
3.4.3	Graphical and Formal Representation of the Dimensional Fact Model for the Credit Risk Scenario	114

3.5	Formalizing the Temporal Framework	122
3.5.1	Formal Definition of the Temporal Windows	123
3.5.2	Prediction-Oriented OLAP Operators	129
3.5.3	Window Configurations and Dynamics	132
3.5.4	Application to Default Risk Prediction Example: Temporal Data Structuring	133
3.6	Defining the PO DFM Framework	142
3.6.1	Graphical Representation of the PO DFM	142
3.6.2	PO DFM Metamodel	148
3.6.3	The PO DFM Formal Definition	149
3.6.3.1	The Single-Fact PO DFM	150
3.6.3.2	The Constellation-Level PO DFM	152
3.6.3.3	Explicit Predictive Dependencies	153
3.7	Implementation and Empirical Validation of the PO Framework . . .	157
3.7.1	Mapping Conceptual Constructs to Executable Artefacts . . .	157
3.7.2	Python Realization	160
3.7.3	SQL Realization: Database-Centric Approach	171
3.8	Conclusion	180
4	Conclusions	182
4.1	Synthesis of the Thesis	183
4.2	Main Scientific Contributions	185
4.3	Evaluation and Scope of the Evidence	186
4.4	Limitations	188
4.5	Future Research Directions	190
4.6	Closing Remarks	192
A	StarUML Extension for Conceptual Modelling of Log Data	193

B Detailed Implementation of the Prediction-Oriented Framework	198
B.1 Python Implementation	199
B.2 SQL Implementation	223
Bibliography	263

Chapter 1

Thesis Overview

The increasing scale, heterogeneity, and temporal complexity of contemporary data have significantly challenged traditional approaches to data modelling and analysis. Modern information systems generate and consume data in diverse forms, ranging from semi-structured system event logs to highly structured multidimensional datasets used for analytical reporting. While data processing technologies and analytical tools have rapidly evolved, the conceptual foundations used to model, structure, and reason about such data have not progressed at the same pace.

Conceptual modelling plays a fundamental role in bridging this gap. By providing abstract, technology-independent representations of data structures, semantics, and relationships, conceptual models support shared understanding among stakeholders, guide system design, and enable systematic analysis. In data analytics, conceptual modelling is particularly valuable for making implicit assumptions explicit, clarifying semantics, and structuring data in ways that align with analytical objectives rather than purely operational constraints.

However, applying conceptual modelling effectively in modern data-intensive contexts remains challenging. Many established modelling approaches were originally developed for relatively stable, well-structured datasets and do not adequately

address characteristics such as schema variability, semantic heterogeneity, and complex temporal dependencies. These limitations become especially evident in two increasingly important analytical scenarios: the management and analysis of system event logs, and prediction-oriented analysis over multidimensional data.

This thesis addresses these challenges by investigating how conceptual modelling can be extended and adapted to support advanced data analytics in the presence of heterogeneity and temporal complexity. The work is structured around two complementary contributions. The first focuses on the conceptual modelling of log data, proposing a formal framework for representing the structure and semantics of system event logs. The second introduces a prediction-oriented conceptual framework for multidimensional data analysis, explicitly incorporating multiple temporal roles to support predictive reasoning. Together, these contributions demonstrate how conceptual modelling can act as an enabling foundation for advanced analytics, even in environments where traditional modelling assumptions no longer hold.

1.1 Context

A central challenge in data analysis is the ability to structure and relate data in a way that supports meaningful interpretation and reasoning. In the context of multidimensional data analysis, this challenge manifests in the representation of relationships between dimensions and facts, as well as in the integration of multiple fact schemas describing related aspects of a domain. Analysts frequently encounter situations in which insights require combining information from different multidimensional cubes, each designed independently and often under different assumptions.

Traditional multidimensional modelling approaches rely on the notion of shared or conforming dimensions as a prerequisite for integration. According to widely

adopted design principles, meaningful navigation across multiple cubes is possible only when dimensions originate from the same instance or share identical schemas and semantics. While this assumption simplifies aggregation and ensures summarizability, it proves overly restrictive in real-world analytical environments, where data sources are heterogeneous, evolve independently, and reflect different temporal perspectives.

When multidimensional cubes do not share common dimensions, analysts face the non-conformity problem. Although such cubes may describe different facets of the same underlying reality, their lack of shared dimensions prevents straightforward integration and limits analytical possibilities. Attempts to overcome this limitation through heterogeneous or loosely coupled schemas often introduce aggregation inconsistencies and obscure semantic relationships rather than resolving them.

Temporal aspects further exacerbate these challenges. In traditional multidimensional models, time is typically represented as a single dimension with a uniform semantic role. This representation is adequate for descriptive and historical analysis but insufficient for prediction-oriented tasks, where different temporal horizons play distinct roles. For example, historical data used to derive explanatory features, latency periods during which outcomes cannot yet be observed, and future intervals in which predictions are evaluated all have different semantic meanings that are not captured by conventional models.

Similar conceptual limitations arise in the analysis of system event logs. Log data is inherently temporal, frequently semi-structured, and generated by heterogeneous systems with varying conventions and semantics. While logs provide detailed traces of system behaviour, their lack of explicit conceptual structure and shared semantics makes systematic analysis, integration, and interpretation difficult. Existing approaches often focus on parsing and storage, leaving higher-level conceptual understanding implicit or entirely absent.

These observations motivate the need for conceptual modelling frameworks that explicitly address heterogeneity, temporal semantics, and analytical intent. Rather than forcing diverse data into rigid, uniform structures, such frameworks should make relationships, roles, and assumptions explicit. This thesis adopts this perspective, positioning conceptual modelling as a foundational mechanism for enabling advanced analytics across both log-centric and multidimensional data environments.

1.2 Structure of the Thesis

This thesis is organized as a progressive investigation of conceptual modelling as a foundation for advanced data analysis under conditions of structural heterogeneity and temporal complexity.

Chapter 2 addresses the conceptual modelling of system event logs. It examines the structural and semantic heterogeneity of log data, reviews related work in log management and conceptual modelling, and introduces a UML-based modelling approach aligned with the Elastic Common Schema while supporting domain-specific extensions. The chapter formalizes the proposed modelling constructs, illustrates them through case studies, and presents tool support for bidirectional transformations between UML models and JSON-based log representations.

Chapter 3 develops the Prediction-Oriented Framework for multidimensional data analysis. It reviews multidimensional and temporal data modelling together with process mining and predictive process monitoring, motivates the need for explicit temporal roles, and formalizes the Observation, Waiting, and Prediction Windows. The chapter introduces the Prediction-Oriented Dimensional Fact Model and Prediction-Oriented OLAP operators, and provides implementation and validation evidence through Python- and SQL-based realizations. Detailed source-code-oriented material is separated from the main scientific narrative and reported in

Appendix B.

Chapter 4 presents the conclusions of the thesis as a whole. It synthesizes the two research contributions, discusses the evidence supporting them, clarifies their scientific and practical implications, positions the work with respect to process mining and predictive process monitoring, and identifies the principal limitations and directions for future research.

Appendix A documents the software artefacts and repositories associated with the thesis. Appendix B provides the detailed Python and SQL implementation material, extended testing procedures, source-code listings, and supporting screenshots required for reproducibility.

Taken together, the chapters develop a unified research argument: explicit conceptual modelling of structure, semantics, and temporal roles provides a disciplined basis for advanced analytics over heterogeneous log data and multidimensional information.

Chapter 2

Conceptual Modelling of Log Data

2.1 Introduction

System event logs constitute a critical, often primary, source of information for monitoring, managing, diagnosing, and securing contemporary information systems [4, 7, 8]. They provide detailed, time-stamped records of activities and events, offering invaluable insights into system behaviour, performance bottlenecks, and potential security incidents [16, 20, 28]. However, the increasing scale and complexity of modern systems, coupled with diverse development practices and reliance on legacy components, frequently lead to log data that is voluminous, heterogeneous, unstructured or semi-structured, and semantically inconsistent [6, 11–13, 15]. This inherent complexity presents significant challenges for effective log parsing, analysis, and comprehension, hindering timely troubleshooting, accurate root cause analysis, and proactive system management [16, 18, 19].

The lack of standardized structure and semantics in many log sources means that raw log messages often exist as free-form text, making automated processing difficult and error-prone [6, 11]. Furthermore, inconsistencies in severity level reporting and ambiguous terminology can pollute analysis results and mislead oper-

ators [15]. While various logging formats (e.g., Syslog, CEF, Windows Event Logs) and schema initiatives (e.g., Elastic Common Schema - ECS) exist [11,20,34], a unifying, high-level abstraction for designing, understanding, and communicating the structure and semantics of log data across different formats and domains remains underdeveloped [11,18,19].

This chapter addresses this gap by proposing a novel conceptual modelling approach specifically tailored for system event logs. Our primary contribution is a framework that integrates the strengths of visual modelling via the Unified Modelling Language (UML) with the structured data representation capabilities of JSON, grounded in the standardized field definitions provided by the Elastic Common Schema (ECS), while explicitly accommodating user-defined extensions. Key novelties of this work include:

- **A UML Profile for Log Data**

We define specific extensions to UML Class Diagrams, including tailored stereotypes, a distinct visual structure for log field sets (differentiating core, extended, and custom fields), and dedicated notations for representing log-specific concepts like nesting levels, field value enumerations, value aggregation, and semantic dependencies between fields.

- **ECS Integration and Extension**

The approach leverages ECS as a foundational semantic layer, promoting standardization, while providing a formal mechanism to incorporate necessary domain-specific or custom fields, thus balancing standardization with flexibility.

- **Bidirectional UML-JSON Mapping**

We establish a clear mapping between the visual UML representation and a structured JSON format, enabling model representation in either paradigm.

- **Tool Support**

We introduce a prototype extension for the StarUML modelling tool, which automates the generation of the JSON representation from the UML model and vice-versa, based on our defined mapping rules, thereby facilitating practical adoption and integration into development workflows.

The conceptual approach is designed to enhance the clarity and interoperability of log data structures for a wide range of stakeholders, including developers, system operators, analysts, and security personnel, regardless of their direct involvement in the initial system design. It supports both top-down creation of new logging strategies and bottom-up analysis, abstraction, and documentation of existing log formats. By offering a clear and structured abstraction of log data, the proposed framework improves overall data quality, facilitates seamless integration with prominent log analysis platforms such as the Elastic Stack (Elasticsearch, Kibana and Logstash: ELK) [24] and Splunk [26], and strengthens Extract, Transform, Load (ETL) pipelines. Ultimately, this contributes to the development of more resilient, maintainable, and scalable centralized log management systems.

This chapter is structured as follows. Section 2.2 discusses the foundational concepts and reviews related work in log management and conceptual modelling. Section 2.3 elaborates on the challenges motivating this research, highlighting issues with existing log practices. Section 2.4 details the proposed conceptual modelling framework, including its metamodel, notation, and key features; Subsection 2.4.3 illustrates the application of the approach using case studies, demonstrating both top-down and bottom-up usage. Section 2.5 describes the supporting software tool developed, discusses different perspectives applicable within the modelling approach

and provides with a practical example in Subsection 2.5.2. Finally, Section 2.6 summarizes the contributions and potential impact of this work.

2.2 Background and Related Work

2.2.1 Background

The conceptual modelling approach proposed in this research leverages the Unified Modelling Language, specifically its class diagrams, as a foundational element. UML class diagrams offer a standardized, visual formalism for representing the structure and relationships within object-oriented systems. This established methodology provides a clear and communicable way to articulate the conceptual structure of log files, thereby facilitating shared understanding, analysis, and collaborative refinement among stakeholders during the log modelling process.

Recognizing the increasing prevalence and utility of structured logging formats, particularly JSON (JavaScript Object Notation), we have extended and adapted the standard UML class diagram notation. This adaptation is designed specifically to capture the hierarchical, nested structures inherent in JSON-formatted log files. By tailoring UML's established notation, developers and analysts can more effectively represent and document the key data elements, their types, and the complex relationships within these nested log structures.

However, a significant challenge persists in practice. Log data is often generated as unstructured or semi-structured text, lacking adherence to consistent standards. This frequently stems from the complexities of industrial development environments and the reliance on legacy systems that may not incorporate modern logging practices. Such inconsistency severely hampers the ability to efficiently extract meaningful insights from log files. Without a defined structure, discerning the relationships between data elements becomes arduous, impeding effective analysis, troubleshooting, and automated processing. Consequently, the interpretation of these logs can become highly challenging for developers, data analysts, or monitoring systems not intimately familiar with their implicit structure and semantics.

A formal conceptual data model for logs addresses these issues by providing mechanisms for both top-down design and bottom-up abstraction. In a top-down approach, the conceptual model serves as a blueprint for designing standardized and well-structured logging systems from the outset. Conversely, in a bottom-up approach, the model provides a high-level, graphical representation (an abstraction) of existing, potentially complex or unstructured, log files. This work explores these dual applications, discussing the inherent challenges and proposed solutions within our modelling framework, illustrated through case studies in banking (top-down) and networking (bottom-up) domains.

Conceptual modelling is therefore positioned as a cornerstone for standardizing log files, which in turn unlocks more powerful analytical capabilities. Implementing such models facilitates a range of data operations, including robust data extraction, transformation, and loading (ETL) pipelines, as well as sophisticated data visualization and analysis. Standardized logs, defined through a conceptual model, enable organizations to reliably compare and aggregate data from diverse sources. This leads to more informed decision-making based on comprehensive insights. Furthermore, conceptual modelling paves the way for integrating log data with other organizational data sources, such as customer records or transaction histories, creating richer datasets for holistic analysis.

This research represents an initial, yet foundational, step towards developing robust, potentially centralized, log file management systems built on sound modelling principles. We adopt the Elastic Common Schema (ECS) as a pragmatic starting point, leveraging its established field sets for common log data elements. Our proposed solution integrates UML diagrams and JSON schema definitions, incorporating both standard ECS fields and allowing for user-defined extensions. To support this, we present an extension for the StarUML modelling tool, which facilitates the creation of log models in either UML or JSON format and automatically

generates the corresponding representation in the other format, enforcing consistency according to our proposed approach.

Modern information systems generate vast quantities of event data, typically captured in logs. These event logs chronicle system activities and state changes, offering invaluable data for understanding system behaviour, detecting anomalies, and identifying performance patterns. However, the sheer volume and often intricate structure of these logs present significant analytical challenges. The complexity of contemporary systems and their associated processes [12, 13, 15] exacerbates this issue, increasing the potential for errors, data loss, or security vulnerabilities like sensitive data leakage [15, 19] to be hidden within the noise.

For decades, log management has been a critical discipline for monitoring system health, diagnosing issues, and mitigating risks [4, 11, 13]. Effective system oversight relies heavily on proactive log monitoring and analysis. Nevertheless, selecting and implementing appropriate log management strategies remains challenging [16, 18]. Key difficulties include choosing solutions capable of handling heterogeneous log formats from diverse sources, ensuring scalability to manage high-volume data streams in near real-time, and structuring the log data itself to enable meaningful analysis – a core problem addressed by our focus on conceptual modelling.

2.2.2 Related Work

Conceptual data modelling is widely recognized in information systems engineering for providing high-level, abstract representations of application domains, including event data [3, 5, 9]. Such models foster a deeper understanding of the domain reality, enhance communication between technical and non-technical stakeholders, and improve the likelihood of accurately meeting user requirements [11]. A key advantage of conceptual modelling is its independence from specific underlying technologies

(e.g., databases, file formats), allowing the same conceptual understanding to be implemented using various logical models.

Despite these benefits, a critical gap exists in the log management domain: the lack of a widely adopted, standardized conceptual model specifically designed for log files. While numerous logical formats and schemas exist (e.g., syslog, JSON-based formats, ECS), a unifying conceptual framework is missing.

JSON has emerged as a *de facto* standard for structured logging due to its characteristics: it is lightweight, human-readable, easily machine-parsable by virtually all programming languages, and inherently supports nested structures. Its flexibility makes it highly suitable for enriching log events with additional context and meta-data, aligning well with the requirements of detailed log management, monitoring, and serving as a potential target representation derived from a conceptual model.

The UML provides mechanisms for extension and adaptation, allowing its core concepts to be tailored for specific domains or purposes [29, 32, 36]. A notable example is SysML, which extended UML constructs like the class to create SysML blocks for systems engineering [32]. This inherent extensibility is the mechanism we leverage to adapt UML class diagrams for log file modelling.

Furthermore, in this research, UML itself serves a dual purpose. Beyond modelling log files (the primary goal), we also utilize UML (specifically, its metamodeling capabilities) to formally define our proposed log modelling approach. That is, we are defining a new type of structural diagram, derived from the class diagram, thereby establishing clear semantics and syntax at the meta-level for the modelling language itself.

This aligns with the metamodeling hierarchy defined by the Meta-Object Facility (MOF), which structures modelling languages [36]. This hierarchy typically includes:

- M3 (Meta-metamodel):

MOF itself, defining the language for specifying metamodels;

- M2 (Metamodel):

Languages like UML, defining constructs for specifying models;

- M1 (Model):

User-defined models (like our specific log file model for a banking application), describing a particular semantic domain;

- M0 (Instances Data):

Concrete instances or log entries conforming to an M1 model.

In describing our modelling concept (the rules and structure of our log modelling diagrams), we operate primarily at the M2 level (defining an extension to UML) and the M1 level (providing the metamodel for log structures). When applying the approach to specific domains (like banking or networking), the resulting models reside at the M1 level, while the actual log files generated or analysed are instances at the M0 level. Leveraging UML’s extension mechanisms inherently involves working across these levels of abstraction [29, 37].

Therefore, this work builds upon established principles of conceptual modelling, leverages the practicality of JSON and the standardization potential of ECS, and utilizes the formal power and extensibility of UML and its metamodeling foundations to address the identified gap - the need for a robust conceptual modelling framework specifically tailored for log files.

The representation of log structure developed in this chapter is related to, but distinct from, process mining. Process-mining methods generally assume that event data can be associated with cases, activities, timestamps, and relevant attributes. The present contribution addresses an upstream modelling problem: how heterogeneous log structures, field semantics, identifiers, and dependencies can be repre-

sented explicitly before process-aware analytical methods are applied. The relationship to process mining and predictive process monitoring is examined in detail in Section 3.2.4.

2.3 Motivation and Problem Statement

2.3.1 Motivating Example from Real-World Log Data

Log management has long been indispensable for maintaining system health, diagnosing faults, and ensuring operational integrity in distributed software systems [7, 8]. Proactive monitoring and analysis of log data are fundamental to effective system oversight. However, the escalating complexity and heterogeneity characteristic of modern systems introduce significant challenges in structuring, parsing, and reasoning over the vast quantities of generated log data [11, 18].

Selecting and implementing an appropriate log management strategy remains a non-trivial task [27, 31]. The sheer volume, high velocity, and diverse variety of logs—particularly within highly regulated and critical domains such as finance, healthcare, and telecommunications—mandate the adoption of scalable and semantically-aware solutions. A critical limitation in current approaches lies in the absence of formal, conceptual modelling frameworks specifically tailored for log data. This chapter articulates these concerns by presenting concrete real-world log examples and identifying the key conceptual modelling challenges that arise from the inherent characteristics and inconsistencies of contemporary log data.

Example 2.3.1: Heterogeneous Logs in a Payment Workflow

We begin by analysing a series of log entries from a hypothetical, yet realistic, financial application executed on April 25, 2025. The system processes card payments and invokes multiple loosely coupled services, including gateway, fraud detection, currency conversion, and orchestration components.

Listing 1 reports the complete trace for a single transaction (`txn_id = 9283746`) as recorded by different services in the execution pipeline.

Listing 1: Heterogeneous log trace for transaction 9283746

```
2025-04-25 14:03:10,145|INFO|gateway|GatewayService.java|112|
Received payment initiation: {txn_id: 9283746, amount: 200.00, currency:
→  "USD"}

2025-04-25 14:03:11,022|ERROR|fraud-check|FraudService.java|340|
Transaction flagged: Rule 'high_frequency' triggered for user ID 49321

2025-04-25 14:03:12,000|DEBUG|conversion|FxConverter.java|205|
Rate lookup failed for USD->EUR on 2025-04-25T14:03Z

2025-04-25 14:03:12,187|INFO|orchestrator|Orchestrator.java|909|
Failure checkpoint reached; rerouting to fallback processor

2025-04-25 14:03:12,330|WARNING|gateway|GatewayService.java|119|
Final response delayed; retry queue size exceeded threshold

2025-04-25 14:03:13,514|INFO|***|UnknownModule|???|
TXN_STATUS_UPDATE: txn=9283746 status="HOLD" reason="FAILSAFE_TRIGGER"
```

This trace exposes several systemic issues that are common in distributed, log-centric architectures:

- **Structural inconsistency:** Although all entries include timestamps and source metadata, the internal structure of log messages varies widely. Some entries embed JSON-like payloads, others rely on free text, and certain fields (e.g., module name or source file) are missing or obfuscated.
- **Semantic ambiguity of severity levels:** An `ERROR` in the fraud service is followed by an `INFO` log indicating a failure checkpoint, and later by

a **WARNING** unrelated to the original cause. This weakens the meaning of severity levels and complicates filtering, alerting, and root-cause analysis.

- **Fragmented execution context:** While all log lines refer to the same transaction identifier, no explicit notion of process state or execution step is shared across services. As a result, it is unclear which phase of the payment workflow has failed or been suspended.
- **Undefined status transitions:** The final entry reports a **HOLD** status, but provides no formal definition of this state, its position in the transaction lifecycle, or its implications for downstream systems and customer-facing behaviour.

From a business perspective, a payment transaction is not a single operation but an orchestration of interdependent activities, including validation, fraud assessment, currency conversion, routing, and audit logging. Such workflows are typically modelled using process-aware formalisms (e.g., BPMN). However, when emitted logs are not explicitly linked to these conceptual process models, they remain isolated technical artifacts that cannot be reliably interpreted, validated, or analysed at the process level.

This example illustrates that, in the absence of shared semantics and explicit execution context, log data alone is insufficient to support robust observability, compliance analysis, or predictive reasoning in complex financial systems.

Example 2.3.2: Asynchronous Reconciliation and Notification Events

To extend the discussion on temporal misalignment and semantic ambiguity, we consider a log trace illustrating asynchronous processing across loosely coupled system components. The transaction is processed on April 30, 2025 and involves

delayed reconciliation and messaging operations. Listing 2 reports the sequence of log events generated by different services within a payment processing system. All entries refer to the same transaction identifier TXN1058329, yet they are produced by independent components with distinct execution timelines.

Listing 2: Asynchronous log trace for transaction TXN1058329

```
2025-04-30 18:45:03,110 | INFO      | payment-core      |  
→ PaymentProcessor.java:512 |  
Payment submitted: txn_id=TXN1058329, method=CARD, amount=49.99 USD  
  
2025-04-30 18:45:03,711 | INFO      | audit-service     | AuditTrail.java:199 |  
Created audit log entry for transaction: TXN1058329  
  
2025-04-30 18:45:08,472 | DEBUG     | message-broker    | KafkaProducer.java:77 |  
Published event to topic 'payment.status':  
{\"txn_id\":\"TXN1058329\", \"status\":\"POSTED\"}  
  
2025-04-30 18:46:21,650 | INFO      | reconciliation     | ReconWorker.java:288 |  
No matching ledger entry found for txn_id=TXN1058329; will retry in batch job  
  
2025-04-30 18:47:00,142 | WARNING   | notification      | SmsSender.java:89 |  
Delay detected in message queue for txn_id=TXN1058329 (age: 112s)  
  
2025-04-30 18:48:02,301 | INFO      | reconciliation     | ReconWorker.java:312 |  
Transaction TXN1058329 reconciled in ledger batch run #49328
```

This example highlights several structural issues commonly observed in distributed, event-driven architectures:

- *Temporal decoupling* Reconciliation, notification, and event publication occur asynchronously. Events referring to the same transaction are sepa-

rated by minutes, complicating correlation and real-time interpretation.

- *Lack of a status ontology* Status labels such as `POSTED` and `RECONCILED` appear in different logs, yet no explicit semantic relationships, lifecycle ordering, or state transitions are defined between them.
- *Traceability gaps* Although the transaction identifier is consistently propagated, the logs are not linked to an explicit process model or execution trace. As a result, it is unclear whether the observed sequence represents normal progression, delay, or exception handling.
- *Operational uncertainty* Warning messages about delayed notifications are not formally linked to service-level objectives, retry policies, or escalation rules, limiting their usefulness for automated monitoring and control.

Overall, the example illustrates how the absence of a shared conceptual and temporal model hinders observability, auditing, and analytical reasoning. These limitations motivate the need for explicit temporal semantics and window-based structuring, as later formalized in the Prediction-Oriented Dimensional Fact Model.

Example 2.3.3: Structured JSON Logs and Process Semantics

To provide a final, complementary illustration, we examine a structured log trace for a transaction processed on May 2, 2025. This example shows that even syntactically well-formed logs (e.g., JSON) may still exhibit significant conceptual and semantic ambiguity.

Listing 3 reports a representative sequence of log events produced by multiple services involved in the processing of transaction `TXN982301`. Although each entry is machine-readable and richly annotated, the trace does not explicitly

encode the underlying business process semantics.

Listing 3: Structured JSON log trace for transaction TXN982301

```
{
  "timestamp": "2025-05-02T14:22:07.456Z",
  "level": "INFO",
  "service": "api-gateway",
  "component": "PaymentInitiator",
  "txn_id": "TXN982301",
  "user_id": "USR34912",
  "message": "Payment request received",
  "payload": {
    "amount": 120.50,
    "currency": "USD",
    "payment_method": "CARD",
    "channel": "mobile_app"
  }
}

{
  "timestamp": "2025-05-02T14:22:07.892Z",
  "level": "INFO",
  "service": "fraud-engine",
  "component": "FraudScorer",
  "txn_id": "TXN982301",
  "risk_score": 0.82,
  "decision": "REVIEW",
  "rule_triggered": "geo_velocity_high",
  "message": "Transaction flagged for manual review"
}

{
  "timestamp": "2025-05-02T14:22:08.147Z",
  "level": "INFO",
```

```
"service": "fx-service",
"component": "ExchangeRateResolver",
"txn_id": "TXN982301",
"from_currency": "USD",
"to_currency": "EUR",
"rate": 0.9112,
"message": "FX rate applied"
}
{
  "timestamp": "2025-05-02T14:22:09.103Z",
  "level": "INFO",
  "service": "payment-orchestrator",
  "component": "RoutingEngine",
  "txn_id": "TXN982301",
  "route": "provider_b",
  "message": "Payment routed to processor"
}
{
  "timestamp": "2025-05-02T14:22:10.405Z",
  "level": "INFO",
  "service": "payment-processor",
  "component": "ProviderBAdapter",
  "txn_id": "TXN982301",
  "provider_response": {
    "status": "AUTHORIZED",
    "auth_code": "94H28D",
    "processor_txn_id": "PB-TXN-7710"
  },
  "message": "Authorization successful"
}
{
  "timestamp": "2025-05-02T14:22:15.003Z",
```

```
"level": "INFO",
"service": "settlement-engine",
"component": "BatchScheduler",
"txn_id": "TXN982301",
"batch_id": "SETTLE-APR-002",
"message": "Transaction included in settlement batch"
}
{
  "timestamp": "2025-05-02T14:22:18.712Z",
  "level": "INFO",
  "service": "notification-service",
  "component": "SmsSender",
  "txn_id": "TXN982301",
  "recipient": "+15556667777",
  "template": "payment_success",
  "message": "User notified of successful payment"
}
```

Despite its structured format, the trace raises fundamental modelling questions that cannot be resolved from the logs alone:

- What constitutes a *complete* transaction from a business or accounting perspective?
- How are events emitted by different services semantically related to a single business process instance?
- Which stages in the transaction lifecycle are mandatory, conditional, or optional?
- How are failures, retries, compensations, or alternative paths represented or inferred?

In the absence of an explicit process model or shared semantic framework, the reconstruction, validation, and analysis of business behaviour from log data alone becomes inherently ambiguous. This example illustrates that syntactic structure is insufficient to guarantee semantic clarity, reinforcing the need for explicit temporal and conceptual modelling, as later addressed by the Prediction-Oriented Dimensional Fact Model.

2.3.2 Problem Statement

A transactional event, such as a payment, is not an atomic action but the result of multiple interdependent processes across various system components. These include steps like request receipt, fraud scoring, currency conversion, routing, and settlement or notification. However, without a formal process model (e.g., BPMN), the relationships and flow between these activities are implicit, making them difficult to track and understand. This issue is exacerbated when logs are analysed retroactively by teams unfamiliar with the system’s internal design.

Several key challenges arise from this lack of process modelling and inconsistency in log data:

1. *Inconsistent Naming of Entities:* Variations in entity naming (e.g., `txn_id` vs. `transaction_id`) across services complicate data integration and analysis.
2. *Lack of Canonical Event Typing:* Informal, free-text event descriptions (e.g., “Authorization successful”) hinder reusable event definitions and make it hard to compare event sequences.
3. *Implicit Process Semantics:* Log entries lack explicit references to the underlying process, making it difficult to map them to structured models like BPMN.

<i>Challenge</i>	<i>Description</i>	<i>Mitigation Strategy</i>
<i>Inconsistent Naming</i>	Variations in entity naming across services	Lexical normalization, field mapping dictionaries, standardized naming conventions.
<i>Lack of Canonical Event Typing</i>	Free-text event descriptions complicate comparisons and reuse	Define formal event taxonomies and map log messages to typed events.
<i>Implicit Process Semantics</i>	Logs lack explicit references to the process model	Map log events to steps within structured process models; introduce process instance IDs.
<i>Missing Contextual Information</i>	Logs may lack crucial contextual data for understanding events	Enforce enriched logging standards with mandatory context fields.
<i>Misuse of Severity/Status Indicators</i>	Inconsistent use of severity levels obstructs monitoring	Adopt formal vocabularies for domain-level statuses; enforce consistent severity usage.
<i>Missing Process Identity</i>	Lack of broader context identifiers limits tracking across business processes	Introduce explicit case or process identifiers at the point of log generation.

Table 2.1: Summary of Conceptual Modelling Challenges and Mitigation Strategies

4. *Missing Contextual Information:* Some logs omit crucial context (e.g., service failures without currency details), affecting traceability and event understanding.
5. *Misuse of Severity/Status Indicators:* Inconsistent use of severity levels (e.g., logging critical errors as `INFO`) obstructs accurate monitoring and classification.
6. *Missing Process Identity:* The absence of a broader context identifier (e.g., `case_id`) limits the ability to track related activities within a larger business process.

These challenges significantly impede reliable process analysis, trace reconstruction, and system observability. To overcome these limitations, logs should be closely

tied to standardized conceptual models, such as structured data models (e.g., JSON schema), UML diagrams, and formal process definitions like BPMN. This integration ensures that logs are semantically rich, enabling effective monitoring, auditing, and compliance, and allowing accurate traceability of business processes.

2.4 The Proposed Modelling Approach

The UML provides a wide spectrum of modelling possibilities [29]. As our objective is to support developers and super users in understanding and leveraging log data, our approach emphasizes the more formal end of UML [30].

To support this, we utilize standard UML extension mechanisms:

- **Stereotypes** to introduce new modelling elements tailored to logs;
- **Tagged values** to add specific metadata or attributes;
- **Constraints** to define precise semantics and validation rules.

We rely primarily on the stereotype extension by deriving new block types (classes) from standard UML elements. These stereotypes are applied to classify and annotate classes for log-specific semantics. We also define enumeration types, specialized attributes, and enriched associations to align the UML schema with JSON-structured log data.

2.4.1 Metamodel for Log Files Modelling

The UML metamodel (M2) is designed using UML 2.5 class diagram elements. Figure 2.1 illustrates the metamodel that supports conceptual modelling of structured logs. It defines how log entities, attributes, and their relationships can be formally described.

To accommodate typical JSON logs, the metamodel:

- Supports nested structures using associations and multiplicities;
- Encodes log fields as class **Attributes**;
- Differentiates log components using **Stereotypes**;

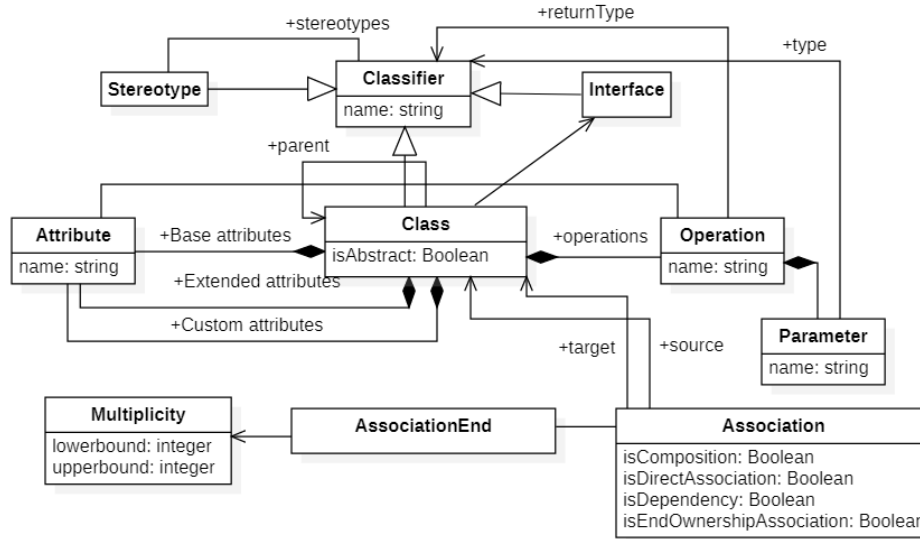


Figure 2.1: The simplified metamodel based on the UML class diagram

- Enables extensibility with tagged values and enumerations;
- Simplifies operations and interfaces in favor of data-centric modelling.

Key Elements of the Metamodel:

<i>Element</i>	<i>Description</i>
<i>Class</i>	Represents log entities (e.g., <code>LogEntry</code> , <code>UserSession</code>); may be abstract.
<i>Attribute</i>	Defines individual fields within a log entry (e.g., <code>timestamp</code> , <code>severity</code>).
<i>Stereotype</i>	Extends UML classes with domain-specific semantics (e.g., <code><<Event>></code> , <code><<Error>></code>).
<i>Association</i>	Represents structural relationships (e.g., one session may have many log entries).
<i>Multiplicity</i>	Defines cardinality, useful for arrays or optional elements in JSON.
<i>Operation</i>	Included for completeness but generally unused in log-centric models.

Table 2.2: Key elements of the metamodel and their roles in log file modelling

To demonstrate the application of our metamodel, Table 2.3 presents a simplified JSON log entry and its corresponding UML-based conceptual model.

<i>Sample JSON Log Entry</i>	<i>Mapped UML Instance (Conceptual)</i>
<pre>{ "timestamp": "2023-06-01T12:00:00Z", "level": "ERROR", "message": "Payment failed", "transactionId": "TX12345", "user": { "id": "U001", "role": "customer" } }</pre>	<pre>Class <<LogEntry>> - timestamp: string - level: SeverityEnum - message: string - transactionId: string Assoc: LogEntry -> User [1] Class <<User>> - id: string - role: string</pre>

Table 2.3: Mapping a JSON log structure to UML model instances

This illustrates how nested JSON elements and semantics are represented using UML classes, associations, and stereotypes-providing a clear foundation for automated analysis, validation, and model-driven engineering.

2.4.2 Conceptual Model and Key Features Definition

The proposed UML-based modelling approach leverages class diagrams extended with suitable stereotypes [29]. Class diagrams are particularly well-suited for representing the hierarchical and nested structures typically found in log records. Logs are primarily composed of attributes and *field sets*, where each field set groups related attributes under a common functional theme.

In our conceptual model, field set names and their associated attributes are derived from the Elastic Common Schema (ECS) specification. A representative structure of a log record is illustrated in Fig. 2.2.

To capture nesting relationships among field sets, we utilize composition associations within UML. Field sets are modelled as class-like elements and may themselves contain other nested field sets, enabling recursive reuse. Each field set is visually

structured into three compartments to distinguish between *core*, *extended*, and *custom* fields. For nested field sets within the same scope, we adopt an ad-hoc graphical notation to clearly indicate their local hierarchy.

In addition to structural nesting, our conceptual model incorporates several other important features:

- **Attribute Typing:** Explicit support for defining attribute data types, including primitives, enumerations, and arrays.
- **Attribute Associations:** Logical relationships between attributes (e.g., value dependencies or constraints).
- **Enumeration Types:** To constrain attributes to controlled vocabularies.
- **ECS Metadata:** Used for event categorization and semantic enrichment.
- **Custom Extensions:** Support for application- or domain-specific field definitions.

The model is based on UML class diagram constructs but includes extensions and enhancements specifically tailored for log data representation. These extensions support a wide range of practical use cases, especially in complex industrial scenarios.

While ECS provides a structured vocabulary for fields and categorizes them into *core* and *extended* levels, it leaves the definition of *custom* fields to the discretion of the user or system integrator [34]. Our approach formally incorporates custom fields and even custom field sets as first-class citizens in the model. These are defined freely by developers or domain experts to accommodate specific event logging needs outside ECS’s standard scope.

This flexibility is intentional: it ensures that the model remains adaptable and extensible, particularly in scenarios where ECS coverage is insufficient. However, it is

worth noting that introducing custom fields entails a slight risk of name collision with future versions of ECS-especially if logs are consumed by tools with rigid schemas. Nonetheless, this risk can be mitigated through careful naming conventions and by employing schema validation strategies.

When integrated into a custom or schema-flexible log management infrastructure, the proposed conceptual model enables structured reasoning, validation, and higher-level abstractions for log analysis. This results in improved traceability, monitoring, and compliance, offering significant advantages over ad hoc or unstructured logging practices.

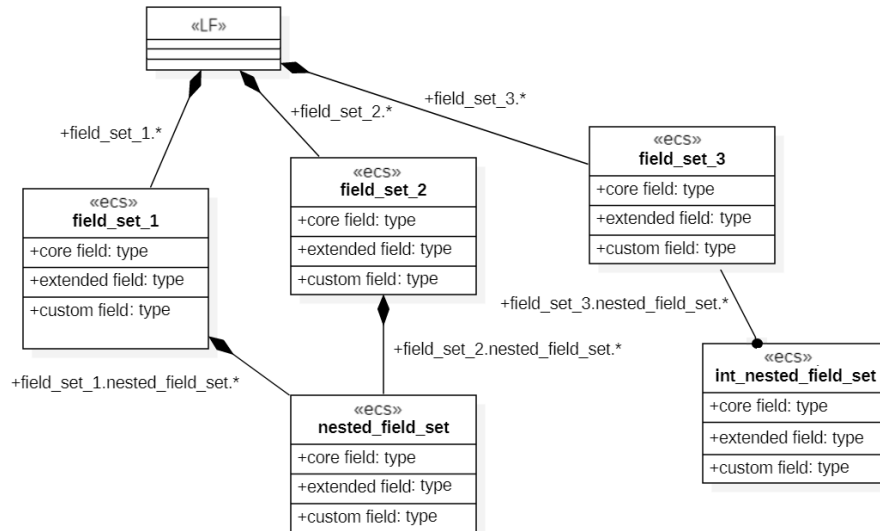


Figure 2.2: Representation of a log record as an extended UML diagram: structure of field sets boxes and their nesting

Defining Class and Class Stereotype

To represent the field sets, we extend a class diagram box, by representing them as a four-box structure. Features of the redefined class are the attributes only, distinguished through the field sets. An extended class for the log files modelling can also remind a data type is a classifier, as attributes are identified by their values/datatype (Fig. 2.2).

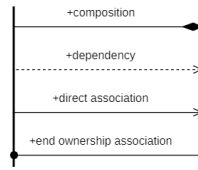


Figure 2.3: Redefined notations, used in the proposed extended UML model

- The first sub-box is the name of the field set and as the stereotype, we indicate belonging to either the standard ECS group of field sets, the so-called meta-level field sets (such as categorization fields) or the extended schema defined by the user.
- The second, third and fourth sub-boxes contain the core, extended and custom fields and their data types, respectively. For assuredly defined fields we use "+", for the ones currently considered as beta we use "-" (UML reference - visibility).

In our model, we define the following stereotypes:

- LR - is a class that defines the core of the log record and contains all the other nested classes (field sets). It could be followed by the name of the class (for the representation or due to the JSON structure).
- ecs - is a class that specifies the field set, when all the fields are adopted from ECS.
- custom - is a class that has custom fields or the whole field set is custom.
- ecscf - is a class that includes categorization fields with «enumeration».

Special Use Cases of Field Sets

For more convenience and simplification we propose to use the star symbol "*" for the core and extended fields to show that all fields predefined to the particular set are used.

The box with the field set which is not used could be omitted. Omitting is possible only if it is explicit what types of field sets are shown on the diagram, whether they are base, extended or custom.

Nesting

Nesting in the model is defined on different levels because it follows to meet the particular needs of the representation. In the proposed approach, we distinguish two kinds of nesting. While the first one is explicitly specified through different field sets of ECS, the second one is a weaker nesting not corresponding to explicitly nested field sets.

As for *Nested Field Sets Under Root Level*, such field sets are not expected to be used directly at the root of the log record structure and are required to be exclusively nested. For the graphical representation of dependencies was chosen "line and black diamond" association (UML composition) to describe a relationship between a root and nested parts.

As for *Internal Nesting*, the graphical representation "line and black dot", end ownership association, is used to depict the internal nesting between field set parts. By default, the dot notation is used to represent the structure complexity and dependencies between different fields.

Ownership of the association ends with a small filled circle, where the line meets the associated classifier. It could be interpreted as showing that the model exclusively includes the field sets represented by the classifier touched by the dot. These field sets cannot be nested in the others.

The box structure of log records and the nesting association are shown in Fig. 2.2.

Fields in the Root of Log Files

Some fields could define a unique group set, however, not intended to be nested but written directly at the root of the log record. Such an example is a base field set (Fig. 2.4). As we will see in the following case studies of bank and network domains

(Fig. 2.7 and Fig. 2.9 relatively).

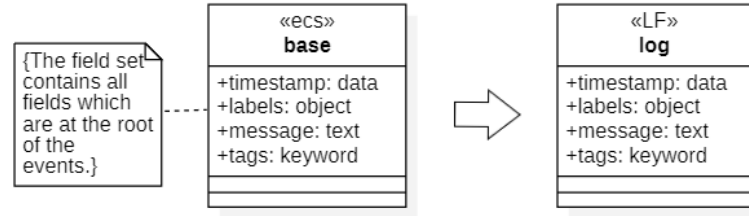


Figure 2.4: The illustration of the base field set in the root of a log record

These fields are common across all types of events, "timestamp" is the required field for all events as well as "ecs.version" (Fig. 2.7).

Predefined Field Values

An example of a field with predefined values is the categorization field set. For the graphical representation was chosen enumeration advanced model element (in UML class diagrams) to represent already developed under the current model or user-defined (custom) data types.

This classification can vary, depending on the type of the particular enumeration: constraint or advisory. This means that values can be strictly chosen from the predefined set or there is a recommended set yet it is allowed to use values free of choice concerning set up data type.

We do recommend to use as names for enumeration capital first letters of the field sets names for the fixed values (example: "CF1", "CF2" as will be shown following in Fig. 2.7 and Fig. 2.9) and for the recommended ones "R". Several enumerations related to the same field set must be listed according to recommendations above plus a number in ascending order. Values that are not in the list of allowed values should not be used.

Dependency for Representing Collected Values

In particular cases, redundancy of the schema is allowed, i.e. collecting several field values of the same data type in one collecting field. For instance, the "related.ip"

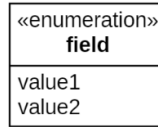


Figure 2.5: Enumeration for predefined field values

field captures all the present in the event IPs in a single array (Fig. 2.9).

For the graphical representation, the arrow line notation (UML dependency) was chosen. Together with the notation, the name of the collected values must be specified.

Dependency is a directed relationship, that shows that some UML element or a set of elements depends on other model elements. In our model, the notation represents the collecting of data from different sources (same data type) in one array. It is a so-called "supplier-client relationship"; a supplier provides, and a client is dependent on its elements. Modification of the value in a certain field changes the values in the collected value field.

The field of the collected values facilitates the search when processing on a chosen platform, for example, ELK. Since these values can appear in many fields, a simple search can be performed, regardless of which field they appear in, with a single query [34].

Association Between Fields

When an event contains details about the sending and receiving hosts, the baseline for capturing these values will be the source and destination fields.

Some events may also indicate each host's role in the exchange: client or server. When this information is available, the client and server fields should be used in addition to the source and destination fields. The fields and values mapped under source/destination should be copied under client/server [25].

The proposed representation for the case of an association, when one field of the

particular set must be populated exclusively in one of two other fields, is shown in Fig. 2.9.

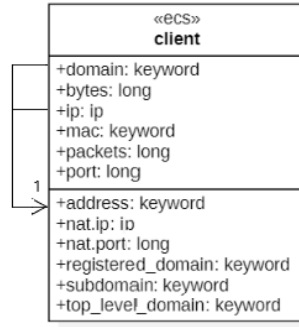


Figure 2.6: Association between fields of the client field set

XOR Constraint

XOR constraint applies to associations or even two values. The constraint is shown as a dashed line between the elements labelled by the constraint field, where field set names or values could be specified (Fig. 2.8).

Categorization Fields

Four categorization fields are defined for this purpose, each of which falls under the "event.*" field set. Values that are not in the list of allowed values should not be used. When a specific event does not fit into any of the defined allowed categorization values, the field should be left empty. The structure and features of the representation are explicitly shown in Fig. 2.7.

At a high level, the schema provides fields to classify events in two different ways: "where it is from" (e.g., "event.module", "event.dataset", "agent.type", "observer.type", etc.), and "what it is." The categorization fields hold the "what it is" information, independent of the source of the events.

Dependency for Duplicated Values

Most commonly the proposed notation is used for the network events to capture the details of one device communicating with another. The initiator is referred to

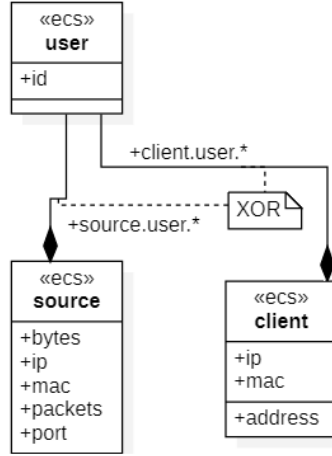


Figure 2.8: The constraint XOR between two field sets

2.4.3 Implementation on Industrial Case Studies

In considering real-world applications for the proposed conceptual modelling approach, we followed a top-down strategy, to design log record structures at a conceptual level, also to support the implementation of possibly required adjustments or changes, and possibly allow different formats such as JSON, XML, and so on. Fig. 2.7 shows a UML-based diagram for the batch log of the custom application in the banking domain created with the implementation of the proposed conceptual model.

We then considered a bottom-up approach, on existing log files, to provide a high-level description of the log file structure, and also to support a clear understanding of the different fields when designing suitable analysis on such data.

Fig. 2.9 shows the conceptual schema from the network logs. In this case, it is standardized, and there are no problems with format and parsing or subsequent analysis, despite that logs are obtained from very different sources (firewalls, servers, databases, networks, applications, and so on). A clear comprehension of its structure, in any case, would help in building and sharing related data analyses.

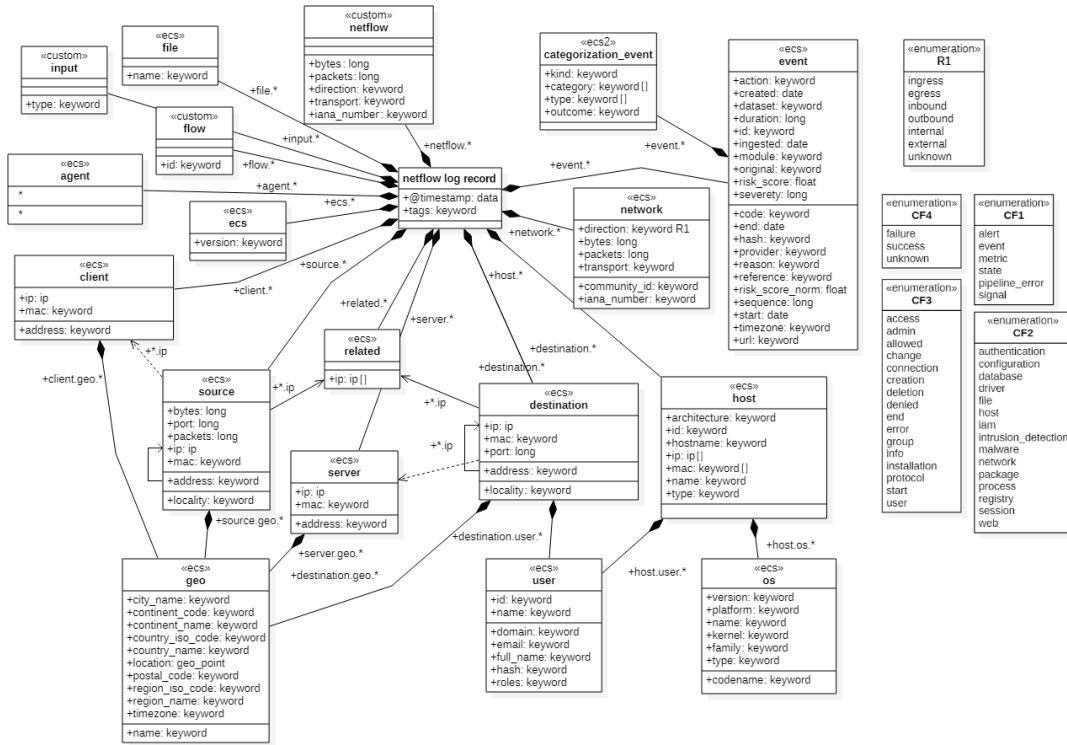


Figure 2.9: Log files of the network event domain: graphical representation through the proposed modelling approach

Perspectives of the Diagram

The choice of perspective depends on the stage of the modelling process and the type of representation. We propose interpretation from two perspectives:

- conceptual, that represents the concepts in the domain (values but not their type), as in the Fig. 2.2 and Fig. 2.7;
- specification, that focuses on the data type.

While modelling a template for the log files with custom values the specification perspective is required. Yet when the types of values are explicitly predefined (through ECS, for example), they could be committed to the schema. Modelling could start with the conceptual perspective and then evolve into the specification.

2.5 Tool Support for Conceptual Modelling of Log Data

The conceptual modelling approach developed in this chapter is accompanied by a dedicated StarUML extension that operationalises the principal constructs of the proposed modelling language. The extension provides executable support for the creation, transformation, inspection, and validation of conceptual models of structured and semi-structured log data. Its purpose is twofold. First, it enables the proposed notation and UML–JSON mappings to be applied in practical modelling workflows. Second, it provides implementation evidence that the metamodel, stereotypes, structural constraints, and transformation rules defined in the preceding sections can be realised within an existing modelling environment.

The software artefact should therefore be understood as an implementation of the conceptual contribution, rather than as a separate modelling contribution. The semantics enforced by the extension are derived from the metamodel and mapping rules defined in this chapter; they are not introduced independently at the software level. The source code, installation information, development provenance, and repository address are documented in Appendix A.

2.5.1 Design and Functional Scope of the Modelling Tool

StarUML was selected as the host environment because it provides an extensible architecture for UML-based modelling, programmatic access to model elements, and native support for class diagrams. These properties make it possible to introduce log-specific modelling operations without developing a standalone diagramming environment. The extension integrates its commands into the StarUML interface and operates directly on the classes, attributes, stereotypes, associations, multiplicities,

and constraints used to represent the conceptual structure of log records.

The extension supports the two modelling directions considered throughout this chapter. In a *top-down* workflow, a modeller creates a UML representation of the intended log structure and uses the extension to generate a corresponding JSON template. This workflow supports the specification of logging structures before their implementation. In a *bottom-up* workflow, existing JSON or JSONL log data are analysed and transformed into a UML class diagram that provides a higher-level representation of fields, nested structures, and recurring components. The resulting model can subsequently be inspected, refined, and validated.

The implementation also supports validation against the Elastic Common Schema (ECS). This functionality checks whether model elements are consistent with the field names, data types, and nesting structures represented in the locally available ECS definitions. Since ECS evolves over time, the extension includes a synchronisation mechanism for retrieving updated field-set definitions and incorporating them into the validation resources used by the tool. Diagram export through PlantUML provides an additional representation suitable for documentation and for the production of scalable figures.

These operations provide practical support for the central methodological claim of this chapter: an explicit conceptual representation can mediate between technology-independent log semantics and concrete structured-log representations. The extension demonstrates the technical feasibility of this mediation and enables the examples presented in this chapter to be reproduced and inspected. It should nevertheless be regarded as a research prototype rather than as a complete production-grade log-governance platform.

Figure 2.10 shows the integration of the extension within the StarUML interface. The menu exposes the principal transformation, validation, schema-update, and export operations without requiring users to manipulate the internal representation

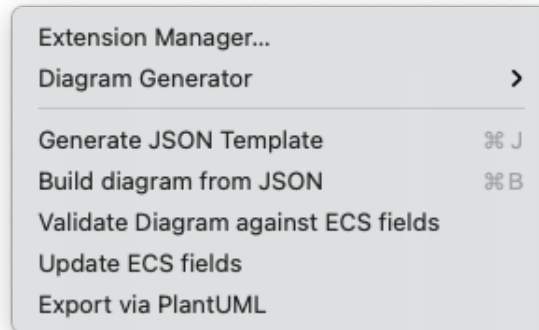


Figure 2.10: StarUML menu commands provided by the conceptual log-modelling extension

of the model directly. The individual capabilities are described below.

extension Capabilities

1. Automatic JSON Template Generation from UML Diagrams

The extension can traverse a UML class diagram-starting from a designated root element-and automatically generate a JSON schema template. This is achieved through a Breadth-First Search (BFS)-based graph traversal. During traversal, the structural representation of the model is incrementally converted into a JSON object, which is subsequently saved to the file system.

2. Diagram Generation from JSON and JSONL Log Files

Conversely, the extension can construct UML class diagrams by parsing various types of JSON input:

- **Flat JSON files:** containing a single JSON object.
- **JSON arrays:** where the extension merges duplicated structures and optimises the diagram layout by detecting reusable components.

- **JSONL (JSON Lines) files:** each line contains a discrete JSON object. The extension supports parsing JSON objects spread across multiple lines, with line grouping customisable via extension settings.

A recursive, child-first search algorithm is used to identify fields and nesting relationships while parsing the input data.

3. Schema Validation Against the ECS Specification

To ensure model compliance with ECS, the extension enables structural validation of diagrams. This includes checking that field names, data types, and field nesting adhere to the ECS field sets and conventions. A validation operation can only proceed when a root element is selected in the model.

4. ECS Field Set Synchronisation from the Official GitHub Repository

Given that the ECS evolves regularly, the extension allows the user to update its internal schema repository directly from the official ECS GitHub repository. ECS definitions are fetched as YAML files and internally converted into JSON for integration within the extension. This guarantees up-to-date validation against the ECS definitions retrieved by the extension.

5. Diagram Export via PlantUML

For documentation and external visualisation purposes, the extension can export diagrams using PlantUML. Output formats include PNG and SVG. While PNG exports may be limited by resolution constraints, SVG exports provide scalable, high-resolution representations ideal for integration into academic publications. This feature also circumvents some limitations of the native StarUML rendering engine, particularly with respect to rendering operations without parentheses.

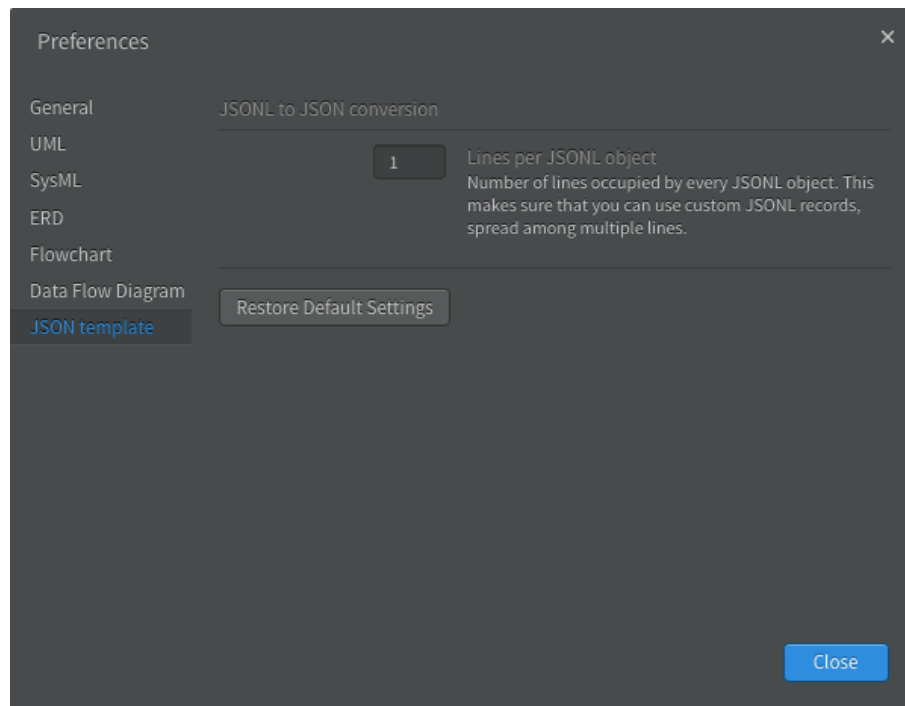


Figure 2.11: extension settings interface, including configuration for parsing JSONL files

Figure 2.11 shows the extension settings window. Here, users can configure parsing options, such as the expected number of lines per JSON object in JSONL files, enabling compatibility with diverse log generation tools.

Once a JSON log file is imported, the extension constructs a graph-based intermediate representation. This graph is then optimised by removing redundancies and merging structurally similar field sets before being rendered as a UML class diagram, as illustrated in Figure 2.12.

The graph is internally constructed using StarUML APIs and serves as the foundation for subsequent JSON template generation, validation, and visualisation. Successful execution is confirmed via toast notifications, as shown in Figure 2.13.

structure aligns with the official ECS specification.

Conversely, Figure 2.16 shows an example where validation fails due to structural or schema-related issues in the diagram.

Additional Features

- **PlantUML Export in SVG and PNG Formats:** The extension supports export to both formats. SVG is recommended for larger diagrams due to resolution constraints in PNG exports.
- **Real-Time ECS Field Updates:** ECS metadata is retrieved dynamically, allowing for immediate access to the latest field definitions and improved model validation.
- **Empty Diagram Handling:** The extension includes error handling for PlantUML export when the diagram is empty, notifying the user accordingly.

This modelling tool, by combining visual abstraction, structured schema generation, and integration with ECS, forms a key enabler for advanced, model-driven log analysis. It supports both forward (diagram-to-JSON) and reverse (JSON-to-diagram) engineering, validation workflows, and export capabilities-all of which contribute to establishing a rigorous conceptual framework for log data representation in complex software systems.

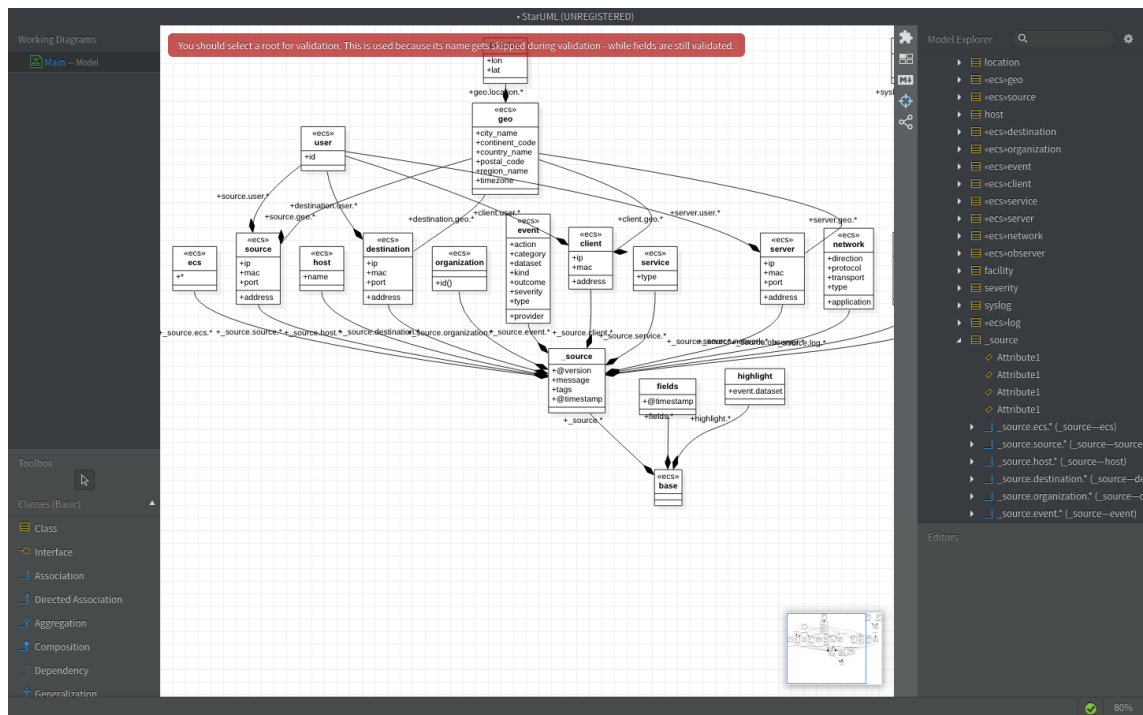


Figure 2.14: Validation requires selection of a root element in the UML model

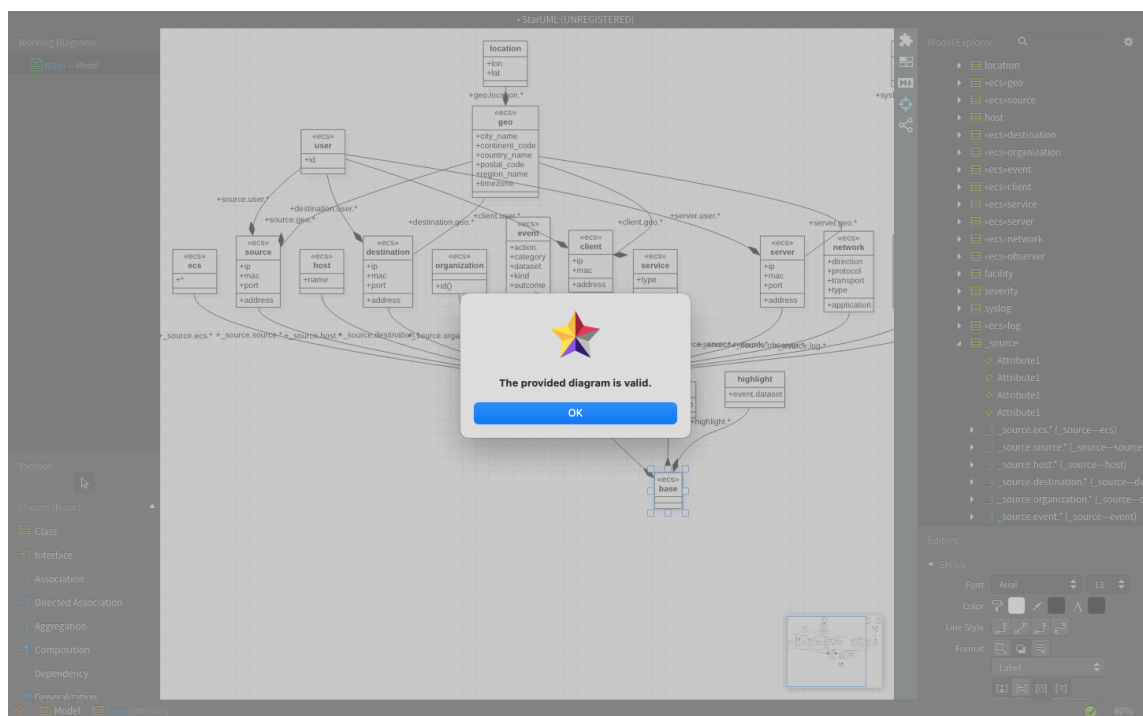


Figure 2.15: Validation completed successfully against the ECS specification

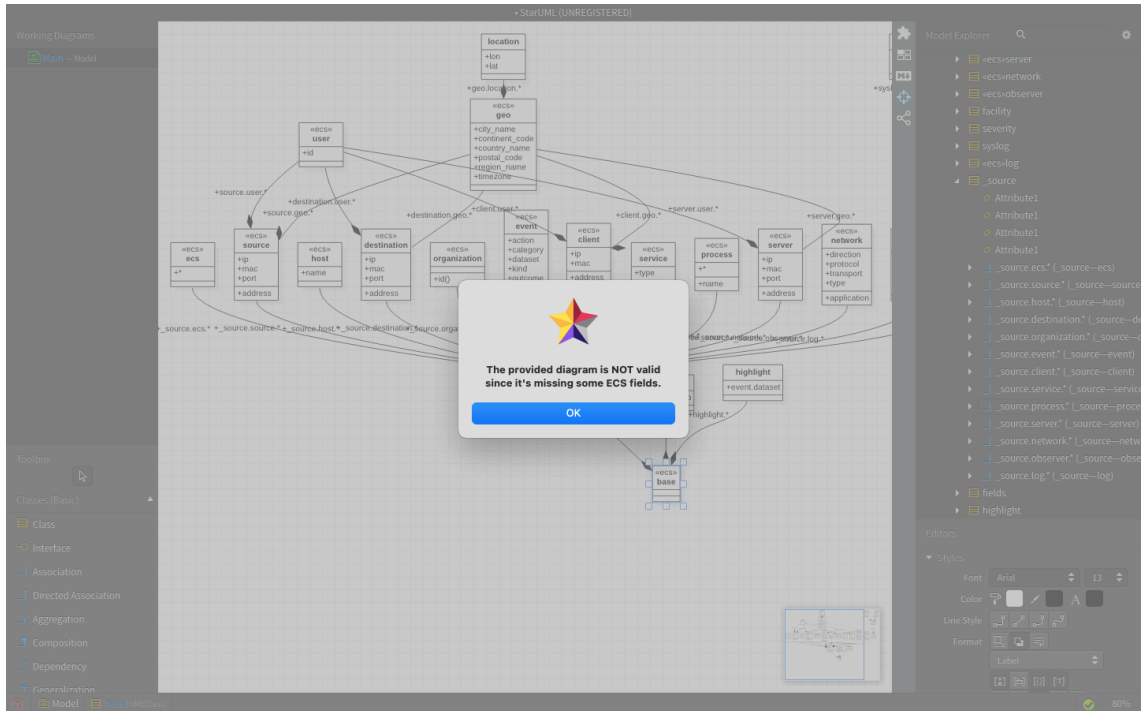


Figure 2.16: Validation failed due to schema inconsistencies or missing root element

2.6 Conclusion

In this chapter, we introduced a novel, model-driven approach for the conceptual modelling of log files, with the objective of improving the design, validation, and interpretability of log data across a wide range of software systems. The proposed methodology directly addresses the absence of modelling support in current log management practices, where the design and structure of logs are often ad hoc and error-prone.

Through the developed extension for StarUML, we have demonstrated how this modelling approach can be operationalised into a practical and extensible tool. The extension enables both forward and reverse engineering of log structures-facilitating the generation of JSON templates from UML diagrams and the automatic reconstruction of UML models from structured JSON or JSONL log files. This bidirectional capability ensures not only the correctness of the log design but also stream-

lines log schema maintenance and documentation.

One of the key insights uncovered is the critical role of early-stage modelling. Our findings underscore that errors or omissions during the modelling phase are among the most difficult to detect and correct later in the system lifecycle. The conceptual modelling approach mitigates this risk by introducing formality and transparency early in the design process, thus enabling rigorous validation of log structures against evolving standards such as the Elastic Common Schema (ECS).

Furthermore, we showed that the proposed approach improves both human and machine interpretability of logs. By abstracting log structures into UML-based representations, we provide a language-agnostic and domain-independent method that is accessible to stakeholders with diverse technical backgrounds—including software developers, data analysts, system architects, and non-technical domain experts. This interdisciplinary accessibility enhances communication and collaboration throughout the software engineering and operational lifecycle.

The tool’s effectiveness and adaptability were validated through illustrative examples across distinct domains, including financial systems and networking environments. These case studies confirmed the versatility of the approach in modelling heterogeneous log formats and revealed its potential as a foundational step toward more robust log analysis pipelines, automated monitoring, and compliance auditing.

In summary, the contributions of this chapter include:

- The formulation of a conceptual modelling methodology for log data.
- The implementation of a StarUML extension to support automated log model generation and validation.
- Support for ECS compliance verification and continuous schema updates.
- Practical demonstrations showcasing domain-agnostic applicability and bidi-

rectional transformation between JSON and UML representations.

This work lays the groundwork for further exploration of automated reasoning over log models, integration with logging frameworks and observability platforms, and the formalisation of log data lifecycle management in complex software ecosystems.

Chapter 3

A Prediction-Oriented Framework for Multidimensional Data Analysis

3.1 Introduction

Organizations today possess vast quantities of data from diverse sources, including transactions, log files, and financial records [11, 57, 128]. Data warehousing (DW) and multidimensional data models have become crucial for transforming this data into insights for decision-making [38, 39, 89]. These models excel at representing business data for historical analysis and reporting through On-Line Analytical Processing (OLAP). However, there's a growing need to leverage these rich multidimensional datasets not just for understanding past events, but also for **supporting the prediction of future outcomes**. Standard multidimensional modelling and OLAP operations, while powerful for descriptive analytics, often fall short in explicitly structuring data and guiding analysis for predictive tasks, particularly those involving complex temporal dependencies. Key challenges include systematically

organizing data across different time horizons relevant to prediction (e.g., observation periods for feature generation, waiting periods for event maturation, and future periods for prediction targets) and adapting analytical operations to support this predictive lifecycle.

Despite growing interest in machine learning (ML), it is not always feasible for organizations to implement full-scale ML pipelines. Practical constraints such as lack of technical expertise, limited data resources, strict regulatory environments, or internal bureaucracy can hinder ML adoption [45,51]. Even when predictive needs are recognised, companies may not have the infrastructure, tools or time required to implement machine learning models [49, 50]. As a result, organizations often resort to enhanced descriptive and diagnostic analytics, relying on techniques that are easier to implement and explain, and that align with existing workflows. For our advanced prediction-oriented analytics, we build upon the foundational concept of the Analytical Base Table (ABT), inherited from traditional machine learning workflows, which provides a unified, flat table structure integrating features and labels [42].

In fact, recent studies show that the majority of companies remain at the advanced analytics stage, without fully transitioning to predictive or prescriptive analytics using machine learning [47,48,51]. In many business contexts, such as finance, customer management, operations, or fraud detection - there is still a need to detect patterns and anticipate outcomes. This is commonly approached using standard tools like SQL, Python scripts, Excel-based dashboards, or enterprise platforms such as Power BI, SAP, and Oracle Analytics. While dimensional data modelling and traditional data analytics provide a solid foundation for exploration and summarization, they often lack built-in capabilities for managing temporal constraints or structuring prediction problems. This creates a gap between what the data infrastructure supports and what predictive analysis requires - a gap this research aims

to address.

To address these limitations, this chapter introduces the **Prediction-Oriented Framework using Three Temporal Windows (PO Framework)**. This novel conceptual modelling framework is specifically designed to enhance multidimensional data analysis for uncovering predictive dependencies. At its core, the PO Framework employs an approach centered around three distinct yet interconnected time windows: an **Observation Window (OW)** for capturing historical data and enabling predictive features, a **Waiting Window (WW)** to account for delays before an outcome can be determined, and a **Prediction Window (PW)** to define the period for which predictions are made and outcomes are observed.

The use of temporal windows to structure prediction problems has been explored in prior research [118,119,121,161]. Building upon such concepts, the **distinct novelty of the PO Framework** lies in its comprehensive integration of this specific three-window (OW, WW, PW) structure into a formal conceptual model for multidimensional data. This includes:

- Extending the traditional Dimensional Fact Model (DFM) to explicitly incorporate these temporal phases.
- Defining how data should be structured and linked across these windows, supporting attributes that evolve over different, ordered time points (drawing on a multi-temporal relational model).
- Providing a basis for introducing OLAP-style operations to support prediction-oriented queries and feature extraction.
- Facilitating the design and construction of prediction-oriented data marts optimized for identifying predictive relationships, accommodating approximation and error measures rather than purely deterministic links.

The PO Framework is designed to be both theoretically sound and practically applicable. Its ability to model and extract predictive insights is supported by a formal metamodel and demonstrated through a Python-based implementation. This implementation includes an open-source web application and shows integration with commercial analytical tools like Power BI, validated using real-world datasets [87, 88]. This allows for the efficient identification of both linear and non-linear relationships within multidimensional data cubes prepared according to the framework's principles [85].

This chapter progressively develops the Prediction-Oriented Framework from its theoretical foundations to its implementation and evaluation. Section 3.2 reviews multidimensional modelling, temporal data warehousing, prediction-oriented analytics, process mining, and predictive process monitoring, and positions the proposed framework relative to these research areas. Section 3.3 introduces the credit-risk motivating scenario. Section 3.4 develops the multidimensional modelling foundations and the fact-constellation constructs. Section 3.5 formalizes the Observation, Waiting, and Prediction Windows together with the Prediction-Oriented OLAP operators. Section 3.6 defines the Prediction-Oriented Dimensional Fact Model. Section 3.7 presents the mapping from the conceptual model to Python- and SQL-based realizations and reports the principal validation evidence. Detailed source-code listings, execution procedures, and extended testing material are provided in Appendix B. Section 3.8 summarizes the contribution of the chapter.

Section 3.5 formalizes the temporal dimension of the framework by defining the three temporal windows - Observation, Waiting, and Prediction, and by introducing Prediction-Oriented OLAP operators and window configurations. Building on these elements, Section 3.6 consolidates the proposed concepts into the formal Prediction-Oriented Dimensional Fact Model, providing both graphical and metamodel-level definitions and explicitly modelling predictive dependencies. Finally, Section 3.7

demonstrates how the framework can be operationalized through concrete implementations, illustrating both Python-based and database-centric (SQL) approaches and validating the proposed methodology through realistic case studies.

3.2 Background and Related Work

A Data Warehouse (DW) serves as a vast repository of historical and integrated data, designed to support informed decision-making by providing analysts with mechanisms for effective information extraction [38, 89]. DWs typically organize data within a multidimensional environment. In this paradigm, events of interest are represented as *facts*, which are characterized by a set of quantifiable *measures* (e.g., sales amount, units sold) [90]. These facts are contextualized by various *dimensions* (e.g., time, product, location), which define the perspectives for searching, examining, aggregating, and sorting data, essentially specifying the granularity of the fact measurements [91]. Dimensions are often structured with *hierarchies* composed of different *levels* (e.g., a time dimension might have a hierarchy Day $\rightarrow$ Month $\rightarrow$ Quarter $\rightarrow$ Year), enabling data analysis at various levels of detail and aggregation [61]. The choice of the lowest level of detail for measures, often termed the *grain*, is a critical design decision as it dictates the types of analyses that can be performed.

The multidimensional data model, with its core components like data *cubes* (a conceptual and often physical representation of facts and measures organized by dimensions), is integral to On-Line Analytical Processing (OLAP). OLAP systems are designed to provide rapid responses to complex analytical queries posed iteratively during interactive sessions [110, 142]. The inherent structure of the multidimensional model is optimized to handle such complex queries efficiently in near real-time.

3.2.1 Foundational Concepts in Multidimensional Data Modelling

Multidimensional Data Modelling (MDM) provides a powerful paradigm for representing and analysing complex datasets characterized by multiple dimensions and

attributes [99]. As organizations accumulate vast amounts of data, MDM techniques become essential for extracting meaningful insights that traditional data processing methods might struggle to deliver in a timely or effective manner, especially given the inherent complexity and high dimensionality of real-world data [85, 100, 104].

Effective MDM hinges on robust *conceptual design*. A well-defined conceptual model is a fundamental prerequisite for constructing a documented and effective data warehouse or data mart that accurately reflects user requirements and leverages the distinctive features of a multidimensional perspective [9, 10, 38, 39, 41, 54, 55, 74, 81]. Developing appropriate functional dependencies, which define the hierarchical relationships within dimensions, is particularly reliant on a sound conceptual model [10, 39, 63]. While traditional MDM is well-suited for descriptive analytics, this thesis will later explore extensions to better support prediction-oriented tasks, particularly those involving complex temporal structures [102, 118, 119, 121].

A crucial extension to MDM involves the explicit management of temporal aspects of data. In temporal multidimensional modelling, two key temporal notions are commonly distinguished: *valid time*, representing the period during which a fact is true in the real world, and *transaction time*, representing the period during which that fact is stored within the database system [141]. These concepts are foundational for supporting historical and predictive analyses, as they enable precise tracking of state changes over time and are often reflected in both conceptual and physical design stages of data modelling.

Conceptual modelling in Multidimensional Modelling (MDM) allows designers to describe the data warehousing architecture at a high level of abstraction [10, 29, 39, 40]. Historically, MDM solutions have followed two main trajectories. On the one hand, some approaches emphasize a rigorous formal foundation for data cubes, hierarchies, and admissible OLAP operations, often prioritizing expressive power and theoretical soundness over graphical usability. On the other hand, alter-

native approaches focus on conceptual clarity through rich graphical representations, explicitly designed to support communication between designers and stakeholders during requirement elicitation, validation, and schema evolution [3, 62, 79].

These graphical models play a crucial role in making multidimensional structures, aggregation semantics, and navigation paths intelligible to both technical and non-technical users. The choice of conceptual formalism, whether Entity–Relationship-based, object-oriented, or based on dedicated multidimensional visual notations, has a direct impact on the model’s readability, expressiveness, and suitability for supporting well-defined OLAP operations within a given analytical context.

Several notable conceptual models have shaped the MDM landscape. Below are brief descriptions of some influential approaches that illustrate the diversity and evolution of MDM thinking.

The Husemann Multidimensional Model (HMM)

The Husemann Multidimensional Model (HMM) centres on a multidimensional space comprising dimensions, hierarchies, and measures [81]. Its methodology spans requirement determination, design of schema elements (dimensions, measures, hierarchies), fact table creation, data population, and user interface implementation. A key feature of HMM is its emphasis on a clear and understandable schema structure, often realized using star or snowflake schemes [94]. The star schema features a central fact table linked to dimension tables, while the snowflake schema offers a more normalized structure for dimension hierarchies. HMM also advocates for a metadata repository to manage schema information, thereby supporting data integration, quality, and lineage. It has found application in diverse domains such as finance, retail (as exemplified in Figure 3.1), and healthcare.

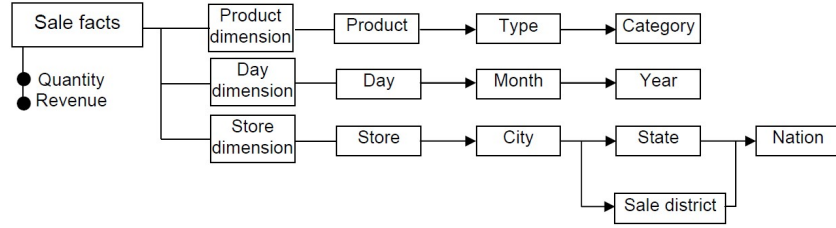


Figure 3.1: Example of Husemann Model: Sales Fact [81]

The Lujan-Mora Multidimensional Model (LMMM)

The Lujan-Mora Multidimensional Model (LMMM) offers a comprehensive framework for designing data warehouses and multidimensional databases, structured around dimensions, measures, and cubes [55]. A distinctive aspect of LMMM is its use of three abstraction levels, employing UML package grouping mechanisms to manage complexity in large DW systems, though these can be merged for simpler designs [55,56]. LMMM places strong emphasis on hierarchies for capturing relationships between dimension levels and attributes (e.g., product category $\rightarrow$ subcategory $\rightarrow$ product) as could be seen in Figure 3.2 . It also incorporates aggregate functions (sum, count, average, etc.) as integral for data summarization and analysis.

The Dimensional Fact Model (DFM) of Golfarelli and Rizzi

The Dimensional Fact Model (DFM), proposed by Golfarelli and Rizzi, is a widely recognized conceptual model based on the hypercube concept [39,40,63]. It emphasizes usability and flexibility, enabling end-users to intuitively analyse data via the hypercube structure and allowing designers to adapt the data warehouse to evolving business requirements.

A Fact Schema (FS), the core construct of the DFM, represents a multidimensional fragment of reality centred on a specific fact - an event, process, or state that can be measured and analysed. It consists of a central fact characterized by quanti-

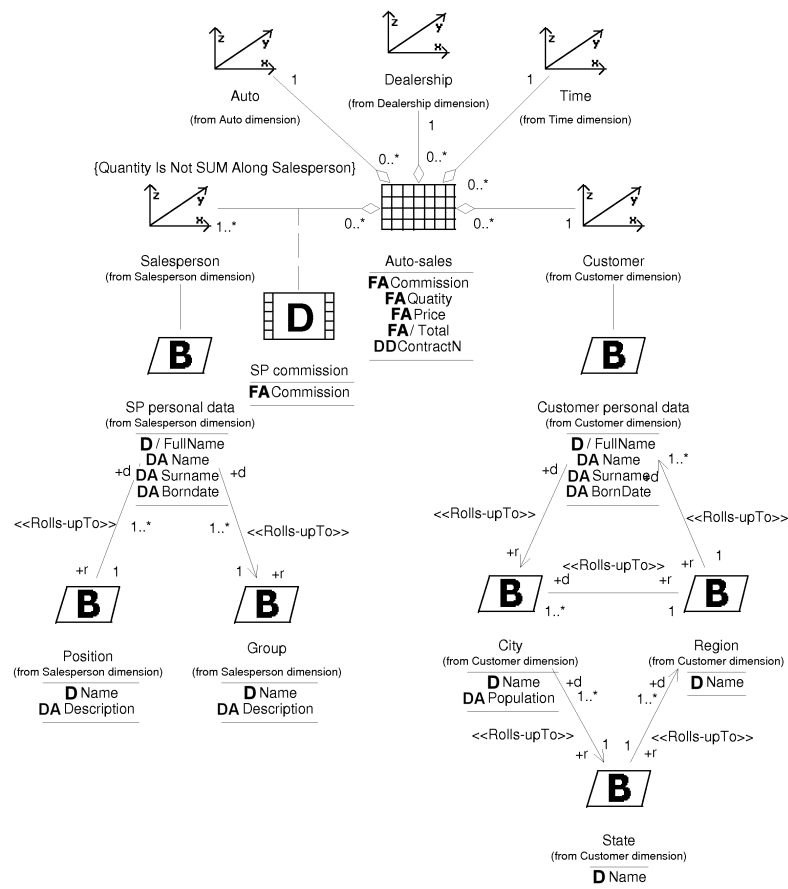


Figure 3.2: An example of modelling in the LMMM

tative measures (e.g., sales amount, account balance) and a set of dimensions that provide the contextual axes for analysis (e.g., time, customer, product). The FS is typically represented as a quasi-tree, with the fact node at the root and dimensions branching out hierarchically through attributes and levels of granularity.

The DFM methodology typically involves identifying business needs, designing schema components (dimensions, measures, hierarchies), creating fact tables, and implementing the user interface. It has proven applicable in various domains, including retail, finance, and healthcare, as illustrated by the Sales FS in Figure 3.3. Due to its clarity, robust foundation, and extensibility, the DFM was selected as the primary reference model for the research presented in this thesis and will be discussed in detail in the following sections.

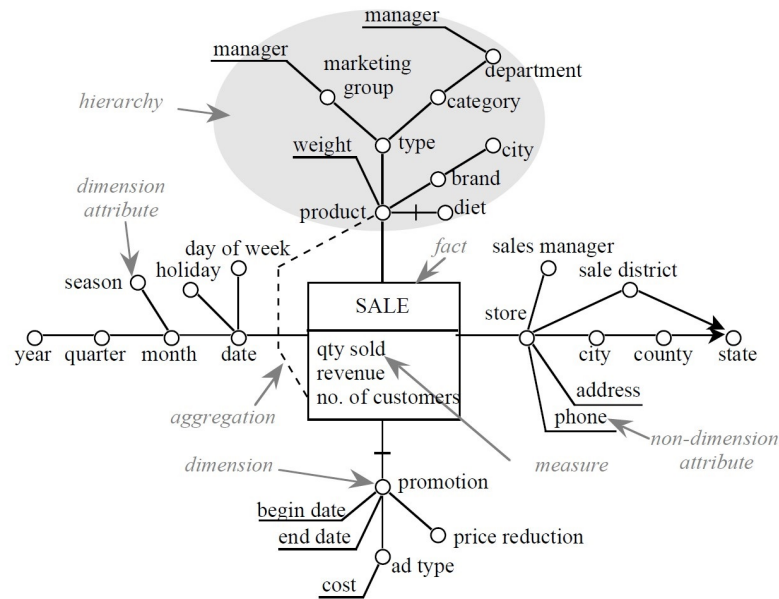


Figure 3.3: The Golfarelli and Rizzi DFM Example for Sales [39]

Contributions by Vaisman and Zimányi

Vaisman and Zimányi have made significant contributions to dimensional modelling and data warehousing [41]. Their work notably addresses the complexities of tem-

poral dimensions, including techniques for managing slowly changing dimensions (SCDs) [38] and temporal hierarchies, which are crucial for historical trend analysis. They have also extended dimensional models to incorporate spatial data, complete with spatial hierarchies and measures for geographic analysis. Their research advocates for systematic data warehouse design methodologies aligned with business processes and emphasizes advanced OLAP techniques and comprehensive metadata management to ensure data quality, integrity, and effective governance. These contributions are foundational for robust BI systems, complex event processing, and data integration from heterogeneous sources.

Manipulations and Properties of Multidimensional Data

Multidimensional data, often conceptualized as OLAP cubes, underpins powerful analytical processing. Users explore these datasets through a set of standard OLAP operations that facilitate interactive data exploration, reporting, and insight generation [110].

Fundamental OLAP Operations. Core OLAP operations allow analysts to navigate and restructure the data cube dynamically. Other common operations, often supported by query languages like SQL, include filtering, grouping, and ranking data within the multidimensional context (see Table 3.1 for a summary). These operations are performed on OLAP cubes where each cell typically contains a measure aggregated according to the selected dimension values.

Dimension Conformity and Compatibility. For effective data integration and analysis across different fact tables or data marts, the concept of **dimension conformity** is critical. Conformed dimensions share consistent structure and semantics (e.g., a 'Date' dimension used across Sales and Inventory facts should be identical or compatible) [61, 117]. This allows for coherent aggregation and comparison. While strict conformity (identical structures) is ideal, **attribute-level**

Operation	Description
Slicing	Reduces dimensionality by selecting a fixed value for one or more dimensions.
Dicing	Produces a sub-cube by applying filters or selecting ranges across multiple dimensions.
Roll-up	Aggregates data along a dimension hierarchy to a more summarized level.
Drill-down	Navigates from summarized data to more detailed data within a dimension hierarchy.
Pivoting	Reorients the data cube’s dimensional axes for alternative analytical perspectives.
Grouping	Aggregates data based on specified dimension attributes (e.g., via SQL <code>GROUP BY</code>).
Filtering	Applies conditions to restrict the data retrieved (e.g., via SQL <code>WHERE</code> or slicers).
Ranking	Assigns ranks to measure values within specified partitions (e.g., using SQL window functions).

Table 3.1: Common OLAP and Data Manipulation Operations

conformity (semantically equivalent attributes with compatible domains) offers a more flexible approach when integrating disparate systems.

Drilling across involves analysing data from multiple fact tables by leveraging these shared or compatible dimensions. Traditional drill-across assumes strict conformity, but modern approaches also consider semantic mapping and dynamic linking to handle partial conformity or heterogeneous sources [111,128]. This thesis framework considers a relaxed notion of **dimension compatibility**, focusing on analytical utility and supporting dynamic data linking across fact tables, even with structural or granularity differences. Such **dynamic OLAP linking** capabilities are crucial for generating custom views from multiple sources (e.g., Customer Relationship Management (CRM), finance) in diverse analytical scenarios [64,82–84].

Shrunk Dimensions and Aggregation Shrunk dimensions are derived subsets of a more detailed base dimension, reduced in terms of rows (e.g.,

"Active Customers" from "All Customers"), columns, or both [38]. They are often employed in designing aggregate fact tables (summary tables) or when business processes are naturally captured at coarser granularities (e.g., monthly forecasts vs. daily transactions). Understanding shrunken dimensions is important for managing data aggregation and ensuring analytical consistency.

3.2.2 Data Modelling of Temporal Data for Analytics

Effective data modelling for temporal data is crucial for extracting meaningful insights from dynamic systems, particularly in analytical and predictive contexts. This process, much like general data modelling, follows distinct stages and leverages specific techniques to handle the time dimension.

The data modelling process is inherently iterative and typically unfolds across multiple distinct stages, transitioning from abstract business understanding to concrete technical implementation. These stages are critical for ensuring that the final data model accurately reflects organizational requirements while supporting efficient data processing and analysis, including temporal aspects [31].

- **Conceptual Data Modelling:** This foundational phase centres on capturing and representing the high-level requirements of the system or analytical task. It involves identifying the primary entities or concepts, their attributes, and the relationships between them, including any temporal characteristics. At this stage, the model remains technology-agnostic and focused on business semantics [96].
- **Logical Data Modelling:** Building on the conceptual model, this phase translates the identified entities and relationships into formal structures suitable for implementation within a specific data modelling paradigm (e.g., relational or multidimensional). For temporal data, this might involve defining

how time is represented (e.g., time points, intervals) and how temporal relationships are structured, for instance, in normalized tables or star/snowflake schemes with dedicated time dimensions [38].

- **Physical Data Modelling:** The final stage adapts the logical model for deployment within a specific Database Management System (DBMS). This includes specifying data types for temporal attributes, indexing strategies for efficient time-based queries (e.g., on valid time or transaction time), storage formats, partitioning schemes (often by time), and optimizations for performance and scalability of temporal queries [89].

Together, these stages ensure that the data model is robust and traceable to business needs. It is also capable of supporting both traditional analytical tasks and more advanced prediction-oriented scenarios, as discussed in [138] and developed in this thesis.

Temporal Data Windows in Analytics

Understanding and analysing dynamic systems, especially for future-oriented decision-making, requires a structured approach to temporal data. Central to this is the use of **temporal windows**, which define discrete intervals for the collection, aggregation, and interpretation of time-based data. Such methods are foundational across disciplines like time series analysis, temporal data mining, and predictive analytics [134, 135].

Several standard strategies exist for temporal segmentation. These include *sliding windows*, which move incrementally across time, often with overlaps between consecutive windows, and *fixed (or tumbling) windows*, which create distinct, non-overlapping partitions of time [136]. The configuration of these windows—such as

their size (duration), step (for sliding windows), and anchor points—directly influences the discovery and interpretation of temporal patterns and extracted features [162].

In prediction-oriented analytics, temporal windows typically serve dual roles: they define the scope of historical data used as input for modelling (observation period) and delineate the future horizon for which predictions are targeted (prediction period). Many existing frameworks and studies adopt variations of observation and prediction windows, often separated by a temporal gap to prevent information loss or to account for inherent delays [137].

Building upon these established concepts, the framework developed in this thesis refines and formalizes a structured **three-time-window approach**, specifically optimized for analytical modelling and insight generation using multidimensional data and OLAP operations, rather than being solely tailored for machine learning model training. This proposed triadic structure consists of:

- **Observation Window (OW):** The historical interval from which data is gathered to identify patterns, signals, and anomalies that are potentially predictive of future events or outcomes [160].
- **Waiting Window (WW):** A deliberately incorporated temporal gap between the OW and the Prediction Window. This models operational or natural delays inherent between observed behaviours (or data capture) and their eventual consequences or the availability of outcome data. The WW is crucial for avoiding look-ahead bias and reflecting real-world lags due to processing times, response delays, or system latencies [137, 160].
- **Prediction Window (PW):** The future period targeted for analysis, forecasting, or the observation of outcomes. Insights and features derived from the OW, and contextualized by the WW, are intended to inform or predict events occurring within this window.

In practice, the OW, WW, and PW windows are applied within OLAP environments using standard operations like slice, dice, drill-down, and roll-up, all constrained by temporal dimensions. This allows analysts to trace the evolution of metrics, identify leading indicators, and examine temporal dependencies within structured data cubes, often using SQL queries or Python-driven workflows, thus providing a transparent basis for predictive reasoning [73, 134]. Window intervals can be fixed or adaptive, depending on the domain context and data characteristics; for instance, shorter OWs might suit high-frequency event detection, while longer OWs can reveal seasonal or structural trends [159]. Such windowing structures have seen application in diverse fields like healthcare for analysing patient trajectories [75, 144] and software engineering for improving effort prediction models through careful training data selection [76, 160].

Our framework also incorporates concepts from Moving-Window Temporal Aggregation (MWTa), adapted for an OLAP context. Techniques like sliding windows and various forms of moving averages (e.g., Exponential Moving Averages (EMA) [157, 158]) can be applied via SQL (e.g., using `OVER` clauses) or Python transformations to support trend detection and signal reinforcement within the OW, anticipating outcomes in the PW after accounting for the WW.

In essence, the three windows approach offers a principled and explainable methodology for modelling and reasoning about temporal dynamics in prediction-oriented analysis. By aligning this model with OLAP operations and multidimensional data structures, our approach seeks to balance analytical rigor, temporal sensitivity, and the generation of actionable insights.

Temporal OLAP Operations

Temporal OLAP extends traditional multidimensional analysis by incorporating the explicit management of temporal dimensions and the evolution of both facts and di-

mensions over time. Temporal OLAP extends traditional multidimensional analysis by incorporating the explicit management of temporal dimensions and the evolution of both facts and dimensions over time [102].

Unlike standard OLAP, which treats time merely as another dimension, temporal OLAP (TOLAP) provides operators and models capable of querying and aggregating data across different temporal granularities, validities, and versions [3, 41, 105, 106].

Temporal data warehousing frameworks therefore support two fundamental temporal notions: *valid time* (the period during which a fact holds true in the real world) and *transaction time* (the period during which the fact is stored in the database). Managing both temporal axes allows a warehouse to maintain a complete historical record of data and schema evolution, enabling consistent temporal analysis and auditability [106, 120, 141]. A related challenge is handling *slowly changing dimensions* (SCDs), where dimension members evolve, merge, or split over time [108, 141]. Building on these principles, several temporal OLAP frameworks have been proposed to enrich multidimensional models with explicit time semantics. Koncilia and Eder's COMET model extends the traditional star schema to support bi-temporal data, enabling users to track and query changes in both dimensions and facts with respect to valid time and transaction time [105, 106].

Similarly, Vaisman and Zimányi, and later Ahmed et al., introduced temporal multidimensional models that define a comprehensive set of time-aware OLAP operations, such as temporal roll-up, drill-down, slice, and dice, operating over time-variant facts and dimensions [41, 104, 121]. Combi et al. further proposed the T^+ MultiDim model, which distinguishes between instant-based and interval-based temporal semantics, allowing a more precise representation of events, states, and their evolution within multidimensional analytical settings [3, 114].

Key temporal OLAP operations include:

- **Temporal Roll-up and Drill-down:** Aggregate or disaggregate measures across temporal hierarchies (e.g., day $\rightarrow$ month $\rightarrow$ quarter $\rightarrow$ year), while correctly accounting for the temporal validity of facts and dimensions.
- **Temporal Slice and Dice:** Select subsets of data valid at a specific instant or during an interval (e.g., all sales valid during Q1 2023 or transactions recorded between two timestamps).
- **Temporal Join and Drill-across:** Combine facts or dimensions that belong to different temporal contexts (e.g., join observation-period data with prediction-period data to study temporal dependencies).
- **Temporal Comparison:** Compare the state of a cube at two different instants (e.g., before and after a policy change) to detect temporal evolution or change patterns.
- **Temporal Aggregation:** Apply moving or cumulative functions (e.g., cumulative sum or moving average) over valid-time intervals or transaction-time periods to derive temporal trends [73, 134, 139].

Modern implementations of temporal OLAP exploit SQL extensions, such as temporal predicates (`AS OF`, `PERIOD`), and analytical window functions for efficient computation of time-dependent aggregates. These capabilities, supported by in-memory and cloud-based architectures, enable interactive exploration and predictive analysis across evolving temporal datasets, ensuring both performance and temporal consistency [107].

3.2.3 prediction-oriented and Temporal Analytics

This section reviews relevant research in multidimensional modelling, OLAP, and specifically, approaches addressing temporal aspects and prediction-oriented analysis

within these contexts. We examine challenges in traditional MDM that motivate the need for advanced frameworks, discuss existing work on temporal data warehousing and analytics, explore techniques for prediction-oriented multidimensional analysis, and survey relevant visualization methods.

Challenges in Multidimensional Data Integration and Flexibility

Effective advanced analytics often requires integrating data from diverse sources and navigating complex data structures. Traditional MDM approaches, while powerful, present several challenges in this regard.

A common restriction in multidimensional data models is the assumption of homogeneous and balanced dimension hierarchies, which can be too rigid for many real-world applications. Abelló et al. [52] proposed a framework for classifying and modelling complex multidimensional data, focusing on conceptually reshaping heterogeneous dimensions into homogeneous "sub-dimensions" and enforcing summarizability across all dimension hierarchies.

The **drill-across operation**, vital for integrating data from multiple fact tables (or cubes), is traditionally defined as linking tables at the same granularity and sharing conformed dimensions [38,84]. If dimensions are not shared or differ in grain, standard drill-across is inapplicable. Abelló [54] extended this definition to allow navigation between cubes even with non-shared dimensions, offering more flexible analysis. However, such flexibility necessitates careful attention to data aggregation to ensure correctness. Lehner et al. [126] and Lenz et al. [96] emphasized the critical importance of maintaining summarizability, the correct computation of aggregate values from finer to coarser levels of detail, when merging data, to prevent erroneous results. While many-to-one associations typically satisfy summarizability, many-to-many (M:M) relationships can violate it, potentially leading to data replication if not handled appropriately.

Samos et al. [113] further explored semantic relationships between facts, noting that their proposed drill-across operator allows users to navigate between cubes based on connections between facts themselves, even if dimensions are not directly related. They highlight that M:M relationships between facts must be resolved to a one-to-one correspondence at the aggregation level before drill-across to avoid data replication and double-counting. modelling M:M relationships between facts and dimensions has been addressed through various techniques. For instance, Kimball's method uses a bridge table for accurate weighted summaries and reduced redundancy [38], while Inmon's approach involves separate bridge tables for flexibility, potentially at the cost of increased complexity [89]. Other techniques involve modifying table structures, adjusting fact table granularity, or denormalizing dimension tables with positional or non-positional flag attributes.

Integrating data with **non-conformed dimension attributes** presents another hurdle. Riazati et al. [128] broadened the drill-across definition to accommodate analysis across multiple cubes with such attributes, provided there's a semantic correspondence (e.g., two different attributes both representing a 'location' concept). Their work includes methods to assess information loss when joining attributes with dissimilar values, thereby enhancing analytical depth with complex data models.

For handling multi-source data analyses where schema conformity is lacking, Giovinazzo et al. [117] proposed "Unified Cubes," a graphical notation extending star schemes for "constellations" (sets of star schemes sharing dimensions), though relationships between distinct sub-schemes were not deeply explored. Vaisman and Zimányi [41] outlined two data source integration approaches: loosely coupled (identifying common information and joining sources on-the-fly) and tightly coupled (creating a materialized, merged view against which queries are performed).

To handle the sheer scale and dimensionality of modern datasets, Akid et al. [122] proposed NoSQL databases for their scalability and flexibility, addressing the "curse

of dimensionality.” Complementing these storage solutions, business intelligence tools and advanced visualization techniques are crucial for comprehending multi-dimensional datasets [124].

Temporal Data Warehousing and Analytics: Existing Approaches

Incorporating the temporal dimension effectively is critical for both descriptive and predictive analytics. While foundational MDM models provide basic time dimensions, specialized temporal data warehousing techniques address more complex temporal requirements.

Key aspects include modelling strategies for capturing *valid time* (when a fact is true in the real world) and *transaction time* (when a fact is recorded/alterd in the database), leading to bi-temporal data models, and managing *slowly changing dimensions* (SCDs) to track historical attribute changes [140, 141].

Ahmed et al. [121] proposed real-time temporal data warehousing cubing, focusing on incorporating temporal features directly into OLAP for more accurate predictions. This highlights a move towards making temporal data more readily usable for predictive tasks within the DWH environment.

A crucial aspect of temporal warehouse design is the management of Slowly Changing Dimensions (SCDs), which track the evolution of attribute values over time. Among the classic approaches (Types 0-6), the Type 2 method, which creates new dimension records for each change along with time validity metadata, is most suitable for advanced analytics scenarios where historical accuracy is required. This is particularly important for predictive analysis: to train reliable models, it is essential to know the exact state of each descriptive dimension at the time a fact event occurred. Without proper SCD management, analytic processes risk attributing events to incorrect dimensional states, leading to temporal leakage and biased model performance. As such, robust SCD strategies form a foundational prerequisite

for temporally consistent, prediction-oriented multidimensional analysis.

Frameworks and Techniques for prediction-oriented Multidimensional Analysis

While traditional OLAP excels at retrospective analysis, there is a growing body of work aiming to bridge OLAP with predictive capabilities, making multidimensional data more directly amenable to forecasting and other predictive tasks.

The general literature acknowledges limitations in current MDM for predictive tasks. For instance, Boukraa et al. [123] note that while multidimensional models effectively capture analytical perspectives, their direct use for predictive modelling often requires extensions. Viswanathan and Schneider [125] addressed challenges in spatial data modelling within MDM, emphasizing needs like new spatial aggregation operations, which parallel the need for new prediction-oriented operations. Pedersen [90] also highlighted unfulfilled MDM requirements such as better support for M:M relationships, built-in mechanisms for time and change, uncertainty handling, and varied data granularity, all of which can impact the quality of data fed into predictive models.

Visualization of Complex Multidimensional Data

Visual exploration is a key component of multidimensional data analysis, aiding in pattern discovery and hypothesis generation, which can precede or complement predictive modelling. Pioneering works in multidimensional data visualization include Vinnik's tree-shaped method and Stolte and Tang's Polaris [131], a tool that divides data into groups for visualization in separate panes, facilitating the analysis of multiple cubes. Eick's ADVIZOR [130] employs linked views on a single screen, with each view representing different measurements. Visual Pivot by Conkin, Prabhakar, and North [133] addresses polyarchies (multiple, intersecting hierarchies),

enabling information tracking across these structures. A fundamental CoDecide by Gebhardt [132] is designed for cooperative analysis of data cubes from various perspectives. While these techniques offer rich interactive exploration, pivot tables remain widely used due to their familiar tabular format, readability, and ease of manipulation, with nested pivot tables leveraging conformed attributes for comprehensive analysis across multiple sources.

Further research addresses modern approaches to multidimensional data visualization and OLAP. For instance, Nguyen et al. [173] utilized bar charts and diagrammatic reasoning for visualizing complex multidimensional port event data. Bondarev [165] discussed visual analysis for cluster structures in multidimensional datasets. Di et al. [169] highlighted limitations of existing techniques for visualizing multidimensional data in OLAP contexts, pointing to the need for more efficient schemes. Korobko [172] presented an approach to exploratory OLAP addressing multidimensional model visualization and formal conceptual analysis. The practical significance of effective visualization for business intelligence is underscored by Abelló and Romero [164], who discussed how enterprises use OLAP to derive knowledge from multidimensional data. Shu [171] explored hierarchical visualization using star diagrams, while Dzemyda et al. [170] presented a web application for large-scale multidimensional data visualization with data mining services. These studies collectively emphasize the ongoing importance and evolution of advanced visualization methods for interpreting complex multidimensional datasets.

3.2.4 Process Mining and Predictive Monitoring of Business Processes

Process mining and the framework developed in this thesis operate at different, although potentially complementary, levels of abstraction. Process mining is process-

aware: its principal analytical objects are cases, event traces, activities, and process models. Its central questions concern how a process is executed, whether observed behaviour conforms to an expected model, and how the process may be improved. Predictive process monitoring extends this perspective by using complete historical traces and prefixes of running cases to predict future activities, outcomes, remaining time, or other process-performance indicators [146–149].

The Prediction-Oriented Framework is instead data-model-aware. Its principal analytical objects are fact schemes, dimensions, temporal bindings, analytical anchors, and multidimensional snapshots. Its purpose is not to discover a process model or directly learn a predictor from an execution prefix. It specifies which data are admissible as historical observations, which interval represents latency or outcome maturation, which data define future targets, and how heterogeneous facts can be integrated without violating temporal precedence.

The distinction also clarifies the role of the log modelling contribution in Chapter 2. The proposed UML-based model can document the structure and semantics of event data before a process-mining pipeline is executed. In particular, explicit event identifiers, process-context identifiers, timestamps, controlled vocabularies, field dependencies, and nesting structures can support more consistent event correlation and data preparation. The model does not itself perform process discovery or conformance checking.

Accordingly, the contributions of this thesis should be viewed as complementary to, rather than replacements for, process mining and predictive process monitoring. The log model supplies an upstream semantic representation for heterogeneous event data, whereas the PO Framework supplies a multidimensional and temporally explicit data-preparation layer. Downstream process-aware or machine-learning-based methods may consume data structured through these layers, but the evaluation of such an integrated pipeline remains a direction for future research.

3.3 Motivation and Problem Statement

This section introduces a framework for prediction-oriented analysis over multidimensional data structures. Building upon the OLAP operations and data integration strategies discussed previously, we shift the focus toward enhancing analytical workflows that support forecasting, risk assessment, and decision-making. Standard analytical methods often treat time as a linear, monolithic dimension, which limits their predictive power. We argue that for effective prediction-oriented analytics, the timeline must be partitioned into distinct, purposeful segments. To illustrate this necessity, we begin by presenting a motivating use case from the financial sector where this limitation often becomes a critical bottleneck.

3.3.1 Use Case: prediction-oriented Credit Risk Analysis

Financial institutions rely heavily on analytical systems to manage credit risk. These systems ingest historical data from various sources - transactional records, loan histories, customer profiles, and external economic indicators - to produce risk reports and scorecards [166–168]. However, a crucial challenge lies in structuring this temporal data to accurately “predict” future events, rather than merely describing past trends.

Consider a risk analyst at a mid-sized bank. Their primary task is not just to identify high-risk customers, but to build a stable, reliable predictive model that can be deployed in a real-world operational cycle. Specifically, the bank wants to **identify customers who are likely to default on their obligations within a future 90-day period.**

To build this analytical model, the analyst must navigate a complex temporal landscape.

1. *Feature Engineering Period:* The analyst needs to look at a customer’s past be-

haviour to create predictive features. For example, they might look at the last *180 days* of credit card transactions and loan repayments to calculate metrics like `average\_credit\_utilization`, `number\_of\_missed\_payments`, or `consistency\_of\_income-to-debt\_ratio`. This is the historical observation period.

2. *Outcome Period*: The objective of the model is to predict default within the future *90-day period*. This is the window where the event we are trying to predict (default) either occurs or does not occur.
3. *The Latency Gap*: A critical real-world constraint is that data is never truly real-time. There is a delay, or *latency*, between when an event occurs and when it is reliably recorded in the analytical system. A payment might be missed on the 15th, but it may only be officially flagged as delinquent a week later. A customer might functionally default, but the account's final status is only confirmed and logged at the end of the monthly accounting cycle. This creates a "blind spot" or a quarantine period. It is unsafe to use data from this period for prediction, as the outcome might already be unfolding but is not yet visible in the data, a phenomenon known as *target leakage* in machine learning [115]. This leakage risk is not confined to machine learning pipelines; even in standard OLAP queries, including data from the latency gap can inadvertently incorporate future information into historical aggregates, leading to overly optimistic or misleading analytical conclusions. Let's assume this is a *30-day period*.

This structured approach forces the analyst to confront questions that are impossible to answer with standard OLAP tools:

- *How does a pattern of behaviour within a defined historical period correlate with the probability of default in a specific future outcome period?*

- *How can we systematically create training datasets that respect the data latency gap, ensuring that our models are not contaminated by information that would not be available at the time of prediction?*
- *How do we align slow-moving economic indicators (e.g., quarterly unemployment rates) with fast-moving customer transactions within this partitioned timeline?*

This problem structure reveals the core limitations of traditional tools. While a conventional OLAP query focuses on aggregating past data, it cannot enforce the crucial separation between the historical, latency, and outcome periods. It cannot systematically test hypotheses across these distinct timeframes (e.g., "does a shorter historical period reduce observational accuracy?") or manage the alignment of features from disparate sources according to this structure.

This scenario, while rooted in finance, epitomizes challenges in other domains:

- *Healthcare:* Observing and analysing hospital readmission involves analysing patient history before a discharge-to-readmission gap to forecast risk in a subsequent 30-day period.
- *Manufacturing:* Predicting equipment failure requires using sensor data history before a maintenance and reporting lag to predict failure probability in the next operational cycle.
- *Retail:* Predicting customer churn often involves modelling purchase history before a cooling-off period to estimate the likelihood of churn in the next quarter.

In each case, the ability to formally define, separate, and analyse across these temporal periods is a core for building robust and reliable decision-support models.

3.3.2 Limitations of Traditional Multidimensional Models

Let us now define a simplified schema and provide sample data to ground the discussion. The schema includes two primary fact tables capturing behavioural transactions - `Credit Card Fact` and `Loan Fact` - as well as supporting dimensions like `Customer` and `Economic_Indicators`. These tables are connected via shared dimensions such as `CustomerID`, `Date` and `Region` fields, enabling multi-source feature construction.

Customer ID	Name	Birth Date	Region	Risk Category
C123	John Doe	1985-05-20	North	Medium
C456	Jane Smith	1992-11-10	South	Low
C789	Alice Brown	1978-02-15	North	High

Table 3.2: Customer Dimension

Trx ID	C. ID	Date (TS)	Total Balance	Limit	Pay Due	Pay Made	Trx Amount
1	C123	2023-11-15 10:00:00	4500	5000	400	400	200
2	C123	2023-12-15 14:00:00	4800	5000	500	300	250
3	C456	2023-12-20 09:00:00	1200	10000	200	200	180
4	C123	2024-01-15 11:00:00	4950	5000	600	0	300

Table 3.3: Credit Card Fact Table

Date	Customer ID	Loan Amount	Installment Due	Installment Paid	Status
2023-10-01	C789	25000	500	500	On-Time
2023-11-01	C789	25000	500	0	Delayed
2024-03-10	C789	25000	500	0	Default

Table 3.4: Loan Fact Table

Date	Region	Inflation Rate	Unemployment Rate
2023-10-01	North	0.035	0.052
2023-10-01	South	0.031	0.041
2024-01-01	North	0.038	0.055
2024-01-01	South	0.033	0.043

Table 3.5: Economic Indicators Dimension

analysing the Temporal Challenge

To structure the analysis, we first define a specific **anchor point**, t_0 , which represents the "present" moment from which historical and future data are managed. Let's set $t_0 = '2024-01-01'$. Based on the intervals from our use case, the three distinct time periods are:

- **The Historical Period (for Feature Engineering):**

$$[t_0 - 180 \text{ days}, t_0] \rightarrow ['2023-07-05', '2024-01-01']$$

- **The Latency Gap (for Data Delays):**

$$[t_0, t_0 + 30 \text{ days}] \rightarrow ['2024-01-01', '2024-01-31']$$

- **The Outcome Period (for Target labelling):**

$$[t_0 + 30 \text{ days}, t_0 + 120 \text{ days}] \rightarrow ['2024-01-31', '2024-05-01']$$

With this structure, the sample data illustrate the core challenge:

- **For Customer C789**, the model must use the 'Delayed' payment status from their history (2023-11-01) as a feature to link to the "future" 'Default' status that occurs in the outcome period on 2024-03-10 (*as shown in Table 3.4, the delayed instalment on 2023-11-01 precedes the default event on 2024-03-10*).
- **For Customer C123**, the missed payment on 2023-12-15 is a valid feature from the historical period. However, the missed payment on 2024-01-15 falls within the latency gap and must be excluded from feature engineering to prevent target leakage.
- The **Economic Indicators** for Q4 2023 (dated '2023-10-01') provide the macroeconomic context for the historical period, while the Q1 2024 data align with the subsequent latency and outcome periods.

Example 3.3.1: OLAP Query for Missed Payments

A standard analytical approach might involve aggregating data to identify trends, as shown in the following OLAP query in Listing 4 , which computes missed payments by region and month.

```
SELECT
    c.Region,
    DATE_TRUNC('month', f.Date) AS Month,
    SUM(CASE WHEN f.PaymentDue > f.PaymentMade THEN 1 ELSE 0 END) AS
    ↪ MissedPayments
FROM CreditCard_Fact f
JOIN Customer c ON f.CustomerID = c.CustomerID
GROUP BY c.Region, DATE_TRUNC('month', f.Date)
ORDER BY c.Region, Month;
```

Listing 4: Monthly Aggregation of Missed Payments by Region

The query output (Figure 3.4) summarizes the monthly evolution of missed

payments across customer regions. In the **North** region, missed payments first appear in October 2023 and persist through subsequent months: one missed payment in both October and November 2023, two in December 2023, and one in January 2024. In contrast, the **South** region exhibits consistently stable behaviour, with no missed payments recorded across the entire observation period. These results provide a simple descriptive profile of repayment behaviour, reflecting both temporal and regional variation in credit risk exposure.

	Region	Month	MissedPayments
1	North	2023-07-01 00:00:00.000	0
2	North	2023-08-01 00:00:00.000	0
3	North	2023-09-01 00:00:00.000	0
4	North	2023-10-01 00:00:00.000	1
5	North	2023-11-01 00:00:00.000	1
6	North	2023-12-01 00:00:00.000	2
7	North	2024-01-01 00:00:00.000	1
8	South	2023-09-01 00:00:00.000	0
9	South	2023-10-01 00:00:00.000	0
10	South	2023-12-01 00:00:00.000	0

Figure 3.4: Query Output: Missed Payments by Region and Month

While informative, this query illustrates a fundamental limitation of traditional OLAP analysis: time is treated as a flat, undifferentiated dimension. The aggregation captures historical variation but lacks an explicit notion of analytical roles such as a feature period or a target period. Consequently, although the results reveal an increasing trend in missed payments in the **North** region, the query cannot distinguish whether these events belong to the past (used to explain future behaviour) or to the future (used as prediction targets).

As a result, standard OLAP queries frequently conflate temporally distinct data types—features, delayed observations, and outcomes—because the temporal separation between them is not explicitly modelled. Instead, this separation is

expressed only transiently within the **WHERE** clause of an individual query, making analytical procedures error-prone, inconsistent across analyses, and difficult to reproduce or govern.

To clarify this limitation, the following example introduces a query that explicitly distinguishes between a *historical period* (used for explanatory feature computation) and an *outcome period* (used for target evaluation). By making temporal roles explicit within the analytical process, this approach yields a structure that is transparent, auditable, and aligned with the principles later formalized in the three-window Prediction-Oriented Framework.

As a result, queries often conflate temporally distinct data types—such as features, delayed observations, and labels—because the system lacks built-in semantics for enforcing strict temporal partitioning. The critical rules that separate time periods are not embedded in the data model; instead, they are expressed implicitly and transiently within the **WHERE** clause of individual queries. This approach makes temporal separation error-prone, inconsistent across analyses, and difficult to govern or validate systematically.

To clarify this point, the following query presents an example that explicitly distinguishes between a *historical period* and an *outcome period*. The structure separates explanatory variables and target data within a single analytical query by means of explicit temporal constraints, rather than relying on a single, implicit **WHERE** clause.

```

-- Example: Distinguishing Historical vs. Outcome Periods in SQL
WITH
-- 1. Historical period: past data used to compute explanatory features
historical_features AS (
    SELECT
        f.CustomerID,
        SUM(CASE WHEN f.PaymentDue > f.PaymentMade THEN 1 ELSE 0 END) AS
        ↪ missed_payments_hist,
        AVG(f.TotalBalance / NULLIF(f.CreditLimit, 0)) AS avg_utilization_hist
    FROM CreditCard_Fact f
    WHERE f.Date BETWEEN DATE '2023-07-05' AND DATE '2024-01-01' -- historical
        ↪ period
    GROUP BY f.CustomerID
),
-- 2. Outcome period: future data used to define target variable
future_outcomes AS (
    SELECT
        CustomerID,
        MAX(CASE WHEN Status = 'Default' THEN 1 ELSE 0 END) AS
        ↪ has_defaulted_future
    FROM Loan_Fact
    WHERE Date BETWEEN DATE '2024-01-31' AND DATE '2024-05-01' -- outcome
        ↪ period
    GROUP BY CustomerID
)
-- 3. Combine both to analyse dependencies across time
SELECT
    h.CustomerID,
    h.missed_payments_hist,
    h.avg_utilization_hist,
    f.has_defaulted_future
FROM historical_features h
LEFT JOIN future_outcomes f ON h.CustomerID = f.CustomerID
ORDER BY h.CustomerID;

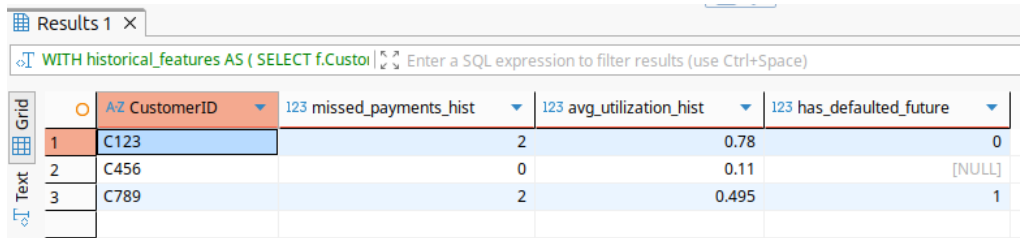
```

Listing 5: Distinguishing Historical vs. Outcome Periods in SQL

Unlike standard OLAP queries that mix past and future events within a single date filter, the query in Listing 5 explicitly partitions time into two semantically distinct ranges: a **historical period** for feature computation and an **outcome period** for target evaluation. Each subquery defines its temporal boundaries independently, making the logic transparent, auditable, and reproducible. This explicit separation of temporal roles operationalizes the principle of **temporal precedence**, antici-

pating the need for formalized frameworks that will be introduced in the following sections.

The resulting output, shown in Figure 3.5, clearly demonstrates the analytical benefit of separating time into distinct functional windows. Customer C789, who accumulated two missed payments and maintained a moderate average credit utilization (≈ 0.50), later defaults in the outcome period (`has_defaulted_future` = 1). Customer C123, with a similarly high utilization rate (≈ 0.78) and two missed payments, does not default, suggesting short-term payment difficulty without long-term insolvency. In contrast, customer C456, characterized by low utilization (≈ 0.11) and no missed payments, shows no default activity, representing a stable financial behaviour profile.



	A-Z CustomerID	missed_payments_hist	avg_utilization_hist	has_defaulted_future
1	C123	2	0.78	0
2	C456	0	0.11	[NULL]
3	C789	2	0.495	1

Figure 3.5: Query Output: Historical Features and Future Outcomes by Customer

This concise example demonstrates how a minimal two-window query design can preserve temporal precedence while yielding interpretable, entity-level patterns that connect historical behaviour to future outcomes. However, traditional multidimensional systems lack mechanisms to enforce such distinctions automatically. This deficiency becomes particularly critical in prediction-oriented analytical settings, where maintaining strict temporal partitioning is essential to ensure analytical validity and to prevent missing data.

Three main challenges emerge:

1. **Lack of Temporal Partitioning:** Standard OLAP tools do not guarantee that feature variables are confined to a historical window or that target

variables are restricted to a future outcome window. This omission risks contamination from overlapping time periods, undermining analytical soundness.

2. **Loss of Temporal Resolution:** Aggregation by coarse temporal units (e.g., month or quarter) obscures the ordering of events and erases fine-grained behavioural patterns that may be critical for predictive modelling or anomaly detection.
3. **No Entity-Centric Alignment:** OLAP systems typically lack constructs for aligning temporal analysis around entity-specific anchors (e.g., customer-level timelines). Without this alignment, comparisons across time windows cannot consistently reflect individualized behavioural dynamics.

Furthermore, OLAP environments often operate across **siloed data cubes**, each optimized for a specific business process. These cubes are typically:

- Developed independently, without harmonized dimensions or synchronized schemes [38, 101];
- Characterized by heterogeneous temporal and semantic granularities [39, 103];
- Incompatible for cross-domain or drill-across operations when datasets (e.g., credit and macroeconomic indicators) differ in aggregation level or update frequency.

Table 3.6 summarizes these structural and semantic limitations in the context of predictive credit risk modelling.

Example 3.3.2: OLAP Query for a Unified Financial Summary

To further illustrate the problem, consider a more advanced OLAP task in which an analyst attempts to create a unified financial summary for each customer. Listing 6 illustrates a classic *drill-across* query that aggregates heterogeneous

Limitation	Manifestation in Queries and Analysis
Rigid Temporal Structure	No built-in support for distinguishing observation, latency, and prediction windows; increases the risk of data leakage and invalid feature-target relationships.
Siloed Data Cubes	Fragmented data representation limits the integration of multi-domain features (e.g., linking credit utilization with economic indicators).
No Predictive modelling Support	Traditional OLAP systems cannot define labels, generate training instances, or enforce temporal validation splits required for supervised learning.
Inflexible Dimension Conformity	Coarse-grained dimensions (e.g., region-level economic data) are difficult to align with fine-grained, entity-level behavioural timelines.

Table 3.6: Limitations of Traditional OLAP for prediction-oriented Credit Risk Analysis

financial `CreditCard_Fact` and `Loan_Fact` tables to a common monthly grain before joining them.

Listing 6: Unified Monthly Shortfall Aggregation Across Credit and Loan Products

```
-- Attempting to create a unified monthly summary for each customer
WITH
-- 1. Aggregate credit card data to a monthly level
credit_summary AS (
    SELECT
        CustomerID,
        DATE_TRUNC('month', Date) AS activity_month,
        SUM(PaymentDue - PaymentMade) AS credit_shortfall
    FROM CreditCard_Fact
    GROUP BY CustomerID, activity_month
),
-- 2. Aggregate loan data to a monthly level
loan_summary AS (
    SELECT
        CustomerID,
        DATE_TRUNC('month', Date) AS activity_month,
        SUM(InstallmentDue - InstallmentPaid) AS loan_shortfall
```

```

    FROM Loan_Fact
    GROUP BY CustomerID, activity_month
)
-- 3. Join the monthly summaries to get a unified view
SELECT
    c.CustomerID,
    c.Name,
    COALESCE(cs.activity_month, ls.activity_month) AS month,
    COALESCE(cs.credit_shortfall, 0) AS credit_shortfall,
    COALESCE(ls.loan_shortfall, 0) AS loan_shortfall,
    (COALESCE(cs.credit_shortfall, 0)
     + COALESCE(ls.loan_shortfall, 0)) AS total_monthly_shortfall
FROM Customer c
LEFT JOIN credit_summary cs
    ON c.CustomerID = cs.CustomerID
LEFT JOIN loan_summary ls
    ON c.CustomerID = ls.CustomerID
    AND cs.activity_month = ls.activity_month
ORDER BY c.CustomerID, month;

```

Figure 3.6 presents the result of the drill-across query described above. Each row represents a customer’s unified financial shortfall at the monthly level, integrating both credit card and loan data into a single summary.

As shown in the query output, customer C123 (John Doe) begins to accumulate shortfalls starting in October 2023, with a combined amount of 100 that rises to 200 in December 2023 and 600 in January 2024. This pattern reflects a steady deterioration of payment behaviour over time, visible only through temporal aggregation. Customer C456 (Jane Smith) exhibits no financial shortfalls across all observed months, while C789 (Alice Brown) displays both credit and loan shortfalls concentrated around November and December 2023, with a joint

deficit of 1000 in November and 600 in December, indicating multi-source risk exposure.

	AZ CustomerID	AZ Name	month	123 credit_shortfall	123 loan_shortfall	123 total_monthly_shortfall
1	C123	John Doe	2023-07-01 00:00:00.000	0	0	0
2	C123	John Doe	2023-08-01 00:00:00.000	0	0	0
3	C123	John Doe	2023-09-01 00:00:00.000	0	0	0
4	C123	John Doe	2023-10-01 00:00:00.000	100	0	100
5	C123	John Doe	2023-12-01 00:00:00.000	200	0	200
6	C123	John Doe	2024-01-01 00:00:00.000	600	0	600
7	C456	Jane Smith	2023-09-01 00:00:00.000	0	0	0
8	C456	Jane Smith	2023-10-01 00:00:00.000	0	0	0
9	C456	Jane Smith	2023-12-01 00:00:00.000	0	0	0
10	C789	Alice Brown	2023-08-01 00:00:00.000	0	0	0
11	C789	Alice Brown	2023-10-01 00:00:00.000	0	0	0
12	C789	Alice Brown	2023-11-01 00:00:00.000	500	500	1,000
13	C789	Alice Brown	2023-12-01 00:00:00.000	600	0	600

Figure 3.6: Query Output: Unified Monthly Financial Shortfall by Customer

This example highlights a fundamental limitation of traditional OLAP-style processing: the absence of explicit temporal structuring and predictive framing. OLAP queries are designed to summarize data at predefined temporal granularities (e.g., month or quarter), prioritizing aggregation efficiency over the preservation of event sequencing. As a result, all events occurring within the same aggregation bucket are treated as temporally equivalent, regardless of their actual order or relative timing.

In the present case, monthly aggregation conceals the precise sequence of customer actions, obscuring behaviourally meaningful patterns such as whether a missed credit card payment occurred before or after a loan instalment within the same month. From a predictive standpoint, this distinction is critical, as the ordering of events often conveys important information about the evolution of financial risk. By collapsing heterogeneous events into a single time slice, the query discards ordering information and prevents the analyst from identifying

temporally ordered dependencies between observed behaviours and subsequent outcomes.

This phenomenon corresponds to the problem of asymmetric temporal granularity in analytical systems, where coarse aggregation imposes a fixed temporal resolution that masks important event ordering and causal structure. Early work in temporal data warehousing documents how changes in temporal dimensions and granularity semantics affect the interpretability of analytical queries [105]. More broadly, the data warehousing and temporal analytics literature shows that such mismatches lead to information loss and biased interpretations when the temporal resolution of explanatory variables does not align with the dynamics of the target behaviour [118, 156].

Furthermore, the query lacks an explicit temporal anchor (t_0) and does not distinguish between historical, latency, and outcome windows. In the absence of clearly defined temporal boundaries, there is no formal guarantee that explanatory features are derived exclusively from past data relative to the prediction point. This omission exposes the analysis to temporal entanglement, whereby future or contemporaneous information may inadvertently influence historical aggregates, resulting in target leakage. Such leakage undermines both the validity of predictive models and the interpretability of analytical conclusions.

Thus, while the drill-across query provides a valid descriptive summary for reporting, it remains structurally unsuitable for predictive modelling. Its design merges events across incompatible time horizons and discards sequential dependencies that are critical to forecasting default or delinquency risk. The example reinforces the need for a prediction-oriented modelling approach capable of preserving event-level granularity, enforcing window-based temporal approaches, and integrating heterogeneous facts within a semantically consistent structure.

In summary, although both `CreditCard_Fact` and `Loan_Fact` describe related customer-level financial behaviours, they differ in structure, granularity, and semantic scope, connected only through the shared `CustomerID`. Meanwhile, macroeconomic signals from the `Economic_Indicators` table operate at a regional level and are temporally misaligned with transaction-level events.

This temporal and semantic fragmentation underscores the necessity of a unified, Prediction-Oriented Framework. The following sections introduce such a structure through: (i) rigorous temporal partitioning within a three-window design, (ii) entity-centric alignment via DFM constellation models, and (iii) prediction-oriented feature semantics grounded in explicit observational boundaries.

3.4 Fundamental Concepts for Developing a Framework for Multidimensional Analysis

This section establishes the formal underpinnings of the PO Framework. Building on the conceptual design and the motivating example of financial credit risk, we now define the specific data structures required to support our proposed analytical workflows. The primary goal is to formalize a model that can handle the integration of data from multiple interconnected domains, a common requirement in any sophisticated, forward-looking decision-making process.

3.4.1 The Enhanced Fact Model

At the heart of our framework is the Prediction-Oriented Dimensional Fact Model (PO DFM). This model extends established data warehousing principles to address the unique challenges of prediction-oriented data analysis. In domains like credit risk, analysts must routinely integrate data from multiple, partially overlapping FSs (e.g., combining credit card transactions with loan repayment histories). These schemes often share common dimensions like 'Customer' or 'Date', enabling powerful cross-domain analysis.

The design of a PO DFM is guided by the intended analytical workloads, particularly the queries needed for feature engineering and outcome labelling as dictated by our PO structure. Our model therefore formalizes the structures and operators-such as the crucial *drill-across* operator for navigating between linked data cubes-that are essential for these tasks. It provides a robust, formal foundation for integrating the temporal windowing logic directly into a multidimensional data model.

3.4.1.1 Standard Dimensional Fact Model Definitions

We ground our work in the widely recognized Dimensional Fact Model (DFM) proposed by Golfarelli and Rizzi [39, 40]. DFM provides a formal yet intuitive way to model a real-world domain of interest (e.g., a bank’s credit operations) as a collection of interconnected Fact Schemes (FSs). Each FS represents a specific subject, such as ‘Credit Card Transactions’ or ‘Loan Performance’, and is built on three core concepts:

- **Facts:** Events or snapshots of interest, representing a point in a multidimensional space (e.g., a single loan application, a monthly account statement).
- **Dimensions:** The axes of analysis that provide context to a fact (e.g., ‘Who’, ‘What’, ‘When’, ‘Where’). Dimensions are typically organized into hierarchies, which allow data to be aggregated at various levels of granularity (e.g., a ‘Date’ dimension with a ‘Day’ $\rightarrow$ ‘Month’ $\rightarrow$ ‘Year’ hierarchy).
- **Measures:** Numeric or categorical values that quantify a fact (e.g., ‘TransactionAmount’, ‘DelinquencyStatus’).

For clarity, we assume dimension names are unique within any given schema. The following formal definitions are adapted from the DFM literature.

Definition 3.4.1 (Fact Schema). A **Fact Schema (FS)** is a sextuple defined as:

$$f = (M, A, N, R, O, S), \text{ where:}$$

- M is a set of **measures** (e.g., `TransactionAmount`, `IsDelinquentFlag`).
- A is a set of **dimension attributes**, each with a discrete domain (e.g., `TransactionDate`, `AccountType`, `CustomerAgeGroup`).

- N is a set of **non-dimension attributes** that provide supplementary, descriptive context (e.g., `CustomerFullName`, `LoanPurposeDescription`).
- R is a set of relationships that structure all attributes into a rooted quasi-tree, defining the dimensional and hierarchical pathways. A conceptual dummy attribute a_0 serves as the root, representing the fact itself. The edges (a_0, a_i) define the primary dimensions of the fact.
- $O \subseteq R$ is a set of **optional relationships**, indicating that some attributes may contain null values (e.g., a transaction may not have a 'PromotionCode').
- S is a set of **aggregation statements** of the form (m_j, d_k, Ω_l) , specifying that a measure m_j is aggregable along a dimension d_k using an operator Ω_l (e.g., `(TotalLoanAmount, BranchRegion, SUM)`) [40].

The set of **dimensions** for a FS f , denoted $Dim(f)$, consists of all attributes that are direct children of the conceptual root a_0 . Each dimension can contain one or more hierarchies, which are defined by the paths in the attribute quasi-tree.

For a more high-level, practical representation, a FS can also be expressed concisely.

Definition 3.4.2 (Concise FS [39]). A **FS (FS)** f can be represented as a pair $f = (H, M)$, where:

- H is a finite set of dimension hierarchies (e.g., $H = \{h_{\text{Date}}, h_{\text{Customer}}\}$).
- M is a finite set of measures (e.g., $M = \{\text{OutstandingBalance}, \text{NumberOfDefaults}\}$).

If H contains n hierarchies, the FS is said to be *n-dimensional*. We denote the set of all attributes in f as $Attr(f)$ and the set of all measures as $Meas(f)$.

A dimensional schema (DS) is a collection of one or more FSs that are interconnected through shared dimensions and model a broader business domain. This foundational DFM provides the necessary vocabulary to formally define our prediction-oriented enhancements. The subsequent sections will build upon these definitions to introduce the novel concepts of our framework, including the formalization of inter-schema relationships and the integration of our PO temporal constructs. The corresponding graphical representation of the DFM for our credit risk scenario, showing the interconnected FSs for transactions and loans, will be detailed below.

3.4.1.2 Proposed Symbolic Notation for Prediction-Oriented Dimensional Fact Model Representation

To improve clarity and intuitiveness in representing Fact Schemes (FSs) within the Prediction-Oriented Dimensional Fact Model (PO DFM), we propose a compact symbolic notation. The notation is designed to explicitly distinguish dimension attributes from non-dimensional (descriptive) attributes and to make hierarchical structures within dimensions immediately visible.

The classical DFM literature [39, 40, 62] does not prescribe a standardized symbolic notation that simultaneously exposes keys, hierarchies, descriptive attributes, and contextual dependencies within a single textual schema representation. The notation proposed here fills this gap by providing a uniform, readable representation that remains fully compatible with the original DFM semantics. The following notation conventions are adopted:

- Any set of primary components, such as measures, fact table contents, or collections of dimensions, is enclosed in curly brackets $\{\dots\}$.
- A dimension definition, including its key, dimensional attributes, and hierarchies, is enclosed in square brackets $[\dots]$.

- Hierarchical structures or distinct branches within a dimension are enclosed in angle brackets $\langle \dots \rangle$. Each bracketed group represents a hierarchy or hierarchy branch originating from a common root attribute.
- Non-dimensional (descriptive) attributes are enclosed in vertical bars $|\dots|$ and appear after the dimensional attributes or hierarchies to which they relate. Attributes participating in hierarchies are listed within angle brackets and do not require additional annotation.

This notation enhances the readability of complex multidimensional schemas by enabling immediate identification of attribute roles (key, hierarchical, descriptive, contextual) while remaining formal and compact.

Example 3.4.1: Symbolic Representation of the Financial Credit Risk Schema

We illustrate the proposed notation using the financial credit risk scenario introduced earlier in the chapter. The schema comprises two fact schemes capturing behavioural and repayment-related events, together with shared and contextual dimensions. The concrete data instances are shown in Tables 3.2–3.5.

$$M_{\text{FinancialRisk}} = \{\text{TransactionAmount:Currency}, \text{TotalBalance:Currency}, \\ \text{CreditLimit:Currency}, \text{InstallmentDue:Currency}, \text{InstallmentPaid:Currency}\};$$

$$F_{\text{CreditCard}} = \{\text{TransactionID}, \text{CustomerID}, \text{Date}, \text{TransactionAmount}, \\ \text{TotalBalance}, \text{CreditLimit}\};$$

$$F_{\text{Loan}} = \{\text{CustomerID}, \text{Date}, \text{LoanAmount}, \text{InstallmentDue}, \text{InstallmentPaid}\};$$

$$\begin{aligned}
H_{\text{FinancialRisk}} = \{ & [\text{DateKey}; \text{ActualDate}, \langle \text{Day} \rightarrow \text{Month} \rightarrow \text{Quarter} \rightarrow \text{Year} \rangle], \\
& [\text{CustomerKey}; \text{CustomerID}, \langle \text{Region} \rightarrow \text{Country} \rangle, |\text{Name}|, |\text{BirthDate:Date}|, \\
& |\text{RiskCategory}|], [\text{EconomicIndicatorKey}; (\text{Region}, \text{Date}) \rightarrow \langle \text{IndicatorType} \rangle, \\
& |\text{InflationRate}|, |\text{UnemploymentRate}|] \}.
\end{aligned}$$

The **Customer** and **Date** dimensions are conformed across both fact schemes, enabling drill-across analysis. The attribute **BirthDate** is modelled as a role-playing temporal attribute of the **Date** dimension. The **Economic_Indicators** dimension is contextual and is functionally dependent on **(Region, Date)**, enriching both fact schemes without affecting summarizability.

In the proposed symbolic notation, each dimension is represented as a structured container of attributes. The semicolon separates the dimension key from the remaining components. Attributes listed immediately after the key denote base dimensional attributes. Hierarchies are enclosed in angle brackets $\langle \cdot \rangle$ and represent ordered roll-up paths within the dimension, from the lowest to the highest aggregation level. Multiple hierarchies may be associated with the same dimension by listing multiple angle-bracketed expressions. Non-dimensional descriptive attributes are enclosed in vertical bars and do not participate in aggregation hierarchies.

We emphasize that this symbolic extension is introduced solely to improve the clarity and compactness of schema representations within this study. It does not alter the formal semantics of the DFM or PO DFM, which remain defined by the respective metamodels. The notation is fully compatible with the classical DFM metamodel shown in Figure 3.7 and with the PO DFM metamodel introduced earlier.

Given a fact scheme f , each n -tuple of values drawn from the domains of its n dimensions defines a cell containing one primary fact instance. Formally, a primary fact instance corresponding to $(\alpha_1, \dots, \alpha_n) \in \text{Dom}(d_1) \times \dots \times \text{Dom}(d_n)$ is denoted

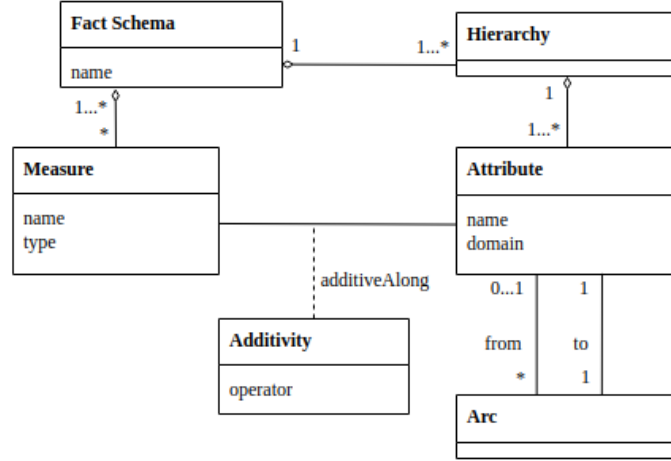


Figure 3.7: Metamodel of the Dimensional Fact Model

by $pf(\alpha_1, \dots, \alpha_n)$, in accordance with the standard DFM definition.

Detailed instance-level tables for the dimensions and fact schemes illustrating this schema are provided earlier in the chapter and are consistent with the two-fact constellation (**CreditCard_Fact** and **Loan_Fact**) used throughout the financial default-risk case study.

3.4.2 Modelling Inter-Schema Relationships: The Fact Constellation

A central challenge in multidimensional analysis is the integration of data originating from multiple, distinct FSs. While a single star schema offers a coherent view over one data domain, meaningful analytical insight, particularly in prediction-oriented tasks, often arises from the synthesis of complementary signals across domains. For instance, assessing customer credit risk depends not only on credit card transaction behaviour but also on loan repayment patterns and macroeconomic context. This section introduces a formal approach to modelling such cross-domain relationships,

leading to a unified structure capable of supporting rich, multivariate analysis across diverse sources.

3.4.2.1 Linking Fact Schemes: From Strict Overlap to General Relationships

The basis for integrating two FSs is **compatibility**, which exists if they share at least one common dimension attribute, allowing them to be semantically related.

Definition 3.4.3 (Facts Compatibility [40]). Two FSs, f' and f'' , are **compatible** if their sets of dimension attributes have a non-empty intersection ($A' \cap A'' \neq \emptyset$) and the domains of these common attributes also have a non-empty intersection.

The simplest and most rigid form of integration is a strict overlap, which is possible only when the two FSs are **strictly compatible**. Strict compatibility requires that the shared dimensions not only exist but have perfectly identical hierarchical structures. This can be formally verified by contracting the quasi-tree representation of each schema's hierarchies onto the set of shared attributes and checking for equality.

Definition 3.4.4 (Strict Facts Compatibility [40]). Two compatible FSs, f' and f'' , are said to be **strictly compatible** if their quasi-trees, when *contracted* on the set of shared dimension attributes $I = A' \cap A''$, are identical:

$$cnt(qt(f'), I) = cnt(qt(f''), I).$$

Here, the *contraction* of a quasi-tree $qt(f)$ on I denotes the reduction of the fact schema structure to the minimal subgraph that preserves only the dimensions and attributes belonging to the shared set I , removing or collapsing all other branches.

When FSs are strictly compatible, their **strict overlap**, denoted $f' \otimes f''$, creates a new, unified FS by merging their measures and intersecting their dimensions and attributes. While theoretically sound, the requirement for strict compatibility is often too restrictive for real-world scenarios, where schemes developed independently across departments and applications or for different purposes rarely have perfectly aligned hierarchies.

In practice, analysts often need to integrate schemes that are only **non-strictly compatible**: they may partially share dimensions or relate via looser correspondences that are not directly expressed in the schema design. In some cases, these relationships may involve more abstract mappings, such as temporal alignment or geographic containment, though formalizing such functional dependencies across schemes remains challenging. To address this, we introduce the **Fact Constellation**, a flexible organizing principle for integrating related, but not fully conformable, FSs.

3.4.2.2 The Fact Constellation

A Fact Constellation provides a formal structure for modelling a collection of Fact Schemes (FSs) and the explicit semantic links between them. These links may represent strict overlaps, non-strict overlaps with conflict-resolution rules, or other relationships that enable coherent cross-schema navigation.

Definition 3.4.5 (Fact Constellation). Let $\mathcal{F} = \{f_1, f_2, \dots, f_k\}$ be a set of $k \geq 2$ FSs. A **Fact Constellation (FC)** is a tuple:

$$FC = (N_{FC}, F_{FC}, D_{FC}, L_{FC}, Cons_{FC}), \text{ where:}$$

- N_{FC} is a unique name identifying the constellation.

- F_{FC} is the set of FSs included in the constellation.
- D_{FC} is the union of all dimensions appearing in any FS in F_{FC} .
- L_{FC} is a set of *link definitions* between FSs in F_{FC} . A link may be defined as:
 - a *strict overlap*, when the linked schemes are strictly compatible;
 - a *non-strict overlap*, where the link includes rules for resolving hierarchical or attribute conflicts;
 - a *functional link*, based on shared entity identifiers or conceptual relationships that enable navigation even with few shared dimensions (e.g., linking a **Promotion** fact to a **Sales** fact via a **CampaignID**).
- $Cons_{FC}$ is a set of *global constraints and policies* governing analysis across the constellation. This is a key extension in our PO DFM and is intentionally *extensible*, allowing additional constraint classes to be introduced as required by specific analytical domains. In particular, $Cons_{FC}$ may formalize rules for:
 - *Domain Alignment*: specifying the effective domain of shared attributes across linked facts, e.g., $Dom_{FC}(a) = Dom_{f'}(a) \cap Dom_{f''}(a)$;
 - *Granularity Reconciliation*: defining how to align measures and dimensions with mismatched levels (e.g., rolling customer-level data to region-level or quarter-level for integration);
 - *Aggregation Semantics*: constraining valid aggregation paths when querying across facts, particularly when navigating through non-strict or functional links, in order to prevent semantic errors such as double counting or temporal misalignment;
 - *Dependency Constraints*: specifying functional dependencies between dimension attributes, expressed as $X \rightarrow Y$, meaning that the values of

attributes Y are functionally determined by the values of attributes X within the constellation;

- *Measure Additivity and Temporal Constraints*: classifying measures according to their additivity properties (additive, semi/non-additive) and defining admissible temporal aggregation windows for predictive and analytical queries;
- *Participation Constraints*: regulating aggregation over many-to-many relationships through weighting or participation rules, ensuring correct contribution of shared entities.

The Fact Constellation model enables integrated queries and analytical reasoning across multiple domains without requiring all data to conform to a single dimensional standard. It is particularly suited to analytical tasks such as credit risk modelling, where transactional behaviour, credit exposure, and macroeconomic context must be combined in a coherent and semantically controlled manner.

Example 3.4.2: Semantic Integration in a Financial Risk Fact Constellation

To illustrate these concepts, consider the **Financial Risk Fact Constellation** that integrates multiple fact schemes involved in credit risk analysis. Formally, we define:

$$\mathcal{FC}_{\text{FinancialRisk}} = (N_{FC}, F_{FC}, D_{FC}, L_{FC}, Cons_{FC}), \text{ where:}$$

- $N_{FC} = \text{"FinancialRisk"}$ is the unique identifier of the constellation.
- $F_{FC} = \{\text{CreditCard_Fact}, \text{Loan_Fact}\}$ is the set of fact schemes.
- $D_{FC} = D_{\text{shared}} \cup D_{\text{contextual}}$ is the union of all dimensions appearing in the

two fact schemes, where:

$$D_{\text{shared}} = \{\text{Customer}, \text{Date}\}, \quad D_{\text{contextual}} = \{\text{Economic_Indicators}\}$$

- L_{FC} contains the following link definitions:
 - A *strict overlap* between **CreditCard_Fact** and **Loan_Fact** on the conformed dimensions $\{\text{Customer}, \text{Date}\}$, enabling drill-across operations at a common *Customer-Month* grain.
 - A *functional link* from both **CreditCard_Fact** and **Loan_Fact** to the contextual dimension **Economic_Indicators**, defined via the determinant $(\text{Region}, \text{Date})$. This link enables navigation even though **Economic_Indicators** is not directly present in either fact scheme.

$$L_{FC} = \left\{ l_1 : \text{CreditCard_Fact} \leftrightarrow \text{Loan_Fact} \text{ (strict overlap on Customer, Date)}, \right. \\ \left. l_2 : \text{Loan_Fact} \leftrightarrow \text{Economic_Indicators} \text{ (functional link via (Region, Date))} \right\}$$

- $Cons_{FC}$ represents global constraints ensuring semantic alignment across the constellation and governing cross-fact analysis.

The fact schemes in F_{FC} operate at different levels of temporal and spatial resolution. Both **CreditCard_Fact** and **Loan_Fact** are defined at the *Customer-Month* grain, whereas the contextual dimension **Economic_Indicators** is available at the *Region-Quarter* level. Granularity reconciliation is therefore defined by the alignment function:

$$G_{\text{align}} : (\text{Customer}, \text{Date}) \rightarrow (\text{Region}, \text{Quarter}),$$

which specifies that measures from transactional and loan facts must be aggre-

gated by region and quarter before integration with macroeconomic indicators.

For example, all customer-level transactions occurring in the **North** region between January 2024 and March 2024 are aggregated into the corresponding **Q1 2024** period, ensuring compatibility with the temporal granularity of the **Economic_Indicators** dimension.

Aggregation constraints in $Cons_{FC}$ define admissible summarization paths and prevent semantic errors such as double counting or temporal misalignment. As an example, consider the derived constellation-level analytical measure **TotalExposure**, defined at query time as:

$$\text{TotalExposure} = \sum_{(\text{Customer}, \text{Date})} \left(\text{LoanAmount}_{\text{Loan_Fact}} + \text{TotalBalance}_{\text{CreditCard_Fact}} \right),$$

subject to the constraint that both measures are first aggregated at the common $(\text{Customer}, \text{Date})$ grain before any higher-level roll-up is applied.

Temporal aggregation constraints further enforce causal consistency in analytical queries. In particular, outcome measures must not be aggregated with contemporaneous or future explanatory variables:

$\text{LoanDefaults}_{t+1}$ may only be analysed against $\text{CreditUtilization}_t$, ensuring that predictors are derived exclusively from non-overlapping historical observation windows.

The contextual dimension **Economic_Indicators** introduces macroeconomic attributes such as **InflationRate** and **UnemploymentRate**, functionally determined by:

$$(\text{Region}, \text{Date}) \rightarrow \{\text{InflationRate}, \text{UnemploymentRate}\}.$$

Through the functional links in L_{FC} , these attributes can be associated with both

fact schemes, enabling enriched analysis that situates customer- and loan-level behavior within the prevailing economic context while preserving summarizability constraints.

Through the explicit specification of $(F_{FC}, D_{FC}, L_{FC}, Cons_{FC})$, the Fact Constellation enforces:

1. *Semantic coherence* through domain alignment;
2. *Analytical comparability* through granularity reconciliation;
3. *Logical integrity* through formally defined aggregation semantics.

As a result, $FC_{\text{FinancialRisk}}$ provides a unified and semantically consistent multidimensional framework that supports coherent cross-domain analysis across transactional, credit, and macroeconomic data.

3.4.2.3 Managing Summarizability and Aggregation Scope

A core challenge when combining data from multiple FSs is maintaining summarizability - the ability to aggregate data correctly and consistently to derive meaningful insights. As established by foundational work [95], summarizability is guaranteed along many-to-one hierarchical relationships but is easily compromised by *many-to-many relationships* or improperly handled joins between facts.

In the financial domain, such relationships arise naturally. Consider the association between **Customer** and **Loan** entities: a single customer may hold multiple loans, and a single loan may be co-signed by multiple customers. This relationship is therefore many-to-many:

$$\text{Customer} \leftrightarrow \text{Loan}.$$

If an analyst attempts to aggregate **OutstandingBalance** or **DefaultCount** directly

across both dimensions without disambiguating this relationship (e.g., by creating a bridge table or defining a participation weight), the results will be distorted - double-counting shared loans or overestimating exposure at the customer level. Similarly, when joining `Loan_Fact` with `CreditCard_Fact`, the same customer may appear multiple times in each fact due to distinct transactional histories, creating another implicit many-to-many mapping through the shared `CustomerID` dimension. In our PO DFM, ensuring valid aggregation across multiple FSs is a primary design objective. The summarizability rules defined in $Cons_{FC}$ within a Fact Constellation explicitly address this challenge by constraining how measures from different fact schemes may be combined and which aggregation paths are admissible when a query navigates across a link $l \in L_{FC}$.

For each link l , the aggregation-related constraints specified in $Cons_{FC}$ can be viewed as inducing an *aggregation constraint set*, denoted by $AggCons(l)$. Formally:

$$AggCons(l) = \left\{ (m, f', f'', g) \mid \begin{array}{l} m \in M_{f'}, \\ g \text{ is an aggregation operator permitted by } Cons_{FC} \\ \text{when combining measures from } f' \text{ with } f'' \end{array} \right\}.$$

Here, m is a measure originating from fact scheme f' , f'' denotes the target fact scheme reached through link l , and g specifies the admissible aggregation operator or grouping semantics.

Consider a risk analyst who wishes to compute a derived constellation-level measure representing customer exposure:

$$\text{TotalExposure} = \text{Loan_Balance} + \text{CreditCard_Balance}.$$

In the absence of explicit constraints, joining `Loan_Fact` and `CreditCard_Fact`

at the transactional level would introduce duplication, as the same customer may appear multiple times within each fact due to distinct transactional histories.

To prevent this, $Cons_{FC}$ specifies a required aggregation grain for the link between the two fact schemes. This constraint can be expressed as:

$$AggLevel(l_{\text{Loan-Credit}}) = \{\text{CustomerID}, \text{Date}\},$$

indicating that both facts must be pre-aggregated at the **Customer--Date** level prior to combination. As a result, each customer contributes at most one aggregated record per time unit, preserving additivity and preventing inflated totals.

When integrating macroeconomic context from the **Economic_Indicators** dimension, summarizability is further protected through explicit granularity reconciliation constraints defined in $Cons_{FC}$. For the corresponding functional link, the admissible aggregation level is given by:

$$AggLevel(l_{\text{Loan-Econ}}) = \{\text{Region}, \text{Quarter}\}.$$

This constraint enforces that loan-level measures are first aggregated by region and quarter before being associated with regional economic indicators. Any attempt to combine measures at incompatible grains, for example, directly linking customer-level loan balances with regional unemployment rates, violates the constraints encoded in $Cons_{FC}$ and is therefore disallowed.

Together, these summarizability and aggregation-scope constraints operationalize analytical correctness within the PO-DFM Framework. They ensure that derived measures remain both semantically and numerically valid, even when queries traverse heterogeneous fact schemes. By explicitly modelling aggregation admissibility within the Fact Constellation, the framework provides a robust foundation for

building integrated prediction-oriented data marts in which analytical precision is preserved across transactional, credit, and macroeconomic domains.

3.4.3 Graphical and Formal Representation of the Dimensional Fact Model for the Credit Risk Scenario

To illustrate the application of the classical Dimensional Fact Model (DFM) as introduced by Golfarelli and Rizzi [39,40], we represent the financial credit risk analytical environment defined in Section 3.3. This environment contains two principal **Fact Schemes (FSs)** - **CreditCard_Fact** and **Loan_Fact** - supported by shared and contextual dimensions **Customer** and **Date**. Each FS is structured as a quasi-tree rooted at the fact node, with connected dimension hierarchies and descriptive attributes following the DFM formalism.

The text-based definitions below correspond to the conceptual structures depicted later in Figure 3.8, which follows the graphical conventions of the standard DFM. For concision, in the worked examples below we use the reduced tuple $f = (M, A, N, R)$ and omit O (optional relationships) and S (aggregation statements); these can be reintroduced without loss of generality.

Fact Scheme: CreditCard_Fact.

$$f_{CC} = (M_{CC}, A_{CC}, N_{CC}, R_{CC}), \text{ where:}$$

- *Name:* $f_{CC} = \text{CreditCard_Fact}$.
- *Grain:* The grain of the fact scheme is implicitly defined by the lowest-level attributes of the dimensions directly connected to the fact in R_{CC} . In this case, each fact instance corresponds to one *Customer-Transaction-Date*.

- *Measures:*

$$M_{CC} = \{\text{TransactionAmount}, \text{PaymentMade}, \text{PaymentDue}, \\ \text{CreditLimit}, \text{TotalBalance}\}.$$

- *Attributes:*

- *Contextual Dimension Attributes (via Functional Links):* **InflationRate**, **UnemploymentRate**. These are quantitative, non-additive attributes belonging to an external contextual dimension and are not measures of the fact scheme; they are analytically associated with fact instances through shared dimensional attributes.

- *Non-dimensional Attributes:* **Risk**, **Name**

$$N_{CC} = \begin{cases} \text{CustomerID} \rightarrow \text{Risk} \\ \text{CustomerID} \rightarrow \text{Name} \end{cases}$$

- *Hierarchies / Relationships:*

$$R_{CC} = \begin{cases} \text{CustomerID} \rightarrow \text{Region} \rightarrow \text{Country} \\ \quad \searrow \text{BirthDate:Date} \rightarrow \text{Month} \rightarrow \text{Quarter} \rightarrow \text{Year} \\ \text{TrxDate:Date} \rightarrow \text{Month} \rightarrow \text{Quarter} \rightarrow \text{Year} \end{cases}$$

The `Economic_Indicators` dimension is modelled as a *separate contextual dimension*, not as part of either fact scheme. It provides macroeconomic context through attributes such as **InflationRate** and **UnemploymentRate**, which are shared across facts via the common attribute pair (**Region**, **Date**).

Formally, the attributes of the **Economic_Indicators** dimension are defined by the following functional dependencies:

$$A_{EI} = \begin{cases} (\text{Region}, \text{Date}) \rightarrow \text{InflationRate} \\ (\text{Region}, \text{Date}) \rightarrow \text{UnemploymentRate} \end{cases}$$

The attributes in A_{EI} are accessible from **CreditCard_Fact** (and analogously from **Loan_Fact**) through the shared dimensions **Region** and **Date**. Since these attributes are invariant within each $(\text{Region}, \text{Date})$ pair and non-additive, they are modelled as dimension attributes rather than fact measures.

In the graphical schema (Figure 3.8), **Economic_Indicators** appear as distinct cross-dimensional nodes connected to both facts through the common attributes **Region** and **Date**.

An interesting modelling consideration arises with the attribute **BirthDate**. Although it is a temporal attribute, it belongs to the **Customer** dimension rather than representing an event time. In the classical DFM formalism, every attribute is strictly associated with a single dimension, which raises the question of how to represent time-related attributes that describe entities rather than events.

Two modelling strategies are typically adopted in the literature:

- *Option 1 – Descriptive Attribute:* **BirthDate** is modelled as a descriptive attribute of the **Customer** dimension:

$$\text{Customer} \rightarrow \text{BirthDate}.$$

In this case, **BirthDate** is treated as static and primarily used for filtering or procedural age computation.

- *Option 2 – Role-Playing Temporal Attribute:* **BirthDate** is modelled as a role-

playing instance of the **Date** dimension:

$$\text{Customer} \rightarrow \text{BirthDate:Date}.$$

This representation enables cohort analysis, age segmentation, and calendar-aligned analytics within the multidimensional structure.

In this example, we adopt the second interpretation, modelling **BirthDate** as a *role-playing instance of the **Date** dimension*. This distinction avoids ambiguity between event time (*TrxDate*) and entity time (*BirthDate*), while preserving a single, shared calendar hierarchy. Accordingly, in the graphical schema (Figure 3.8), the relationship labelled *BirthDate* (role of **Date**) connects the **Customer** and **Date** dimensions.

Fact Scheme: Loan_Fact.

$$f_L = (M_L, A_L, N_L, R_L), \text{ where:}$$

- *Name:* $f_L = \text{Loan_Fact}$
- *Grain:* The grain of the fact scheme is implicitly defined by the lowest-level attributes of the connected dimensions in R_L . In this case, each fact instance corresponds to one *Customer–Loan–Date* snapshot.
- *Measures:*

$$M_L = \{\text{LoanAmount}, \text{InstallmentDue}, \text{InstallmentPaid}\}.$$

- *Hierarchies / Relationships:*

$$R_L = \begin{cases} \text{CustomerID} \rightarrow \text{Region} \rightarrow \text{Country} \\ \text{InstDate:Date} \rightarrow \text{Month} \rightarrow \text{Quarter} \rightarrow \text{Year} \end{cases}$$

- *Attributes:*

- *Contextual Dimension Attributes (Functional Links):* `InflationRate`, `UnemploymentRate`. These are quantitative, non-additive attributes belonging to the external contextual dimension `Economic_Indicators`. They are not measures of the fact scheme and are accessed through shared dimensional attributes.

- *Descriptive Attributes: Status.*

$$N_L = \text{Status}$$

These measures capture both the static loan contract value (`LoanAmount`) and the periodic repayment dynamics (`InstallmentDue`, `InstallmentPaid`). Their difference represents a customer's shortfall or delay for a given month.

The `Loan_Fact` schema models the long-term credit dimension of the analytical environment. Each fact instance represents a snapshot of a customer's loan repayment at a specific point in time. The attribute `Status` provides the categorical evaluation of repayment behaviour - `On-Time`, `Delayed`, or `Default`.

Shared and Contextual Dimensions. Both `CreditCard_Fact` and `Loan_Fact` share a set of conformed dimensions that ensure analytical coherence across the constellation. The two key shared dimensions are:

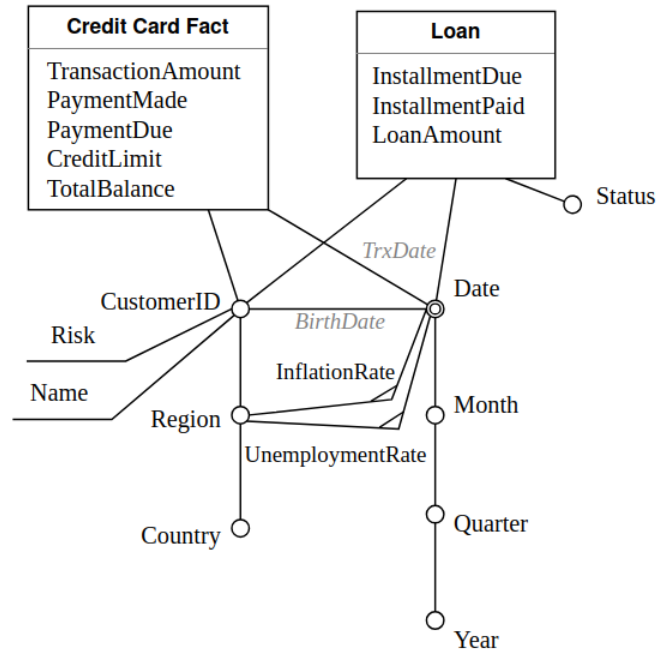


Figure 3.8: Standard DFM Constellation for the Financial Credit Risk and Loan Domain

- **Customer Dimension:** Common to both facts, it provides the identification and segmentation context for each client. Its hierarchy is defined as:

$$\text{CustomerID} \rightarrow \text{Region} \rightarrow \text{Country},$$

and it includes non-dimensional descriptive attributes **Name**, **BirthDate** (role-playing *BirthDate* temporal attribute of **Date**), and **Risk**. This shared dimension enables consistent analysis of customer-level behaviour across transactional and loan perspectives.

- **Date Dimension:** Provides the temporal axis common to all facts, defined

by the hierarchy:

$$\text{Date} \rightarrow \text{Month} \rightarrow \text{Quarter} \rightarrow \text{Year}.$$

It represents both the transaction execution date (*TrxDate*) and the loan instalment observation date.

In addition, both facts reference a contextual dimension *Economic_Indicators*, with attributes such as *InflationRate* and *UnemploymentRate*. This dimension is linked to the constellation via the shared pair (*Region, Date*), formally defined by the functional dependencies:

$$(\text{Region}, \text{Date}) \rightarrow \text{InflationRate}, \quad (\text{Region}, \text{Date}) \rightarrow \text{UnemploymentRate}.$$

The inclusion of *Economic_Indicators* as a contextual dimension extends the analytical space beyond customer behaviour, allowing the integration of external, region-level economic context into both the transactional and credit domains of the model. In the graphical schema (Figure 3.8), it appears as a distinct dimension node connected to both facts through the shared attributes *Region* and *Date*.

Fact Constellation. The two Fact Schemes *CreditCard_Fact* and *Loan_Fact* are integrated into a single Fact Constellation through shared conformed and contextual dimensions. Formally, we define:

$$\mathcal{FC}_{\text{FinancialRisk}} = (N_{FC}, F_{FC}, D_{FC}, L_{FC}, Cons_{FC}), \quad F_{FC} = \{f_{CC}, f_L\}.$$

The shared dimensions (*Customer, Date*) ensure conformance of analysis across both facts, allowing consistent aggregation and comparison of customer-level and time-based data. The *Economic_Indicators*, linked via (*Region, Date*), provides

macroeconomic variables, `InflationRate` and `UnemploymentRate`, enabling enrichment of each fact with regional and temporal economic context.

This constellation supports *drill-across* operations between transactional and loan data, enabling unified financial summaries and cross-domain indicators, for example, correlating customer spending patterns with subsequent loan repayment behaviour.

At this stage, all facts are treated as coexisting within the same analytical space, and relationships between them are purely structural. Temporal sequencing and predictive interpretation will be introduced later through the extended semantics of the Prediction-Oriented DFM (PO DFM) in Section 3.5.

This conceptual structure thus represents the baseline multidimensional schema, a standard DFM constellation according to Golfarelli and Rizzi, which serves as the formal foundation for the temporal and prediction-oriented extensions implemented in the PO framework.

3.5 Formalizing the Temporal Framework

This section develops the theoretical foundations of the proposed temporal framework for prediction-oriented data analysis. As argued in the motivating example, the transition from descriptive Business Intelligence (BI) to Machine Learning (ML) requires a deeper and more disciplined treatment of time. Traditional multidimensional data warehouses treat time as a standard dimension, suitable for roll-ups, drill-downs, and historical summaries, but they do not provide mechanisms to enforce temporal precedence, manage data latency, or structure the relationship between explanatory features and future outcomes. These limitations make classical DW/OLAP models insufficient for constructing prediction-ready analytical datasets.

To address this gap, we introduce a temporal framework based on three semantically and functionally distinct windows. Unlike descriptive temporal models or classical moving-window aggregation schemes, our framework incorporates temporal roles and data-availability constraints directly into the data model, enabling the design of prediction-oriented data marts with explicit safeguards against information loss. The result is a principled, domain-agnostic structure that preserves analytical validity and supports predictive decision-making across a wide range of applications.

The proposed approach generalises and significantly extends the Moving-Window Temporal Aggregation paradigm. MWTA, as cited in time-series literature [57], defines a single backward-looking interval: $\omega(t) = [t - \Delta, t]$, from which descriptive statistics are derived. MWTA is effective for localized temporal summaries, but it lacks (i) a mechanism to distinguish between features and outcomes, (ii) a representation of latency or data-maturation delays, and (iii) support for temporal directionality. Therefore, MWTA is descriptive, not predictive.

Our approach also relates to interval-based temporal multidimensional models as T+MultiDim [3], which augment classical DFMs with validity intervals. However,

the goal of T+MultiDim is to describe the evolution of facts and dimensions over time, whereas our model treats temporal analytical roles, not temporal evolution. The shift from “what was true when” to “what can be used to predict what” represents a conceptual and operational departure from prior temporal OLAP research. In what follows, we use the term *PO DFM* to denote the structural, multidimensional schema extended with temporal window semantics, and *PO Framework* to denote the broader methodological procedure integrating modelling, window configuration, and ABT construction.

The proposed temporal formalism organizes data around a central anchor point and a structured set of windows, each representing a distinct analytical role. This structure transforms the data warehouse from a passive repository of historical facts into a temporally disciplined environment capable of supporting predictive modelling and Prediction-Oriented OLAP (PO OLAP).

The framework addresses this gap by defining a coordinated set of three windows, each with a specific and non-overlapping role in the prediction-oriented workflow: feature generation, latency management, and outcome labelling. This structure transforms the data warehouse from a passive repository of facts into an active, organized environment for generating predictive insights.

3.5.1 Formal Definition of the Temporal Windows

We formalise a temporal window configuration as a tuple

$$\Omega(t_0) = \{\omega_{\text{OW}}, \omega_{\text{WW}}, \omega_{\text{PW}}\},$$

where each ω_i is defined as a contiguous time interval on a one-dimensional temporal axis. All temporal semantics are defined relative to a single anchor point t_0 , representing the conceptual *present* of analysis. At time t_0 , all historical information

is assumed to be known, while future information is unknown.

Each window is modelled as a half-open interval:

$$\omega_i = [s_i, e_i), \quad s_i < e_i,$$

with s_i and e_i denoting its start and end times, respectively. Unlike MWTA's single-window formulation, the tuple $\Omega(t_0)$ explicitly distinguishes between three semantically different temporal intervals:

- the *Observation Window* (ω_{OW}), covering the historical period used for feature extraction;
- the *Waiting Window* (ω_{WW}), representing a latency interval during which information may be unavailable or intentionally excluded;
- the *Prediction Window* (ω_{PW}), defining the future interval over which outcomes are observed.

To reason about temporal ordering, we introduce a notion of *complete precedence* between intervals. Given two windows $\omega_a = [s_a, e_a)$ and $\omega_b = [s_b, e_b)$, we define:

$$\omega_a < \omega_b \iff e_a \leq s_b.$$

This relation ensures that the two intervals are disjoint and that ω_a entirely precedes ω_b on the temporal axis.

Under this definition, the temporal window configuration is required to satisfy the strict ordering constraint:

$$\omega_{\text{OW}} < \omega_{\text{WW}} < \omega_{\text{PW}},$$

which enforces a complete temporal separation between historical observations, latency effects, and outcome realisation. As a consequence, no event contributing to feature construction can overlap with either the waiting period or the outcome window.

This ordering induces an explicit temporal directionality in analytical queries:

$$\omega_{\text{OW}} \Rightarrow \omega_{\text{PW}},$$

ensuring that predictors are derived exclusively from past information and are never contaminated by contemporaneous or future outcomes.

By explicitly modelling temporal windows as ordered intervals, the framework transforms the data warehouse from a passive repository of historical facts into a structured analytical environment that supports prediction-oriented feature construction under strict temporal constraints.

The Role and Nuances of the Observation Window (OW). The Observation Window is the historical period ending immediately before the anchor point t_0 . Its purpose is to serve as the source of all data used to engineer predictive features. The richness of these features is paramount for model performance. Within the OW, analysts can derive a wide array of metrics, such as:

- **Simple aggregations** (sums, averages, counts),
for example `total_spending_last_6_months`;
- **Trends and momentum measures**,
for example `is_debt_increasing_or_decreasing`;
- **Recency and frequency metrics**, capturing time since last activity or rate of events,

for example `days_since_last_payment` or `monthly_transaction_frequency`.

The choice of the Observation Window length, Δt_{ow} , is a critical modelling decision. A longer OW captures more historical data and long-term trends but may dilute the signal from more recent, impactful events. Conversely, a shorter OW is more sensitive to recent behaviour but may fail to capture broader contextual patterns.

Feature generation within the Observation Window is not limited to simple aggregations computed over the entire period. More sophisticated techniques, drawn from event stream processing, can be employed to capture complex temporal dynamics. These techniques partition the OW into smaller, well-defined sub-windows to extract more granular features. The primary windowing strategies are as follows:

- **Tumbling Windows:** Fixed-size, non-overlapping, and sequential partitions of the OW. For example, to model seasonality within a one-year OW, features can be computed for each month using twelve consecutive tumbling windows of one-month duration. This enables the model to distinguish between behavioural patterns observed in different periods of the year.
- **Hopping Windows:** Fixed-size, overlapping windows that advance by a fixed hop interval. This strategy is well suited for computing rolling metrics. For instance, within a 90-day OW, a `30_day_rolling_average_spend` feature can be updated every seven days, providing smoother and more responsive trend indicators than tumbling windows. A **sliding window** represents the limiting case of a hopping window with a minimal hop interval (e.g., one day or one event).
- **Session Windows:** Variable-length windows that group events based on activity patterns. A session window aggregates events for a given entity (e.g.,

a customer) and closes after a predefined period of inactivity. This strategy is particularly effective for modelling engagement behaviour. Within the OW, it enables the extraction of features such as `average_session_duration`, `number_of_actions_per_session`, or `time_between_sessions`, which cannot be captured using fixed-length windows.

Together, these advanced windowing strategies transform the Observation Window from a simple container of historical data into a structured feature-generation environment, supporting the construction of temporally disciplined and highly informative predictive variables.

The Critical Role of the Waiting Window (WW). The Waiting Window is the temporal buffer, that immediately follows the anchor point:

$$\omega_{\text{WW}} = [t_0, t_0 + \Delta t_{\text{WW}}),$$

representing a period in which outcomes may have occurred but cannot yet be known or recorded due to latency, settling times, or reporting delays. From an analytical standpoint, ω_{WW} enforces strict temporal separation between features and outcomes. The WW accounts for:

- **Data Latency:** Delays between a business event occurring and it being recorded and available for analysis.
- **Outcome Maturation:** Time required for an outcome to be definitively recorded (e.g., default classification).
- **Temporal Separation:** By excluding WW data from input aggregates, we ensure a clear temporal separation between observed behaviour and subsequent outcomes, maintaining the correct chronological sequence in the data.

Defining the Target in the Prediction Window (PW) The Prediction Window is the period following the WW. It is the timeframe within which the target outcome (e.g., customer default) is monitored. The PW may be used for the definition of the ground truth for supervised learning models, which could be suitably applied after or concurrently with the temporal Prediction-Oriented OLAP task. For instance, a target variable `hasDefaultedIn90Days` is true if and only if a default is recorded within the customer's PW.

The Prediction Window is the outcome interval:

$$\omega_{PW} = [t_0 + \Delta t_{WW}, t_0 + \Delta t_{WW} + \Delta t_{PW}),$$

during which the target variable is observed. This window operationalizes the supervised learning ground truth or the validation period in Prediction-Oriented OLAP tasks.

The length ω_{PW} is driven by the business context - short for tactical decisions, long for strategic forecasts.

Let ω_{OW} , ω_{WW} , and ω_{PW} denote the intervals of the three windows, as depicted in Figure 3.9:

- **Observation Window (OW):** $[t_0 - \Delta t_{OW}, t_0]$;
- **Waiting Window (WW):** $[t_0, t_0 + \Delta t_{WW}]$;
- **Prediction Window (PW):** $[t_0 + \Delta t_{WW}, t_0 + \Delta t_{WW} + \Delta t_{PW}]$.

Together, these windows enforce a strict temporal separation between feature extraction, latency effects, and outcome observation. This separation guarantees that target variables are never influenced by contemporaneous or future information relative to the observation period, ensuring temporal validity in predictive analysis.

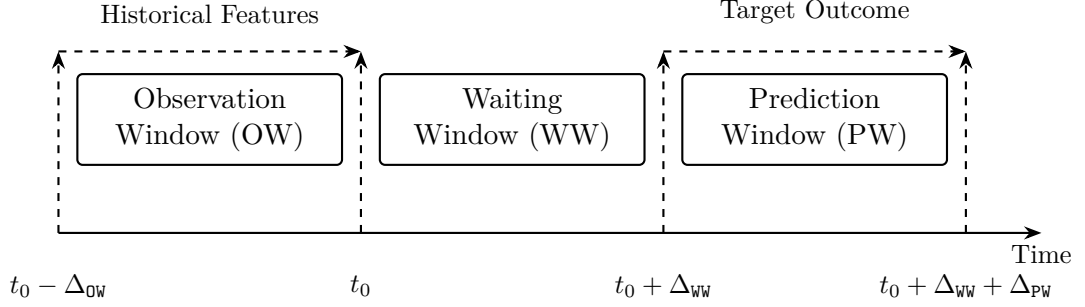


Figure 3.9: Temporal Windowing Structure: Observation, Waiting, and Prediction

The operational manipulation of these temporal windows (slicing, shifting, or combining them during analytical queries) is addressed separately through a set of Prediction-Oriented OLAP operators, which are formally introduced in the next subsection and instantiated in the implementation chapter.

3.5.2 Prediction-Oriented OLAP Operators

Prediction-Oriented OLAP (PO OLAP) extends classical OLAP by introducing explicit temporal semantics that govern how historical observations are related to future outcomes. Within this framework, OLAP operators are defined over the temporal window configuration $\Omega(t_0)$ and are designed to support analytical tasks that connect the Observation Window (OW) to the Prediction Window (PW) while strictly respecting temporal constraints.

In particular, PO OLAP operators must ensure that: (i) feature extraction is confined to ω_{OW} , (ii) outcome variables are observed only in ω_{PW} , and (iii) no operator allows information leakage across window boundaries.

We group the supported operators into three conceptual classes: window-restriction operators, window-transformation operators, and cross-window operators. Among these, cross-window operators play a central role, as they explicitly link OW-derived features with PW-defined targets.

Window-Restriction Operators. These operators limit the temporal scope of analysis to a specific window without altering its structure. A fundamental example is:

$$\text{SLICE}(\omega_i),$$

which restricts fact data to the interval defined by window $\omega_i \in \Omega(t_0)$. For instance, $\text{SLICE}(\omega_{\text{OW}})$ selects only historical records eligible for feature computation, while $\text{SLICE}(\omega_{\text{PW}})$ isolates outcome observations.

Window-Transformation Operators. Window-transformation operators modify the temporal positioning or extent of windows, enabling rolling and time-shifted analyses. A representative operator is:

$$\text{SHIFT}(k),$$

which translates all window boundaries by a temporal offset k . This operator supports longitudinal evaluation scenarios, such as retraining a predictive model at successive anchor times $t_0 + k$, while preserving the relative ordering and separation of OW, WW, and PW.

Another transformation operator is:

$$\text{ROLLUP}(\Delta),$$

which aggregates measures across multiple adjacent windows of duration Δ . This operator enables temporal smoothing and multi-horizon analysis, for example aggregating monthly OW features into quarterly summaries.

Cross-Window Operators (Linking OW and PW). Cross-window operators explicitly relate information extracted from the Observation Window to outcomes observed in the Prediction Window. These operators are the defining mechanism that connects historical features to future targets in PO OLAP.

The primary operator in this class is:

$$\text{JOIN}(\omega_{\text{OW}}, \omega_{\text{PW}}),$$

which associates feature vectors computed over ω_{OW} with target variables defined over ω_{PW} , under the temporal implication:

$$\omega_{\text{OW}} \Rightarrow \omega_{\text{PW}}.$$

This operator does not perform a conventional relational join on overlapping records; instead, it enforces a semantic association between past observations and future outcomes for the same analytical entity (e.g., a customer).

Consider a credit risk analysis task in which customer spending behaviour is summarised over the Observation Window using measures such as average transaction amount and credit utilisation. The operator $\text{JOIN}(\omega_{\text{OW}}, \omega_{\text{PW}})$ associates these features with a binary outcome indicating whether the customer defaulted within the subsequent Prediction Window. The resulting dataset can be used either for supervised model training or for prediction-oriented analytical exploration.

By explicitly distinguishing cross-window operators from classical in-window aggregations, PO OLAP ensures that analytical workflows respect temporal causality and prevent leakage of future information into historical feature construction.

The operators defined in this section provide a conceptual and implementation-neutral interface for prediction-oriented analysis. Their concrete realization in SQL-

based data warehouses and Python-based analytical pipelines is discussed in the implementation chapter, where we demonstrate how PO OLAP operators can be instantiated over real datasets without violating the formal temporal constraints defined in Section 3.5.1.

3.5.3 Window Configurations and Dynamics

The power of this framework lies in its flexibility. The window lengths (Δt_{ow} , Δt_{ww} , Δt_{pw}) and the anchor point (t_0) can be configured statically or dynamically, as depicted below.

Window Lengths (Fixed vs. Variable)

- **Fixed-Length Windows:** Predefined intervals (e.g., 180, 30, 90 days). Ideal for reproducible training data in regulatory or operational settings.
- **Variable-Length Windows:** Window lengths depend on entity-specific timestamps (e.g., customer tenure from `signup_date` to t_0). Offers greater flexibility but requires careful modelling.

A special case is $\Delta t_{\text{ww}} = 0$, appropriate only when latency is negligible (e.g., high-frequency trading).

Window Position (Fixed vs. Sliding)

- **Fixed Position:** t_0 is a single, constant point (e.g., '2025-01-01'). Suitable for one-time batch scoring or static analysis.
- **Sliding Position:** t_0 moves to generate multiple OW-WW-PW triplets for each entity:

- *Periodic Sliding*: t_0 advances in fixed intervals (e.g., monthly). Supports training over extended histories with seasonality.
- *Event-Driven Sliding*: t_0 is triggered by events (e.g., loan applications, credit limit changes).

While powerful, these configurations involve trade-offs. Sliding windows, especially periodic ones, can yield massive datasets. Variable-length windows introduce complexity in modelling pipelines. Ultimately, configurations must balance analytical precision against scalability and model feasibility. Taken together, these configuration modes determine how $\Omega(t_0)$ is instantiated for a given analytical task and provide the flexibility required to support both retrospective analysis and large-scale model training pipelines.

3.5.4 Application to Default Risk Prediction Example: Temporal Data Structuring

To illustrate these principles with a concrete case, we use a simplified dataset containing `Loan_Fact`, `CreditCard_Fact`, `Customer`, and `Economic_Indicators` tables. These correspond respectively to loan repayment events, credit-card transactions, customer attributes, and macroeconomic context. Each table becomes part of the star schema introduced in Section 3.4.3, where the `Customer` and `Date` dimensions are conformed and `Economic_Indicators` acts as a contextual dimension linked via `(Region, Date)`. The raw operational data are thus restructured into a coherent constellation of fact schemes forming the foundation for the PO Framework.

The dataset can be formally organized following the transformation from raw operational tables $\rightarrow$ classical Star Schema (DFM) $\rightarrow$ extended Prediction-Oriented Dimensional Fact Model (PO DFM).

Step 1: Classical DFM Schema. Two primary Fact Schemes (FSs) are identified:

- **CreditCard_Fact** - captures transactional behaviour at the granularity of (CustomerID, TransactionID, Date), with measures such as TransactionAmount, PaymentDue, PaymentMade, and TotalBalance.
- **Loan_Fact** - records instalment-level repayment activity at the granularity of (CustomerID, Date), with measures InstallmentDue, InstallmentPaid, and LoanAmount.

These facts share three conformed dimensions:

- **Customer** (CustomerID, Name, BirthDate, Region, RiskCategory);
- **Date** (Date, Month, Quarter, Year);
- **Economic_Indicators** (Date, Region, InflationRate, UnemploymentRate)
- a contextual dimension shared across both facts through (Region, Date).

The resulting constellation supports traditional *drill-across* operations, enabling cross-domain queries, comparing credit utilisation trends with subsequent loan repayment behaviour.

Step 2: Temporal Extension through PO DFM. The same constellation is then extended along a temporal axis defined by three analytical windows:

- **Observation Window (OW):** past behavioural data used to compute explanatory features;
- **Waiting Window (WW):** an intermediate latency period excluding data used for prediction;

- **Prediction Window (PW):** future events providing target outcomes.

We set the anchor time $t_0 = 2024-01-01$, with $\Delta t_{OW} = 180$ days, $\Delta t_{WW} = 30$ days, and $\Delta t_{PW} = 90$ days.

$$\omega_{OW} = [2023-07-05, 2023-12-31),$$

$$\omega_{WW} = [2024-01-01, 2024-01-31),$$

$$\omega_{PW} = [2024-02-01, 2024-05-01).$$

We apply the three-window semantics defined in Section 3.5.1, **Loan_Fact** is assigned to ω_{PW} and **CreditCard_Fact** to ω_{OW} , and contextual dimensions to the roles described previously (for example, **Economic_Indicators**).

Let each dataset participating in a Fact Constellation (either a fact scheme or a contextual dimension) be assigned a temporal role through a *binding function*:

$$\psi : \mathcal{D}_{FC} \rightarrow \{OW, WW, PW, \text{context}\},$$

where $\mathcal{D}_{FC}$ denotes the set of datasets associated with the constellation. The function ψ specifies the temporal window in which a given dataset is permitted to contribute information for a specific analytical task.

This binding enforces temporal validity by ensuring that each dataset provides data only within its semantically admissible window. Datasets bound to **OW** may contribute explanatory features, datasets bound to **PW** may contribute outcome variables, datasets bound to **WW** are excluded from both feature and target construction, and datasets bound to **context** may enrich multiple windows without participating in temporal precedence constraints.

In the default-risk prediction scenario considered here, the binding is defined as

follows:

$$\psi(\text{CreditCard_Fact}) = \text{OW}, \psi(\text{Loan_Fact}) = \text{PW},$$

$$\psi(\text{Economic_Indicators}) = \text{context}.$$

The value **context** denotes a special temporal role reserved for contextual dimensions whose attributes are exogenous to the behavioural process being analysed. Datasets assigned to the **context** role do not define targets or behavioural features; instead, they provide auxiliary information that may be associated with multiple windows.

Formally, a dataset bound to **context** is not subject to temporal precedence constraints between OW and PW. Its attributes may be joined to data in ω_{OW} or ω_{PW} provided that their own internal temporal dependencies (e.g., $(\text{Region}, \text{Date}) \rightarrow \text{InflationRate}$) are respected.

The binding function ψ is not fixed globally; it may be redefined across analytical tasks or prediction problems. This flexibility allows the same Fact Constellation to support different prediction-oriented analyses (e.g., churn, fraud, or demand forecasting) by reallocating datasets to different temporal roles while preserving the underlying schema.

Table 3.7 specifies the temporal scope of each dataset according to the binding function ψ defined earlier. These assignments determine which facts contribute features and which contribute outcomes. The labels "partial" and "pending" indicate datasets whose records may temporally overlap the corresponding window boundaries but are not used for feature extraction or target labelling in that window.

Contextual dimensions, such as **Economic_Indicators** in this example, may validly span multiple temporal windows. As datasets bound to the **context** role through the binding function ψ , they are exogenous and non-behavioural, and their attributes are invariant within each $(\text{Region}, \text{Date})$ pair. Consequently, contextual dimensions do not participate in temporal precedence constraints between the Ob-

Entity	OW	WW	PW	Role
CreditCard_Fact	+	partial	-	Feature source
Loan_Fact	-	pending	+	Outcome target
Date	+	+	+	Temporal axis
Customer	+	+	+	Shared entity
Economic_Indicators	+	-	+	Contextual dimension

Table 3.7: Temporal Windows Membership Summary

servation Window (OW) and the Prediction Window (PW); instead, they enrich both windows with auxiliary information that supports interpretation without affecting causal ordering.

The issues illustrated in Figure 3.6 (temporal ambiguity, lack of predictive framing, and semantic misalignment) are directly addressed through the PO DFM Framework and the construction of an Analytical Base Table (ABT). The ABT represents a structured, entity-level analytical dataset that integrates explanatory features and outcome labels within explicitly defined temporal boundaries. By anchoring feature variables and target outcomes in the Observation Window (OW), Waiting Window (WW), and Prediction Window (PW), the framework enforces strict *temporal precedence* between inputs and outputs, thereby eliminating violations of the implemented temporal checks and ensuring analytical consistency.

Unlike traditional OLAP aggregations, which are retrospective and often agnostic to event ordering, ABT construction in the PO DFM relies on explicit **temporal ordering constraints** that govern how data may flow across time. These constraints are operationalized through the prediction-oriented OLAP operators introduced in Section 3.5.2, in particular through the OW–PW linkage induced by the operator chain $\text{SLICE}(\omega_{\text{OW}})$, $\text{LABEL}(\omega_{\text{PW}})$, and $\text{JOIN}(\omega_{\text{OW}}, \omega_{\text{PW}})$. Together, these operators ensure that only information available prior to the anchor time (t_0) contributes

to feature computation, while all target variables are derived strictly after the latency period defined by the WW. The resulting analytical process enforces a well-defined ordering of events and data dependencies that is consistent with real-world information availability.

Furthermore, entity-centric integration across multiple data sources, combined with fine-grained, event-level aggregation techniques (e.g., rolling transaction averages or session-based spending summaries), produces features that are both semantically coherent and temporally aligned. This structured approach transforms raw, loosely coupled OLAP data into a temporally valid, prediction-ready dataset. Table 3.8 contrasts the traditional OLAP paradigm with ABT-oriented data construction, highlighting how explicit temporal structuring enables analytical reliability and supports operational deployment.

Dimension	OLAP Drill-Across (Traditional)	PO ABT Construction (Proposed)
Temporal Granularity	Implicit or coarse-grained (e.g., monthly)	Explicitly defined OW, WW, PW
Time Scoping	Aggregates may mix past, present, or future data	Strict forward-only scoping avoids leakage
Semantics	Aggregation logic is source-specific and ad hoc	Features are semantically aligned across sources
Predictive Framing	Post hoc descriptive; no clear target window	Forward-facing, target-aligned (e.g., future default)
Data Integration	Simple joins, minimal harmonization	Entity-centric, temporally scoped, cross-domain fusion
Model Readiness	Not directly usable for ML	ABT is modelling-ready, with clear labels
Operational Use	Limited to dashboards or KPIs	Enables forecasting, alerts, and prescriptive action

Table 3.8: Comparison of OLAP Drill-Across vs. PO ABT Construction

As defined in Section 3.5.3, we instantiate the temporal windows with:

$$(\Delta t_{\text{OW}}, \Delta t_{\text{WW}}, \Delta t_{\text{PW}}) = (180, 30, 90)\text{days},$$

relative to the anchor time $t_0 = 2024-01-01$. The temporal assignment of each dataset follows the window semantics formalised in Section 3.5.1 and the binding function ψ . In this application, $\psi(\text{CreditCard_Fact}) = \text{OW}$, $\psi(\text{Loan_Fact}) = \text{PW}$, and $\psi(\text{Economic_Indicators}) = \text{context}$, allowing the contextual dimension to enrich both windows without violating temporal precedence.

We now instantiate the PO DFM by constructing an Analytical Base Table that is fully consistent with the temporal window semantics and Prediction-Oriented OLAP operators defined above.

Example 3.5.1: Generating an ABT for Default Risk

The available data include instalment-level records in `Loan_Fact` (CustomerID, Date, LoanAmount, InstallmentDue, InstallmentPaid, Status), transaction-level entries in `CreditCard_Fact` (TransactionID, CustomerID, Date, TotalBalance, CreditLimit, PaymentDue, PaymentMade, TransactionAmount), customer descriptors (Name, BirthDate, Region, RiskCategory), and macroeconomic indicators (InflationRate, UnemploymentRate by Region and Date). These will be mapped to the appropriate windows as defined by the PO Framework.

We return to the financial institution aiming to identify customers at risk of default. The goal is to construct a dataset where each row represents one customer at a specific point in time, containing features calculated from their recent past (OW) and a target label indicating if they defaulted in the near future (PW).

Parameter Definition

The analyst defines the temporal parameters based on the business problem and data characteristics. For this scenario:

- **Anchor Point (t_0):** "2024-01-01". The analysis will be from the perspec-

tive of this date.

- **Observation Window (Δt_{OW}):** 180 days. Features are based on behaviour from July 5, 2023, up to Jan 1, 2024.
- **Waiting Window (Δt_{WW}):** 30 days. A buffer for data latency, covering January 2024.
- **Prediction Window (Δt_{PW}):** 90 days. Default events are monitored from Feb 1, 2024, to May 1, 2024.

These temporal allocations follow directly from the window definitions introduced earlier.

ABT construction at the logical level. Given a constellation $\mathcal{FC}$ and a temporal configuration ΔT , an ABT for entity e at anchor time t_0 is defined as:

$$\text{ABT}(e, t_0) = \text{JOIN}\left(\text{AGGREGATE}(\text{SLICE}(f_{OW}, \omega_{OW})), \text{LABEL}(\text{SLICE}(f_{PW}, \omega_{PW}))\right),$$

where f_{OW} is any fact scheme with $\psi(f_{OW}) = OW$ and f_{PW} any fact scheme with $\psi(f_{PW}) = PW$. The Waiting Window ω_{WW} is excluded by construction.

The implementation and validation of this construction are summarized in Section 3.7. Detailed Python and SQL realizations are provided in Appendix B.

The novelty of this formalization lies in bridging the gap between temporal data warehousing and predictive analytics. By redefining the DFM in terms of temporally coordinated analytical windows, the framework introduces a new class of PO DFMs. Unlike existing temporal OLAP models, which focus on describing temporal evolution, the PO Framework explicitly formalizes temporal precedence, data latency, and predictive causality as structural elements of the data model. This represents

a fundamental shift from descriptive to prescriptive multidimensional analysis.

3.6 Defining the PO DFM Framework

3.6.1 Graphical Representation of the PO DFM

The **Prediction-Oriented Dimensional Fact Model (PO DFM)** extends the classical DFM of Golfarelli and Rizzi [39, 40] by enriching the schema with explicit temporal and predictive semantics. Whereas the standard DFM provides a static representation of fact schemes and conformed dimensions, the PO DFM embeds temporal roles, predictive directionality, and PO OLAP operator semantics directly into the graphical notation.

This upgraded notation is motivated by the temporal formalism introduced in Section 3.5.1 and by the Prediction-Oriented OLAP operators defined in Section 3.5.2. Accordingly, the graphical model is no longer a static constellation: it becomes a temporally stratified analytical blueprint that visually encodes the information constraints under which prediction-oriented pipelines operate.

Temporal stratification of fact schemes. In contrast to classical multidimensional modelling, where facts exist without explicit temporal semantics, the PO DFM situates fact schemes along a *three-window temporal axis*:

$$\omega_{\text{OW}} \rightarrow \omega_{\text{WW}} \rightarrow \omega_{\text{PW}},$$

which reflects enforced temporal precedence between historical observation, latency handling, and outcome realization.

This temporal ordering is operationalized through the prediction-oriented OLAP operators formally introduced in Section 3.5.2. Each operator is explicitly scoped to one or more temporal windows defined in Section 3.5.1. The canonical prediction-

oriented execution pattern is:

$$\text{SLICE}(\omega_{\text{OW}}) \rightarrow \text{AGGREGATE}(\omega_{\text{OW}}) \rightarrow \text{LABEL}(\omega_{\text{PW}}) \rightarrow \text{JOIN}(\omega_{\text{OW}}, \omega_{\text{PW}}).$$

As a result, feature extraction, aggregation, and target labelling are performed only within their semantically admissible temporal intervals.

The graphical PO DFM thus serves as the structural counterpart of the formal PO OLAP semantics, visually encoding both temporal roles and predictive directionality.

Scope of the graphical notation. To express the alignment between the temporal framework and the predictive semantics of the PO DFM, we adopt a concise graphical notation. Rather than reproducing the full classical DFM notation, the representation used in this dissertation focuses on a reduced, application-oriented subset that is sufficient for modelling the three-window predictive constellation employed throughout the financial default-risk case study.

Graphical elements of the PO DFM. The simplified notation consists of the following elements:

1. **Horizontal analytical window bands** The diagram is divided into three horizontal bands corresponding to the Observation Window (OW), Waiting Window (WW), and Prediction Window (PW). Each fact scheme is drawn inside the band representing the window in which it participates:
 - OW band: contains fact schemes contributing historical features.
 - PW band: contains fact schemes providing outcome variables.
 - WW band: represents a temporal control interval used for latency handling; it is typically empty and does not contain fact schemes.

2. **Border styles for fact schemes** Fact schemes are distinguished by border style, independently of band placement:

- **Solid borders:** fact schemes used for feature extraction in the Observation Window.
- **Dashed borders:** fact schemes supplying target outcomes in the Prediction Window.

This encoding makes the analytical role of each fact visually explicit.

3. **Predictive dependency connector** The only inter-fact relationship explicitly represented is the *predictive dependency* from an OW fact to a PW fact. It is drawn as a double-line arrow from the OW fact scheme to the PW fact scheme, indicating that the former supplies predictive features used to estimate outcomes recorded in the latter.

4. **Vertical alignment of shared dimensions** Dimensions participating in both OW and PW (e.g., `Customer`, `Date`) are drawn vertically so that their nodes appear aligned across the relevant bands. Dimensions participating in only one window appear solely within the corresponding band. Dimensions that are not window-specific remain outside the temporal bands.

5. **Window-specific temporal connectors** When a temporal dimension participates in multiple analytical windows, its participation is shown through multiple, window-specific connectors. Each connector corresponds to a distinct temporal role and is annotated accordingly:

- **Dot-dashed connector:** temporal role used in the Observation Window (e.g., `TrxDate[OW]`).

- **Dotted connector:** temporal role associated with the Waiting Window (used only to mark latency boundaries; optional).
- **Dashed connector:** temporal role used in the Prediction Window (e.g., `LoanDate[PW]`).

Multiple connectors may originate from the same temporal dimension node, indicating that different physical timestamp attributes or semantic interpretations of time are used across windows.

6. **Inline temporal tags (optional)** Temporal membership may optionally be indicated using inline tags such as `[OW]` or `[PW]` next to fact names, although band placement and border style are sufficient to determine window participation.

This notation directly reflects the semantics of the SQL-based ABT construction presented later: all computations on `CreditCard_Fact` occur in the Observation Window, all computations on `Loan_Fact` occur in the Prediction Window, and the only inter-fact relationship is the predictive dependency

$$\text{CreditCard_Fact}[\text{OW}] \Rightarrow \text{Loan_Fact}[\text{PW}].$$

Role-annotated temporal dimensions. In addition to assigning temporal roles to fact schemes, the PO DFM explicitly represents the semantic roles played by the temporal dimension that defines window boundaries. Rather than introducing multiple temporal dimensions, the PO DFM adopts a role-annotation mechanism consistent with the classical DFM notion of role-playing dimensions.

A single temporal dimension (e.g., `Date`) is drawn once in the diagram, but its connections to fact schemes are annotated with window-specific role labels:

`Date[OW]`, `Date[WW]`, and `Date[PW]`. These labels indicate the semantic interpretation of time within each window.

The role `Date[OW]` denotes timestamps used for historical feature extraction (e.g., transaction execution time such as `TrxDate`). The role `Date[WW]` denotes temporal boundaries used for latency handling and data exclusion and may not be associated with any fact scheme. The role `Date[PW]` denotes timestamps used for outcome realization and target labelling (e.g., instalment observation or default occurrence time such as `LoanDate`).

Graphically, role annotations are placed next to the connectors linking the `Date` dimension to fact schemes located in the corresponding window band. This makes explicit that a shared calendar hierarchy is used across windows, while its semantic interpretation may vary depending on the analytical role and the underlying temporal attribute.

Dimensions and temporal metadata. The PO DFM retains the quasi-tree dimension structure of the classical DFM, but dimensions may carry additional temporal metadata:

- attributes such as `RiskCategory` may evolve through SCD-Type 2 versions;
- the `Date` dimension acts as the temporal backbone marking the boundaries of ω_{OW} , ω_{WW} , and ω_{PW} ;
- contextual dimensions (e.g., `Economic_Indicators`) may span multiple windows and are linked via contextual connectors rather than predictive dependencies.

The resulting PO DFM constellation for the financial scenario is illustrated in Figure 3.10. Three horizontal temporal bands locate fact schemes according to their analytical role, vertical alignment ensures conformance across windows, and

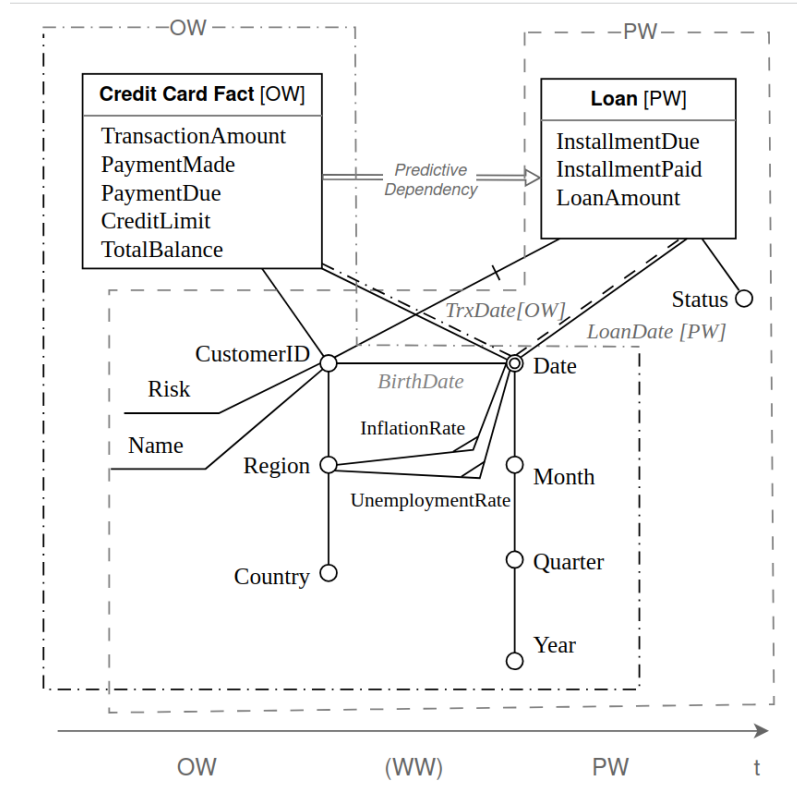


Figure 3.10: PO DFM constellation using the proposed graphical notation

directional connectors encode predictive semantics consistent with the $\omega_{OW} \rightarrow \omega_{PW}$ flow inherent in the PO OLAP operator chain.

Predictive semantics in the graphical notation.

- horizontal bands represent ω_{OW} , ω_{WW} , and ω_{PW} ;
- border styles distinguish feature-providing and outcome-providing fact schemes;
- predictive dependency arrows encode forward-looking analytical flow;
- role-annotated temporal dimensions make window-specific notions of time explicit;
- vertical alignment ensures drill-across and PO OLAP compatibility.

3.6.2 PO DFM Metamodel

Figure 3.11 presents the formal metamodel underlying the PO DFM notation. It extends the classical DFM metamodel by adding constructs that model temporality, window participation, and prediction-specific dependencies.

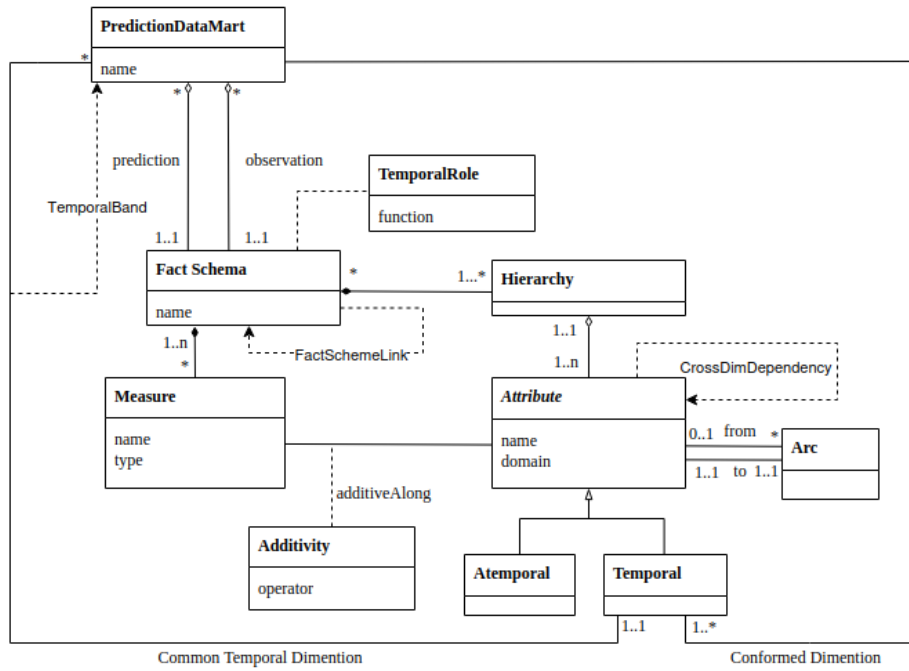


Figure 3.11: Metamodel of the Prediction-Oriented Dimensional Fact Model

The metamodel includes the following constructs:

- **TemporalBand**: Defines OW, WW, and PW and their precedence relationships;
- **TemporalRole**: A role construct associating fact schemes, dimensions, or individual temporal attributes with one or more temporal bands (e.g., `Date[OW]`, `Date[PW]`), as formalized by the binding function ψ ;
- **FactScheme**: Classical DFM fact scheme extended with temporal bindings;

- **Measure and Attribute:** Classical DFM elements inherited without structural modification, whose admissible usage is constrained by their associated TemporalRole within the PO DFM;
- **TemporalAttribute** (optional specialization): A specialization of Attribute representing time-stamped values (e.g., **TrxDate**, **LoanDate**) that may play different roles across temporal bands;
- **FactSchemeLink:** Represents directed predictive or temporal dependencies between fact schemes;
- **ConformedDimension:** (Implicit via Dimension) ensuring consistent alignment across windows;
- **CrossDimDependency:** Analytical or contextual relationships (e.g., macroeconomic enrichment).

This metamodel provides the conceptual substrate for the PO DFM notation, ensuring that structural, temporal, and predictive semantics are consistently represented, both graphically and formally. The metamodel is intentionally minimal, extending the classical DFM only where required to express temporal precedence and predictive semantics, while inheriting all remaining structural constructs unchanged.

3.6.3 The PO DFM Formal Definition

The Prediction-Oriented Dimensional Fact Model provides the formal foundation of the temporal-analytical framework introduced in the previous sections. Whereas the classical DFM [39, 40] offers a structural representation of fact schemes and their dimensional contexts, it remains agnostic to the temporal and causal constraints

inherent in predictive analytics. In particular, traditional multidimensional models lack (i) an explicit representation of learning windows, (ii) a mechanism for enforcing temporally valid feature - target separation, and (iii) a formal way to encode directed dependencies between distinct fact schemes.

The PO DFM addresses these shortcomings by enriching the DFM with a mathematically grounded *temporal layer* and a set of *prediction-oriented role bindings*. At its core, the PO DFM introduces a temporal structure centred on an analytical anchor point and embeds this structure into both single fact schemes and fact constellations. The resulting model supports PO OLAP, temporal slicing, and causal drill-across queries in a formally disciplined manner.

Formally, the PO DFM operates at two complementary levels:

1. **Temporal Structure (Window Model)** - a mathematically defined partition of time into three analytical windows relative to an anchor time t_0 .
2. **Model Specification (PO DFM)** - the application of this temporal structure to:
 - a *single* fact scheme (Single-Fact PO DFM), and
 - an *interconnected constellation* of multiple fact schemes (Constellation level PO DFM).

Together, these elements form a unified temporal-structural formalism suitable for prediction-based data warehouse design and automated construction of learning-ready datasets.

3.6.3.1 The Single-Fact PO DFM

We begin by extending the standard DFM semantics of an individual fact scheme to include an associated temporal structure. Recall Definition 3.4.2, where a fact

scheme is represented concisely as $f = (H, M)$. For the purpose of the PO DFM, we explicitly identify the set of dimensions D derived from the root of the hierarchies in H .

Definition 3.6.1 (Single-Fact PO DFM). A **Prediction-Oriented DFM for a single fact scheme** f is defined as the triple:

$$\mathcal{M}_{\text{PO}}^{(1)} = (f, \Delta T, \Psi_f), \text{ where:}$$

- $f = (H, M)$ is the underlying structural fact schema.
- $\Delta T = (\Delta t_{\text{OW}}, \Delta t_{\text{WW}}, \Delta t_{\text{PW}})$ is the temporal configuration vector that induces the window tuple $\Omega(t_0)$ relative to an anchor t_0 .
- Ψ_f is a set of **Temporal Role Bindings**. Each binding is a pair:

$$\psi = (\rho, \omega), \text{ where:}$$

$\rho \in \{\text{FEATURE}, \text{TARGET}, \text{CONTEXT}\}$ is a semantic role, and $\omega \in \Omega(t_0)$ is the specific temporal window associated with that role.

The bindings in Ψ_f induce canonical projections of the fact f onto specific time intervals. Let $t(r)$ denote the value of the temporal dimension attribute for a fact instance $r \in f$. The projections are defined as:

$$f_{\text{FEATURE}} = \text{SLICE}(f, \omega_{\text{OW}}) = \{r \in f \mid t(r) \in \omega_{\text{OW}}\}$$

$$f_{\text{TARGET}} = \text{SLICE}(f, \omega_{\text{PW}}) = \{r \in f \mid t(r) \in \omega_{\text{PW}}\}$$

$$f_{\text{CONTEXT}} = \text{SLICE}(f, \omega_{\text{OW}} \cup \omega_{\text{PW}})$$

This formalism allows a single physical table to play multiple analytical roles depending on its temporal scope. For instance, a **Loan_Fact** scheme can serve

simultaneously as:

- an **input** (past instalments) when bound to $\rho = \text{FEATURE}$,
- an **output** (default states) when bound to $\rho = \text{TARGET}$.

Thus, the Single-Fact PO DFM equips individual cubes with intrinsic predictive semantics, enabling time-aware aggregates, rolling windows, and PO-OLAP transformations without altering the underlying physical schema.

3.6.3.2 The Constellation-Level PO DFM

Predictive analytics often requires the integration of multiple heterogeneous sources (e.g., credit behaviour, loan performance, and macroeconomic conditions). The Constellation-Level PO DFM formalizes this integration under a shared temporal structure.

Definition 3.6.2 (Constellation-Level PO DFM). A **prediction-oriented Fact Constellation** is the triple:

$$\mathcal{M}_{\text{PO}} = (\mathcal{FC}, \Delta T, \Psi_{\mathcal{FC}}), \text{ where:}$$

- $\mathcal{FC}$ is a classical **Fact Constellation** (Def. 3.4.5) consisting of a set of fact schemes $\mathcal{F} = \{f_1, \dots, f_k\}$ interconnected via shared and conformed dimensions.
- ΔT is the global temporal configuration shared across all fact schemes in $\mathcal{F}$.
- $\Psi_{\mathcal{FC}} = \bigcup_{i=1}^k \Psi_{f_i}$ is the collective set of temporal role bindings, assigning each fact $f_i \in \mathcal{F}$ and its associated temporal attributes to one or more windows (e.g., `TrxDate[OW]`, `LoanDate[PW]`), and thereby determining their graphi-

cal placement and admissible PO OLAP operations (e.g., `CreditCard_Fact` bound to ω_{DW} , `Loan_Fact` bound to ω_{PW}).

The single-fact model $\mathcal{M}_{\text{PO}}^{(1)}$ appears as the special case where $|\mathcal{F}| = 1$.

Inter-fact Temporal and Structural Coordination. Facts in the constellation interact through connecting dimensions. For any two facts f_i and f_j , the set of valid drill-across dimensions is simply the intersection of their dimension sets:

$$D_{\text{connect}}(f_i, f_j) = \text{Dim}(f_i) \cap \text{Dim}(f_j)$$

If $D_{\text{connect}} \neq \emptyset$, the facts are structurally compatible, allowing operations that remain semantically coherent and temporally aligned according to the global ΔT .

3.6.3.3 Explicit Predictive Dependencies

Classical constellations capture only *structural compatibility* (shared dimensions). The PO DFM additionally models *temporal precedence* and explicit predictive associations between facts.

Definition 3.6.3 (Explicit Link). An **Explicit Link** is a quadruple:

$$\Lambda = (\lambda_{\text{name}}, f_{\text{source}}, f_{\text{target}}, \mu), \text{ where:}$$

- λ_{name} is the unique identifier of the relationship (e.g., `predicts_outcome`).
- f_{source} and f_{target} are the predictor (feature source) and outcome (target source) fact schemes, respectively.
- μ is a mapping function that defines instance-level correspondence (e.g., matching `CustomerID`).

Crucially, an Explicit Link must satisfy the Temporal Precedence Constraint:

$$\forall(r_s, r_t) \text{ linked by } \Lambda \implies t(r_s) \in \omega_{\text{OW}} \wedge t(r_t) \in \omega_{\text{PW}}.$$

An explicit link formalizes the directional dependency $f_{\text{source}}[\omega_{\text{OW}}] \Rightarrow f_{\text{target}}[\omega_{\text{PW}}]$, establishing that the source fact is strictly historical relative to the target fact.

Example 3.6.1: Financial Constellation

Consider a constellation involving `CreditCard_Fact` and `Loan_Fact`. Both schemes share the `Customer` dimension.

- **Shared dimensions** enable contextual alignment:

$$D_{\text{connect}} = \{\text{Customer}, \text{Date}\}.$$

This allows queries like "Compare average spending vs. loan balance for Customer X in 2023."

- **Explicit Links** enable predictive association:

$$\Lambda_{\text{risk}} = (\text{spending_predicts_default}, \text{CreditCard_Fact}, \text{Loan_Fact}, (\text{CustomerID}))$$

This strictly links *past* credit behaviour to *future* loan outcomes for the purpose of model training.

The resulting predictive query, which constructs a supervised learning dataset, is formally expressed as:

$$\mathcal{D}_{\text{train}} = \sigma_{\omega_{\text{PW}}}(\text{Loan_Fact}) \bowtie_{\Lambda} \gamma_{\text{aggregates}}(\sigma_{\omega_{\text{OW}}}(\text{CreditCard_Fact})), \text{ where:}$$

$\bowtie_{\Lambda}$ represents the join constrained by the explicit link mapping and the temporal precedence defined by the windows.

The PO DFM enables two complementary analytical perspectives within the same data warehouse environment:

1. A single fact scheme is logically partitioned into window-specific slices. This supports the generation of rolling-window features (e.g., "avg spend last 3 months") under the specified window constraints, without physical data duplication.
2. Multiple facts interact through shared dimensions and explicit predictive links. This supports predictive drill-across, directed predictive dependencies between fact schemes, and the automatic construction of ABTs for prediction-oriented analytics.

Beyond fact schemes and shared dimensions, the Constellation-Level PO DFM supports additional formal constructs that admit a direct graphical representation in the PO DFM notation.

- **Temporal Bands.** The global temporal configuration ΔT is rendered as three horizontally ordered analytical bands corresponding to ω_{OW} , ω_{WW} , and ω_{PW} . These bands visually encode temporal precedence and constrain admissible analytical flows across the constellation.
- **Explicit Predictive Links.** Each Explicit Link Λ is rendered as a directed connector from f_{source} to f_{target} , spanning the OW and PW bands. The arrow direction encodes predictive causality $f_{\text{source}}[\omega_{OW}] \Rightarrow f_{\text{target}}[\omega_{PW}]$ and enforces the Temporal Precedence Constraint.
- **Role-Annotated Temporal Dimensions.** Temporal dimensions participating in window definitions are annotated with explicit roles (e.g., $\text{TrxDate}[OW]$),

`LoanDate[PW]`). This representation makes visible that different temporal attributes or different semantic roles of the same calendar dimension, govern window membership for different facts.

- **Contextual Dimensions Across Windows.** Contextual dimensions (e.g., `Economic_Indicators`) are shown as spanning multiple temporal bands without directional arrows. This visually distinguishes exogenous, non-behavioural enrichment from predictive dependencies.
- **Window-Specific Fact Participation.** Fact schemes are rendered with band-specific placement and border styles, reflecting their temporal binding function $\Psi_{\mathcal{FC}}$ and making explicit whether a fact contributes features, outcomes, or neither within a given window.

Thus, the PO DFM unifies structural multidimensional modelling with temporal semantics and predictive dependencies, forming a rigorous foundation for time-aware analytical data engineering.

3.7 Implementation and Empirical Validation of the PO Framework

This section presents the implementation and empirical validation of the Prediction-Oriented Framework at the level required to demonstrate implementability, reproducibility, and temporal correctness. Detailed Python and SQL source listings, package structures, execution procedures, and extended testing material are provided in Appendix B. The present section focuses on the mapping from the conceptual specification to executable artefacts, the two implementation paths, the validation protocol, and the evidence obtained in the evaluated configurations.

The Python and SQL implementations are not independent research contributions. They are complementary realizations of the same conceptual and formal specification. The Python realization supports modular and exploratory analytical workflows, whereas the SQL realization demonstrates that the same temporal semantics can be expressed using relational and data-warehouse operations.

3.7.1 Mapping Conceptual Constructs to Executable Artefacts

Implementing the PO Framework requires adapting the conceptual structures introduced in Section 3.6 into executable logical schemas and physical processes.

While the PO DFM specifies the analytical semantics, such as fact schemes, temporal bindings $\psi(f_i) \in \{\text{OW}, \text{WW}, \text{PW}, \text{CONTEXT}\}$, and temporal parameters $\Delta T = (\Delta t_{\text{OW}}, \Delta t_{\text{WW}}, \Delta t_{\text{PW}})$, predictive dependencies and window assignments - the technological implementations (Python-based and SQL-based) operationalize these semantics within concrete data processing environments.

Role of the Conceptual Model. The PO DFM constellation serves as the authoritative specification of the analytical environment. It defines:

- the **facts** participating in the predictive workflow,
- the **dimensions** that provide conformed analytical context,
- the **temporal roles** assigned to each fact through the binding function $\psi(f_i)$,
- and the **predictive dependencies** connecting OW and PW fact schemes.

This conceptual representation is implementation-agnostic: it specifies what analytic objects exist and how they relate, but not how they are technically executed.

The PO Framework implementation follows a layered architecture that separates conceptual modelling from technological implementation while maintaining strict adherence to temporal semantics. Figure 3.13 illustrates the foundational three-window structure that governs both implementation approaches.

Transition to Logical Modelling. The logical modelling layer translates the PO DFM into functional data-processing abstractions. This layer differs for each implementation:

- **In the Python implementation**, the logical layer is expressed through reusable classes and functions (e.g., `WindowConfig`, `WindowExtractor`, feature modules, ABT builders). These abstractions mirror the conceptual constructs—windowed facts, temporal precedence, and feature/label separation—while enabling programmatic composition of analytical workflows.
- **In the SQL implementation**, the logical layer is realized through relational schemas, temporal views (e.g., `Fact_OW`, `Fact_PW`), and procedural templates for PO OLAP operations such as `SLICE`, `AGGREGATE`, and `JOIN(OW,PW)`. This

provides a declarative execution environment grounded in the temporal semantics of the PO Framework.

At this level, the three-window temporal structure becomes explicit: each fact scheme is exposed through a suite of window-specific logical views, and each analytical transformation is mapped to a corresponding temporal operation.

Transition to Physical Implementation. The physical layer instantiates the logical design within concrete technological environments, translating conceptual PO DFM constructs and PO OLAP operations into executable data processing artefacts:

- **Python Physical Model:** In-memory dataframes with explicit temporal indexing, vectorized feature computation over Observation and Prediction Windows, and seamless integration with visualization and machine learning libraries. Physical persistence is achieved through serialized Analytical Base Tables (e.g., Parquet files) and derived analytical outputs such as plots and reports.
- **SQL Physical Model:** Relational implementations based on materialized views or partitioned tables corresponding to the OW, WW, and PW windows; indexing strategies over (`CustomerID`, `Date`); and optimized execution plans supporting Prediction-Oriented OLAP operations such as window-restricted aggregation, temporal joins, and target labelling.

Both implementations were designed to operationalize the same temporal semantics, and the reported case-study configurations were checked against the same window and precedence constraints.

$$\text{CreditCard_Fact[OW]} \Rightarrow \text{Loan_Fact[PW]}$$

Purpose of Dual Implementations. Providing both a Python and an SQL implementation serves two complementary goals:

1. **Analytical Flexibility:** Python supports exploratory, iterative, and machine learning-driven workflows, enabling advanced feature engineering and automated figure production.
2. **Operational Integration:** SQL supports enterprise data warehouse environments and BI reporting, leveraging existing infrastructure and ensuring reproducibility through declarative temporal views.

Despite differences in technology, both implementations operationalize the same conceptual model and produced consistent outputs for the reported case-study configurations and the outputs that were explicitly compared. Each implementation thus represents an instantiation of the PO DFM in a distinct physical ecosystem, demonstrating the framework’s generality, portability, and methodological rigour. All the code and relevant files, as well as instructions to execution can be found in **the dedicated repository**.

3.7.2 Python Realization

The Python realization follows a modular pipeline that mirrors the conceptual structure of the PO Framework. A temporal configuration defines the anchor point and the three window lengths. Window-specific extraction then separates the records admissible for feature generation from those used for outcome labelling. Feature modules generate behavioural and contextual indicators from OW data, while target variables are derived exclusively from PW data. The two components are joined at the analytical grain to construct the ABT, after which the temporal validator checks window membership, precedence, separation, and basic data-quality conditions.

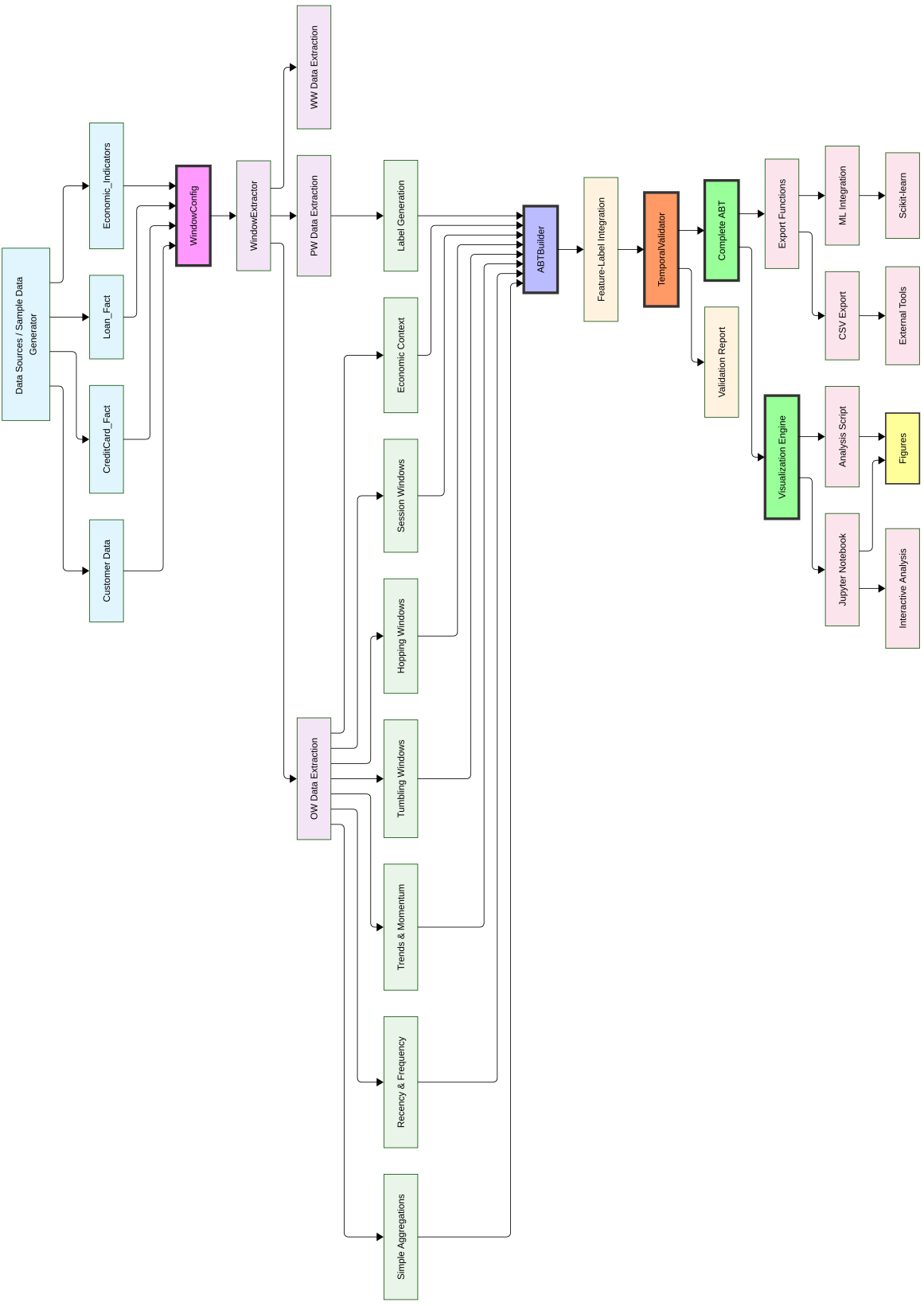


Figure 3.12: Layered architecture of the Python realization of the Prediction-Oriented Framework

The Python implementation follows a layered architecture that separates concerns while maintaining cohesive integration. Figure 3.12 illustrates the architectural components and their interactions. The architecture clearly separates data management, temporal constraint enforcement, feature engineering, and user interfaces, while ensuring that strict temporal ordering is preserved throughout the pipeline. It supports both interactive analysis through Jupyter notebooks and automated execution via stand-alone scripts, with comprehensive figure generation and validation reporting integrated at each layer.

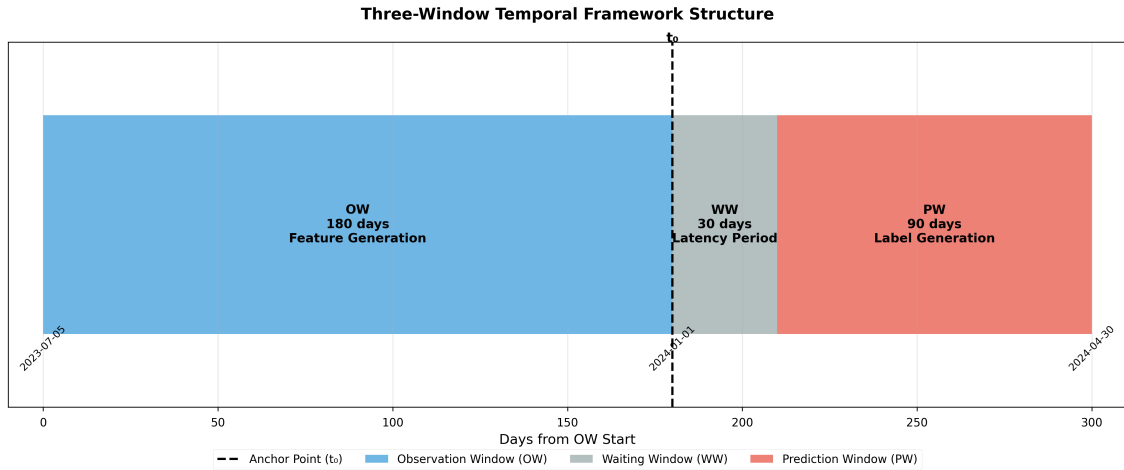


Figure 3.13: Three-Window Temporal Framework Structure

The core of the framework is built around an explicit partitioning of time into three non-overlapping analytical windows (Figure 3.13): the *Observation Window* (OW, 180 days), used for historical feature generation; the *Waiting Window* (WW, 30 days), which introduces a temporal separation buffer; and the *Prediction Window* (PW, 90 days), used for outcome observation. All windows are defined relative to a single anchor point t_0 (2024-01-01), which represents the conceptual present of the analysis. This design enforces strict temporal precedence, ensuring that all features are computed exclusively from past data relative to t_0 , while all labels are derived from subsequent observations. By construction, the windows do not overlap.

The detailed package tree, API examples, notebook workflow, executable listings, and generated diagnostic figures are reported in Appendix B.1.

Python Implementation on Credit Risk Case Study

The Python implementation is validated through a comprehensive test using the `po_framework_analysis.py` script, which demonstrates the complete analytical workflow from data generation to figure creation. The test configuration employs the following parameters:

- *Customer Base*: 10,000 customers with diverse risk profiles;
- *Temporal Coverage*: January 1, 2022 to June 1, 2025 (3.5 years);
- *Anchor Point*: t_0 = January 1, 2024;
- *Window Configuration*: OW = 180 days, WW = 30 days, PW = 90 days;
- *Random Seed*: 100 (for reproducible results).

The script generates realistic financial data with the following characteristics:

- *Credit Card Transactions*: Approximately 306,737 transactions distributed across the temporal range with risk-based patterns;
- *Loan Instalments*: Approximately 152,493 loan records with status patterns reflecting customer risk categories;
- *Economic Indicators*: 24 contextual records across four regions and six time periods;
- *Customer Demographics*: Balanced distribution across regions (North, South, East, West) and risk categories (Low, Medium, High).

After applying the window configuration, the test yields:

- *OW Records*: 44,335 credit card transactions for feature generation;
- *PW Records*: 12,000 loan instalments for label generation;
- *Active Customers*: 9,802 customers with OW activity, 8,757 with PW outcomes;
- *Final ABT*: 9,802 rows $\times$ 75 columns (55 features + metadata + labels).

This comprehensive test configuration demonstrates the framework’s scalability and validates its temporal correctness across a substantial dataset that approximates real-world analytical scenarios.

Figure 3.14 demonstrates the temporal data distribution across the three windows, confirming proper segmentation and the absence of temporal overlap that could compromise analytical validity. The figure presents four complementary views of temporal data distribution: (a) daily transaction volume over time with window boundaries highlighted in blue (OW), gray (WW), and red (PW), showing clear temporal separation; (b) transaction counts by window, with 44,335 records in OW, 7,373 in WW, and 12,000 in PW; (c) cumulative transaction growth demonstrating data accumulation patterns; and (d) active customers by window, with 9,802 unique customers in OW and 8,757 in PW. The visualization confirms proper temporal segmentation with no overlap between windows.

The Python implementation constructs multidimensional data cubes for both OW and PW data, enabling OLAP-style analysis within the temporal framework. The OW data cube organizes historical behavioural data across Time (monthly granularity) and Region dimensions, facilitating drill-down and roll-up operations for feature generation.

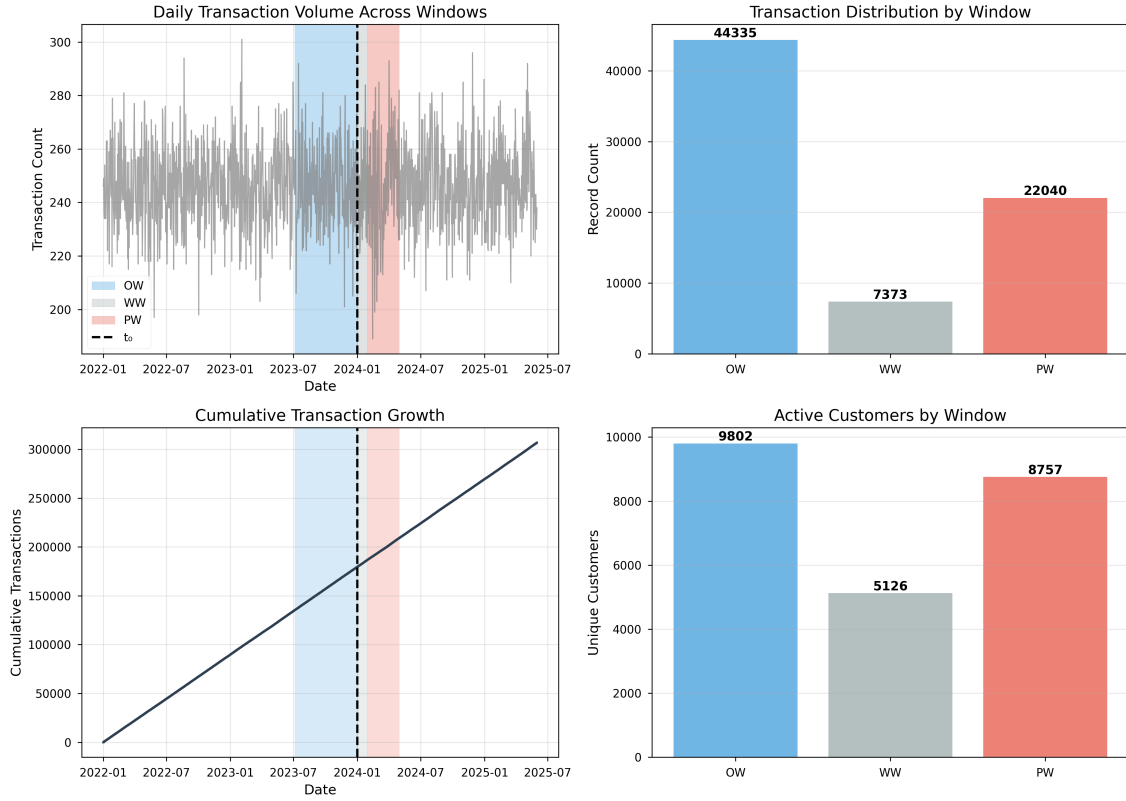


Figure 3.14: Transaction Distribution Across Temporal Windows

The OW data is organized as a multidimensional cube (Figure 3.15), enabling OLAP-style aggregation across Time and Region dimensions for comprehensive behavioural pattern analysis. The OW data is organized as a multidimensional cube with Time (monthly granularity) and Region dimensions, comprising 24 cells. The figure shows (a) total spending by region and month as a heatmap, revealing temporal and spatial spending patterns; (b) monthly transaction volume across the six-month OW period; (c) average transaction amount by region, showing regional behavioural differences; and (d) active customers by region, indicating customer distribution. This cube structure enables OLAP-style drill-down, roll-up, and slice operations for feature generation, demonstrating how the framework integrates multidimensional modelling with temporal analytics.

Similarly, the PW data cube captures future outcomes across the same dimen-

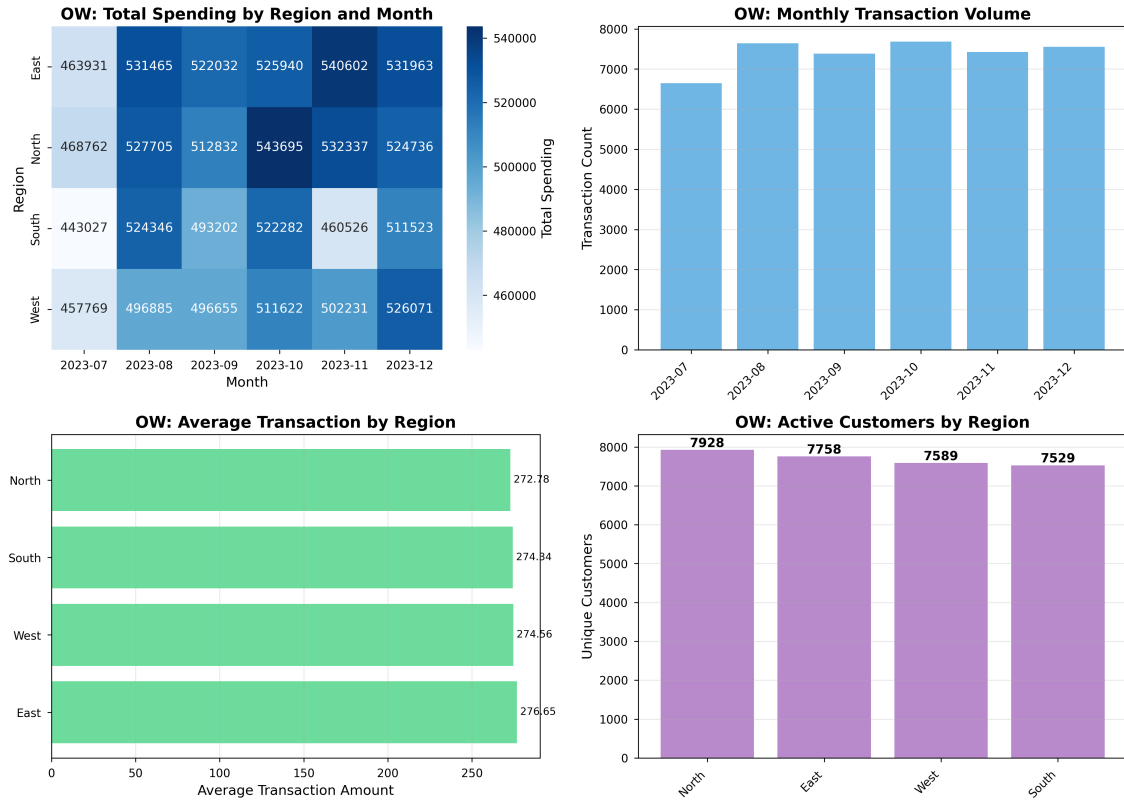


Figure 3.15: Observation Window Multidimensional Data Cube Analysis

sional structure, maintaining temporal separation while enabling outcome analysis. Figure 3.16 illustrates the PW data cube structure, showing the distribution of outcomes across temporal and spatial dimensions. The PW data is organized as a multidimensional cube with Time (monthly granularity) and Region dimensions, comprising 16 cells. The figure shows (a) payment rate by region and month as a heatmap, with values ranging from 69% to 99%, indicating varying payment performance; (b) monthly default events, showing temporal concentration of defaults; (c) payment performance comparison between amounts due and paid by region; and (d) default events by region, revealing spatial patterns in credit risk. This cube structure captures future outcomes used for label generation, maintaining strict temporal separation from the OW through the intervening WW.

For procedural pipelines, Python provides an expressive and reproducible im-

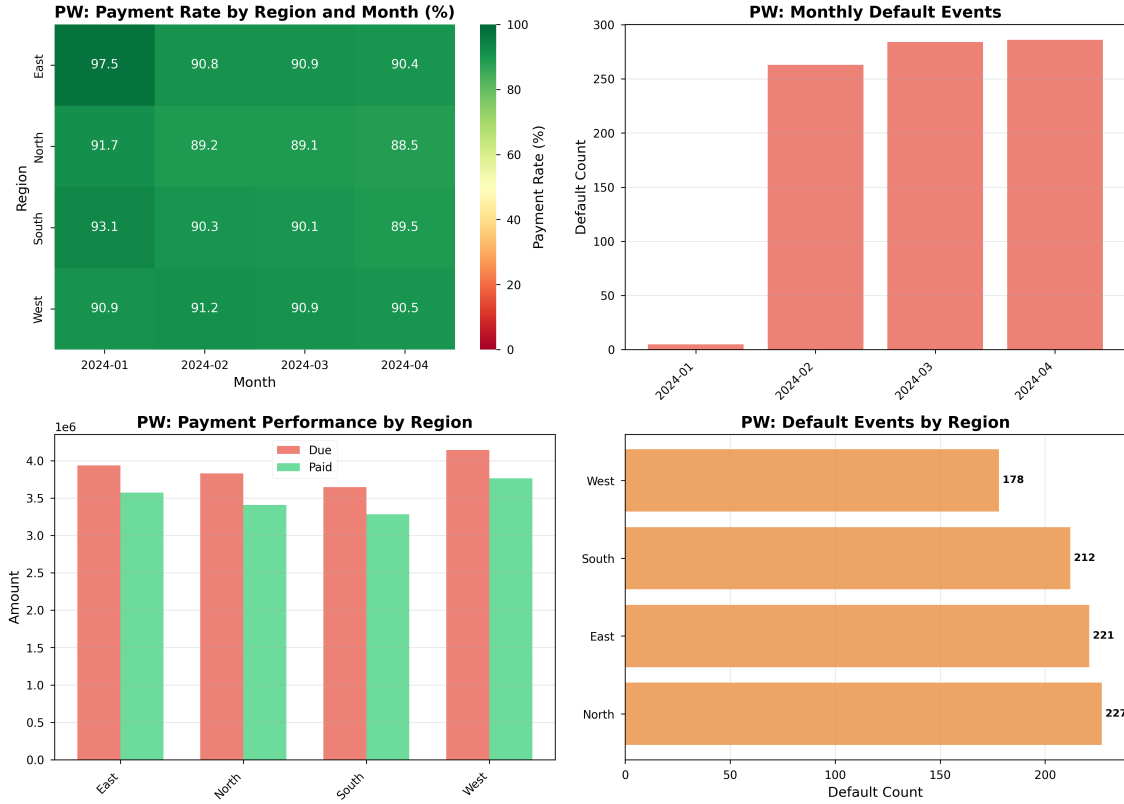


Figure 3.16: Prediction Window Multidimensional Data Cube Analysis

plementation of the Prediction-Oriented Framework. Window extraction is implemented through reusable functions that mirror the semantics of window-specific SQL views, enabling consistent behaviour across execution environments.

Data extraction and ABT construction can be implemented in Python with a small set of reusable components that mirror the Prediction-Oriented OLAP operators. A centralized `WindowConfig` specifies the anchor time t_0 and the lengths of the Observation, Waiting, and Prediction Windows, and validates basic constraints. Based on this configuration, `WindowExtractor` operationalizes SLICE by producing OW and PW subsets with explicit temporal predicates and traceability metadata.

Feature engineering applies `AGGREGATE` to OW data through modular feature groups (e.g., simple aggregations, recency/frequency, trends, and advanced windowing), while `LabelGenerator` derives targets exclusively from PW data, ensuring

temporal precedence. **ABTBuilder** composes extraction, feature computation, and labelling and performs the required joins to produce a reproducible Analytical Base Table (ABT). For backtesting, the same pipeline is executed over a sequence of anchor points to generate rolling ABTs.

The implementation is designed to enforce (i) temporal precedence, (ii) isolation of windows, (iii) reproducibility as a function of $(t_0, \Delta T)$, and (iv) generality across fact tables and domains. The corresponding code listings, parameter examples, and the full temporal validation procedure are reported in Appendix B.1.

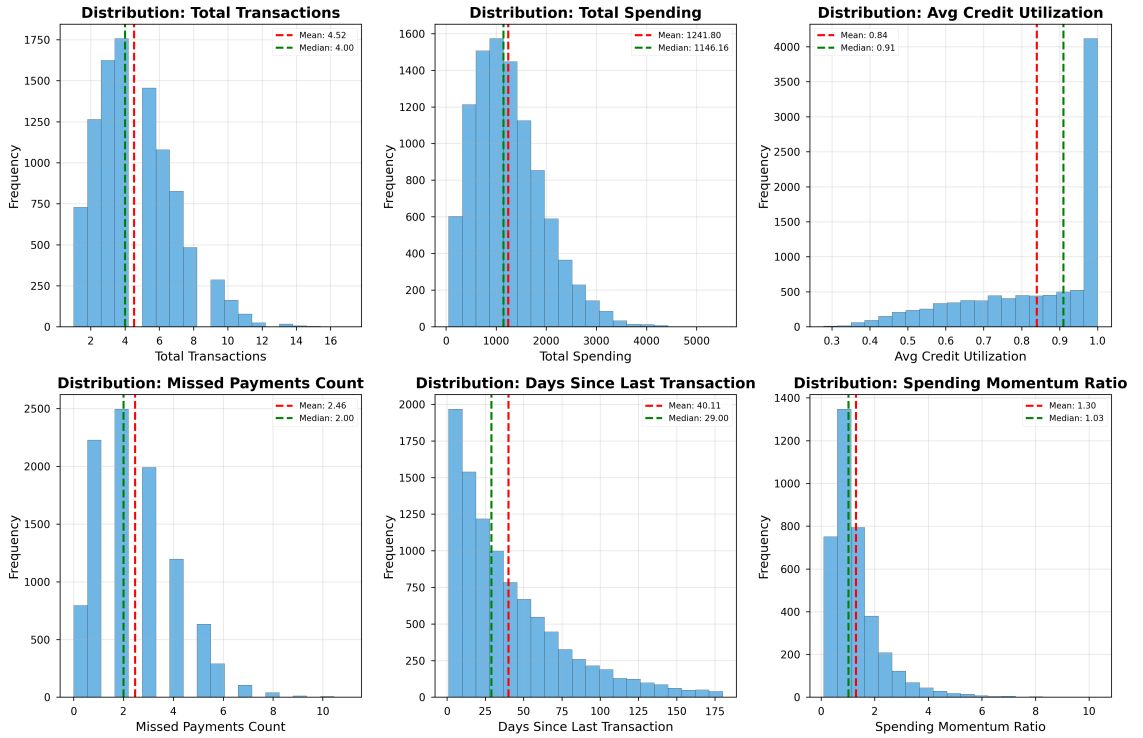


Figure 3.17: Distribution of Key Engineered Features

Figure 3.17 presents the distribution of key engineered features, revealing statistical properties that inform feature scaling and transformation decisions in downstream predictive modelling. The figure shows histograms for six representative features from the 10,000 customer test: total transactions ($mean = 4.52$), total spending ($mean = 1,241.80$), average credit utilization ($mean = 0.84$), missed pay-

ments count ($mean = 2.46$), days since last transaction, and spending momentum ratio. Mean values are indicated by red dashed lines and median values by green dashed lines. The distributions reveal right-skewed patterns typical of financial behavioural data, with most customers exhibiting moderate activity levels and a tail of high-activity customers. These statistical properties inform feature scaling and transformation decisions in downstream predictive modelling.



Figure 3.18: Feature-Target Correlation Analysis

The predictive relationships are quantified in Figure 3.18, which Pearson correlation coefficients between six key features and the binary default target variable (`has_defaulted`). Negative correlations (red bars) indicate features inversely related to default risk: average credit utilization (-0.155), missed payments count (-0.118), spending momentum ratio (-0.089), total spending (-0.064), and total

transactions (-0.045). While correlations are modest in magnitude, they reveal measurable linear relationships between historical behavioural patterns captured in the OW and future default outcomes observed in the PW. These relationships validate the framework’s ability to identify predictive dependencies through temporal structuring, though non-linear relationships may be stronger and require advanced modelling techniques.

The constructed Analytical Base Table (ABT) can be used directly in standard machine learning frameworks. Because it is flat and entity-centric, with features and targets clearly separated, it integrates with libraries such as scikit-learn without extra temporal logic. All temporal constraints are handled during ABT construction, so downstream models operate on a time-consistent dataset without reasoning about window boundaries or record ordering (see Appendix B.1). The framework supports exporting the ABT and metadata to common formats for BI tools, batch ML workflows, and storage systems.

The Python implementation demonstrates:

- **Modularity:** Each component is independent and reusable;
- **Flexibility:** Easy to extend with new feature types;
- **Usability:** Simple API with clear documentation;
- **Validation:** Comprehensive temporal constraint checking;
- **Integration:** Seamless connection to ML frameworks;
- **Performance:** Efficient processing for typical analytical workloads.

3.7.3 SQL Realization: Database-Centric Approach

The SQL realization provides a database-native implementation of the same Prediction-Oriented semantics used by the Python workflow. It is intended for analytical environments in which temporal filtering, feature engineering, target construction, and validation are executed close to data stored in a relational database or data warehouse. The implementation was developed and tested in DuckDB and uses configuration metadata, window-specific views, modular feature views, a Prediction-Window label view, an Analytical Base Table (ABT) view, and explicit validation queries.

The SQL realization maps the conceptual constructs introduced in Sections 3.5.1 and 3.5.4 to relational artefacts. The temporal configuration $(t_0, \Delta t_{OW}, \Delta t_{WW}, \Delta t_{PW})$ is stored in a dedicated configuration relation. Observation- and Prediction-Window views implement the temporal bindings of the participating fact schemes. Feature views apply aggregations exclusively to OW data, while target views derive labels from PW data. The resulting relations are joined at the analytical grain to construct the ABT, after which dedicated queries evaluate window membership and temporal precedence.

The canonical execution sequence is:

SLICE(OW) $\rightarrow$ AGGREGATE $\rightarrow$ LABEL(PW) $\rightarrow$ JOIN(OW,PW) $\rightarrow$ VALIDATE.

This sequence expresses temporal direction explicitly: OW-derived features precede PW-defined outcomes, while the WW remains outside both feature and target construction. It represents a directed predictive dependency rather than a causal claim.

Figure 3.19 separates source data and temporal configuration from window-specific views, modular feature and label construction, ABT assembly, validation,

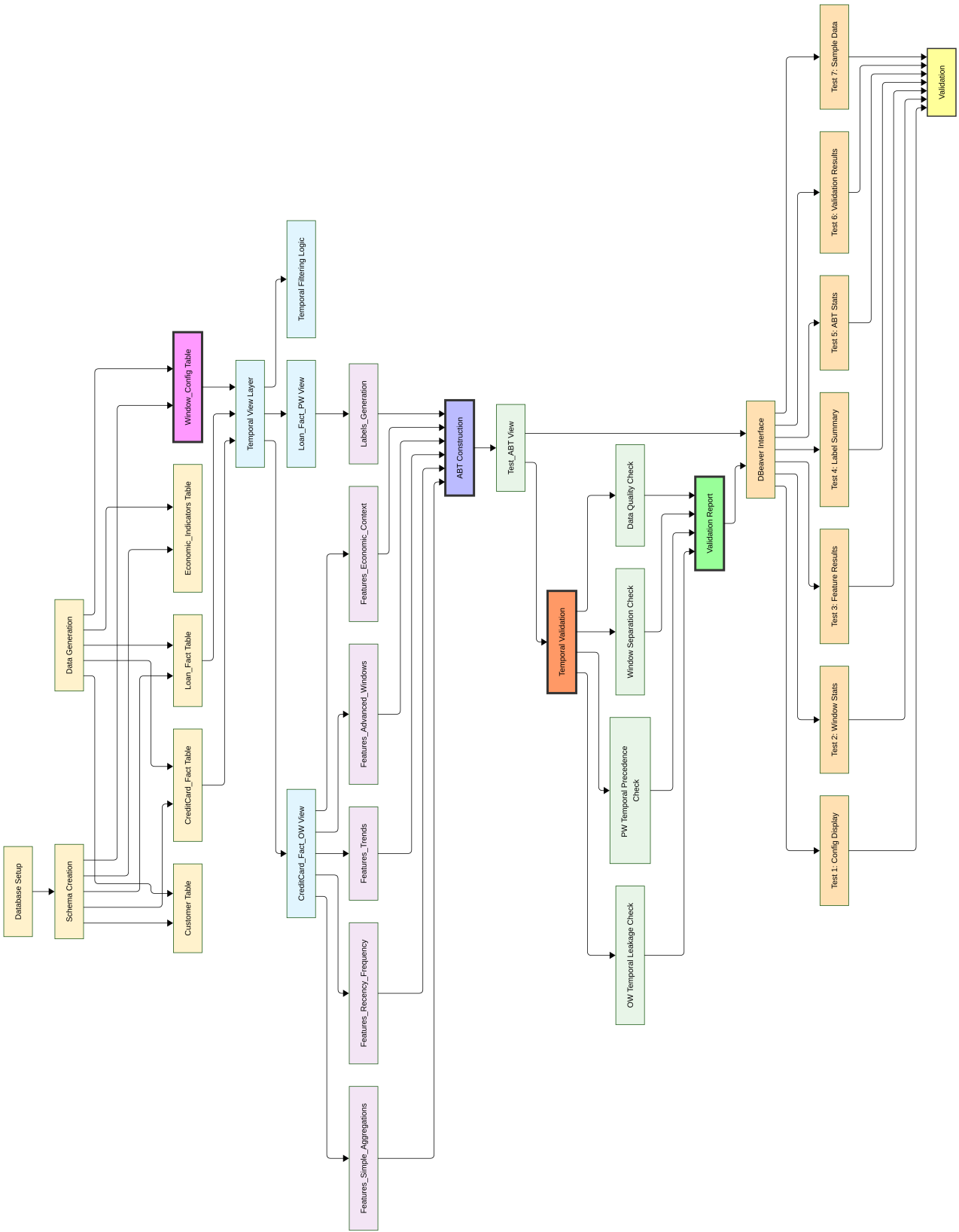


Figure 3.19: Layered architecture of the database-centric realization of the Prediction-Oriented Framework.

and downstream analytical outputs. The architecture is deliberately metadata-driven: changing an analytical anchor or a window duration does not require rewriting the conceptual operator sequence. Complete table definitions, view definitions, feature queries, rolling-anchor configurations, compatibility notes, and testing scripts are provided in Appendix B.2.

The SQL realization was evaluated through a controlled credit-risk workflow executed in DuckDB through DBeaver. The test environment contained 100 customers, approximately 4,000 credit-card transactions, 720 loan instalments, and 24 economic-indicator records. Three scripts created the schema, generated the test data, and executed seven checks covering temporal configuration, window-specific views, feature generation, target construction, ABT assembly, temporal precedence, and inspection of representative output records. The complete scripts and the test-by-test procedure are reported in Appendix B.2.

The compact three-customer example first demonstrates the correspondence between the conceptual operator chain and the generated analytical dataset. Figure 3.20 shows OW-derived behavioural indicators and a PW-derived default label for the motivating case. Customer **C789** has the highest utilization ratio in the example, two missed payments, and a default observed in the PW. Customer **C123** has two missed payments but no PW default, whereas **C456** provides a low-utilization, no-default reference case. These records illustrate the structure of the ABT; they do not constitute a predictive-performance evaluation.

The larger controlled workflow verifies the same sequence at database level. Figure 3.21 reports the window-view statistics. The OW view contains 1,310 records covering all 100 customers, with observed dates from 5 July 2023 to 31 December 2023. The PW view contains 124 records for 79 customers, with observed dates from 25 February 2024 to 25 April 2024. These ranges fall within the configured temporal intervals and remain separated by the waiting period.

	Row #1	Row #2	Row #3
CustomerID	C123	C456	C789
avg_utilization_ow	0.78	0.11	0.495
missed_payments_ow	2	0	2
avg_hopping_30d_txn_amount	200.4545454545	120	292.2222222222
spending_last_30d	250	180	350
spending_prev_30d	[NULL]	[NULL]	[NULL]
spending_trend_30d_vs_60d	[NULL]	[NULL]	[NULL]
avg_txns_per_session_ow	1	1	1
max_unemployment_ow	0.052	0.041	0.052
default_label	0	0	1

Figure 3.20: Illustrative SQL-generated Analytical Base Table for the three-customer default-risk example.

The feature view computes eight customer-level behavioural measures from OW data: transaction count, total and average spending, spending variability, average and maximum credit utilization, missed-payment count, and average payment ratio. Figure 3.22 shows representative customers ordered by missed payments and spending. The output demonstrates that the **AGGREGATE** stage produces one feature record per customer without drawing records from the PW.

Target construction is performed independently on PW data. As shown in Figure 3.23, 79 customers have loan activity in the PW, 12 of whom exhibit at least one default event, corresponding to an observed default rate of 15.19% in that PW-active subset. The SQL view also derives a broader payment-issue indicator, total default amount, and delayed-payment count; their complete definitions are given in Appendix B.2.

Figure 3.24 reports the ABT construction statistics. The final view contains 100 customer rows and non-null feature and label fields for all rows under the implementation’s join and default-value convention. Only 79 customers have observed PW loan activity; therefore, the reported completeness reflects the rule that an absent

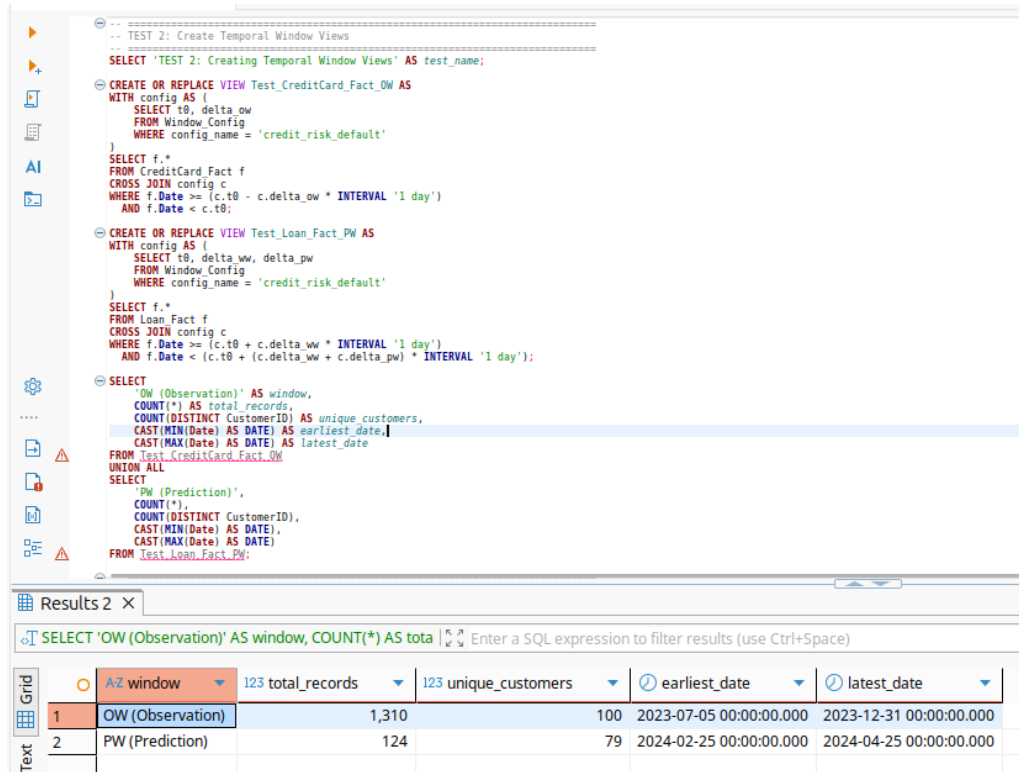


Figure 3.21: Record coverage of the Observation- and Prediction-Window SQL views.

PW default event is encoded as zero. It should not be interpreted as evidence that an outcome was directly observed for every customer.

The final validation queries test two core temporal constraints: whether any OW record occurs on or after the analytical anchor and whether any PW record occurs before the end of the waiting period. Figure 3.25 shows zero violations for both checks. This result provides implementation evidence that the specified predicates were enforced for the evaluated data; it does not establish the absence of every possible source of information leakage in downstream modelling.

Figure 3.26 provides a record-level inspection of the generated ABT. Historical transaction activity, spending, utilization, and missed-payment features are shown together with the future default label. The sample illustrates the final analytical grain and confirms that the output can be inspected and audited using standard

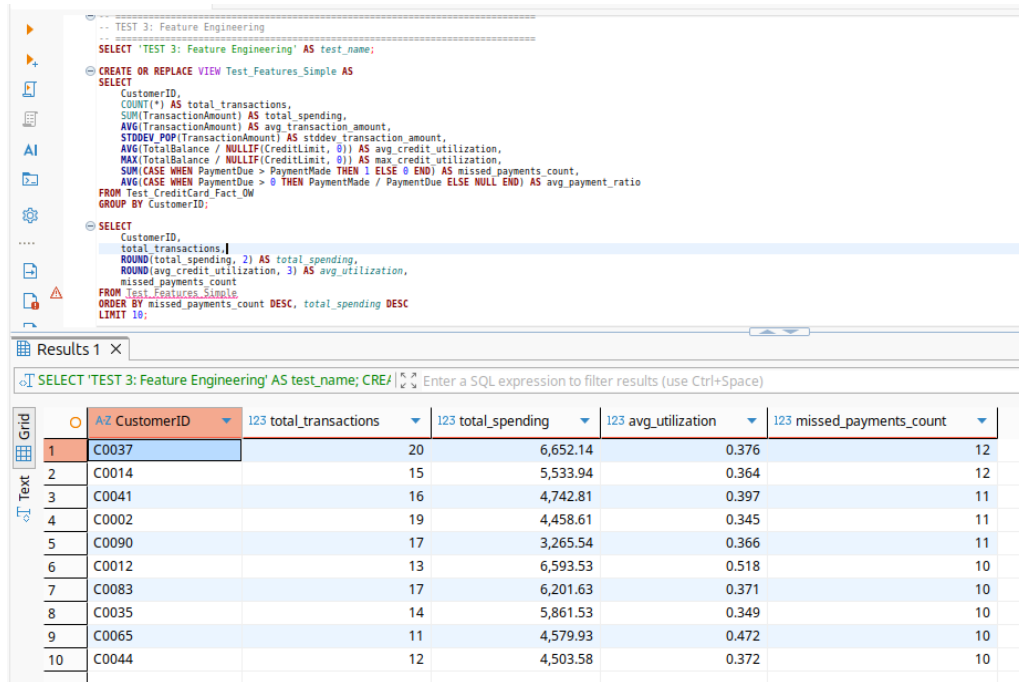


Figure 3.22: Representative customer-level features generated exclusively from Observation-Window data.

database tools.

Taken together, the SQL results show that the formal temporal semantics can be expressed using database-native operations, that feature and target relations can be assembled into an auditable ABT, and that selected temporal constraints can be checked automatically. The evidence concerns implementability and temporal correctness for the reported configuration. It does not establish predictive-model accuracy, superiority over alternative data-preparation methods, universal leakage prevention, or performance portability across arbitrary database platforms. Detailed SQL listings, compatibility notes, DBeaver procedures, and the additional configuration and implementation screenshots are retained in Appendix B.2.

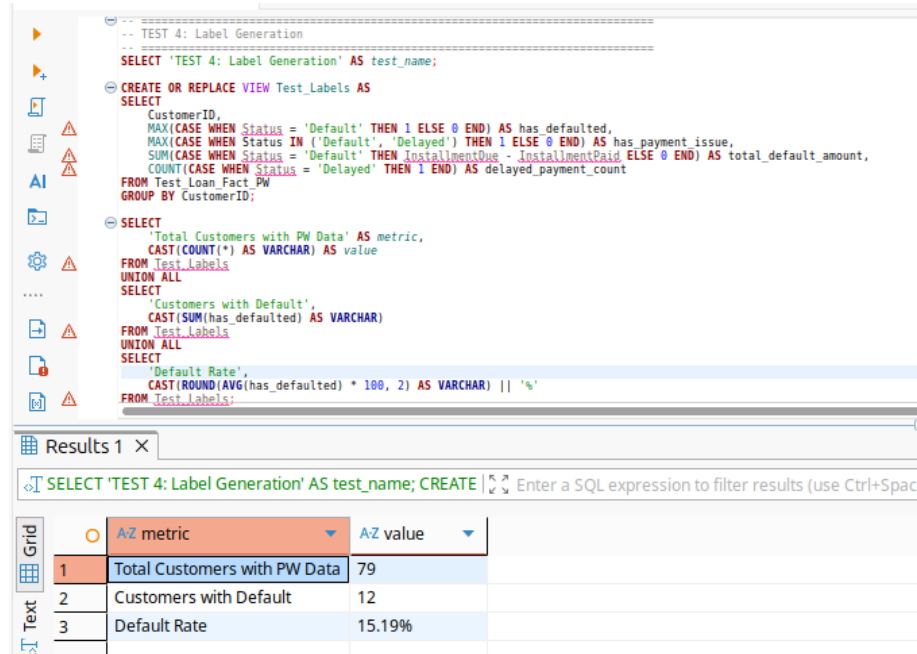


Figure 3.23: Prediction-Window label-generation summary for the controlled SQL test dataset.

Table 3.9: Summary of the SQL implementation evidence.

Evaluated component	Observed result
Temporal views	OW: 1,310 records for 100 customers; PW: 124 records for 79 customers; the observed date ranges satisfy the configured window predicates.
Feature construction	Eight behavioural measures were generated at customer grain from OW data.
Target construction	79 customers had PW loan activity; 12 had at least one default event; observed default rate 15.19%.
ABT assembly	100 customer rows were materialized; non-null labels for customers without PW default events result from the implementation's zero-default convention.
Temporal validation	Zero OW-membership violations and zero PW-precedence violations were reported by the two implemented checks.

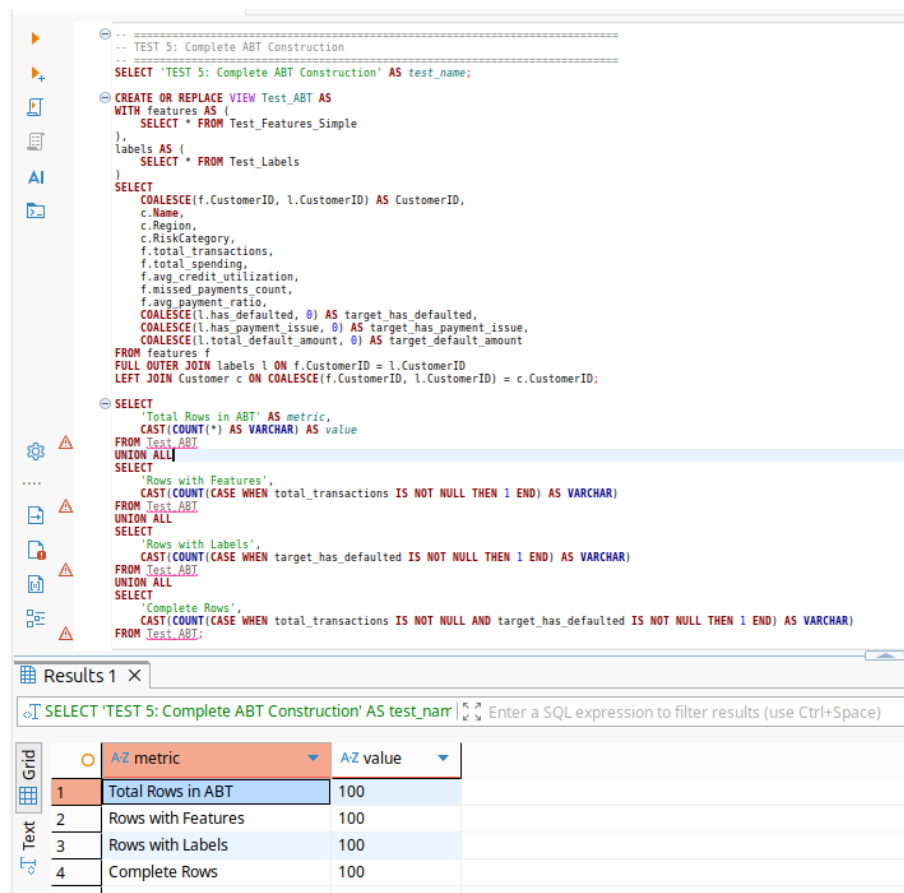


Figure 3.24: Analytical Base Table construction statistics for the controlled SQL workflow.

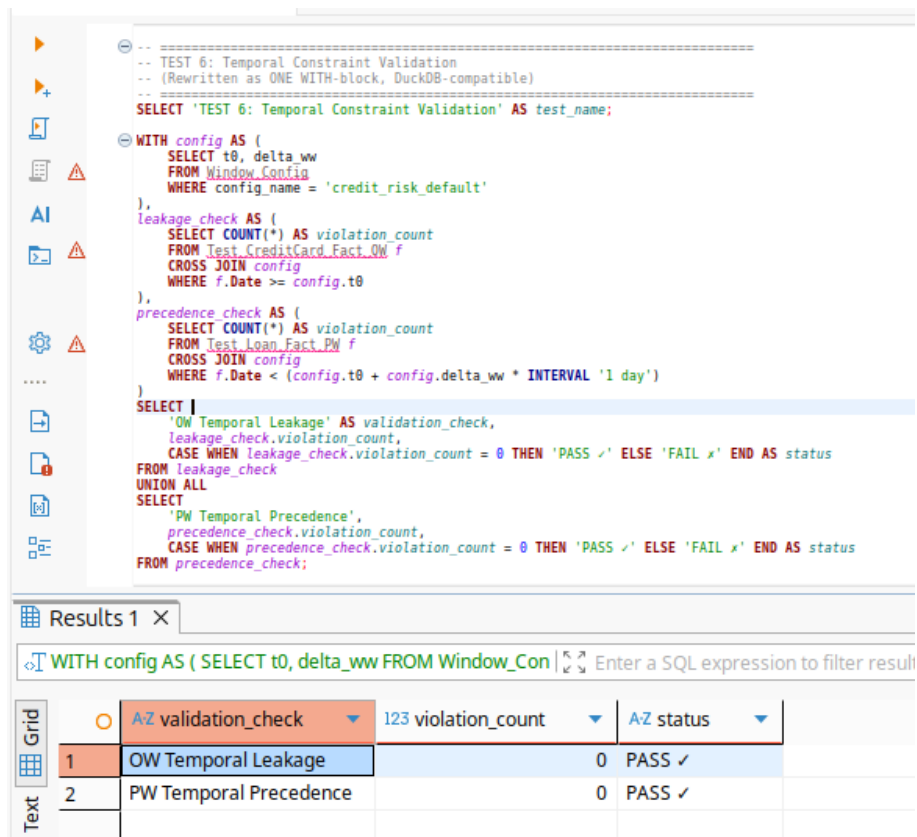


Figure 3.25: Results of the SQL temporal-membership and precedence checks.

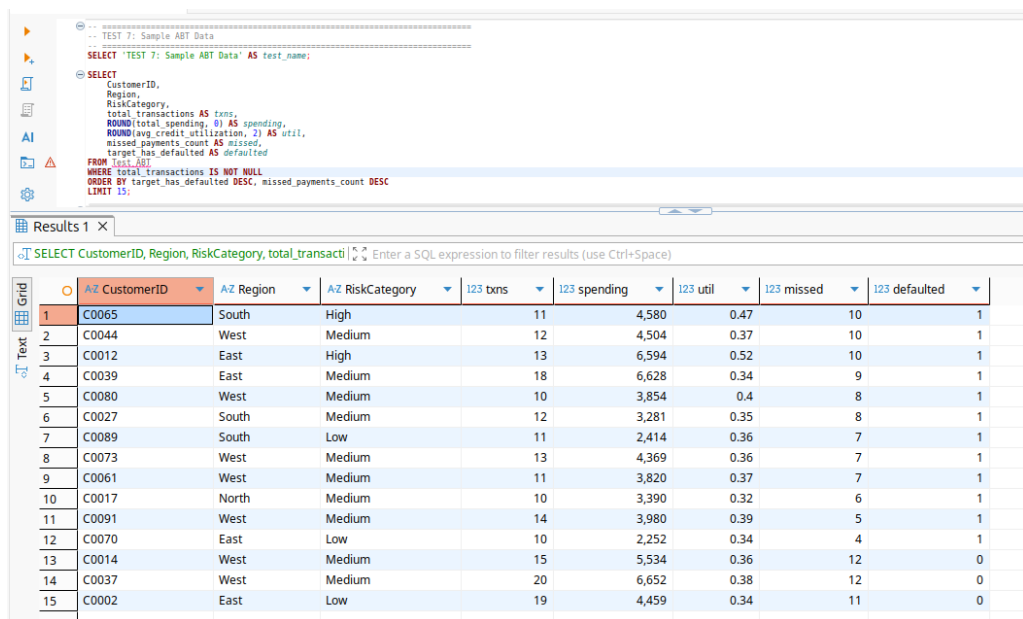


Figure 3.26: Representative records from the SQL-generated Analytical Base Table.

3.8 Conclusion

This chapter introduced a Prediction-Oriented Framework for multidimensional data analysis, addressing fundamental limitations of traditional multidimensional models when applied to predictive tasks. While conventional OLAP and dimensional modelling techniques are highly effective for descriptive and historical analysis, they lack explicit mechanisms for structuring data according to the temporal semantics required for prediction-oriented reasoning. This chapter demonstrated that such limitations can be systematically overcome through conceptual modelling, without relying on full-scale machine learning pipelines.

The core contribution of the chapter is the formalization of a three-window temporal framework, consisting of the Observation Window, the Waiting Window, and the Prediction Window. By explicitly distinguishing these temporal roles, the framework enables a clear separation between historical data used for feature generation, latency periods required for outcome maturation, and future intervals in which predictive targets are defined and evaluated. This explicit temporal structuring provides a principled foundation for prediction-oriented analysis that is absent in traditional multidimensional models.

Building on this temporal foundation, the chapter extended the Dimensional Fact Model to support prediction-oriented use cases. Enhancements include the representation of temporally evolving attributes, the modelling of inter-schema relationships through fact constellations, and the explicit encoding of predictive dependencies. These extensions preserve compatibility with established multidimensional modelling principles while significantly expanding their analytical expressiveness. The resulting Prediction-Oriented Dimensional Fact Model provides both graphical and formal representations, ensuring conceptual clarity and analytical rigor.

In addition to its theoretical contributions, the chapter demonstrated the practi-

cal applicability of the proposed framework through concrete implementations and empirical validation. The realization of Prediction-Oriented OLAP operators and the systematic construction of Analytical Base Tables illustrate how the framework can be operationalized using widely adopted technologies, including Python-based analytical environments and relational database systems. Both implementation paths support the end-to-end workflow of PO analysis, from data preparation and temporal structuring to feature extraction and target definition. The framework is further validated through the generation of dedicated datasets designed to test different temporal configurations, window dynamics, and prediction-oriented scenarios. These datasets enable controlled experimentation and facilitate reproducibility of results. Extensive testing procedures were conducted to verify the correctness of temporal window definitions, the consistency of derived features, and the integrity of prediction targets across both Python and SQL implementations. The presented case studies, particularly in the context of credit risk analysis, show that prediction-oriented analytical datasets and descriptive patterns can be produced using standard analytical tools, even in environments where the adoption of machine learning techniques is constrained.

Overall, this chapter establishes conceptual modelling as a powerful enabler for prediction-oriented analytics in multidimensional data environments. By making temporal assumptions and analytical intent explicit, the proposed framework bridges the gap between descriptive data infrastructures and predictive analytical needs. The concepts and mechanisms introduced here complement the log data modelling approach presented in the previous chapter, reinforcing the overarching thesis that explicit conceptual and temporal modelling is essential for advanced data analysis in complex, real-world settings.

Chapter 4

Conclusions

This thesis investigated how conceptual modelling can be extended to support advanced data analytics in environments characterized by structural heterogeneity, semantic ambiguity, and complex temporal relationships. The research addressed two complementary problems. The first concerns the explicit representation of heterogeneous system event logs, whose structures and semantics are frequently implicit, inconsistent, or dependent on specific technological formats. The second concerns the conceptual organization of multidimensional data for prediction-oriented analysis, where historical observations, data-availability delays, and future outcomes must be represented as distinct temporal roles.

The central argument developed throughout the thesis is that analytical capability depends not only on storage technologies, query languages, or predictive algorithms, but also on the explicit representation of the assumptions on which an analysis is based. These assumptions include the structure and meaning of data elements, the relationships among heterogeneous sources, the analytical grain at which facts are integrated, and the temporal conditions under which information becomes available. When such assumptions remain implicit, log data become difficult to interpret and integrate, while prediction-oriented datasets may inadvertently combine

information belonging to incompatible temporal stages.

The two contributions of the thesis address these problems at the conceptual modelling level. Although they concern different analytical settings, they share a common research principle: structural, semantic, and temporal assumptions should be represented explicitly before analytical operations are defined or executed. The first contribution applies this principle to heterogeneous log data; the second applies it to multidimensional data prepared for prediction-oriented analysis.

4.1 Synthesis of the Thesis

The first part of the thesis developed a conceptual modelling approach for system event logs. The proposed approach extends UML class-diagram constructs with log-specific stereotypes, field-set representations, nesting relationships, controlled values, semantic dependencies, and explicit distinctions among standard, extended, and custom fields. The Elastic Common Schema provides a shared vocabulary and structural reference, while custom extensions allow the model to represent domain-specific requirements that are not covered by the standard schema.

A central element of this contribution is the bidirectional relationship between the UML representation and JSON-based log structures. In a top-down modelling process, the conceptual model can be used to specify the intended structure of log records before their implementation. In a bottom-up process, existing JSON or JSONL logs can be abstracted into a UML representation that makes their fields, nesting structures, and dependencies easier to inspect and document. The accompanying StarUML extension provides implementation support for these transformations and for selected structural validation operations against ECS definitions.

The contribution of this part of the thesis is therefore not a new process-mining algorithm or a complete log-management platform. It is a conceptual and model-

driven layer for representing the structure and semantics of heterogeneous event data. This layer can support subsequent integration, governance, documentation, and analytical processing by making assumptions that would otherwise remain embedded in source formats or application-specific code explicit and inspectable.

The second part of the thesis developed the Prediction-Oriented Framework for multidimensional data analysis. The framework addresses the limitations of traditional multidimensional models when data are prepared for analytical tasks in which historical information is related to future outcomes. Its principal innovation is the explicit representation of three temporal roles: the Observation Window, the Waiting Window, and the Prediction Window. The Observation Window contains information admissible for feature construction; the Waiting Window represents latency, outcome maturation, or periods during which information must not be used; and the Prediction Window contains the future events or states used to define analytical outcomes.

These temporal roles are integrated into the multidimensional conceptual model through temporal bindings, explicit predictive dependencies, and Prediction-Oriented OLAP operators. The Prediction-Oriented Dimensional Fact Model extends the traditional DFM by making the temporal function of facts and dimensions visible at the schema level. This enables the analyst to specify not only which facts and dimensions participate in an analysis, but also when their information is admissible and how facts associated with different temporal roles may be combined.

The Python and SQL realizations demonstrate that the framework can be operationalized in both programmatic and database-centric analytical environments. The implementations construct Analytical Base Tables from window-specific data, generate features and target variables, and execute validation checks concerning temporal membership and precedence. Their role is to provide evidence of implementability, reproducibility, and temporal correctness for the evaluated configurations. They are

not presented as independent scientific contributions or as comparative evaluations of predictive algorithms.

4.2 Main Scientific Contributions

Table 4.1 summarizes the research problems, scientific artefacts, supporting evidence, and scope of the two principal contributions.

Table 4.1: Summary of the thesis contributions and supporting evidence.

Research problem	Scientific artefact	Supporting evidence	Scope
Heterogeneous and semantically implicit system event logs	UML-based conceptual model for log data, UML–JSON mappings, ECS-aligned constructs, custom extensions, and modelling-tool support	Metamodel definition, graphical notation, top-down and bottom-up case studies, bidirectional transformations, and prototype implementation	Representation, documentation, and structural validation of log data; not process discovery, conformance checking, or complete process-semantic modelling
Implicit temporal assumptions in prediction-oriented multidimensional analysis	Three-window Prediction-Oriented Framework, Prediction-Oriented OLAP operators, explicit temporal bindings and dependencies, and PO DFM	Formal definitions, credit-risk scenario, Python and SQL realizations, ABT construction, and temporal-validation tests	Conceptual and operational support for temporally valid data construction; not evidence of superior predictive accuracy or universal scalability

The first contribution provides a formal abstraction for log data that is independent of a particular logging technology while remaining compatible with widely adopted structured representations. Its scientific value lies in defining a modelling vocabulary through which heterogeneous log structures can be described, compared, validated, and transformed. The accompanying tool shows that the proposed map-

ping rules and structural constraints can be implemented within an existing UML modelling environment.

The second contribution provides a formal abstraction for the temporal organization of prediction-oriented multidimensional data. Its scientific value lies in integrating temporal roles and predictive dependencies directly into the conceptual model, rather than treating temporal separation as an implementation-specific convention expressed only through individual SQL predicates or data-preparation scripts. The framework makes the conditions under which features and outcomes are constructed visible, reviewable, and reproducible.

Taken together, the two contributions establish a progression from the representation of heterogeneous event data to the construction of temporally structured analytical datasets. This progression should not be interpreted as a fully implemented end-to-end pipeline in which the Chapter 2 artefact automatically produces the Chapter 3 analytical structures. Rather, the contributions are connected by a shared conceptual principle: data structure, semantics, and temporal intent should be made explicit before downstream analytical methods are applied.

4.3 Evaluation and Scope of the Evidence

The thesis follows a design-science perspective in which scientific contributions are expressed through artefacts and evaluated through a combination of formal definition, implementation, and case-based analysis. The evaluation evidence therefore operates at several levels.

At the conceptual level, the thesis defines the constructs, relationships, and constraints required to represent the two problem domains. For log modelling, this includes the metamodel, stereotypes, field sets, nesting mechanisms, enumerations, dependencies, and UML-JSON mappings. For prediction-oriented multidimensional

modelling, it includes the three-window temporal structure, fact bindings, predictive dependencies, PO OLAP operators, and the PO DFM.

At the implementation level, the StarUML extension demonstrates that the log modelling constructs and transformations can be operationalized. Similarly, the Python and SQL implementations demonstrate that the temporal semantics of the PO Framework can be translated into executable data-processing artefacts. These implementations show that Observation-Window features, Prediction-Window labels, and Analytical Base Tables can be generated while checking selected temporal constraints.

At the empirical level, the case studies illustrate the applicability of the proposed artefacts to representative analytical settings. The log-modelling examples examine heterogeneous structures in financial and network-oriented contexts. The PO Framework is evaluated primarily through a credit-risk scenario in which transactional, repayment, customer, and contextual data are combined under explicit temporal conditions.

The evidence supports the feasibility and internal consistency of the proposed modelling approaches within the examined settings. It does not establish that the artefacts are optimal for every domain, that every possible semantic constraint can be represented, or that downstream predictive models will necessarily achieve higher accuracy. In particular, the PO Framework is a conceptual and data-preparation framework rather than a predictive algorithm. Its contribution concerns the construction and validation of temporally structured analytical data, not the selection or comparative evaluation of a machine-learning model.

The expanded state-of-the-art analysis positions the contributions of this thesis as complementary to process mining and predictive process monitoring. Process mining is primarily concerned with the discovery, conformance, and enhancement of business processes using event traces. Predictive process monitoring extends

this perspective by applying statistical or machine-learning methods to complete historical traces and prefixes of running cases in order to predict future activities, outcomes, or performance indicators.

The log conceptual model developed in this thesis addresses an upstream problem. It provides mechanisms for making event structures, identifiers, timestamps, vocabularies, nesting relationships, and contextual information explicit before process-aware analytical techniques are applied. It does not discover a process model, identify process variants, or perform conformance checking.

The Prediction-Oriented Framework addresses a different but related layer. Rather than representing a running case as a trace prefix, it represents multidimensional facts, dimensions, analytical anchors, and window-specific roles. Its objective is to specify how historical observations, latency periods, and future outcomes should be separated and combined when constructing an analytical dataset. It therefore focuses on data organization and temporal validity rather than on the predictive algorithm itself.

The relationship among these areas is therefore complementary. Process-mining and predictive-monitoring methods may consume event data whose structure and semantics have been documented through the Chapter 2 model. Prediction-oriented or machine-learning workflows may consume Analytical Base Tables constructed according to the Chapter 3 framework. The evaluation of a fully integrated pipeline combining these artefacts remains a direction for future research.

4.4 Limitations

Several limitations delimit the claims that can be made from the research presented in this thesis.

First, the log conceptual model focuses principally on the representation of log

structure and selected semantic relationships. Although the proposed notation can represent identifiers, contextual attributes, field dependencies, and controlled values, it does not provide a complete formalization of business process behaviour. Process states, control-flow semantics, concurrency, compensation, and lifecycle constraints would require integration with process-aware modelling languages or process-mining techniques.

Second, the evaluation of the log-modelling approach is based on a limited number of representative case studies and on a research prototype. These examples demonstrate expressiveness and implementation feasibility, but broader evaluation across additional organizations, logging frameworks, and industrial domains would be required to establish general applicability. Structural validation against ECS also does not guarantee that a log design is semantically complete or operationally appropriate for every analytical purpose.

Third, the PO Framework is evaluated primarily through the credit-risk scenario and controlled or generated datasets. Although the formal definitions are domain-independent, additional empirical evaluation is needed in domains with different temporal characteristics, such as healthcare, telecommunications, manufacturing, cybersecurity, and customer-behaviour analysis.

Fourth, the implementation experiments evaluate data construction and temporal constraint enforcement rather than predictive-model performance. The thesis does not compare alternative classifiers, evaluate calibration, optimize hyperparameters, or establish that the proposed framework improves predictive accuracy relative to other data-preparation approaches. Such questions concern the downstream analytical lifecycle and require separate experimental designs.

Fifth, the reported implementation performance is specific to the evaluated configurations and technologies. The experiments do not constitute a general scalability guarantee for distributed, streaming, or continuously evolving industrial en-

vironments. Platform-specific behaviour, indexing strategies, query optimization, memory requirements, and data distribution would need to be examined before operational deployment.

Finally, the three-window structure represents temporal precedence through a single analytical anchor and explicitly separated intervals. Real information systems may involve late-arriving events, corrections, multiple clocks, transaction time, valid time, processing time, changing outcome definitions, and incomplete future observations. These issues require richer temporal and provenance mechanisms than those included in the current framework.

4.5 Future Research Directions

The limitations identified above define several directions for future research.

A first direction concerns tighter integration between the conceptual log model and process-mining environments. The model could be extended with explicit mappings to process cases, activities, lifecycle transitions, business objects, and process-instance identifiers. Such mappings would make it possible to investigate whether conceptual validation of log structures improves event correlation, process discovery, or conformance analysis.

A second direction is the integration of the PO Framework with predictive process monitoring. This would require mapping process prefixes, case attributes, and future process outcomes to fact schemes and temporal roles. The resulting research could examine how explicit Observation, Waiting, and Prediction Windows affect prefix construction, target definition, evaluation protocols, and information leakage.

A third direction concerns object-centric event data. Many operational processes involve several interacting business objects rather than a single case identifier. Fact constellations and explicit predictive dependencies provide a possible starting point

for relating object-centric event structures to multidimensional analytical models.

A fourth direction is broader empirical evaluation. The proposed artefacts should be applied to real-world datasets from multiple domains and compared with alternative modelling and data-preparation approaches. For the PO Framework, such evaluation should consider not only the correctness of ABT construction, but also the downstream effects of different temporal configurations on predictive performance, stability, calibration, and decision quality.

A fifth direction concerns richer temporal and provenance semantics. Future extensions could distinguish event time, processing time, valid time, and transaction time; represent corrections and late-arriving events; and record the provenance of individual features and labels. These capabilities would be particularly important in regulated and auditable analytical environments.

A sixth direction concerns model-driven automation. Higher-level specifications could be used to generate Python or SQL pipelines directly from a PO DFM, while modelling tools could identify incompatible aggregation grains, missing temporal bindings, invalid predictive dependencies, or potential leakage before execution. Similar automation could connect the Chapter 2 log models with schema registries, logging frameworks, and observability platforms.

Finally, future work should examine the governance implications of the proposed approaches. Explicit conceptual models can support documentation, accountability, reproducibility, and communication among data engineers, analysts, domain experts, and decision makers. Formalizing these governance functions would extend the contribution beyond data preparation toward the management of analytical assumptions throughout the complete data and model lifecycle.

4.6 Closing Remarks

This thesis demonstrates that conceptual modelling remains a relevant research instrument for advanced analytics. Its role is not limited to documenting static database structures; it can also make heterogeneous event semantics, analytical intent, temporal admissibility, and cross-source relationships explicit.

By providing a conceptual representation for heterogeneous log data and a prediction-oriented multidimensional framework with explicit temporal roles, the thesis contributes two complementary artefacts for structuring complex analytical information. Their principal value lies not in replacing process mining, predictive monitoring, machine learning, or existing data-warehouse technologies, but in making the assumptions on which those downstream techniques depend more transparent, inspectable, and reproducible.

The results indicate that explicit modelling of structure, semantics, and temporal conditions can reduce ambiguity between conceptual requirements and executable analytical workflows. Within the scope of the evaluation, the proposed artefacts provide a disciplined foundation for the preparation, integration, and validation of data used in subsequent process-aware, multidimensional, and machine-learning-based analyses.

Appendix A

StarUML Extension for Conceptual Modelling of Log Data

This appendix documents the software artefact associated with the conceptual modelling contribution presented in Chapter 2. The artefact is a StarUML extension that operationalises the log-specific UML constructs, structural rules, and UML–JSON mappings developed in the dissertation. Its role is to provide an executable realisation of the proposed modelling approach, support the reproduction of the modelling workflows discussed in subsection* 2.5, and make the implementation available for inspection, reuse, and further experimentation.

The extension does not introduce a separate conceptual model. Its behaviour is determined by the modelling constructs defined in Chapter 2, including the representation of log field sets, nested structures, core, extended, and custom attributes, controlled values, semantic dependencies, and ECS-aligned terminology. The software therefore establishes traceability between the conceptual specification and a concrete modelling environment.

Repository and Availability

The source code and supporting material are publicly available in the following GitHub repository:

`https://github.com/ravelina/phd\_staruml`

The repository provides the implementation of the StarUML extension together with the information required to install and execute it. It constitutes the software record accompanying the conceptual modelling contribution and enables the transformations and validation procedures described in subsection* 2.5 to be inspected independently of the dissertation text.

The principal functions implemented by the extension are:

- **Generation of JSON templates from UML models** Starting from a designated root element, the extension traverses the UML model and generates a structured JSON representation corresponding to the classes, attributes, associations, and nesting relationships defined in the diagram.
- **Generation of UML diagrams from JSON and JSONL data** Existing structured logs can be parsed to identify fields, nested objects, arrays, and recurring structures. These elements are represented as UML model elements, thereby supporting bottom-up abstraction and documentation of existing log formats.
- **Structural validation against ECS** The extension checks field names, data types, and nesting relationships against the ECS definitions available to the tool. The validation process is intended to identify structural inconsistencies between a conceptual model and the selected ECS vocabulary.

- **Synchronisation of ECS definitions** Updated ECS field-set definitions can be retrieved from the official ECS repository and incorporated into the extension’s local validation resources. This mechanism reduces the risk that validation is performed against obsolete schema definitions.
- **PlantUML-based diagram export** UML models can be exported through PlantUML in raster or vector formats. In particular, SVG export supports scalable representations suitable for technical documentation and academic publication.

These functions support both directions of the modelling process considered in the dissertation. The UML-to-JSON transformation supports the design of new logging structures from an explicit conceptual specification, whereas the JSON-to-UML transformation supports the abstraction, documentation, and analysis of existing structured logs. ECS validation connects both workflows to a shared field vocabulary and provides a mechanism for detecting selected structural inconsistencies.

Development Provenance

The software implementation was carried out by Marco Garosi as part of a Master’s thesis project conducted under the supervision of Prof. Carlo Combi and Evelina Rakhmetova. The conceptual requirements implemented by the extension derive from the modelling approach developed in this dissertation. In particular, the extension realises the proposed log-modelling constructs, the distinction among ECS and custom fields, the representation of nested field sets, and the bidirectional mappings between UML models and JSON-based log representations.

This attribution distinguishes the software-engineering implementation from the scientific contribution of the dissertation. The doctoral research defines and motivates the conceptual modelling approach, its metamodel, notation, semantics, trans-

formation requirements, and evaluation context. The Master’s thesis project provides a concrete software realisation of those requirements within StarUML.

Research Basis and Related Publications

The conceptual foundations underlying the extension and the earlier stages of its development are reported in the peer-reviewed publications [1, 2]. These publications describe the research problem, the initial modelling constructs, and the use of UML-based representations for structured log data. The dissertation extends and consolidates that work by providing a more complete metamodel, a systematic treatment of ECS-aligned and custom structures, explicit transformation rules, additional modelling scenarios, and an integrated account of the supporting software artefact.

Scope of the Software Artefact

The extension is provided as a research prototype intended to demonstrate the implementability of the proposed conceptual modelling approach and to support reproducible experimentation. The reported examples show that UML models can be transformed into JSON representations, that structured JSON and JSONL data can be abstracted into UML diagrams, and that selected structural properties can be checked against ECS definitions.

The artefact is not presented as a complete production-grade log-management system or as a comprehensive validator for every semantic constraint that may arise in an operational logging environment. In particular, successful structural validation does not by itself establish the semantic correctness, completeness, or operational suitability of a logging design. Those properties also depend on domain requirements, organisational conventions, runtime behaviour, and the intended downstream

analyses. Within these limits, the extension provides concrete implementation evidence for the feasibility of the model-driven approach developed in Chapter 2.

Appendix B

Detailed Implementation of the Prediction-Oriented Framework

This appendix provides the detailed implementation material supporting Section 3.7. It documents the Python and SQL artefacts used to operationalize the Prediction-Oriented Framework, including package structures, source-code listings, execution procedures, generated diagnostic material, validation tests, and implementation screenshots. The material is reported separately from the main scientific narrative in order to preserve reproducibility and traceability without obscuring the conceptual and empirical argument developed in Chapter 3. The appendix documents the realization of the framework; it does not introduce an additional scientific contribution.

The implementations follow the same temporal configuration and conceptual semantics described in Chapter 3. In particular, features are derived from the Observation Window, the Waiting Window is excluded from feature and target construction, and labels are derived from the Prediction Window. The high-level Python and SQL architectures remain in Figures 3.12 and 3.19; the present appendix provides the corresponding implementation-level detail.

The source code is available at the following repository:

`https://github.com/ravelina/phd\_podfm`

This framework was developed by *Evelina Rakhmetova* as part of the research activities of this PhD thesis. It implements the proposed Prediction-Oriented Dimensional Fact Model and supports both Python-based and SQL-based analytical workflows.

The framework directly reflects the conceptual foundations and formal definitions introduced in this thesis and serves as a concrete validation of the proposed prediction-oriented modelling approach.

B.1 Python Implementation

The Python implementation is organized as a modular analytical package. Its components provide temporal configuration, window-specific extraction, feature engineering, label generation, ABT construction, validation, visualization, and export. The following text documents the package structure and the execution flow in sufficient detail to reproduce the reported experiments.

Package Structure and Execution Modes

The package separates the core temporal logic from feature-generation, validation, utility, and notebook components. This organization allows the same conceptual workflow to be executed interactively in a Jupyter notebook or automatically through a standalone analysis script.

The Python implementation follows a modular design with a clear separation of concerns, supporting both interactive analysis and production deployment. The package architecture is organized into five main components, as shown in Listing 7.

Listing 7: Python Package Structure of the Prediction-Oriented Framework

```
po_framework/
  core/                                # Core logic
    window_config.py                  # Window configuration management
    window_extractor.py                # Temporal data extraction
    abt_builder.py                     # ABT construction pipeline
  features/                             # Feature engineering modules
    simple_aggregations.py             # Basic statistical features
    recency_frequency.py               # Temporal behavioural features
    trends.py                          # Trend and momentum analysis
    advanced_windows.py                # Tumbling/hopping/session windows
    economic_context.py                # Contextual dimension integration
    labels.py                          # Target variable generation
  validation/                           # Temporal constraint validation
    temporal_validator.py              # Comprehensive validation suite
  utils/                                # Utility functions
    data_loader.py                     # Data loading and preprocessing
    sample_data.py                     # Synthetic data generation
  notebooks/                            # Interactive analysis environment
    po_framework_visualization.ipynb  # Interactive notebook
    po_framework_analysis.py           # Standalone script
  figures/                              # Generated visualizations
```

This architecture ensures that each component can be used independently or composed into complete analytical pipelines, supporting both exploratory analysis and production deployment.

Interactive PO Analysis Environment

The Python implementation provides two complementary interfaces for analysis: an interactive Jupyter notebook for exploratory work and a standalone script for automated execution and figure generation.

Jupyter Notebook Interface

The `po_framework_visualization.ipynb` notebook provides an interactive environment for step-by-step analysis and experimentation. The notebook is organized into logical sections that mirror the analytical workflow:

1. *Data Generation*: Creates synthetic financial data with realistic patterns and distributions
2. *Window Configuration*: Sets up temporal parameters and validates window boundaries
3. *Temporal Extraction*: Demonstrates window-based data partitioning
4. *Multidimensional Analysis*: Constructs and analyses OW and PW data cubes
5. *Feature Engineering*: Generates comprehensive feature sets across seven categories
6. *Temporal Validation*: Performs rigorous constraint checking
7. *Visualization*: Creates publication-quality figures inline
8. *Results Analysis*: Provides detailed interpretation of findings

The notebook interface enables analysts to modify parameters, experiment with different configurations, and observe immediate results, facilitating both learning and research applications.

Automated Analysis Script

The `po_framework_analysis.py` script provides a complete, automated implementation that generates all analytical results and visualizations without user intervention. This script serves multiple purposes:

- *Reproducible Research*: Ensures consistent results across different execution environments
- *Batch Processing*: Enables automated execution in production or research pipelines
- *Figure Generation*: Automatically creates all eight publication-quality figures used in this thesis
- *Validation Testing*: Performs comprehensive temporal constraint validation with detailed reporting

The script execution produces the following outputs:

- *Console Output*: Detailed progress reporting and validation results
- *Figure Files*: Eight PNG files saved to `notebooks/figures/`
- *Data Files*: Intermediate datasets for further analysis
- *Summary Report*: Comprehensive analysis conclusions

Automated Figure Generation and Data Processing

A key strength of the Python implementation is its ability to automatically generate both test data and visualizations during code execution.

The `po_framework_analysis.py` script performs end-to-end processing that demonstrates the complete analytical workflow while producing visual evidence of the framework’s capabilities.

Data Generation Process: The script begins by generating synthetic financial data that mimics real-world patterns and distributions:

- *Customer Generation*: Creates customized number customers with diverse risk profiles (Low: 60%, Medium: 25%, High: 15%) distributed across four regions
- *Transaction Generation*: Produces credit card transactions with realistic amounts, dates, and payment behaviours
- *Loan Generation*: Creates 720 loan instalments with status patterns reflecting risk categories (default rates: Low 5%, Medium 15%, High 30%)
- *Economic Context*: Generates 24 economic indicator records across regions and time periods

Figure Generation Process: During execution, the script automatically creates eight publication-quality figures saved to `.../notebooks/figures/`:

1. `fig1_temporal_windows.png` - Three-Window Framework Structure (Figure 3.13);
2. `fig2_transaction_distribution.png` - Data Distribution Analysis (Figure 3.14);
3. `fig3_ow_data_cube.png` - OW Multidimensional Cube (Figure 3.15);
4. `fig4_pw_data_cube.png` - PW Multidimensional Cube (Figure 3.16);
5. `fig5_feature_distributions.png` - Statistical Feature Analysis (Figure 3.17);
6. `fig6_feature_correlations.png` - Feature-Target Relationships (Figure 3.18);
7. `fig7_validation_results.png` - Temporal Constraint Validation (Figure B.2);
8. `fig8_feature_categories.png` - Feature Engineering Overview (Figure B.1).

These figures serve dual purposes: they provide visual validation of the framework's correctness during development and testing, and they generate the visualizations integrated throughout this thesis that support the theoretical basis introduced in

earlier chapters. The automated generation ensures reproducibility and consistency across different execution environments.

Figure B.1 summarizes the comprehensive feature engineering capabilities, showing the distribution of features across seven categories generated during script execution. The framework generates 55 features across seven categories: Simple Aggregations (18 features, 32.7%), Recency & Frequency (7 features, 12.7%), Trends & Momentum (7 features, 12.7%), Tumbling Windows (6 features, 10.9%), Hopping Windows (4 features, 7.3%), Session Windows (7 features, 12.7%), and Economic Context (6 features, 10.9%). The left panel shows the absolute count by category, while the right panel displays the proportional distribution. This comprehensive feature set captures diverse temporal patterns from the OW data, enabling multifaceted analysis of behavioural dynamics.

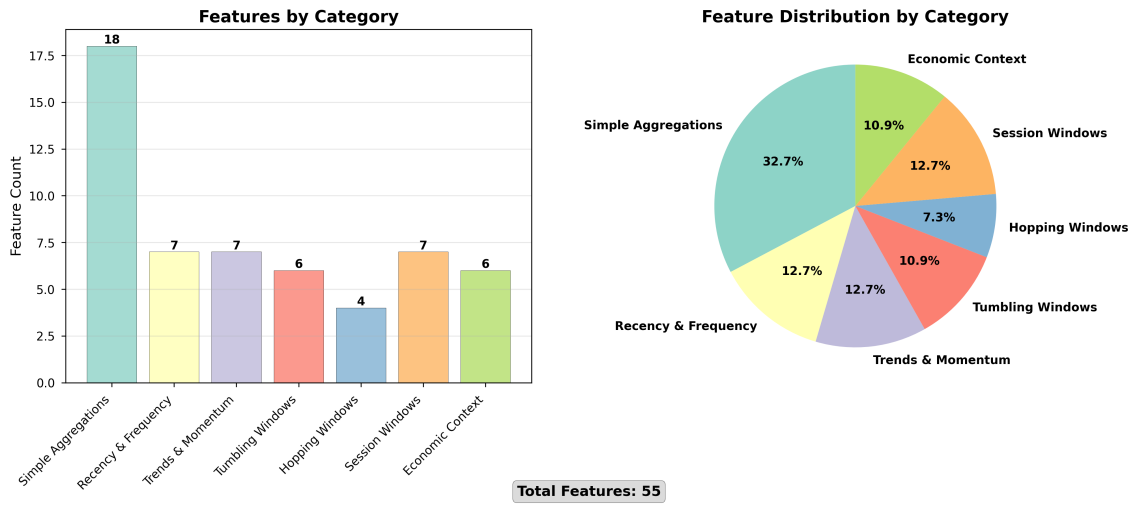


Figure B.1: Feature Engineering Categories and Distribution

The Python implementation demonstrates comprehensive feature engineering across seven categories, each implemented as a specialized module:

1. *Simple Aggregations* (18 features): Basic statistical measures including transaction counts, spending totals, averages, and standard deviations;

2. *Recency & Frequency* (7 features): Temporal behavioural patterns, for example, days since last transaction and transaction frequency metrics;
3. *Trends & Momentum* (7 features): Time-series analysis including spending trends, volatility measures, and momentum indicators;
4. *Tumbling Windows* (6 features): Fixed-size, non-overlapping temporal segments for periodic pattern analysis;
5. *Hopping Windows* (4 features): Overlapping temporal segments for rolling metric calculations;
6. *Session Windows* (7 features): Activity-based temporal grouping with configurable inactivity thresholds;
7. *Economic Context* (6 features): Integration of external economic indicators across temporal and spatial dimensions.

Temporal Configuration and Window Extraction

The `WindowConfig` class encapsulates all temporal parameters of the Prediction-Oriented Framework and exposes computed properties corresponding to the boundaries of the Observation, Waiting, and Prediction Windows (see Figure 3.13). This centralization ensures that all subsequent analytical steps are governed by a single, consistent temporal configuration.

Listing 8 illustrates the initialization of a window configuration and the derivation of the associated temporal intervals.

Listing 8: Window Configuration and Boundary Computation in Python

```
from datetime import datetime
from po_framework import WindowConfig
```

```
# Define temporal parameters
config = WindowConfig(
    t0=datetime(2024, 1, 1),      # Anchor point
    delta_ow=180,                 # Observation window (days)
    delta_ww=30,                 # Waiting window (days)
    delta_pw=90,                 # Prediction window (days)
    config_name='credit_risk_default'
)

# Access computed boundaries
print(f"OW: {config.ow_start} to {config.ow_end}")
print(f"WW: {config.ww_start} to {config.ww_end}")
print(f"PW: {config.pw_start} to {config.pw_end}")

# Output:
# OW: 2023-07-04 to 2024-01-01
# WW: 2024-01-01 to 2024-01-31
# PW: 2024-01-31 to 2024-05-01
```

At initialization time, the configuration object validates all temporal parameters, enforcing the constraints $\Delta_{OW} > 0$, $\Delta_{WW} \geq 0$, and $\Delta_{PW} > 0$. This prevents invalid window specifications and ensures the consistency of downstream processing.

Temporal Data Extraction

Window-based data extraction is handled by the `WindowExtractor` class, which applies explicit temporal constraints to input data frames in order to produce window-specific subsets. This component operationalizes the `SLICE` operator defined in the Prediction-Oriented OLAP operator set.

Listing 9 demonstrates how raw fact tables are filtered into Observation Window

data for feature computation and Prediction Window data for target labelling.

Listing 9: Window-Based Data Extraction Using WindowExtractor

```
from po_framework import WindowExtractor

extractor = WindowExtractor(config)

# Extract observation window data (features)
ow_data = extractor.extract_ow(
    df=creditcard_fact,
    date_column='Date'
)

# Extract prediction window data (labels)
pw_data = extractor.extract_pw(
    df=loan_fact,
    date_column='Date'
)

# Verify temporal separation
print(f"OW records: {len(ow_data)}")
print(f"PW records: {len(pw_data)}")
print(f"Latest OW date: {ow_data['Date'].max()}")
print(f"Earliest PW date: {pw_data['Date'].min()}")
```

In addition to filtering records by date, the extractor automatically augments the resulting data frames with metadata columns (e.g., `anchor_time`, `window_type`) to support downstream validation and traceability. All filtering operations rely on vectorized pandas primitives to ensure computational efficiency.

Feature Engineering Pipeline

Feature engineering is organized into a collection of specialized modules, each responsible for a distinct category of features. This modular design allows analysts to selectively enable or disable feature groups depending on the analytical task at hand.

Listing 10 illustrates the invocation of several feature modules on Observation Window data.

Listing 10: Invocation of Modular Feature Engineering Components

```
from po_framework.features import (
    SimpleAggregationFeatures,
    RecencyFrequencyFeatures,
    TrendFeatures,
    TumblingWindowFeatures
)

# Simple aggregations (18 features)
simple_features = SimpleAggregationFeatures()
agg_df = simple_features.compute(ow_data)

# Recency and frequency (7 features)
recency_features = RecencyFrequencyFeatures(config)
recency_df = recency_features.compute(ow_data)

# Trends and momentum (7 features)
trend_features = TrendFeatures()
trend_df = trend_features.compute(ow_data)

# Tumbling windows (6 features)
tumbling_features = TumblingWindowFeatures()
tumbling_df = tumbling_features.compute(ow_data)
```

Each feature module returns a data frame indexed by `CustomerID`, which enables straightforward integration during Analytical Base Table construction. Feature names are descriptive and follow consistent naming conventions to support interpretability and downstream model inspection.

Simple Aggregation Features

The `SimpleAggregationFeatures` module computes a set of standard statistical measures over the Observation Window, capturing overall activity levels and payment behaviour.

Listing 11 provides a simplified view of the implementation.

Listing 11: Simple Aggregation Feature Computation

```
class SimpleAggregationFeatures:
    """Compute basic statistical aggregations over OW data."""

    def compute(self, ow_data: pd.DataFrame) -> pd.DataFrame:
        """
        Generate 18 simple aggregation features, including:
        total_transactions, total_spending,
        avg_transaction_amount, stddev_transaction_amount,
        avg_credit_utilization, max_credit_utilization,
        missed_payments_count, and active_months.
        """
        features = ow_data.groupby('CustomerID').agg({
            'TransactionID': 'count',
            'TransactionAmount': ['sum', 'mean', 'std', 'min', 'max'],
            'TotalBalance': 'mean',
            'CreditLimit': 'mean',
            'Date': lambda x: x.dt.to_period('M').nunique()
        })
```

```
# Derived feature: credit utilization
features['avg_credit_utilization'] = (
    features['TotalBalance'] / features['CreditLimit']
)

# Count missed payments
features['missed_payments_count'] = (
    ow_data[ow_data['PaymentDue'] > ow_data['PaymentMade']]
    .groupby('CustomerID').size()
)

return features
```

Temporal Features: Recency and Frequency

Recency and frequency features capture both the timing and rate of customer behaviours within the Observation Window. These features are particularly important for distinguishing recent behavioural changes from long-term patterns.

Listing 12 shows the computation of a subset of these features.

Listing 12: Recency and Frequency Feature Computation

```
class RecencyFrequencyFeatures:
    """Compute recency and frequency features."""

    def __init__(self, config: WindowConfig):
        self.config = config

    def compute(self, ow_data: pd.DataFrame) -> pd.DataFrame:
        """Generate 7 recency and frequency features."""
        features = pd.DataFrame()

        # Recency: days since last transaction
```

```
last_transaction = ow_data.groupby('CustomerID')['Date'].max()
features['days_since_last_transaction'] = (
    (self.config.t0 - last_transaction).dt.days
)

# Recency: days since last missed payment
missed = ow_data[ow_data['PaymentDue'] > ow_data['PaymentMade']]
last_missed = missed.groupby('CustomerID')['Date'].max()
features['days_since_last_missed_payment'] = (
    (self.config.t0 - last_missed).dt.days
)

# Frequency: transactions per day
date_range = ow_data.groupby('CustomerID')['Date'].agg(['min', 'max'])
duration = (date_range['max'] - date_range['min']).dt.days
transaction_count = ow_data.groupby('CustomerID').size()
features['transactions_per_day'] = transaction_count / duration

return features
```

Advanced Windowing: Tumbling Windows

Tumbling windows partition the observation window into fixed-size, non-overlapping segments to capture periodic patterns:

```
class TumblingWindowFeatures:
    """Compute features using tumbling (non-overlapping) windows."""
    def compute(self, ow_data: pd.DataFrame) -> pd.DataFrame:
        """Generate 6 tumbling window features (monthly partitions)."""
        # Add month identifier
        ow_data['month'] = ow_data['Date'].dt.to_period('M')
        # Aggregate by customer and month
        monthly = ow_data.groupby(['CustomerID', 'month']).agg({
```

```

        'TransactionAmount': 'sum',
        'PaymentDue': lambda x: (x > ow_data.loc[x.index,
        ↪ 'PaymentMade']).sum()
    }).reset_index()
# Extract features
    features = pd.DataFrame()
# Most recent month spending
    latest_month = monthly.groupby('CustomerID')['month'].max()
    features['spending_most_recent_month'] = (
        monthly[monthly['month'] == latest_month]
        .set_index('CustomerID')['TransactionAmount']
    )
# Number of active months
    features['num_active_months'] = (
        monthly.groupby('CustomerID')['month'].nunique()
    )
# Average monthly spending
    features['avg_monthly_spending'] = (
        monthly.groupby('CustomerID')['TransactionAmount'].mean()
    )
    return features

```

Label Generation

Target variables are derived exclusively from the Prediction Window, ensuring strict temporal separation from feature computation. The responsibility for label extraction is encapsulated in the `LabelGenerator` component, which derives multiple target formulations from future observations.

Listing 13 illustrates the generation of target variables from Prediction Window data.

Listing 13: Label Generation from the Prediction Window

```
from po_framework.features import LabelGenerator

label_gen = LabelGenerator(config)
labels = label_gen.compute(pw_data)

# Generated labels include:
# - has_defaulted: binary classification target
# - has_payment_issue: multi-class target
# - total_default_amount: regression target
# - days_until_default: time-to-event target
# - delayed_payment_count: count target
```

The `LabelGenerator` handles missing Prediction Window data gracefully by assigning appropriate default values (e.g., `has_defaulted = 0` when no default event is observed). This design ensures that customers without future events remain valid training instances rather than being discarded.

Complete ABT Construction

The `ABTBuilder` component orchestrates the full prediction-oriented pipeline, integrating window extraction, feature computation, and label generation into a single reproducible process.

Listing 14 shows the construction of a complete Analytical Base Table using all available feature groups.

Listing 14: End-to-End Analytical Base Table Construction

```
from po_framework import ABTBuilder

builder = ABTBuilder(config)

# Build complete ABT with all feature types
```

```
abt = builder.build(
    creditcard_fact=creditcard_fact,
    loan_fact=loan_fact,
    customers=customers,
    economic_indicators=economic_indicators,
    include_advanced_windows=True # Tumbling, hopping, and session windows
)

# ABT structure:
# - CustomerID (entity identifier)
# - Demographics (age, region, risk category)
# - 50+ engineered features
# - 6 target variables
# - Temporal metadata (anchor_time, window boundaries)
# - Quality flags (missing_ow_data, missing_pw_data)

print(f"ABT shape: {abt.shape}")
print(f"Columns: {list(abt.columns)}")
```

The builder performs left joins to preserve all customers observed in the Observation Window, including those without Prediction Window records. In addition, quality flags are attached to each row to support downstream filtering and robustness checks. The implications of missing future observations must be assessed separately in downstream analysis.

Rolling ABT Generation

To support model training, validation, and temporal backtesting, the framework allows ABTs to be generated across multiple anchor points, producing a sequence of temporally consistent snapshots.

Listing 15 illustrates the generation of a rolling ABT using monthly anchor points.

Listing 15: Rolling ABT Generation Across Multiple Anchor Points

```
from po_framework.core.window_config import generate_rolling_configs
from datetime import datetime

# Generate monthly anchor configurations
configs = generate_rolling_configs(
    start_t0=datetime(2023, 7, 1),
    end_t0=datetime(2024, 3, 1),
    freq_days=30,
    delta_ow=180,
    delta_ww=30,
    delta_pw=90
)

# Build rolling ABT
rolling_abt = builder.build_rolling_abt(
    creditcard_fact=creditcard_fact,
    loan_fact=loan_fact,
    customers=customers,
    configs=configs
)

print(f"Rolling ABT shape: {rolling_abt.shape}")
print(f"Unique customers: {rolling_abt['CustomerID'].nunique()}")
print(f"Anchor points: {rolling_abt['anchor_time'].nunique()}")
```

Each customer may appear multiple times in the resulting dataset, once per anchor point, enabling supervised learning with temporal replication while preserving strict window separation.

Temporal Validation

To ensure the correctness of the implementation, all intermediate and final artifacts are subjected to explicit temporal validation. The `TemporalValidator` component verifies that window boundaries are respected and that no data violates ordering constraints.

Listing 16 presents an example validation run.

Listing 16: Temporal Validation Report

```
from po_framework.validation import TemporalValidator

validator = TemporalValidator(config)

# Run all validation checks
validator.print_validation_report(ow_data, pw_data, abt)

# Example output:
# =====
# TEMPORAL VALIDATION REPORT
# =====
# Configuration: credit_risk_default
# Anchor Point (t0): 2024-01-01
# -----
# OW Temporal Leakage          PASS          (0 violations)
# PW Temporal Precedence       PASS          (0 violations)
# Window Separation           PASS          (0 violations)
# Data Quality                 PASS          (0 violations)
# -----
# Summary: 4/4 checks passed
# =====
```

The validator performs four categories of checks:

- **OW Temporal Leakage:** Verifies no OW data occurs after t_0 ;

- **PW Temporal Precedence:** Ensures all PW data occurs after WW;
- **Window Separation:** Confirms proper temporal gaps between windows;
- **Data Quality:** Checks for missing or invalid data.

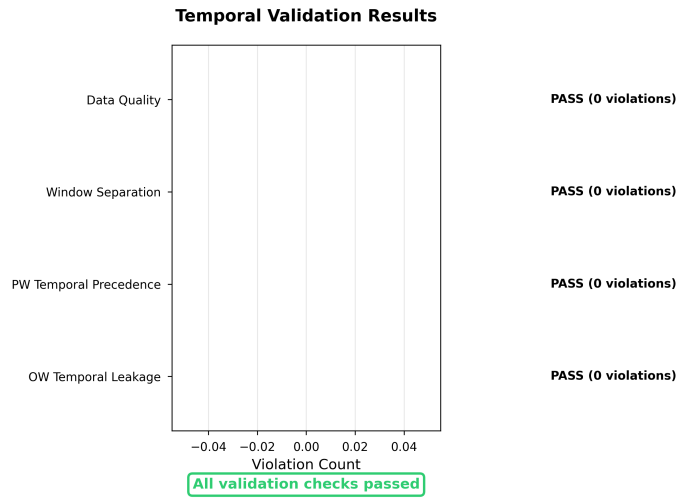


Figure B.2: Temporal Validation Results for 10,000 Customer Test

Figure B.2 presents the comprehensive validation results, confirming that the implementation maintains strict temporal precedence. The figure shows the results of four validation checks: OW Temporal Leakage (verifying no OW data occurs after t_0), PW Temporal Precedence (ensuring all PW data occurs after WW), Window Separation (confirming proper temporal gaps), and Data Quality (checking for missing or invalid data). All checks passed with zero violations, as indicated by the green bars and PASS status labels. The validation was performed on 44,335 OW records and 12,000 PW records. The summary confirms 100% validation pass rate, demonstrating the implementation's temporal correctness.

For the 10,000-customer configuration, the implementation evaluated four categories of checks: OW membership, PW precedence, window separation, and basic data quality. No violations were reported for the evaluated data, see Table B.1. This

result provides implementation evidence that the specified temporal constraints were enforced in this configuration; it does not establish the absence of every possible source of leakage in downstream modelling.

Validation Check	Violations	Status	Description
OW Temporal Leakage	0	PASS	All OW data before t_0
PW Temporal Precedence	0	PASS	All PW data after WW
Window Separation	0	PASS	Proper temporal gaps
Data Quality	0	PASS	No critical data issues
Overall	0	PASS	4/4 checks passed

Table B.1: Python temporal-validation results for the 10,000-customer configuration

Integration with Machine Learning

Once constructed, the Analytical Base Table can be consumed directly by standard machine learning frameworks. Because the ABT conforms to a flat, entity-centric schema with clearly separated features and targets, it integrates seamlessly with libraries (for example, scikit-learn) without requiring additional temporal handling. Listing 17 illustrates a typical workflow in which the ABT is split into training and test sets, used to train a classification model, and evaluated using standard performance metrics.

Listing 17: Integration of the ABT with a Machine Learning Pipeline

```

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Prepare feature matrix and target vector
feature_cols = [
    col for col in abt.columns

```

```
        if col not in ['CustomerID', 'has_defaulted',
                        'anchor_time', 'config_name']
    ]

X = abt[feature_cols]
y = abt['has_defaulted']

# Train-test split with class stratification
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Train classification model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Evaluate model
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))

# Inspect feature importance
feature_importance = pd.DataFrame({
    'feature': feature_cols,
    'importance': model.feature_importances_
}).sort_values('importance', ascending=False)

print("\nTop 10 Most Important Features:")
print(feature_importance.head(10))
```

This example demonstrates that all temporal considerations are resolved upstream during ABT construction. As a result, downstream models operate on a time-consistent dataset without needing to reason explicitly about window boundaries or

ordering constraints.

Data Export and Interoperability

To support integration with external analytical tools and production pipelines, the framework provides flexible data export capabilities. The ABT and its associated metadata can be serialized in multiple formats, enabling use in business intelligence platforms, batch machine learning workflows, and long-term storage systems.

Listing 18 illustrates several supported export options.

Listing 18: ABT Export and Interoperability Options

```
# Export to CSV (e.g., Excel, Power BI, Tableau)
abt.to_csv('abt_credit_risk.csv', index=False)

# Export to Parquet (efficient columnar format)
abt.to_parquet('abt_credit_risk.parquet', index=False)

# Export to Excel with multiple sheets
with pd.ExcelWriter('abt_analysis.xlsx') as writer:
    abt.to_excel(writer, sheet_name='ABT', index=False)
    feature_importance.to_excel(
        writer, sheet_name='Feature_Importance', index=False
    )
    validation_results.to_excel(
        writer, sheet_name='Validation', index=False
    )

# Export feature subset for machine learning pipelines
ml_dataset = abt[feature_cols + ['has_defaulted']]
ml_dataset.to_csv('ml_training_data.csv', index=False)
```

By decoupling feature construction from model training and deployment, the frame-

work supports a wide range of analytical workflows while preserving the temporal guarantees established during data preparation.

Performance Characteristics and Scalability

The Python implementation has been validated with comprehensive testing across multiple dataset sizes. Table B.2 presents performance benchmarks measured on a standard development workstation (Intel Core i7, 16GB RAM). Processing time increased with workload size in the reported experiments. The largest evaluated configuration contained 10,000 customers, 306,737 credit-card transactions, and 152,493 loan instalments. It produced an ABT with 9,802 rows and 75 columns, including 55 analytical features together with metadata and labels. The implementation processed this configuration in 110.6 seconds on the reported workstation.

These measurements demonstrate feasibility for the experimental workloads used in this study. They should not be interpreted as a general scalability guarantee, a formal complexity analysis, or evidence about arbitrary industrial workloads.

Dataset Size	Credit card transactions	Time	Throughput	Features
100 customers	3,152	5.2 s	606 rec/s	55
1,000 customers	31,520	18.4 s	1,713 rec/s	55
10,000 customers	306,737	110.6 s	4,151 rec/s	55

Table B.2: Observed processing performance of the Python realization for the evaluated workloads

The 10,000 customer test configuration employed the following parameters: $t_0 = 2024-01-01$, $\Delta t_{OW} = 180$ days, $\Delta t_{WW} = 30$ days, $\Delta t_{PW} = 90$ days. This configuration processed 306,737 credit card transactions and 152,493 loan instalments, extracting 44,335 OW records and 12,000 PW records, ultimately generating an ABT with 9,802 rows and 75 columns (55 features plus metadata and labels). All

temporal validation checks passed with zero violations, confirming strict temporal precedence and data quality.

Performance can be further optimized by:

- Setting `include_advanced_windows=False` for faster processing,
- Using fewer anchor points in rolling ABT generation,
- Processing data in batches for very large datasets,
- Leveraging Dask or PySpark for distributed processing.

B.2 SQL Implementation

The SQL implementation provides the database-centric realization of the Prediction-Oriented Framework summarized in Section 3.7.3. This section retains the complete executable material required to reproduce the reported workflow: the compact end-to-end DuckDB query, the metadata-driven configuration layer, window-specific views, modular feature and target views, ABT construction, rolling-anchor support, compatibility notes, and the DBeaver validation procedure. The high-level architecture and selected empirical outputs remain in Chapter 3, where they support the scientific argument without interrupting it with source-code-oriented detail.

Compact End-to-End Realization of the PO OLAP Operator Chain

This section presents the physical realization of the prediction-oriented OLAP (PO OLAP) operators introduced in the conceptual part of this chapter. Specifically, we demonstrate how the temporal semantics, window assignments, and operator composition defined in Section 3.5.1 and instantiated in the default-risk example of Section 3.5.4 are translated into an executable, database-centric workflow.

The goal of this section is not to introduce new modelling concepts, but to show how a formally defined PO DFM and its associated PO OLAP operators can be operationalized within a standard relational analytics environment. The implementation uses SQL (DuckDB dialect) to materialize the Observation Window (OW), Waiting Window (WW), and Prediction Window (PW), and to enforce temporal precedence during Analytical Base Table (ABT) construction.

The example presented here constitutes the direct physical counterpart of the conceptual default-risk scenario defined in Section 3.5.4. All temporal parameters, dataset bindings, and window roles (e.g., $\psi(\text{CreditCard_Fact}) = \text{OW}$ and

$\psi(\text{Loan_Fact}) = \text{PW}$) are reused without modification. What changes is the level of abstraction: conceptual operators such as **SLICE**, **AGGREGATE**, **LABEL**, and **JOIN(OW, PW)** are here instantiated as concrete SQL transformations.

This section therefore serves as a validation of the PO Framework’s implementability, illustrating how leakage-free feature engineering and target labelling can be enforced directly at query execution time, without requiring specialized storage engines or non-standard database primitives.

Feature Engineering (within the OW) As shown in Listing 19, the Analytical Base Table is constructed by combining feature engineering in the Observation Window with target labelling in the Prediction Window under strict temporal constraints. To illustrate this process concretely, the following listing presents the full SQL query in DuckDB syntax, using hardcoded dates in place of variables, that implements this integration and assigns a default label to each customer based on events observed in the Prediction Window.

Listing 19: ABT Construction with Feature Engineering and Label Generation in DuckDB

```
-- ABT Construction in DuckDB
-- Default Risk Example - PO Framework
-- t0 = 2024-01-01, OW = 180 days, WW = 30 days, PW = 90 days

WITH
-- Step 0: Construct 30-day rolling averages (hopping window features)
windowed_txns AS (
  SELECT
    f.CustomerID,
    d.Date,
    AVG(f.TransactionAmount) AS avg_txn_amount_30d
  FROM (
    SELECT DISTINCT Date
```

```

        FROM CreditCard_Fact
        WHERE Date >= DATE '2024-01-01' - INTERVAL 180 DAY
              AND Date < DATE '2024-01-01'
    ) d
    JOIN CreditCard_Fact f
        ON f.Date BETWEEN d.Date - INTERVAL 29 DAY AND d.Date
    GROUP BY f.CustomerID, d.Date
),

-- Step 1: Feature engineering in the Observation Window (OW)
features_ow AS (
    SELECT
        c.CustomerID,

        -- Overall credit utilization
        AVG(f.TotalBalance / NULLIF(f.CreditLimit, 0)) AS avg_utilization_ow,

        -- Count of missed payments
        SUM(CASE WHEN f.PaymentDue > f.PaymentMade THEN 1 ELSE 0 END)
            AS missed_payments_ow,

        -- Hopping window feature
        (
            SELECT AVG(avg_txn_amount_30d)
            FROM windowed_txns wt
            WHERE wt.CustomerID = c.CustomerID
                  AND wt.Date >= DATE '2024-01-01' - INTERVAL 180 DAY
                  AND wt.Date < DATE '2024-01-01'
        ) AS avg_hopping_30d_txn_amount,

        -- Tumbling windows
        SUM(f.TransactionAmount) FILTER (

```

```
        WHERE f.Date >= DATE '2024-01-01' - INTERVAL 30 DAY
    ) AS spending_last_30d,

    SUM(f.TransactionAmount) FILTER (
        WHERE f.Date >= DATE '2024-01-01' - INTERVAL 60 DAY
        AND f.Date < DATE '2024-01-01' - INTERVAL 30 DAY
    ) AS spending_prev_30d,

    -- Spending trend
    SUM(f.TransactionAmount) FILTER (
        WHERE f.Date >= DATE '2024-01-01' - INTERVAL 30 DAY
    ) -
    SUM(f.TransactionAmount) FILTER (
        WHERE f.Date >= DATE '2024-01-01' - INTERVAL 60 DAY
        AND f.Date < DATE '2024-01-01' - INTERVAL 30 DAY
    ) AS spending_trend_30d_vs_60d,

    -- Session-level feature
    COUNT(f.TransactionID) * 1.0 / COUNT(DISTINCT f.session_id)
    AS avg_txns_per_session_ow,

    -- Macroeconomic context
    MAX(e.UnemploymentRate) AS max_unemployment_ow

FROM Customer c
LEFT JOIN (
    -- Sessionization using inactivity gaps
    SELECT *,
        SUM(CASE WHEN is_new_session = 1 THEN 1 ELSE 0 END)
        OVER (PARTITION BY CustomerID ORDER BY Date) AS session_id
    FROM (
        SELECT *,
```

```

        CASE
            WHEN Date - LAG(Date)
                OVER (PARTITION BY CustomerID ORDER BY Date)
                    > INTERVAL 1 HOUR
            THEN 1 ELSE 0
        END AS is_new_session
    FROM CreditCard_Fact
    WHERE Date >= DATE '2024-01-01' - INTERVAL 180 DAY
        AND Date < DATE '2024-01-01'
    ) sessions
) f ON f.CustomerID = c.CustomerID
LEFT JOIN Economic_Indicators e
    ON c.Region = e.Region
    AND e.Date >= DATE '2024-01-01' - INTERVAL 180 DAY
    AND e.Date < DATE '2024-01-01'
GROUP BY c.CustomerID
),

-- Step 2: Target labelling in the Prediction Window (PW)
target_pw AS (
    SELECT
        CustomerID,
        MAX(1) AS has_defaulted_pw
    FROM Loan_Fact
    WHERE Status = 'Default'
        AND Date >= DATE '2024-01-01' + INTERVAL 30 DAY
        AND Date < DATE '2024-01-01' + INTERVAL 120 DAY
    GROUP BY CustomerID
)

-- Step 3: Assemble the final ABT
SELECT

```

```

f.CustomerID,
f.avg_utilization_ow,
f.missed_payments_ow,
f.avg_hopping_30d_txn_amount,
f.spending_last_30d,
f.spending_prev_30d,
f.spending_trend_30d_vs_60d,
f.avg_txns_per_session_ow,
f.max_unemployment_ow,
COALESCE(t.has_defaulted_pw, 0) AS default_label
FROM features_ow f
LEFT JOIN target_pw t ON f.CustomerID = t.CustomerID;

```

The query sequence is a concrete realization of the proposed PO OLAP operator chain:

$$\text{SLICE}(\text{OW}) \rightarrow \text{AGGREGATE} \rightarrow \text{LABEL}(\text{PW}) \rightarrow \text{JOIN}(\text{OW}, \text{PW}).$$

For the reported configuration, this composition makes the temporal direction and window membership explicit at query execution time. Unlike standard OLAP queries that perform post-hoc aggregation, the PO OLAP chain separates OW-derived features from PW-defined outcomes. The directed dependency is predictive and temporal; it is not a causal claim.

The query builds the ABT step by step, with each block corresponding to a well-defined logical transformation:

- **Step 0 (Hopping Window):** Creates a rolling 30-day window of average transaction amounts. Grouping by (*CustomerID*, *Date*) enables smooth day-level rolling aggregation.
- **Step 1 (Feature Engineering):** Aggregates behavioural and contextual indicators within the Observation Window. Session boundaries are detected

via `LAG()` over time-ordered transactions.

- **Step 2 (Target Labelling):** Generates a binary default label for customers with at least one default event in the Prediction Window.
- **Step 3 (Assembly):** Combines OW-derived features and PW labels into the final ABT, enforcing strict temporal separation.

This query exemplifies how the PO Framework operationalizes **temporal precedence**: all features depend exclusively on information available prior to the anchor time t_0 , enforcing the specified temporal predicates for the evaluated workflow.

The output produced by this compact query is shown and discussed in Figure 3.20 and Section 3.7.3.

While effective, the SQL formulation also highlights practical challenges: expressing complex temporal constructs (e.g., hopping windows or session-based aggregation) often leads to verbose and difficult-to-maintain queries. This observation motivates the need for higher-level abstractions or declarative extensions to better support prediction-oriented temporal analytics.

Modular SQL Architecture and Views

The SQL realization operationalizes the same PO semantics using relational tables, configuration metadata, window-specific views, modular feature views, a prediction-window label view, an ABT view, and dedicated validation queries. The implementation was developed and tested using DuckDB. Its logical design relies primarily on standard relational operations, including filtering, aggregation, joins, common table expressions, and analytical window functions.

The canonical execution sequence is:

SLICE(OW) → AGGREGATE → LABEL(PW) → JOIN(OW,PW) → VALIDATE.

Observation-window records are filtered and aggregated to construct features. Prediction-window records are used to define outcome labels. The resulting relations are joined at the analytical grain, after which validation queries test window membership and temporal precedence. The high-level data and validation flow is shown in Figure 3.19. The following material documents the corresponding SQL definitions and execution procedures.

Interactive SQL Testing with DBeaver

To validate the SQL implementation and demonstrate its practical usability, a comprehensive testing suite was developed for execution in DBeaver, a popular open-source database management tool. This testing approach serves multiple purposes: (1) it validates the correctness of the SQL implementation across different database systems, (2) it provides an accessible demonstration of the framework for practitioners without requiring programming expertise, and (3) it generates visual evidence of the framework’s temporal constraint enforcement and analytical capabilities.

The testing suite implements a systematic three-step workflow that mirrors real-world deployment scenarios:

Step 1: Database Schema Setup (`setup_database.sql`) This script creates the complete database schema including all fact tables, dimension tables, and supporting structures:

- *Fact Tables*: `CreditCard_Fact` (observation window source), `Loan_Fact` (prediction window source);
- *Dimension Tables*: `Customer`, `Economic_Indicators`;

- *Configuration Table*: `Window_Config` with temporal parameters;
- *Performance Optimization*: Indexes on (`CustomerID`, `Date`) for efficient temporal filtering;
- *Data Integrity*: Foreign key constraints and check constraints for data quality assurance.

Step 2: Test Data Generation (`generate_data.sql`) This script populates the database with synthetic but realistic financial data:

- *Customer Generation*: 100 customers with diverse risk profiles distributed across four regions (North, South, East, West);
- *Transaction Generation*: Approximately 4,000 credit card transactions with realistic temporal distribution and risk-based patterns;
- *Loan Generation*: 720 loan instalments with status patterns reflecting customer risk categories;
- *Economic Context*: 24 economic indicator records across regions and time periods;
- *Window Configuration*: Three predefined configurations for different analytical scenarios.

Step 3: Comprehensive Testing (`run_tests.sql`) This script executes seven comprehensive tests that validate all aspects of the PO Framework implementation:

1. *Window Configuration Verification*,
2. *Temporal View Creation and Statistics*,
3. *Feature Engineering Validation*,

4. *Label Generation Testing*,
5. *Complete ABT Construction*,
6. *Temporal Constraint Validation* (Critical),
7. *Sample Data Inspection*.

Each test in the suite is designed to validate specific aspects of the PO Framework implementation, with the generation of visual evidence through the DBeaver result interface. Following the execution of each test, the purpose of the test, the manner of its execution, and the expected outcomes are detailed.

Window Configuration Layer

The SQL implementation of the Prediction-Oriented Framework is grounded in a metadata-driven configuration layer that externalizes all temporal parameters. Rather than hard-coding window boundaries into analytical queries, the framework stores anchor points and window durations explicitly in a dedicated configuration table. This design decouples temporal specification from query logic and enables systematic reuse across analytical scenarios.

Listing 20 presents the definition of the `Window_Config` table together with an example configuration used in the financial credit risk case study.

Listing 20: Metadata-Driven Window Configuration Table

```
CREATE TABLE Window_Config (  
  config_id INTEGER PRIMARY KEY,           -- DuckDB: use INTEGER  
  config_name VARCHAR(100) NOT NULL,  
  t0 DATE NOT NULL,                       -- Anchor point  
  delta_ow INTEGER NOT NULL,              -- Observation Window (days)  
  delta_ww INTEGER NOT NULL,              -- Waiting Window (days)  
  delta_pw INTEGER NOT NULL,              -- Prediction Window (days)
```

```
description TEXT,  
CONSTRAINT positive_deltas CHECK (  
    delta_ow > 0 AND delta_ww >= 0 AND delta_pw > 0  
)  
);  
  
-- Example configuration for credit risk analysis  
INSERT INTO Window_Config (  
    config_id, config_name, t0,  
    delta_ow, delta_ww, delta_pw  
)  
VALUES (  
    1, 'credit_risk_default',  
    DATE '2024-01-01',  
    180, 30, 90  
)  
);
```

By externalizing temporal parameters into metadata, the configuration layer provides several key advantages:

- Temporal constraints and window definitions are centralized and fully auditable;
- Analytical queries become reusable across multiple window configurations;
- Analysts can experiment with alternative window lengths without modifying query logic;
- Temporal configurations can be versioned, documented, and compared systematically.

Temporal Window Views

Building on the configuration layer, the second architectural layer materializes the three-window structure through a set of window-specific SQL views. Each view filters fact data according to the temporal boundaries defined in `Window_Config`, thereby enforcing a disciplined separation between historical observations and future outcomes.

Listing 21 shows the definition of the Observation Window and Prediction Window views for the credit risk example.

Listing 21: Temporal Window Views for Observation and Prediction Windows

```
-- Observation Window: historical features
CREATE OR REPLACE VIEW CreditCard_Fact_OW AS
WITH config AS (
    SELECT t0, delta_ow
    FROM Window_Config
    WHERE config_name = 'credit_risk_default'
)
SELECT f.*, c.t0 AS anchor_time
FROM CreditCard_Fact f
CROSS JOIN config c
WHERE f.Date >= (c.t0 - c.delta_ow * INTERVAL '1 day')
    AND f.Date < c.t0;

-- Prediction Window: target outcomes
CREATE OR REPLACE VIEW Loan_Fact_PW AS
WITH config AS (
    SELECT t0, delta_ww, delta_pw
    FROM Window_Config
    WHERE config_name = 'credit_risk_default'
)
SELECT f.*
```

```
FROM Loan_Fact f
CROSS JOIN config c
WHERE f.Date >= (c.t0 + c.delta_ww * INTERVAL '1 day')
      AND f.Date < (c.t0 + (c.delta_ww + c.delta_pw) * INTERVAL '1 day');
```

These views implement the specified temporal separation: records in `CreditCard_Fact_OW` satisfy the OW predicate, whereas records in `Loan_Fact_PW` satisfy the PW predicate. The Waiting Window provides a deliberate temporal buffer between the two relations. The correctness of these predicates is evaluated separately by the validation queries.

Feature Engineering: Simple Aggregations

The first layer of feature engineering applies standard aggregation functions to Observation Window data. These features summarize customer behaviour over the entire historical window and form the baseline input for predictive modelling.

Listing 22 presents the construction of basic aggregation features derived from `CreditCard_Fact_OW`.

Listing 22: Simple Aggregation Features from the Observation Window

```
CREATE OR REPLACE VIEW Features_Simple_Aggregations AS
WITH ow_data AS (
    SELECT * FROM CreditCard_Fact_OW
)
SELECT
    CustomerID,
    anchor_time,

    -- Transaction volume features
    COUNT(*) AS total_transactions,
    SUM(TransactionAmount) AS total_spending,
```

```

AVG(TransactionAmount) AS avg_transaction_amount,
STDDEV(TransactionAmount) AS stddev_transaction_amount,

-- Credit utilization features
AVG(TotalBalance / NULLIF(CreditLimit, 0)) AS avg_credit_utilization,
MAX(TotalBalance / NULLIF(CreditLimit, 0)) AS max_credit_utilization,

-- Payment behaviour features
SUM(CASE WHEN PaymentDue > PaymentMade THEN 1 ELSE 0 END)
    AS missed_payments_count,
AVG(CASE
    WHEN PaymentDue > 0
    THEN PaymentMade::FLOAT / PaymentDue
    ELSE NULL
END) AS avg_payment_ratio
FROM ow_data
GROUP BY CustomerID, anchor_time;

```

Conceptually, the Observation Window data can be viewed as a temporally bounded multidimensional cube (Figure 3.15), enabling OLAP-style operations (roll-up, drill-down, and slicing) while preserving strict temporal alignment. This illustrates how the framework integrates classical multidimensional modelling with window-based analytical constraints.

Feature Engineering: Recency and Frequency

Beyond global aggregations, recency and frequency features capture the timing and intensity of customer activity within the Observation Window. These features are particularly important for distinguishing recent behavioural changes from longer-term patterns.

Listing 23 shows the computation of recency and frequency features using Observa-

tion Window data and the shared anchor time.

Listing 23: Recency and Frequency Feature Computation

```
CREATE OR REPLACE VIEW Features_Recency_Frequency AS
WITH ow_data AS (
    SELECT * FROM CreditCard_Fact_OW
),
config AS (
    SELECT t0
    FROM Window_Config
    WHERE config_name = 'credit_risk_default'
)
SELECT
    ow.CustomerID,
    c.t0 AS anchor_time,

    -- Recency features
    DATE_DIFF('day', MAX(ow.Date), c.t0)
        AS days_since_last_transaction,
    DATE_DIFF(
        'day',
        MAX(CASE WHEN ow.PaymentDue > ow.PaymentMade THEN ow.Date END),
        c.t0
    ) AS days_since_last_missed_payment,

    -- Frequency features
    COUNT(*) /
        NULLIF(DATE_DIFF('day', MIN(ow.Date), MAX(ow.Date)), 0)
        AS transactions_per_day,
    COUNT(*) /
        NULLIF(COUNT(DISTINCT DATE_TRUNC('month', ow.Date)), 0)
        AS transactions_per_active_month
FROM ow_data ow
```

```
CROSS JOIN config c
GROUP BY ow.CustomerID, c.t0;
```

Feature Engineering: Trends and Momentum

Trend and momentum features characterize how customer behaviour evolves within the Observation Window by comparing recent activity to earlier periods. Rather than summarizing the OW as a single aggregate, these features capture directional changes and variability over time, which are often indicative of emerging financial stress.

Listing 24 shows the computation of trend-based features using monthly aggregates derived from Observation Window data.

Listing 24: Trend and Momentum Feature Computation

```
CREATE OR REPLACE VIEW Features_Trends AS
WITH ow_data AS (
    SELECT * FROM CreditCard_Fact_OW
),
monthly_aggregates AS (
    SELECT
        CustomerID,
        DATE_TRUNC('month', Date) AS month,
        SUM(TransactionAmount) AS monthly_spending,
        AVG(TotalBalance / NULLIF(CreditLimit, 0))
            AS monthly_avg_utilization
    FROM ow_data
    GROUP BY CustomerID, DATE_TRUNC('month', Date)
),
trend_calc AS (
    SELECT
        CustomerID,
        month,
```

```
        monthly_spending,  
        LAG(monthly_spending, 1) OVER (  
            PARTITION BY CustomerID ORDER BY month  
        ) AS prev_month_spending  
FROM monthly_aggregates  
)  
SELECT  
    CustomerID,  
    AVG(monthly_spending - prev_month_spending)  
        AS avg_monthly_spending_change,  
    STDEV(monthly_spending) AS spending_volatility  
FROM trend_calc  
GROUP BY CustomerID;
```

These features capture both the direction (increase or decrease in spending) and the stability of customer behaviour across the Observation Window.

Advanced Windowing: Tumbling Windows

Tumbling windows partition the Observation Window into fixed-size, non-overlapping segments. This strategy enables the extraction of periodic patterns, like month-by-month behaviour, while preserving strict temporal alignment.

Listing 25 illustrates the computation of tumbling window features using monthly partitions of OW data.

Listing 25: Tumbling Window Feature Computation

```
CREATE OR REPLACE VIEW Features_Tumbling_Windows AS  
WITH ow_data AS (  
    SELECT * FROM CreditCard_Fact_OW  
)  
monthly_windows AS (  
    SELECT
```

```
CustomerID,
DATE_TRUNC('month', Date) AS window_month,
SUM(TransactionAmount) AS spending_in_window,
SUM(CASE WHEN PaymentDue > PaymentMade THEN 1 ELSE 0 END)
    AS missed_payments_in_window
FROM ow_data
GROUP BY CustomerID, DATE_TRUNC('month', Date)
)
SELECT
    CustomerID,
    MAX(
        CASE
            WHEN window_month = (
                SELECT MAX(window_month)
                FROM monthly_windows
                WHERE CustomerID = mw.CustomerID
            )
            THEN spending_in_window
        END
    ) AS spending_most_recent_month,
    COUNT(DISTINCT window_month) AS num_active_months,
    AVG(spending_in_window) AS avg_monthly_spending
FROM monthly_windows mw
GROUP BY CustomerID;
```

Tumbling windows allow the model to distinguish behaviour across successive periods (e.g., seasonal effects) without introducing overlap between windows.

Advanced Windowing: Hopping Windows

Hopping windows generalize tumbling windows by allowing overlapping segments, enabling the computation of rolling metrics that evolve smoothly over time. This

approach is particularly effective for capturing short-term dynamics, such as recent spending intensity.

Listing 26 shows the construction of a 30-day rolling spending feature with weekly hop intervals.

Listing 26: Hopping Window Feature Computation

```
WITH config AS (  
    SELECT t0, delta_ow  
    FROM Window_Config  
    WHERE config_name = 'credit_risk_default'  
) ,  
hop_points AS (  
    SELECT  
        c.t0 - INTERVAL ((c.delta_ow - (n * 7)) || ' days') AS hop_date  
    FROM config c  
    CROSS JOIN generate_series(  
        0 ,  
        (SELECT delta_ow / 7 FROM config)  
    ) AS t(n)  
) ,  
hopping_aggregates AS (  
    SELECT  
        ow.CustomerID ,  
        hp.hop_date ,  
        SUM(ow.TransactionAmount) AS spending_in_hop  
    FROM CreditCard_Fact_OW ow  
    CROSS JOIN hop_points hp  
    WHERE ow.Date >= (hp.hop_date - INTERVAL '30 days')  
        AND ow.Date < hp.hop_date  
    GROUP BY ow.CustomerID, hp.hop_date  
)  
SELECT
```

```
CustomerID,
MAX(
    CASE
        WHEN hop_date = (
            SELECT MAX(hop_date)
            FROM hopping_aggregates
            WHERE CustomerID = ha.CustomerID
        )
        THEN spending_in_hop
    END
) AS spending_last_30days
FROM hopping_aggregates ha
GROUP BY CustomerID;
```

Advanced Windowing: Session Windows

Session windows group events based on activity continuity rather than fixed time intervals. Sessions are defined by inactivity gaps, allowing the analysis to capture bursts of behaviour and engagement patterns that are not aligned to calendar boundaries.

Listing 27 illustrates session-based feature extraction using a 14-day inactivity threshold.

Listing 27: Session-Based Feature Computation

```
WITH ow_data AS (
    SELECT * FROM CreditCard_Fact_OW
),
session_boundaries AS (
    SELECT
        CustomerID,
        Date,
        TransactionAmount,
```

```

CASE
    WHEN Date - LAG(Date) OVER (
        PARTITION BY CustomerID ORDER BY Date
    ) > INTERVAL '14 days'
    OR LAG(Date) OVER (
        PARTITION BY CustomerID ORDER BY Date
    ) IS NULL
    THEN 1 ELSE 0
END AS is_session_start
FROM ow_data
),
sessions AS (
    SELECT
        CustomerID,
        Date,
        TransactionAmount,
        SUM(is_session_start) OVER (
            PARTITION BY CustomerID ORDER BY Date
        ) AS session_id
    FROM session_boundaries
),
session_aggregates AS (
    SELECT
        CustomerID,
        session_id,
        COUNT(*) AS transactions_in_session,
        SUM(TransactionAmount) AS spending_in_session,
        MAX(Date) - MIN(Date) AS session_duration_days
    FROM sessions
    GROUP BY CustomerID, session_id
)
SELECT

```

```
CustomerID,  
COUNT(*) AS total_sessions,  
AVG(transactions_in_session) AS avg_transactions_per_session,  
AVG(spending_in_session) AS avg_spending_per_session  
FROM session_aggregates  
GROUP BY CustomerID;
```

Contextual Dimension Integration

In addition to behavioural facts, the framework supports the integration of contextual dimensions whose role is informational rather than event-driven. Macroeconomic indicators, for example, provide exogenous context that may influence customer behaviour without being subject to the same temporal precedence constraints as transactional data.

Listing 28 illustrates the integration of economic indicators by attaching the most recent available values prior to the anchor time t_0 to each customer.

Listing 28: Contextual Feature Integration from Economic Indicators

```
CREATE OR REPLACE VIEW Features_Economic_Context AS  
WITH customer_region AS (  
    SELECT CustomerID, Region FROM Customer  
) ,  
config AS (  
    SELECT t0  
    FROM Window_Config  
    WHERE config_name = 'credit_risk_default'  
) ,  
economic_ow_end AS (  
    SELECT  
        cr.CustomerID,  
        ei.InflationRate AS inflation_rate_ow_end,
```

```
        ei.UnemploymentRate AS unemployment_rate_ow_end
FROM customer_region cr
CROSS JOIN config c
LEFT JOIN Economic_Indicators ei
    ON ei.Region = cr.Region
    AND ei.Date = (
        SELECT MAX(Date)
        FROM Economic_Indicators
        WHERE Region = cr.Region
        AND Date < c.t0
    )
)
SELECT * FROM economic_ow_end;
```

By anchoring contextual attributes to the end of the Observation Window, the framework ensures that only information available at prediction time is used, while allowing such dimensions to enrich both feature interpretation and model performance.

Label Generation from the Prediction Window

Target variables are derived from the Prediction-Window view, thereby applying the specified separation from OW-based feature computation. As illustrated in Figure 3.16, the PW relation captures future outcomes after the waiting period in the evaluated workflow.

Listing 29 presents the construction of multiple target variables from Prediction Window data.

Listing 29: Target Variable Generation from the Prediction Window

```
CREATE OR REPLACE VIEW Labels_Default_Prediction AS
WITH pw_data AS (
```

```
SELECT * FROM Loan_Fact_PW
)
SELECT
    CustomerID,
    pw_start AS prediction_window_start,
    pw_end AS prediction_window_end,

    -- Binary classification target
    MAX(CASE WHEN Status = 'Default' THEN 1 ELSE 0 END)
        AS has_defaulted,

    -- Time-to-event target
    DATE_DIFF(
        'day',
        pw_start,
        MIN(CASE WHEN Status = 'Default' THEN Date END)
    ) AS days_until_default,

    -- Regression target
    SUM(
        CASE WHEN Status = 'Default'
            THEN InstallmentDue - InstallmentPaid
            ELSE 0
        END
    ) AS total_default_amount
FROM pw_data
GROUP BY CustomerID, pw_start, pw_end;
```

This design supports multiple predictive tasks, including classification, regression, and time-to-event modelling, while maintaining strict separation between explanatory variables and outcomes.

Complete ABT Construction

The final Analytical Base Table integrates all engineered features with target labels through a sequence of left joins. Customers observed in the Observation Window are preserved even when no Prediction Window events are present.

Listing 30 shows the assembly of the complete ABT.

Listing 30: Complete Analytical Base Table Construction

```
CREATE OR REPLACE VIEW ABT_Credit_Risk_Complete AS
WITH customers_in_ow AS (
    SELECT DISTINCT CustomerID, anchor_time
    FROM CreditCard_Fact_OW
),
features_combined AS (
    SELECT
        c.CustomerID,
        c.anchor_time,
        cust.Region,
        cust.RiskCategory,
        sa.total_transactions,
        sa.total_spending,
        sa.avg_credit_utilization,
        sa.missed_payments_count,
        rf.days_since_last_transaction,
        tr.spending_volatility,
        tw.spending_most_recent_month,
        hw.spending_last_30days,
        sw.total_sessions,
        ec.inflation_rate_ow_end
    FROM customers_in_ow c
    LEFT JOIN Customer cust
        ON c.CustomerID = cust.CustomerID
    LEFT JOIN Features_Simple_Aggregations sa
```

```
        ON c.CustomerID = sa.CustomerID
LEFT JOIN Features_Recency_Frequency rf
        ON c.CustomerID = rf.CustomerID
LEFT JOIN Features_Trends tr
        ON c.CustomerID = tr.CustomerID
LEFT JOIN Features_Tumbling_Windows tw
        ON c.CustomerID = tw.CustomerID
LEFT JOIN Features_Hopping_Windows hw
        ON c.CustomerID = hw.CustomerID
LEFT JOIN Features_Session_Windows sw
        ON c.CustomerID = sw.CustomerID
LEFT JOIN Features_Economic_Context ec
        ON c.CustomerID = ec.CustomerID
)
SELECT
    f.*,
    COALESCE(l.has_defaulted, 0) AS target_has_defaulted,
    l.days_until_default,
    COALESCE(l.total_default_amount, 0) AS target_default_amount,
    l.prediction_window_start,
    l.prediction_window_end
FROM features_combined f
LEFT JOIN Labels_Default_Prediction l
    ON f.CustomerID = l.CustomerID;
```

Temporal Constraint Validation

To ensure the correctness of the implementation, explicit validation queries are used to verify that all temporal constraints are respected and that no leakage occurs across window boundaries.

Listing 31 illustrates a representative validation check for Observation Window leak-

age, together with the construction of a composite validation report.

Listing 31: Temporal Constraint Validation Queries

```
-- Check for temporal leakage (OW data after t0)
CREATE OR REPLACE VIEW Validation_OW_Temporal_Leakage AS
WITH config AS (
    SELECT t0
    FROM Window_Config
    WHERE config_name = 'credit_risk_default'
)
SELECT
    'OW Temporal Leakage' AS validation_check,
    COUNT(*) AS violation_count,
    CASE WHEN COUNT(*) = 0 THEN 'PASS' ELSE 'FAIL' END AS status
FROM CreditCard_Fact f
CROSS JOIN config c
WHERE f.Date >= c.t0
    AND EXISTS (
        SELECT 1
        FROM CreditCard_Fact_OW ow
        WHERE ow.TransactionID = f.TransactionID
    );

-- Consolidated validation report
CREATE OR REPLACE VIEW Validation_Report AS
SELECT * FROM Validation_OW_Temporal_Leakage
UNION ALL
SELECT * FROM Validation_PW_Temporal_Precedence
UNION ALL
SELECT * FROM Validation_Window_Separation;
```

Rolling ABT Generation

To support model training and longitudinal evaluation, the framework allows the anchor point t_0 to vary over time, producing multiple temporally aligned snapshots of the Analytical Base Table. Each snapshot corresponds to a distinct analytical “present” and is associated with its own Observation, Waiting, and Prediction Windows. This mechanism enables the construction of rolling training datasets while preserving strict temporal separation for each snapshot.

Listing 32 illustrates the generation of a sequence of monthly anchor points together with fixed window durations.

Listing 32: Generation of Rolling Anchor Points for ABT Construction

```
-- Generate monthly anchor points
CREATE TABLE Anchor_Points (
    anchor_id INTEGER PRIMARY KEY,
    t0 DATE NOT NULL,
    delta_ow INTEGER NOT NULL,
    delta_ww INTEGER NOT NULL,
    delta_pw INTEGER NOT NULL,
    cohort_name VARCHAR(100)
);

INSERT INTO Anchor_Points
SELECT
    n AS anchor_id,
    DATE '2023-07-01' + INTERVAL ((n || ' months')) AS t0,
    180 AS delta_ow,
    30 AS delta_ww,
    90 AS delta_pw,
    'cohort_' || STRFTIME(
        DATE '2023-07-01' + INTERVAL ((n || ' months')),
        '%Y_%m'
```

```
    ) AS cohort_name
FROM generate_series(0, 11) AS t(n);

-- Rolling ABT across all anchor points
CREATE OR REPLACE VIEW ABT_Rolling_All_Anchors AS
-- [Implementation combines feature and label generation for each anchor point]
-- See supplementary SQL material for the complete definition
;
```

Each anchor point defines an independent analytical snapshot, ensuring that features and targets remain temporally consistent within each cohort while supporting time-aware model training and evaluation.

Implementation Summary

The SQL implementation presented in this section demonstrates several key properties of the Prediction-Oriented Framework when realized in a database-centric environment:

- *Modularity*: Each feature category is implemented as an independent view, allowing feature definitions to be developed, tested, and maintained in isolation without impacting the remainder of the pipeline;
- *Reusability*: A metadata-driven configuration layer enables the same query templates to be reused across multiple temporal window specifications and analytical scenarios;
- *Temporal Explicitness*: Window boundaries anchored at t_0 make the admissible OW and PW records explicit and allow the corresponding membership and precedence predicates to be checked automatically;

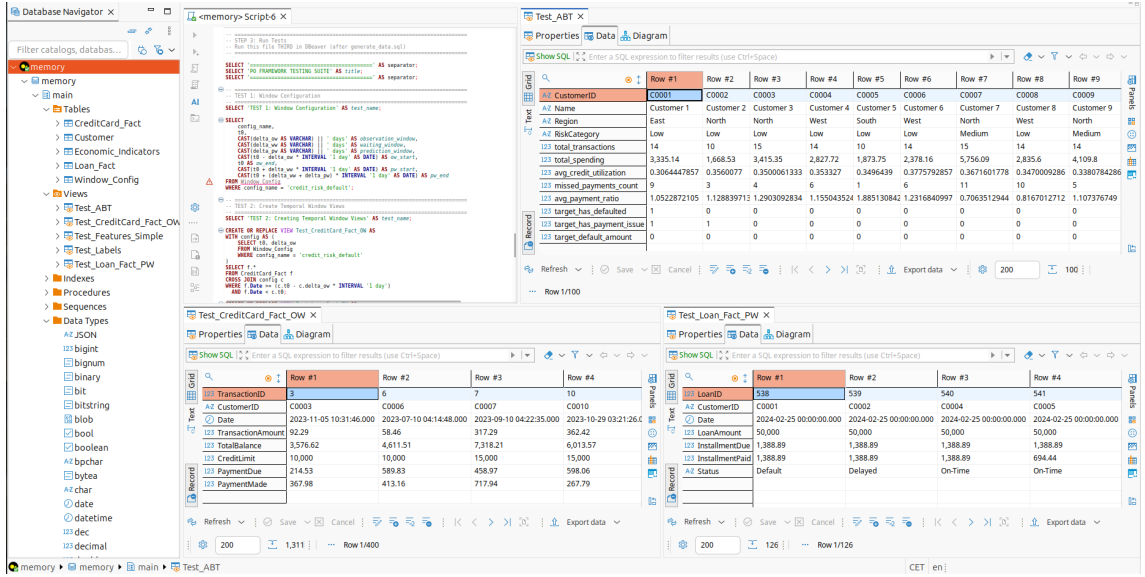


Figure B.3: Complete structure of the SQL-based Prediction-Oriented Framework implementation in DuckDB

- *Performance Engineering*: Views, optional materialization, and indexing provide implementation options whose effectiveness must be assessed for each target workload and database platform;
- *Auditability*: All temporal assumptions, window definitions, and analytical transformations are explicit, traceable, and reproducible;
- *Extensibility*: Additional feature categories or contextual dimensions can be incorporated without modification to existing components.

The complete SQL implementation, including automatically generated temporal tables, window-specific fact views, modular feature definitions, and prediction-oriented operator templates, is provided in the supplementary materials accompanying this thesis. An overview of all tables and temporal views is shown in Figure B.3, which illustrates the full structure of the SQL-based PO Framework implementation.

Taken together, these components form a fully operational and temporally disciplined analytical environment. The resulting window-partitioned datasets and An-

alytical Base Tables can be explored directly through SQL clients or integrated into Business Intelligence platforms, supporting interactive dashboards, reporting, and downstream predictive model deployment in tools such as Power BI or Tableau.

Database Compatibility Notes

The SQL examples presented in this chapter use DuckDB-compatible syntax, selected for its modern analytical capabilities, ease of experimentation, and performance characteristics. Key elements include:

- *Interval arithmetic*: expressions such as `n * INTERVAL '1 day'` for parameterized temporal calculations;
- *Date functions*: `DATEDIFF('day', start, end)` for computing temporal distances;
- *String concatenation*: `CAST(n AS VARCHAR) || ' days'` for constructing dynamic expressions;
- *Series generation*: `generate_series` for producing sequences of anchor points or synthetic test data;
- *Aggregate functions*: standard SQL aggregates, including `STDDEV_POP()`.

For deployment on PostgreSQL, only minor syntactic adaptations are required:

- `generate_series` is natively supported in PostgreSQL;
- DuckDB `DATEDIFF('day', d1, d2)` can be replaced by `d2 - d1`;
- `SERIAL` may be used in place of `INTEGER` for auto-incrementing primary keys;
- Interval arithmetic (`n * INTERVAL '1 day'`) is supported in both systems.

Both DuckDB and PostgreSQL therefore support the core constructs required by the Prediction-Oriented Framework, including window-based filtering, common table expressions, window functions, and materialized views, enabling a smooth transition from experimental environments to potential deployment after platform-specific validation and performance evaluations.

Testing Architecture

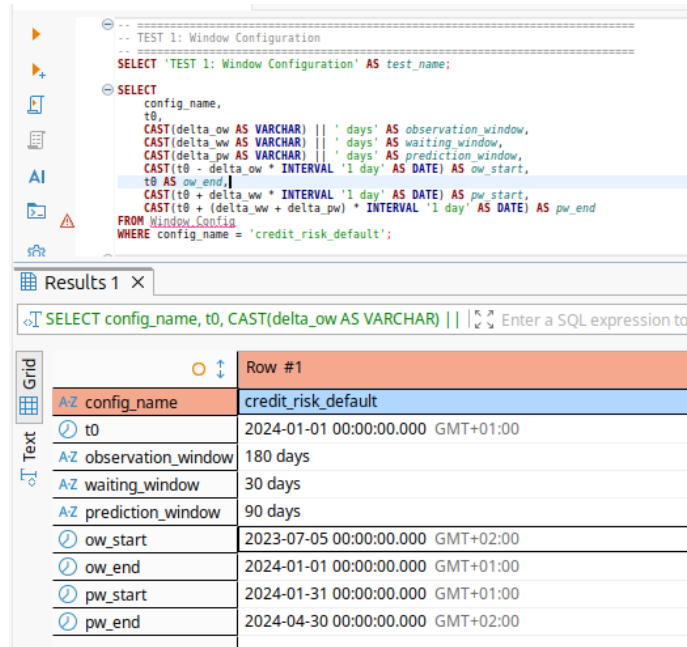
The testing suite consists of three sequential SQL scripts that together implement a complete end-to-end validation workflow:

1. *Database Setup* (`setup_database.sql`): Creates the complete schema, including fact tables (`CreditCard_Fact`, `Loan_Fact`), dimension tables (`Customer`, `Region`), contextual tables (`Economic_Indicators`), and the configuration table (`Window_Config`). All existing objects are dropped prior to creation to ensure a clean testing environment.
2. *Data Generation* (`generate_data.sql`): Populates the database with synthetic but realistic data. The validation dataset comprises 100 customers, approximately 4,000 credit card transactions, 720 loan instalments, and 24 economic indicator records, distributed across the OW, WW, and PW according to the configured temporal boundaries.
3. *Test Execution* (`run_tests.sql`): Executes a sequence of seven validation tests that assess window configuration, temporal view correctness, feature engineering, label extraction, ABT construction, and temporal constraint enforcement.

Test 1: Window Configuration Verification This test queries the `Window_Config` table to display the configured temporal parameters and computed

window boundaries. For the credit risk scenario, it confirms:

- Anchor point: $t_0 = 2024-01-01$,
- Observation Window: 180 days (2023-07-04 to 2024-01-01),
- Waiting Window: 30 days (2024-01-01 to 2024-01-31),
- Prediction Window: 90 days (2024-01-31 to 2024-05-01).



The screenshot shows the DBeaver SQL editor with a query titled 'TEST 1: Window Configuration'. The query selects various temporal parameters for a 'credit_risk_default' configuration. Below the editor, the 'Results 1' tab displays the query output in a grid format.

	Row #1
config_name	credit_risk_default
t0	2024-01-01 00:00:00.000 GMT+01:00
observation_window	180 days
waiting_window	30 days
prediction_window	90 days
ow_start	2023-07-05 00:00:00.000 GMT+02:00
ow_end	2024-01-01 00:00:00.000 GMT+01:00
pw_start	2024-01-31 00:00:00.000 GMT+01:00
pw_end	2024-04-30 00:00:00.000 GMT+02:00

Figure B.4: DBeaver Test 1: Window Configuration Display

Figure B.4 confirms that all temporal boundaries are correctly derived from the base parameters $(t_0, \Delta t_{OW}, \Delta t_{WW}, \Delta t_{PW})$, demonstrating the effectiveness of the metadata-driven configuration approach.

Test 2: Temporal Window View Creation

This test validates the creation of the `Test_CreditCard_Fact_OW` and `Test_Loan_Fact_PW` views by applying temporal filters to the base fact tables. Statistics on record counts, customer coverage, and date ranges confirm that:

- OW records lie strictly in $[t_0 - \Delta t_{OW}, t_0)$;
- PW records lie strictly in $[t_0 + \Delta t_{WW}, t_0 + \Delta t_{WW} + \Delta t_{PW})$;
- No temporal overlap exists between windows;
- Data coverage is appropriate for both feature and label generation.

Figure 3.21 provides visual confirmation of correct temporal segmentation.

Test 3: Feature Engineering Validation

This test validates the implementation of the simple aggregation feature category by creating the `Test_Features_Simple` view, which computes customer-level statistical summaries exclusively from Observation Window (OW) data. In total, eight behavioural feature measures are generated per customer:

- *Transaction volume*: `total_transactions`;
- *Spending metrics*: `total_spending`, `avg_transaction_amount`, `stddev_transaction_amount`;
- *Credit utilization*: `avg_credit_utilization`, `max_credit_utilization`;
- *Payment behaviour*: `missed_payments_count`, `avg_payment_ratio`.

To facilitate inspection, the test displays the top 10 customers ranked by missed payments and total spending. This ranking highlights the framework's ability to surface meaningful behavioural patterns from historical transactional data. The test confirms that raw OW-level transactions are correctly transformed into stable, customer-level behavioural descriptors, capturing multiple dimensions of credit behaviour such as activity intensity, spending magnitude, utilization pressure, and payment discipline. These features constitute the foundational input for subsequent predictive modelling stages.

Figure 3.22 presents the resulting aggregated feature values at the customer level. The results validate the correct implementation of the **AGGREGATE** operator within the PO OLAP framework, demonstrating that feature engineering is performed strictly within the Observation Window (OW) and fully complies with the temporal scoping and precedence constraints defined by the PO semantics.

Test 4: Label Generation Validation

This test validates target variable extraction by creating the **Test_Labels** view, which derives outcome labels exclusively from Prediction Window (PW) data. For each customer with loan activity in the PW, the following targets are generated:

- *Binary classification target:* **has_defaulted** (1 if any default occurred, 0 otherwise);
- *Extended classification target:* **has_payment_issue** (capturing both defaults and delayed payments);
- *Regression target:* **total_default_amount** (sum of unpaid amounts);
- *Count metric:* **delayed_payment_count**.

The test computes summary statistics, including the number of customers with PW activity, the number of observed defaults, and the overall default rate. These statistics provide context for the predictive task while confirming that labels are derived solely from future observations. Crucially, this test demonstrates that outcome variables are identified only after the temporal separation enforced by the Waiting Window (WW), ensuring that no PW information can influence OW-derived features.

Figure 3.23 displays the label generation summary statistics, confirming correct extraction of outcome variables and strict temporal separation between features and labels.

Test 5: Complete ABT Construction

This test evaluates the construction of the Analytical Base Table (ABT) by creating the `Test_ABT` view. The ABT is formed by joining the feature view (`Test_Features_Simple`) with the label view (`Test_Labels`) using a FULL OUTER JOIN, thereby preserving all customers appearing in either window. The test computes the following statistics:

- Total rows in the ABT (union of feature and label populations);
- Rows with features (customers active in the OW);
- Rows with labels (customers active in the PW);
- Complete rows (customers with both features and labels).

This test validates the `JOIN(OW,PW)` operation within the PO OLAP framework and demonstrates the framework’s ability to integrate historical behavioural features with future outcome labels into a unified analytical dataset.

Figure 3.24 reports 100 materialized customer rows and non-null feature and label fields for all rows. This result must be interpreted together with Test 4: only 79 customers have observed PW loan activity. The remaining label values arise from the implementation’s COALESCE-based zero-default convention. Consequently, the completeness statistic documents the shape of the materialized ABT; it does not establish that an outcome was directly observed for every customer.

Test 6: Temporal Constraint Validation

This critical test verifies that strict temporal precedence is maintained throughout the pipeline by checking for two classes of violations:

- **OW Temporal Leakage:** Detection of any OW records with timestamps on or after t_0 , indicating leakage of present or future information into the observation period;

- **PW Temporal Precedence:** Detection of any PW records with timestamps prior to $t_0 + \Delta t_{WW}$, violating the temporal buffer imposed by the waiting window.

For a correct implementation, both checks must return zero violations and a PASS status. Figure 3.25 reports zero violations for the two implemented predicates. This result shows that the OW-membership and PW-precedence conditions were satisfied by the controlled dataset. The test does not assess missing-data mechanisms or every possible form of downstream information leakage.

Test 7: Sample ABT Data Inspection

The final test performs a qualitative inspection of the `Test_ABT` view by displaying a sample of complete records. Results are ordered by default status and missed payments to emphasize contrasts between different customer risk profiles. This inspection provides a concrete illustration of the framework’s output: each row represents a customer with features derived from historical OW data paired with outcome labels observed in the PW.

Figure 3.26 shows representative ABT records, illustrating the integration of transactional behaviour (activity, spending, utilization, missed payments) with future outcomes (default status). The diversity of observed patterns demonstrates how the framework captures both low-risk customers with stable repayment behaviour and high-risk customers exhibiting missed payments and eventual defaults.

Testing Results and Validation

The DBeaver testing suite was executed successfully on DuckDB, validating all aspects of the SQL-based implementation. The principal results are as follows:

- **Window Configuration:** All temporal boundaries were computed correctly from the base parameters;

- **Temporal Views:** OW and PW data were correctly partitioned for all 100 customers with no overlap;
- **Feature Engineering:** Eight behavioural feature measures per customer were generated from OW data, capturing transaction volume, spending patterns, credit utilization, and payment behaviour;
- **Label Generation:** Binary and continuous outcome variables were extracted from PW data; 79 customers had PW activity and 12 had at least one default, corresponding to an observed default rate of 15.19%;
- **ABT Construction:** Features and labels were successfully integrated, materializing 100 rows with non-null feature and label fields under the implemented join and zero-default convention;
- **Temporal Validation:** **Zero violations** were detected for both OW leakage and PW precedence checks.

The zero counts provide implementation evidence that the two specified temporal predicates were enforced for the evaluated dataset and configuration. They do not establish the absence of every possible source of leakage in downstream analytical or predictive workflows.

Practical Implications

The DBeaver-based testing approach highlights several practical strengths of the Prediction-Oriented Framework that are directly relevant for real-world adoption. First, the SQL-based implementation significantly lowers the barrier to entry for prediction-oriented analysis. Because the framework can be executed entirely through standard SQL, it is accessible not only to data scientists but also to data

analysts and business intelligence practitioners who may not have advanced programming expertise in Python or machine learning libraries. This broadens the framework’s applicability to traditional data warehouse and BI environments.

Second, the testing workflow emphasizes transparency and auditability. The step-by-step structure of the tests makes all temporal assumptions and transformations explicit. Each test can be inspected, modified, and re-executed independently, allowing analysts to trace how features and labels are constructed and to verify that temporal constraints are respected. This explicitness is particularly important in regulated or risk-sensitive domains, where analytical processes must be explainable and defensible.

Third, the implementation indicates portability potential because its logical design relies primarily on standard relational constructs. Deployment on another database platform nevertheless requires verification of date and interval semantics, analytical functions, materialized-view support, indexing, and execution plans.

The comprehensive testing suite also serves a validation role beyond this specific implementation. It provides a reusable template for verifying new deployments, alternative window configurations, or domain-specific adaptations of the framework. By checking window configuration, feature generation, label extraction, ABT construction, and temporal precedence, the suite provides a reusable validation template. Each new deployment or window configuration must nevertheless be tested against its own data and platform semantics.

Finally, the testing workflow has clear educational value. By exposing prediction-oriented concepts, such as observation windows, waiting windows, and future outcome labelling, through familiar SQL interfaces, the framework can be used as a teaching tool for students and practitioners learning how to align multidimensional data modelling with predictive analytics requirements.

The successful execution of all tests with zero temporal violations provides evi-

dence that the implementation enforces the specified window constraints at the data-construction layer in the evaluated configurations, complementing the theoretical formalisation presented earlier in the chapter. The complete implementation in both Python and SQL, including all feature modules, the validation suite, and usage examples, is provided in the supplementary materials accompanying this thesis. The Python and SQL implementations were cross-checked for the reported case-study workflows under equivalent window configurations and parameter settings and produced consistent results for the compared outputs.

Overall, the evaluated implementation provides evidence that the PO Framework can unify conceptual modelling, explicit temporal constraint enforcement, and prediction-oriented data engineering within a single analytical pipeline. It supports reproducible analyses, was feasible for the evaluated workloads, and provides a portable foundation for prediction-oriented data processing grounded in formally defined PO OLAP semantics.

Bibliography

- [1] Rakhmetova, E., Garosi, M., and Combi, C. (2022). *A Comprehensive Approach to Conceptual Modelling and Visual Representation of Log Files*. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing (SAC 2022)*, pp. 385–388.
- [2] Rakhmetova, E., Combi, C., and Fruggi, A. (2021). *Conceptual Modelling of Log Files: From a UML-Based Design to JSON Files*. In *Proceedings of the ER 2021 Demos and Posters*, CEUR Workshop Proceedings, Vol. 2958, pp. 13–18.
- [3] C. Combi, B. Oliboni, G. Pozzi, A. Sabaini, and E. Zimányi. (2020) Enabling instant- and interval-based semantics in multidimensional data models: The T+ MultiDim model, *Information Sciences*, vol. 528, pp. 401-426.
- [4] C. Bertero, M. Roy, C. Sauvanaud, and G. Tredan, Experience report: Log mining using natural language processing and application to anomaly detection, in *Proc. IEEE 28th Int. Symp. Software Reliability Engineering (ISSRE)*, Toulouse, France, 2017, pp. 351-360.
- [5] C. Combi, B. Oliboni, M. Weske, and F. Zerbato, Seamless conceptual modeling of processes with transactional and analytical data, *Data & Knowledge Engineering*, vol. 134, Jul. 2021.
- [6] P. He, J. Zhu, P. Xu, Z. Zheng, and M. R. Lyu, A directed acyclic graph approach to online log parsing, *arXiv preprint arXiv:1803.05103*, 2018.
- [7] R. Kanagasabai, A. Veeramani, H. Shangfeng, K. Sangaralingam, and G. Manai, Classification of massive mobile web log URLs for customer profiling and analytics, in *Proc. IEEE Int. Conf. Big Data*, Washington, DC, USA, 2016, pp. 1609-1614.
- [8] K. Kc and X. Gu, ELT: Efficient log-based troubleshooting system for cloud computing infrastructures, in *Proc. IEEE 30th Int. Symp. Reliable Distributed Systems*, Madrid, Spain, 2011, pp. 11-20.
- [9] C. Combi, B. Oliboni, M. Weske, and F. Zerbato, Conceptual modeling of inter-dependencies between processes and data, in *Proc. 33rd ACM Symp. Applied Computing (SAC '18)*, 2018, pp. 110-119.

- [10] C. Combi, B. Oliboni, M. Weske, and F. Zerbato, Conceptual modeling of processes and data: Connecting different perspectives, in Proc. 37th Int. Conf. Conceptual Modeling (ER 2018), Xi'an, China, Oct. 2018.
- [11] A. Chuvakin, K. Schmidt, and C. Phillips, Logging and Log Management: The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management, Syngress Publishing, 2012.
- [12] M. Astekin, H. Zengin, and H. Sözer, DILAF: A framework for distributed analysis of large-scale system logs for anomaly detection, *Software: Practice and Experience*, vol. 49, pp. 153-170, 2018.
- [13] N. Zhao et al., Automatically and adaptively identifying severe alerts for online service systems, in Proc. IEEE INFOCOM 2020, Toronto, ON, Canada, 2020, pp. 2420-2429.
- [14] H. Zhang, G. Chen, et al., In-memory big data management and processing: A survey, *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 7, pp. 1920-1948, 2015.
- [15] N. Afzaliseresht, Y. Miao, S. Michalska, Q. Liu, and H. Wang, From logs to stories: Human-centred data mining for cyber threat intelligence, *IEEE Access*, vol. 8, pp. 19089-19099, 2020.
- [16] S. Hallé, Explainable queries over event logs, in Proc. IEEE 24th Int. Enterprise Distributed Object Computing Conf. (EDOC), Eindhoven, Netherlands, 2020, pp. 171-180.
- [17] A. Cuzzocrea, R. Moussa, G. Xu, and G. Grasso, Cloud-based OLAP over big data: Application scenarios and performance analysis, in Proc. IEEE/ACM 15th Int. Symp. Cluster, Cloud and Grid Computing (CCGrid), 2015, pp. 921-927.
- [18] F. Skopik, M. Wurzenberger, and M. Landauer, *Smart Log Data Analytics: Techniques for Advanced Security Analysis*, 1st ed., Springer-Verlag, New York, 2021.
- [19] P. Nimbalkar, V. Mulwad, N. Puranik, A. Joshi, and T. Finin, Semantic interpretation of structured log files, in Proc. IEEE 17th Int. Conf. Information Reuse and Integration (IRI), Pittsburgh, PA, USA, 2016, pp. 549-555.
- [20] M. Cinque, D. Cotroneo, and A. Pecchia, Event logs for the analysis of software failures: A rule-based approach, *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 806-821, Jun. 2013.
- [21] M. Cinque, D. Cotroneo, R. D. Corte, and A. Pecchia, Characterizing direct monitoring techniques in software systems, *IEEE Trans. Rel.*, vol. 65, no. 4, pp. 1665-1681, Dec. 2016.

- [22] N. Rao, C. Bansal, T. Zimmermann, A. H. Awadallah, and N. Nagappan, Analyzing web search behavior for software engineering tasks, in Proc. IEEE Int. Conf. Big Data, Atlanta, GA, USA, 2020, pp. 768-777.
- [23] F. Xhafa, D. Ramirez, D. Garcia, and S. Caballé, Performance evaluation of data mining frameworks in Hadoop cluster using virtual campus log files, in Proc. Int. Conf. Intelligent Networking and Collaborative Systems, Taipei, Taiwan, 2015, pp. 217-222.
- [24] Elasticsearch, Products and Solutions That Support ECS, 2021. [Online]. Available: <https://www.elastic.co/elastic-stack/>
- [25] Elasticsearch, Mapping Network Events. [Online]. Available: <https://www.elastic.co/guide/en/ecs/current/ecs-mapping-network-events.html>
- [26] Splunk, Log Files: Definition & Introduction. [Online]. Available: https://www.splunk.com/en_us/blog/learn/log-files.html
- [27] Q. Lin, H. Zhang, J. Lou, Y. Zhang, and X. Chen, Log clustering based problem identification for online service systems, in Proc. 38th Int. Conf. Software Engineering Companion (ICSE '16), 2016, pp. 102-111.
- [28] J. Polpinij and K. Namee, Internet usage patterns mining from firewall event logs, in Proc. 2019 Int. Conf. Big Data and Education (ICBDE'19), ACM, 2019, pp. 93-97.
- [29] Object Management Group (OMG), OMG Unified Modeling Language (OMG UML), Version 2.5.1. <https://www.omg.org/spec/UML/2.5.1/PDF>, October 10, 2021.
- [30] Booch, G., Rumbaugh, J., Jacobson, I., The Unified Modeling Language User Guide, 2nd ed., Addison-Wesley, 1999.
- [31] Nisbet, R., Miner, G., Yale, K., Handbook of Statistical Analysis and Data Mining Applications, 2nd ed., Academic Press, 2018.
- [32] Friedenthal, S., Moore, A., Steiner, R., A Practical Guide to SysML, 2nd ed., Morgan Kaufmann, 2012.
- [33] Object Management Group (OMG), Getting It Right on the Dot, https://www.omg.org/ocup-2/documents/getting_it_right_on_the_dot.pdf.
- [34] Elasticsearch, Elastic Common Schema (ECS) Reference, <https://www.elastic.co/guide/en/ecs/current/index.html>.

- [35] Paige, R.F., Drivalos, N., Kolovos, D.S., et al., Rigorous identification and encoding of trace-links in model-driven engineering, *Software and Systems Modeling*, 10, pp. 469-487, 2011.
- [36] Object Management Group (OMG), Meta-Modeling and the OMG Meta Object Facility (MOF), <https://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf>.
- [37] Lano, K., Kolahdouz-Rahimi, Sh., Poernomo, I., Comparative Evaluation of Model Transformation Specification Approaches, *International Journal of Software and Informatics*, 6(2): 233-269, 2012.
- [38] Kimball, R., Ross, M., *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*, 3rd ed., Wiley, 2013.
- [39] Golfarelli, M., Rizzi, S., *Data Warehouse Design: Modern Principles and Methodologies*, McGraw-Hill, 2009.
- [40] Golfarelli, M., Maio, D., Rizzi, S., The Dimensional Fact Model: A Conceptual Model for Data Warehouses, *International Journal of Cooperative Information Systems*, 7(2): 215-247, 1998.
- [41] Vaisman, A., Zimányi, E., *Data Warehouse Systems: Design and Implementation*, Springer, 2014.
- [42] Hand, D. J., Mannila, H., Smyth, P., *Principles of Data Mining*, MIT Press, 2001.
- [43] Cabibbo, L., Torlone, R., *Querying and Embedding of Multidimensional Queries in Object-Relational DBMSs*, 1998.
- [44] Gartner, *Gartner Analytics Maturity Model: Survey Finds Most Organizations Still in Descriptive Analytics Phase*, Gartner Research, 2023.
- [45] Wamba, S. F., Akter, S., Edwards, A., Chopin, G., & Gnanzou, D. How ‘big data’ can make big impact: Findings from a systematic review and a research agenda. *International Journal of Production Economics*, vol. 165, pp. 234-246, 2017.
- [46] Jöhnk, J., Weißert, M., & Wyrich, M. Ready for Artificial Intelligence? Investigating the Organizational Preconditions for AI Adoption in Firms. *Business & Information Systems Engineering*, vol. 63, no. 1, pp. 5-20, 2021.
- [47] Van der Meulen, R., & Rivera, J. *Gartner Survey Shows Organizations Are Slow to Advance in Data and Analytics*. Gartner Press Release, 2017.

- [48] LaValle, S., Lesser, E., Shockley, R., Hopkins, M. S., & Kruschwitz, N. Big Data, Analytics and the Path From Insights to Value. MIT Sloan Management Review, vol. 52, no. 2, pp. 21-32, 2011.
- [49] Henke, N., Levine, J., & McInerney, P. Analytics comes of age. McKinsey Analytics, 2020.
- [50] Davenport, T. H., & Harris, J. G. Competing on Analytics: The New Science of Winning. Harvard Business Press, 2007.
- [51] Langer, B. Understanding Data & Analytics Maturity: A Systematic Review of Maturity Model Composition. Schmalenbach Journal of Business Research, vol. 77, pp. 205-227, 2025.
- [52] Abelló, A., Samos, J., Saltor, F., A Framework for the Classification and Description of Multidimensional Data Models, DEXA, LNCS 2113, pp. 668-677, Springer, 2001.
- [53] Abelló, A., Samos, J., Saltor, F., Understanding Analysis Dimensions in a Multidimensional Object-Oriented Model, DMDW Workshop, 2001.
- [54] Abelló, A., *YAM²*: A Multidimensional Conceptual Model, Ph.D. Thesis, Polytechnic University of Catalonia, 2002.
- [55] Luján-Mora, S., Trujillo, J., A Comprehensive Method for Data Warehouse Design, DMDW Workshop, 2003.
- [56] Luján-Mora, S., Trujillo, J., Song, I., A UML Profile for Multidimensional Modeling in Data Warehouses, Data & Knowledge Engineering, 59(3): 725-769, 2006.
- [57] Zivot, E., Wang, J., Modeling Financial Time Series with S-PLUS, 2nd ed., Springer, 2006.
- [58] Gluchowski, P., Kurze, C., Schieder, C., A Modeling Tool for Multidimensional Data Using the ADAPT Notation, 2009.
- [59] Rizzi, S., Abelló, A., Lechtenbörger, J., Trujillo, J., Research in Data Warehouse Modeling and Design: Dead or Alive?, DOLAP, ACM, 2006, pp. 3-10.
- [60] Abelló, A., et al., Fusion Cubes: Towards Self-Service Business Intelligence, IJDWM, 9(2): 66-88, 2013.
- [61] The Kimball Group, Dimensional Modeling Techniques, <https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dimensional-modeling-techniques/>.

- [62] Golfarelli, M., Rizzi, S., From Star Schemas to Big Data: 20+ Years of Data Warehouse Research, in *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years*, pp. 93-107, 2018.
- [63] Golfarelli, M., Maio, D., Rizzi, S., Conceptual Design of Data Warehouses from E/R Schemes, *HICSS*, 1998, pp. 334-343.
- [64] Rizzi, S., Gallinucci, E., Golfarelli, M., Abelló, A., Romero, O., Towards Exploratory OLAP on Linked Data, *SEBD*, 2016, pp. 86-93.
- [65] Aligon, J., Gallinucci, E., Golfarelli, M., Marcel, P., Rizzi, S., A Collaborative Filtering Approach for Recommending OLAP Sessions, *Decision Support Systems*, 69: 20-30, 2015.
- [66] Berger, S., Schrefl, M., Analysing Multi-Dimensional Data Across Autonomous Data Warehouses, *DaWaK*, 2006, pp. 120-133.
- [67] Berkani, N., Bellatreche, L., Benatallah, B., A Value-Added Approach to Design BI Applications, *DaWaK*, 2016, pp. 361-375.
- [68] Chouder, M.L., Rizzi, S., Chalal, R., Enabling Self-Service BI on Document Stores, *DOLAP*, 2017.
- [69] Cuzzocrea, A., Saccà, D., Balancing Accuracy and Privacy of OLAP Aggregations on Data Cubes, *DOLAP*, 2010, pp. 93-98.
- [70] Eick, S., Visualizing Multi-Dimensional Data, *SIGGRAPH Comput. Graph.*, 2000.
- [71] Golfarelli, M., Mandreoli, F., Penzo, W., Rizzi, S., Turricchia, E., OLAP Query Reformulation in Peer-to-Peer Data Warehousing, *Information Systems*, 37(5): 393-411, 2012.
- [72] Giorgini, P., Rizzi, S., Garzetti, M., GRAnD: A Goal-Oriented Approach to Requirement Analysis in Data Warehouses, *Decision Support Systems*, 45(1): 4-21, 2008.
- [73] D. Yin, J. Li, G. Wu. 2021. Solving the Data Sparsity Problem in Predicting the Success of the Startups with Machine Learning Methods. *arXiv:2112.07985 [cs.LG]*.
- [74] E. Franconi, A. Kamble. A data warehouse conceptual data model. In *Proceedings of SSDBM*, 2004, pp. 435-436.
- [75] A. Younas, M. S. A. Malik, Khalil-ur-Rehman, R. Shahid. A detailed study on temporal data visualization techniques in electronic health records. In *Proceedings of the Sixth International Conference on Innovative Computing Technology (INTECH)*, Dublin, Ireland, 2016, pp. 638-643.

- [76] Ch. Lokan, E. Mendes. 2014. Investigating the use of duration-based moving windows to improve software effort prediction: A replicated study. *Information and Software Technology*, 56(9), 1063-1075.
- [77] E. Gallinucci, M. Golfarelli, S. Rizzi. Advanced topic modeling for social business intelligence. *Information Systems*, 53, 87-106 (2015).
- [78] M. Golfarelli, S. Rizzi, P. Biondi. myOLAP: an approach to express and evaluate OLAP preferences. *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 23(7), 1050-1064 (2011).
- [79] M. Golfarelli. From User Requirements to Conceptual Design in Data Warehouse Design-a Survey. In *Data Warehousing Design and Advanced Engineering Applications: Methods for Complex Construction*, 2009.
- [80] M. Francia, M. Golfarelli, S. Rizzi. A methodology for social BI. In *Proceedings of IDEAS*, 2014, pp. 207-216.
- [81] B. Hüsemann, J. Lechtenbörger, G. Vossen. Conceptual data warehouse design. In *Proceedings of the DMDW*, 2000.
- [82] M.E. Jones, I. Song. Dimensional modeling: identification, classification, and evaluation of patterns. *Decision Support Systems*, 45(1), 59-76 (2008).
- [83] B. Kämpgen, S. O’Riain, A. Harth. Interacting with statistical linked data via OLAP operations. In *Proceedings of the Semantic Web Satellite Events*, 2015.
- [84] Kämpgen, B. (2015). Flexible integration and efficient analysis of multidimensional datasets from the web.
- [85] C. Combi, B. Oliboni, F. Zerbato. Integrated Exploration of Data-Intensive Business Processes. *IEEE Transactions on Services Computing*, 2021, pp. 1-1.
- [86] C.S. Jensen, R.T. Snodgrass. Absolute Time. In: LIU, L., ÖZSU, M.T. (eds), *Encyclopedia of Database Systems*. Springer, Boston, MA, 2009.
- [87] Microsoft. PowerBI Service Samples. Marketing Sample-no-PV.xlsx. <https://github.com/microsoft/powerbi-desktop-samples/blob/main/powerbi-service-samples/Sales%20and%20Marketing>
- [88] Microsoft. MS Learn DAX PowerBI Activities. <https://github.com/MicrosoftDocs/mslearn-dax-power-bi/blob/main/activities>
- [89] W. H. Inmon. *Building the Data Warehouse*, 3rd Edition. John Wiley & Sons, Inc., USA, 2002.

- [90] T. Pedersen. Measure. In: LIU, L., ÖZSU, M.T. (eds), *Encyclopedia of Database Systems*. Springer, Boston, MA, 2009.
- [91] T.B. Pedersen. Hierarchy. In: LIU, L., ÖZSU, M.T. (eds), *Encyclopedia of Database Systems*. Springer, Boston, MA, 2009.
- [92] T.B. Pedersen. Dimension. In: LIU, L., ÖZSU, M.T. (eds), *Encyclopedia of Database Systems*. Springer, Boston, MA, 2009.
- [93] T.B. Pedersen, C.S. Jensen. Multidimensional data modeling for complex data. In *Proceedings of the 15th International Conference on Data Engineering*, Sydney, Australia, 1999, pp. 336-345.
- [94] K. Morfonios, Y. Ioannidis. Star Schema. In: LIU, L., ÖZSU, M.T. (eds), *Encyclopedia of Database Systems*. Springer, Boston, MA, 2009.
- [95] A. Shoshani. Summarizability. In: LIU, L., ÖZSU, M.T. (eds), *Encyclopedia of Database Systems*. Springer, Boston, MA, 2009.
- [96] H.J. Lenz, A. Shoshani. Summarizability in OLAP and statistical databases. In *Proceedings of the Ninth International Conference on Scientific and Statistical Database Management*, Olympia, WA, USA, 1997, pp. 132-143.
- [97] A. Sabaini, Z. Esteban, C. Combi. Extending the Multidimensional Model for Linking Cubes. In *Journées Francophones sur les Entrepôts de Données et l'Analyse en ligne*, 2015.
- [98] T. Pedersen, C.S. Jensen, C.E. Dyreson. A foundation for capturing and querying complex multidimensional data. *Information Systems*, 26, 383-423 (2001).
- [99] T. Pedersen. Multidimensional Modeling. In: LIU, L., ÖZSU, M.T. (eds), *Encyclopedia of Database Systems*. Springer, Boston, MA, 2009.
- [100] A. Mendelzon, A. Vaisman. Temporal queries in OLAP. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB)*, 2000, pp. 242-253.
- [101] P. Vassiliadis, T.K. Sellis. Modeling multidimensional databases: Conceptual models and design algorithms. In *Proceedings of the 12th International Conference on Scientific and Statistical Database Management*, Berlin, Germany, 2000, pp. 76-85.
- [102] Vaisman, A.A., Zimányi, E., Wrembel, R. (2024). Temporal Data Warehousing: Recent Advances and Open Challenges. *The VLDB Journal*, Springer.

- [103] A. Abelló, J. Samos, F. Saltor. Multidimensional modeling: Issues and research directions. *Information Technology and Management*, Vol. 7, No. 1, 2006, pp. 21-39.
- [104] Vaisman, A.A., Mendelzon, A.O. (2002). A Temporal Query Language for OLAP: Implementation and a Case Study. In: Ghelli, G., Grahne, G. (eds) Database Programming Languages, DBPL 2001. Lecture Notes in Computer Science, vol 2397. Springer, Berlin, Heidelberg.
- [105] Eder, J., Koncilia, Ch. (2001). Changes of Dimension Data in Temporal Data Warehouses. *DaWaK. 2114.*, pp. 284-293.
- [106] Koncilia, C., Eder, J. (2002). The COMET Metamodel for Temporal Data Warehouses. In: *Proceedings of CAiSE 2002*, pp. 819-834.
- [107] Eder, J., Koncilia, C. (2002). Representing Temporal Data in Non-Temporal OLAP Systems. *DEXA Workshops 2002*, pp. 817-821.
- [108] Eder, J., Wiggisser, K. (2008). Modeling Transformations between Versions of a Temporal Data Warehouse. *ER Workshops 2008*, pp. 68-77.
- [109] Awiti, J., Wrembel, R., Zimányi, E. (2025). Repairing ETL Processes using Extended Relational Algebra. *Foundations of Computing and Decision Sciences*, 50, pp. 157-190.
- [110] Scholl, M.H., Mansmann, S. (2009). Visual On-Line Analytical Processing (OLAP). In: Liu, L., Özsu, M.T. (eds) *Encyclopedia of Database Systems*. Springer, Boston, MA.
- [111] Cabibbo, L., Torlone, R. (2004). Dimension Compatibility for Data Mart Integration. *Sistemi Evoluti per Basi di Dati*.
- [112] Abelló, A., Samos, J., Saltor, F. (2003). Implementing Operations to Navigate Semantic Star Schemas. In: *Proc. of the 6th ACM International Workshop on Data Warehousing and OLAP (DOLAP)*, New Orleans, Louisiana, USA, ACM Press, pp. 56-62.
- [113] Abelló, A., Samos, J., Saltor, F. (2002). On Relationships Offering New Drill-Across Possibilities. In: *Proc. of the 5th ACM International Workshop on Data Warehousing and OLAP (DOLAP)*, McLean, Virginia, USA, ACM Press, pp. 7-13.
- [114] Koncilia, C., Morzy, T., Wrembel, R., Eder, J. (2014). Interval OLAP: Analyzing Interval Data. *DaWaK 2014*, pp. 233-244.

- [115] S. Kaufman, S. Rosset, C. Perlich. Leakage in data mining: Formulation, detection, and avoidance. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2011, pp. 556-563.
- [116] Vinnik, S., Mansmann, F. (2006). From Analysis to Interactive Exploration: Building Visual Hierarchies from OLAP Cubes. In: *Proc. of the 10th International Conference on Extending Database Technology (EDBT)*, Munich, Germany. LNCS 3896, Springer, pp. 496-514.
- [117] Giovinazzo, W. (2003). *Object Oriented Data Warehouse Design: Building a Star Schema*. Prentice Hall PTR, NJ.
- [118] Böhlen, M., Gamper, J., Jensen, C.S. (2006). Multi-dimensional Aggregation for Temporal Data. *Lecture Notes in Computer Science*, vol. 3896, Springer.
- [119] Woo, J., Lee, B.S., Lee, M., Loh, W., Whang, K. (2007). Temporal Aggregation Using a Multidimensional Index. *Journal of Database Management (JDM)*, 18(2), 62-79.
- [120] Eder, J., Koncilia, C., Wiggisser, K. (2006). Maintaining Temporal Warehouse Models. *CONFENIS 2006*, pp. 21-30.
- [121] Ahmed, U., Tchounikine, A., Miquel, M., Servigne, S. (2010). Real-time Temporal Data Warehouse Cubing. In: *Advances in Databases and Information Systems*, pp. 159-167.
- [122] Akid, H., Frey, G., Ayed, M., Lachiche, N. (2022). Performance of NoSQL Graph Implementations of Star vs. Snowflake Schemas. *IEEE Access*, 10, 48603-48614.
- [123] Boukraa, D., Boussaïd, O., Bentayeb, F. OLAP Operators for Complex Object Data Cubes. *Advances in Databases and Information Systems (ADBIS 2010)*, *Lecture Notes in Computer Science*, vol. 6295, pp. 183-196, Springer, Berlin, Heidelberg.
- [124] Chaudhuri, S., Dayal, U. (1997). An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record*, 26(1), 65-74.
- [125] Viswanathan, G., Schneider, M. (2011). On the Requirements for User-Centric Spatial Data Warehousing and SOLAP. In: *Advances in Conceptual Modeling*, pp. 144-155.
- [126] Lehner, W., Albrecht, J., Wedekind, H. (1998). Normal Forms for Multidimensional Databases. In: *Scientific and Statistical Database Management*, IEEE, pp. 63-72.

- [127] Lenz, H.-J., Shoshani, A. (1997). Summarizability in OLAP and Statistical Databases. In: Proc. of the 9th Int. Conference on Scientific and Statistical Database Management, IEEE, pp. 132-143.
- [128] Riazati, D., Thom, J.A., Zhang, X. (2008). Drill Across and Visualization of Cubes with Non-Conformed Dimensions. In: Proc. of the 19th Australasian Database Conference, vol. 75, pp. 97-105. Australian Computer Society, Inc.
- [129] Pirasteh, P., Nowaczyk, S., Pashami, S., et al. (2019). Interactive Feature Extraction for Diagnostic Trouble Codes in Predictive Maintenance: A Case Study from Automotive Domain. In: Workshop on Interactive Data Mining (WIDM'19). ACM.
- [130] Eick, S.G. (2000). Visual Discovery and Analysis. *IEEE Transactions on Visualization and Computer Graphics*, 6(1), 44-58.
- [131] Stolte, C., Tang, D., Hanrahan, P. (2002). Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1), 52-65.
- [132] Gebhardt, M., Jarke, M., Jeusfeld, M. A., Quix, C., and Sklorz, S. (1998). Tools for Data Warehouse Quality. In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, Washington, DC, IEEE Computer Society, pp. 229-232.
- [133] Conkin, N., Prabhakar, S., and North, C. (2002). Multiple foci-drill-down through tuple and attribute aggregation polyarchies in tabular data. In *IEEE Symposium on Information Visualization*, Minnesota, USA, pp. 131-134.
- [134] Roddick, J.-F., and Spiliopoulou, M. (1999). Temporal data mining: Survey and issues. *Research Report ACRC-99-007*, University of South Australia.
- [135] Han, J., Kamber, M., and Pei, J. (2011). *Data Mining: Concepts and Techniques* (3rd ed.). Morgan Kaufmann.
- [136] Widom, J., and Yang, J. (2001). Temporal data warehousing. In *International Conference on Scientific and Statistical Database Management (SS-DBM)*, IEEE, pp. 132-143.
- [137] Shickel, B., Tighe, P. J., Bihorac, A., and Rashidi, P. (2018). Deep EHR: A survey of recent advances in deep learning techniques for electronic health record (EHR) analysis. *IEEE Journal of Biomedical and Health Informatics*, 22(5), 1589-1604.
- [138] Eder, J., Franceschetti, M., Lubas, J. (2021). Time in Data Models. *FDSE 2021*, pp. 23-35.

-
- [139] Eder, J., Pichler, H. (2002). Duration Histograms for Workflow Systems. *Engineering Information Systems in the Internet Context*, pp. 239–253.
 - [140] Jensen, C. S., et al. (1998). The consensus glossary of temporal database research. *ACM Computing Surveys (CSUR)*, 29(3), 280-313.
 - [141] Snodgrass, R. T. (2000). *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann.
 - [142] Golfarelli, M., and Rizzi, S. (2009). A survey on temporal data warehousing. *International Journal of Data Warehousing & Mining (IJDWM)*, 5(1), 1-17.
 - [143] Younas, M., and Zhou, X. (2015). Temporal data visualization techniques: A survey. *International Journal of Data Warehousing and Mining (IJDWM)*, 11(3), 1-20.
 - [144] Sáez, C., Gutiérrez-Sacristán, A., Kohane, I., García-Gómez, J. M., and Avilach, P. (2020). EHRtemporalvariability: Delineating temporal data-set shifts in electronic health records. *BMC Medical Informatics and Decision Making*, 20(1), 1-14.
 - [145] Lokan, C. (2008). A comparison of static and dynamic models for software effort prediction. In *International Conference on Predictor Models in Software Engineering (PROMISE)*, IEEE, pp. 81-90.
 - [146] van der Aalst, W. M. P. (2016). *Process Mining: Data Science in Action* (2nd ed.). Springer.
 - [147] Márquez-Chamorro, A. E., Resinas, M., and Ruiz-Cortés, A. (2018). Predictive monitoring of business processes: A survey. *IEEE Transactions on Services Computing*, 11(6), 962–977.
 - [148] Teinemaa, I., Dumas, M., La Rosa, M., and Maggi, F. M. (2019). Outcome-oriented predictive process monitoring: Review and benchmark. *ACM Transactions on Knowledge Discovery from Data*, 13(2), Article 17.
 - [149] Verenich, I., Dumas, M., La Rosa, M., Maggi, F. M., and Teinemaa, I. (2019). Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring. *ACM Transactions on Intelligent Systems and Technology*, 10(4), Article 34.
 - [150] Rama-Maneiro, E., Vidal, J. C., and Lama, M. (2023). Deep learning for predictive business process monitoring: Review and benchmark. *IEEE Transactions on Services Computing*, 16(1), 739–756.

- [151] Ceravolo, P., Comuzzi, M., De Weerd, J., Di Francescomarino, C., and Maggi, F. M. (2024). Predictive process monitoring: Concepts, challenges, and future research directions. *Process Science*, 1, Article 2.
- [152] Pfeiffer, P., Abb, L., Fettke, P., and Rehse, J.-R. (2025). Learning from the data to predict the process: Generalization capabilities of next activity prediction algorithms. *Business & Information Systems Engineering*, 67, 357–383.
- [153] Mehdiyev, N., Majlatow, M., and Fettke, P. (2025). Interpretable and explainable machine learning methods for predictive process monitoring: A systematic literature review. *Artificial Intelligence Review*, 58, Article 378.
- [154] Fioretto, S., and Masciari, E. (2025). A comparative analysis of predictive process monitoring: Object-centric versus classical event logs. *Knowledge and Information Systems*, 67, 7355–7398.
- [155] Nisbet, R., Elder, J., and Miner, G. (2005). *Handbook of Statistical Analysis and Data Mining Applications*. Academic Press.
- [156] Jensen, C. S., Kjærgaard, S., and Mark, L. (2000). Temporal aggregation in moving windows. In *International Workshop on Temporal Databases*, Springer, pp. 13-32.
- [157] Roberts, S. W. (1959). Control chart complements when assignable causes are absent. *Journal of the American Statistical Association*, 54(288), 685-729.
- [158] Hunter, J. S. (1986). The exponentially weighted moving average. *Journal of Quality Technology*, 18(4), 239-250.
- [159] Gama, J. (2004). *Learning from Data Streams: Techniques and Applications*. Springer.
- [160] Amico, B., and Combi, C. (2022). A 3-window framework for the discovery and interpretation of predictive temporal functional dependencies. In Michalowski, M., Abidi, S. S. R., and Abidi, S. (Eds.), *AIME*, Springer, Cham, pp. 299-309.
- [161] Amico, B., and Combi, C., Rizzi R., Sala P. (2024). Predictive mining of multi-temporal relations, *Information and Computation*, Volume 301, Part A, ISSN 0890-5401.
- [162] Keogh, E., Chakrabarti, K., Pazzani, M., and Hattori, S. (2001). Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 3(3), 263-288.
- [163] Gaber, M. M., Krishnaswamy, S., and Zaslavsky, A. (2005). Mining data streams: A survey. *ACM SIGKDD Explorations Newsletter*, 7(1), 51-62.

- [164] Abelló, A., Romero, O., Pedersen, T., Berlanga, R., Nebot, V., Aramburu, M., and Simitzis, A. (2015). Using semantic web technologies for exploratory OLAP: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 27(2), 571-588.
- [165] Bondarev, A. (2017). Visual analysis and processing of clusters structures in multidimensional datasets. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W4, 151-154.
- [166] R. Anderson. The Credit Scoring Toolkit: Theory and Practice for Retail Credit Risk Management and Decision Automation. Oxford University Press, 2007.
- [167] L. C. Thomas, D. B. Edelman, J. N. Crook. Credit Scoring and Its Applications, 2nd ed., SIAM, 2017.
- [168] Basel Committee on Banking Supervision. International Convergence of Capital Measurement and Capital Standards: A Revised Framework. Bank for International Settlements, 2006.
- [169] Di, L., Sapino, M., and Candan, K. (2011). Unraveling multi-dimensional data using pDView.
- [170] Dzemyda, G., Marcinkevičius, V., and Medvedev, V. (2011). Web application for large-scale multidimensional data visualization. *Mathematical Modelling and Analysis*, 16(1), 273-285.
- [171] He, X., and Shu, L. Hierarchical visualization of multidimensional data. Proceedings of the 2007 International Conference on Intelligent Systems and Knowledge Engineering (ISKE 2007), pp. 1260-1263.
- [172] Korobko, A. Technology of exploratory OLAP based on the integral analytical model. Proceedings of the 2016 2nd International Conference on Artificial Intelligence and Industrial Engineering (AIIE 2016), pp. 43-47.
- [173] Nguyen, Q., Zhang, K., and Simoff, S. (2015). Unlocking the complexity of port data with visualization. *IEEE Transactions on Human-Machine Systems*, 45(2), pp. 272-279.