JOHANNES KEPLER
UNIVERSITY LINZ

Author
Ilko Kovačić, BSc MSc

Submission
Institute of Business Informatics - Data & Knowledge Engineering

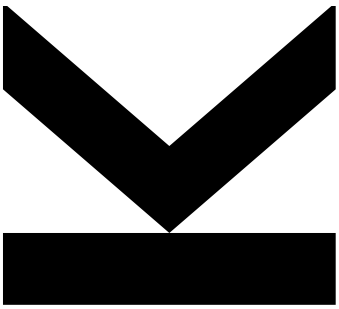
First Supervisor
o. Univ.-Prof. Dipl.-Ing. Dr. Michael Schrefl

Second Supervisor
Mag. Dr. Christoph Schütz

Assistant Thesis Supervisor
Mag. Dr. Bernd Neumayr

October 2021

OLAP Patterns: A Pattern-Based Approach to Multidimensional Data Analysis



Doctoral Thesis

to obtain the academic degree of

Doktor der Sozial- und Wirtschaftswissenschaften

in the Doctoral Program

Sozial- und Wirtschaftswissenschaften

JOHANNES KEPLER UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at

Sworn Declaration

I, Ilko Kovačić, hereby declare under oath that the submitted Doctoral Thesis "OLAP Patterns: A Pattern-Based Approach to Multidimensional Data Analysis" has been written solely by me without any third-party assistance, information other than provided sources or aids have not been used and those used have been fully documented. Sources for literal, paraphrased and cited quotes have been accurately credited.

The submitted document here present is identical to the electronically submitted text document.

Linz, 22.10.2021

Place, Date

Ilko Kovačić

Ilko Kovačić, MSc

Acknowledgments

I would like to thank the many dedicated people who have actively supported me in the preparation of this thesis. First of all, I would like to thank my supervisor Michael Schrefl, without whom this thesis would not have been possible. I would like to thank him especially for the creative and exhaustive discussions in which we thought through envisaged ideas.

I owe special thanks to Christoph Schütz, who contributed significantly to this thesis and showed me how challenging and emotional scientific work can be. I also owe thanks to my colleague Bernd Neumayr, who provided valuable input and cheered me up by reminding me that there are far worse things in the world than an unfinished thesis. My thanks also go to Margit Brandl, who saved me from administrative mischief during the thesis. In addition, I thank the master students Michael Moritz, who implemented the pattern-based approach presented in this thesis, and Simon Schausberger, who implemented the prototype in agriProKnow.

In addition, I would like to acknowledge all external circumstances, fortunate opportunities, and coincidences beyond my control that contributed significantly to making this thesis possible. First and foremost, the fact that I live in Europe and that it is possible to pursue an academic path in Austria despite my parents' lack of academic background.

Besides my superiors, colleagues, and fate, I would like to thank my mother Ora Kovačić, who taught me the value of education and achievement from an early age. I also thank my siblings Mara Rührlinger and Ivo Kovačić and their partners Thomas Rührlinger and Bettina Buchendorfer, who have always stood by me. I also thank myself for being so pain-free and disciplined and not following tempting offers from the private sector. Finally, I would like to express my deepest gratitude to my partner, Marlene Bachleitner, for her patience, everlasting support, and companionship throughout this bumpy journey.

Preface

Problem solving, in the absence of experience, requires a great deal of cognitive effort, both to understand the problem and to design and elicit possible solutions. Even if the chosen solution proves to be sufficient for the problem, it may not be the best solution because possible consequences of following it have not been considered. However, in order to apply the best possible solution, the experiences of others must be externalized in order to draw on them. For this purpose, a pattern-based approach has proven successful in a wide variety of domains, as patterns can be used to document and communicate best-practice solutions for recurring problems.

In the agriProKnow research and development project, funded by the Austrian Research Promotion Agency under grant number FFG-848610 to develop a business intelligence (BI) system for precision dairy, we applied a pattern-based approach to document, communicate, and reuse best-practice solutions for composing ad hoc OLAP queries to meet specific types of information needs. The lessons learned during the agriProKnow project with respect to the use of a pattern-based approach to compose ad hoc OLAP queries, formed the foundation for this thesis. Therefore, this thesis is relevant to anyone involved in the implementation and support of BI projects. In particular, for those responsible for implementing ad hoc queries to satisfy information needs, which includes both experienced users such as IT developers, data engineers, data scientists and less experienced users such as domain experts, decision makers, and business analysts.

This thesis is largely based on publications that have been developed by myself with Michael Schrefl, Christoph Schütz, Bernd Neumayr, Simon Schausberger, and Roman Sumereder [1], [2] and a manuscript titled *OLAP Patterns: A Pattern-Based Approach to Multidimensional Data Analysis*, which is still under review at the time of submission. Since I am the primary author with respect to last work, excerpts from it are taken directly and adapted where appropriate and not explicitly identified.

Kurzfassung

Anwender*innen von Business-Intelligence (BI)-Systemen verwenden einen Ansatz, der als Online Analytical Processing (OLAP) bezeichnet wird, um multidimensionale Daten aus verschiedenen Perspektiven zu betrachten. Abfragesprachen wie SQL oder MDX ermöglichen dabei die flexible Abfrage von multidimensionalen Daten. Für viele Anwender*innen ist die Formulierung solcher Abfragen jedoch kognitiv anspruchsvoll und daher oft zeitaufwendig. Alternativen zur Verwendung einer Abfragesprache wie grafische OLAP-Clients, parametrisierte Berichte oder Dashboards sind oft keine vollwertige Alternative zur Verwendung einer Abfragesprache. Die Erfahrungen aus Forschungsprojekten in Kooperation mit Industriepartnern führten zu den folgenden Beobachtungen hinsichtlich der Verwendung von OLAP-Abfragen in der Praxis: Erstens werden innerhalb derselben Organisation wiederholt ähnliche OLAP-Abfragen von Grund auf neu zusammengestellt, um ähnliche Informationsbedürfnisse zu befriedigen. Zweitens werden ähnlich strukturierte OLAP-Abfragen über verschiedene Organisationen und sogar Domänen hinweg wiederholt von Grund auf neu erstellt. Schließlich können vage Anforderungen an häufig benötigte OLAP-Abfragen in der Frühphase eines Projekts zu einer überstürzten Entwicklung in späteren Phasen führen, was bis zu einem gewissen Grad vermieden werden kann, indem bewährte Lösungen (Best-Practices) für die Zusammenstellung von OLAP-Abfragen herangezogen werden. Im Ingenieurwesen wird das Wissen über Best-Practice-Lösungen für häufig auftretende Herausforderungen oft in Form von Mustern (Patterns) dokumentiert. In diesem Sinne beschreibt ein OLAP-Muster eine generische Lösung für die Zusammenstellung einer Abfrage, die es BI-Anwender*innen ermöglicht, eine bestimmte Art von Informationsbedarf anhand von Fragmenten eines konzeptuellen Modells zu befriedigen. In dieser Arbeit wird eine formale Definition von OLAP-Mustern sowie eine ausdrucksstarke, flexible und allgemein anwendbare Definitionssprache vorgestellt.

Abstract

Users of a business intelligence (BI) system employ an approach referred to as online analytical processing (OLAP) to view multidimensional data from different perspectives. Query languages, e.g., SQL or MDX, allow for flexible querying of multidimensional data but query formulation is often time-consuming and cognitively challenging for many users. Alternatives to using a query language, e.g., graphical OLAP clients, parameterized reports, or dashboards, are often not a full-blown alternative to using a query language. Experience in cooperative research projects with industry led to the following observations regarding the use of OLAP queries in practice. First, within the same organization, similar OLAP queries are repeatedly composed from scratch in order to satisfy similar information needs. Second, across different organizations and even domains, OLAP queries with similar structures are repeatedly composed from scratch. Finally, vague requirements regarding frequently composed OLAP queries in the early stages of a project potentially lead to rushed development in later stages, which can be alleviated by following best practices for OLAP query composition. In engineering, knowledge about best-practice solutions to frequently arising challenges is often documented and represented using patterns. In that spirit, an OLAP pattern describes a generic solution for composing a query that allows a BI user to satisfy a certain type of information need given fragments of a conceptual model. This thesis introduces a formal definition of OLAP patterns as well as an expressive, flexible, and generally applicable definition language.

Contents

Chapter 1

Introduction

Multidimensional data analysis is central to descriptive analytics in general and online analytical processing (OLAP) in particular. In this thesis, we present a pattern-based approach to multidimensional data analysis, the development of which was informed by experience from cooperative research projects with industry participation. In the following, we describe the motivation behind the pattern-based approach to multidimensional data analysis (Section 1.1), sketch the presented approach (Section 1.2), explain the contributions of this thesis (Section 1.2), and give an overview of the organization of the remainder of this thesis (Section 1.4).

1.1 Motivation

The data warehouse remains a staple of modern business intelligence (BI) and analytics, providing integrated, clean, and consistent data using a representation suitable for data analysis. The predominant form of data representation in data warehouses is the multidimensional model, which arranges business events of interest (facts) in a multidimensional space (cube) along dimensions with hierarchically organized levels of granularity. Multidimensional data models allow to view data from different perspectives, using an approach referred to as *online analytical processing* (OLAP) for the composition of analytical queries over the cubes.

Existing OLAP systems present BI users with different options regarding the composition of analytical queries over a data warehouse. Using a query language, e.g., SQL or MDX, allows for querying the data warehouse in a flexible, ad hoc manner but is often time-consuming and cognitively challenging [3]. Graphical OLAP clients as well as parameterized reports, dashboards, and interactive stories, while considerably facilitating data analysis, are often not a full-blown alternative to using a query language due to reduced expressiveness and

increased rigidity regarding the supported types of analytical queries. Analytical queries are also often an important preparatory step for further statistical analysis and data mining.

Over the years, we have been involved in multiple BI and analytics research projects across different domains, in collaboration with various partners from industry and academia. In particular, those projects focused on data analysis for precision dairy farming [4], ontology-driven data analysis [5], with specific applications in the health insurance domain, and reference modeling for data analysis [6], [7], including data analysis processes [8], with applications in manufacturing and services industries. In these projects, we assumed the roles of researchers, consultants, and developers. While working on these projects we made the following observations which motivated the design of OLAP patterns:

1. *Within the same organization, similar OLAP queries are repeatedly composed from scratch in order to satisfy similar information needs.* For example, an OLAP query that compares the average milk yield of cattle at farm site A with the average milk yield of cattle at farm site B per month in 2019 is similar to an OLAP query that compares the maximum milk yield of cattle at farm site A with the maximum milk yield of cattle at farm site B per week in May 2018. Both OLAP queries select sets of facts and compare measure values for those sets with each other.

2. *Not only within the same organization but also across different organizations and even domains, OLAP queries with similar structures are repeatedly composed from scratch.* OLAP queries relevant for one organization are possibly relevant for other organizations in the same domain since context and, therefore, the arising information needs are similar. For example, many different dairy farms execute an OLAP query that compares the average β-hydroxybutyric acid (BHB) level in the blood of cattle under heat stress with the average BHB level in the blood of cattle free of heat stress. Furthermore, even across vastly different domains, queries concerning the same type of information need tend to be similarly structured even though the captured events of interest and the employed business vocabulary vary. For example, the previous query from the dairy farming domain is similar in structure to an OLAP query that compares the average blood sugar level of patients suffering from diabetes mellitus type 1 with the average blood sugar level of patients suffering from diabetes mellitus type 2 per state.

3. *In the early stages of a BI project, requirements regarding the OLAP queries that are to be composed often remain unclear, potentially resulting in rushed development in later stages when schedules are tight.* What queries have to be composed in an OLAP system depends on the available data sources that are integrated in the

data warehouse as well as the designed multidimensional data model. Requirements typically evolve during the project as the components are implemented. Consequently, new user demands regarding interesting analysis that should be easily carried out in the final system also arise during the later stages of a BI project.

The previous observations can be attributed to the fact that there is typically little or no documentation of knowledge about the composition of OLAP queries. The knowledge that goes into the composition of OLAP queries for satisfying specific information needs or types of information needs is typically neither being externalized nor documented in an accurate, unambiguous, complete, and easily accessible manner [9], [10]. The composition of OLAP queries requires familiarity with the employed query language as well as knowledge of the underlying data model and the relevant business terms. The lack of documentation of such knowledge – knowledge that could speed up query composition – hinders its proliferation within an organization, within a domain, and across different domains. Documentation, if available at all, is usually limited to a concrete implementation, i.e., a specific, possibly parameterized, OLAP query composed in a target language for a specific data model. Consequently, the extent of documentation is often barely sufficient to be helpful for the composition of queries for similar information needs in the same organization, let alone in a different organization or domain. Even in cases where query reuse is possible, the required modification of the existing queries makes reuse more error-prone and yields less accurate results in general than query composition from scratch [3].

1.2 Approach

A lack of knowledge transfer hampers the development and widespread adoption of best practices regarding OLAP query composition for common types of information needs – users keep "reinventing the wheel" when it comes to query composition. Following an established set of best practices, however, may reduce the cognitive effort required for the composition of correct queries, help users avoid common pitfalls, and ultimately reduce composition time. Following best practices may also result in less obfuscated solutions, which would in turn increase maintainability of OLAP queries. In addition, best practices make it possible to identify and anticipate possible common requirements in BI projects so that the schedule can be met without the increased development effort in later phases that would otherwise be necessary. Proper documentation and representation of knowledge regarding OLAP query composition to answer common types of information needs is a prerequiste for the establishment of best practices within individual organizations as well as across different organizations and even domains.

In engineering, knowledge about best-practice solutions to frequently arising challenges is often documented and represented using patterns. Broadly speaking, a pattern is "an idea that has been useful in one practical context and will probably be useful in others" [11, p. 8]. In his seminal work, Alexander describes a pattern as a rule "which establishes a relationship between a context, a system of forces which arises in that context, and a configuration which allows these forces to resolve themselves in that context" [12, p. 253]. Alexander's comprehensive view on patterns echoes in the Gang of Four's work on object-oriented software design patterns, where a pattern "names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design" [13, p. 4]. More recently, solution patterns have been employed in the field of BI and analytics to support the development of machine learning applications [14], [15]. Hence, we propose to employ patterns in order to communicate and represent key aspects of OLAP query composition to answer different types of information needs in specific data analysis contexts under consideration of logical dependencies.

An OLAP pattern describes a generic solution for composing a query that allows a BI user to satisfy a certain type of information need given fragments of a conceptual model, regardless the concrete logical representation of a specific data warehouse. Each OLAP pattern is characterized by a set of parameters and defines conditions at the conceptual level which a multidimensional model has to satisfy in order for the pattern to be applicable to the model. Furthermore, OLAP patterns may exist at multiple levels of abstraction, ranging from domain-independent to domain- and organization-specific patterns. The boundaries are fluid and the more specific patterns may derive from the more general patterns. Templates for different target query languages and data warehouse systems link the pattern to its implementation enabling automatic generation of queries and thus describe how a particular information need specified over a conceptual model translates into an executable OLAP query in the context of a particular data warehouse system.

In order to benefit from automatic query generation via OLAP patterns, users instantiate an OLAP pattern by substituting parameters with elements from an enriched multidimensional model (eMDM). The conceptual multidimensional model of a data warehouse system in conjunction with the definitions of business terms constitute an eMDM. In this regard, the definition of a business term includes a description as well as an expression for a target language and system which allows users to employ business terms in queries. For example, the business term `Mid Lactation Phase` in a dairy farming setting could be employed to select facts of interest concerning animals in their phase of high milk output, while the business term `Frequent Patient` in a health insurance setting could be employed to select facts of interest concerning persons frequently visiting general practitioners within a year.

The idea of OLAP patterns was first developed in the course of the agriProKnow project – while also drawing from experience in previous projects [5], [6] – in order to deal with uncertainties regarding the stakeholders' requirements with respect to the OLAP queries that would have to be composed [2]. The agriProKnow project was a joint research and development effort between industry and academia to design and implement a data warehouse for precision dairy farming, integrating sensor data as well as available operational databases [4]. During requirements elicitation, when it came to identifying interesting business questions we faced stakeholders who were vague regarding the required queries that would have to be composed eventually, partly due to uncertainties concerning the data warehouse's still evolving schema. The project's rather strict schedule and budget, however, meant that shifting effort for the implementation of queries to later stages was only possible to a limited extent. Thus, in order to reduce effort in later stages, we started developing a user interface for query composition based on domain-independent OLAP patterns. The reasoning was simple: Domain-independent OLAP patterns represent abstract solutions to specific types of information needs and, therefore, future domain- and organization-specific information needs can be satisfied by adapting the abstract solutions to suit the current situation without composing them from scratch. Identification of domain-independent OLAP patterns was grounded in experience from previous research projects, e.g., the Semantic Cockpit project [5] in the health care domain. Obtaining patterns from practical experience is a common approach in the pattern community: Patterns are primarily *identified* in a practical context rather than being the result of scientific invention [11, p. 7]. Based on domain-independent patterns, patterns specific to dairy farming and, ultimately, patterns and queries specific to the agriProKnow project's demonstration farms could later be defined and operationalized with small effort. One such domain-independent pattern was *set-to-complement comparison*, where a set of facts is compared with all other facts in the same grouping. In the dairy farming domain, set-to-complement comparison frequently takes the form of comparing a group of animals having a certain characteristic within a farm to all the other animals within that same farm. Ultimately, the pattern-based approach turned out to be successful in the agriProKnow project, as most of the required queries could be expressed using previously identified OLAP patterns.

1.3 Contributions

In this thesis, we carefully revise and significantly extend our previous work on OLAP patterns [1], [2], which established the basic notion of OLAP patterns and was later applied in the context of linked open data on the semantic web [16]. The original pattern-based approach to data analysis was developed in an ad hoc manner, out of necessity, tailored to the specific needs of the agriProKnow project – a byproduct to cope with the uncertainties

related to the project's requirements. This thesis presents a formal definition of OLAP patterns as well as a more expressive, flexible, and generally applicable conceptualization and definition language while relaxing some of the original restrictions and assumptions. In particular, the contributions presented in this thesis are as follows:

1. **Formal definition of eMDMs and OLAP patterns.** A consistent formal definition of the notions of enriched multidimensional model and OLAP patterns is provided. This includes the definition of necessary constructs for representing multidimensional models and various types of business terms defined on top of those multidimensional models as well as constructs for representing OLAP patterns.
2. **Formalization of OLAP pattern usage.** A formalization of the necessary steps to use patterns with respect to an associated eMDM is provided. For this purpose, the instantiation of a pattern, its grounding, and subsequent execution in the context of an eMDM are formally defined. Furthermore, the conditions of a pattern are defined, which describe when the pattern is applicable to an associated eMDM and thus also executable.
3. **Language to define eMDMs as well as to define and use OLAP patterns.** We introduce a language for creating and deleting eMDMs and OLAP patterns. This language also supports statements for using and organizing patterns.
4. **Graphical notation for illustration of eMDMs and OLAP patterns.** A visual notation is provided to facilitate communication between stakeholders. The graphical representation of elements of multidimensional models, business terms, and patterns follows design principles to achieve a cognitively effective visualization.
5. **Means for the organization of patterns.** The organization of patterns into organization-specific, domain-specific, and domain-independent patterns is described to enable the knowledge transfer within organizations and domains as well as across domains. This comprises the definition of a mechanism for the systematic derivation of patterns from other, more generic patterns.
6. **Prototypical implementation of the core OLAP pattern approach.** The pattern-based approach to multidimensional data analysis is realized by a prototype that facilitates the definition and organization of eMDMs and patterns as well as the usage of patterns. To this end, the prototype interprets statements that are formulated in the introduced language in order to execute the corresponding actions.

The OLAP pattern approach developed in the course of agriProKnow was limited to enriched multidimensional models in which each element was uniquely named, but this proved to

be too rigid in general for other practical applications. Therefore, the presented approach assumes a relaxed unique name assumption, thereby increasing the practical relevance of the presented pattern-based approach to data analysis.

1.4 Outline

The remainder of this thesis is organized as follows:

Chapter 2 – The Pattern-Based Approach in a Nutshell sketches our pattern-based approach to multidimensional data analysis by detailing the steps necessary to define OLAP patterns and to use them in a certain context. To this end, both pattern definition and usage are exemplified by introducing a running example from the domain of precision dairy farming.

Chapter 3 – Related Work reviews classical and more recent work on patterns in general, approaches for (language-specific) pattern-based data analysis, and visual and model-driven analytics approaches.

Chapter 4 – Enriched Multidimensional Models defines the notion of the enriched multidimensional model by formalizing the elements of a multidimensional model and the business terms that can be defined on top of the multidimensional model. Furthermore, the application of business terms to a multidimensional model and the validity of this application are described in detail. The chapter ends with the introduction of a graphical notation to visualize the introduced concepts.

Chapter 5 – Pattern Definition defines the structure of OLAP patterns. To this end, pattern variables, derivation rules, local cubes, and pattern constraints are introduced. In addition, it also defines the conditions that characterise a well-formed OLAP pattern. Finally, a graphical notation to visualize OLAP patterns and its elements is presented.

Chapter 6 – Pattern Usage formalizes the steps that are necessary to use patterns in the context of an associated enriched multidimensional model. This covers instantiation, grounding, and execution of patterns.

Chapter 7 – Pattern Organization describes how OLAP patterns and corresponding elements of an enriched multidimensional model can be organized on the domain-independent, domain-specific, and organization-specific level of abstraction. It also describes how OLAP

patterns and elements of an enriched multidimensional model can be grouped in particular fields of applications by defining pattern catalogs, business term vocabularies, and multidimensional models containing cubes and dimensions.

Chapter 8 – Proof-of-Concept Prototype presents a prototype implementation that allows to manage and use OLAP patterns and enriched multidimensional models by formulating statements that adhere to the introduced pattern language. To this end, the overall architecture with its components is outlined and the provided functionality is discussed. Finally, it is shown how the agriProKnow use case (running example) can be realized using the prototype implementation.

Chapter 9 – Evaluation evaluates the presented pattern-based approach to multidimensional data analysis by discussing practical relevance and expressiveness. In addition, the approach is evaluated against a framework for assessing the quality of domain-specific languages.

Chapter 10 – Extensions describes extensions that can be introduced to enhance the core pattern-based approach to multidimensional data analysis. Possible extensions include composite types and collections, optional variables, generic business terms, and enhanced value set descriptions.

Chapter 11 – Conclusion summarizes and discusses the presented pattern-based approach to multidimensional data analysis.

Chapter 2

The Pattern-Based Approach in a Nutshell

In this chapter we outline the pattern-based approach to multidimensional data analysis in Section 2.1 by describing the steps necessary to define OLAP patterns and the subsequent steps for using OLAP patterns in general. Following this, in Section 2.2 we present a running example from the dairy industry to describe how an appropriately enriched multidimensional model can be defined. Based on this, we define a pattern specific to this domain and show how it can be used with regard to the defined enriched multidimensional model.

2.1 Pattern Definition and Usage

OLAP patterns are a means of documentation for best-practice solutions regarding query composition – with support for automatic query generation. Therefore, the proposed pattern-based approach to multidimensional data analysis distinguishes two separate activities: Pattern definition and pattern usage. Pattern definition comprises bottom-up specification of patterns in the context of a specific data warehouse system as well as top-down derivation of more specific patterns from more general patterns in a catalog. Pattern usage, on the other hand, requires the pattern user to bind values to pattern parameters, which allows for the subsequent grounding and execution of the pattern in the context of a specific data warehouse system. Different business intelligence users are involved in the course of pattern definition and usage [17]. While pattern definition will typically involve *pattern authors* with a profound BI understanding at least to some extent, such as IT developers, data engineers, and data scientists, pattern usage directly concerns the *pattern users* with a basic BI understanding, such as decision makers, domain experts, business analysts, and power users (see Figure 2.1). In this sense, OLAP patterns serve to communicate knowledge of best-practice solutions for satisfying generic types of information needs from experienced

pattern authors to pattern users. It should be noted that a person can assume both the role of a pattern author and pattern user.

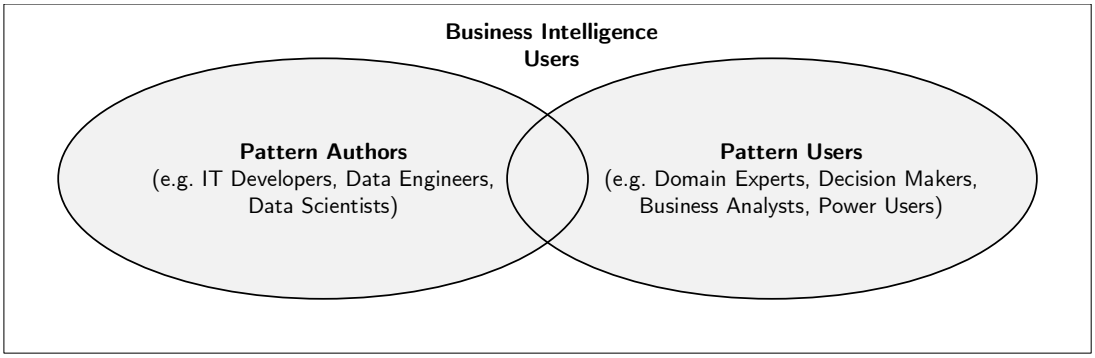


Figure 2.1: Venn diagram of the roles of the pattern-based approach to multidimensional data analysis

In addition, OLAP patterns could facilitate communication between users by fostering a shared understanding of best practices to be followed in order to satisfy a certain type of information need. A catalog of OLAP patterns allows users to communicate solutions for specific information needs without having to describe the query composition process in detail. In this way, OLAP patterns can make it easier for users to interpret analytical results obtained through the application of a specific OLAP pattern. When using traditional reports, a user can assume that the analytical results were obtained with predefined procedures that do not change, thus reducing the cognitive effort required to interpret the results once they are understood. OLAP patterns present all of those advantages but, in contrast to traditional reports, OLAP patterns do not rely on specific, individual OLAP queries.

The definition of an OLAP pattern includes (i) textual descriptions, (ii) a context specification, and (iii) query templates. As with software design patterns [13], the textual description helps to communicate the problem to be solved and the solution to be followed. In particular, the textual descriptions consist of a list of alias names for the pattern, a statement of the (analytical) problem tackled by the pattern, and a description of the solution. The context specification, in turn, generically describes the environment which the pattern can be applied in; it should be noted that the multidimensional query to be composed is not conceptually represented such as in Varga et al. [18]. In particular, the context specification comprises a declaration of pattern parameters as well as constraints, which the enriched multidimensional model of a data warehouse must satisfy in order for the pattern to be applicable in the context of that data warehouse. Furthermore, the context specification comprises definitions of auxiliary constructs and the schemas of intermediary results, i.e., derived elements and local cubes, respectively. These constraints, derived elements with corresponding derivation rules, and local cubes can be classified as different

types of business rules according to Hay et al. business rules [19]; constraints correspond to action and structure assertions, derived elements correspond to derivations, and local cubes correspond to structure assertions. In analogy to software design patterns [13], which can be implemented using different object-oriented languages, an OLAP pattern's query templates are realizations in various target query languages, taking into account the type of database (relational or other), the modeling paradigm (star or snowflake schema), and the specific database management system. A pattern template is an expression that is interspersed with placeholders for pattern parameters and derived elements as well as macro calls. Templates may serve as blueprints for manually writing executable queries, that is, templates link the pattern to its implementation. More importantly, however, templates allow for automatic generation of a query that is executable in a target system. In this thesis, we focus on SQL as a target language over a relational data warehouse following the star schema modeling paradigm. Previous work demonstrated feasibility of adopting the pattern-based approach for the analysis of linked open data using SPARQL as a target language over RDF data [16].

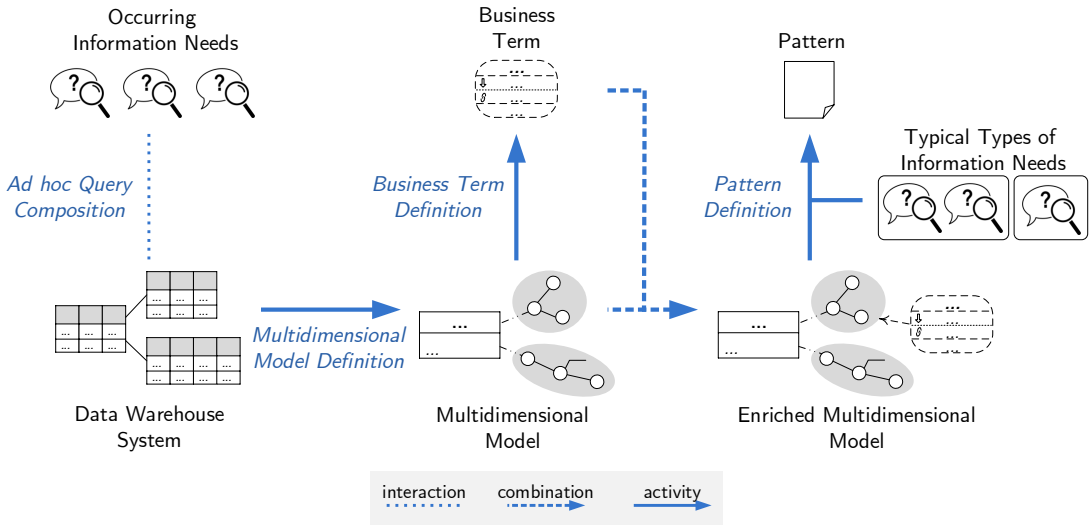


Figure 2.2: General steps of a *pattern author* in the pattern *definition* process

Patterns are usually discovered rather than invented [11, p. 7]: The challenge lies in abstracting the essential structure of queries and logical data models without losing the wholeness [20, p. 49] – too specific patterns have a limited scope, while too general patterns are difficult to apply in concrete analysis situations. Starting from the logical data model of a specific data warehouse system, patterns may be specified in a bottom-up manner by abstracting the structure of the logical data model and the typical analytical queries to obtain a generic query pattern at a higher level of abstraction (see Figure 2.2). First, the logical data model can be relational or other. Therefore, a conceptual representation of the logical

schema of the data serves as the basis for pattern definition, which fosters reusability of the identified patterns. In addition to the conceptual multidimensional model, the pattern-based approach relies on the unambiguous definitions of business terms, i.e., the multidimensional model is enriched with business term definitions and becomes an *enriched multidimensional model* (eMDM). Each business term is either applied to cubes or dimensions and can represent different query functionality, i.e., calculated measures, predicates, groupings, or orderings. Business terms and multidimensional model elements in an eMDM represent vocabulary used in day-to-day operations with expressions specifying the meaning of the business terms in a way that can be operationalized for query composition, hence, users may employ the vocabulary to instantiate the patterns. Business terms are thus characterized by expressions for a target language and state conditions that cubes and dimensions must satisfy in order for a specific business term to be applicable. Domain ontologies, e.g., AGROVOC [21] in dairy farming or SNOMED CT [22] in the health insurance domain, may form the basis for the definition of a vocabulary of business terms [23].

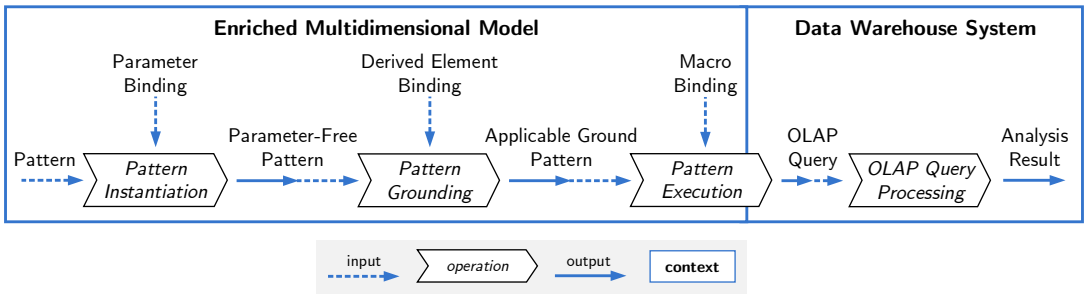


Figure 2.3: General steps of a *pattern user* in the pattern *usage* process

Figure 2.3 shows the necessary steps to put to use once defined patterns that have a corresponding query template attached. Users instantiate a pattern by binding the parameters with names of elements (arguments) from an eMDM. Partial instantiation of a pattern, i.e., binding only a subset of the pattern's parameters, yields a more specific pattern, which can be kept for later reuse, constituting a case of top-down derivation of more specific patterns from more general patterns. Ultimately, the result of (possibly repeated) pattern instantiation is a *parameter-free* pattern. In the absence of derived elements, a pattern with all parameters bound is also *ground*, i.e., pattern grounding can be omitted. Otherwise, in order to obtain a ground pattern, a parameter-free pattern undergoes the process of pattern grounding, i.e., existing derivation rules for derived elements are evaluated over an eMDM to determine the values to be bound to the corresponding derived elements. A ground pattern is *applicable* in the context of a data warehouse system if all of the pattern's constraints can be satisfied in the eMDM and all business terms applied to cubes or dimensions are valid business term applications. Execution of a ground pattern over an

eMDM of a data warehouse in the context of which the pattern is applicable then produces an OLAP query, which is processed by the data warehouse system to obtain an analysis result. During pattern execution, macro calls in the attached query templates employ the variable bindings in order to obtain code snippets in a target language which are substituted for the macro calls in the corresponding pattern template.

2.2 Running Example

In the following, we illustrate the pattern-based approach using the setting of precision dairy farming inspired by our own experience in the agriProKnow project; the setting serves as running example throughout the thesis. Consider (fictitious) dairy company *Happy Milk*, which consists of three farms located at different sites, tending to a herd of about a thousand animals in total, approximately half of which are Holstein breed, the other half Jersey breed. Happy Milk runs its farms as precision dairy farming operations [24] aiming to increase efficiency and reduce losses due to animal illness through monitoring the animals' health status and proactively induce necessary treatments. In precision dairy farming, animals are typically monitored using a wide range of sensors capturing a multitude of data concerning, e.g., microclimate in the barns (temperature and moisture, among others), animal movement within farms, food consumption, milk yield and composition of the produced milk, and information about calvings. In the example setting, data gathered by various sensors are integrated into a relational data warehouse system realized using a star schema which is conceptually represented using entities, i.e., cubes and dimensions, and properties, i.e., measures, dimension roles, levels, and attributes, in a multidimensional model.

Figure 2.4 shows an example `Milking` cube similar to an actual cube developed for the agriProKnow project, using a notation that is based on the Dimension Fact Model [25] where boxes denote cubes, circles denote levels, and lines between circles denote roll-up relationships; levels are grouped into dimensions, which are represented as a gray area. The `Milking` cube captures, for analysis purposes, aggregated data about milking events quantified by the measure `Milk Yield` (in liters) and `Fat Content` (as percentage); value sets `Liquid In Liter` and `Percent Per Liter` are the types of `Milk Yield` and `Fat Content`, respectively. `Milking` refers to the `Animal` dimension via the `Cattle` dimension role, `Time` dimension via the `Milking Time` role, `Lactation` dimension via the `Lactation` dimension role, `Calving` dimension via the `Calving` role, and `Farm` dimension via the `Farm` role. Hence, the cube records milk yield and fat content on a daily basis per animal and farm as well as calving period and phase in the animal's lactation cycle. The dimensions have multiple levels and attributes – each of which is associated with a value set (see [26] for more

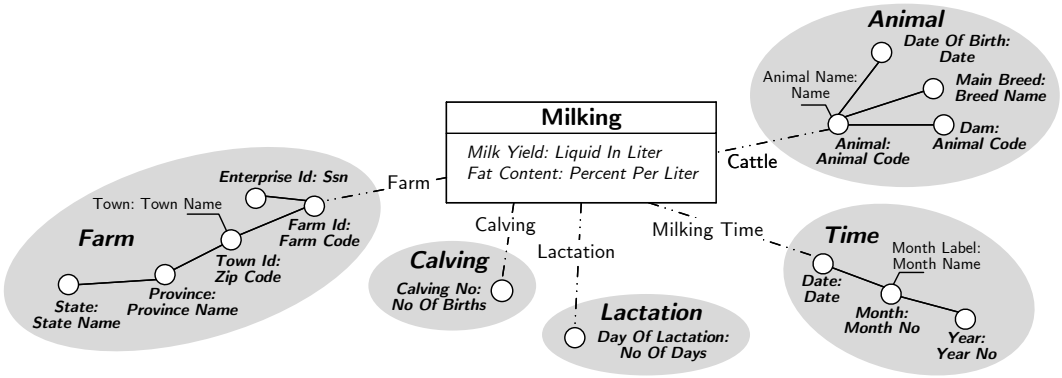


Figure 2.4: The milking cube of the multidimensional model of the *Happy Milk* data warehouse system

information on value sets) – which allow for further aggregation and selection of subsets of the data. For example, the `Animal` dimension has an `Animal` level, the values of which are taken from the `Animal Code` value set. The `Animal` level is further described by an attribute `Animal Name` from the `Name` value set. The `Animal` level rolls up to levels `Dam` (mother animal), `Main Breed`, and `Date of Birth`.

Even though Happy Milk maintains a multitude of reports for monitoring and controlling the dairy herd, covering all future information needs by predefined reports alone is impossible. Relying on reports is not enough because they are limited by the underlying query and thus not all information needs that arise can be covered by predefined reports, because not every analysis situation can be identified beforehand – only about 60-80% of all arising information needs can typically be satisfied by reports [27, p. 19]. Thus the scope of a predefined report is limited and the composition of ad hoc OLAP queries will eventually become necessary. Parameterized reports somewhat alleviate the problem but still lack the flexibility of ad hoc querying. Happy Milk's users with a basic BI understanding, for example, can be assumed to understand the multidimensional model to some degree but have only a basic understanding of the underlying data warehouse system and the corresponding query language. The composition of ad hoc OLAP queries, however, is a challenging task as it requires a profound understanding of the query language, the data warehouse system, and the specific multidimensional model. For example, to perform a comparison of the average milk yield of two distinct groups of cattle of a certain breed, a BI user has to specify selection conditions defining the two groups, obtain the average milk yield for each group at the desired granularity level, and relate the groups using the appropriate join condition before comparing the milk yield by calculating a comparative measure such as the ratio between one group's average milk yield and the other's. Thus, users with basic BI understanding will typically require further assistance for ad hoc query composition.

Figure 2.5: Breed-specific subset-subset comparison pattern's aliases, problem, and context

Breed-Specific Subset-Subset Comparison 

Also Known As

Breed-Specific Cattle Group to Cattle Group Comparison

Problem

Retrieve aggregated measure values for two specified groups of facts about cattle relating to a specific breed from a single source cube, which should be compared by calculating a comparative measure.

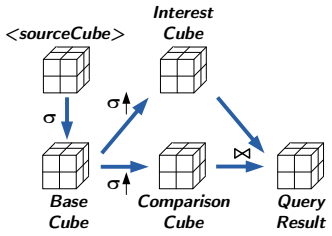
Context

```
PARAMETERS
   <sourceCube>:CUBE;
   <baseCubeSlice>:UNARY_CUBE_PREDICATE;
   <baseDimSlice>:UNARY_DIMENSION_PREDICATE;
   <compDimRole>:DIMENSION_ROLE;
   <iDimSlice>:UNARY_DIMENSION_PREDICATE;
   <cDimSlice>:UNARY_DIMENSION_PREDICATE;
   <joinDimRole>:DIMENSION_ROLE;
   <groupCond>:DIMENSION_GROUPING;
   <cubeMeasure>:UNARY_CALCULATED_MEASURE;
   <compMeasure>:BINARY_CALCULATED_MEASURE;
END PARAMETERS;

DERIVED ELEMENTS
   <baseDim>:DIMENSION <= <sourceCube>."Cattle";
   <compDim>:DIMENSION <= <sourceCube>.<compDimRole>;
   <joinDim>:DIMENSION <= <sourceCube>.<joinDimRole>;
   <cubeMeasureDom>:NUMBER_VALUE_SET <= <cubeMeasure>.RETURNS;
END DERIVED ELEMENTS;

LOCAL CUBES
   "interestCube":CUBE;
   "interestCube" HAS MEASURE <cubeMeasure>;
   "interestCube".<cubeMeasure>:<cubeMeasureDom>;
   "comparisonCube":CUBE;
   "comparisonCube" HAS MEASURE <cubeMeasure>;
   "comparisonCube".<cubeMeasure>:<cubeMeasureDom>;
END LOCAL CUBES;

CONSTRAINTS
   <sourceCube> HAS DIMENSION_ROLE "Cattle";
   <sourceCube> HAS DIMENSION_ROLE <compDimRole>;
   <sourceCube> HAS DIMENSION_ROLE <joinDimRole>;
   <sourceCube>."Cattle":<baseDim>;
   <baseDim> HAS LEVEL "Main Breed";
   <baseDim>."Main Breed":"Breed Name";
   <sourceCube>.<compDimRole>:<compDim>;
   <sourceCube>.<joinDimRole>:<joinDim>;
   <baseDimSlice> IS_APPLICABLE_TO <baseDim>;
   <baseCubeSlice> IS_APPLICABLE_TO <sourceCube>;
   <iDimSlice> IS_APPLICABLE_TO <compDim>;
   <cDimSlice> IS_APPLICABLE_TO <compDim>;
   <groupCond> IS_APPLICABLE_TO <joinDim>;
   <cubeMeasure> IS_APPLICABLE_TO <sourceCube>;
   <compMeasure> IS_APPLICABLE_TO ("interestCube","comparisonCube");
END CONSTRAINTS;
```

Figure 2.6: Breed-specific subset-subset comparison pattern with its solution and a template (continued from Figure 2.5)

Solution

From the ⟨sourceCube⟩, select the set of relevant facts using the unary cube predicate ⟨baseCubeSlice⟩ and dimension predicate ⟨baseDimSlice⟩, which selects over the `Main Breed` level of a dimension ⟨baseDim⟩ referenced by the dimension role `Cattle` – the result serves as the *base cube* for further analysis. From that base cube, select *interest cube* and *comparison cube* using conditions over the dimension role ⟨compDimRole⟩ according to the unary dimension predicates ⟨iDimSlice⟩ and ⟨cDimSlice⟩, respectively. Perform a roll-up for interest and comparison cube according to the ⟨groupCond⟩ dimension grouping over the ⟨joinDim⟩ dimension referenced by the dimension role ⟨joinDimRole⟩ and compute a unary calculated measure ⟨cubeMeasure⟩. To obtain the query result cube, join the interest cube and comparison cube over the ⟨groupCond⟩ dimension grouping and compute a binary calculated comparative measure ⟨compMeasure⟩.



Template (Data Model: Relational, Variant: Star Schema, Language: SQL, Dialect: ORACLEv11)

```
WITH baseCube AS (
  SELECT *
  FROM   <sourceCube> sc
         JOIN <baseDim> a ON
            sc."Cattle"=a.$dimKey(<baseDim>)
  WHERE  $expr(<baseCubeSlice>, sc) AND
         $expr(<baseDimSlice>, a)
),
interestCube AS (
  SELECT $expr(<groupCond>, jd),
         $expr(<cubeMeasure>, bc) AS <cubeMeasure>
  FROM   baseCube bc
         JOIN <joinDim> jd ON
            bc.<joinDimRole>=jd.$dimKey(<joinDim>)
         JOIN <compDim> cd ON
            bc.<compDimRole>=cd.$dimKey(<compDim>)
  WHERE  $expr(<iDimSlice>, cd)
  GROUP BY $expr(<groupCond>, jd)
),
comparisonCube AS (
  SELECT $expr(<groupCond>, jd),
         $expr(<cubeMeasure>, bc) AS <cubeMeasure>
  FROM   baseCube bc
         JOIN <joinDim> jd ON
            bc.<joinDimRole>=jd.$dimKey(<joinDim>)
         JOIN <compDim> cd ON
            bc.<compDimRole>=cd.$dimKey(<compDim>)
  WHERE  $expr(<cDimSlice>, cd)
  GROUP BY $expr(<groupCond>, jd)
)
SELECT $expr(<groupCond>, ic),
       ic.<cubeMeasure> AS "Group of Interest",
       cc.<cubeMeasure> AS "Group of Comparison",
       $expr(<compMeasure>, ic, cc) AS <compMeasure>
FROM   interestCube ic
       JOIN comparisonCube cc ON
          $expr(<groupCond>, ic)=$expr(<groupCond>, cc)
```

At this point, we explain the intuition of pattern definition and usage over the example of a specific OLAP pattern in order to foster a general understanding of the pattern-based approach while referring to later sections for a formal definition. From previous experience, Happy Milk's pattern authors together with the future pattern users identify breed-specific subset-subset comparison of cattle to be relevant in future analysis situations, leading to the definition of a *breed-specific subset-subset comparison* pattern (see Figure 2.5 and Figure 2.6, and Appendix C for the complete agriProKnow use case). This pattern describes how to compose breed-specific comparisons, i.e., comparisons of measure values of two groups of facts from a single cube, restricted to a specified breed. Breed-specific subset-subset comparison is characterized by a number of pattern parameters as well as derived elements and local cubes, which are defined in the *Context* part of the definition. The ⟨sourceCube⟩ parameter represents the cube that provides the facts for the analysis (Figure 2.5—Context, Line 2). *Constraints* restrict valid arguments for ⟨sourceCube⟩ to cubes having a dimension role `Cattle` (Line 31) that refers to a dimension represented by the derived element ⟨baseDim⟩ (Line 34) that contains a level `Main Breed` (Line 35) of type `Breed Name` (Line 36). The name of the dimension referenced by ⟨sourceCube⟩ via `Cattle` is then also referred to as ⟨baseDim⟩ (Line 15). The ⟨baseDimSlice⟩ parameter represents a unary dimension predicate, which restricts dimension property values, i.e., level and attribute values (Line 4), and that must be applicable to the dimension ⟨baseDim⟩ (Line 39), e.g., a restriction of `Main Breed` to Holstein. Thus, using a unary dimension predicate defined over `Main Breed` will be a valid argument for the ⟨baseDimSlice⟩ parameter. The ⟨baseCubeSlice⟩ parameter represents a unary cube predicate (Line 3) that filters the considered facts from the source cube by measure and dimension role value, e.g., to select only those facts where the fat content exceeds a certain threshold. The actual unary cube predicate supplied for ⟨baseCubeSlice⟩ must be applicable to ⟨sourceCube⟩ (Line 40), e.g., the ⟨sourceCube⟩ must have a `Fat Content` measure in order for a unary cube predicate that restricts by `Fat Content` to be a valid argument for the ⟨baseCubeSlice⟩ parameter. Both ⟨baseDimSlice⟩ and ⟨baseCubeSlice⟩ serve to restrict the base facts used for the analysis. The ⟨compDimRole⟩ parameter then represents the dimension role (Line 5) of the ⟨sourceCube⟩ (Line 32) used to define the facts of interest and the facts of comparison. The name of the dimension referenced by ⟨sourceCube⟩ via ⟨compDimRole⟩ is then also referred to as ⟨compDim⟩ (Line 37 and Line 16). The ⟨iDimSlice⟩ and ⟨cDimSlice⟩ parameters (Lines 6-7) represent the unary dimension predicates over dimensions that serve to make the actual selection of facts of interest and facts of comparison, respectively; the unary dimension predicates must be applicable to ⟨compDim⟩ (Lines 41-42). The name of the dimension referenced by ⟨sourceCube⟩ via ⟨joinDimRole⟩ (Line 8 and Line 33) is also referred to as ⟨joinDim⟩ (Line 38 and Line 17). The ⟨groupCond⟩ parameter (Line 9) represents a dimension grouping condition used to

perform a roll-up (aggregate) of the facts of interest and comparison, which must be applicable to the ⟨joinDim⟩ dimension (Line 43). The parameter ⟨cubeMeasure⟩ (Line 10) represents the unary calculated measure calculated for the aggregated facts of interest and comparison based on the facts of the cube represented by ⟨sourceCube⟩ (Line 44). The thus obtained facts of interest and facts of comparison are retrieved by subqueries in the underlying template(s) and are represented as the interestCube local cube (Line 22) and the comparisonCube local cube (Line 25), respectively, which both have a measure that has the same name and domain as ⟨cubeMeasure⟩ (Lines 23-24 and Lines 26-27). A local cube specifies the expected schema of the result returned by a certain subquery in the template(s) relative to the dynamically bound values of parameters; local cubes facilitate the definition of queries and increase expressiveness of the pattern-based approach. The derived element ⟨cubeMeasureDom⟩ (Line 18) represents the returned domain of ⟨cubeMeasure⟩ which serves to declare the measure of the *local cubes* (Line 24 and Line 27). Finally, the ⟨compMeasure⟩ (Line 11) represents a comparative measure applied to the ⟨cubeMeasure⟩ values contained in the interestCube and comparisonCube local cubes (Line 45). Pattern parameters, derived elements, and local cubes then have to be arranged within an OLAP query in a meaningful way to obtain the desired results.

In what way the pattern parameters, derived elements, and local cubes must be arranged within an OLAP query is defined informally in the *Solution* part and more formally in the *Template* part of the pattern definition. The example template in Figure 2.6 employs SQL in the Oracle dialect and is specific to a relational star schema implementation that obeys certain conventions; it should be noted that quoted identifiers refer to case-sensitive names of tables and columns. The template, at specific positions in the code, makes reference to the pattern variables (pattern parameters or derived elements) defined in the context specification, e.g., ⟨sourceCube⟩ is referenced in the FROM clause of a common table expression (Figure 2.6—Template, Line 3). When instantiating the pattern, certain references to pattern variables, e.g., ⟨sourceCube⟩, are simply replaced in the template by the name bound to the respective variable while others are used as parameter bindings for macro calls, which return a code snippet replacing the reference to the respective variable in the template. For example, the $expr(⟨baseCubeSlice⟩, sc) macro call (Line 6) returns the ground expression of the unary cube predicate referred to by the name bound to the ⟨baseCubeSlice⟩ parameter with the alias name of the cube bound to the ⟨sourceCube⟩ parameter in the pattern template. Table 2.1 describes the intuition of basic macros employed in the example template in Figure 2.6.

The context specification clearly defines the conditions that must be met for a pattern in order to be applicable in an eMDM. Following the design-by-contract metaphor from software engineering [28], the specification of the context can be seen as a contract that

defines the conditions to be met. A pattern author ensures that an executable query based on the pattern's templates (Figure 2.5—*Template*) can be generated when the pattern user instantiates the pattern with names of eMDM elements that satisfy the constraints. It should be noted that the pattern author guarantees that the templates are formulated correctly. In turn, a pattern user can expect to obtain an executable OLAP query by using a particular OLAP pattern if names of eMDM elements of an associated eMDM are bound that satisfy the pattern's constraints.

Table 2.1: Basic macros for pattern templates

Macro	Description
`$expr`	The expression macro `$expr(⟨term⟩, ⟨param1⟩,...,⟨paramN⟩)` expects the name of a business term as its first parameter and one or more parameter bindings, depending on the arity of the specified business term. The macro is evaluated in the context of an enriched multidimensional model and returns an expression snippet of the business term that matches the data model, language, and dialect of the pattern template. The returned expression snippet corresponds to the business term's template with all variables substituted by the provided parameter bindings according to the macro call.
`$dimKey`	The dimension key macro `$dimKey(⟨dim⟩)` expects exactly one dimension name as its parameter binding and is evaluated in the context of an enriched multidimensional model to return the dimension's base level, i.e., a level that is not a parent level.

In order to fully benefit from the pattern-based approach, Happy Milk canonizes additional business vocabulary used in day-to-day discussions and analyses by representing business terms in an unambiguous manner. Consider, for example, the following business terms with SQL expressions formulated in the Oracle dialect. First, the unary calculated measure `Average Milk Yield` calculates the average milk yield from a cube with a `Milk Yield` measure that captures `Liquid In Liter` values, which is represented by the expression `AVG(⟨ctx⟩."Milk Yield")`, where the cube is referenced by the parameter `⟨ctx⟩`. Second, the binary calculated measure `Average Milk Yield Ratio` calculates the ratio of the average milk yields from two cubes, each having an `Average Milk Yield` measure that captures `Liquid In Liter` values, which is represented by the expression `(⟨ctx⟩[1]."Average Milk Yield"/⟨ctx⟩[2]."Average Milk Yield")`, where the cubes are referenced by the parameter `⟨ctx⟩[1]` and `⟨ctx⟩[2]`. Third, the unary cube predicate `Mid Lactation Phase` limits the milkings to animals after between 100 to 200 days of lactation from a cube with a `Lactation` dimension role that captures `No Of Days` values, which is represented by the expression `⟨ctx⟩."Lactation" BETWEEN 100 AND 200`, where the cube is referenced by the parameter

⟨ctx⟩. Finally, the unary dimension predicates `Young Cattle` and `Old Cattle` limit dimensions with a `Date Of Birth` property to less than three or exactly three years or older, which is represented by the expressions `trunc((SYSDATE-`⟨ctx⟩`."Date Of Birth")/365.25)<3` and `trunc((SYSDATE-`⟨ctx⟩`."Date Of Birth")/365.25)>=3`, where the dimensions are referenced via parameters ⟨ctx⟩. Other types of interesting business terms to be defined for Happy Milk are dimensional groupings such as grouping per farm, dimension orderings such as the ordering of facts about farms by town name in ascending order, selection of high average milk yield ratios, and selection of animals that were under heat stress on a particular date based on entries in a precomputed lookup cube.

```
1  INSTANTIATE PATTERN "Breed-Specific Subset-Subset Comparison" AS
2    "Dairy- and Breed-Specific Subset-Subset Comparison" WITH
3          <sourceCube> = "Milking",
4       <baseCubeSlice> = "Mid Lactation Phase",
5        <baseDimSlice> = "Holstein",
6         <compDimRole> = "Cattle",
7           <iDimSlice> = "Young Cattle",
8           <cDimSlice> = "Old Cattle",
9         <joinDimRole> = "Farm",
10          <groupCond> = "Per Farm",
11        <cubeMeasure> = "Average Milk Yield",
12        <compMeasure> = "Average Milk Yield Ratio";
```

Listing 2.1: Instantiation of the breed-specific subset-subset comparison in Figure 2.5

Coming back to the Happy Milk scenario: Assume the company detects a drop in the milk yield of Holstein cattle. A decrease in milk yield may have many reasons but, first of all, the affected Holstein population must be identified. A BI user may start the investigation by calculating the ratio between the average milk yield of young Holstein cattle and the average milk yield of old Holstein cattle, considering only animals with in their mid lactation phase. On a more abstract level, that query corresponds to breed-specific subset-subset comparison: computation of a ratio between the average measure values for each subset of facts referring to a certain breed – in this case, Holstein – where each group is distinguished according to specified characteristics – in this case, young and old.

In order to obtain an executable OLAP query using the breed-specific subset-subset comparison pattern, the pattern user (BI user) simply binds names of available eMDM elements for each pattern parameter in the course of pattern instantiation (Listing 2.1). In the Happy Milk scenario, the cube name `Milking`, which in the Happy Milk eMDM refers to a cube that captures milk yield at farms, is bound to ⟨sourceCube⟩ (Line 3). The

Milking cube in the Happy Milk eMDM has a dimension role Cattle that references a dimension Animal as its domain and thus satisfies the domain and property constraints defined for the cube to be bound to ⟨sourceCube⟩. The name Holstein, which refers to a unary dimension predicate, is bound to the parameter ⟨baseDimSlice⟩ (Line 5) whereas the unary cube predicate name Mid Lactation Phase is bound to the parameter ⟨baseCubeSlice⟩ (Line 4). The ⟨compDimRole⟩ parameter is set to the name Cattle (Line 6) since that role refers to a dimension which can be used for further restriction. The groups of facts that are to be compared are specified by the Young Cattle unary dimension predicate for the ⟨iDimSlice⟩ (Line 7) and the Old Cattle unary dimension predicate for the ⟨cDimSlice⟩ (Line 8). Since the groups should be compared per farm the Farm dimension role is specified as the ⟨joinDimRole⟩ (Line 9) while the ⟨groupCond⟩ parameter is assigned the Per Farm dimension grouping predicate (Line 10). The Average Milk Yield for the ⟨cubeMeasure⟩ (Line 11) indicates the unary calculated measure to be calculated per group. Finally, the Average Milk Yield Ratio for the ⟨compMeasure⟩ (Line 12) indicates the comparative binary calculated measure.

During pattern instantiation, when the pattern user binds values to all parameters, pattern instantiation yields a parameter-free pattern but any derived elements remain unbound until the fully instantiated pattern is grounded in the context of a particular eMDM. In the Happy Milk scenario, with parameters for the breed-specific subset-subset comparison bound as in the previous example, pattern grounding over the Happy Milk eMDM yields an applicable ground pattern that is executed resulting in the SQL query shown in Listing 2.2, which compares per farm the average milk yield of cattle younger than three years that are in their mid lactation phase with cattle older than three years that are in their mid lactation phase, by calculating the ratio over the average milk yield per group.

```
WITH baseCube AS (
  SELECT *
  FROM   "Milking" sc
         JOIN "Animal" a ON
            sc."Cattle" = a."Animal"
  WHERE  sc."Lactation" BETWEEN 100 AND 200 AND
         a."Main Breed" = "Holstein"
),
interestCube AS (
  SELECT jd."Farm Id",
         AVG(bc."Milk Yield") AS "Average Milk Yield"
  FROM   baseCube bc
         JOIN "Farm" jd ON
            bc."Farm" = jd."Farm Id"
         JOIN "Animal" cd ON
            bc."Cattle" = cd."Animal"
  WHERE  trunc((SYSDATE-cd."Date Of Birth")/365.25)<3
  GROUP BY jd."Farm Id"
),
comparisonCube AS (
  SELECT jd."Farm Id",
         AVG(bc."Milk Yield") AS "Average Milk Yield"
  FROM   baseCube bc
         JOIN "Farm" jd ON
            bc."Farm" = jd."Farm Id"
         JOIN "Animal" cd ON
            bc."Cattle" = cd."Animal"
  WHERE  trunc((SYSDATE-cd."Date Of Birth")/365.25)>=3
  GROUP BY jd."Farm Id"
),
SELECT ic."Farm Id",
       ic."Average Milk Yield" AS "Group of Interest",
       cc."Average Milk Yield" AS "Group of Comparison",
       (ic."Average Milk Yield"/cc."Average Milk Yield") AS
           "Average Milk Yield Ratio"
FROM   interestCube ic
       JOIN comparisonCube cc ON
          ic."Farm Id" = cc."Farm Id"
```

Listing 2.2: Executable OLAP query resulting from the instantiation of the breed-specific subset-subset comparison

Chapter 3

Related Work

In this chapter, we first review existing literature on the notion of patterns and discuss the origins of patterns and their application in different domains (Section 3.1). We continue with an overview of patterns in data modeling (Section 3.2) before investigating the role of patterns in data warehouse design (Section 3.2.1). In addition, we provide an overview of pattern-based approaches to data analysis (Section 3.3). In this regard, we discuss existing approaches to composing SQL queries based on query patterns (Section 3.3.1) and pattern-based solutions for exploratory data analysis (Section 3.3.2). We conclude this chapter with an overview of visualization- and model-driven techniques for data analysis (Section 3.4). This chapter is based in parts on related work sections from our previous publications [1], [2]. In summary, no related work covers in its entirety the functionality of the pattern-based approach to multidimensional data analysis.

3.1 Patterns in General

The notion of patterns originates from the field of architecture, where Alexander recognized that the character of places is given by the character of the patterns of events that occur there, which in turn are interlocked with "geometric patterns in the space" [12, p. x]. Such geometric patterns should be considered in the design of buildings and regions in order to maintain a livable environment for the people who inhabit these places [12, pp. x, xi]. To this end, Alexander et al. define such geometric patterns as rules that describe the relationship between the problem, the context in which the problem occurs, and the solution to the problem in that context [29, p. xi]. Thus, a pattern provides instructions to create buildings and regions that solve a recurring problem in a particular environment, where the result may vary each time the pattern is applied in a certain context [12, p. x-xii]. Furthermore, Alexander et al. define a pattern language consisting of 253 geometric patterns [29, p. xi-xii]. Patterns in the pattern language are sorted in descending order by

the scale level that is considered the context under consideration, i.e., the language consists of patterns concerning entire regions or cities as well as patterns for specific rooms of a building [29, p. xii]. This arrangement creates a hierarchy of patterns, allowing a larger pattern to be broken down into smaller ones or viewed in the context of a higher-level pattern [29, p. xii]. Finally, this hierarchy of patterns causes the application of one pattern to potentially affect the application of other patterns, as it may change the context to be considered, e.g., the application of larger patterns defines the context to be considered for smaller patterns [12, p. xiv].

Since its inception by Alexander, the notion of patterns spread across multiple fields. Beck and Cunningham were the first to introduce a kind of patterns to teach object-oriented design for software development in the Smalltalk programming language [30]. To this end, Beck and Cunningham introduced Class, Responsibility, and Collaboration (CRC) cards to teach novices in object-oriented programming the Model-View-Control design pattern [30]. The work of Beck and Cunningham [30] as well as Alexander et al. [12], [29] thus laid the foundation for the development of the software design patterns by Coad [31] as well as Gamma, Helm, Johnson, and Vlissides (also know as the Gang of Four, or GoF) [13]. By identifying patterns in software design, the GoF developed a framework of object-oriented design patterns to facilitate bottom-up design of software systems [13]. These design patterns thus act as building blocks that promote reuse and maintainability throughout the design process. The presented software design patterns were defined in a domain-independent manner to cover object-oriented designs in terms of design, structural, and behavioral aspects [13]. These design patterns have also become accepted as a common communication vocabulary during the software design process [13]. It should be noted that patterns are distinct from frameworks, which provide the context for the employment of reusable components, which in turn may be based on software design patterns [32]. Nevertheless, frameworks may represent reusable design in addition to mere code reuse, depending on their scope and application area. To organize and drive efforts in pattern development in the software engineering domain, the Pattern Languages of Programs (PLoP) conference series was established.

Apart from patterns for software engineering, Coplien, who initially focused on software patterns [33], proposed, in collaboration with Neil and Harrison, organizational patterns for agile software development [34]. The definition of organizational patterns was inspired by findings of Conway, who observed that the structure of software systems is to some extent predetermined by the communication structures of the organization implementing those communication structures [35]; that observation has since become known as *Conway's Law*. Based on Conway's law and the fact that software design patterns [13] are concerned with recurring structures within the software rather than the organization of software

development, the catalog of organizational patterns introduced by Coplien et al. [34] aims to improve the organization through permanent and incremental changes, while taking into account the organization processes, the structure of the organization behind it, and the principles and values that determine the organization's identity [36]. This generative nature of organizational patterns is similar to that of architectural patterns proposed by Alexander et al. in that the application of the pattern changes the context, i.e., the organization. It is worth noting that the organizational patterns of Coplien et al. [34] have laid the foundation for the proliferation of agile software development.

In addition to the architectural and software engineering domain, the notion of pattern is prevalent in other domains as well. In this context, Kohls' extensive work on patterns in general and specifically on the use of patterns in e-learning [20] is particularly noteworthy. Furthermore, in the context of didactics, Muller and Haberman's patterns are worth noting [37], which are used to teach algorithms. For the design and development of business rule management systems, Graham presents a pattern language [38]. In the context of business process modeling, van der Aalst et al. identified workflow patterns [39]. A more comprehensive overview of patterns in other domains is presented by Kohls [20, p. 18-19].

The pattern-based approach to multidimensional data analysis, does not share the generative character of the patterns introduced by Alexander et al. [12] and Coplien et al. [34], since the application of OLAP patterns does not lead to any change in the considered context, i.e., the associated eMDM. Similarly to the software design patterns by Gamma et al. [13], OLAP patterns can be defined in a domain-independent manner. Furthermore, OLAP patterns can also be defined in a domain- and enterprise-specific manner, which allows for pattern organization along different levels of abstraction. However, the hierarchical organization of OLAP patterns must be distinguished from Alexander's hierarchical pattern organization [12], which unlike the hierarchical organization of OLAP patterns refers to a composition of smaller patterns into larger patterns. OLAP patterns, on the other hand, are organized based on their level of abstraction, i.e. their genericity, which allows for OLAP patterns to be applied to different domains and companies. Still, multiple smaller patterns could be defined within the same level of abstraction and then used as building blocks for the composition of more complex patterns.

3.2 Data Modeling Patterns

In contrast to software design patterns, which focus primarily on the design of software, Fowler [11] provides a catalog of analysis patterns that represent conceptual models "that have been useful in one practical context and will probably be useful in others" [11, p. 8]. Thus, Fowler's patterns of analysis focus on representing how people in a certain domain

perceive their world [11, p. xvi]; the thus externalized mental models serve to understand and simplify the problem. Although some analysis pattern may seem (in parts) trivial at first glance, an enormous amount of work can be required to work them out during the process of conceptual modeling to uncover the foundations of a problem [11]. Even though Fowler follows an object-oriented modeling paradigm, Folwer does not focus on the design of software, rather providing conceptual models of businesses [11, p. xv]. In addition to analysis patterns, which are "groups of concepts that represent a common concept in business modeling" [11, p. 8], Fowler also defines supporting patterns that describe how to use and apply analysis patterns [11, p. 8]. In contrast to the patterns described by Alexander et al. [29] and the GoF [13], Fowler discusses not just one but several possible solutions to a problem in a given context in a way that does not follow the pattern styles used by Alexander et al. and the GoF. Fowler's patterns of analysis emerged from a practical context and should be viewed as suggestions rather than prescriptions [11, pp. 8, 12]. Nevertheless, Fowler acknowledges that analysis patterns can be applied across domains since the patterns represent the (domain-independent) business and not the software to be developed [11, p. 10].

Comparable to Fowler [11], Hay [40] defines conceptual models in a relational modeling style. Hay identifies models across businesses and government agencies that can be modeled in a standardized way [40, p. 4]. Hay's focus is on identifying the fundamental nature of enterprises and representing them in a readable format that in turn promotes communication among users, analysts, and designers [40, p. 2]. To this end, Hay defines modeling conventions, i.e., syntactic, positional, and semantic conventions that lead to standardized model representations [40, p. 4-5, 10–22]. Data modeling patterns represented in this way can be reused across multiple business domains, especially early in the modeling process [40, p. 4]. Hay's data modeling patterns promote – in addition to readability – robustness to change and maintainability [40, p. 4]. Other authors, such as Silverston [41], also in collaboration with Agnew [42], along with Blaha [43], Arlows [44], and Batra [45], have identified and described other data modeling patterns at different levels of abstraction. In particular, the work of Silverston and Agnew is noteworthy because they distinguish, different levels of abstraction [41], from patterns for specific domains to domain-independent patterns [42], [46]. It is also worth noting that Blaha identified antipatterns that should be avoided during data modeling [43, p. 95-118]. Blaha also presents – in addition to the domain-independent data modeling patterns – patterns for metamodels (*canonical models*) [43, pp. 157–202]. In this context, more recently, Hay also published patterns for metamodels for modeling data, activities, functions and processes, places, people and organizations, and event and time data [47].

Similarly to Fowler's analysis patterns, OLAP patterns have emerged from experiences

of various industrial research projects. Another similarity to Fowler's patterns is that the pattern-based approach to multidimensional data analysis does not define strict prescriptions, as OLAP patterns can be seen as a starting point for a particular analysis situation, offering suggestions that can be followed to satisfy the information need at hand. Furthermore, the cross-domain nature of analysis patterns identified by Fowler is also reflected in our pattern-based approach by describing the context at a conceptual level rather than at a logical level. In addition, our pattern-based approach to multidimensional data analysis takes into account the levels of abstraction in modeling described by Silverston and Agnew, which are addressed by enterprise-specific, domain-specific, and domain-independent OLAP patterns. Similarly to the metamodels introduced by Blaha and Hay, we present a metamodel for representing patterns in a multidimensional environment, where OLAP patterns are instances of this model. Finally, it is worth noting that unlike Blaha's antipatterns, our OLAP patterns do not present practices to be avoided.

3.2.1 Data Warehouse Design Patterns

The idea of employing patterns in data modeling is not limited to the design and development of online transaction processing (OLTP) systems for data in an operational environment (see Inmon [48, p. 16-18] for more information on OLTP vs. OLAP). Rather, patterns can also be employed for the design and development of OLAP systems or data warehouse systems. Blaha describes the most commonly used design pattern for relational data warehouses, the star schema [43, pp. 84–90]. Even though this design is well-known in the data warehouse domain [48, p. 126-134] [49, p. 123-126], it has not been explicitly described as a pattern. It is worth noting that the star schema (in addition to the snowflake schema [49, p. 121-127] and combinations thereof) is a pattern for the logical design of relational data warehouses. Similarly, Schneider and Forsch-Wilke claim to describe analysis patterns, but in fact present logical relational design patterns for data warehouses [50].

Complementing Blaha, Silverston describes how data model patterns for specific industries, i.e., multidimensional reference models, can be implemented in data warehouses [41, pp. 436-437] and which data model patterns for specific business events can be reused when designing data warehouses [46, pp. 337-406]. Domain-specific data modeling patterns for data warehouses are also a focus of the BIRD approach introduced by Schuetz et al., which provides a metamodel and process for adaptable multidimensional reference models [6]. The BIRD approach is exemplified by the representation of a reference model for the manufacturing domain, which is adapted for a specific company. Other multidimensional reference models are presented in literature, e.g., by Zaiane et al. [51], who describe a data modeling pattern for OLAP cubes to analyze web access, while Boulil et al. describe a data modeling pattern for analysis of large and complex watercourse data [52].

In addition to domain-specific and domain-independent reference data modeling patterns, Viqarunnisa et al. [53] present 17 generic patterns for designing relational data warehouses, ranging from trivial patterns describing the multiple use of dimensions via roles to more complex patterns for dealing with changing dimensions. On top of relational design patterns, Viqarunnisa et al. also define the data warehouse bus matrix pattern, which describes how business goals can be linked to the design [53]. However, the patterns presented by Viqarunnisa et al. are described only superficially and without using a pattern-specific style [33]. A more in-depth description of patterns can be found in Corr [54, p. 163-254], who describes patterns for agile data warehouse design. Corr identifies patterns for modeling dimensions to represent entities (*who and what*), location and time (*where and when*), cause and effects (*why and how*), and corresponding measurements (*how many*) [54, p. 163-254]. In contrast to Viqarunnisa et al., Corr also provides a detailed description of how to implement the patterns [54, p. 163-254]. Similarly to Corr, Jones et al. also identify modeling patterns for dimensions [55].

Finally, Poole et al. describe the Common Warehouse Metamodel (CWM), a standard under the auspices of the Object Management Group (OMG), which is a comprehensive metamodel on the description, access, and exchange of metadata generated during data warehouse processes [56, p. 75-124]. The CWM serves as the basis for describing metadata interchange patterns [56, p.138-168]. The CWM is intended to promote interoperability between different data warehouse systems and applications, which is particularly important when defining ETL processes across data warehouse systems.

The patterns in data warehouse design are to some extent relevant to the pattern-based approach to multidimensional data analysis, as we conceptualize the data warehouse that contains the data to be queried. Compared to Blaha and Schneider as well as Frosch-Wilke, an enriched multidimensional model (eMDM) is described purely on the conceptual level. Nevertheless, OLAP patterns assume that the underlying logical representation follows certain conventions, e.g., a star schema implementation is assumed at the logical level. However, providing additional mapping information would allow for the use of an implementation using a snowflake schema. An eMDM in our pattern-based approach may represent a reference model for a specific domain or may describe general dimensions, and to some extent represent domain-independent cubes. Thus, the reference data modeling patterns presented by Silverston as well as Schuetz et al. can be considered when describing domain-specific patterns. In addition, on the domain-independent level, the universal patterns of Silverston and Agnew, Blaha, and Corr can be considered. Unlike the patterns presented by Viqarunnisa et al., our pattern-based approach does not describe how the multidimensional model should be constructed or what should be considered. Furthermore, no additional metadata for the data warehouse design process is captured in our metamodel.

3.3 Data Analysis Patterns

The analysis of data within a data warehouse is limited to the cubes and dimensions that capture the occurrence of different business events. For this reason, we restrict the notion of data analysis in this section to the definition given by Zohuri and Moghaddam, who define data analysis in the context of data warehouses as a "process for obtaining raw data and converting it into information useful for decision-making by users" [57]. Therefore, to provide useful information for decision making, data from various sources must be preprocessed as part of an ETL routine and inserted into the data warehouse at the appropriate granularity. Once the data warehouse is populated, analysis can be performed by composing OLAP queries, defining reports (based on OLAP query templates), performing statistical analysis, and data mining [58]. In the following, we review related work on the composition of SQL queries in Section 3.3.1, followed by related work on pattern-based solutions for data analysis in Section 3.3.2.

3.3.1 Query Formulation Using SQL Patterns

Previous research by Tropashko [59] and Al-Shuaily [60] investigates the transfer of SQL knowledge in different contexts by introducing SQL patterns. Al-Shuaily introduces SQL patterns for teaching SQL to novices, noting that novices lack the ability to abstract a particular information need in order to map that information need to existing solution strategies [60, p. 84]. Novices also lack problem solving skills, as they find it difficult to divide an information need into sub-problems, and have difficulties in composing the corresponding query [60, pp. 201-203]. The reasons for these issues can be found in a lack of knowledge and experience as well as a missing understanding of basic SQL concepts. The proposed SQL patterns by Al-Shuaily aim to tackle these drawbacks. Al-Shuaily focuses on educational impacts of using SQL patterns to teach basic SQL, which is why the proposed patterns, i.e., dynamic filtering, filter-by-existence, natural- and self-join, and the grouping result pattern, focus on basic SQL query functionality. Nevertheless, the proposed SQL patterns have a positive impact on the understanding of SQL as well as the formulation, translation, and writing of queries [60, p. 300-331]. These SQL patterns are described textually but also using figures and tables to illustrate the operations that are to be performed. The textual description follows the common pattern style used by Alexander et al. [29] and Gamma et al. [13], i.e., each pattern is named, an example is provided, the problem and the context in which it occurs with possibles forces to consider are stated, the solution is detailed, and finally, possible consequences to consider are specified. It should be noted that Al-Shuaily's SQL patterns focus on learning the basic SQL operations but without focusing on the formal basics such as relational algebra. Therefore, in the pedagogical

patterns described by Renaud and Biljon [61], which describe a process for teaching SQL, Al-Shuaily's SQL patterns can only be used as one step of many. Nonetheless, based on Al-Shuaily's work, Sundin and Cutts [62] further developed the pattern approach to teach basic data analysis operations in SQL, in addition to Python and R. Sundin and Cutts aim to teach novice users basic data science operations within a short session using cheat sheets, which in turn provide patterns for basic operations with a linked implementation in the target language. The available basic operations are represented by a set of seven symbols used as placeholders in the corresponding language templates. Sundin and Cutts show that most novices were able to solve the given task in the given time, with the unexpected result that SQL queries were the easiest to formulate in the process [62].

Unlike Al-Shuaily as well as Sundin and Cutts, Tropashko proposes SQL patterns for an expert audience. The SQL patterns described by Tropashko include not only the simple use of conditional and concatenation operators but also workarounds for common SQL problems as well as complex tasks such as handling trees and graphs in SQL, complex constraints based on (user-defined) functions and materialized views, and the mapping of statistical and mathematical questions in SQL [59]. Tropashko discusses common SQL problems without following a classical pattern style but he describes for each problem the underlying causes and multiple solutions. To this end, the reader is guided step by step to understand the rationale behind the solution and how it can be formulated for a specific use case. In contrast to Al-Shuaily's patterns, Tropashko's patterns are not designed to be used in a pedagogical context but as a reference guide experienced users.

There are approaches that facilitate the identification of SQL patterns. Nagy and Cleve sourced the online forum Stack Overflow for SQL queries to find SQL antipatterns by applying data mining techniques [63]. Such SQL antipatterns are also described by Karwin [64] and Dintyala et al. [65]. Dintyala et al. also propose an approach for automatic detection and repair (to some degree) of antipatterns. It is worth noting that Renaud and Biljon [61] advise against using antipatterns to teach SQL to novices, as this would lead to misleading persisted mental models. Nevertheless, the same techniques for identifying antipatterns could be used to find common solutions to certain types of information needs. In addition to obtaining queries from public repositories, queries from benchmarks can also be used as a basis to identify patterns. For example, the well-known benchmark of the Transaction Processing Performance Council (TPC) for decision support systems (TPC-DS) developed by Poess et al. [66], [67], can be considered since it allows common OLAP queries to be generated based on 99 templates for a given use case. Identifying similar groups of queries (or users) is also a field in recommender systems that aim to provide users with appropriate queries for particular analysis situations. Common recommender systems [68], [69] are used to suggest queries to a user based on the user's previous query

behavior (content-based) or on the behavior of similar users (collaborative).

Although Al-Shuaily shares some objectives with the pattern-based approach to multidimensional data analysis, Al-Shuaily focuses on basic SQL query functionality without even considering complex applications such as OLAP and data analysis. Moreover, neither the SQL patterns presented by Al-Shuaily nor those presented by Nagy and Cleve as well as Tropashko are grounded in formal definitions. In addition, Al-Shuaily's patterns do not support query generation as the query format is specified informally, i.e., constants and variables are distinguished from keywords solely by using different fonts. Al-Shuaily lacks constructs to represent optional expressions, also abstracted subqueries are simply specified by an appropriate keyword [60, p. 433-442]. In contrast, Nagy and Cleve provide simple but instantiable query templates, i.e., they replace operation symbols with specific values. This approach is similar to OLAP pattern templates, which link the pattern to its implementation. Nevertheless, Al-Shuaily as well as Nagay and Cleve lack the formality and flexibility of OLAP patterns. Although the solutions (mostly written for Oracle) in Tropashko's patterns are described in general terms, they contain only specific queries without embedded variables and macros. Tropashko also does not provide templates since automatic query generation is not considered. Consequently, users must manually adapt the existing queries to their current context, with all the potential pitfalls that can arise from such reuse of specific queries (see for details Allen and Parson [3]). Furthermore, Tropashko does not consider the data warehouse context, nor does Tropashko follow common styles for pattern description. Instead, Tropashko discusses solutions along with specific examples.

3.3.2 Pattern-Based Approaches to Data Analysis

Nalchigar and Yu [14], [15] introduce solution patterns for machine learning to support organizations with garnering business value by applying machine learning approaches. Nalchigar and Yu take into account typical business cases, i.e. credit approval, fraud detection, or task assignment, which could be supported using machine learning strategies. The proposed solution patterns are based on a conceptual framework for designing business analytics, which consists of three views: business view, analytics design view, and data preparation view. The *business view* allows to model an actor's intention, i.e., to represent strategic goals that can be achieved through decision making based on questions, which in turn are answered by insights from available data. By representing an actor's intent from a business perspective, Nalchigar and Yu diverge from traditional representations for patterns [33]. The *analytics design view* is used to facilitate the selection of suitable approaches to analytical problems, while the *data preparation view* allows for the representation of data preparation tasks that apply a sequence of operators to entities to obtain the data set of interest, i.e., integration, transformation, reduction, and cleaning tasks. In other

words, the data preparation view allows for the representation of extraction, loading, and transformation (ETL) processes based on a conceptual representation of the logical schema. Since the solution patterns for machine learning, at least from a business perspective, focus on generic tasks that may be of relevance in different domains, they can be considered domain-independent. However, if a specific data preparation view is incorporated, it must take into account existing data models, i.e., domain- and enterprise-specific models.

In statistics, data analysis patterns were defined by Unwin [70] to describe the methodology of exploratory data analysis (EDA). These data analysis patterns aim at guiding users through common data exploration tasks, focusing on graphical representation and helpful transformations of the examined data, e.g., outlier detection and comparison of proportions. Unwin organizes these analysis patterns at different levels of abstraction, depending on the genericity of the EDA task being supported. For Unwin, the inclusion of human judgment is essential to a pattern of analysis, which is why Unwin does not see patterns as strict procedures but rather as guidelines. Unwin also views patterns as methods that cannot be automated because they require human involvement.

In contrast to the business view of solution patterns proposed by Nalchigar and Yu, OLAP patterns consider only the question (type) that needs to be answered, without addressing broader decisions or strategic goals. Moreover, the analytics design view of solution patterns finds no counterpart in the OLAP pattern approach. Nonetheless, such an analytics design view could be useful for OLAP patterns, especially if several possible OLAP patterns can be applied to satisfy a particular type of information need. The logical schema considered by the data preparation view is also considered by our pattern-based approach through pattern templates. These pattern templates link the OLAP pattern to its implementation, although they are defined considering its conceptual representation. Unlike Unwin's data analysis patterns that supports EDA, OLAP patterns help users to answer specific questions that arise in particular analysis situations. Similarly to OLAP patterns, Unwin's data analysis patterns are described by stating name, problem, solution, and examples but Unwin's data analysis patterns also describe the resulting context, the rationale behind the pattern, and other related patterns [70]. Furthermore, Unwin acknowledges that patterns must be organized along different levels of abstraction, pointing out that the assignment to these levels is rather subjective.

3.4 Visual Analytics and Model-Driven Analytics

Analysis of multidimensional data can also be supported by specifying the query in other ways than manually composing the query in a platform-specific language. Such query specification can be supported by comprehensive visual interfaces or by explicit modeling of the desired

query. In Section 3.4.1 we present work on specifying queries via graphical interfaces and in Section 3.4.2 we review model-driven analysis approaches to query specification.

3.4.1 Visual Analytics Approaches

Frequently, a visual interface is employed for query composition, sparing the user the intricacies of the underlying (multidimensional) data model. The Duet approach [71] provides, for example, group comparison capabilities for data analysis novices. The idea behind Duet is that novices have difficulties both in selecting strategies and in performing low-level operations for group comparisons (execution barrier) and in interpreting the results, i.e., the visualizations (interpretation barrier) [71]. Duet provides a visualization to define the comparison according to a minimal specification approach, i.e., when a user specifies a group, Duet suggests the most similar and different other groups, and when a user specifies two groups, Duet suggests the most similar and different attributes between these groups [71]. The result is visualized and enriched by a generated simple description to facilitate interpretation.

The insights gained through the development of Duet led to the development of its successor Duo, which aims to support pairwise comparisons of tabular data [72]. Duo is a spreadsheet application that supports comparisons of exactly two groups by decomposing the pairwise comparison into rules that follow a sloppy syntax, i.e., rules to define groups and rules to define attributes to be compared [72]. During the development of Duo, Law et al. identified types of comparisons by collecting questions from 398 unique crowd-workers who were asked for interesting comparisons concerning eight fictitious situations, which resulted in a taxonomy of pairwise comparisons [72]. The taxonomy classifies twelve types of pairwise comparisons along three dimensions. First, the *repetition* dimension indicates whether an object of interest is compared to another single object (simple one-to-one) or whether an object of interest is considered a reference which multiple objects are compared with (simple one-to-many). The *group* dimension indicates the comparison of an object group of interest to a single object group (group one-to-one), an object group of interest to multiple other object groups (group one-to-many), and multiple object groups of interest to multiple other object groups (group many-to-many). Finally, the *attribute* dimension indicates whether or not the attribute to be compared is explicitly specified [72]. The results were incorporated into the design of Duo. Duo is not designed to be used in a data warehouse context which requires, besides the ability to incorporate analytical functions, the dynamic definitions of measures as well as join conditions. The aim of Law et al. is to support exploratory data analysis by guiding novices through the comparison of groups.

Prior to Duet and Duo, Stolte and Hanrahan developed the query visualization interface

Polaris [73], which led to the development of Tableau[1]. Polaris focuses on analyzing, querying, and visualizing multidimensional relational databases, although newer versions of Tableau support other data structures as well. Rather than focusing on ad hoc queries, Polaris primarily supports exploratory data analysis by providing an interactive visualization of both the query and the result. The query is defined by a visual specification within a table-based interface that allows dimensions, measures as well as grouping and filter criteria to be specified along with possible visualisation options. An underlying *table algebra* [73] is used to generate the corresponding queries and, thus, queries can be formulated in an ad hoc manner.

Based on Böhnlein's preliminary work on the Semantic Data Warehouse Model (SDWM) [74], Böhnlein et al. present an approach for visually specifying multidimensional queries [75]. The SDWM represents data warehouses conceptually via measures and dimensions and follows the idea that a potential user is primarily interested in measures and that these can therefore be used as a starting point for the analysis. The visual specification of a multidimensional query represents the elements of the underlying SDWM as graphical elements by encoding the semantics using different shapes, colors, contours, and formats.

The SDWM-based approach allows users to be provided with visual templates based on the SDWM, the templates serving as configurable reports [75]. Each of these templates consists of predefined measures, dimensions, and dimension attributes that are related to each other. The relationships between these template elements are visualized to represent the dependencies between the measures and dimensions. The user specifies the OLAP query by either adding/removing new measures or by selecting the dimension hierarchy levels. Additivity checks are performed to restrict certain aggregations of measures along specific dimensions.

In addition to the composition of queries, QueryViz, introduced by Danaparamita and Gatterbauer [76] and evaluated by Leventidis et al. [77], is a visualization-based approach to represent and understand the meaning of existing SQL queries [77]. The visualization is thereby based on a logical representation of SQL queries – following the first-order logic foundation of SQL – and can be generated by simply providing a textual representation of the SQL query and relevant schema fragments [76], [77]. Although SQL allows a logical statement to be represented by multiple constructs, QueryViz represents each construct uniquely by considering the logical statement behind the employed construct. This allows semantically equivalent SQL queries to be represented in the same way. It should be noted that QueryViz is primarily limited to those SQL queries that express logical statements with negations and quantifiers and can also be represented by certain logical patterns, i.e.,

[1] https://www.tableau.com

simple conjunctive queries, simple cross-table queries, group-by queries with aggregates, simple nested not-exists queries, double-nested not-exists queries, and double-nested for-all queries [77].

Duet uses a predefined dataset as the starting point for data analysis. Law et al. [71], however, do not described how this dataset can be obtained from source data through joins, aggregations, and measure calculations. Thus, a result from applying an OLAP pattern could be used as input for Duet. Unlike Duet, the OLAP pattern approach supports the combination, aggregation, calculation of measures, and comparison of data residing in a multidimensional data model by using the available query capabilities of the underlying host-specific language. Furthermore, our pattern-based approach to multidimensional data analysis addresses the execution barrier described by Law et al. by providing low-level executable templates, while the interpretation barrier is addressed by defining the semantics of the business terms that are to be applied and of the measures that are to be computed. However, our OLAP patterns could be extended to propose appropriate visualizations that take into account the level of measurement and the scale of value sets, i.e. nominal, ordinal, interval, and ratios. The taxonomy of pairwise comparisons identified during the development of Duo provides further confirmation of the general usefulness of the OLAP patterns identified during the agriProKnow project. One-to-one respectively many-to-many comparisons are covered by the subset-to-subset comparison pattern, and one-to-many comparisons are concerned by the subset-to-complement comparison pattern. These OLAP patterns focus on single comparisons with respect to the repetition dimension (simple one-to-one); comparisons where a subset is taken as a reference to compare with others (simple one-to-many) has not yet been covered by an OLAP pattern. Furthermore, the absence of an attribute to compare can be accommodated by the pattern-based approach to multidimensional data analysis by defining different versions of a pattern. The OLAP pattern approach is not limited to pairwise comparisons. Hence, a wide variety of comparison types can be supported through OLAP patterns. Unlike Duet and Duo, we do not consider exploratory data analysis as the main activity, i.e., we expect the BI user to be aware of the business question to be answered.

Similar to Duet and Duo, Polaris supports the application of filters which are, however, restricted to simple expressions; complex business terms as in our pattern-based approach are not directly supported. In addition, computed metrics apply only to that fact class (cube) for which they were defined, while calculated measures in our approach are independent of a concrete cube as long as the necessary structure is provided in the application context. This is possible because our OLAP patterns – differing from the Polaris approach [73] – are not defined on a logical relational data model but on a conceptual multidimensional data model.

The visual approach proposed by Böhnlein et al. that uses the SDWM to specify multidimensional queries lacks the abstraction of OLAP patterns because the visual specification focuses only on case-specific templates that are restricted to a corresponding cube. The user is therefore limited to adapting the existing templates by (de)selecting measures and simple restrictions. In addition to the limited ability to create ad hoc queries, OLAP queries targeting multiple cubes are not considered. In contrast to Böhnlein et al., our pattern-based approach offers the necessary expressiveness and flexibility to define complex business terms and to support heterogeneous OLAP patterns. Although Böhnlein et al. do not identify patterns in the traditional sense they classify measures into atomic and complex measures. Atomic measures in SDWM can be compared to simple multidimensional aggregations, which are captured in our approach by the non-comparative OLAP patterns. Complex measures in SDWM are limited to measure ratios relying on independent sets, subsets and base sets, or further dimension level conditions. The pattern-based approach to multidimensional data analysis avoids these limitations by supporting arbitrary comparisons.

Finally, the logical patterns identified by Danaparamita and Gatterbauer represent non-comparative query patterns that can be added to the catalog of existing OLAP patterns (see Appendix B). Furthermore, the OLAP pattern approach can benefit from QueryViz as it can be used to visualize the SQL templates of an OLAP pattern, thus positively contributing to the comprehensibility of OLAP patterns.

3.4.2 Model-Driven Analysis Approaches

Model-driven approaches to data analysis are also common. For example, Pardillo et al. observed that OLAP queries that should be composed are usually not considered during the design of the system but only once the data warehouse has been implemented [78]. However, to allow designers to verify their conceptual data warehouse design, Pardillo et al. extend the Object Constraint Language (OCL) of the Unified Modeling Language (UML) with a predefined set of OLAP operators. This Model-Driven Architecture (MDA) allows to represent platform-independent OLAP queries that can be automatically translated into executable SQL code [78]. In addition, the conceptual query representation allows for early validation of the designed data warehouse and also allows for consideration of the expected workload through appropriate index and view selection [78]. It is worth noting that for modeling OLAP queries over a conceptual multidimensional model, Pardillo et al. use the conceptual representation of the actual multidimensional data, i.e., the logical schema represented by UML class diagrams serves as the basis. OLAP operations are represented in OCL by macros that can be parameterized. Pardillo et al. translate the conceptual OLAP query into SQL by substituting the macros with the corresponding piece of code that is customized by the specified parameters, while the OCL code is directly translated into SQL.

To this end, different SQL templates are considered depending on the logical realization, i.e., star or snowflake schema.

Cabot et al. extend OCL with aggregation functions [79] that can be used to compose OLAP queries following Pardillo et al. To this end, distributive functions, algebraic functions, and holistic functions are implemented by OCL operators [79]. This allows OLAP queries represented in OCL to be easily transformed into executable SQL statements by – in addition to the steps described by Pardillo et al. – unfolding the aggregation functions used, i.e., the unfolded aggregation functions contain only OCL operators for which existing transformation rules can be applied [79].

In contrast to Pardillo et al., our pattern-based approach to multidimensional data analysis does not directly model OLAP queries conceptually. Rather, our approach specifies an interface for when and how to compose the corresponding OLAP query. Nevertheless, we share common features such as automatic code generation. Although we did not consider the MDA from the outset, our pattern-based approach can be classified as model-driven engineering, since we allow for automatic code execution from a pattern model. However, our pattern-based approach differs from the standard MDA perspective [80] where a Platform-Independent Model (PIM) is transformed into a Platform-Dependent Model, which in turn is transformed into executable code. The eMDM without expressions represents a PIM, but an eMDM with business terms that are associated with platform-specific templates represents a Platform-Specific Model (PSM). Similarly, OLAP patterns without templates represent PIMs, but when templates are also considered, OLAP patterns represent PSMs. In addition, our pattern-based approach does not perform model-to-model transformation. Rather, OLAP patterns are transformed directly into platform-specific code. Furthermore, Pardillo et al. conceptually represent OLAP queries using OCL on UML class diagrams, as these represent the actually stored data. Our pattern-based approach instead assumes that the conceptual multidimensional model also represents the stored data. The transformation of OLAP queries defined in OCL is similar to the execution of OLAP patterns, as Pardillo et al. use parameterized macros that are replaced by the corresponding code snippets. Unlike Cabot et al. aggregation functions do not need to be conceptually represented in our pattern-based approach; only the corresponding expression is defined as a template. While this makes the pattern-based approach more flexible, it always requires manual definition of the expressions. However, by manually defining the template expressions, the OLAP patterns can take advantage of all the functionalities offered by the underlying query language.

Chapter 4

Enriched Multidimensional Models

In this chapter, we present and define the notion of enriched multidimensional model (eMDM) in Section 4.1, which forms the basis for OLAP pattern definition and usage. To this end, we provide formal definitions for multidimensional models that are enriched by different types of business terms. In addition, we define how business terms can be used in patterns. Furthermore, we introduce a definition language for eMDM; we refer to Appendix A for a definition of the grammar. Finally, we introduce a graphical representation for the elements of enriched multidimensional models in Section 4.3.

4.1 Enriching Multidimensional Models with Business Terms

An eMDM comprises (i) a multidimensional model (also known as multidimensional schema) with multiple value sets that is enriched by (ii) a set of business terms. Multidimensional models are composed of various types of entities and properties. Cubes (or fact schemas) and dimensions are the primary types of elements in multidimensional models – the entity types. Apart from these entity types, measures, dimension roles, levels, and attributes exist – the property types. Furthermore, value set types serve to define domains of measures, levels, and attributes. An eMDM also comprises different types of business terms that can be applied to different types of entities, serving different query purposes, i.e., selection, grouping, ordering, and computation of calculated measures. The type of business term determines on which type of entity the business term can be applied; types of business terms are (cube) calculated measure, cube predicate, cube ordering, dimension grouping, dimension predicate, and dimension ordering.

Definition 1 (Model Element Types)**.** An enriched multidimensional model is based on the set of *model element types* $\mathbb{O}$. Set $\mathbb{O}$ consists of the model element types $Cube$ (short: Q), $Dimension$ (D), $Measure$ (M), $DimensionRole$ (R), $Level$ (L), $Attribute$ (A), $NumberValueSet$ (U), $StringValueSet$ (I), $UnaryCalculatedMeasure$ (QMU), $BinaryCalculatedMeasure$ (QMB), $CubeOrdering$ (QO), $UnaryCubePredicate$ (QPU), $BinaryCubePredicate$ (QPB), $DimensionGrouping$ (DG), $DimensionOrdering$ (DO), $UnaryDimensionPredicate$ (DPU), $BinaryDimensionPredicate$ (DPB). The sets of entity types $\mathbb{E} = \{\mathsf{Q}, \mathsf{D}\}$, property types $\mathbb{P} = \{\mathsf{M}, \mathsf{R}, \mathsf{L}, \mathsf{A}\}$, value set type $\mathbb{V} = \{\mathsf{U}, \mathsf{I}\}$, and business term types $\mathbb{B} = \{\mathsf{QMU}, \mathsf{QMB}, \mathsf{QO}, \mathsf{QPU}, \mathsf{QPB}, \mathsf{DG}, \mathsf{DO}, \mathsf{DPU}, \mathsf{DPB}\}$, are subsets of set $\mathbb{O}$, i.e., $\mathbb{E} \subset \mathbb{O}$, $\mathbb{P} \subset \mathbb{O}$, $\mathbb{V} \subset \mathbb{O}$, and $\mathbb{B} \subset \mathbb{O}$.

For the purposes of OLAP pattern definition and usage, a multidimensional model serves as a conceptual representation of a specific data warehouse schema and consists of entities that are interrelated and further described by properties. Each entity can be characterized by multiple properties, each property being owned by exactly one entity. A cube may have cube properties, i.e., measures and dimension roles, a dimension may have dimension properties, i.e., levels and (descriptive) attributes. Measures quantify business events of interest that are represented as data points in a multidimensional cube. For each measure a number value set specifies the measure's domain, i.e., the measure maps into a specific set of values. Specifying the domain for measures avoids flawed comparisons of measure values and enables comparison across domains if corresponding conversions are specified. For example, measure values from the number value set `Liquid In Liter` can be compared to measure values from the number value set `Liquid In Milliliter` if converted. Dimension roles link dimensions to cubes, with the linked dimensions allowing BI users to view measure values from different perspectives and at different granularity levels. Within a cube, dimension roles serve as alias names for dimensions, which allows to assign a dimension to the same cube multiple times using different alias names. A dimension's alias then corresponds to a dimension role name and the referenced dimension becomes the dimension role's domain. Each dimension is characterized by hierarchically ordered levels of granularity. For each dimension property, its domain is specified by a value set to avoid incorrect joins and restrictions. A unary dimension predicate, for example, that restricts a breed level considering abbreviated breed names cannot be applied to a level with non-abbreviated breed names even though both levels may have the same name. In order to keep the definitions concise, we restrict the formalization of multidimensional models to the relevant concepts for the definition of OLAP patterns. Therefore, we omit definitions of the usual consistency criteria for dimension hierarchies and refer to related work [81] for a comprehensive study of dimension hierarchies; the pattern-based approach is agnostic to the supported types of dimension hierarchies. This is also the reason why we did not reuse the Common Warehouse

Metamodel [56, p. 235-278].

Definition 2 (Enriched Multidimensional Model)**.** An *enriched multidimensional model* $s = (\mathbb{O}, O, N, \texttt{type}, \texttt{name}, \texttt{owner}, \texttt{domain}, \texttt{return}, \texttt{rollsUpTo}, \texttt{describe})$ consists of the sets of model element types $\mathbb{O}$, model elements O, model element names N, and a set T for each model element type $\mathsf{T} \in \mathbb{O}$ denoted by $T = \{o \in O \mid \texttt{type}(o) = \mathsf{T}\}$, where total function $\texttt{type} : O \to \mathbb{O}$ defines the type of each model element. These sets of model elements of a model element type serve to define the sets of entities $E = Q \cup D$, cube properties $PQ = M \cup R$, dimension properties $PD = L \cup A$, properties $P = PQ \cup PD$, value sets $V = U \cup I$, cube business terms $BQ = QMU \cup QMB \cup QPU \cup QPB \cup QO$, dimension business terms $BD = DQ \cup DPU \cup DPB \cup DO$, unary business terms $BU = QMU \cup QPU \cup QO \cup DG \cup DPU \cup DO$, binary business terms $BB = QMB \cup QPB \cup DPB$, and business terms $B = BQ \cup BD$ (Figure 4.1). Besides, the type function the enriched multidimensional model s also comprises the following functions:

- Total function $\texttt{name} : O \to N$ assigns to each model element a name.
- Total function $\texttt{owner} : P \to E$ defines the owning entity of each property, such that $\texttt{owner}|_{M \cup R} : (M \cup R) \to Q$, and $\texttt{owner}|_{L \cup A} : (L \cup A) \to D$.
- Total function $\texttt{domain} : P \to (D \cup V)$ defines the domain of each property, such that $\texttt{domain}|_R : R \to D$, $\texttt{domain}|_M : M \to U$, $\texttt{domain}|_L : L \to V$, and $\texttt{domain}|_A : A \to I$.
- Total function $\texttt{return} : (QMU \cup QMB) \to U$ defines the type of the return value of a calculated measure by a number value set.
- Total function $\texttt{rollsUpTo} : L \to 2^L$ defines for a level its parent levels. A level and its parent level types belong to the same dimension, i.e., $\forall l, l' \in L : l \in \texttt{rollsUpTo}(l') \Rightarrow \texttt{owner}(l) = \texttt{owner}(l')$.
- Finally, total function $\texttt{describe} : A \to L$ defines the level that an attribute describes. An attribute and the level that the attribute describes belong to the same dimension, i.e., $\forall a \in A : \texttt{owner}(a) = \texttt{owner}(\texttt{describe}(a))$.

The different model element types introduced in set $\mathbb{O}$ allow to declare the type of business term parameters and pattern variables (pattern parameters and derived elements). This makes it possible to restrict the names of the eMDM elements to be bound to those of a specific model element type and also to define the scope in which business term parameters and pattern variables can be used within the template of a business term or pattern. In addition, the set of model element types $\mathbb{O}$ can be easily extended if the expressiveness of

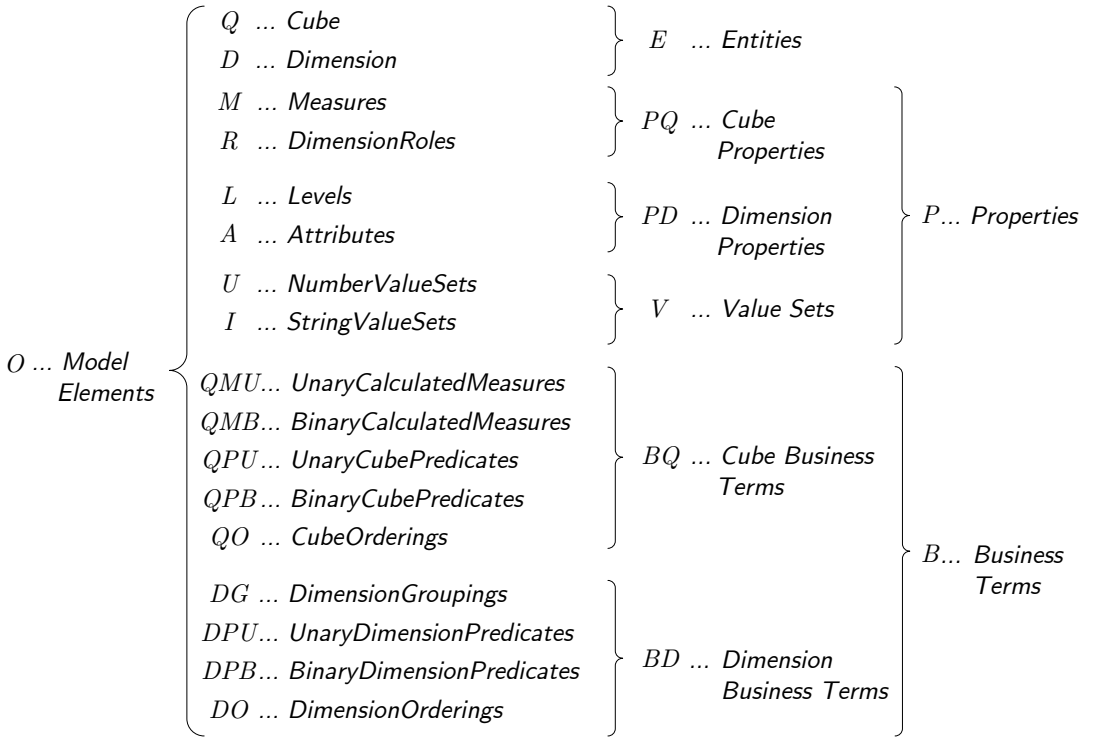


Figure 4.1: Sets of model elements introduced in Definition 2 (except sets of unary and binary business terms)

the eMDM is to be improved by introducing new types of multidimensional model elements and business terms.

Example 4.1 (Multidimensional model)**.** Listing 4.1 defines a cube named `Milking` which comprises two measures (Lines 2-5), one of which is named `Milk Yield`, which has a number value set named `Liquid In Liter` as domain (Line 3), i.e., $\exists q \in Q : \exists m \in M\ \exists u \in U : \texttt{name}(q) = \text{“}Milking\text{”} \wedge \texttt{name}(m) = \text{“}Milk\ Yield\text{”} \wedge \texttt{owner}(m) = q \wedge \texttt{name}(u) = \text{“}Liquid\ In\ Liter\text{”} \wedge \texttt{domain}(m) = u$ (see Figure 2.4). The `Milking` cube further owns five dimension roles, with the corresponding dimensions as their domains (Lines 7-13), one of which is a dimension role named `Cattle` with a dimension named `Animal` as its domain (Line 8), i.e., $\exists q \in Q : \exists r \in R\ \exists d \in D : \texttt{name}(q) = \text{“}Milking\text{”} \wedge \texttt{name}(r) = \text{“}Cattle\text{”} \wedge \texttt{owner}(r) = q \wedge \texttt{name}(d) = \text{“}Animal\text{”} \wedge \texttt{domain}(r) = d$. Listing 4.2 defines a dimension named `Animal` which owns four levels (Lines 2-7), one of which is named `Animal` (Line 3), i.e., $\exists d \in D : \exists l \in L : \texttt{name}(d) = \text{“}Animal\text{”} \wedge \texttt{name}(l) = \text{“}Animal\text{”} \wedge \texttt{owner}(l) = d$ (see Figure 2.4). The `Animal` level is further described by an attribute named `Animal Name` (Line 10 and Line 17), i.e., $\exists d \in D : \exists l \in L\ \exists a \in A : \texttt{name}(d) = \text{“}Animal\text{”} \wedge \texttt{name}(l) = \text{“}Animal\text{”} \wedge \texttt{owner}(l) = d \wedge \texttt{name}(a) = \text{“}Animal\ Name\text{”} \wedge \texttt{owner}(a) = d \wedge \texttt{describe}(a) = l$. The levels of the `Animal` dimension are arranged in a aggregation hierarchy (Lines 14-16),

e.g., the `Animal` level rolls-up to the `Main Breed` level, i.e., $\exists d \in D : \exists l, l' \in L : \texttt{owner}(l) = d \wedge \texttt{owner}(l') = d \wedge \texttt{name}(l) = \text{“}Animal\text{”} \wedge \texttt{name}(l') = \text{“}Main\ Breed\text{”} \wedge \texttt{rollsUpTo}(l) = l'$. Each level and attribute has a domain (Lines 3-6 and Line 10), one of which is a level named `Main Breed` that has a value set named `Breed Name` as its domain, i.e., $\exists l \in L : \exists v \in V : \texttt{name}(l) = \text{“}Main\ Breed\text{”} \wedge \texttt{name}(v) = \text{“}Breed\ Name\text{”} \wedge \texttt{domain}(l) = v$. ♢

```
1  CREATE CUBE "Milking" WITH
2     MEASURE PROPERTIES
3        "Milk Yield":"Liquid In Liter";
4        "Fat Content":"Percent Per Liter";
5     END MEASURE PROPERTIES;
6
7     DIMENSION_ROLE PROPERTIES
8        "Cattle":"Animal";
9        "Milking Time":"Time";
10       "Lactation":"Lactation";
11       "Calving":"Calving";
12       "Farm":"Farm";
13    END DIMENSION_ROLE PROPERTIES;
14 END CUBE;
```

Listing 4.1: Definition of Happy Milk's `Milking` cube

```
1  CREATE DIMENSION "Animal" WITH
2     LEVEL PROPERTIES
3        "Animal":"Animal Code";
4        "Date Of Birth":"Date";
5        "Main Breed":"Breed Name";
6        "Dam":"Animal Code";
7     END LEVEL PROPERTIES;
8
9     ATTRIBUTE PROPERTIES
10       "Animal Name":"Name";
11    END ATTRIBUTE PROPERTIES;
12
13    CONSTRAINTS
14       "Animal" ROLLS_UP_TO "Date Of Birth";
15       "Animal" ROLLS_UP_TO "Main Breed";
16       "Animal" ROLLS_UP_TO "Dam";
17       "Animal" DESCRIBED_BY "Animal Name";
18    END CONSTRAINTS;
19 END DIMENSION;
```

Listing 4.2: Definition of Happy Milk's `Animal` dimension

Applying the unique name assumption to entities, values sets, business terms, and properties in the agriProKnow project [4] turned out to be too rigid, as it limited the ability to express sophisticated multidimensional models. In this thesis we relax the unique name assumption. For simplicity's sake, without loss of generality, we assume that names of entities, value sets, and business terms are unique within an eMDM whereas property names are unique only per owning entity.

Definition 3 (Proper Naming in Enriched Multidimensional Models)**.** The elements of an enriched multidimensional model s are *properly named*, if, and only if,

- entities, value sets, and business terms have a unique name, i.e., $\texttt{name}|_{E \cup V \cup B}$ is injective, and
- property names are unique in the context of the owning entity, i.e., $\forall p, p' \in P : \texttt{name}(p) = \texttt{name}(p') \wedge \texttt{owner}(p) = \texttt{owner}(p') \Rightarrow p = p'$.

The names of entities, properties, and value sets along with the corresponding types available in an eMDM constitute the vocabulary used to define business terms. For the purposes of formalization, in order to refer to the names of all the model elements of a specific type in an eMDM, corresponding sets of element names are defined, e.g., the set N_{D} which consists of the names of the dimensions in an eMDM.

Definition 4 (Element Names of an Enriched Multidimensional Model)**.** For each set T for a particular model element type in an enriched multidimensional model s a set of model element names N_T is denoted by $N_T = \{n \in N \mid \exists o \in T : \texttt{name}(o) = n\}$.

The business terms defined in an eMDM conceptually represent – in addition to the multidimensional model elements – business vocabulary used in data analysis. A business term consists of a set of constraints and a set of expression templates. In addition, for business terms that represent calculated measures, a return type is specified. A business term is either unary or binary, i.e., it is applied to either one or two entities. The entities to be applied to are passed during business term application as context parameters. A unary business term has one context parameter (arity one), a binary business term has two (arity two). A context parameter is a cube for cube business terms and a dimension for dimension business terms. Although business terms with an arity above two could easily be included, we consider only unary and binary business terms sufficient for most applications. Constraints over the context parameters impose restrictions on the model elements that may be bound to the parameters during application. The expression template is specific to a query language and dialect, representing executable semantics of the business term

as a code snippet, with references to the parameters. A business term may have different expression templates for different systems.

Definition 5 (Business Term)**.** A *business term* $b = (BP, CT, CP, CD, TPL)$ in an enriched multidimensional model s comprises a set of implicit context parameters BP derived from the type of b in s, the constraint relations CT, CP, and CD (see Definition 6) as well as the set TPL of templates. Depending on the arity of b's type in s, the set of context parameters can be derived as follows:

- if b is defined over a single entity (unary business term) the set of context parameters BP consists of an implicit context parameter $\langle \texttt{ctx} \rangle$, i.e., $\forall b \in BU : BP = \{\langle \texttt{ctx} \rangle\}$, or
- if b is defined over a pair of entities (binary business term) the set of context parameters BP consists of two implicit context parameters $\langle \texttt{ctx} \rangle[1]$ and $\langle \texttt{ctx} \rangle[2]$, i.e., $\forall b \in BB : BP = \{\langle \texttt{ctx} \rangle[1], \langle \texttt{ctx} \rangle[2]\}$,
- with a derived total function $\texttt{bptype} : BP \to \mathbb{E}$ that assigns an entity type as domain to each context parameter, the type of which is derived from the business term's type, such that for a cube business term the entity type is Q, i.e., $\forall b \in BQ : \forall v \in BP : \texttt{bptype}(v) = \text{Q}$, and for a dimension business term the entity type is D, i.e., $\forall b \in BD : \forall v \in BP : \texttt{bptype}(v) = \text{D}$.

Example 4.2 (Business Term)**.** Listing 4.3 defines a business term named `Average Milk Yield` on top of Happy Milk's multidimensional model as a unary calculated measure, i.e., $\exists b \in QMU : \texttt{name}(b) = \text{“}Average\ Milk\ Yield\text{”}$. `Average Milk Yield` is applied to a cube referred to by the context parameter $\langle \texttt{ctx} \rangle$ (Line 1), i.e., $BP = \{\langle \texttt{ctx} \rangle\}$ and $\texttt{bptype}(\langle \texttt{ctx} \rangle) = \text{Q}$. The application of the business term to the argument cube leads to the computation of a calculated measure from a number value set named `Liquid In Liter` (Line 7). The cube, in turn, can be specified by binding a name to the context parameter $\langle \texttt{ctx} \rangle$. Listing 4.4 defines, in addition to a description (Lines 1-5), a SQL template expression $t \in TPL$ for an Oracle database providing an executable definition of the term's semantics (Lines 7-11). ♢

```
1 CREATE UNARY_CALCULATED_MEASURE "Average Milk Yield"
     APPLIES TO <ctx>:CUBE WITH
2   CONSTRAINTS
3     <ctx> HAS MEASURE "Milk Yield";
4     <ctx>."Milk Yield":"Liquid In Liter";
5   END CONSTRAINTS;
6
7   RETURNS "Liquid In Liter";
8 END UNARY_CALCULATED_MEASURE;
```

Listing 4.3: Definition of `Average Milk Yield` business term

```
1 CREATE TERM DESCRIPTION FOR "Average Milk Yield" WITH
2   LANGUAGE = "English";
3   ALIAS = "Mean Milk Yield";
4   DESCRIPTION = "Calculation of the average milk
       yield of milking events";
5 END TERM DESCRIPTION;
6
7 CREATE TERM TEMPLATE FOR "Average Milk Yield" WITH
8   LANGUAGE = "SQL";
9   DIALECT = "ORACLEv11";
10  EXPRESSION = "AVG(<ctx>.""Milk Yield"")";
11 END TERM TEMPLATE;
```

Listing 4.4: Description and template definition of `Average Milk Yield` buiness term

Business term constraints define conditions to be satisfied by elements (of the associated multidimensional model) referred to by their name in the definition or during the application of the business term. The associated multidimensional model is represented by the cube, dimension and value set elements available in the eMDM of the business term. Again, by borrowing the design-by-contract metaphor from software engineering, the specification of the constraints can be seen as a contract that defines the conditions to be met in order to obtain an executable expression. In turn, it is ensured that the template expressions of a business term are formulated to produce an executable expression when entities are bound to the context parameters that satisfy the conditions. We distinguish type, property, and domain constraints for business terms. A *type constraint* specifies the expected type of an entity – denoted by a constant name, i.e., an entity with the specified name and type must exist in the associated multidimensional model. Type constraints ensure that references to entities are used in an expression template only in places where it makes sense semantically. A *property constraint* specifies for an entity – referenced by the name bound to a context

parameter or by a constant name – which property of what type is expected to be present in a multidimensional model in the context of which the business term is applied; the property to be provided is referred to by a name constant. A *domain constraint* specifies that an entity – referenced by a name bound to a parameter or by a constant name – must have a property of a certain domain. Depending on the type of property, the domain refers either to a value set or a dimension; both the property to be provided and the domain are referred to by name constants.

Definition 6 (Business Term Constraints)**.** A business term b's constraints are represented by relations:

- $CT \subseteq N_E \times \mathbb{E}$ that represents a business term's type constraints,
- $CP \subseteq (BP \cup N_E) \times \mathbb{P} \times N_P$ that represents the business term's property constraints, and
- $CD \subseteq (BP \cup N_E) \times N_P \times (N_V \cup N_D)$ that represents the business term's domain constraints.

Example 4.3 (Business Term Constraints)**.** In Listing 4.3, the `Average Milk Yield` unary calculated measure defines two constraints. A property constraint requires a measure property named `Milk Yield` to be part of the entity referred to by the context parameter ⟨`ctx`⟩ (Line 3), i.e., (⟨`ctx`⟩, M, "$Milk\ Yield$") $\in CP$. A domain constraint requires the entity referred to by the context parameter ⟨`ctx`⟩ to provide a property named `Milk Yield` with a domain named `Liquid In Liter` (Line 4), i.e., (⟨`ctx`⟩, "$Milk\ Yield$", "$Liquid\ In\ Liter$") $\in CD$. ♢

OLAP patterns are formulated using the names of model elements along with the available types in an eMDM. Similar to relational databases, where users interact through queries that reference tables and columns by name rather than by object identifier, patterns are defined by specifying names of eMDM elements. For the purposes of formalization, we define a set of views over the eMDM to represent the available model elements and their relationships using names; these views represent the vocabulary for pattern definition. We distinguish type, property, and domain views as well as views regarding the return type of business terms and the business terms itself. The *type view* represents entities, value sets, and business terms by its name and type. The *property view* represents each property by the name of its owning entity, its type, and its name, while a *domain view* represents the domain of each property by the name of the property's owning entity, the name of the property, and the name of its domain. In addition, the *return view* also represents the number value set returned by calculated measures by the name of the business term and

the name of the number value set. Finally, the *term view* represents the business term definitions along with their names.

Definition 7 (Multidimensional Model Views)**.** For an enriched multidimensional model s with proper naming, *view* relations are defined and populated as follows:

- $\texttt{hasType} \subseteq (N_E \cup N_V \cup N_B) \times (\mathbb{O} \setminus \mathbb{P})$,
  with $\forall o \in (E \cup V \cup B) : (\texttt{name}(o), \texttt{type}(o)) \in \texttt{hasType}$,
- $\texttt{hasProperty} \subseteq N_E \times \mathbb{P} \times N_P$,
  with $\forall p \in P : (\texttt{name}(\texttt{owner}(p)), \texttt{type}(p), \texttt{name}(p)) \in \texttt{hasProperty}$,
- $\texttt{hasDomain} \subseteq N_E \times N_P \times (N_V \cup N_D)$,
  with $\forall p \in P : (\texttt{name}(\texttt{owner}(p)), \texttt{name}(p), \texttt{name}(\texttt{domain}(p))) \in \texttt{hasDomain}$,
- $\texttt{hasReturn} \subseteq (N_{QMU} \cup N_{QMB}) \times N_U$,
  with $\forall b \in (QMU \cup QMB) : (\texttt{name}(b), \texttt{return}(b)) \in \texttt{hasReturn}$
- $\texttt{hasTerm} \subseteq N_B \times B$,
  with $\forall b \in B : (\texttt{name}(b), b) \in \texttt{hasTerm}$

Example 4.4. Considering Happy Milk's eMDM as defined by Example 4.1 and Example 4.2, the following view entries can be derived: $\{(\text{“}Milking\text{”}, \mathsf{Q}), (\text{“}Animal\text{”}, \mathsf{D}), (\text{“}Liquid\ In\ Liter\text{”}, \mathsf{U})\} \subseteq \texttt{hasType}$, $\{(\text{“}Milking\text{”}, \mathsf{M}, \text{“}Milk\ Yield\text{”}), (\text{“}Animal\text{”}, \mathsf{L}, \text{“}Animal\text{”})\} \subseteq \texttt{hasProperty}$, $\{(\text{“}Milking\text{”}, \text{“}Milk\ Yield\text{”}, \text{“}Liquid\ In\ Liter\text{”}), (\text{“}Animal\text{”}, \text{“}Animal\text{”}, \text{“}Animal\ Name\text{”})\} \subseteq \texttt{hasDomain}$, $\{(\text{“}Average\ Milk\ Yield\text{”}, \text{“}Liquid\ In\ Liter\text{”})\} \subseteq \texttt{hasReturn}$, $\exists b \in QMU : (\text{“}Average\ Milk\ Yield\text{”}, b) \in \texttt{hasTerm}$. ♢

4.2 Usage of Business Terms

The use of a business term in the context of OLAP patterns requires the user to *apply* the business term by binding names to the term's context parameters. The application returns a new business term, the context parameters of which are *substituted* with names of entities in the constraints. In addition, for each context parameter, the returned business term contains a derived type constraint consisting of the name bound to the parameter and its declared type. It should be noted that the substitution does not affect the business term's template.

Definition 8 (Business Term Application)**.** An *application* δ^b_σ of a business term $b = (BP, CT, CP, CD, TPL)$ given a total substitution function $\sigma : BP \to N_E$, with BP the set of b's context parameters, returns a new business term b', with $b' = (BP, CT', CP', CD', TPL)$ where $CT' = \{(\sigma(v), \texttt{bptype}(v)) | v \in BP\} \cup CT$, and CP', CD' correspond to CP, CD, respectively, with variables substituted according to σ.

Example 4.5 (Business Term Application)**.** The application of the `Average Milk Yield` business term (Example 4.2) with the name `Milking` bound to the context parameter $\langle$`ctx`$\rangle$, i.e., δ^b_σ with $\sigma(\langle\texttt{ctx}\rangle) =$ "$Milking$", yields a new business term, with all occurrences of the context parameter $\langle$`ctx`$\rangle$ being substituted by `Milking` in the constraints of the new business term and a new type constraint for the bound context parameter $\langle$`ctx`$\rangle$, i.e., ("$Milking$", Q) $\in CT$. ♢

Since only names of expected entities are provided during business term application, the resulting business term may not be valid when considering the associated multidimensional model, i.e., no entities corresponding to the bound names may be available that satisfy the specified constraints. Thus, a business term application is *valid* considering the associated multidimensional model if, and only if, elements can be found in the multidimensional model that conform to the expected names and constraints, i.e., (i) a type constraint in the context of the associated multidimensional model is satisfied if, and only if, an entity with the expected type and name can be identified in the associated multidimensional model, (ii) a property constraint in the context of the associated multidimensional model is satisfied if, and only if, an entity with the corresponding property of the specified property type exists in the associated multidimensional model, and (iii) a domain constraint in the context the associated multidimensional model is satisfied if, and only if, an entity exists that has a property with the specified domain.

Definition 9 (Valid Business Term Application)**.** An application δ^b_σ of a business term $b = (BP, CT, CP, CD, TPL)$ given a total substitution function σ is a *valid business term application* to an enriched multidimensional model s with multidimensional model views `hasType`, `hasProperty`, and `hasDomain` if, and only if, it returns a business term b' whose constraints are satisfied, i.e., $CT \subseteq$ `hasType`, $CP \subseteq$ `hasProperty`, and $CD \subseteq$ `hasDomain`.

Example 4.6 (Valid Business Term Application)**.** The business term application in Example 4.5 is valid, since the constraints of the returned business term can be satisfied in the context of Happy Milk's multidimensional model as follows. A cube named `Milking` is present (Listing 4.1), satisfying the derived type constraint, i.e., ("$Milking$", Q) $\in$ `hasType`. The `Milking` cube has a measure named `Milk Yield` (Listing 4.1, Line 3), satisfying the property constraint, i.e., ("$Milking$", M, "$Milk\ Yield$") $\in$ `hasProperty`. The `Milking` cube

also has a property named `Milk Yield` with domain `Liquid In Liter` (Listing 4.1, Line 3), satisfying the domain constraint, i.e., $(\text{“}Milking\text{”}, \text{“}Milk\ Yield\text{”}, \text{“}Liquid\ In\ Liter\text{”}) \in$ `hasDomain`. ♢

The representation of a domain-specific vocabulary through business terms frees BI users from the burden of having to define the term's expression manually, especially since domain vocabularies can contain several thousand terms (see AGROVOC [21] in dairy farming or SNOMED CT [22] in the health insurance domain). In addition, business terms free BI users from having to check whether a business term is even suitable for use within an OLAP query.

4.3 Enriched Multidimensional Model Notation

We employ a graphical notation for the illustration of eMDM definitions. Even if a graphical and a textual representation convey the same information, a graphical representation can be perceived more quickly, more easily, and more accurately, so that the information can be processed more efficiently [82]; visual notations impact cognitive effectiveness as much as formal descriptions of semantics [83]. The design principles for graphical notations proposed by Moody [83] serve as guidelines to achieve a cognitively effective graphical eMDM notation. The proposed eMDM notation aims for semiotic clarity, as each model element is represented by exactly one notation element. Types of notational elements are distinguished by using the *visual variables* color and shape for encoding. Instances of notational elements, on the other hand, are distinguished by text, i.e., eMDM element names, which promotes *perceptual discriminability* and *visual expressiveness*. All textual representations, as part of the graphical notation, refer to grammar rules that are defined by corresponding syntax definitions in Appendix A, Listing A.2 and Listing A.3. *Semantic transparency*, i.e., the

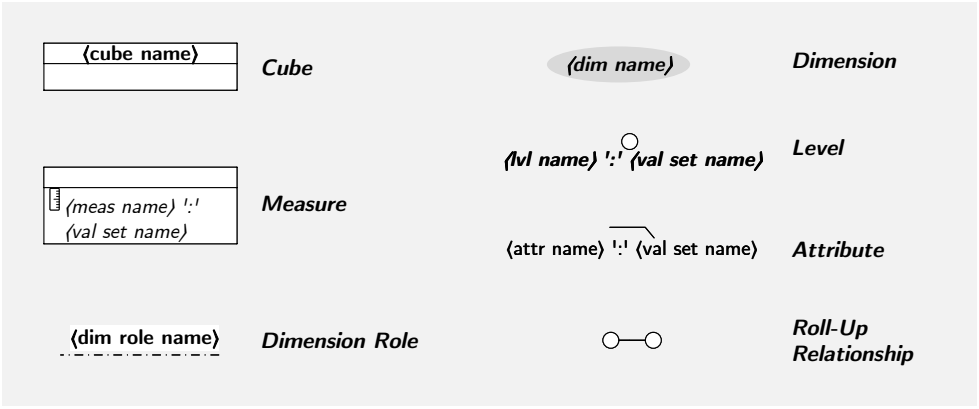


Figure 4.2: Multidimensional model notation

ability to infer the meaning based on the appearance of representations, of the eMDM notation comes from adopting the common dimensional fact model (DFM) notation as a basis for the multidimensional model notation [25]. Although the notation of defined business terms has to be learned by both novices and experts, experts familiar with concepts of the multidimensional model are likely to derive the multidimensional model notation's meaning from the representation without further explanations. Means for complexity management that take into account perceptual and cognitive limits are not considered here. Concerns of pattern and eMDM organization, however, are addressed in Chapter 7. We continue to follow the principle of graphic economy by focusing only on those aspects of eMDM elements that are relevant to the OLAP pattern approach. Finally, the proposed eMDM notation promotes cognitive integration by adding icons to business terms that correspond to the type of the expected target multidimensional model element.

The graphical notation for multidimensional model elements is as follows (see Figure 4.2). Boxes with solid lines represent cubes, the top compartment of which contains the cube's name, the second lists the measures. Dashed-and-dotted lines represent dimension roles, gray areas indicate membership of levels and attributes to dimensions, white circles represent levels, annotation lines of levels represent attributes, and edges between those levels represent roll-up relationships. Measures in cubes are additionally annotated with a ruler icon; the redundant visual notation promotes cognitive integration. The domains of measures, levels, and attributes follow a colon after the property name. A pointer from the cube to the base level of a dimension specifies the domain of a dimension role. Figure 4.4 then shows an example multidimensional model, corresponding to Example 4.1, that is enriched with the

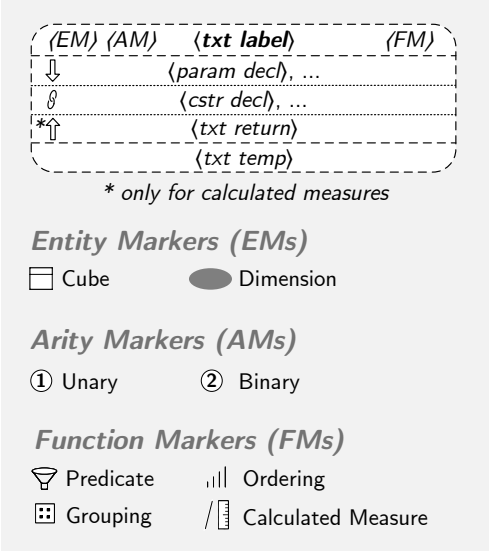


Figure 4.3: Business term notation

definition of business terms.

In the eMDM visualization, business terms are represented by dashed rectangles with round edges, divided into four or five compartments, depending on the type of business term (see Figure 4.3). The first compartment contains the name along with entity, arity, and function markers. Entity markers denote the entity type for which a business term is defined, i.e., either a cube or a dimension icon. Arity markers denote the number of context parameters to be provided. Function markers denote the query functionality provided, i.e., a funnel icon denotes predicates, an icon consisting of four framed black dots denotes groupings, an icon consisting of four vertical bars of increasing length denotes orderings, and a dotted measure icon preceded by a slash denotes calculated measures. The second compartment contains context parameters, visually marked by a downward-pointing arrow. The third compartment contains all constraints, visually marked by a chain icon. The fourth compartment is optional and contains the return type of calculated measures, visually marked by an upward-pointing arrow. The fifth compartment contains one or more expression templates, possibly annotated with target language and system (which are omitted in the examples). Both context parameters and constraints are represented only textually; a visual representation would lead to cluttered representation of business terms.

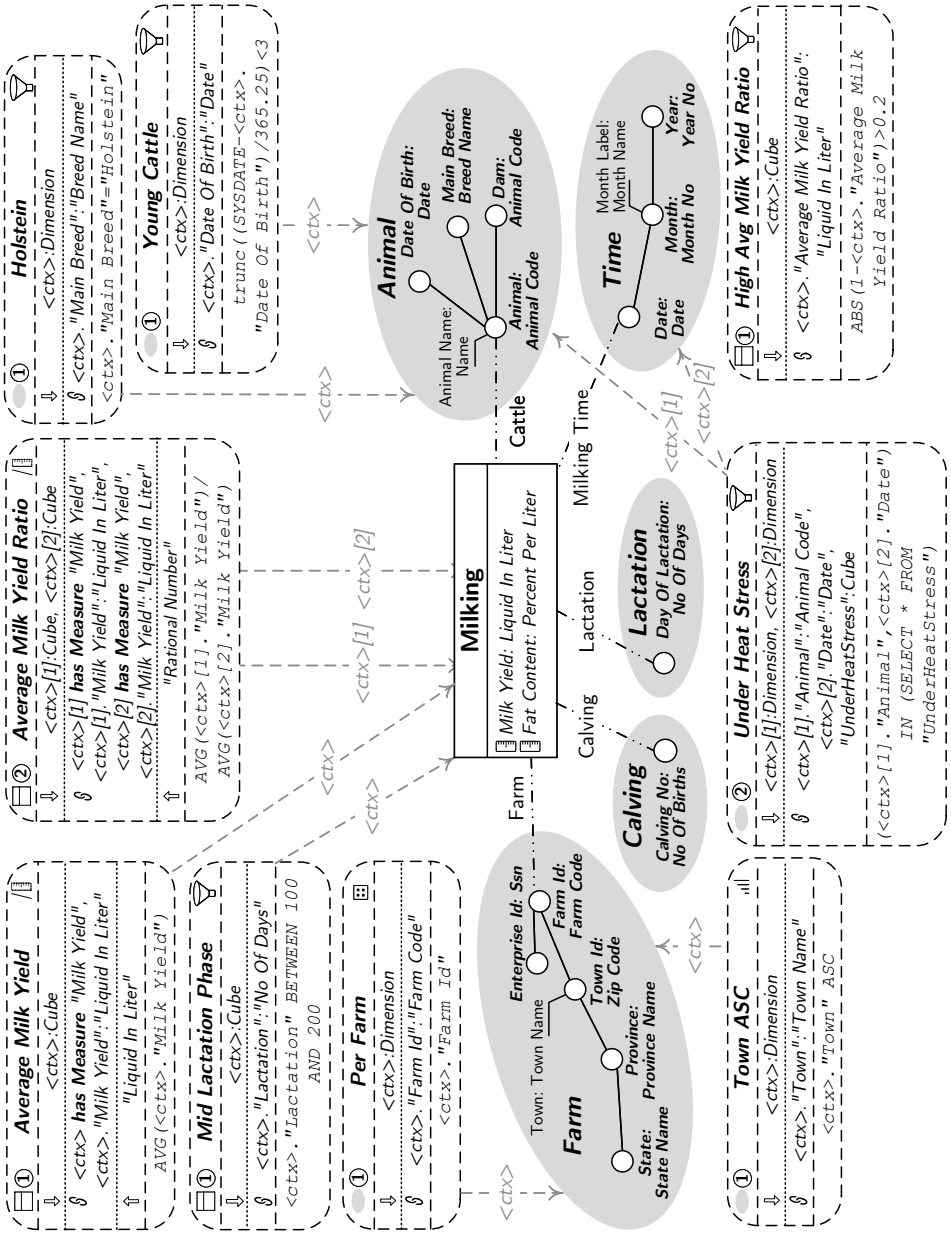


Figure 4.4: Happy Milk's enriched multidimensional model (potential valid business term applications are indicated by dashed grey arrows, representing the binding of the corresponding context parameter by the entity name pointed to)

Chapter 5

Pattern Definition

In this chapter, we formalize the notion of pattern and introduce a pattern definition language; we refer to Appendix A for a definition of the grammar. In Section 5.1 we focus on the structural aspects of patterns, i.e., pattern parameters, derived elements, and local cubes, the definition of which is a prerequisite to pattern usage with automatic query generation (see Figure 2.2 in Section 2.1), which we further describe in Chapter 6. We conclude this chapter by introducing a graphical notation in Section 5.2.

5.1 Formal Pattern Foundations

The vocabulary to define a pattern is provided by an associated eMDM, i.e., the names of model elements and the corresponding types, which are arranged in a meaningful way in order to satisfy a generic information need. Pattern definition involves declaration of pattern parameters and derived elements, derivation rules for each derived element, constraints, local cubes representing intermediary results, and templates. A pattern also comprises alias names, a description of the problem to be solved, an informal explanation of the solution, and an exemplary description of usage, aimed towards human comprehensibility and not referred to in the formalization.

Definition 10 (Pattern)**.** A *pattern* $p = (\mathbb{O}, PP, DE, \texttt{pvtype}, \texttt{derivation}, CT, CP, CD, CR, CA^1, CA^2, TPL, N_{loc}, \texttt{hasType}_{loc}, \texttt{hasProperty}_{loc}, \texttt{hasDomain}_{loc})$, associated with an enriched multidimensional model s, consists of

- the set of model element types $\mathbb{O}$ and the disjoint sets of pattern parameters PP and derived elements DE, which together constitute p's set of pattern variables $PV = PP \cup DE$,

- a total function pvtype : $PV \rightarrow \mathbb{O}$ that declares types of pattern variables, with $\texttt{type}|_{DE} : DE \rightarrow \{\mathsf{D}, \mathsf{U}, \mathsf{I}\}$,
- a set of templates TPL,
- derivation rules derivation (see Definition 11), constraint relations CT, CP, CD, CR, CA^1, and CA^2 (see Definition 13), local cubes N_{loc}, $\texttt{hasType}_{loc}$, $\texttt{hasProperty}_{loc}$, $\texttt{hasDomain}_{loc}$ (see Definition 12), and
- derived subsets of
  - pattern variables that represent entities $PV_E = \{v \in PV \mid \texttt{pvtype}(v) \in \mathbb{E}\}$, value sets $PV_V = \{v \in PV \mid \texttt{pvtype}(v) \in \mathbb{V}\}$, calculated measures $PV_{QM} = \{v \in PV \mid \texttt{pvtype}(v) \in \{\mathsf{QMU}, \mathsf{QMB}\}\}$, dimensions $PV_D = \{v \in PV \mid \texttt{pvtype}(v) = \mathsf{D}\}$, as well as
  - pattern parameters that represent business terms $PP_B = \{v \in PV \mid \texttt{pvtype}(v) \in \mathbb{B}\}$ and properties $PP_P = \{v \in PV \mid \texttt{pvtype}(v) \in \mathbb{P}\}$.

A pattern consists of multiple variables. Apart from the pattern parameters, the values of which the user specifies upon instantiation, a pattern's set of variables typically also comprises a number of derived elements. A derived element is a variable, the value of which ultimately derives from pattern parameters and other derived elements according to a derivation rule, which reduces the number of elements that must be specified by the user upon instantiation. Derived elements serve to structure the pattern and facilitate pattern usage, for without derived elements users would have to specify the missing information using parameters. For example, a unary dimension predicate can be applied to a derived element representing a dimension. The dimension to be restricted can be determined by evaluating the corresponding derivation rule. Thus, a user only states a cube's dimension role instead of specifying both the dimension role and the dimension. It should be noted that the dimension role as well as its referenced dimension are required for a restriction since one dimension can be referenced by multiple roles in one cube. Just as parameters, derived elements can occur at various positions in the language-specific templates.

Example 5.1 (Pattern Variables)**.** The breed-specific subset-subset comparison introduced in Chapter 2 has ten pattern parameters (Figure 2.5–Context, Lines 1-12). One of the pattern parameters is ⟨sourceCube⟩, which refers to a cube (Line 2). The pattern parameter ⟨compDimRole⟩ refers to a dimension role and the pattern parameter ⟨iDimSlice⟩ refers to a unary dimension predicate, i.e., $\exists$ ⟨sourceCube⟩, ⟨compDimRole⟩, ⟨iDimSlice⟩ $\in PP$: $\texttt{pvtype}(\langle\texttt{sourceCube}\rangle) = \mathsf{Q} \wedge \texttt{pvtype}(\langle\texttt{compDimRole}\rangle) = \mathsf{R} \wedge \texttt{pvtype}(\langle\texttt{iDimSlice}\rangle) = \mathsf{DPU}$. The pattern further has four derived elements, ⟨baseDim⟩, ⟨compDim⟩, ⟨joinDim⟩, and

⟨cubeMeasureDom⟩ (Lines 14-19), i.e., ∃ ⟨baseDim⟩, ⟨compDim⟩, ⟨joinDim⟩, ⟨cubeMeasureDom⟩ $\in DE$: pvtype(⟨baseDim⟩) = D ∧ pvtype(⟨compDim⟩) = D ∧ pvtype(⟨joinDim⟩) = D ∧ pvtype(⟨cubeMeasureDom⟩) = U. ♢

A pattern definition comprises templates for various target languages for the purposes of illustration but also for automatic query generation. The templates of a pattern are a realization of the solution, which is described informally in text. A pattern template is formulated for a target language and refers to each pattern parameters and to some derived elements at specific positions in code in the target language, either directly or as input for macro calls. A template represents the core OLAP query structure as an incomplete implementation formulated in a target language, taking into account a specific data model and implementation variant. The template is a combination of static content, represented by code in a target language, and dynamic content, i.e., pattern variables, derived elements, and macro calls, which are assigned specific values of an associated eMDM upon instantiation. Since the dynamic contents of the pattern are embedded in the template, the entire functionality of the target language is available to the pattern author, which allows for the composition of expressive queries.

For each derived element, a pattern must have a corresponding derivation rule. During pattern grounding in a specific eMDM (see Chapter 6), derivation rules serve to determine the names of value sets and dimensions that replace the derived elements in the pattern. A derived element is either derived via a property from an entity or via the return type of a calculated measure. Derivation rules involving an entity's property return the name of that property's domain, i.e., a number value set, a string value set, or a dimension name, while derivation rules involving calculated measures' return types return the names of number value sets.

Definition 11 (Pattern Derivation Rule)**.** A pattern p's total function derivation : $DE \rightarrow (((PV_E \cup N_E) \times (PP_P \cup N_P)) \cup (PP_B \cup N_B))$ defines derivation rules that either involve the domain of an entity's property or the return type of business terms.

Example 5.2 (Pattern Derivation Rule)**.** The name of the derived element ⟨compDim⟩ defined in the breed-specific subset-subset comparison pattern (see Figure 2.5—Context, Line 16), for example, can be obtained from an entity referenced by the pattern parameter ⟨sourceCube⟩ via the property referenced by the pattern parameter ⟨compDimRole⟩, i.e.,∃⟨compDim⟩∈ DE : ∃⟨sourceCube⟩,⟨compDimRole⟩∈ PP : derivation(⟨compDim⟩) = (⟨sourceCube⟩,⟨compDimRole⟩). The name of the ⟨compDimRole⟩ property's referenced domain can then be bound to ⟨compDim⟩. In addition, the derived element ⟨cubeMeasureDom⟩ is defined as the return type of the unary calculated measure ⟨cubeMeasure⟩ (Line 18), e.g.,

$\exists\langle\texttt{cubeMeasureDom}\rangle \in DE : \texttt{derivation}(\langle\texttt{cubeMeasureDom}\rangle) = \langle\texttt{cubeMeasure}\rangle$. The name of the value set returned by the $\langle\texttt{cubeMeasure}\rangle$ business term can then be bound to the derived element $\langle\texttt{cubeMeasureDom}\rangle$. ♢

We introduce local cubes to avoid the definition of complex constraints and to make the description of the solution more understandable, as they allow to refer to specific parts of a pattern's template (which is necessary to explain complex analytical queries). Complex analytical queries typically involve subqueries, the result of which can be thought of as interim local cubes that derive from some base cube and exist only in the context of the query. For example, the comparison of two aggregated measures for separate groups of facts requires the computation of an interest cube and a comparison cube with the measure that should be compared having been aggregated at the desired level of granularity prior to computing the comparative measure. In a pattern definition, the comparative measure could be simply constrained to the interest and comparison cubes, which must provide the necessary aggregation measures. Without these local cubes, the comparison measure would have to be constrained to both the base cube and the business terms that represent the necessary aggregation measures for that base cube.

Definition 12 (Local Cubes)**.** A pattern p's *local cubes* are defined by a set of names N_{loc} used to describe local cubes, i.e., $N_{loc} = N_{Q::loc} \cup N_{P::loc}$, with the subsets of cube names $N_{Q::loc}$, i.e., $N_{Q::loc} \cap N_Q = \emptyset$, and property names of local cubes $N_{P::loc}$, and the relations:

- $\texttt{hasType}_{loc} \subseteq N_{Q::loc} \times \{\mathsf{Q}\}$ representing the pattern's local cubes,
- $\texttt{hasProperty}_{loc} \subseteq N_{Q::loc} \times \{\mathsf{M}, \mathsf{R}\} \times (PP_P \cup N_{P::loc})$ representing the pattern's local cube properties,
- and $\texttt{hasDomain}_{loc} \subseteq N_{Q::loc} \times (PP_P \cup N_{P::loc}) \times (PV_V \cup PV_D)$ representing the pattern's local cube properties' domain.

A pattern includes definitions of the structure of local cubes as part of the context specification. For each local cube, the context specification defines a name by which it is referred to in the query, a number of measures, and the domain of each measure. It should be noted that a local cube's properties depend on the multidimensional model elements actually bound to pattern parameters and derived elements upon instantiation.

Example 5.3 (Local Cubes)**.** The template of the breed-specific subset-subset comparison represents the core OLAP query structure as described by the solution (Figure 2.6–Solution).

The local cubes baseCube (Figure 2.6–Template, Lines 1-8), interestCube (Lines 9-19), and comparisonCube (Lines 20-30) are the result of subqueries in the template. The baseCube is not represented as a local cube, as it is directly based on the ⟨sourceCube⟩ cube and it is not defined as an application target for business terms representing calculated measures (Figure 2.5–Context, Lines 39-45). Only the local cubes interestCube and comparisonCube are represented, as their calculated measures, which are described by the name of ⟨cubeMeasure⟩ and its the return type (⟨cubeMeasureDom⟩), are further used for calculations (Lines 21-28), e.g., ("$interestCube$", Q) ∈ $\texttt{hasType}_{loc}$, ("$interestCube$", M, ⟨cubeMeasure⟩) ∈ $\texttt{hasProperty}_{loc}$, and ("$interestCube$", ⟨cubeMeasure⟩, ⟨cubeMeasureDom⟩) ∈ $\texttt{hasDomain}_{loc}$. ♢

Pattern constraints represent conditions that have to be satisfied by elements of the associated eMDM or by the local cubes, referred to by name in the definition or instantiation of a pattern (see Chapter 6), in order for the pattern to be applicable in the context of a data warehouse system. Pattern constraints are similar to business term constraints (see Chapter 4 — Definition 6) but in addition to type, property, and domain constraints a pattern can also have return and applicable-to constraints. A *return* constraint requires the business terms bound to certain pattern parameters or constants to have a certain return type. An *applicable-to* constraint requires business terms bound to certain pattern parameters or constants to be applicable to particular entities; the application of the business term must be valid in the context of the associated eMDM and the pattern's local cubes (see Chapter 4 —Definition 8). Such applicable-to constraints can either be unary or binary, depending on the number of context parameters of the constrained business term.

Definition 13 (Pattern Constraints)**.** A pattern p's *pattern constraints* are represented by the following relations:

- $CT \subseteq (PV_E \cup PV_V \cup PP_B \cup N_E \cup N_{Q::loc} \cup N_V \cup N_B) \times \mathbb{O}$ that represents p's *type* constraints,
- $CP \subseteq (PV_E \cup N_E \cup N_{Q::loc}) \times \mathbb{P} \times (PP_P \cup N_P \cup N_{P::loc})$ that represents p's *property* constraints,
- $CD \subseteq (PV_E \cup N_E \cup N_{Q::loc}) \times (PP_P \cup N_P \cup N_{P::loc}) \times (PV_V \cup PV_D \cup N_V \cup N_D)$ that represents p's *domain* constraints,
- $CR \subseteq (PV_{QM} \cup N_{QMU} \cup N_{QMB}) \times (PV_U \cup N_U)$ that represents the pattern's *return constraints*,
- $CA^1 \subseteq (PP_B \cup N_B) \times (PV_E \cup N_E \cup N_{Q::loc})$ that represents p's *unary applicable-to* constraints,

- and $CA^2 \subseteq (PP_B \cup N_B) \times (PV_E \cup N_E \cup N_{Q::loc}) \times (PV_E \cup N_E \cup N_{Q::loc})$ that represents p's *binary applicable-to* constraints.

Example 5.4 (Pattern Constraints)**.** The breed-specific subset-subset comparison pattern has four property constraints defined (see Figure 2.5—Context visualized in Figure 5.2), two of which require that a dimension role named `Cattle` has to be part of the entity referred to by the parameter ⟨`sourceCube`⟩ (Line 31) and that a `Level` named `Main Breed` has to be part of an entity referred to by the ⟨`baseDim`⟩ derived element (Line 35), respectively, i.e., $(\langle \texttt{sourceCube} \rangle, \mathsf{R}, \text{“}Cattle\text{”}) \in CP$ and $(\langle \texttt{baseDim} \rangle, \mathsf{L}, \text{“}Main\ Breed\text{”}) \in CP$. Four domain constraints are defined, one of which requires that a property named `Main Breed` of an entity referred to by the derived element ⟨`baseDim`⟩ with a domain named `Breed Name` exists (Line 36), i.e., $(\langle \texttt{baseDim} \rangle, \text{“}Main\ Breed\text{”}, \text{“}Breed\ Name\text{”}) \in CD$. The pattern definition, furthermore, comprises six unary applicable-to constraints, which require that, for example, the business term bound to the parameter ⟨`baseDimSlice`⟩ during pattern instantiation is applicable to an entity referred to by the derived element ⟨`baseDim`⟩ (Line 39), i.e., $(\langle \texttt{baseDimSlice} \rangle, \langle \texttt{baseDim} \rangle) \in CA^1$, and one binary applicable-to constraint defines that ⟨`compMeasure`⟩ is required to be applicable to the local cubes `interestCube` and `comparisonCube` (Line 45), i.e., $(\langle \texttt{compMeasure} \rangle, \text{“}interestCube\text{”}, \text{“}comparisonCube\text{”}) \in CA^2$. ♢

Each derivation rule also implicitly represents a constraint. A derivation rule involving property of an entity implicitly defines a domain constraint, where the domain is represented by the corresponding derived element. A derivation rule involving the return type of a business term implicitly defines a return constraint, i.e., it restricts the expected return type of a calculated measure to the corresponding derived element.

Definition 14 (Derived Pattern Constraints)**.** For a pattern p, pattern *constraints can be derived* as follows:

- a domain constraint is derived for each derived element with a derivation rule involving the domain of a property, i.e., $\forall v \in DE : \exists y \in (PV_E \cup N_E)\ \exists y' \in (PP_P \cup N_P) : \texttt{derivation}(v) = (y, y') \Rightarrow (y, y', v) \in CD$,
- a return constraint is derived for each derived element with a derivation rule involving the return type of a business term, i.e., $\forall v \in DE : \exists y \in (PV_{QM} \cup N_{QMU} \cup N_{QMB}) : \texttt{derivation}(v) = y \Rightarrow (y, v) \in CR$.

Example 5.5 (Derived Pattern Constraints)**.** In the breed-specific subset-subset comparison (Figure 2.5—Context) three domain constraints can be derived for the derived elements ⟨`baseDim`⟩, ⟨`compDim`⟩, and ⟨`joinDim`⟩ that reference corresponding dimensions

(Lines 15-17), i.e., $\{(\langle\texttt{sourceCube}\rangle, \textit{"Cattle"}, \langle\texttt{baseDim}\rangle), (\langle\texttt{sourceCube}\rangle, \langle\texttt{compDimRole}\rangle, \langle\texttt{compDim}\rangle), (\langle\texttt{sourceCube}\rangle, \langle\texttt{joinDimRole}\rangle, \langle\texttt{joinDim}\rangle)\} \in CD$, and one return constraint can be derived for the derived element $\langle\texttt{cubeMeasureDom}\rangle$ (Line 18), i.e., $(\langle\texttt{cubeMeasure}\rangle, \langle\texttt{cubeMeasureDom}\rangle)\} \in CR$. ♢

Finally, a pattern must be *well-formed* in order to be used, i.e., there must not exist circular definitions of derivation rules.

Definition 15 (Well-Formed Pattern)**.** A pattern p is *well-formed* if, and only if, the derivation rules defined for properties are acyclic. Let $DR \subseteq DE \times (PV_E \cup N_E)$ denote the direct derivation relationships, then $\forall y \in DE\ \forall y' \in (PV_E \cup N_E) : (\exists y'' \in (PP_P \cup N_P) : \texttt{derivation}(y) = (y', y'')) \Leftrightarrow DR(y, y')$. Let DR^+ denote the transitive closure of DR, then $\forall y \in (PV_E \cup N_E) : (y, y) \notin DR^+$.

Example 5.6 (Well-Formed Pattern)**.** The breed-specific subset-subset comparison is well-formed since the derivation rules involving the domains of properties (Lines 16-17) are acyclic. ♢

5.2 Graphical Pattern Notation

Pattern documentation consists of the pattern's name and alias names, a textual description of the problem to be solved, a formal definition of the context as represented by parameters and derived elements as well as both constraints and local cubes, a textual description of the solution, possibly with an informal illustration, a number of templates in various target languages, and a list of related patterns (see Figure 5.1 for the notation, Figure 2.5, and Figure 2.6). Due to space considerations, the example usage is omitted in the documentation of breed-specific subset-subset comparison in Figure 2.5 and Figure 2.6; usage of this pattern is illustrated in the running example of this thesis.

Pattern context can be visualized by representing the parameters, derived elements, constraints, local cubes, and constants as following the context notation for patterns (see Figure 5.3 and Figure 5.2 for an example). Parameters and derived elements are represented by their names inside angle brackets – not to be confused with the names bound to the parameters and derived elements during instantiation – while name constants are included as is. Derived elements are prefixed with a slash; the corresponding derivation rules are represented by domain and return constraints. The notation of multidimensional models and business terms is reused in a simplified fashion – depicting only the business term's name with entity, arity, and function markers and if necessary the return type representing return constraints – to visualize type constraints. Domain and property constraints

Pattern Definition:

⟨p name⟩
Also Known As (⟨p name⟩ (',' ⟨p name⟩)*)?
Problem ⟨*prob txt*⟩
Solution ⟨*sol txt*⟩
Context (⟨*param decl*⟩ | ⟨*derv decl*⟩ | ⟨*cstr decl*⟩ | ⟨*lc decl*⟩)*
Template ⟨*temp elem*⟩
Example ⟨*ex txt*⟩
Related Pattern (⟨p name⟩ (',' ⟨p name⟩)*)?

Figure 5.1: Pattern notation

are represented by visualizing the entities and properties eMDM notation. Applicable-to constraints are visualized by relating a parameter or name constant denoting a business term to a parameter, derived element, or name constant denoting an entity using a dashed arrow for each business term parameter. In order to distinguish the context parameters for business terms, the context parameters' names are added to the dashed arrows. Finally, local cubes are visualized by reusing the notation for type, property, and domain constraints except that dotted lines are used instead of solid lines.

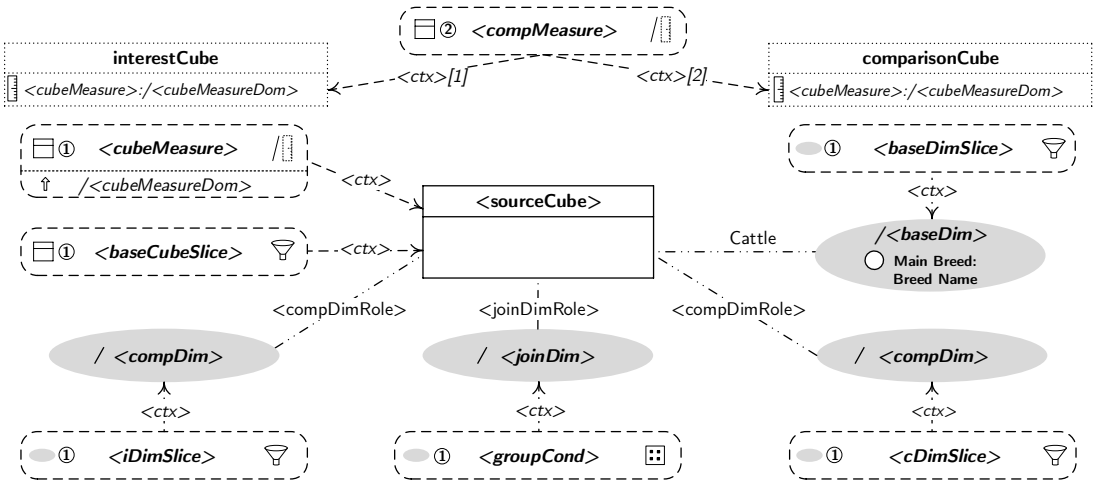


Figure 5.2: Illustration of the *Context* section of breed-specific subset-subset comparison from Figure 2.5

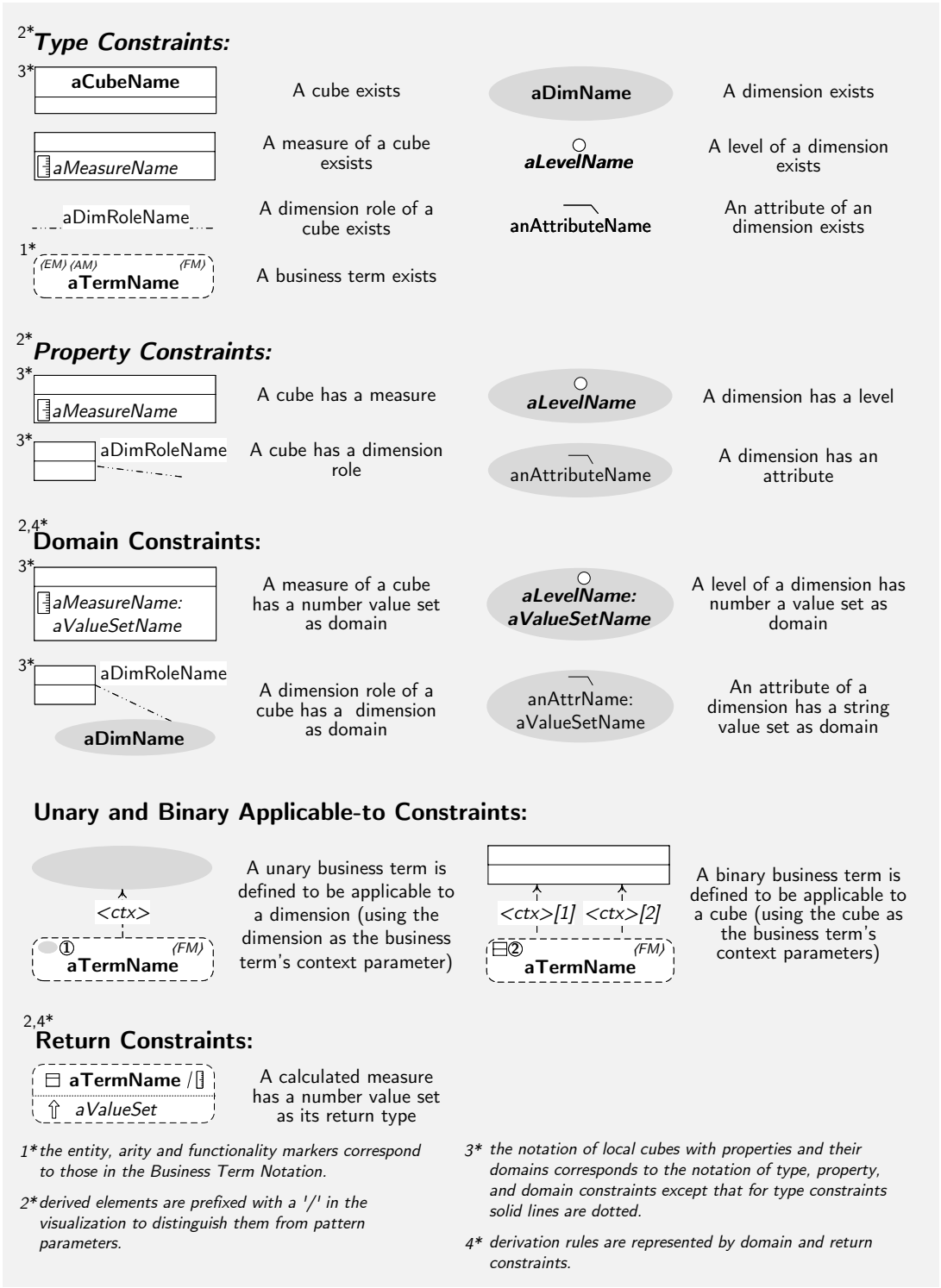


Figure 5.3: Notation for the context of patterns

Chapter 6

Pattern Usage

In this chapter we formalize the steps necessary to use patterns in the context of an associated enriched multidimensional model. For this purpose, in Section 6.1 we provide a formal definition of pattern operations, i.e., instantiation and grounding of patterns, and pattern states, i.e., parameter-free, grounded, and applicable pattern. Finally, Section 6.2 describes how to execute an applicable patterns to obtain the executable OLAP query.

6.1 Pattern Instantiation and Grounding

The proposed pattern-based approach supports automatic query generation when using a pattern associated to an eMDM in the context of a specific data warehouse that follows that associated eMDM (see Figure 2.3). To this end, the user instantiates a pattern, first, by binding names of model elements available in the associated eMDM to the pattern parameters resulting in a *parameter-free* pattern. Second, the parameter-free pattern is then grounded in the context of the associated eMDM, which leads to the resolution of derived elements via the derivation rules, resulting in a *ground* pattern. Finally, if the ground pattern is *applicable* to the associated eMDM – while also considering the pattern's local cubes –, i.e., all the pattern constraints can be satisfied, the ground pattern can be executed, i.e., the template can be transformed into an executable query for a target system. In this chapter, we formalize the notion of pattern instantiation and pattern grounding while also discussing the notion of pattern execution. Although part of pattern usage, we do not describe the actual processing of the generated query, which is performed by the data warehouse system and not specific to the proposed pattern-based approach.

```
DERIVED ELEMENTS
  <baseDim>:DIMENSION <= "Milking"."Cattle";
  <compDim>:DIMENSION <= "Milking"."Cattle";
  <joinDim>:DIMENSION <= "Milking"."Farm";
  <cubeMeasureDom>:NUMBER_VALUE_SET <= "Average Milk Yield".RETURNS;
END DERIVED ELEMENTS;

LOCAL CUBES
  "interestCube":CUBE;
  "interestCube" HAS MEASURE "Average Milk Yield";
  "interestCube"."Average Milk Yield":<cubeMeasureDom>;
  "comparisonCube":CUBE;
  "comparisonCube" HAS MEASURE "Average Milk Yield";
  "comparisonCube"."Average Milk Yield":<cubeMeasureDom>;
END LOCAL CUBES;

CONSTRAINTS
  "Milking":CUBE;
  "Low Daily Milk Yield":UNARY_CUBE_PREDICATE;
  "Holstein":UNARY_DIMENSION_PREDICATE;
  "Young Cattle":UNARY_DIMENSION_PREDICATE;
  "Old Cattle":UNARY_DIMENSION_PREDICATE;
  "Per Farm":DIMENSION_GROUPING;
  "Average Milk Yield":UNARY_CALCULATED_MEASURE;
  "Average Milk Yield Ratio":BINARY_CALCULATED_MEASURE;

  "Milking" HAS DIMENSION_ROLE "Cattle";
  "Milking" HAS DIMENSION_ROLE "Farm";
  "Milking"."Cattle":<baseDim>;
  <baseDim> HAS LEVEL "Main Breed";
  <baseDim>."Main Breed":"Breed Name";
  "Milking"."Cattle":<compDim>;
  "Milking"."Farm":<joinDim>;
  "Average Milk Yield" RETURNS <cubeMeasureDom>;

  "Holstein" IS_APPLICABLE_TO <baseDim>;
  "Low Daily Milk Yield" IS_APPLICABLE_TO "Milking";
  "Young Cattle" IS_APPLICABLE_TO <compDim>;
  "Old Cattle" IS_APPLICABLE_TO <compDim>;
  "Per Farm" IS_APPLICABLE_TO <joinDim>;
  "Average Milk Yield" IS_APPLICABLE_TO "Milking";
  "Average Milk Yield Ratio" IS_APPLICABLE_TO
      ("interestCube","comparisonCube");
END CONSTRAINTS;
```

Listing 6.1: Derivation rules, local cubes, and constraints of *dairy*- and breed-specific subset-subset comparison as a result of instantiation in Listing 2.1

We use the notion of *pattern variable substitution* for the definition of *pattern instantiation*. Pattern variables are substituted in the constraints, derivation rules, templates, and local cubes with names of elements according to a substitution function, representing the binding of names to variables. The result of pattern instantiation is a new pattern with a reduced set of pattern parameters where all substituted parameters from the original pattern are not included. As pattern variables are being removed during pattern variable substitution, the variable's declared type should nevertheless not be lost, i.e., for entity, value set, and business term pattern variables, corresponding type constraints, and for property pattern variables that are used in domain constraints, corresponding property constraints are derived.

Definition 16 (Pattern Variable Substitution)**.** The application $p\sigma$ of a partial *substitution* function $\sigma : PV \rightarrow N$ to a pattern $p = (\mathbb{O}, PP, DE, \texttt{pvtype}, \texttt{derivation}, CT, CP, CD, CR, CA^1, CA^2, TPL, N_{loc}, \texttt{hasType}_{loc}, \texttt{hasProperty}_{loc}, \texttt{hasDomain}_{loc})$ returns a new pattern $p' = (\mathbb{O}, PP', DE', \texttt{pvtype}', \texttt{derivation}', CT', CP', CD', CR', CA^{1\prime}, CA^{2\prime}, TPL', N_{loc}, \texttt{hasType}_{loc}, \texttt{hasProperty}'_{loc}, \texttt{hasDomain}'_{loc})$ where

- pattern variables with a name bound are removed, i.e., $PP' = PP \setminus \{v \in PP \mid \sigma(v)\ is\ defined\}$ and $DE' = DE \setminus \{v \in DE \mid \sigma(v)\ is\ defined\}$,
- type declarations for PV' are taken over, i.e., $\forall v \in PV' : \texttt{pvtype}'(v) = \texttt{pvtype}(v)$,
- derivation rules for DE' are taken over, i.e., $\forall v \in DE' : \texttt{derivation}'(v) = \texttt{derivation}(v)$ with variables in the codomain of `derivation` substituted according to σ,
- type constraints for substituted pattern variables representing entities, business terms, and value sets are derived, i.e., $CT' = \{(\sigma(v), \texttt{pvtype}(v)) | v \in (PV_E \cup PV_B \cup PV_V) \wedge \sigma(v)\ is\ defined\} \cup CT$,
- property constraints for substituted pattern variables representing properties are derived, i.e., $CP' = \{(y, \texttt{pvtype}(v), \sigma(v)) | \exists y \in (PV_E \cup N_E \cup N_{Q::loc})\ \exists y' \in (PV_D \cup PV_V \cup N_D \cup N_V) : v \in PP_P \wedge \sigma(v)\ is\ defined\ \wedge (y, v, y') \in CD\} \cup CP$ with variables in CP' substituted according to σ,
- and CD', CR', $CA^{1\prime}$, $CA^{2\prime}$, TPL', $\texttt{hasProperty}'_{loc}$, $\texttt{hasDomain}'_{loc}$ correspond to CD, CR, CA^1, CA^2, TPL, $\texttt{hasProperty}_{loc}$, $\texttt{hasDomain}_{loc}$ respectively, with variables substituted according to σ.

Definition 17 (Pattern Instantiation)**.** An *instantiation* δ^p_σ of a well-formed pattern p given a partial substitution function $\sigma : PP \rightarrow N$, with PP the set of p's pattern parameters, corresponds to $p\sigma$, the application of σ to p.

Example 6.1 (Pattern Instantiation)**.** Consider the definition of breed-specific subset-subset comparison pattern (Figure 2.5 and Figure 2.6). Listing 2.1 shows an instantiation of breed-specific subset-subset comparison with bindings for each of the pattern's parameters. For example, the name `Holstein`, denoting a business term in the dairy domain, is bound to the ⟨`baseDimSlice`⟩ parameter (Listing 2.1, Line 5). Parameters from the instantiated pattern for which a binding is specified are not part of the resulting pattern's set of parameters. For example, since the name `Holstein` is bound to the ⟨`baseDimSlice`⟩ parameter during instantiation, ⟨`baseDimSlice`⟩ is not a parameter of the resulting pattern; however, a type constraint is derived stating that a `Holstein` business term is expected to be available. Listing 6.1 shows the context specification of a pattern that is equivalent to the result of the pattern instantiation in Listing 2.1; it should be noted that the lack of parameters in that pattern. Compared to the original pattern definition (Figure 2.5—Context), the definition in Listing 6.1, while corresponding to the original pattern, has all occurrences of the parameters from the original pattern substituted with name constants in the derivation rules (Listing 6.1, Lines 1-6), local cubes (Lines 8-15), and constraints (Lines 17-43). Likewise, the template corresponds to the original pattern's template (Figure 2.5—Template) with occurrences of parameters substituted with name constants. It should be noted that for each bound pattern parameter representing an entity or a business term a corresponding type constraint is derived (Lines 18-25), while for bound pattern parameter representing a property a corresponding property constraint is derived (Lines 27-28). To avoid redundancy, derived constraints that convey the same information are depicted only once. ♢

Pattern instantiation does not necessarily mean substitution of *all* of a pattern's parameters. Partial instantiation can be a means for stepwise refinement from more generic to more specific patterns, gradually reducing a pattern's level of abstraction. Thus, partial instantiation serves for the organization of patterns into comprehensive pattern catalogs (see Chapter 7). Only a *parameter-free* pattern, however, can be grounded and executed in the context of a particular data warehouse; a parameter-free pattern cannot be further instantiated.

Definition 18 (Parameter-Free Pattern)**.** A pattern p is *parameter-free* if, and only if, it is well-formed and the set of pattern parameters is empty, i.e., $PP = \emptyset$.

Example 6.2 (Parameter-Free Pattern)**.** The dairy- and breed-specific subset-subset comparison pattern (Listing 6.1) corresponds to the result of a complete instantiation of the breed-specific subset-subset comparison pattern (see Listing 2.1) and is parameter-free since it has no parameters. ♢

```
LOCAL CUBES
  "interestCube":CUBE;
  "interestCube" HAS MEASURE "Average Milk Yield";
  "interestCube"."Average Milk Yield":"Liquid In Liter";
  "comparisonCube":CUBE;
  "comparisonCube" HAS MEASURE "Average Milk Yield";
  "comparisonCube"."Average Milk Yield":"Liquid In Liter";
END LOCAL CUBES;

CONSTRAINTS
  "Milking":CUBE;
  "Low Daily Milk Yield":UNARY_CUBE_PREDICATE;
  "Holstein":UNARY_DIMENSION_PREDICATE;
  "Young Cattle":UNARY_DIMENSION_PREDICATE;
  "Old Cattle":UNARY_DIMENSION_PREDICATE;
  "Per Farm":DIMENSION_GROUPING;
  "Average Milk Yield":UNARY_CALCULATED_MEASURE;
  "Average Milk Yield Ratio":BINARY_CALCULATED_MEASURE;
  "Animal":DIMENSION;
  "Farm":DIMENSION;
  "Liquid In Liter":NUMBER_VALUE_SET;

  "Milking" HAS DIMENSION_ROLE "Cattle";
  "Milking" HAS DIMENSION_ROLE "Farm";
  "Milking"."Cattle":"Animal";
  "Animal" HAS LEVEL "Main Breed";
  "Animal"."Main Breed":"Breed Name";
  "Milking"."Cattle":"Animal";
  "Milking"."Farm":"Farm";
  "Average Milk Yield" RETURNS "Liquid In Liter";

  "Holstein" IS_APPLICABLE_TO "Animal";
  "Low Daily Milk Yield" IS_APPLICABLE_TO "Milking";
  "Young Cattle" IS_APPLICABLE_TO "Animal";
  "Old Cattle" IS_APPLICABLE_TO "Animal";
  "Per Farm" IS_APPLICABLE_TO "Farm";
  "Average Milk Yield" IS_APPLICABLE_TO "Milking";
  "Average Milk Yield Ratio" IS_APPLICABLE_TO
      ("interestCube","comparisonCube");
END CONSTRAINTS;
```

Listing 6.2: Derivation rules and constraints of grounded dairy- and breed-specific subset-subset comparison as a result of grounding in Listing 6.4

```
WITH baseCube AS (
  SELECT *
  FROM   "Milking" sc
         JOIN "Animal" a ON
            sc."Cattle" = a.$dimKey("Animal")
  WHERE  $expr("Mid Lactation Phase", sc) AND
         $expr("Holstein", a)
),
interestCube AS (
  SELECT $expr("Per Farm", jd),
         $expr("Average Milk Yield", bc) AS "Average Milk Yield"
  FROM   baseCube bc
         JOIN "Farm" jd ON
            bc."Farm" = jd.$dimKey("Farm")
         JOIN "Animal" cd ON
            bc."Cattle" = cd.$dimKey("Animal")
  WHERE  $expr("Young Cattle", cd)
  GROUP BY $expr("Per Farm", jd)
),
comparisonCube AS (
  SELECT $expr("Per Farm", jd),
         $expr("Average Milk Yield", bc) AS "Average Milk Yield"
  FROM   baseCube bc
         JOIN "Farm" jd ON
            bc."Farm" = jd.$dimKey("Farm")
         JOIN "Animal" cd ON
            bc."Cattle" = cd.$dimKey("Animal")
  WHERE  $expr("Old Cattle", cd)
  GROUP BY $expr("Per Farm", jd)
)
SELECT $expr("Per Farm", ic),
       ic."Average Milk Yield" AS "Group of Interest",
       cc."Average Milk Yield" AS "Group of Comparison"
       $expr("Average Milk Yield Ratio", ic, cc) AS "Average Milk
           Yield Ratio"
FROM   interestCube ic
       JOIN comparisonCube cc ON
          $expr("Per Farm", ic) = $expr("Per Farm", cc)
```

Listing 6.3: Grounded template for grounded dairy- and breed-specific subset-subset comparison

A parameter-free pattern is not necessarily free of variables: Values for derived elements must be obtained through *pattern grounding* with respect to the associated eMDM. Pattern grounding corresponds to the application of a substitution to a parameter-free pattern where the derivation rules determine the substitution of variables with names of elements from the associated eMDM while also considering the pattern's local cubes. All derived-element variables in derivation rules, constraints, templates, and local cubes for which a binding could be determined are replaced with the determined name.

Definition 19 (Pattern Grounding)**.** The *grounding* of a parameter-free pattern p in an enriched multidimensional model s is given by Δ_s^p, the application $p\sigma$ of a substitution $\sigma : DE \to N$ to p, where DE is the set of p's derived elements and σ is derived according to p's *definite* derivation rules. A derivation rule of a derived element is *definite* if the first compartment of the codomain is a name or a derived element $v' \in DE$ where $\sigma(v')\ is\ defined$. The substitution function σ is derived from s according to p's acyclic derivation rules by repeating the following steps as long as a derived element $v \in DE$ exists with $\texttt{derivation}(v)\ is\ definite$ and $\sigma(v)\ is\ undefined$:

1. evaluate derivation rules involving the return type of a named business term, i.e., $\forall v \in DE : \exists n \in N_B\ \exists n' \in N_U : \texttt{derivation}(v) = n \wedge (n, n') \in \texttt{hasReturn} \Rightarrow \sigma(v) = n'$
2. evaluate derivation rules involving the domain of a property of a named entity, i.e., $\forall v \in DE : \exists n \in (N_E \cup N_{Q::loc})\ \exists n' \in (N_P \cup N_{P::loc})\ \exists n'' \in (N_D \cup N_V) : \texttt{derivation}(v) = (n, n') \wedge ((n, n', n'') \in \texttt{hasDomain}_{loc} \vee (n, n', n'') \in \texttt{hasDomain}) \Rightarrow \sigma(v) = n''$
3. evaluate derivation rules involving the domain of a property of an entity referred to by a derived element where σ is defined, i.e., $\forall v \in DE : \exists v' \in DE\ \exists n \in (N_P \cup N_{P::loc})\ \exists n' \in (N_D \cup N_V) : \texttt{derivation}(v) = (v', n) \wedge \sigma(v')\ is\ defined \wedge ((\sigma(v'), n, n') \in \texttt{hasDomain}_{loc} \vee (\sigma(v'), n, n') \in \texttt{hasDomain}) \Rightarrow \sigma(v) = n'$

```
1 GROUND PATTERN "Dairy- and Breed-Specific Subset-Subset Comparison"
2    AS "Grounded Dairy- and Breed-Specific Subset-Subset Comparison"
3    FOR "Happy Milk eMDM";
```

Listing 6.4: *Grounding* of the *dairy-* and breed-specific subset-subset comparison

Example 6.3 (Pattern Grounding)**.** In Listing 6.4, the dairy- and breed-specific subset-subset comparison is grounded to Happy Milk's eMDM (Figure 4.4). After instantiation, the

derivation rule for ⟨compDim⟩, for example, defines that the name to be bound can be derived from the domain of a property named Cattle that is owned by an entity named Milking (Listing 6.1, Line 3). The evaluation of this derivation rule with respect to Happy Milks' eMDM (Listing 4.1, Line 8) returns the name Animal, which substitutes for all occurrences of the derived element ⟨compDim⟩ in constraints, derivation rules, and templates (Listing 6.2 and Listing 6.3). It should be noted that a corresponding type constraint is derived for ⟨compDim⟩ and its bound name Animal (Listing 6.2, Line 19). ♢

Ideally, the result of pattern grounding is a *ground* pattern, i.e., a pattern without parameters and derived elements. If all derivation rules can be evaluated properly in the context of the associated eMDM and the pattern's local cubes, the result of pattern grounding is a ground pattern. Otherwise, the associated eMDM or the pattern's local cubes do not contain all of the required model elements for using the pattern. The proper evaluation of the derivation rules is part of the contract specified in the pattern's context.

Definition 20 (Ground Pattern)**.** A pattern p is *ground* if, and only if, p is parameter-free and the set of derived elements is empty, i.e., $DE = \emptyset$.

Example 6.4 (Ground Pattern)**.** The grounding of dairy- and breed-specific subset-subset comparison pattern to the associated eMDM for HappyMilk yields a ground pattern which is free of derived elements since all derivation rules could be evaluated (Listing 6.2). ♢

In order for a ground pattern to be *applicable*, either the associated eMDM or the pattern's local cubes, i.e., template(s), must provide elements that comply to the pattern's grounded type, property, domain, and return constraints. Furthermore, each applicable-to constraint – specifying names of argument entities for a business term – mandates a check whether application of each business term using the indicated entity names as arguments yields a valid business term application.

Definition 21 (Applicable Pattern)**.** A ground pattern p associated to an enriched multidimensional model s is *applicable* in a data warehouse conforming to s if, and only if, the constraints in p are satisfied over s or p's local cubes, i.e., $CT \subseteq (\texttt{hasType} \cup \texttt{hasType}_{loc})$, $CP \subseteq (\texttt{hasProperty} \cup \texttt{hasProperty}_{loc})$, $CD \subseteq (\texttt{hasDomain} \cup \texttt{hasDomain}_{loc})$, $CR \subseteq$ hasReturn, and each applicable-to constraint must be satisfied over s:

1. For each name referring to a business term in an applicable-to constraint there must be a corresponding business term in s with an appropriate type, i.e., $\forall n \in N_B\ \forall n' \in N_E : (n, n') \in CA^1 \Rightarrow \exists \mathsf{T} \in \{\mathsf{QMU}, \mathsf{QPU}, \mathsf{QO}, \mathsf{DG}, \mathsf{DPU}, \mathsf{DO}\} : (n, \mathsf{T}) \in \texttt{hasType}$ and $\forall n \in N_B\ \forall n', n'' \in N_E : (n, n', n'') \in CA^2 \Rightarrow \exists \mathsf{T} \in \{\mathsf{QMB}, \mathsf{QPB}, \mathsf{DPB}\}) : (n, \mathsf{T}) \in \texttt{hasType}$.

2. For each business term b in s, let n denote its name in s, i.e., $(n, b) \in \texttt{hasTerm}$,

   - if $(n, n') \in CA^1$ for any $n' \in (N_E \cup N_{Q::loc})$ then the application δ^b_σ with $\sigma(\langle \texttt{ctx} \rangle) = n'$, or
   - if $(n, n', n'') \in CA^2$ for any $n', n'' \in (N_E \cup N_{Q::loc})$ then the application δ^b_σ with $\sigma(\langle \texttt{ctx} \rangle[1]) = n'$ and $\sigma(\langle \texttt{ctx} \rangle[2]) = n'$

   must be a valid business term application over s extended by p's local cubes.

Example 6.5 (Applicable Pattern)**.** The grounded dairy- and breed-specific subset-subset comparison pattern is applicable to its associated eMDM since elements that have names as specified in the pattern and comply with the constraints are available in the eMDM or in the pattern's set of local cubes, respectively. ♢

6.2 Pattern Execution

An applicable pattern can be automatically translated into an OLAP query for execution in a target system, the logical data model of which is represented by the associated eMDM. To this end, macro calls in the pattern's template for a target system must be processed first. The macro calls are parameterized and represent necessary preprocessing functionalities (see for details Table 2.1). Processing of the macro calls then returns code snippets that replace the corresponding macro calls in the template.

The arguments bound to the parameters of the macros are simply considered as strings that reference corresponding eMDM elements either directly by specifying the corresponding name or implicitly by aliasing. It should be noted that the alias is only available within a specific template. The preprocessing of the macro calls does not perform any check of the arguments passed, as it is assumed that only applicable patterns are executed, i.e., the eMDM elements referenced by the passed strings fulfil all necessary constraints.

```
1 EXECUTE PATTERN "Grounded Dairy- and Breed-Specific Subset-Subset
      Comparison"
2    FOR "Happy Milk eMDM"
3    WITH TEMPLATE
4      DATA_MODEL = "Relational",
5      VARIANT    = "Star Schema",
6      LANGUAGE   = "SQL",
7      DIALECT    = "ORACLEv11";
```

Listing 6.5: *Execution* of the *grounded* and *applicable* dairy- and breed-specific subset-subset comparison

Example 6.6 (Execute Pattern)**.** The statement in Listing 6.5 executes the grounded and applicable dairy- and breed-specific subset-subset comparison pattern to obtain an executable OLAP query (Listing 2.2) that is based on the pattern's grounded template (Listing 6.3), i.e., that is executable for a relational system realized as star schema using SQL with the Oraclev11 dialect. To this end, the macro call `$expr("Average Milk Yield", bc)` (Listing 6.3, Line 11), for example, is being processed by substituting the parameter ⟨`ctx`⟩ (Listing 4.3, Line 1) by `bc` in the template expression (Listing 4.4, Line 10) in order to obtain the grounded template expression (Listing 2.2, Line 11) which takes the place of the macro call `$expr("Average Milk Yield", bc)` in the template. It should be noted that `bc` is defined as an alias for the `baseCube` common table expression (Listing 6.3, Line 12), which in turn references the reduced `Milking` source cube that fulfils the specified constraints. In addition, the macro call `$dimKey("Farm")` (Listing 6.3, Line 14), for example, is processed to return `Farm Id` which is used to replace the macro call `$dimKey("Farm")`. ♢

Automatic code generation using the presented macros requires the logical model to adhere to certain conventions. In particular, the logical model must be a star schema with denormalized dimension tables where the table names correspond to the entity names in an eMDM and the column names correspond to property names. Mazón and Trujillo that developed an model-driven architecture for data warehouses also assuming an star schema realization [84]. We stress, though, that the pattern approach would also work with other forms of organization. The naming convention between conceptual and logical representation of model elements can, for example, be omitted by capturing additional mapping information. One *mapping table* per data warehouse system can be introduced with an entry for each model element name and the corresponding name used in the logical representation. The model element name used for entities consists of their unique name whereas for properties the name prefixed by the name of the owning entity is used. The templates and expression snippets have to be extended by the `$map` macro to consider this additional mapping information. The `$map` macro translates names that refer to model elements to the name of the corresponding logical representation by utilizing the available mapping table. Furthermore, snowflake schemas can be supported by capturing which normalized dimension tables exist and how they can be joined. This would allow for denormalizing dimensions with a corresponding macro. However, in this thesis we followed the *convention over configuration* paradigm.

Chapter 7

Pattern Organization

In this chapter we describe how OLAP patterns can be defined at different levels of abstraction. In particular, we distinguish – from more abstract to more specific – domain-independent (Section 7.2), domain-specific (Section 7.3), and organization-specific (Section 7.4) patterns. These levels of abstraction may inform the collection of patterns into comprehensive *pattern catalogs* (Section 7.5). Likewise, the business terms of an eMDM may be collected into *business term vocabularies*, while cubes and dimensions of an eMDM may be collected into *multidimensional models* at different levels of abstraction.

7.1 Levels of Abstraction

An abstraction level groups patterns that share the same business vocabulary. Patterns of a particular abstraction level either refer to the same generic business vocabulary (domain-independent), the same business vocabulary for a particular domain (domain-specific), or the same business vocabulary that can only be understood within a particular organization (organization-specific). The business vocabularies that are referenced by the OLAP patterns are eMDMs, which consist of different types of elements, i.e., business terms, cubes, and dimensions. Thus, just like the patterns themselves, the referenced eMDM elements are grouped into abstraction levels.

The organization of patterns along abstraction levels – from domain-independent to domain-specific and organization-specific – can promote the *discoverability* of OLAP patterns for target audiences and application areas, thereby facilitating the reuse of OLAP patterns across domains. This organization of OLAP patterns is inspired by the work of Silverston and Agnew, who define data model patterns at different levels of abstraction, from industry-specific (domain-specific) data model patterns [41] to patterns for all enterprises (domain-independent) [42], [46]. In contrast to Silverston and Agnew [42, p. 7-14], we use only

three levels of abstraction instead of four since we believe that the distinction between, in particular, Pattern Level 1 and Pattern Level 2 is too subjective. Nevertheless, we share the same motivation as Silverston and Agnew, i.e., the more specific a pattern is, the more static it becomes. To assign a specific pattern to a corresponding abstraction level, the following principle is applied. If a pattern contains references to business terms or entities from an eMDM, then these elements must also be present at the same level of abstraction as the pattern. However, if the referenced eMDM elements are associated with a more specific level than the pattern then the pattern must be moved to the same level in the catalog as those referenced eMDM elements. Thus, the employed business vocabulary, i.e., the referenced eMDM elements, determines the level of abstraction of a pattern. Furthermore, the business vocabulary used in the local cubes must also be taken into account when assigning a pattern to an abstraction level.

Example 7.1 (Levels of Abstraction)**.** Figure 7.1 illustrates the organization of domain-independent, domain-specific, and organization-specific patterns, business terms, cubes, and dimensions. The level-specific subset-subset comparison, the year-specific subset-subset comparison, and the subset-complement comparison pattern are assigned to the *domain-independent* level of abstraction, since they do not contain references to domain-specific or organization-specific eMDM elements. It should be noted that the year-specific subset-subset comparison is obtained through partial instantiation of the level-specific subset-subset comparison. In addition, the domain-independent dimension `Time` and the unary dimension predicate `2019` are also assigned to the domain-independent level. The *domain-specific* level of abstraction contains the patterns *breed-specific subset-subset comparison* and *farm-specific subset-subset comparison*. The former can obtained through partial instantiation of the level-specific subset-subset comparison pattern by binding the level name `Main Breed` to one of the parameters, the latter can be obtained through partial instantiation by binding the level name `Farm` to one of the parameters. Finally, the *organization-specific* level of abstraction contains the subset-subset comparison of facts from Happy Milk's first farm site, which is obtained through partial instantiation of the domain-specific *farm-specific subset-subset comparison* pattern by binding the unary dimension predicate name `Farm Site 1` to one of the parameters. ♢

The organization principle for patterns also applies to business terms, with the exception that only referenced entities of an eMDM can be considered as the business vocabulary used. Nevertheless, future work may propose different organization principles. Likewise, there may be different definitions of what qualifies as "domain-specific", and the classification of a pattern as domain-specific may be subjective. For example, one may argue that a `Sales` cube is domain-independent since products and services are sold in multiple domains, while

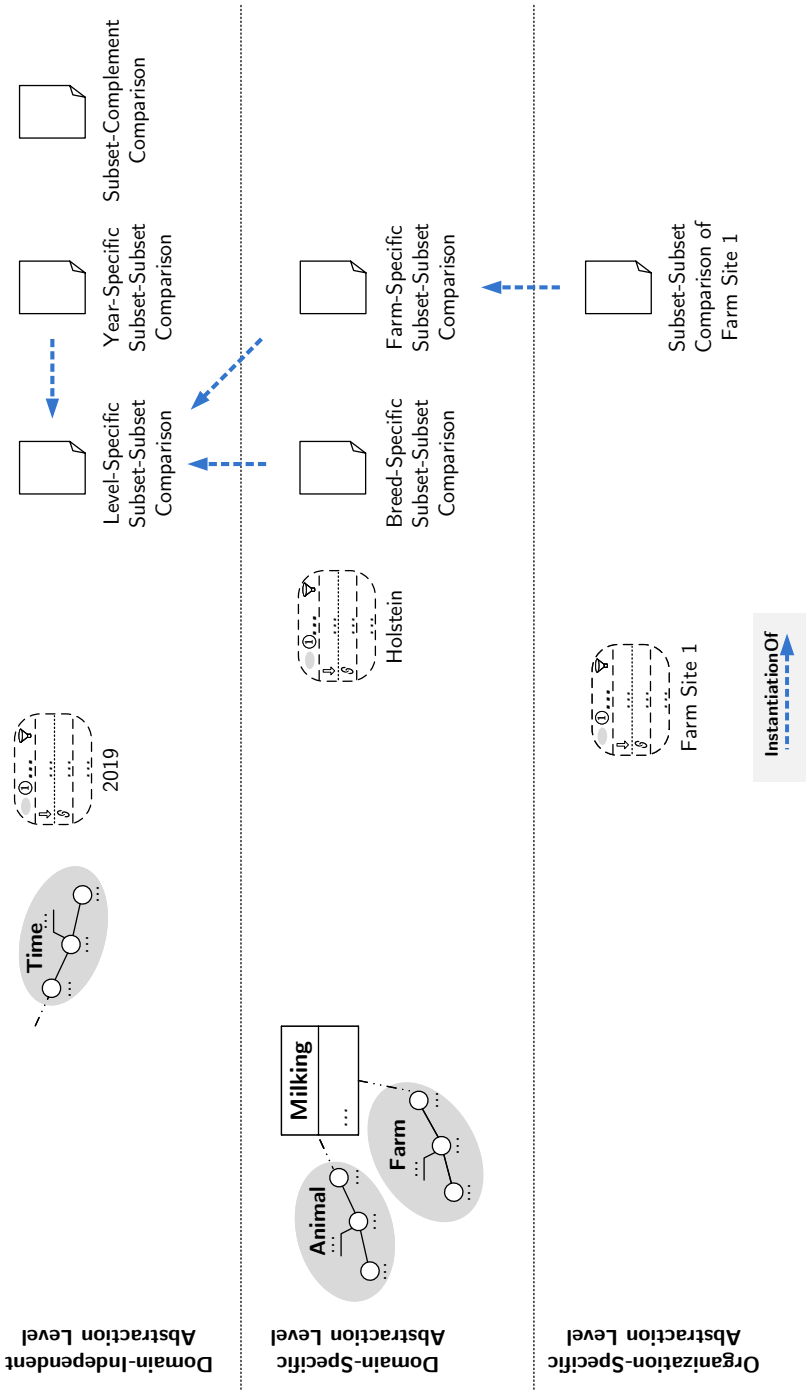


Figure 7.1: Exemplified organization of OLAP patterns and elements of an eMDM along levels of abstraction. Patterns from the same or a more specific abstraction level are obtained through (partial) instantiation.

another would argue that a sales cube is domain-specific for the retail sector.

7.2 Domain-Independent Patterns

Domain-independent patterns do not refer to any particular, domain-specific eMDM elements. Hence, domain-independent patterns contain only typed variables and constraints without domain-specific name constants. Thus, either no eMDM elements at all or only fragments of domain-independent eMDMs are referenced by a domain-independent pattern. For example, the comparison of two groups of facts derived from one cube represents a domain-independent pattern without any references to domain-specific cubes or dimensions. Likewise, the comparison of a measure value for one period with the measure value for the same period in the previous year can be considered a domain-independent pattern, which refers to a domain-independent time dimension.

OLAP patterns that are domain-independent describe abstract solutions to specific types of information needs without referencing domain knowledge and are therefore applicable to a wide range of domain-specific and organization-specific information needs. The descriptions and templates of domain-independent patterns remain generic, which allows them to be understood in various contexts, at the cost of reduced expressiveness and complexity of the supported queries.

```
1  CREATE DIMENSION "Time" WITH
2    LEVEL PROPERTIES
3      "Date":"Date";
4      "Month":"Month No";
5      "Year":"Year No";
6    END LEVEL PROPERTIES;
7
8    ATTRIBUTE PROPERTIES
9      "Month Label":"Month Name";
10   END ATTRIBUTE PROPERTIES;
11
12   CONSTRAINTS
13     "Date" ROLLS_UP_TO "Month";
14     "Month" ROLLS_UP_TO "Year";
15     "Month" DESCRIBED_BY "Month Label";
16   END CONSTRAINTS;
17 END DIMENSION;
```

Listing 7.1: Definition of domain-independent dimension `Time`

The identification and definition of domain-independent patterns is challenging since the

discovery and description of solutions that are independent of a domain usually requires cross-domain experience and knowledge. As pointed out by Fowler [11, p. 7], the identification of domain-independent patterns is especially challenging since one cannot be certain of its validity beyond its original domain. Nevertheless, we were able to identify several domain-independent patterns (see Appendix B) by drawing from experience from previous projects [5], [6] carried out in different domains.

Example 7.2 (Domain-Independent Pattern)**.** The domain-independent level of abstraction depicted in Figure 7.1 contains the *level-specific subset-subset comparison* pattern (Figure 7.2—Context), which allows to compare two aggregated measure values of two groups of facts from a single source cube, where both groups relate to a common specific level. The level can be specified by the parameter ⟨`baseLevel`⟩ allowing to restrict possible dimensions bound to ⟨`baseDim`⟩ to those that provide a level with the bound name. The ⟨`baseLevel`⟩ can in turn be restricted by a unary dimension predicate specified for the parameter ⟨`baseDimSlice`⟩. The domain-independent character of the level-specific subset-subset comparison pattern allows its application to satisfy a wide range of possible specific information needs (Figure 7.2 and Figure 7.3), but it may be difficult for pattern users to understand the pattern since its description remains generic (Figure 7.2—Problem and Solution). The pattern can also be used to derive specializations through partial instantiation. The year-specific subset-subset comparison, for example, is obtained through partial instantiation of the level-specific subset-subset comparison by binding the level name `Year` to the parameter ⟨`baseLevel`⟩ and `Year No` to the parameter ⟨`baseLevelDom`⟩ (Listing 7.3). The level `Year` is, for example, provided by the domain-independent dimension `Time` (Listing 7.1, Line 5), which in turn can be restricted to the year 2019 by the domain-independent unary dimension predicate `2019` (Listing 7.2). ♢

```
CREATE UNARY_DIMENSION_PREDICATE "2019" APPLIES TO <ctx>:DIMENSION WITH
  CONSTRAINTS
    <ctx> HAS LEVEL "Year";
    <ctx>."Year":"Year No";
  END CONSTRAINTS;
END UNARY_DIMENSION_PREDICATE;

CREATE TERM DESCRIPTION FOR "2019" WITH
  LANGUAGE = "English";
  ALIAS = "Year 2019";
  DESCRIPTION = "Restriction to year 2019.";
END TERM DESCRIPTION;

CREATE TERM TEMPLATE FOR "2019" WITH
  LANGUAGE = "SQL";
  DIALECT = "ORACLEv11";
  EXPRESSION = "<ctx>=2019";
END TERM TEMPLATE;
```

Listing 7.2: Definition of domain-independent unary dimension predicate `2019`

Figure 7.2: Aliases, problem, solution, and context of the level-specific subset-subset comparison pattern

Level-Specific Subset-Subset Comparison

Also Known As

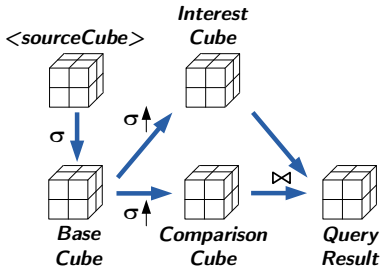
Level-Specific Group-Group Comparison

Problem

Retrieve aggregated measure values for two specified groups of facts relating to a specific level from a single source cube, which should be compared by calculating a comparative measure.

Solution

From the ⟨sourceCube⟩, select the set of relevant facts using the unary cube predicate ⟨baseCubeSlice⟩ and the unary dimension predicate ⟨baseDimSlice⟩ – the result serves as the *base cube* for further analysis. ⟨baseDimSlice⟩ selects over the dimension referenced by the ⟨baseDimRole⟩ that provides the level referenced by ⟨baseLevel⟩ with a domain referenced by ⟨baseLevelDom⟩. From that base cube, select *interest cube* and *comparison cube* using conditions over the dimension role ⟨compDimRole⟩ according to the unary dimension predicates ⟨iDimSlice⟩ and ⟨cDimSlice⟩, respectively. Perform a roll-up for *interest cube* and *comparison cube* according to the ⟨groupCond⟩ dimension grouping over the ⟨joinDim⟩ dimension referenced by the dimension role ⟨joinDimRole⟩ and compute a unary calculated measure ⟨cubeMeasure⟩. To obtain the *query result cube*, join the *interest cube* and *comparison cube* over the ⟨groupCond⟩ dimension grouping and compute a binary calculated measure ⟨compMeasure⟩.



Context

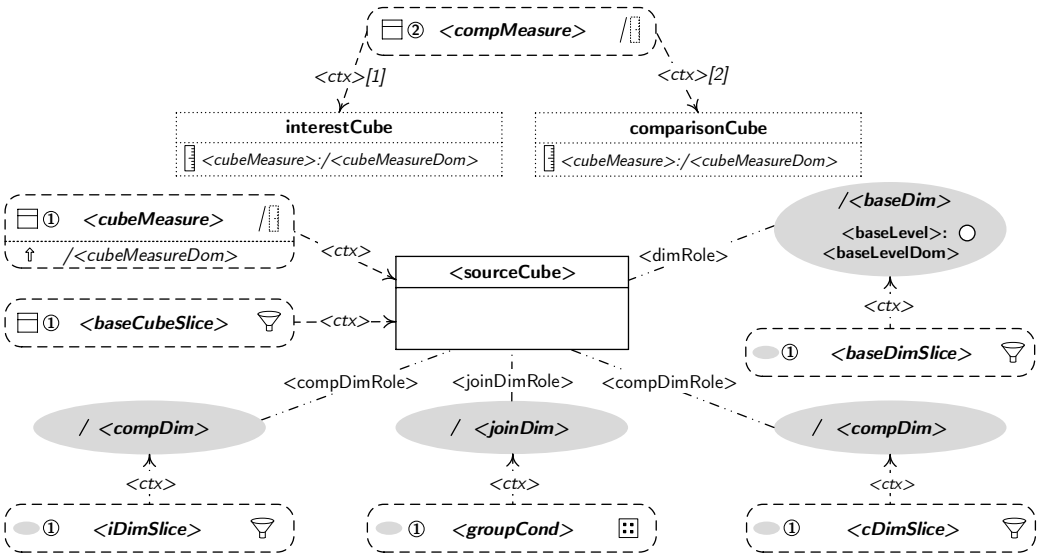


Figure 7.3: A template and related patterns for level-specific subset-subset comparison (continued from Figure 7.2)

Template (Data Model: Relational, Variant: Star Schema, Language: SQL, Dialect: ORACLEv11)

```
WITH baseCube AS (
  SELECT  *
  FROM    <sourceCube> sc
          JOIN <baseDim> bd ON
             sc.<baseDimRole> = bd.$dimKey(<baseDim>)
  WHERE   $expr(<baseCubeSlice>, sc) AND
          $expr(<baseDimSlice>, bd)
),
interestCube AS (
  SELECT $expr(<groupCond>, jd),
         $expr(<cubeMeasure>, bc) AS <cubeMeasure>
  FROM   baseCube bc
         JOIN <compDim> cd ON
            bc.<compDimRole>=cd.$dimKey(<compDim>)
         JOIN <joinDim> jd ON
            bc.<joinDimRole>=jd.$dimKey(<joinDim>)
  WHERE  $expr(<iDimSlice>, cd)
  GROUP BY $expr(<groupCond>, jd)
),
comparisonCube AS (
  SELECT $expr(<groupCond>, jd),
         $expr(<cubeMeasure>, bc) AS <cubeMeasure>
  FROM   baseCube bc
         JOIN <compDim> cd ON
            bc.<compDimRole>=cd.$dimKey(<compDim>)
         JOIN <joinDim> jd ON
            bc.<joinDimRole>=jd.$dimKey(<joinDim>)
  WHERE  $expr(<cDimSlice>, cd)
  GROUP BY $expr(<groupCond>, jd)
),
SELECT $expr(<groupCond>, ic),
       ic.<cubeMeasure> AS "Group of Interest",
       cc.<cubeMeasure> AS "Group of Comparison",
       $expr(<compMeasure>, ic, cc) AS <compMeasure>
FROM   interestCube ic
       JOIN comparisonCube cc ON
         $expr(<groupCond>, ic) = $expr(<groupCond>, cc)
```

Related Patterns

Breed-Specific Subset-Subset Comparison

```
1 INSTANTIATE PATTERN "Level-Specific Subset-Subset Comparison" AS
2   "Year-Specific Subset-Subset Comparison" WITH
3      <baseLevel> = "Year",
4      <baseLevelDom> = "Year No";
```

Listing 7.3: Obtaining domain-independent year-specific subset-subset comparison through partial instantiation of level-specific subset-subset comparison

The organization of patterns in the domain-independent level of abstraction can facilitate the discovery of general solutions for general types of information needs. First, there are few domain-independent patterns compared to domain-specific and organisation-specific patterns. The limited number of patterns at the domain-independent level enables pattern users to identify appropriate solutions more quickly. Second, pattern users can be confident that the patterns at this level are formulated in such a way that they can understand those patterns regardless of their domain background. Thus, pattern users regardless of their domain-specific and organization-specific knowledge should be able to understand general solutions and apply the patterns to a specific domain or organization. For this purpose, it is only required to take into account the business vocabulary during instantiation, which is particularly advantageous for pattern authors who have to work as experts in different domains.

7.3 Domain-Specific Patterns

Domain-specific patterns contain descriptions and templates that use domain-specific business vocabulary, i.e., in contrast to domain-independent patterns, domain-specific patterns are typically defined over an associated domain-specific reference eMDM [6]. A reference eMDM includes domain-specific multidimensional model elements and a vocabulary of business terms. For example, a pattern specific to the manufacturing sector may refer to manufacturing-related calculated measures, cubes, dimensions, and predicates. Standard catalogs of key performance indicators for different domains, e.g., engineering [85], construction [86], or hospitality [87], may form the basis for such domain-specific vocabularies of business terms. Constraints and derivation rules set the boundaries for sensible applications of domain-specific patterns in the context of a particular data warehouse.

The domain-specific descriptions and templates, render such patterns more understandable and usable for pattern users, with higher expressiveness and complexity of the supported queries. Pattern users from other domains with similar information needs, however, may struggle to understand and use domain-specific patterns, which potentially hinders knowledge transfer across domains. Potential pattern users are often not well acquainted with business

vocabulary specific to an unfamiliar domain and, consequently, users will not or only partially understand the meaning and purpose of a pattern from an unfamiliar domain.

In contrast to domain-independent patterns, domain-specific patterns can be more easily identified because pattern authors will usually have roots in a specific application domain and, consequently, have an in-depth understanding of the types of information needs to be satisfied in that domain. In addition, the abstraction to a domain-specific level of abstraction requires less experience and knowledge as only a single domain has to be considered. We have identified domain-specific patterns both by performing a bottom-up abstraction of specific OLAP queries as well as by performing a top-down specialization from domain-independent patterns (see Appendix C).

```
1 INSTANTIATE PATTERN "Level-Specific Subset-Subset Comparison" AS
2   "Breed-Specific Subset-Subset Comparison" WITH
3     <baseDimRole> = "Cattle",
4     <baseLevel> = "Main Breed",
5     <baseLevelDom> = "Breed Name";
```

Listing 7.4: Obtaining domain-specific breed-specific subset-subset comparison through partial instantiation of level-specific subset-subset comparison

```
1  CREATE UNARY_DIMENSION_PREDICATE "Holstein" APPLIES TO <ctx> WITH
2    CONSTRAINTS
3       <ctx>."Main Breed":"Breed Name";
4    END CONSTRAINTS;
5  END UNARY_DIMENSION_PREDICATE;
6
7  CREATE TERM DESCRIPTION FOR "Holstein" WITH
8    LANGUAGE = "English";
9    ALIAS = "Cattle Breed Holstein";
10   DESCRIPTION = "Restriction of result to cattle of main breed
         Holstein";
11 END TERM DESCRIPTION;
12
13 CREATE TERM TEMPLATE FOR "Holstein" WITH
14   LANGUAGE = "SQL" ;
15   DIALECT = "ORACLEv11" ;
16   EXPRESSION = "<ctx>.""Main Breed"" = ""Holstein""";
17 END TERM TEMPLATE;
```

Listing 7.5: Definition of domain-specific unary dimension predicate `Holstein`

Example 7.3 (Domain-Specific Pattern)**.** The domain-specifc level of abstraction depicted in Figure 7.1 contains the breed-specific and the farm-specific subset-subset comparison pattern. The breed-specific subset-subset comparison (Figure 2.5 and Figure 2.6) allows to compare two aggregated measure values of two groups of facts from a single source cube where both groups relate to a common level `Main Breed`. The breed-specific subset-subset comparison pattern is obtained through partial instantiation of level-specific subset-subset comparison (Listing 7.4) by binding `Cattle` to the parameter ⟨`baseDimRole`⟩, `Main Breed` to the parameter ⟨`baseLevel`⟩, and `Breed Name` to the parameter ⟨`baseLevelDom`⟩ (Figure 7.2–Context). The level referenced by the bound parameter ⟨`baseLevel`⟩ restricts possible dimensions bound to ⟨`baseDim`⟩ to those that provide a level of that name with a corresponding domain defined by ⟨`baseLevelDom`⟩. The referenced level `Main Breed` with the domain `Breed Name` can be, for example, provided by the domain-independent dimension `Animal` (Listing 4.2, Line 5), while the domain-independent unary dimension predicate `Holstein` can be used to restrict the level `Main Breed` to cattle of breed Holstein (Listing 7.5). ♢

Organizing patterns at the domain-specific level of abstraction should promote discoverability of solutions for domain-specific types of information needs, as pattern users can browse for patterns that are relevant to their domain. However, pattern users may struggle with applying solutions from a different domain to their own, meaning that the transfer of knowledge between domains may be hindered.

7.4 Organization-Specific Patterns

Organization-specific patterns are defined with respect to the eMDM of a particular organization data warehouse. Organization-specific patterns take into account the specific business vocabulary from the company's corporate language, which can deviate from domain business vocabulary. For example, an organization-specific pattern for a manufacturing company that produces brushes may refer to a multidimensional model that has been extended and adapted from a reference eMDM for the manufacturing domain [6]. Patterns based on the eMDM for that brush manufacturer are more understandable for in-house BI users of the company's various departments and subsidiaries since those patterns employ common organization-specific business vocabulary.

Compared to domain-independent and domain-specific patterns, the specificity of the descriptions as well as the significance and complexity of the supported queries increase with organization-specific patterns. This is based on the fact that similar organisation-specific information needs have to be abstracted only to a small extent in order to obtain

corresponding organisation-specific patterns. As a result, less detail is lost, as patterns in higher levels of abstraction have to be defined more generally.

Example 7.4 (Organization-Specific Pattern)**.** The organization-specific level of abstraction depicted in Figure 7.1 contains the *subset-subset comparison of Farm Site 1* pattern. This pattern allows to compare two aggregated measure values of two groups of facts from a single source cube where both groups relate to a common level `Farm Id` that is being restricted to by the unary dimension predicate `Farm Site 1`. The *subset-subset comparison of Farm Site 1* pattern is obtained through partial instantiation of the farm-specific subset-subset comparison by binding `Farm Site 1` to the parameter ⟨`baseDimSlice`⟩ (Listing 7.6), while the farm-specific subset-subset comparison is obtained through partial instantiation of the level-specific subset-subset comparison by binding `Farm Id` to the parameter ⟨`baseLevel`⟩ and `Farm Code` to the parameter ⟨`baseLevelDom`⟩. The unary dimension predicate `Farm Site 1` allows to restrict dimension properties named `Farm Id` with a `Farm Code` as its domain (Listing 7.6, Line 3) to the `Farm Site 1` by restricting the property to the farm id `FC0001` (Listing 7.6, Line 16). ♢

```
1 INSTANTIATE PATTERN "Farm-Specific Subset-Subset Comparison" AS
2   "Subset-Subset Comparison of Farm Site 1" WITH
3     <baseDimSlice> = "Farm Site 1";
```

Listing 7.6: Obtaining organization-specific subset-subset comparison of Farm Site 1 through partial instantiation of farm-specific subset-subset comparison

```
1 CREATE UNARY_DIMENSION_PREDICATE "Farm Site 1"
      APPLIES TO <ctx> WITH
2   CONSTRAINTS
3       <ctx>."Farm Id":"Farm Code";
4   END CONSTRAINTS;
5 END UNARY_DIMENSION_PREDICATE;
6
7 CREATE TERM DESCRIPTION FOR "Farm Site 1" WITH
8   LANGUAGE = "English";
9   ALIAS = "First Farm Site";
10  DESCRIPTION = "Restriction of dimension to
        Happy Milk's first farm site";
11 END TERM DESCRIPTION;
12
13 CREATE TERM TEMPLATE FOR "Farm Site 1" WITH
14   LANGUAGE = "SQL";
15   DIALECT = "ORACLEv11";
16   EXPRESSION = "<ctx>.""Farm Id"" = ""FC0001""";
17 END TERM TEMPLATE;
```

Listing 7.7: Definition of organization-specific unary dimension predicate `Farm Site 1`

Organization-specific patterns should be particularly beneficial for in-house pattern users. The discovery of solutions for organization-specific types of information needs is fostered as the patterns are easy to understand because organization-specific business vocabulary is used. However, similar to domain-specific patterns, knowledge transfer across domains and even across organization may be limited.

7.5 Pattern Catalogs

Levels of abstraction were discussed with regard to the classification of patterns and eMDM elements, taking into account the employed business vocabulary. Orthogonal to such a classification, patterns and eMDM elements can be grouped by organization elements that take into account their intended use. To this end, we present catalogs for grouping patterns according to their intended use. In addition, vocabularies and multidimensional models are used analogously to group business terms and entities, respectively. It should be noted that an eMDM is represented by a multidimensional model and a vocabulary.

Catalogs can be created at different levels of abstraction to group the patterns of one level. If patterns are grouped in a catalog, it must be ensured that the necessary business terms and entities are also provided, which are referenced by the grouped patterns via constants.

Therefore, the business terms and entities referenced by the grouped patterns must be provided by the eMDM, i.e., the vocabulary and the multidimensional model, of the level of abstraction.

The available patterns that can be grouped within an abstraction level depends on the abstraction level. Patterns in a higher level of abstraction are available to lower levels of abstraction. That is, patterns associated with one level of abstraction can also be associated with a lower level of abstraction at the same time. However, the reverse is not true, i.e. patterns of a lower abstraction level are not available in higher abstraction levels. To group a pattern from a higher level of abstraction in a catalog of a lower level, the pattern from the higher level of abstraction is either directly reused or adapted through (partial) instantiation. Whenever a pattern from a higher level of abstraction is either directly reused or adapted through instantiation, the eMDM of the more specific level of abstraction must provide all eMDM elements of the more abstract eMDM that are required for the reuse of the higher level pattern. Similarly, vocabularies and multidimensional models can group business terms and entities from the same level and from higher levels of abstraction through reuse.

Domain-independent patterns are comparable to software design patterns, which are also domain-independent. Similar to software design patterns, which are grouped into creational, structural, and behavioural software design patterns, domain-independent OLAP patterns can be grouped by considering different aspects [13]. For example, domain-independent patterns can be grouped according to whether only one (homogeneous) or two (heterogeneous) source cubes are considered, whether a comparison of groups is performed, and whether the pattern takes into account domain-independent entities and business terms such as location, time, customer, and product dimensions with corresponding business terms.

Example 7.5 (Domain-Independent Catalog)**.** Figure 7.4 illustrates the `Homogeneous Comparison Catalog`, which groups the level-specific subset-subset, the year-specific subset-subset, and the subset-complement comparison pattern, as each of them represents a domain-independent comparison of two groups from a source cube. Additionally, the domain-independent vocabulary `Temporal Vocabulary` groups the unary dimension predicate `2019`, while the multidimensional model `Temporal MDM` groups the dimension `Time`. ♢

Catalogs of domain-specific patterns are defined to group patterns defined for a specific reference eMDM. These domain-specific patterns can either be defined from scratch or obtained through partial instantiation of domain-independent patterns in the context of a particular domain. Elements matching the names thus bound must be provided by the corresponding domain-specific reference eMDMs, if the instantiated patterns require them.

Example 7.6 (Domain-Specific Catalog)**.** Figure 7.4 illustrates the `agriProKnow Catalog`, which groups the breed-specific subset-subset comparison and the farm-specific subset-subset comparison. Both the breed-specific and the farm-specific subset-subset comparison patterns can be obtained by partially instantiating the level-specific subset-subset comparison with respect to the agriProKnow reference eMDM (see Example 7.3 and Example 7.4). The agriproKnow reference eMDM is in turn represented by the domain-specific `agriProKnow MDM` and the `agriProKnow Vocabulary`, respectively. ♢

Within an organization, catalogs can be created by considering the organization's eMDM or only fragments of it. The organization's eMDM is usually realized by a data warehouse, while fragments of an organization's eMDM are realized by corresponding data marts. The former is considered by a catalog when patterns for the entire organization are to be grouped, while the latter are considered by a catalog when patterns for specific organizational units, e.g., subsidiaries and departments, are to be grouped. An organization's eMDM can be based on an existing reference eMDM by reusing corresponding eMDM elements. Again, the organization-specific eMDM must provide all eMDM elements required by patterns that are reused or partially instantiated from a higher level of abstraction.

Example 7.7 (Organization-Specific Catalog)**.** The `Happy Milk Catalog` in Figure 7.4 groups patterns that are to be applied to Happy Milk's eMDM. Happy Milk's eMDM is represented by `Happy Milk MDM` and `Happy Milk Vocabulary` and is based on agriProKnow's reference eMDM, i.e., eMDM elements of agriProKnow's reference eMDM have been reused. It should be noted that the organization-specific eMDM contains additional organization-specific business terms and entities such as the unary dimension predicate `Farm Site 1`. ♢

The organization elements, i.e., pattern catalogs, business term vocabularies, and multidimensional models, enable the grouping of patterns and eMDM elements within a level of abstraction. Patterns and eMDM elements from higher abstraction levels can thereby be grouped into lower levels by simply reusing them or adapting them to the current abstraction level through partial instantiation. It is important that the eMDM associated with a pattern or business term provides all the necessary eMDM elements referenced by the pattern and business term, respectively.

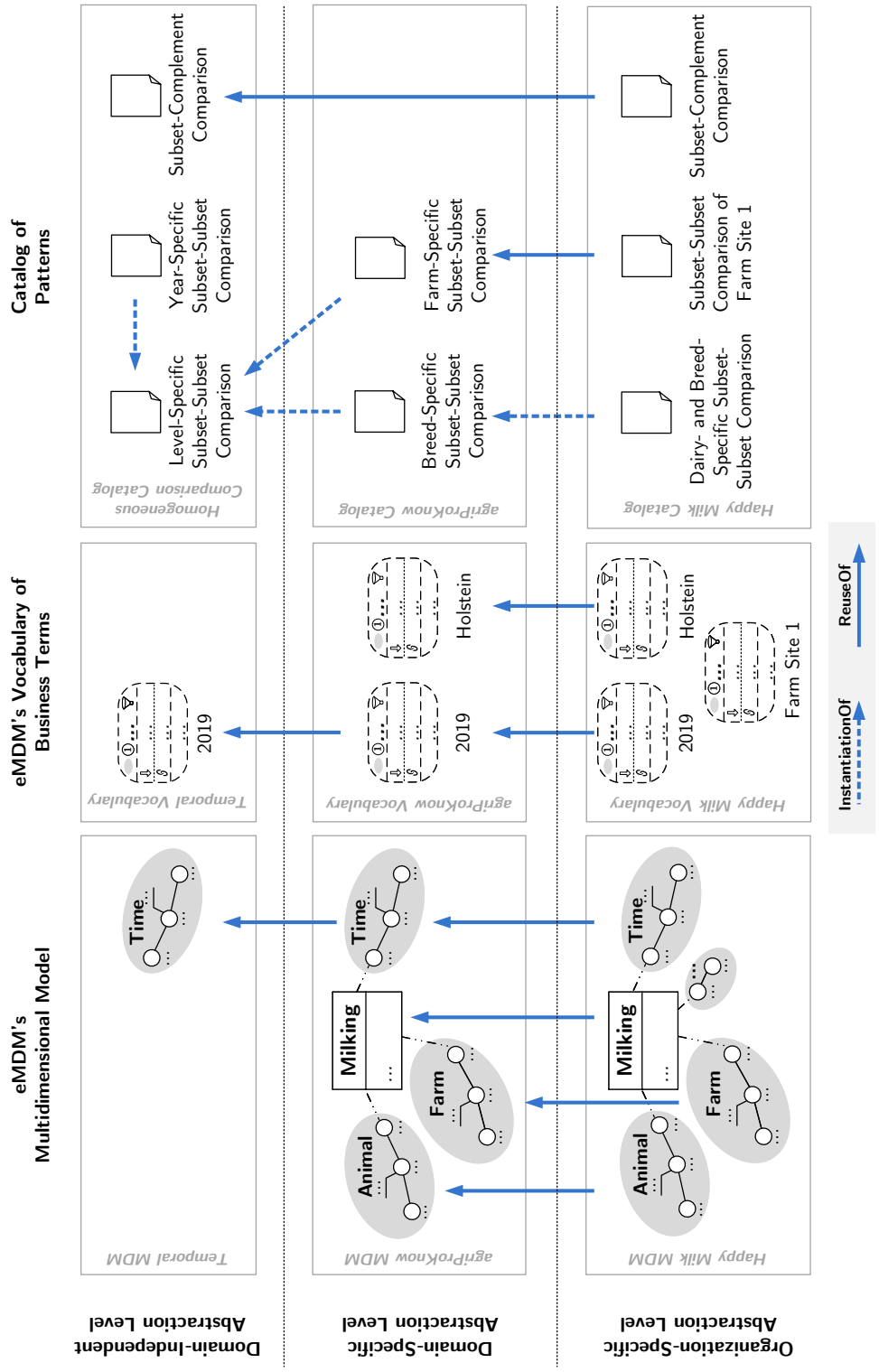


Figure 7.4: Exemplified organization of OLAP patterns and elements of an eMDM into catalogs, vocabularies, and multidimensional models along levels of abstraction.

Chapter 8

Proof-of-Concept Prototype

In this chapter we present a proof-of-concept prototype implementation of the pattern-based approach to multidimensional data analysis that is based on the definitions and grammar for OLAP patterns defined in the previous chapters, following an architecture that we introduce in this chapter. The prototype implementation supports management and execution of OLAP patterns by providing an application for managing the repository and an editor, which have been implemented as part of a master's thesis [88] based on the architecture described here. The prototype presented in this chapter is a new implementation of the pattern-based approach, independent of the prototype developed in the agriProKnow project [1], [2], which was tailored to the needs of that project. In contrast to the prototype developed by Simon Schausberger for the agriProKnow project [89], the proof-of-concept prototype described here implements the concepts necessary to represent enriched multidimensional models and OLAP patterns as introduced in Chapters 4 and 5. In addition, the new prototype supports the operations necessary for using patterns as introduced in Chapter 6. A simple editor facilitates interaction between user and repository by allowing to submit statements to the repository and displaying result messages. We discuss the overall prototype architecture in Section 8.1 and describe the functionality provided by the prototype in Section 8.2. In Section 8.3, we describe the prototype's components. Finally, in Section 8.4, we demonstrate how the running example is supported by the prototype.

8.1 Architecture

The implementation architecture of the OLAP pattern prototype is composed of the following three main components: editor, repository, and the database (Figure 8.1). Although the data warehouse system is not directly part of the implementation, it is referenced in the overall architecture since a pattern author describes the logical realization of a pattern using a multidimensional model of a data warehouse enriched with business terms.

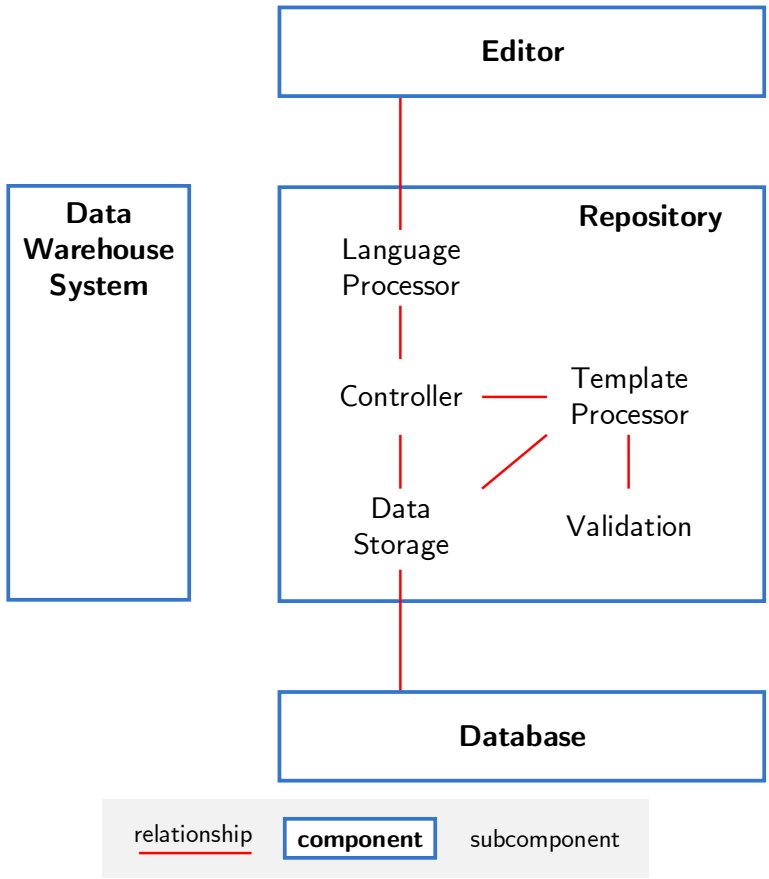


Figure 8.1: System architecture, showing the relationships between the major components and its subcomponents.

The core component is the repository, which includes the following subcomponents: language processor, controller, data storage, template processor, and validation. The language processor allows to process statements formulated by pattern authors and users and to transform those statements into an object representation. The controller takes over the object representation and decides on further steps to realize the actions defined by the statement, while the data storage component accesses the database to store and retrieve necessary object representations. The template processor allows for resolving macro calls in pattern templates and their substitution with corresponding business term expressions. Finally, the validation component, first, grounds derived elements by evaluating the corresponding derivation rules and, second, checks whether a ground pattern that is to be executed is actually applicable, taking into account the associated eMDM represented by the specified multidimensional model and vocabulary.

It should be noted that the editor was implemented along with core components of the repository as part of a master's thesis [88] according to the specifications here and in the

previous chapters. The master's thesis thus describes the implementation of all components except for the validation component, which was completed separately and is described here.

8.2 Functionality

The core components presented in Section 8.1 reflect the basic structure of the prototype without describing the functionalities relevant for a BI user. In this section we describe the functionality provided by the prototype from the perspective of both a pattern author and a user, i.e., we address the functionality provided by the prototype for the application of the pattern-based approach to multidimensional data analysis, which includes the definition of eMDM elements and the definition of patterns (see Chapters 4 and 5) but also the usage of patterns (see Chapter 6). Furthermore, we describe the functionality that the prototype offers to organize eMDM elements and patterns (see Chapter 7).

In order to clearly describe the functionality, we distinguish between content and organization elements:

- *Content elements* represent cubes and dimensions, different types of business terms, and OLAP patterns; descriptions and templates of patterns and business terms are also considered as content elements.
- In contrast, *organization elements* represent multidimensional models, vocabularies, and catalogs that allow for the arrangement of content elements into hierarchical structures. Additionally, repository elements allow to group multidimensional models, vocabularies, and catalogs.

It should be noted that an enriched multidimensional model is thus represented by a multidimensional model and a vocabulary.

The functionality provided by the prototype is represented by the language statements that can be processed. For this purpose, the supported language statements are described with respect to the organization and content elements to be managed and used. It should be noted that the language statements that can be processed by the prototype differ from those already presented in the previous chapters as follows.

- *Element identification via full path:* Content and organization elements in the repository application are identified not just by their name but by the combination of the path to the parent element and their name. For example, a pattern can be identified if its full path is specified, consisting of the name of the pattern's repository, the name of the catalog, and the name of the pattern itself, separated by slashes.

- *Omission of implicit context parameters:* Furthermore, for business terms, the context parameters are not specified as they are implicitly defined by the specified type of the business term.
- *Marking query language text:* Expression and query texts are enclosed within the delimiters *{ and }* to facilitate the identification of query specific strings during macro preprocessing via an island grammar [90, p. 109], while macro calls are embedded between these query texts.
- *Vocabulary keyword:* Additionally, the prototype supports the keyword `GLOSSARY` instead of the keyword `VOCABULARY`, as the latter represents a reserved keyword in the used ANTLR4 implementation.
- *Omission of explicit grounding:* The `GROUNDING` statement is omitted since grounding is implemented as part of the `EXECUTE` statement.

The functionality for managing organization elements and content elements is provided by `CREATE` and `DELETE` statements as follows.

- `CREATE` statements provide the functionality to define content elements and organization elements. A pattern author specifies the name and type of the element that is to be created. In addition, for multidimensional models, vocabularies, and catalogs, the pattern author also specifies the repository, i.e., the path, to which the element is added. For a cube, a pattern author specifies name, measures, and dimension roles, whereas for a dimension, a pattern author specifies the name, levels, and attributes with roll-up and described-by relationships between them. Regardless of whether a cube or a dimension is created, the pattern author specifies the path and name of the multidimensional model which the newly created element should be added to. For a business term, a pattern author specifies the name, type, constraints for expected types of elements, their properties, and their properties' domains, and the vocabulary which the business term will be added to. In addition, for calculated measures, the return type is also specified. Business terms can be further described by adding metadata: textual description of the business terms, information about the (natural) language of the textual description, and alias names of the business term. Templates of a business term contain the expression, with further metadata specifying the (formal) language and dialect in which the expression is formulated. Finally, for a pattern, the pattern author specifies name, parameters, local cubes, derived elements, constraints, and the path to the catalog which the pattern will be added to. Similarly to business terms, a pattern author specifies at least one

template for a pattern, which defines the expression along with (formal) language and dialect which the expression is formulated in, the data model and realization variant to be considered, and a description defining its aliases, the problem to be solved, the solution to be followed, an exemplified application, and related patterns.

- `DELETE` statements provide the functionality to remove organization and content elements. For this purpose, a pattern author specifies the path and name of the element to be deleted. The delete functionality is cascading, i.e., deleting an organization element deletes all subordinate organization elements and content elements, while deleting a pattern and business term deletes all subordinated descriptions and templates. Since templates and descriptions of patterns and business terms are unnamed they cannot be deleted by specifying path and name. To delete a particular description, a pattern author defines the path to and the name of the pattern or business term that the description belongs to as well as the language of the description to be deleted; each description in that language is then deleted. Templates are deleted in a similar way, except that all templates for the pattern or business term that correspond to the defined language, dialect, data model, and realization variant are deleted.

The functionality for using patterns is provided by `INSTANTIATE` and `EXECUTE` statements. The execution function incorporates the grounding of the parameter-free pattern being executed; a separate `GROUNDING` statement is not required.

- `INSTANTIATE` statements allow pattern authors and users to bind names of eMDM elements to unbound parameters of a pattern. The pattern to be instantiated is specified by its path and name, while the specialized pattern obtained by the instantiation is specified by a new name and by stating the catalog to which the specialized pattern is added. By specifying paths during instantiation, it is possible to instantiate a pattern from a higher level of abstraction in a catalog (from another repository), while the resulting pattern can be added to a more specific catalog.
- `EXECUTE` statements provide the functionality to execute a specific pattern to obtain at least one executable OLAP query. For this purpose, a pattern user specifies the path and name of the pattern to be executed as well as the paths and names of the multidimensional model and the vocabulary to be considered. If no templates are restricted, the execution returns an executable OLAP query for each available template of the pattern. Alternatively, a pattern user can restrict the pattern templates to be executed by specifying the language, dialect, data model, and realization variant of the template. It should be noted that the execution functionality also includes a

check whether the ground pattern to be executed is also applicable to the specified multidimensional model and vocabulary.

Finally, the functionality to browse organization elements and content elements is provided by corresponding SEARCH and SHOW statements as follows.

- SEARCH statements provide functionality for finding organization elements and content elements by specifying the element type, a search space, and a search term. The element type specifies the type of elements to be found. The search space is represented by a path to an organization element, but note that no search space is specified to search for repositories. For organization elements, the names of existing elements are searched for the search term. For content elements, in addition to the name, the sections in the description (if available) can also be searched. It should be noted that the search is not case sensitive.
- SHOW statements provide the functionality to inspect particular organization and content elements by retrieving the definition of those elements, i.e., the corresponding CREATE statement. For patterns and business terms, the SHOW statement also returns the descriptions and the templates. To issue a SHOW statement, pattern authors and users specify the path and name of the element to be inspected.

The prototype allows pattern authors to describe a data warehouse through a multidimensional model, represent the existing business vocabulary, and to define patterns as shown in Figure 2.3. The prototype also supports pattern users in instantiating patterns and executing parameter-free patterns as shown in Figure 2.3. The prototype thus provides all the functionality necessary to employ the pattern-based approach to multidimensional data analysis in a specific context through appropriate language statements.

8.3 Components

The architecture and the implemented functionality are described in Sections 8.1 and 8.2. This section elaborates on the components that realize the functionality described in the previous sections. We discuss the individual components in detail by examining each component from both a *functionality* and a *process* perspective. The functionality perspective describes the functions that are available to pattern authors and users. The process perspective describes the steps necessary to implement the provided functionality. For a more detailed logical, development, and physical perspective of the components we refer to the master's thesis of Moritz [88].

8.3.1 Repository

The repository application is the central component of the OLAP pattern prototype that provides the functionality for processing language statements, which adhere to the OLAP pattern grammar definition. Those language statements include `CREATE` statements for defining an organizational structure, i.e., creating a repository element with its subordinate multidimensional model, vocabulary, and catalog elements. The repository application supports the creation of multiple repository elements, which allows pattern catalogs, multidimensional models, and business term vocabularies to be managed at different levels of abstraction.

The prototype supports the definition of `CREATE` statements for the definition of content elements and their assignment to corresponding organizational elements, including the definition of cubes, dimensions, multidimensional models, business terms, vocabularies, OLAP patterns, catalogs. Likewise, `CREATE` statements can also be used for the creation of descriptions and templates for patterns and business terms, the templates of which contain query statements enclosed within `*{` and `}*`, with macro calls embedded between these query statements. The repository application persists all created content elements and organization elements in a database. `DELETE` statements also allow for removing organization elements and content elements, including `DELETE` statements that remove descriptions in a particular (natural) language and statements that remove templates for a particular data model, variant, language, and dialect from patterns and business term elements.

`SEARCH` statements allow to find organization elements within repository elements and content elements within the assigned organization element. Once a content element has been found, it can be inspected using the `SHOW` statement, which allow to request the definition of the element.

In addition to the definition of organization elements and content elements, the repository application supports `INSTANITATE` and `EXECUTE` statements, i.e., the instantiation and the subsequent execution of patterns while considering a specific multidimensional model enriched by business terms of a certain vocabulary. To this end, the pattern to be executed is first grounded, i.e., the derivation rules of derived elements are evaluated with respect to the specified multidimensional model and vocabulary yielding a ground pattern. Second, the prototype checks whether the ground pattern is applicable to the multidimensional model enriched with the vocabulary, taking into account local cubes of the pattern, i.e., checking whether all constraints can be satisfied. Only if the ground pattern is also applicable then the specified templates are subsequently processed to obtain an executable OLAP query for each processed template.

Statement Processing

The *language processor* expects the statement to be parsed as input and converts the statement into an object representation which is passed to the controller. The language processor consists of lexer, parser, and a semantic analyzer. The *lexer* represents the input statement consisting of a sequence of characters as words (or tokens). The *parser* then analyzes the structure of the input statement, taking into account the language grammar and the tokens, to obtain a tree representation: the abstract syntax tree. The abstract syntax tree is used by the *semantic analyzer* to ensure that logical conditions are met. For example, only previously defined variables can be used in derivation rules, thus preventing the definition of cyclic derivation rules. As a last step, the language processor transforms the checked abstract syntax tree into an object representation. This object representation also includes the operation to be performed with its arguments. In the case of `CREATE` statements, the object representation additionally contains the representation of the content element or the organization element to be created.

The *controller* implements the business logic to orchestrate the necessary steps that must be performed to process the recognized language statements. For each `CREATE` statement that refers to an organization element, the controller retrieves the corresponding parent organization element (if available), checks whether an organization element with the specified name already exists within the previously retrieved parent organization element, and creates the specified element if no such element already exists. Similarly, for each `CREATE` statement concerning a cube, dimension, business term, or pattern the controller retrieves the parent organization element and creates a corresponding element, replacing an existing element of the same name if such an element exists under the parent organization element. For each `CREATE` statement concerning a template or a description the controller requests the parent content element, i.e., pattern or business term specified by the path and name, checks whether a description with the specified language or a template for the data model and variant formulated in a specific language and dialect already exist before creating the template or description, assigning the created element or replacing an existing element. The controller processes `DELETE` statements by retrieving the parent organization element, if available, and removing the organization or content element from that parent element. For `DELETE` statements concerning templates and descriptions the pattern or business term is retrieved and either one or more templates and descriptions corresponding to the specified arguments are deleted. The controller further processes `INSTANTIATE` statements by retrieving the pattern to be instantiated, assigning the parameter bindings to each parameter occurring in the constraints, derivation rules, and local cubes, and finally creating the new pattern. The templates of the pattern that is instantiated are taken over and assigned to the new pattern, while the descriptions cannot be taken over as they no

longer correspond to the new pattern. Statements to execute parameter-free patterns are delegated from the controller to the template processor; the result of the `EXECUTE` statement is then returned. `SHOW` statements do not specify what type of element should be detailed, as only a path and a name are provided as arguments. The controller breaks this path into single names and retrieves for each name the corresponding element (in the context of its parent element) using the data storage. It should be noted that a corresponding element can be identified because the names of the organization and content elements within a parent element are unique. Finally, `SEARCH` statements are relayed to the data storage, to retrieve the matching elements.

The *data storage* subcomponent encapsulates the database functionality necessary to process the language statements accordingly. The data storage provides functionality to manage database sessions as well as to persist and delete content or organization elements. The data storage subcomponent also supports the retrieval of content and organization elements by name, taking into account organizational structures specified by a corresponding path. Furthermore, data storage supports the search for organization and content elements with a name containing a search term; for patterns and business terms their descriptions are also searched. It should be noted that the data storage supports the mapping from the object representation of elements to their representation in the database. In that case the mapping into a relational database, the mapping being bidirectional, i.e., the elements persisted in a relational representation can be mapped back to its object representation.

The *template processor* provides functionality to execute a pattern with respect to a multidimensional model enriched with a business term vocabulary. Either all templates available are executed or only those that match the specified data model, variant, language, and dialect. The template processor delegates the grounding and the checking for applicability of the parameter-free pattern to be executed to the validation subcomponent (see next section). Once the pattern is successfully grounded and also applicable to the specified multidimensional model and vocabulary, the template processor retrieves the pattern templates to be executed, while taking into account the specified arguments. First, all variables used in the templates are substituted by the names bound to the corresponding variables, resulting in ground templates. Second, the template processor resolves the `$dimKey` and the `$expr` macro calls by performing a language processing based on the grammar for macros. For this purpose, the template process employs a macro-lexer, a macro-parser and a semantic macro-analyzer, considering the macro island grammar. The *macro-lexer* represents a grounded template as tokens while the *macro-parser* provides an abstract syntax tree based on the recognized tokens and the macro island grammar. The *semantic macro-analyzer* then uses the abstract syntax tree to identify and resolve the calls to the macros. Each `$dimKey` macro call is resolved by retrieving the dimension using the name defined by the

argument and looking up the base level, the name of which is then used to substitute the corresponding `$dimKey` macro call in the ground template. Each `$expr` macro call is resolved by first retrieving the business term using the first argument of the macro call as the term's name. The second step is to retrieve the templates of the business term, which correspond to the language and dialect arguments of the `EXECUTE` statements. For unary business terms, the second argument from the `$expr` macro call is used to substitute the context parameter `<ctx>` in the retrieved template(s), while for binary business terms, the second argument is used to substitute the context parameter `<ctx>[1]` and the third argument is used to substitute the context parameter `<ctx>[2]`. The grounded template of the business term is then used to replace the corresponding `$expr` macro call in the template of the ground pattern. Finally, the executable OLAP queries are returned after all macro calls in the template of the ground pattern could be resolved.

Statement Validation

The repository application also offers statement *validation*, which provides the functionality for grounding patterns and checking the ground patterns' applicability to a multidimensional model and a vocabulary. As the validation subcomponent is not covered by the master's thesis of Moritz [88], we discuss the validation functionality in more detail here.

The *grounding* process for a pattern starts with the verification that the pattern is parameter-free. A pattern is parameter-free if there is no parameter without a bound name. Consequently, the grounding fails if the pattern is not free of unbound parameters.

The second grounding step is to recursively calculate a dependency graph for each parameter-free pattern. A dependency graph consists of multiple levels of derivation rules and describes dependencies between derivation rules in adjacent levels. The dependency graph is needed to evaluate potentially interdependent derivation rules of a parameter-free pattern. The semantics of a dependency graph is as follows: A derivation rule from a higher level can only be evaluated if the derivation rule from a lower level that the derivation rule from the higher level depends on is evaluated first. Consequently, a derivation rule from a higher level contains a derived element that must be bound to a name that is determined by evaluating the depending derivation rule from the lower level.

The `constructDependencyGraph` function is called recursively to construct the dependency graph of a parameter-free pattern. The `constructDependencyGraph` function is called by passing an empty dependency graph, the level number zero (indicating the level to be populated), and the derivation rules to be considered. The level (indicated by the passed level number) is populated by iterating over the derivation rules as follows. For each derivation rule, its first part is obtained, which is either a parameter, a constant, or a

derived element. It should be noted that the first part of each derivation rule defined by the domain of a property represents an entity, while it represents a business term for each derivation rule defined by the return type of a business term. If the first part is a parameter or a constant, the derivation rule is added to level zero of the dependency graph. However, if the first part of the derivation rule is a derived element, the `constructDependencyGraph` function checks the dependencies of this derived element as it indicates an entity derived by evaluating a derivation rule. The derivation rule that is currently being iterated is added to the current level if the derivation rule of its derived element (that represents its entity) has as its first part a derived element that is determined by a derivation rule of the lower level. After all derivation rules have been iterated over, the `constructDependencyGraph` checks whether all derivation rules are assigned to the dependency graph. If not all derivation rules have been added to the dependency graph, the `constructDependencyGraph` function is called again by passing the updated dependency graph, the incremented level number, and the remaining derivation rules to be considered. Otherwise, if all derivation rules are assigned, the recursion terminates by returning the constructed dependency graph.

The last step of grounding a parameter-free pattern is the bottom-up evaluation of the derivation rules in the dependency graph. The evaluation starts at level zero of the dependency graph by iterating each derivation rule. For each derivation rule, the validation subcomponent checks whether the rule is defined via the domain of a property or the return type of a business term. Derivation rules over domains are evaluated by first considering the local cubes of the pattern. If a local cube matches the name of the rule's entity, while providing a property matching the name of the rule's property, then the property's domain is retrieved and the domain's name is bound as the value of the corresponding derived element. The multidimensional model is considered next if no suitable domain is found in the properties of a local cube in the following way: If an entity matches the name of the rule's entity, while providing a property matching the name of the rule's property, then the property's domain is retrieved from the multidimensional model and the domain's name is bound as the value of the corresponding derived element. For each derivation rule defined by a return type of a business term, a business term matching the defined name is retrieved from the specified vocabulary. If the received business term provides a return type, the return type's name is assigned as the value of the corresponding derived element. Grounding ends once all derivation rules have been evaluated.

For parameter-free patterns that are successfully grounded, the validation subcomponent checks whether or not they are applicable to the specified multidimensional model and vocabulary. The *applicability check* of a pattern starts with verifying that the pattern is free of unbound parameters and derived elements. Thereafter, the constraints of the pattern are verified as follows: A constraint representing a type constraint is satisfied if an

element with a name matching the element name of the constraint (first compartment) and with a type matching the type name of the constraint (second compartment) can be found among the cubes and properties of the local cubes of the pattern, the entities and properties of the multidimensional model, or among the business terms in the vocabulary. Similarly, a property constraint is satisfied if in the local cubes or in the multidimensional model there is an entity with a name matching the entity name of the constraint (first compartment), while providing a property named after property name of the constraint (second compartment) with a type defined according to the type name of the constraint (third compartment). A domain constraint is satisfied if an entity with a name matching the entity name of the constraint (first compartment) is present in the local cubes of the pattern or in the multidimensional model that provides a property matching the property name of the constraint (second compartment) with a domain named according to the domain name of the constraint (third compartment). A return constraint is satisfied if a business term named after the term name of the constraint (first compartment) is present in the vocabulary that also provides the required return type defined by the return type name of the constraint (second compartment). A unary applicable-to constraint is satisfied if a business term can be found that is named after the term name of the constraint (first compartment); the validation subcomponent also checks whether the type of the retrieved business term matches the arity of the constraint. In addition, the validation subcomponent checks whether the application to the target entity is a valid business term application. To this end, the validation component binds the name of the target entity name of the unary applicable-to constraint (second compartment) to the context parameter <ctx> of the retrieved business term and checks whether the constraints of the business term can be satisfied by the local cubes of the pattern or the multidimensional model while taking into account the binding of the context parameter <ctx>. The check whether or not the type, property, and domain constraints of business terms are satisfied follows the same logic as described for the evaluation of the corresponding pattern constraints. The unary applicable-to constraint is satisfied if the binding leads to a valid business term application. Similarily, the satisfaction of binary applicable-to constraints can be determined in the same way, except that the second compartment of the binary applicable-to constraint is bound to the context parameter <ctx>[1], while the third compartment is bound to the context parameter <ctx>[2]. The ground pattern can be applied to the specified multidimensional model and vocabulary if and only if all constraints can be satisfied.

Ultimately, all of the components of the repository application provide functionality that is consumed by the corresponding editor application, which we describe in the following section.

8.3.2 Editor

The editor application is a client for accessing the functionality provided by the repository application. The editor provides a simple text interface through which a user can write statements and send them to the repository application. The formulated statements must adhere to the OLAP pattern grammar. It should be noted that the editor is a thin client, with no business logic implemented on the client side.

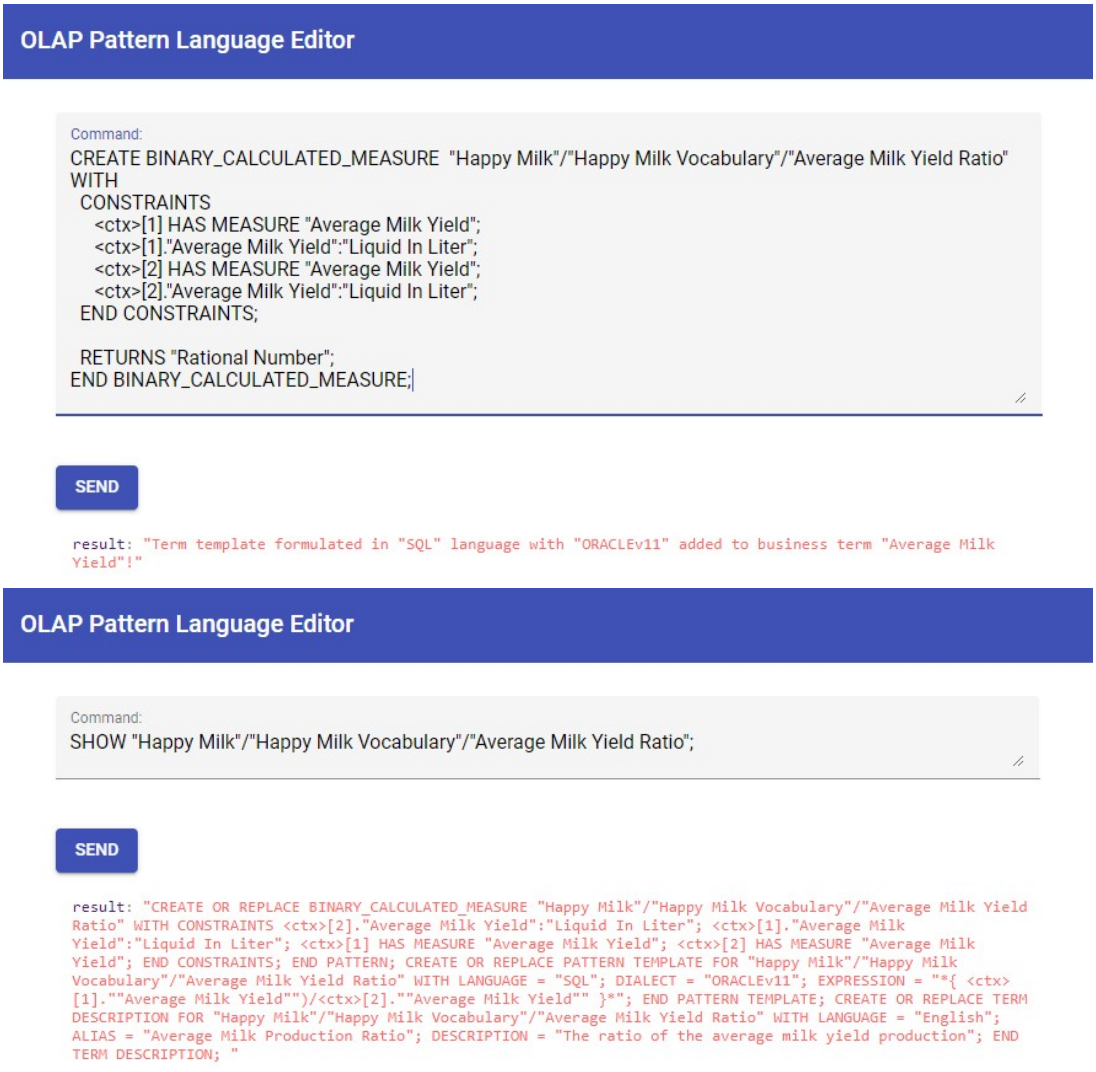


Figure 8.2: Command line interface provided by the Editor application exemplified.

The output received by the repository application is displayed as a JSON response in the result area below the input interface for the statements (see Figure 8.2). The result area also displays errors, for example, if a statement contains a syntactical error the error message from the parser is delegated to the user. In addition, pattern authors and users are informed

about the success or failure of CREATE, DELETE and INSTANTIATE statements. For EXECUTE statements the obtained executable OLAP queries are returned. For SEARCH statements the JSON representation of the elements found is returned, while for SHOW statements the corresponding CREATE statement is returned. It should be noted that SHOW statements concerning patterns and business terms also return the definitions of the corresponding templates and descriptions.

8.4 Usage Scenario

We demonstrate the functionality of the prototype by providing exemplified statements that need to be processed to support the Happy Milk scenario inspired by our experience gained in the agriProKnow project (Section 2.2). In particular, we show example statements necessary to apply the pattern-based approach to multidimensional data analysis in the Happy Milk scenario. It should be noted that only the most important statements are discussed; for the complete set of statement we refer to Appendix D.

```
1 CREATE REPOSITORY "Happy Milk";
2 CREATE CATALOGUE "Happy Milk"/"Happy Milk Catalog";
3 CREATE GLOSSARY "Happy Milk"/"Happy Milk Vocabulary";
4 CREATE MULTIDIMENSIONAL_MODEL "Happy Milk"/"Happy Milk MDM";
```

Listing 8.1: Definition of the Happy Milk company's organization structure

When employing the pattern-based approach to data analysis in the Happy Milk company, a pattern author, e.g., the data engineer of an external precision farming consulting agency, starts by specifying the organization structure. First, the pattern author creates a repository `Happy Milk` and then adds a catalog `Happy Milk Catalog`, a vocabulary `Happy Milk Vocabulary`, and a multidimensional model `Happy Milk MDM` (Listing 8.1).

```
1 CREATE DIMENSION "Happy Milk"/"Happy Milk MDM"/"Lactation" WITH
2    LEVEL PROPERTIES
3       "Day Of Lactation":"No Of Days";
4    END LEVEL PROPERTIES;
5 END DIMENSION;
```

Listing 8.2: Statement to define dimension `Lactation` for the multidimensional model `Happy Milk MDM`.

Once the organization structure is defined, the pattern author continues to conceptually model the company's data warehouse (Figure 2.4) by describing the dimensions and cubes.

The thus defined entities are added to the multidimensional model `Happy Milk MDM`. The exemplified statement in Listing 8.2 describes a dimension `Lactation` including a level `Day Of Lactation` of type `No Of Days` that is added to the `Happy Milk MDM`.

```
1 SHOW "agriProKnow Repository"/"agriProKnow Reference MDM"/"Lactation";
```

Listing 8.3: Statement to inspect dimension `Lactation` in the agriProKnow reference MDM

The statement in Listing 8.2 can be formulated from scratch by the pattern author or it can be reused. Therefore, the corresponding dimension `Lactation` in the `agriProKnow MDM` just needs to be inspected (Listing 8.3).

```
1  CREATE BINARY_CALCULATED_MEASURE  "Happy Milk"/"Happy Milk
       Vocabulary"/"Average Milk Yield Ratio" WITH
2    CONSTRAINTS
3       <ctx>[1] HAS MEASURE "Average Milk Yield";
4       <ctx>[1]."Average Milk Yield":"Liquid In Liter";
5       <ctx>[2] HAS MEASURE "Average Milk Yield";
6       <ctx>[2]."Average Milk Yield":"Liquid In Liter";
7    END CONSTRAINTS;
8
9    RETURNS "Rational Number";
10 END BINARY_CALCULATED_MEASURE;
11
12 CREATE TERM DESCRIPTION FOR "Happy Milk"/"Happy Milk
       Vocabulary"/"Average Milk Yield Ratio" WITH
13   LANGUAGE = "English";
14   ALIAS = "Average Milk Production Ratio";
15   DESCRIPTION = "The ratio of the average milk yield production";
16 END TERM DESCRIPTION;
17
18 CREATE TERM TEMPLATE FOR "Happy Milk"/"Happy Milk
       Vocabulary"/"Average Milk Yield Ratio" WITH
19   LANGUAGE = "SQL";
20   DIALECT = "ORACLEv11";
21   EXPRESSION = "*{ <ctx>[1].""Average Milk
         Yield"")/<ctx>[2].""Average Milk Yield"" }*";
22 END TERM TEMPLATE;
```

Listing 8.4: Definition of the binary cube predicate `Average Milk Yield Ratio` added to the vocabulary `Happy Milk Vocabulary`

Following the definition of the MDM, the pattern author continues to define the available business terms. The pattern author can, for example, define the business term `Average Milk`

Yield Ratio (Listing 8.4) from scratch or by reusing it from the corresponding reference vocabulary for precision dairy farming agriProKnow Reference Vocabulary and add it to Happy Milk Vocabulary.

The CREATE statement for the binary calculated measure Average Milk Yield Ratio omits the explicit specification of the context variables <ctx>[1] and <ctx>[2]. The corresponding CREATE statements concerning the description and template reference the business term by specifying the path to the business term Happy Milk/Happy Milk Vocabulary and its name Average Milk Yield Ratio. It should be noted that the pattern author defines the expression of the business term by embedding query-language-specific text blocks into *{ and }*.

Once the eMDM is defined by the multidimensional model and vocabulary the pattern author continues to define patterns. The pattern author defines the pattern Farm-Specific Subset-Subset Comparison for the Happy Milk Catalog through instantiation of the Level-Specific Subset-Subset Comparison pattern from the reference precision farming catalog agriProKnow Catalog by binding the dimension role name Farm to parameter ⟨baseDimRole⟩, the level name Far Id to parameter ⟨baseLevel⟩, and the domain name Farm Code to the parameter ⟨baseLevelDom⟩ (Listing 8.5).

```
1 INSTANTIATE PATTERN "agriProKnow Repository"/"agriProKnow
    Catalog"/"Level-Specific Subset-Subset Comparison" AS "Happy
    Milk"/"Happy Milk Catalog"/"Farm-Specific Subset-Subset Comparison"
    WITH
2     <baseDimRole> = "Farm",
3       <baseLevel> = "Far Id",
4    <baseLevelDom> = "Farm Code";
```

Listing 8.5: Definition of the Farm-Specific Subset-Subset Comparison pattern through instantiation

After the instantiation, the pattern author notices that the obtained pattern is faulty because there is no dimension providing a level with that name Far Id in Happy Milk MDM. To correct this mistake the pattern author deletes (Listing 8.6) the pattern and repeats the instantiation in Listing 8.5 with the correct level name Farm Id.

```
1 DELETE PATTERN "Happy Milk"/"Happy Milk Catalog"/"Farm-Specific
    Subset-Subset Comparison";
```

Listing 8.6: Deletion of faulty Farm-Specific Subset-Subset Comparison pattern

As soon as the data warehouse has been conceptually modeled and enriched with business terms, and as soon as all relevant patterns have been defined, the pattern author's work is complete.

```
1  INSTANTIATE PATTERN "Happy Milk"/"Happy Milk
       Catalog"/"Farm-Specific Subset-Subset Comparison" AS
2    "Happy Milk"/"Happy Milk Catalog"/"Farm site 1 Jersey-Holstein
         Milk Yield Comparison" WITH
3          <sourceCube> = "Milking",
4       <baseCubeSlice> = "Mid Lactation Phase",
5        <baseDimSlice> = "Farm Site 1",
6         <compDimRole> = "Cattle",
7           <iDimSlice> = "Jersey",
8           <cDimSlice> = "Holstein",
9         <joinDimRole> = "Farm",
10          <groupCond> = "Per Farm",
11        <cubeMeasure> = "Average Milk Yield",
12        <compMeasure> = "Average Milk Yield Ratio";
```

Listing 8.7: Instantiation of the farm-specific subset-subset comparison

Suppose that in the following weeks, the company detects a decrease in milk production of the farm `Farm Site 1`. To investigate this decrease in milk production, a pattern user, e.g., the farm manager in charge, would like to compare the breeds present on the farm to see if the decrease is attributable to a particular breed. The pattern user may start the investigation by calculating the ratio between the average milk yield of Jersey cattle and the average milk yield of old Holstein cattle, considering only animals with in their mid lactation phase from farm `Farm Site 1`. The pattern user recognises that this information need corresponds, at a more abstract level, to a farm-specific subset-subset comparison. For this reason, the pattern user instantiates the corresponding pattern according to Listing 8.7.

Chapter 9

Evaluation

In this chapter, we first discuss practical relevance and expressiveness of the proposed pattern-based approach to multidimensional data analysis, citing experience from cooperative research projects as evidence for the usefulness of the approach in Section 9.1. We then evaluate the approach in Section 9.2 using a framework for assessing the quality of domain-specific languages against the goals of a particular stakeholder.

9.1 Relevance and Expressiveness

Practical relevance of the pattern-based approach to multidimensional data analysis is demonstrated by its application in the agriProKnow cooperative research project for building a data warehouse for decision support in precision dairy farming [4]. In the agriProKnow project, OLAP patterns were originally introduced to be able to deal with uncertainties regarding the stakeholders' requirements with respect to OLAP queries to be answered by the data warehouse. During requirements elicitation, when it came to identifying interesting business questions we faced stakeholders who were vague regarding the required queries that would have to be composed eventually, partly due to uncertainties concerning the data warehouse's still evolving multidimensional model. The multidimensional model ultimately comprised ten cubes and sixteen dimensions and was enriched by 108 predicates, groupings and orderings as well as 89 calculated measures. Instead of deferring the composition of a vast amount of queries towards the end of the project, which had a rather strict deadline set by the funding agency, requirements from experience in previous research projects, e.g., the Semantic Cockpit project [5] and research that led to the development of the BIRD reference modeling approach [6], [7], informed the development of generic query composition facilities. Looking at the results of previous projects led to the identification and specification of common domain-independent OLAP patterns that express query composition solutions for specific types of information needs [2], [4]. While the experience gathered in previous research

projects was domain-specific, it allowed to obtain domain-independent OLAP patterns. Five domain-independent OLAP patterns could be identified and defined. The thus identified OLAP patterns can be divided into two groups (see details in Appendix B). The group of basic patterns comprises the non-comparative analysis pattern, which represents generalized multidimensional aggregation queries where no comparison is performed. The group of comparative patterns, on the other hand, comprises the homogeneous subset-baseset, the homogeneous subset-subset, and the heterogeneous subset-subset comparison patterns [2]. The term "homogeneous" refers to the fact that those patterns take into account only one cube, whereas "heterogeneous" indicates that two separate cubes, possibly with different schemas, are taken into account. Ultimately, the pattern-based approach turned out to be successful in the agriProKnow project, as the required queries to answer the 43 analysis questions (information needs) could be expressed using previously identified OLAP patterns. It should be noted that more complex analysis questions required the successive application of patterns to obtain the necessary input data.

Expressiveness of the pattern-based approach is demonstrated by the fact that this relatively small set of identified patterns covered most of the information needs that arose in the course of the agriProKnow project. It was possible to satisfy the arising information needs promptly with little effort by using the OLAP patterns to query the data warehouse (see [1] for further information). The homogeneous subset-complement comparison pattern had only to be defined and added to the set of already identified OLAP patterns to cover the remaining information needs (see details in Appendix B).

The experience gained in previous research projects led to the definition of domain-independent patterns. In this regard, the information needs that arose in the previous research projects can be regarded as a sort of "training set" of domain-independent patterns. The domain-specific information needs that arose in the course of the agriProKnow project, in turn, can be viewed as the "test set", which allowed for the evaluation of the applicability of domain-independent patterns to a specific domain. Since most of the encountered domain-specific information needs that arose in the agriProKnow project could be satisfied using the small set of previously identified domain-independent OLAP patterns, we conclude that the OLAP pattern approach is both relevant and expressive while also applicable across domains.

9.2 Quality Assessment as a Domain-Specific Language

We evaluate the ensemble of eMDM, pattern definition and usage language as well as template macros as a domain-specific language (DSL) according to the Framework for Qualitative Assessment of DSLs (FQAD) [91]. The FQAD considers the perspective of a stakeholder, i.e., evaluator, by allowing for the evaluator to map their evaluation goal

to each fundamental quality characteristic (QC) of DSLs and its sub-characteristics, i.e., assign an importance level to each QC and a minimum required support level to each sub-characteristic. We evaluate the DSL from the perspective of a language developer, i.e., the goal is to assess whether or not the specified language is completely and consistently specified with regard to the necessary functionality. Assessment of the DSL's support for each QC then amounts to checking whether the DSL supports the QC's sub-characteristics. The DSL must achieve the determined minimum support level for each sub-characteristic. The FQAD defines a set of QCs and sub-characteristics, which serve as the basis for evaluation of the OLAP pattern approach. Tables 9.1 and 9.2 give an overview of evaluation results for each QC and its sub-characteristics.

Table 9.1: Evaluation of the OLAP pattern approach according to the FQAD [91]

#	DSL Quality (Sub-)Characteristic	Importance Level	Min. Required Support Level	Assessed Support Level	Success Level
1	Functional Suitability	Mandatory			Satisfactory
1.1	Completeness		Strong support	Strong support	Satisfactory
1.2	Appropriateness		Strong support	Strong support	Satisfactory
2	Usability	Nice to have			Effective
2.1	Comprehensibility		No support	Not assessed	Satisfactory
2.2	Learnability		No support	Some support	Effective
2.3	Number of activities for task achievement		No support	Strong support	Effective
2.4	Likeability, perception		No support	Not assessed	Satisfactory
2.5	Operability		No support	Full support	Effective
2.6	Attractiveness		No support	Not assessed	Satisfactory
2.7	Compactness		No support	Some support	Effective
3	Reliability	Mandatory			Satisfactory
3.1	Model checking		Strong support	Strong support	Satisfactory
3.2	Correctness		Strong support	Strong support	Satisfactory
4	Maintainability	Desirable			Satisfactory
4.1	Modifiability		Some support	Some support	Satisfactory
4.2	Low coupling		Some support	Some support	Satisfactory
5	Productivity	Mandatory			Effective
5.1	Development time		Strong support	Full support	Effective
5.2	Amount of human resource		Strong support	Strong support	Satisfactory

Table 9.2: Evaluation of the OLAP pattern approach according to the FQAD [91] (continued from Table 9.1)

#	DSL Quality (Sub-)Characteristic	Importance Level	Min. Required Support Level	Assessed Support Level	Success Level
6	Extensibility	Nice to have			Satisfactory
6.1	Mechanisms for users to add new features		No support	No support	Satisfactory
7	Compatibility	Mandatory			Effective
7.1	Domain compatibility		Strong support	Strong support	Satisfactory
7.2	Development process compatibility		Strong support	Full support	Effective
8	Expressiveness	Mandatory			Effective
8.1	Mind to programming mapping		Strong support	Strong support	Satisfactory
8.2	Uniqueness		Strong support	Full support	Effective
8.3	Orthogonality		Strong support	Full support	Effective
8.4	Correspondence to important domain concepts		Strong support	Full support	Effective
8.5	No conflicting elements		Strong support	Full support	Effective
8.6	Right abstraction level		Strong support	Full support	Effective
9	Reusability	Nice to have			Satisfactory
9.1	Reusability		No support	No support	Satisfactory
10	Integrability	Mandatory			Satisfactory
10.1	Integrability		Strong support	Strong support	Satisfactory
	Overall Success				**Satisfactory**

9.2.1 Determination of Importance Levels

The first step in DSL evaluation according to the FQAD is the assignment of an importance level to each QC, which reflects the evaluation goal. The assignment of one of three importance levels – `Mandatory`, `Desirable`, and `Nice to Have` – to each QC constitutes the *evaluator's profile*. The *Importance Level* column in Tables 9.1 and 9.2 shows the importance level for each QC in the context of the OLAP pattern approach, which we discuss in the following.

Mandatory QCs From our evaluator's perspective functional suitability, reliability, productivity, compatibility, expressiveness, and integrability are classified as *mandatory QCs*. Functional suitability is considered mandatory because it refers to the degree to which the language provides all the required functionality, i.e., the degree to which OLAP patterns can be defined and used. Reliability refers to the degree to which OLAP patterns support reliable solutions, that is, the execution of a pattern shall yield a query that meets its specification [92]. Reliability is considered mandatory since the OLAP patterns should help avoid erroneous and ambiguous definitions of recurring types of queries. Productivity is considered mandatory because the goal of the OLAP pattern approach is to reduce the effort required to satisfy BI user's information needs by composing an appropriate OLAP query. Compatibility, in this context, refers to *process* compatibility, i.e., the degree to which the OLAP patterns "can be used in the domain and the development process" [91]. Since OLAP patterns should be used during development of BI and analytics solutions to ease the composition of necessary OLAP queries, compatibility is considered mandatory. Expressiveness is considered mandatory as well because it refers to the degree to which a problem-solving strategy can be naturally represented by a language. Expressiveness is particularly important for OLAP patterns, which represent problem-solving solutions for certain types of information needs. Finally, integrability is considered mandatory: Real-world data warehouse systems employ a variety of logical data models and query languages. OLAP patterns must, therefore, be flexible regarding implementation variants and target languages.

Desirable QCs Maintainability is considered desirable from a language developers perspective, however, it is not necessary for achieving the goal of providing the required functionality.

Nice-to-have QCs In addition to the mandatory DSL-specific QCs, we consider usability, extensibility, and reusability as *nice-to-have QCs* from our evaluator's perspective. Although the usability of a language cannot be neglected, we consider it as nice to have, as we

focus mainly on functional aspects in this work, i.e., on pattern definition and usage. We acknowledge the general importance of usability for the OLAP pattern approach but, at this point, we do not prioritize usability since it depends considerably on graphical notation and user interfaces, which are not the focus of our evaluator's goal. It should be noted that from a DSL user's perspective usability would be considered as mandatory. We do not expect that users require extension mechanisms to define new language features since the language supports the necessary steps for pattern definition and usage. Lastly, we do not expect the pattern definition and usage languages as a starting point for the development of a new language, i.e., reusability is considered nice to have.

The evaluator's profile shown in the *Importance Level* column in Tables 9.1 and 9.2 represents the importance of the QCs for a language developer. That is, it reflects the importance of the QCs with respect to the goal of assessing the completeness and consistency of the specified DSL with respect to the required functionality. It should be noted that this importance classification may differ depending on the stakeholder under consideration [91].

9.2.2 Determining the Support Levels

The second step is to assess the support levels for the sub-characteristics of each DSL-specific QC. For each sub-characteristic the *minimum required level of support* has to be determined and the *actual support level* for the DSL has to be assessed. The support level is one of the following [91]: `No Support`, `Some Support`, `Strong Support`, and `Full Support` is provided by the language. We extend the set of predefined support levels with `Not Assessed`, which for all practical purposes corresponds to `No Support`, except that it allows us to state that we have not assessed a particular quality sub-characteristic or that it cannot be assessed at the moment and hence we cannot state whether it is supported or not – this is particularly relevant for sub-characteristics relating to usability.

The evaluator's profile determines the minimum required support level for each sub-characteristic, i.e., `Mandatory` QCs require at least `Strong Support`, `Desirable` QCs require at least `Some Support`, and `Nice to Have` QCs require at least a `No Support` or, in our case, `Not Assessed`. The results of assessing the support levels are obtained by taking into account the textual syntax, the definitions of the OLAP pattern approach, and to some extent the graphical notation.

Functional Suitability Completeness is considered `Strong Support` since the OLAP pattern approach provides all the language constructs required to employ the pattern-based approach to multidimensional data analysis (QC 1.1). This covers constructs to define and represent multidimensional models that are enriched by business terms. In addition, construct are provided to define and use patterns in the context of an associated eMDM. It

should be noted that more sophisticated concepts of multidimensional data analysis may be relevant in other contexts. Since the proposed OLAP pattern approach in its current form considers only the most important concepts, *Full Support* is not claimed. In addition, appropriateness is considered `Strong Support`, adopting the appropriateness perspectives described by Krogstie [93] (QC 1.2). First, domain appropriateness considers whether the DSL is expressive enough to represent everything relevant in the domain, while avoiding the expression of anything unrelated to the domain [93]. This is especially of relevance as it concerns the quality to externalize the domain knowledge of users in order to be interpreted by other users of the same domain [93]. Domain appropriateness is fulfilled since the approach takes into account only the necessary structural and behavioral aspects of multidimensional models, business terms, and patterns. Second, appropriateness regarding comprehensibility concerns the social user interpretation, i.e., the statements "the audience thinks an externalized model consist of" [93]. This affects the likelihood of errors that can occur when different users read models designed by other different users [93]. We claim that the proposed DSL satisfies comprehensibility appropriateness by avoiding construct redundancy, in that all concepts are defined only once and can be expressed by exactly one language statement and graphical representation, respectively. Comprehensibility appropriateness is further promoted as only a few language constructs and statements are needed to define a potentially high number of OLAP patterns. Furthermore, the DSL allows defined OLAP patterns to vary greatly in their level of detail and flexibility, as there are no restrictions on the pattern variables that can be defined and the built-in features of the query languages used in the pattern templates. Additionally, Krogstie identifies the appropriateness regarding the BI user's language knowledge as relevant, as it describes how close the conceptual representation of the DSL is to the user's perception of reality [93]. The introduced DSL supports BI user language knowledge appropriateness as parts of it are based on common concepts in the domain of data warehousing (see alignment with the DFM [25]), hence, we expect a reasonable learning curve for language users who are familiar with basic OLAP and multidimensional modeling constructs. The BI users language knowledge appropriateness is also promoted as we avoid unnecessary diversity in our DSL that could confuse an inexperienced user; business terms and patterns are similar in their structure, i.e., they consist of some kind of variables, constraints, and templates, and in their use, i.e., variables are being substituted while templates are being preprocessed. Finally, appropriateness regarding technical actor interpretation is fulfilled, as all DSL language constructs are rigorously defined and further detailed by natural language descriptions of their semantics and corresponding examples; this also includes the description of the pattern execution process to obtain executable OLAP queries. It should be noted that Krogstie also mentions appropriateness in terms of the externalizability of knowledge [93] to be relevant,

but we do not consider it here as it primarily concerns usability.

Usability The avoidance of construct redundancy create the necessary conditions to fulfil comprehensibility appropriateness, but the specific effort required to understand the language in terms of comprehensibility (QC 2.1) is not assessed, as this is largely dependent on user interfaces with extensive graphical support. Likewise, the ability to learn and remember the concepts and symbols [91] of the DSL is not assessed, although we provide the necessary foundations to fulfill the BI user's language knowledge appropriateness. However, due to the limited number of concepts, statements, and symbols, and through the reuse of the symbols in the eMDM and pattern notation, we claim that there is some support for the ability to learn the DSL (QC 2.2). Similarly, likeability (QC 2.4), i.e., the ability of users to recognize that the DSL is suitable for their needs, and attractiveness (QC 2.6), i.e., the appealing appearance of symbols, are not assessed as they depend on subjective user interaction with a graphical interface. Nevertheless, our graphic notation for the DSL constructs aims to be easy to perceived, likeable and attractive. To this end, the design principles for graphical notations introduced by Moody [83] are taken into account to achieve a cognitively effective graphical representation (see details in Section 4.3 and Section 5.2). In addition, we adopt widely used and popular graphical notations as the basis for the DSL constructs, i.e., the graphical representation of multidimensional models is based on the DFM notation [25], while the representation of patterns follows the common table- and section-oriented pattern style [33]. Furthermore, the DSL provides a strong support of the activities necessary to achieve the task (QC 2.3) [91], i.e., the minimum number of steps to define a pattern, put it into use, and to organize it are supported. For this purpose, the language provides the necessary language constructs and statements to define eMDMs and patterns (Figure 2.2), to use patterns in the context of an associated eMDM (Figure 2.3), and to organize eMDM elements and patterns (Figure 7.4. The DSL also fully supports operability (QC 2.5) by providing language statements that making it easy to operate and control the language [91], i.e., a concise grammar is provided defining the supported statements to be formulated. Finally, the DSL provides some support to grouping eMDM elements and patterns at different levels of abstraction (see Chapter 7) so that a compact representation is possible (QC 2.7).

Reliability Correctness is considered to be supported strongly since possible errors are avoided by providing strict syntax rules and rigorous definitions (QC 3.2). Furthermore, constraints and derivation rules allow for checking the applicability of a pattern with respect to an eMDM, thus avoiding the generation of non-executable OLAP queries (QC 3.1). To this end, we follow the design-by-contract metaphor [28], which, according to Paige et al. [92], allows to support reliability, i.e., the pattern author defines the context, which can

be seen as a contract that specifies the conditions that have to be satisfied to obtain an executeable OLAP query, while a pattern user can expect to obtain an executable OLAP query by instantiating the pattern in such a way that it satisfies the constraints of the pattern. Consequently, the proposed OLAP pattern language promotes correct and valid definition and usage of patterns. It should be noted that full support for model checking and correctness is not claimed, as the pattern author must guarantee that the pattern templates are formulated to follow the described solution; the proposed OLAP pattern approach cannot check this in advance, since the query language used is considered to be text only and thus its semantics cannot be checked.

Maintainability The DSL supports modifiability (QC 4.1) to some extent. Due to the well-structured and concise formalizations and language definitions, it is easy to determine where changes need to be made and where extensions can be made, respectively. Similarly, the DSL only provides some support for low coupling (QC 4.2), i.e., it depends on the changes and extensions made how much they affect other DSL components. Adding additional functionality to patterns, for example, does not affect the underlying eMDM formalizations, but if changes are made to the underlying eMDM formalizations it may affect the patterns.

Productivity The idea to increase the *productivity* by reducing the effort to compose an OLAP query is inherent to OLAP patterns. Although the definition of an OLAP pattern is a time-consuming task for the pattern author, patterns can considerably reduce the composition time of OLAP queries for numerous different pattern users later on. More importantly, the effort to define a pattern must be invested only once, while individual OLAP queries are composed many times, resulting in potentially huge savings of effort (QC 5.1). Furthermore, users of OLAP patterns only need to consider the conceptual level to obtain executable queries, reducing the knowledge and effort required to obtain the desired OLAP query. This, in turn, could reduce the need to consult pattern authors since BI users can consult the pattern catalog (QC 5.2). In the agriProKnow project, we found that the pattern users, i.e., domain experts, were able to express the desired OLAP query in terms of pattern usage.

Extensibility The DSL does not provide mechanisms for pattern authors to add new functionality, as we assume that the DSL supports all the necessary functionality for defining, using, and organizing patterns. However, from a pattern user's perspective, the defined corpus of patterns constitutes the DSL, i.e., the defined patterns specify the vocabulary available to pattern users in analysis situations. Therefore, pattern authors and user,

respectively, can easily extend their DSL by defining new patterns and instantiating existing patterns.

Compatibility The proposed DSL is compatible with the data warehouse domain as it can be used for a data warehouse system as long as a conceptual representation of the data warehouse is available, i.e., an eMDM exists (QC 7.1). It should be noted that the DSL can be used irrespective of the data warehouse's logical realization, i.e., star/snowflake schema or multidimensional array, since it operates primarily on the conceptual level (assuming necessary mapping information is defined (see Section 6.2)). In addition, domain compatibility is promoted by allowing the vocabulary used in the analysis to be conceptualised by business terms and by allowing the OLAP query composition solution for specific types of information needs to be conceptualised by OLAP patterns. The proposed OLAP pattern language is also compatible with the development processes of data warehouses and OLAP queries (QC 7.2). In the course of designing a data warehouse, for example, a multidimensional model is created, which can be enriched with business terms and considered in future pattern definitions. Additionally, existing OLAP patterns can be integrated into the process of composing (ad hoc) OLAP queries as well as into the process of eliciting requirements of OLAP queries to be composed.

Expressiveness The OLAP approach has `Strong Support` for *expressiveness*, which is one of the most important QCs of DSLs [91] since pitfalls that can occur when designing DSLs have been taken into account and avoided [94]. First, only relevant concepts regarding the definition and usage of patterns in the data warehousing domain are considered, i.e., the definitions are concise, clean, and avoid clutter (QC 8.4). Second, each concept is defined exactly once (QC 8.2) and only one representation is available for each of them in the corresponding syntax and graphical notation (QC 8.3). Note that although patterns and business terms share some design principles, they are uniquely and clearly defined. Third, the DSL allows logical aspects to be taken into account, despite the fact that it is defined at the conceptual level. Thus, the chosen level of abstraction is adequate, since it allows to consider both conceptual and logical aspects in order to generate executable OLAP queries (QC 8.6). Fourth, the definitions are coherent and consistent, i.e., there are no contradictions and conflicting concepts (QC 8.5). Lastly, the DSL supports the mapping of an OLAP query pattern composition solution, since statements considering both the conceptual as well as the logical level can be formulated (QC 8.1).

Reusability The DSL does not support reusability in the sense of full or partial reuse in another DSL. However, this does not exclude that individual DSL constructs, parts of the grammar, or elements of the graphical notation could be reused directly or modified

in other DSLs. For example, the DSL for solution patterns for machine learning [14], [15] presented by Nalchigar and Yu could incorporate the OLAP pattern approach in their data preparation view.

Integrability Finally, the language has `Strong Support` for *integrability* of different data models, logical realization variants, and different host-specific languages by defining corresponding templates (QC 10.1). Furthermore, integrability can be fostered by formulating the templates using generic OLAP query languages that are independent of a specific realization [95], [96].

9.2.3 Determining the Success Levels

The third step according to the FQAD is to determine the success level for each QC and its sub-characteristics. The level of success of the sub-characteristics can be determined by considering the minimum required and assessed support level. The available success levels `Incomplete`, `Satisfactory`, and `Effective` are assigned to each sub-characteristic as follows: a sub-characteristic is `Incomplete` if its assessed support level does not meet the minimum required support level, a sub-characteristic is `Satisfactory` if its support level exactly matches the minimum required support level, and a sub-characteristic is `Effective` if its support level exceeds the minimum required support level [91]. The success level of the sub-characteristics impacts the success level of the corresponding QC as follows: the success level of the corresponding characteristic is `Incomplete`, if there exists one sub-characteristic with an `Incomplete` success level, is `Satisfactory`, if all sub-characteristics have an `Satisfactory` success level, and `Effective`, if there exists only sub-characteristics with at least `Satisfactory` success level and with at least one sub-characteristic with a `Effective` success level (see column *Success Level* in Table 9.1 and Table 9.2). Finally, the overall success can be determined by considering the minimum success level amongst all the quality characteristics. The language evaluation can be determined successful if all QCs are considered at least as `Satisfactory` [91].

Looking at the column *Success Level* in Tables 9.1 and 9.2, we conclude that for each QC a success level could be determined that is better or equal to `Satisfactory`. Thus, we conclude that the presented DSL is successful from the perspective of a language developer, i.e., it is completely and consistently specified with respect to the required functionality to define, use, and organize patterns.

The result of this quality assessment is only meaningful with regard to the perspective of a language developer, but not with respect to other stakeholders. Kahraman and Bilgen identify, in addition to the DSL users who apply the DSL to obtain executable models, other stakeholders, i.e., managers, domain experts and DSL implementers, whose perspective

could also be taken into account [91]. *Managers* focus on the organizational and process aspects of a DSL and whether it can lead to the development of new products, i.e., whether or not it is worth investing in the DSL development [91]. In contrast, *domain experts* define the functional and non-functional requirements based on the existing domain knowledge [91], while *DSL implementers* are responsible for realising the semantics of the DSL, i.e., implementing compilers and associated libraries [91]. We have focused on the language developer's perspective, as the aim of this thesis is to provide a completely and consistently defined DSL supporting the pattern-based approach to data analysis.

Chapter 10

Extensions

In this chapter, we outline how the core pattern-based approach can be extended in multiple directions to further increase the expressiveness and versatility of OLAP patterns. We describe each possible extension informally using exemplified language statements without providing a detailed formalization. We consider each of these extensions independently, without reference to the extensions already described. These extensions are orthogonal to the essence of the pattern-based approach presented in this thesis.

10.1 Composite Types

We presented the core pattern-based approach with atomic variables for parameters and derived elements. Consequently, exactly one name can be bound to such parameters and derived elements. For example, a pattern with only two business term parameters is limited to information needs that require the specification of exactly two business terms. However, this limits the generic nature of such a pattern, as it can impede a user from using it if in order to answer its business question more than two business terms that need to be provided. The core pattern-based approach can circumvent that problem to some extend by creating several "versions" of existing patterns with different sets of parameters – at the cost of clarity.

In addition to constructors for atomic types, we can introduce constructors for composite types and combinations thereof. To this end, the composite types arrays, maps, and tuples are introduced. Arrays and maps are used to represents collections; arrays are indexed by integers, while maps consist of key-value pairs. In contrast, tuples are used to relate values bound to array, map, and atomic variables; tuples are represented as ordered lists of values containing either two values (binary) or three values (ternary).

```
1  CREATE PATTERN "Happy Milk Farm"/"Dairy Catalog"/"Breed-Specific
       Subset-Subset Comparison" WITH
2     PARAMETERS
3        <sourceCube>:CUBE;
4        ...
5        <compDimRole>[]:DIMENSION_ROLE;
6        <iDimSlice>[]:UNARY_DIMENSION_PREDICATE;
7        <iDimSlice2CompDim>()[]:BINARY_TUPLE;
8        ...
9     END PARAMETERS;
10
11    DERIVED ELEMENTS
12       <compDim>*:DIMENSION
13          FOR <r> IN <compDimRole>[] WITH
14             <compDim>*[<r>] <= <sourceCube>.<r>;
15       ...
16    END DERIVED ELEMENTS;
17    ...
18    CONSTRAINTS
19       FOR <r> IN <compDimRole>[] WITH
20          <sourceCube> HAS DIMENSION_ROLE <r>;
21
22       FOR (<i>,<j>) IN <iDimSlice2CompDim>()[] WITH
23          <i> IN <iDimSlice>[],
24          <j> IN <compDim>*,
25          <i> IS_APPLICABLE_TO <j>;
26       ...
27    END CONSTRAINTS;
28 END PATTERN;
```

Listing 10.1: Extract from the redefinition of breed-specific subset-subset comparison using composite type constructors for parameters and derived elements.

Example 10.1 (Pattern definition with composite type constructors)**.** The extract of the redefinition of the breed-specific subset-subset comparison in Listing 10.1 contains parameters and derived elements defined via constructors of composite types. The ⟨compDimRole⟩ parameter is defined as an array of dimension roles (Line 5), the ⟨iDimSlice⟩ parameter is defined as an array of unary dimension predicates (Line 6), and the ⟨iDimSlice2CompDim⟩ parameter is defined as an array of binary tuples (Line 7), while the ⟨compDim⟩ derived element is defined as a map with dimensions as values (Line 12). ♢

With the extension of type constructors to arrays, maps, and tuples, we also introduce extensions of constraint and derivation expressions. We introduce the definition of constraints

and derivation rules over a range of values, i.e., constraints and derivation rules can contain array or map variables, or address columns of tuple variables. In addition, we introduce the definition of constraints for tuple columns, that is, for tuple variables, possible values to be bound to specific tuple columns can be restricted to values bound to specific variables. Furthermore, the meaning of the relationship between the column values of a tuple can be expressed by defining which constraints are to be considered between them.

Example 10.2 (Constraints and derivation rules with composite variables)**.** In addition to the definition of composite parameters, Listing 10.1 contains constraints and derivation rules defined over the range of composite variables or references to columns of tuple variables. For the derived element ⟨compDim⟩∗ the derivation rule for each dimension to be derived is defined by iterating over the parameter ⟨compDimRole⟩[ ]; for each iteration the current dimension role is assigned as the property of the corresponding derivation rule (Line 14. It should be noted that the derived value can be access by accessing ⟨compDim⟩∗ using the dimension role as a key. Note that the derived value can be accessed by accessing ⟨compDim⟩∗ using the dimension role as a key. The relationship between the ⟨sourceCube⟩ cube parameter and the ⟨compDimRole⟩[ ] parameter is again defined by iteration of the ⟨compDimRole⟩[ ] parameter; per iteration the current dimension role is defined as a dimension role that has to exists for the cube bound to ⟨sourceCube⟩. For the binary tuple array parameter ⟨iDimSlice2CompDim⟩()[ ] multiple constraints are defined with respect to the columns of the tuples to be bound as follows: Each value bound to the first column of a tuple must be among the values bound to the parameter ⟨iDimSlice⟩[ ], each value bound to the second column of a tuple must be among the values bound to the parameter ⟨compDim⟩∗, and each value bound to the first column must be applicable to the value bound to the corresponding second column; note that the first column value is restricted to a unary dimension predicate, while the second column value is restricted to a dimension. ♢

Finally, we extend the available set of (first-level) macros with *second-level macros*, which allow to operate on complex variables in pattern templates (see Table 10.1). Second-level macros support the generation of expressions, which in turn can contain other first- and second-level macros. Consequently, second-level macros must be processed before first-level macros. We distinguish second-level macros from first-level macros using the $$ prefix.

Macro	Description
$$for	$$for($$var IN ⟨array⟩[ ]){ ⟨body_exp⟩ } iterates entries of an array (⟨array⟩[ ]) and generates for each entry ($$var) the expression defined in its body. The body, enclosed by curly brackets, can contain constants, other macros, and access the $$var variable.
$$if	$$if(⟨cond⟩){ ⟨body_exp⟩ } evaluates a specified condition (⟨cond⟩) whether it is true or false. If the condition is fulfilled the expression defined in its body is generated. The body is enclosed by curly brackets can contain constants and other macros.
$$last	$$last(⟨entry⟩, ⟨array⟩[ ]) returns true if the entry is the last array entry, otherwise, false.
$$not	$$not(⟨cond⟩) returns true if the condition returns false, otherwise, false.

Table 10.1: Second level macros for handling composite variables in templates

```
1  ...
2  interestCube AS (
3    SELECT $expr(<groupCond>, jd),
4           $expr(<cubeMeasure>, bc) AS <cubeMeasure>
5    FROM   baseCube bc
6           JOIN <joinDim> jd ON
7              bc.<joinDimRole>=jd.$dimKey(<joinDim>)
8           $$for( $$r IN <compDimRole>[] ) {
9           JOIN <compDim>*[$$r] cd_$$r ON
10             bc.$$r=cd_$$r.$dimKey(<compDim>*[$$r])
11          }
12   WHERE  $$for( ($$i,$$j) IN <iDimSlice2CompDim>()[] ) {
13             $expr($$i, $$j)
14             $$if( $$not($$last( ($$i,$$j),
                   <iDimSlice2CompDim>()[] ) ) ) { AND }
15          }
16   GROUP BY $expr(<groupCond>, jd)
17 ),
18 ...
```

Listing 10.2: Extract from a template of the redefined breed-specific subset-subset comparison pattern.

Example 10.3 (Pattern template with second level macros)**.** Listing 10.2 depicts an extract of a template added to the pattern defined in Listing 10.1. The names bound to ⟨compDimRole⟩[ ] are iterated using the $$r variable (Lines 8-11). A join condition expression

is generated for each dimension role bound to $$r as follows: the dimension referenced by the dimension role $$r is obtained by accessing the ⟨compDim⟩∗ using the $$r as the key, the received dimension is provided with an alias consisting of cd_ and $$r (Line 9), then base cube bc is joined via the dimension role $$r with the dimension via its dimension key (Line 10). In addition, for each tuple in ⟨iDimSlice2CompDim⟩[ ] a selection expression is generated (Lines 12-15), consisting of an $expr macro call that contains the first value of the tuple as its first argument and the second value of the tuple as its second argument (Line 13); AND is added to the expression as long as another tuple to be iterated exists (Line 14). ♢

With variables defined via constructors for composite types, patterns can be defined without limiting the number of bindings to variables of a certain type to exactly one. This can significantly increase the genericity of patterns, as it allows a large number of similar information needs of a particular type of information need to be met. Furthermore, already instantiated patterns can also be refined by binding additional names to already bound parameters.

```
1 INSTANTIATE PATTERN "agriProKnow"/"agriProKnow
     Patterns"/"Breed-Specific Subset-Subset Comparison" AS "Happy
     Milk"/"Dairy OLAP Patterns"/"Breed-Specific Subset-Subset
     Milking-Comparison" WITH BINDINGS
2    <sourceCube> = "Milking",
3    ...
4    <compDimRole>[] = { "Milking Time", "Farm" },
5    ...
6    <iDimSlice2CompDim>()[] = {
7                              ("2020-11-05", <compDim>*["Milking Time"]),
8                              ("Linz", <compDim>*["Farm"]) }
9 END BINDINGS;
```

Listing 10.3: Detail of a pattern instantiation with collection variables

Example 10.4 (Instantiation of a pattern with composite parameters)**.** Listing 10.3 represents an extract from an instantiation of the pattern defined in Listing 10.1. A single name is bound to the atomic parameter ⟨sourceCube⟩ (Line 2). An ordered set of names consisting of the two names Milking Time and Farm is bound to the parameter ⟨compDimRole⟩[ ]. Finally, an ordered set of binary tuples consisting of two tuples is bound to the parameter ⟨iDimSlice2CompDim⟩[ ] (Line 6; the first tuple consists of the unary predicate name 2020-11-05 and the dimension referenced by the dimension role named Milking Time Line 7,

while the second tuple consists of the unary predicate name `Linz` and the dimension referenced by the dimension role named `Farm` Line 8. ♢

10.2 Optional Variables

The presented pattern-based core approach is based on mandatory variables, which is why exactly one name must always be linked to corresponding parameters and derived elements. However, the obligation to provide a value for each variable affects the generic nature of patterns. Consider, for example, a pattern with a mandatory parameter representing a unary cube predicate that allows to restrict business events captured by the specified source cube. However, if a user with an information need of the type covered by the pattern does not need to restrict the source cube to answer the business question, the pattern cannot be used. This problem can also be circumvent to some extend by creating several "versions" of existing patterns with different sets of parameters.

We extend the pattern-based core approach with optional variables to overcome this disadvantage. Optional variables allow parameters to be specified as optional, allowing users to omit name bindings to parameters that are unnecessary to satisfy a particular information need. Accordingly, derived elements that contain optional parameters in the corresponding derivation rules are also defined as optional. It should be noted that derivation rules as well as constraints with optional unbound variables are not considered during execution.

```
1  CREATE PATTERN "Happy Milk Farm"/"Dairy Catalog"/"Breed-Specific
       Subset-Subset Comparison" WITH
2     PARAMETERS
3        <sourceCube>:CUBE;
4        <baseCubeSlice>:UNARY_CUBE_PREDICATE IS_OPTIONAL;
5        ...
6     END PARAMETERS;
7     ...
8     CONSTRAINTS
9        <baseCubeSlice> IS_APPLICABLE_TO <sourceCube>;
10       ...
11    END CONSTRAINTS;
12 END PATTERN;
```

Listing 10.4: Extract from the redefinition of breed-specific subset-subset comparison with optional parameters.

Example 10.5 (Pattern definition with optional variables)**.** Listing 10.1 shows an extract of the redefinition of the breed-specific subset-subset comparison containing an optional

parameter. The ⟨baseCubeSlice⟩ parameter is defined as optional (Line 4). It should be noted that the applicable-to constraint that defines that the business term bound to the parameter ⟨baseCubeSlice⟩ parameter to be applicable to the cube bound to the parameter ⟨sourceCube⟩ is only considered (Listing 9) if the ⟨baseCubeSlice⟩ parameter is bound. ♢

Macro	Description
$$empty	$$empty(⟨var⟩) returns true if the variable ⟨var⟩ is unbound, false if not.
$$if	$$if(⟨cond⟩){ ⟨body_exp⟩ } evaluates a specified condition (⟨cond⟩) whether it is true or false. If the condition is fulfilled the expression defined in its body is generated. The body is enclosed by curly brackets can contain constants and other macros.

Table 10.2: Second-level macro for handling optional variables in templates

Finally, we introduce second level macros, which allow to consider unbound optional variable in pattern temples (see Table 10.2). Second-level macros are prefixed by $$ and must be processed before other first-level macros.

```
1  ...
2  WITH baseCube AS (
3    SELECT *
4    FROM   <sourceCube> sc
5           JOIN <baseDim> a ON
6              sc."Cattle"=a.$dimKey(<baseDim>)
7    WHERE  $$if( $$empty(<baseCubeSlice>) ) {
8              $expr(<baseCubeSlice>, sc) AND
9           }
10          $expr(<cattleBreed>, a)
11 )
12 ...
```

Listing 10.5: Extract from a template of the pattern defined in Listing 10.4.

Example 10.6 (Pattern template with optional variables)**.** Listing 10.5 depicts an extract of a template added to the pattern defined in Listing 10.4. The second level macros are used to check whether the ⟨baseCubeSlice⟩ is unbound or not (Line 7, if it is bound the first level macro call in Line 8 is considered as part of the pattern's template. ♢

By defining patterns with optional variables, the genericity of patterns can be significantly increased. Users with information needs that correspond to the type of information need covered by the pattern, but with fewer restrictions necessary, for example, can thus be supported.

10.3 Generic Business Terms

In the presented approach, new business terms have to be defined from scratch if necessary functionality is not provided by existing terms. Although the definition of new business terms is supported, it requires a certain cognitive effort and knowledge of the conceptual model and the respective query language. Users who lack this knowledge will have difficulty defining the business terms they need. In such a case, the application of a pattern may also be prevented. In the core pattern approach, the problem can be avoided to a certain extent by defining a comprehensive vocabulary. Users, however, will still have to define missing business terms themselves if a business term is not avaialable.

In addition to the already presented (specific) business terms, we introduce generic business terms. Generic business terms can reduce the need to define business terms from scratch, as they can be defined once and specialized several times through instantiation, resulting in different (specific) business terms. To this end, generic business terms provide – in addition to context parameters – parameters that enable the adaptation of constraints and expression templates, i.e., constraint and template parameters. Constraint parameters are used to refer to multidimensional model elements and value sets that are used to adapt the constraints of a business term accordingly. This allows the dynamic specification of elements of type, property, and domain constraints that refer to multidimensional model elements and value sets; however, elements of constraints that refer to types of multidimensional elements cannot be specified dynamically. In contrast, template parameters are used to customise expression templates of a business term by referencing values of a certain value set, i.e. primitive values; template parameters are not used to adapt constraints. It should be noted that constraint parameters can also be used to customise the expression templates of a business term.

Example 10.7. (Generic Business Term) The generic business term `Generic Aggregated Measure` (Listing 10.6, Lines 1-17) represents a generic unary calculated measure that applies an aggregation operation to a measure. The constraint parameter ⟨`cubeMeasure`⟩ (Line 3) allows to specify the cube measure (Line 12) to be aggregated, while the constraint parameter ⟨`measureDom`⟩ (Line 4) allows to specify the domain (Line 13) of the aggregated measure; the domain also defines the return type of the unary calculated measure (Line 16). The template parameter ⟨`aggregation`⟩ (Line 8) allows to specify the aggregation operation

to be performed. The specified aggregation operation must be a value from the string value set AGGREGATION_OPERATION in order to be used to adapt the expression template of the term (Lines 19-23). Besides the templates parameter ⟨aggregation⟩, the expression template also contains the constraint parameter ⟨cubeMeasure⟩. ♢

```
1  CREATE UNARY_CALCULATED_MEASURE "Generic Repository"/"Generic
       Business Terms"/"Generic Aggregated Measure" WITH
2     CONSTRAINT PARAMETERS
3        <cubeMeasure>:MEASURE;
4        <measureDom>:NUMBER_VALUE_SET;
5     END CONSTRAINT PARAMETERS;
6
7     TEMPLATE PARAMETERS
8        <aggregation>:AGGREGATION_OPERATION;
9     END TEMPLATE PARAMETERS;
10
11    CONSTRAINTS
12       <ctx> HAS MEASURE <cubeMeasure>;
13       <ctx>.<cubeMeasure>:<measureDom>;
14    END CONSTRAINTS;
15
16    RETURNS <measureDom>;
17 END UNARY_CALCULATED_MEASURE;
18
19 CREATE TERM TEMPLATE FOR "Generic Repository"/"Generic Business
       Terms"/"Generic Aggregated Measure" WITH
20    LANGUAGE = "SQL";
21    DIALECT = "ORACLEv11";
22    EXPRESSION = ""<aggregation>"("<ctx>"."<cubeMeasure>")";
23 END TERM TEMPLATE;
```

Listing 10.6: Definition of the generic unary calculated measure `Generic Aggregated Measure` with its template for the language SQL in the Oracle 11 dialect

As mentioned above, generic business terms can be specialised by instantiation to meet a specific information need. Only generic business term that are free of unbound constraint and template parameters can be applied in the course of a pattern application. The generic business term to be specialized can either be found in vocabulary of an local enriched multidimensional model or in vocabularies defined for more abstract enriched multidimensional models.

Example 10.8 (Generic Business Term Instantiation)**.** The generic business term `Generic Aggregated Measure` (Listing 10.7) can be instantiated as follows to obtain the (specific)

business term Average Milk Yield (Listing 4.3). The constraint parameter ⟨cubeMeasure⟩ is bound to the measure name Milk Yield (Line 3), the name of the numerical value set Lidquid In Liter is bound to the constraint parameter ⟨measureDom⟩ (Line 4), and the aggregation operation AVG is bound to the template parameter ⟨aggregation⟩ (Line 5). ♢

```
1 INSTANTIATE TERM "Generic Repository"/"Generic Business
    Terms"/"Generic Aggregated Measure" AS "Happy
    Milk"/"Diary Vocabulary"/"Average Milk Yield" WITH
2   BINDINGS
3      <cubeMeasure> = "Milk Yield",
4      <measureDom> = "Liquid In Liter",
5      <aggregation> = "AVG"
6   END BINDINGS;
```

Listing 10.7: Instantiation of the generic business term Generic Aggregated Measure

10.4 Description of Value Sets

The presented core concepts take into account only names of value sets. Value sets, however, may be extended by the scale of measurement, i.e., nominal, ordinal, interval, and ratio values, by a base type with conditions that values must meet, and by an enumeration of allowed values. In addition, value sets may be detailed by defining the unit of measurement – conversions between these units can be recorded separately.

We introduce comprehensive descriptions of value sets to cover additional information. To this end, we introduce value spaces, the unit of measurements, and conversions for number value sets. Value spaces allow to define the digits after the comma (precision), the digits before the comma (scale), the level of measurement, and value boundaries (greater than, lower than). Alternatively, a value space can include an enumeration of allowed values instead of value boundaries. In addition, the unit of measurement and its abbreviation are captured. Conversions state how a value from one value set can be represented as a value of another value set. For string value sets only the value space is defined, that is, an ordered list of allowed values as well as its level of measurements can be defined.

```
1 CREATE STRING_VALUE_SET "Happy Milk"/"Diary MDM"/"Breed Name" WITH
2    VALUE_SPACE
3       ALLOWED = { "Friesian", "Guernsey", "Holstein", "Jersey" }
4       LEVEL = "NOMINAL";
5    END VALUE_SPACE;
6 END NUMBER_VALUE_SET;
```

Listing 10.8: Definition of the string value set `Breed Name`

```
1  CREATE NUMBER_VALUE_SET "Happy Milk"/"Diary
       MDM"/"Liquid In Liter" WITH
2     VALUE_SPACE
3        PRECISION = "8";
4        SCALE = "2";
5        LEVEL = "RATIO";
6        GREATER_THAN = "0.00";
7     END VALUE_SPACE;
8
9     UNIT_OF_MEASUREMENT
10       NAME = "LITER";
11       ABBREVIATION = "L";
12    END UNIT_OF_MEASUREMENT;
13
14    CONVERSIONS
15       "Milliliter" = "1000";
16       "Deciliter" = "10";
17       "Hectoliter" = "0.01";
18    END CONVERSIONS;
19 END NUMBER_VALUE_SET;
```

Listing 10.9: Definition of the number value set `Liquid In Liter`

Example 10.9. (Descriptions of Value Sets) Listing 10.9 depicts the definition of the string value set `Breed Name`. The allowed values are limited to the breeds `Friesian`, `Guernsey`, `Holstein` and `Jersey` (Line 3) that are of nominal measurement level (Line 3). The number value set `Liquid In Liter` is defined in Listing 10.9. The values represent liters (Lines 9-12) and are ratio-scaled numbers (Line 5) above zero (Line 6) that are defined with a precision of eight digits and a scale of two digits (Lines 3-4). In addition, conversions to `Milliliter`, `Deciliter`, and `Hectoliter` are defined (Lines 14-18). ♢

Providing comprehensive descriptions for value sets is not essential to the pattern-based approach but it can facilitate the interpretation of results obtained by processing the generated OLAP query.

Chapter 11

Conclusion

In this thesis, we have presented the core concepts of a pattern-based approach to multidimensional data analysis. OLAP patterns allow to capture implicit expert knowledge in data analysis, i.e., solutions for satisfying certain types of information needs by means of OLAP queries, in a form that allows them to be shared across domain boundaries. In addition, OLAP patterns support the generation of queries using templates that can be enriched with corresponding macros to bridge the gap between the conceptual and the logical level. The conceptual representation of a multidimensional model, extended with definitions of business terms that represent different types of query functionality, serves as a basis for the definition and usage of OLAP patterns.

11.1 Summary

BI projects for developing OLAP systems often suffer from requirements of actual BI users being vague. This is because the queries to be supported are usually not known in advance, as they depend on the particular data warehouse design and the data to be integrated. However, when requirements are vague, it is not possible to evaluate the data warehouse design early on, i.e., whether it will meet the information needs that will actually arise. This problem can be addressed to some extent by taking into account knowledge about the composition of OLAP queries from previous experience. Such knowledge may come from the use of existing OLAP systems or previous BI projects within the organization, other organizations, or even other domains.

We have observed that within the same organization and even in other organizations and domains, similar OLAP queries are regularly assembled from scratch to satisfy similar information needs. For this reason, we have presented a pattern-based approach to multidimensional data analysis that allows knowledge to be documented within and across

organizations and domains as OLAP queries are composed. In this way, OLAP patterns enable the capture of best practices in the composition of OLAP queries to answer different types of information needs in the specific data analysis context, taking into account logical dependencies. By documenting the key aspects to consider when composing an OLAP query to answer a specific type of information need, OLAP patterns enable knowledge transfer across organizations and domains. Depending on the required context, OLAP patterns can be defined in a domain-independent, domain-specific, and organization-specific manner. This ultimately enables the sharing of catalogs of OLAP patterns to define a standard vocabulary that can be used in BI projects in specific domains and beyond.

To this end, in this thesis we have presented a formal definition of OLAP patterns and the underlying eMDM. We also formally define how OLAP patterns can be used by means of instantiation, grounding, and execution with respect to an associated eMDM. A textual language is presented to easily define eMDMs and OLAP patterns. Also, a graphical notation is provided to facilitate communication among stakeholders. In addition, means for organizing patterns along the abstraction levels of organization-specific, domain-specific, and domain-independent patterns are presented. This includes the orthogonal organization of OLAP patterns and eMDM elements into catalogs, multidimensional models, and vocabularies. Both the concepts presented for defining and using patterns and the associated language elements have been successfully evaluated as a DSL. Finally, the feasibility of the pattern-based approach to multidimensional data analysis has been demonstrated by a prototype implementation.

11.2 Discussion

The introduction of patterns in other domains, such as architecture [29] and software engineering [13], has shown that patterns can promote the transfer of knowledge about established solutions for certain recurring problems. In the context of the research and development project agriProKnow we could show that domain-independent OLAP patterns can be used to abstract and document knowledge from foreign domains and to reuse it in a concrete BI project. The knowledge transfer through domain-independent OLAP patterns allowed us to consider possible future OLAP requests already at the beginning of the agriProKnow project, which promoted an early evaluation of the data warehouse design. However, the definition of these domain-independent OLAP patterns was only possible due to the extensive know-how from previous BI projects. BI users with rudimentary knowledge are therefore unlikely to independently define OLAP patterns from scratch. Nevertheless, the BI domain could benefit from OLAP patterns because they enable knowledge transfer and can make ad hoc composition of OLAP queries easier and less error-prone. In addition, OLAP

patterns can also enable the establishment of a standard vocabulary for communicating types of information needs and their solutions.

The presented pattern-based approach to multidimensional data analysis is based on an enriched conceptual representation of the data warehouse – the eMDM. Although the eMDM covers the essential expressiveness to represent the most common data warehouse designs, it lacks to represent more comprehensive designs, such as one with common dimension levels. Nevertheless, we could show how the most essential business case in the dairy industry, milkings, can be represented with it. In addition, the naming approach taken in the agriProKnow project, where each element of the eMDM had to be uniquely named, proved to be too rigid, so in this work we relaxed the uniqueness of names to the effect that only entities, business terms, and value sets need to be uniquely named, while properties only need to be uniquely named within their entity.

Our pattern-based approach follows a name-based definition approach, i.e., in the course of defining OLAP patterns, only names and types of expected eMDM elements are defined in constraints and derivation rules. Thus, no specific elements from a particular eMDM are referenced. By not referencing specific eMDM elements, OLAP patterns can be applied to all eMDMs that provide elements that match the defined names and types, i.e., allow the evaluation of all derivation rules as well as the satisfaction of all constraints. This is particularly beneficial when domain-independent and domain-specific OLAP patterns are defined and applied across different organizations. However, this name-based definition approach also has the disadvantage that only one spelling of the name can be considered when checking the existing eMDM elements, even if upper and lower case are not considered. While this can be easily remedied with additional name mappings, it results in extra work during the application of our pattern-based approach.

The presented approach not only enables knowledge transfer across domain boundaries, but also the organization of the defined OLAP patterns along different abstraction levels. In order to make OLAP patterns usable from a higher level of abstraction to lower levels of abstraction, the OLAP patterns are provided with variables that enable the adaptation of the OLAP pattern to a concrete context, the eMDM. For this purpose, an instantiation mechanism has been introduced that allows variables to be bound to concrete names. However, this instantiation mechanism does not check the actual applicability of the customized pattern with respect to an eMDM; the applicability check is performed in a separate step. In this way, domain-independent patterns can be defined without necessarily requiring a reference eMDM, but it is also possible to obtain instantiated patterns that are not necessarily applicable. It should be noted that no mechanism is provided to extend certain OLAP pattern definitions with additional variables, i.e., an extension always requires

redefinition of the pattern.

A fully instantiated pattern can be executed in the context of an associated eMDM if, and only if, all derivation rules can be evaluated and all constraints can be satisfied by corresponding eMDM elements. Thus, each OLAP pattern defines a contract that describes the conditions that must be satisfied in order to apply the pattern. This prevents a pattern user from receiving an OLAP query that cannot be executed if the contract conditions are not met. However, to run an applicable OLAP pattern, the pattern user must currently ensure that the data warehouse system is implemented as a star schema and that the names of the fact and dimension tables or their columns match the names used in eMDM. If name discrepancies still exist, an appropriate name mapping must be specified. This drawback could be addressed by considering conceptual representations that reflect the actual data stored.

We evaluated our pattern-based approach as a domain-specific language from a language developer's perspective by applying the FQAD [91]. The goal of this evaluation is to assess whether the specified language is fully and consistently specified in terms of the necessary functionality for using our pattern-based approach. We were able to show that the pattern-based approach meets the goal, but since only the language developer's perspective was taken, some quality attributes irrelevant to this perspective were not considered. In particular, usability, which is important from the perspective of other stakeholders, was currently not sufficiently considered.

Finally, the feasibility of our pattern-based approach to multidimensional data analysis was demonstrated by its application in the agriProKnow project and by the prototype implementation described in this thesis. The prototype allows defining eMDMs and OLAP patterns and organizing them into multidimensional models, vocabularies and catalogs. However, the prototype provides only a textual interface that does not use the notation presented.

11.3 Future Work

Future work will investigate the usability of the introduced notation. For this purpose, the currently text-based editor application of the prototype has to be extended by a visualization component that displays eMDMs and OLAP patterns according to the introduced notation. In addition, the pattern-based approach can be further extended to support composite types for and optionality of variables, the definition of generic business terms, and the comprehensive description of sets of values. In addition, future work could address how to easily extend the expressive power of the pattern-based approach. In particular, multi-level modeling approaches can be pursued to increase the expressive power of eMDM as needed

to support richer conceptual data warehouse models. In this case, multi-level modeling approaches would also need to be used for patterns to address the potential new constraints associated with them. Finally, future work will also explore the application of the pattern approach with other target languages, e.g., defining patterns for the statistical programming language R or Multidimensional eXpressions (MDX); the possibility of expressing OLAP patterns for target languages other than SQL has already been demonstrated in the context of Linked Open Data using SPARQL as the target language [16].

Bibliography

[1] I. Kovacic, C. G. Schuetz, S. Schausberger, R. Sumereder and M. Schrefl, 'Guided Query Composition with Semantic OLAP Patterns,' in *Proceedings of the Workshops of the EDBT/ICDT 2018 Joint Conference (EDBT/ICDT 2018), Vienna, Austria, March 26, 2018*, N. Augsten, Ed., ser. CEUR Workshop Proceedings, vol. 2083, CEUR-WS.org, 2018, pp. 67–74.

[2] C. G. Schuetz, S. Schausberger, I. Kovacic and M. Schrefl, 'Semantic OLAP Patterns: Elements of Reusable Business Analytics,' in *On the Move to Meaningful Internet Systems. OTM 2017 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017, Proceedings, Part II*, H. Panetto, C. Debruyne, W. Gaaloul *et al.*, Eds., ser. Lecture Notes in Computer Science, vol. 10574, Springer, 2017, pp. 318–336. DOI: 10.1007/978-3-319-69459-7_22.

[3] G. Allen and J. Parsons, 'Is Query Reuse Potentially Harmful? Anchoring and Adjustment in Adapting Existing Database Queries,' *Information Systems Research*, vol. 21, no. 1, pp. 56–77, Mar. 2010, ISSN: 1047-7047. DOI: 10.1287/isre.1080.0189.

[4] C. G. Schuetz, S. Schausberger and M. Schrefl, 'Building an active semantic data warehouse for precision dairy farming,' *J. Org. Computing and E. Commerce*, vol. 28, no. 2, pp. 122–141, 2018. DOI: 10.1080/10919392.2018.1444344.

[5] T. Neuböck, B. Neumayr, M. Schrefl and C. G. Schütz, 'Ontology-driven business intelligence for comparative data analysis,' in *eBISS 2013*, E. Zimányi, Ed., ser. LNBIP, vol. 172, Springer, 2014, pp. 77–120. DOI: 10.1007/978-3-319-05461-2_3.

[6] C. G. Schuetz, B. Neumayr, M. Schrefl and T. Neuböck, 'Reference Modeling for Data Analysis: The BIRD Approach,' *International Journal of Cooperative Information*

Systems, vol. 25, no. 02, pp. 1–46, Jun. 2016, ISSN: 0218-8430. DOI: 10.1142/S0218843016500064.

[7] C. G. Schuetz and M. Schrefl, 'Customization of domain-specific reference models for data warehouses,' in *Proceedings of the 18th IEEE International Enterprise Distributed Object Computing Conference*, M. Reichert, S. Rinderle-Ma and G. Grossmann, Eds., 2014, pp. 61–70. DOI: 10.1109/EDOC.2014.18.

[8] T. Neuböck and M. Schrefl, 'Modelling Knowledge about Data Analysis Processes in Manufacturing,' *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 277–282, 2015, 15th IFAC Symposium on Information Control Problems in Manufacturing, ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2015.06.094.

[9] D. L. Parnas, 'Precise Documentation: The Key to Better Software,' in *The Future of Software Engineering*, 2010, pp. 125–148. DOI: 10.1007/978-3-642-15187-3_8.

[10] B. Renzl, 'Trust in management and knowledge sharing: The mediating effects of fear and knowledge documentation,' *Omega*, vol. 36, no. 2, pp. 206–220, 2008, ISSN: 0305-0483. DOI: 10.1016/j.omega.2006.06.005.

[11] M. Fowler, *Analysis patterns - reusable object models*, ser. Addison-Wesley series in object-oriented software engineering. Addison-Wesley-Longman, 1997, ISBN: 978-0-201-89542-1.

[12] C. Alexander, *The Timeless Way of Building*. Oxford University Press, 1979, ISBN: 978-0-19-502402-9.

[13] E. Gamma, R. Helm, R. E. Johnson and J. M. Vlissides, 'Design patterns: Abstraction and reuse of object-oriented design,' in *ECOOP'93 - Object-Oriented Programming, 7th European Conference, Kaiserslautern, Germany, July 26-30, 1993, Proceedings*, O. Nierstrasz, Ed., ser. Lecture Notes in Computer Science, vol. 707, Springer, 1993, pp. 406–431. DOI: 10.1007/3-540-47910-4_21.

[14] S. Nalchigar and E. S. K. Yu, 'Business-driven data analytics: A conceptual modeling framework,' *Data Knowl. Eng.*, vol. 117, pp. 359–372, 2018. DOI: 10.1016/j.datak.2018.04.006.

[15] S. Nalchigar, E. S. K. Yu, Y. Obeidi, S. Carbajales, J. Green and A. Chan, 'Solution Patterns for Machine Learning,' in *Advanced Information Systems Engineering - 31st International Conference, CAiSE 2019, Rome, Italy, June 3-7, 2019, Proceedings*, P. Giorgini and B. Weber, Eds., ser. LNCS, vol. 11483, Springer, 2019, pp. 627–642. DOI: 10.1007/978-3-030-21290-2_39.

[16] M. Hilal, C. G. Schuetz and M. Schrefl, 'Using superimposed multidimensional schemas and OLAP patterns for RDF data analysis,' *Open Computer Science*, vol. 8, no. 1, pp. 18–37, 2018. DOI: 10.1515/comp-2018-0003.

[17] W. W. Eckerson, 'The Keys to Enterprise Business Intelligence: Critical Success Factors,' *TDWI Best Practices Report*, 2005.

[18] J. Varga, E. Dobrokhotova, O. Romero, T. B. Pedersen and C. Thomsen, 'SM4MQ: A semantic model for multidimensional queries,' in *The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 - June 1, 2017, Proceedings, Part I*, E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler and O. Hartig, Eds., ser. Lecture Notes in Computer Science, vol. 10249, 2017, pp. 449–464. DOI: 10.1007/978-3-319-58068-5_28.

[19] D. Hay, K. A. Healy, J. Hall *et al.*, 'Defining Business Rules: What Are They Really?' Business Rules Group, Tech. Rep., Jul. 2000.

[20] C. Kohls, 'The Theories of Design Patterns and their Practical Implications exemplified for E-Learning Patterns,' Ph.D. dissertation, Catholic University Eichstätt-Ingolstadt, 2014.

[21] C. Caracciolo, A. Stellato, A. Morshed *et al.*, 'The AGROVOC linked dataset,' *Semantic Web*, vol. 4, no. 3, pp. 341–348, 2013. DOI: 10.3233/SW-130106.

[22] K. Donnelly, 'SNOMED-CT: The advanced terminology and coding system for ehealth,' *Studies in health technology and informatics*, vol. 121, p. 279, 2006.

[23] S. Anderlik, B. Neumayr and M. Schrefl, 'Using domain ontologies as semantic dimensions in data warehouses,' in *ER 2012*, P. Atzeni, D. W. Cheung and S. Ram, Eds., ser. LNCS, vol. 7532, Springer, 2012, pp. 88–101. DOI: 10.1007/978-3-642-34002-4_7.

[24] J. Bewley, 'Precision dairy farming: Advanced analysis solutions for future profitability,' in *Proceedings of the first North American conference on precision dairy management*, 2010, pp. 2–5.

[25] M. Golfarelli, D. Maio and S. Rizzi, 'The Dimensional Fact Model: A Conceptual Model for Data Warehouses,' *Int. J. Cooperative Inf. Syst.*, vol. 7, no. 2-3, pp. 215–247, 1998. DOI: 10.1142/S0218843098000118.

[26] P. P.-S. Chen, 'The Entity-Relationship Model—toward a Unified View of Data,' *ACM Trans. Database Syst.*, vol. 1, no. 1, pp. 9–36, 1976, ISSN: 0362-5915. DOI: 10.1145/320434.320440.

[27] W. W. Eckerson, 'Pervasive Business Intelligence: Techniques and Technologies to Deploy BI on an Enterprise Scale,' *TDWI Best Practices Report*, 2008.

[28] B. Meyer, 'Applying "design by contract",' *Computer*, vol. 25, no. 10, pp. 40–51, 1992. DOI: 10.1109/2.161279.

[29] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King and S. Angel, *A Pattern Language - Towns, Buildings, Construction*. Oxford University Press, 1977, ISBN: 978-0-19-501919-3.

[30] K. Beck and W. Cunningham, 'A laboratory for teaching object oriented thinking,' in *Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications*, ser. OOPSLA '89, New Orleans, Louisiana, USA: Association for Computing Machinery, 1989, pp. 1–6, ISBN: 0897913337. DOI: 10.1145/74877.74879.

[31] P. Coad, 'Object-oriented patterns,' *Commun. ACM*, vol. 35, no. 9, pp. 152–159, 1992. DOI: 10.1145/130994.131006.

[32] R. E. Johnson, 'Frameworks = (components + patterns),' *Commun. ACM*, vol. 40, no. 10, pp. 39–42, 1997. DOI: 10.1145/262793.262799.

[33] J. O. Coplien, *Software Patterns. SIGS Management Briefings Series*, 1996.

[34] J. O. Coplien and N. B. Harrison, *Organizational Patterns of Agile Software Development*. USA: Prentice-Hall, Inc., 2004, ISBN: 0131467409.

[35] M. E. Conway, 'How do committees invent,' *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

[36] J. O. Coplien, 'Organizational patterns,' in *Enterprise Information Systems VI*, I. Seruca, J. Cordeiro, S. Hammoudi and J. Filipe, Eds., Dordrecht: Springer Netherlands, 2006, pp. 43–52, ISBN: 978-1-4020-3675-0.

[37] O. Muller and B. Haberman, 'Supporting abstraction processes in problem solving through pattern-oriented instruction,' *Computer Science Education*, vol. 18, no. 3, pp. 187–212, 2008. DOI: 10.1080/08993400802332548.

[38] I. Graham, *Business Rules Management and Service Oriented Architecture: A Pattern Language*. Wiley, 2007, ISBN: 9780470059807.

[39] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski and A. P. Barros, 'Workflow patterns,' *Distributed Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003. DOI: 10.1023/A:1022883727209.

[40] D. C. Hay, *Data Model Patterns: Conventions of Thought*. Dorset House, 1996, ISBN: 9780932633293.

[41] L. Silverston, *The Data Model Resource Book, A Library of Universal Data Models by Industry Types*. John Wiley & Sons, Mar. 2001, vol. 2, ISBN: 0471353485.

[42] L. Silverston and P. Agnew, *The Data Model Resource Book, Universal Patterns for Data Modeling*. John Wiley & Sons, Dec. 2008, vol. 3, ISBN: 0470178450.

[43] M. Blaha, *Patterns of Data Modeling*, 1st. USA: CRC Press, Inc., 2010, ISBN: 1439819890.

[44] J. Arlow and I. Neustadt, *Enterprise patterns and MDA: building better software with archetype patterns and UML*. Addison-Wesley Professional, 2004.

[45] D. Batra, 'Conceptual data modeling patterns: Representation and validation,' *J. Database Manag.*, vol. 16, no. 2, pp. 84–106, 2005. DOI: 10.4018/jdm.2005040105.

[46] L. Silverston, *The Data Model Resource Book, A Library of Universal Data Models for All Enterprises*, R. M. Elliott, Ed. John Wiley & Sons, Apr. 2001, vol. 1, ISBN: 0-471-38023-7.

[47] D. Hay, *Data Model Patterns: A Metadata Map*, ser. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2010, ISBN: 9780080477039.

[48] W. H. Inmon, *Building the Data Warehouse*, Fourth Edition. John Wiley & Sons, Inc, Sep. 2005.

[49] A. A. Vaisman and E. Zimányi, *Data Warehouse Systems - Design and Implementation*, ser. Data-Centric Systems and Applications. Springer, 2014, ISBN: 978-3-642-54654-9. DOI: 10.1007/978-3-642-54655-6.

[50] S. Schneider and D. Frosch-Wilke, 'Analysis patterns in dimensional data modeling,' in *Data Engineering and Management*, R. Kannan and F. Andres, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 109–116, ISBN: 978-3-642-27872-3.

[51] O. R. Zaïane, M. Xin and J. Han, 'Discovering web access patterns and trends by applying OLAP and data mining technology on web logs,' in *Proceedings of the IEEE Forum on Reasearch and Technology Advances in Digital Libraries, IEEE ADL '98, Santa Barbara, California, USA, April 22-24, 1998*, IEEE Computer Society, 1998, pp. 19–29. DOI: 10.1109/ADL.1998.670376.

[52] K. Boulil, F. L. Ber, S. Bimonte, C. Grac and F. Cernesson, 'Multidimensional modeling and analysis of large and complex watercourse data: An olap-based solution,' *Ecol. Informatics*, vol. 24, pp. 90–106, 2014. DOI: 10.1016/j.ecoinf.2014.07.001.

[53] P. Viqarunnisa, H. Laksmiwati and F. N. Azizah, 'Generic Data Model Pattern for Data Warehouse,' in *International Conference on Electrical Engineering and Informatics, ICEEI 2011, Bandung, Indonesia, 17-19 July, 2011*, A. Syaichu-Rohman, D. Hamdani, S. Akbar, W. Adiprawita, R. Razali and N. Sahari, Eds., IEEE, 2011, pp. 1–8. DOI: 10.1109/ICEEI.2011.6021805.

[54] L. Corr and J. Stagnitto, *Agile Data Warehouse Design: Collaborative Dimensional Modeling, from Whiteboard to Star Schema*. Leeds, UK: DecisionOne Press, Nov. 2014, vol. 4, ISBN: 0956817203.

[55] M. E. Jones and I. Song, 'Dimensional modeling: Identification, classification, and evaluation of patterns,' *Decis. Support Syst.*, vol. 45, no. 1, pp. 59–76, 2008. DOI: 10.1016/j.dss.2006.12.004.

[56] J. Poole, D. Chang, D. Tolbert and D. Mellor, *Common Warehouse Metamodel Developer's Guide*. Wiley Publishing, Inc., 2003, ISBN: 0-471-20243-6.

[57] B. Zohuri and M. Moghaddam, 'What Is Data Analysis from Data Warehousing Perspective?' In *Business Resilience System (BRS): Driven Through Boolean, Fuzzy Logics and Cloud Computation: Real and Near Real Time Analysis and Decision Making System*. Springer International Publishing, 2017, pp. 269–289, ISBN: 978-3-319-53417-6. DOI: 10.1007/978-3-319-53417-6_10.

[58] L. Greiner. 'What is Data Analysis and Data Mining? - Database Trends and Applications.' (2011), [Online]. Available: https://www.dbta.com/Editorial/Trends-and-Applications/What-is-Data-Analysis-and-Data-Mining-73503.aspx (visited on 19/01/2021).

[59] V. Tropashko and D. Burleson, *SQL Design Patterns: Expert Guide to SQL Programming*. Rampant TechPress, 2007, ISBN: 0977671542.

[60] H. Al-Shuaily, 'SQL Pattern Design, Development & Evaluation of its Efficacy,' Ph.D. dissertation, University of Glasgow, 2013.

[61] K. Renaud and J. van Biljon, 'Teaching SQL - which pedagogical horse for this course?' In *Key Technologies for Data Management, 21st British National Conference on Databases, BNCOD 21, Edinburgh, UK, July 7-9, 2004, Proceedings*, M. H. Williams and L. M. MacKinnon, Eds., ser. Lecture Notes in Computer Science, vol. 3112, Springer, 2004, pp. 244–256. DOI: 10.1007/978-3-540-27811-5_22.

[62] L. Sundin and Q. I. Cutts, 'Is it feasible to teach query programming in three different languages in a single session?: A study on a pattern-oriented tutorial and cheat sheets,' in *Proceedings of the 1st UK & Ireland Computing Education Research Conference, UKICER 2019, Canterbury, UK, September 5-6, 2019*, J. Carter, B. A. Becker and N. C. C. Brown, Eds., ACM, 2019, 7:1–7:7. DOI: 10.1145/3351287.3351293.

[63] C. Nagy and A. Cleve, 'Mining stack overflow for discovering error patterns in SQL queries,' in *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, R. Koschke, J. Krinke and M. P. Robillard, Eds., IEEE Computer Society, 2015, pp. 516–520. DOI: 10.1109/ICSM.2015.7332505.

[64] B. Karwin, *SQL Antipatterns: Avoiding the Pitfalls of Database Programming*, 1st. Pragmatic Bookshelf, 2010, ISBN: 1934356557.

[65] P. Dintyala, A. Narechania and J. Arulraj, 'Sqlcheck: Automated detection and diagnosis of SQL anti-patterns,' in *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini and H. Q. Ngo, Eds., ACM, 2020, pp. 2331–2345. DOI: 10.1145/3318464.3389754.

[66] M. Poess, B. Smith, L. Kollar and P. Larson, 'Tpc-ds, taking decision support benchmarking to the next level,' in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '02, Madison, Wisconsin: Association for Computing Machinery, 2002, pp. 582–587, ISBN: 1581134975. DOI: 10.1145/564691.564759.

[67] M. Poess and J. M. Stephens, 'Generating thousand benchmark queries in seconds,' in *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, Toronto, Canada, August 31 - September 3 2004*, M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley and K. B. Schiefer, Eds., Morgan Kaufmann, 2004, pp. 1045–1053. DOI: 10.1016/B978-012088469-8.50091-7. [Online]. Available: http://www.vldb.org/conf/2004/IND2P3.PDF.

[68] A. Giacometti, P. Marcel, E. Negre and A. Soulet, 'Query Recommendations for OLAP Discovery Driven Analysis,' in *DOLAP 2009, ACM 12th International Workshop on Data Warehousing and OLAP, Hong Kong, China, November 6, 2009, Proceedings*, I. Song and E. Zimányi, Eds., ACM, 2009, pp. 81–88. DOI: 10.1145/1651291.1651306.

[69] C. Sapia, 'On Modeling and Predicting Query Behavior in OLAP Systems,' in *Proceedings of the Intl. Workshop on Design and Management of Data Warehouses, DMDW'99, Heidelberg, Germany, June 14-15, 1999*, S. Gatziu, M. A. Jeusfeld, M. Staudt and Y. Vassiliou, Eds., ser. CEUR Workshop Proceedings, vol. 19, CEUR-WS.org, 1999, p. 2. [Online]. Available: http://ceur-ws.org/Vol-19/paper2.pdf.

[70] A. Unwin, 'Patterns of Data Analysis,' *Journal of the Korean Statistical Society*, vol. 30, no. 2, pp. 219–230, 2001.

[71] P. Law, R. C. Basole and Y. Wu, 'Duet: Helping data analysis novices conduct pairwise comparisons by minimal specification,' *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 1, pp. 427–437, 2019. DOI: 10.1109/TVCG.2018.2864526.

[72] P.-M. Law, S. Das and R. C. Basole, 'Comparing Apples and Oranges: Taxonomy and Design of Pairwise Comparisons within Tabular Data,' in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19, Glasgow, Scotland Uk: Association for Computing Machinery, 2019. DOI: 10.1145/3290605.3300409.

[73] C. Stolte, D. Tang and P. Hanrahan, 'Polaris: A system for query, analysis, and visualization of multidimensional relational databases,' *IEEE Trans. Vis. Comput. Graph.*, vol. 8, no. 1, pp. 52–65, 2002. DOI: 10.1109/2945.981851.

[74] M. Böhnlein, 'Konstruktion semantischer Data-Warehouse-Schemata,' in *Konstruktion semantischer Data Warehouse-Strukturen auf Grundlage von Geschäftsprozeßmodellen*, Deutscher Universitätsverlag, 2001, ch. 5, pp. 303–376, ISBN: 978-3-663-08649-9. DOI: 10.1007/978-3-663-08649-9_6.

[75] M. Böhnlein, A. Ulbrich-vom Ende and M. Plaha, 'Visual specification of multidimensional queries based on a semantic data model,' in *Vom Data Warehouse zum Corporate Knowledge Center*, E. von Maur and R. Winter, Eds., Heidelberg: Physica-Verlag HD, 2002, pp. 379–397, ISBN: 978-3-642-57491-7.

[76] J. Danaparamita and W. Gatterbauer, 'Queryviz: Helping users understand SQL queries and their patterns,' in *EDBT 2011, 14th International Conference on Extending Database Technology, Uppsala, Sweden, March 21-24, 2011, Proceedings*, A. Ailamaki, S. Amer-Yahia, J. M. Patel, T. Risch, P. Senellart and J. Stoyanovich, Eds., ACM, 2011, pp. 558–561. DOI: 10.1145/1951365.1951440.

[77] A. Leventidis, J. Zhang, C. Dunne, W. Gatterbauer, H. V. Jagadish and M. Riedewald, 'Queryvis: Logic-based diagrams help users understand complicated SQL queries faster,' in *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini and H. Q. Ngo, Eds., ACM, 2020, pp. 2303–2318. DOI: 10.1145/3318464.3389767.

[78] J. Pardillo, J. Mazón and J. Trujillo, 'Extending OCL for OLAP querying on conceptual multidimensional models of data warehouses,' *Inf. Sci.*, vol. 180, no. 5, pp. 584–601, 2010. DOI: 10.1016/j.ins.2009.11.006.

[79] J. Cabot, J. Mazón, J. Pardillo and J. Trujillo, 'Specifying aggregation functions in multidimensional models with OCL,' in *Conceptual Modeling - ER 2010, 29th International Conference on Conceptual Modeling, Vancouver, BC, Canada, November 1-4, 2010. Proceedings*, J. Parsons, M. Saeki, P. Shoval, C. C. Woo and Y. Wand, Eds., ser. Lecture Notes in Computer Science, vol. 6412, Springer, 2010, pp. 419–432. DOI: 10.1007/978-3-642-16373-9_30.

[80] Object Management Group, *Specifications: Model Driven Architecture (MDA)*, 2011. [Online]. Available: https://www.omg.org/mda (visited on 21/01/2021).

[81] E. Malinowski and E. Zimányi, 'Hierarchies in a multidimensional model: From conceptual modeling to logical representation,' *Data & Knowledge Engineering*, vol. 59, no. 2, pp. 348–377, 2006. DOI: 10.1016/j.datak.2005.08.003.

[82] J. H. Larkin and H. A. Simon, 'Why a diagram is (sometimes) worth ten thousand words,' *Cogn. Sci.*, vol. 11, no. 1, pp. 65–100, 1987. DOI: 10.1111/j.1551-6708.1987.tb00863.x.

[83] D. Moody, 'The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering,' *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 756–779, Nov. 2009, ISSN: 2326-3881. DOI: 10.1109/TSE.2009.67.

[84] J. Mazón and J. Trujillo, 'An MDA approach for the development of data warehouses,' in *XIV Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2009), San Sebastián, Spain, September 8-11, 2009*, A. Vallecillo and G. Sagardui, Eds., 2009, pp. 208–208.

[85] Zentralverband Elektrotechnik- und Elektronikindustrie (ZVEI), *ZVEI-Kennzahlensystem: ein Instrument zur Unternehmenssteuerung*, 4th. 1989, list of key performance indicators proposed by the German Association of Electrical and Electronic Manufacturers.

[86] A. P. C. Chan and A. P. L. Chan, 'Key performance indicators for measuring construction success,' *Benchmarking: an international journal*, vol. 11, no. 2, pp. 203–221, 2004. DOI: 10.1108/14635770410532624.

[87] W. Leiderer, *Kennzahlen zur Steuerung von Hotel- und Gaststättenbetrieben*, 2nd. Matthaes, Stuttgart, 1983, list of key performance indicators for hotels and food & beverage companies.

[88] M. Moritz, 'Prototype of a Tool for Managing and Executing OLAP Patterns,' M.S. thesis, Johannes Kepler University Linz, Nov. 2020.

[89] S. Schausberger, 'The Semantic Data Warehouse for the AgriProKnow Project: A First Prototype,' M.S. thesis, Johannes Kepler University Linz, Nov. 2016.

[90] J. Arnoldus, M. G. J. van den Brand, A. Serebrenik and J. Brunekreef, *Code Generation with Templates*, ser. Atlantis Studies in Computing. Atlantis Press, 2012, vol. 1, ISBN: 978-94-91216-55-8. DOI: 10.2991/978-94-91216-56-5.

[91] G. Kahraman and S. Bilgen, 'A Framework for Qualitative Assessment of Domain-specific Languages,' *Softw. Syst. Model.*, vol. 14, no. 4, pp. 1505–1526, Oct. 2015, ISSN: 1619-1366. DOI: 10.1007/s10270-013-0387-8.

[92] R. F. Paige, J. S. Ostroff and P. J. Brooke, 'Principles for modeling language design,' *Inf. Softw. Technol.*, vol. 42, no. 10, pp. 665–675, 2000. DOI: 10.1016/S0950-5849(00)00109-9.

[93] J. Krogstie, 'Evaluating UML Using a Generic Quality Framework,' in *UML and the Unified Process*, IGI Global, Jan. 2003, pp. 1–22. DOI: 10.4018/978-1-93177-744-5.ch001.

[94] S. Kelly and R. Pohjonen, 'Worst practices for domain-specific modeling,' *IEEE Software*, vol. 26, no. 4, pp. 22–29, 2009. DOI: 10.1109/MS.2009.109.

[95] G. Viswanathan and M. Schneider, 'CAL: A generic query and analysis language for data warehouses,' in *ISCA 20th International Conference on Software Engineering and Data Engineering (SEDE-2011) June 20-22, 2011, Imperial Palace Hotel, Las Vegas, Nevada USA*, R. Zalila-Wenkstern and W. Wu, Eds., ISCA, 2011, pp. 18–23.

[96] L. I. Gómez, S. A. Gómez and A. A. Vaisman, 'A generic data model and query language for spatiotemporal olap cube analysis,' in *Proceedings of the 15th International Conference on Extending Database Technology*, ser. EDBT '12, Berlin, Germany: Association for Computing Machinery, 2012, pp. 300–311, ISBN: 9781450307901. DOI: 10.1145/2247596.2247632.

List of Figures

List of Tables

List of Listings

Appendix A

ANTLR4 Grammar Definitions

Listing A.1 defines the types and basic statements available in the language, Listing A.2 extends the syntax for defining multidimensional models, Listing A.3 extends the syntax for defining business terms, Listing A.4 extends the syntax for defining patterns, while Listing A.5 extends the syntax for using of patterns. It should be noted that these syntax definitions are extended to include statements necessary for the prototypical implementation. For this purpose, Listing A.6 extends the syntax for defining organizational elements. In addition, the syntax definition has been formatted for better readability. For the sake of clarity and brevity, a cluttered syntax definition is avoided by not detailing trivial grammar rules in syntax definitions. Instead, we assume that grammar rules ending with `_name` represent strings in double quotes without line breaks, grammar rules ending with `_txt` represent strings in double quotes with lines breaks, while grammar rules ending with `_label` represent strings without double quotes and line breaks. All definitions are case-insensitive.

Listing A.1: Syntax of the types and basic statements of the language

```
1  ⟨type⟩ :
2      ⟨m_entity_type⟩
3      | ⟨m_prop_type⟩
4      | ⟨t_type⟩
5      | ⟨v_type⟩ ;
6
7  ⟨m_entity_type⟩ :
8      CUBE
9      | DIMENSION ;
10
11 ⟨m_prop_type⟩ :
12     MEASURE
13     | DIMENSION_ROLE
14     | LEVEL
15     | ATTRIBUTE ;
16
17 ⟨t_type⟩ :
18     UNARY_CUBE_PREDICATE
19     | BINARY_CUBE_PREDICATE
20     | CUBE_ORDERING
21     | UNARY_CALCULATED_MEASURE
22     | BINARY_CALCULATED_MEASURE
23     | UNARY_DIMENSION_PREDICATE
24     | BINARY_DIMENSION_PREDICATE
25     | DIMENSION_GROUPING
26     | DIMENSION_ORDERING ;
27
28 ⟨v_type⟩ :
29     NUMBER_VALUE_SET
30     | STRING_VALUE_SET
31     | ⟨val_set_name⟩ ;
32
33 ⟨emdm_stmt⟩ :
34     (
35         (
36             ⟨c_stmt⟩
37             | ⟨d_stmt⟩
38             | ⟨i_stmt⟩
39             | ⟨g_stmt⟩
40             | ⟨x_stmt⟩
41             | ⟨f_stmt⟩
42         ) ';'
43     )* ;
44
45 ⟨c_stmt⟩ :
```

```
46      CREATE (
47          ⟨cp_stmt⟩
48          | ⟨ct_stmt⟩
49          | ⟨cm_stmt⟩
50          | ⟨cr_stmt⟩
51      ) ;
52
53  ⟨d_stmt⟩ :
54      DELETE (
55          ⟨p_del⟩
56          | ⟨t_del⟩
57          | ⟨m_del⟩
58          | ⟨r_del⟩
59      ) ;
```

Listing A.2: Syntax of the multidimensional model definition statements

```
1   ⟨cm_stmt⟩ :
2       ⟨cube_def⟩
3       | ⟨dim_def⟩ ;
4
5   ⟨cube_def⟩ :
6       CUBE ⟨cube_name⟩ WITH
7           MEASURE PROPERTIES
8               ⟨meas_decl⟩+
9           END MEASURE PROPERTIES ';'
10          DIMENSION_ROLE PROPERTIES
11              ⟨dim_role_decl⟩+
12          END DIMENSION_ROLE PROPERTIES ';'
13      END CUBE ;
14
15  ⟨dim_def⟩ :
16      DIMENSION ⟨dim_name⟩ WITH
17      (
18          (
19              LEVEL PROPERTIES
20                  ⟨lvl_decl⟩+
21              END LEVEL PROPERTIES ';'
22          )
23          |
24          (
25              ATTRIBUTE PROPERTIES
26                  ⟨attr_decl⟩+
27              END ATTRIBUTE PROPERTIES ';'
28          )
29          |
```

```
30          (
31              CONSTRAINTS
32              (
33                 ⟨roll_up_rel_decl⟩
34                 | ⟨descr_rel_decl⟩
35              )+
36              END CONSTRAINTS ';'
37          )
38      )+
39      END DIMENSION ;
40
41  ⟨m_del⟩ :
42      (
43          CUBE ⟨cube_name⟩
44          | DIMENSION ⟨dim_name⟩
45      ) ;
46
47  ⟨meas_decl⟩ :
48      ⟨meas_name⟩ ':' ⟨val_set_name⟩ ';' ;
49
50  ⟨dim_role_decl⟩ :
51      ⟨dim_role_name⟩ ':' ⟨dim_name⟩ ';' ;
52
53  ⟨lvl_decl⟩ :
54      ⟨lvl_name⟩ ':' ⟨val_set_name⟩ ';' ;
55
56  ⟨attr_decl⟩ :
57      ⟨attr_name⟩ ':' ⟨val_set_name⟩ ';' ;
58
59  ⟨roll_up_rel_decl⟩ :
60      ⟨lvl_name⟩ ROLLS_UP_TO ⟨lvl_name⟩ ';' ;
61
62  ⟨descr_rel_decl⟩ :
63      ⟨lvl_name⟩ DESCRIBED_BY ⟨attr_name⟩ ';' ;
64
65  ⟨cube_name⟩:
66      ⟨path_exp⟩ ;
67
68  ⟨dim_name⟩ :
69      ⟨path_exp⟩ ;
70
71  ⟨path_exp⟩ :
72      ⟨elem_name⟩
73      (
74          '/' ⟨elem_name⟩
75      )* ;
```

Listing A.3: Syntax of the business term definition statements

```
⟨ct_stmt⟩ :
    ⟨t_def⟩
    |
    (
        TERM
        (
            ⟨t_descr⟩
            | ⟨t_temp⟩
        )
    ) ;

⟨t_def⟩ :
    ⟨t_type⟩ ⟨t_name⟩
    (
        APPLIES TO ⟨var_decl⟩  (',' ⟨var_decl⟩ )*
    )?
    WITH
    (
        CONSTRAINTS ⟨cstr_decl⟩+ END CONSTRAINTS ';'
    )?
    (
        RETURNS ⟨value_set_name⟩ ';'
    )?
    END ⟨t_type⟩ ;

⟨t_descr⟩ :
    DESCRIPTION FOR ⟨t_name⟩ WITH
    (
        (
            LANGUAGE '=' ⟨lang_name⟩
            | ALIAS '=' ⟨t_name⟩
              (
                  ',' ⟨t_name⟩
              )*
            | DESCRIPTION '=' ⟨descr_txt⟩
        ) ';'
    )+
    END TERM DESCRIPTION ;

⟨t_temp⟩ :
    TEMPLATE FOR ⟨t_name⟩ WITH
        ⟨temp_elem⟩+
    END TERM TEMPLATE ;

⟨temp_elem⟩ :
```

```
46      (
47          ⟨temp_elem_meta⟩
48          | EXPRESSION '=' ⟨temp_txt⟩
49      ) ';' ;
50
51  ⟨temp_elem_meta⟩ :
52      (
53          DATA_MODEL '=' ⟨model_name⟩
54          | VARIANT '=' ⟨variant_name⟩
55          | LANGUAGE '=' ⟨language_name⟩
56          | DIALECT '=' ⟨dialect_name⟩
57      ) ;
58
59  ⟨var_decl⟩ :
60      ⟨var_exp⟩ ':' ⟨type⟩ ;
61
62  ⟨cstr_decl⟩ :
63      (
64          ⟨type_cstr_decl⟩
65          | ⟨prop_cstr_decl⟩
66          | ⟨dom_cstr_decl⟩
67          | ⟨return_cstr_decl⟩
68          | ⟨app_cstr_decl⟩
69      ) ';' ;
70
71  ⟨type_cstr_decl⟩ :
72      ⟨elem_exp⟩ ':' ⟨type⟩ ;
73
74  ⟨prop_cstr_decl⟩ :
75      ⟨elem_exp⟩ HAS ⟨m_prop_type⟩ ⟨elem_exp⟩ ;
76
77  ⟨dom_cstr_decl⟩ :
78      ⟨elem_exp⟩ '.' ⟨elem_exp⟩ ':' ⟨elem_exp⟩ ;
79
80  ⟨elem_exp⟩ :
81      ⟨var_exp⟩
82      | ⟨const_exp⟩ ;
83
84  ⟨var_exp⟩ :
85      '<' ⟨var_label⟩ '>' ;
86
87  ⟨const_exp⟩ :
88      ⟨const_name⟩ ;
89
90  ⟨prop_exp⟩ :
91      ⟨prop_name⟩ ;
```

```
92
93 ⟨t_del⟩ :
94     TERM
95     (
96         ⟨t_del_descr⟩
97         | ⟨t_del_temp⟩
98         | ⟨t_name⟩
99     ) ;
100
101 ⟨t_del_descr⟩ :
102     DESCRIPTION FOR ⟨t_name⟩
103     (
104         WITH LANGUAGE '=' ⟨lang_name⟩
105     )? ;
106
107 ⟨t_del_temp⟩ :
108     TEMPLATE FOR ⟨t_name⟩
109     (
110         WITH
111         (
112             ⟨temp_elem_meta⟩
113         )+
114         (
115             ',' ⟨temp_elem_meta⟩
116         )*
117     )? ;
```

Listing A.4: Syntax of the pattern definition statements

```
1 ⟨cp_stmt⟩ :
2     PATTERN
3     (
4         ⟨p_def⟩
5         | ⟨p_descr⟩
6         | ⟨p_temp⟩
7     ) ;
8
9 ⟨p_def⟩ :
10     ⟨p_name⟩ WITH
11     (
12         PARAMETERS ⟨param_decl⟩+ END PARAMETERS ';'
13     )?
14     (
15         DERIVED ELEMENTS ⟨derv_decl⟩+ END DERIVED ELEMENTS ';'
16     )?
17     (
```

```
18          LOCAL CUBES ⟨lc_decl⟩+ END LOCAL CUBES ';'
19      )?
20      (
21          CONSTRAINTS ⟨cstr_decl⟩+ END CONSTRAINTS ';'
22      )?
23      END PATTERN ;
24
25  ⟨p_descr⟩ :
26      DESCRIPTION FOR ⟨p_name⟩ WITH
27      (
28          (
29              LANGUAGE '=' ⟨lang_name⟩
30              | ALIAS    '=' ⟨p_name⟩
31                (
32                    ',' ⟨p_name⟩
33                )*
34              | PROBLEM  '=' ⟨prob_txt⟩
35              | SOLUTION '=' ⟨sol_txt⟩
36              | EXAMPLE  '=' ⟨ex_txt⟩
37              | RELATED  '=' ⟨p_name⟩
38                (
39                    ',' ⟨p_name⟩
40                )*
41          ) ';'
42      )+
43      END PATTERN DESCRIPTION ;
44
45  ⟨p_temp⟩ :
46      TEMPLATE FOR ⟨p_name⟩ WITH
47          ⟨temp_elem⟩+
48      END PATTERN TEMPLATE ;
49
50  ⟨param_decl⟩ :
51      ⟨var_decl⟩ ';' ;
52
53  ⟨derv_decl⟩ :
54      ⟨var_decl⟩ '<=' ⟨elem_exp⟩ '.'
55      (
56          ⟨elem_exp⟩
57          | RETURNS
58      )';' ;
59
60  ⟨lc_decl⟩ :
61      ⟨type_lc_decl⟩
62      | ⟨dom_lc_decl⟩
63      | ⟨prop_lc_decl⟩ ;
```

```
64
65 ⟨type_lc_decl⟩ :
66     ⟨const_exp⟩ ':' ⟨type⟩ ;
67
68 ⟨dom_lc_decl⟩ :
69     ⟨elem_exp⟩ '.' ⟨elem_exp⟩ ':' ⟨elem_exp⟩ ;
70
71 ⟨prop_lc_decl⟩ :
72     ⟨elem_exp⟩ HAS ⟨m_prop_type⟩ ⟨elem_exp⟩ ;
73
74 ⟨return_cstr_decl⟩ :
75     ⟨elem_exp⟩ RETURNS ⟨elem_exp⟩ ;
76
77 ⟨app_cstr_decl⟩ :
78     ⟨elem_exp⟩ IS_APPLICABLE_TO
79     (
80         ⟨elem_exp⟩
81         | '(' ⟨elem_exp⟩ ',' ⟨elem_exp⟩ ')'
82     ) ;
83
84 ⟨p_name⟩ :
85     ⟨path_exp⟩;
```

Listing A.5: Syntax of the pattern usage statements

```
1 ⟨i_stmt⟩ :
2     INSTANTIATE ⟨p_inst⟩ ;
3
4 ⟨g_stmt⟩ :
5     GROUND ⟨p_grnd⟩ ;
6
7 ⟨x_stmt⟩ :
8     EXECUTE ⟨p_exec⟩ ;
9
10 ⟨p_inst⟩ :
11     PATTERN ⟨p_name⟩ AS ⟨p_name⟩ WITH BINDINGS? ⟨binding_exp⟩
12     (
13         ',' ⟨binding_exp⟩
14     )*
15     (
16         END BINDINGS
17     )? ;
18
19 ⟨p_grnd⟩ :
20     PATTERN ⟨p_name⟩ FOR ⟨mdm_name⟩ USING ⟨voc_name⟩ ;
21
```

```
22 ⟨p_exec⟩ :
23     PATTERN ⟨p_name⟩ FOR ⟨mdm_name⟩ USING ⟨voc_name⟩
24     (
25         WITH TEMPLATE
26         (
27             ⟨temp_elem_meta⟩
28         )+
29         (
30             ',' ⟨temp_elem_meta⟩
31         )*
32     )? ;
33
34 ⟨p_del⟩ :
35     PATTERN
36     (
37         ⟨p_del_descr⟩
38         | ⟨p_del_temp⟩
39         | ⟨p_name⟩
40     ) ;
41
42 ⟨p_del_descr⟩ :
43     DESCRIPTION FOR ⟨p_name⟩
44     (
45         WITH LANGUAGE '=' ⟨lang_name⟩
46     )? ;
47
48 ⟨p_del_temp⟩ :
49     TEMPLATE FOR ⟨p_name⟩
50     (
51         WITH
52         (
53             ⟨temp_elem_meta⟩
54         )+
55         (
56             ',' ⟨temp_elem_meta⟩
57         )*
58     )? ;
```

Listing A.6: Syntax of repository organization and search statements

```
1 ⟨cr_stmt⟩ :
2     ⟨r_exp⟩ ;
3
4 ⟨s_stmt⟩ :
5     SHOW ⟨path_exp⟩ ;
6
7 ⟨f_stmt⟩ :
```

```
8       SEARCH ⟨s_trgt⟩
9       (
10          IN ⟨path_exp⟩
11      )? ⟨s_exp⟩+ ;
12
13  ⟨s_trgt⟩ :
14      REPOSITORY
15      | CATALOGUE
16      | GLOSSARY
17      | MULTIDIMENSIONAL_MODEL
18      | PATTERN
19      | TERM
20      | CUBE
21      | DIMENSION ;
22
23  ⟨s_exp⟩ :
24      CONTAIN ⟨s_name⟩ IN
25      (
26          ⟨s_sct⟩
27          (
28              ',' ⟨s_sct⟩
29          )*
30      )+ ;
31
32  ⟨s_sct⟩ :
33      NAME
34      | LANGUAGE
35      | ALIAS
36      | PROBLEM
37      | SOLUTION
38      | EXAMPLE
39      | RELATED ;
40
41  ⟨r_del⟩ :
42      ⟨r_exp⟩ ;
43
44  ⟨r_exp⟩ :
45      (
46          REPOSITORY
47        | CATALOGUE
48        | GLOSSARY
49        | MULTIDIMENSIONAL_MODEL
50      ) ⟨s_name⟩ ;
51
52  ⟨s_name⟩ :
53      ⟨path_exp⟩ ;
```

Appendix B

Domain-Independent Pattern Catalog

The domain-independent patterns identified in the course of the agriProKnow research and development projects are presented in the following as a domain-independent catalog of patterns. To this end, we present the domain-independent patterns in two groups. The group of basic patterns comprises the non-comparative pattern (Figure B.1 to Figure B.4), while the group of comparative patterns comprises the homogeneous subset-baseset comparison (Figure B.5 to Figure B.9), the homogeneous subset-complement comparison (Figure B.10 to Figure B.14), the homogeneous subset-subset comparison (Figure B.15 to Figure B.19), and the heterogeneous subset-subset comparison (Figure B.20 to Figure B.24).

B.1 Non-Comparative Pattern

Figure B.1: Aliases, problem, solution, context, and related patterns of the non-comparative pattern

Non-Comparative Query

Also Known As

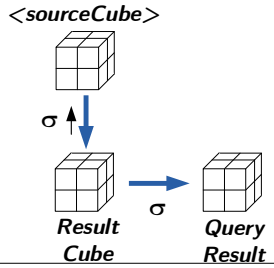
Grouping Query, Aggregation Query, Simple Aggregation

Problem

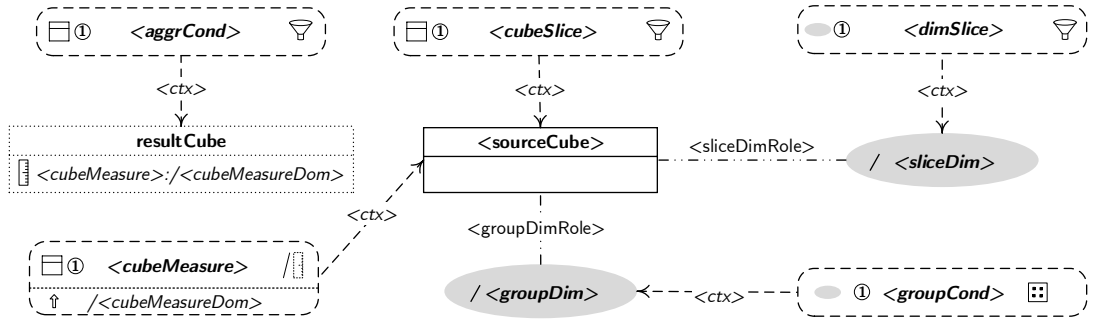
Retrieve aggregated measure values for one specific group of facts from a single source cube, which should be aggregated.

Solution

From the ⟨sourceCube⟩, select the set of relevant facts using the unary cube predicate ⟨cubeSlice⟩. In addition, the relevant facts are restricted by specifying a unary dimension predicate ⟨dimSlice⟩ over the dimension ⟨sliceDim⟩ that is referenced by the cube's dimension role ⟨sliceDimRole⟩. Perform a roll-up according to the ⟨groupCond⟩ dimension grouping over the ⟨groupDim⟩ dimension that is referenced by the cube's dimension role ⟨groupDimRole⟩. Finally, compute a calculated measure ⟨cubeMeasure⟩, then return only the results satisfying the ⟨aggrCond⟩ cube predicate.



Context



Related Patterns

Milking Aggregation, Revenue Aggregation

Figure B.2: A template and an example for non-comparative pattern (continued from Figure B.1)

Template (Data Model: Relational, Variant: Star Schema, Language: SQL, Dialect: ORACLEv11)

```
1  WITH resultCube AS (
2    SELECT $expr(<groupCond>, groupDim),
3           $expr(<cubeMeasure>, c) AS <cubeMeasure>
4    FROM   <sourceCube> c
5           JOIN <sliceDim> sliceDim ON
6              c.<sliceDimRole> = sliceDim.$dimKey(<sliceDim>)
7           JOIN <groupDim> groupDim ON
8              c.<groupDimRole> = groupDim.$dimKey(<groupDim>)
9    WHERE  $expr(<cubeSlice>, c) AND
10          $expr(<dimSlice>, sliceDim)
11   GROUP BY $expr(<groupCond>, groupDim)
12 )
13
14 SELECT   *
15 FROM     resultCube rc
16 WHERE    $expr(<aggrCond>, rc)
```

Example

Consider dairy company *Happy Milk*, which consists of three farms located at different sites, tending to a herd of about a thousand animals in total, half of which are Holstein breed, the other half Jersey breed. Happy Milk runs its farms as precision dairy farming operations aiming to increase efficiency and reduce losses due to animal illness through monitoring the animals' health status and proactively induce necessary treatments.

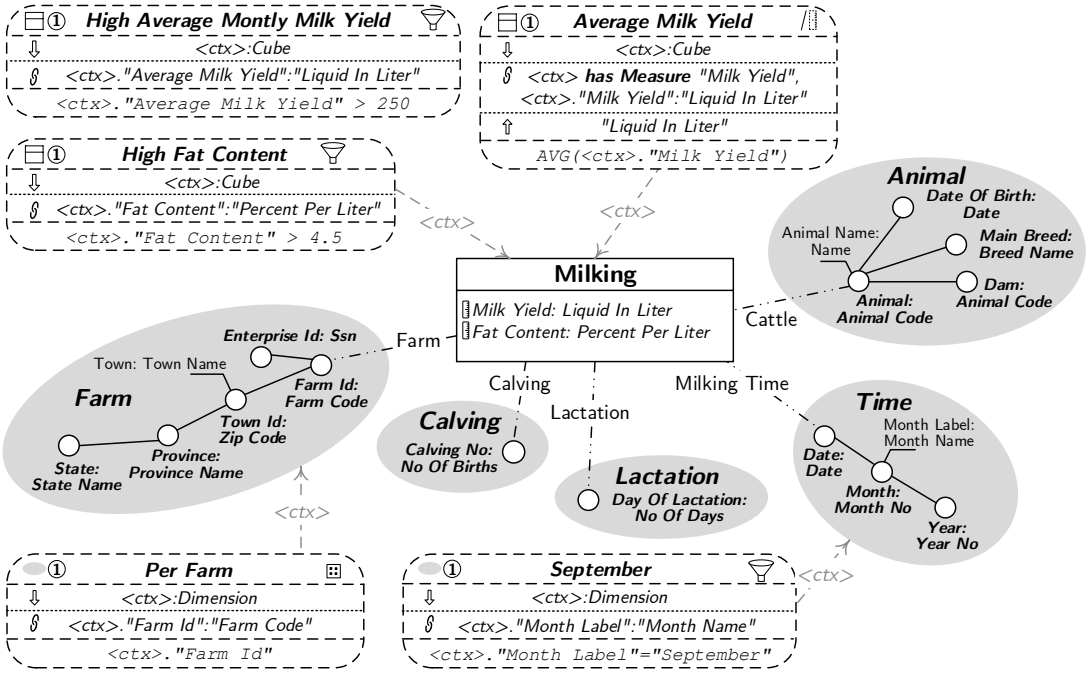


Figure B.3: Example for non-comparative pattern (continued from Figure B.2)

In precision dairy farming, animals are typically monitored using a wide range of sensors capturing a multitude of data concerning, e.g., micro-climate in the barns (temperature and moisture, among others), animal movement within farms, food consumption, milk yield and composition of the produced milk, and information about calvings. Data gathered by various sensors are integrated into a relational data warehouse system realized using a star schema which is conceptually represented using entities and properties in a multidimensional model.

Happy Milk's `Milking` cube captures for analysis purposes aggregated data about milking events quantified by the measures `Milk Yield` (in liters) and `Fat Content` (as percentage); value sets `Liquid In Liter` and `Percent Per Liter` are the types of `Milk Yield` and `Fat Content`, respectively. Hence, the cube records milk yield and fat content on a daily basis per animal and farm as well as calving period and phase in the animal's lactation cycle.

The management of Happy Milk decides that the production of high-fat milk should be increased, since a higher price can be obtained for it. The aim is to identify farms producing a high quantity of high-fat milk which can then be examined with regard to feeding and husbandry conditions in order to derive recommendations for action. Therefore, the average quantity of high-fat milk per farm produced in September has to be calculated, considering only high monthly average milk yields.

To obtain an executable OLAP query based on the non-comparative query pattern, a pattern user (BI user) simply binds names of available eMDM elements for each pattern parameter during pattern instantiation. In the Happy Milk scenario, the cube name `Milking`, which refers to the cube that tracks milk yield of farms in the Happy Milk eMDM, is bound to ⟨`sourceCube`⟩. The cube is restricted by binding the unary cube predicate name `High Fat Content` to the parameter ⟨`cubeSlice`⟩. The `Milking` cube in the Happy Milk eMDM has a dimension role property `Milking Time` that references a dimension `Time` as its domain and thus a time-specific restriction is applicable to this cube – `Milking Time` is bound to ⟨`sliceDimRole`⟩. The unary predicate name `September` is bound to the parameter ⟨`dimSlice`⟩ to restrict a dimension ⟨`sliceDim`⟩ referenced by the cube's dimension role `Milking Time`. The ⟨`groupCond`⟩ parameter is assigned the `Per Farm` dimension grouping to be applied on a dimension referenced by the cube's dimension role `Farm` that is bound to ⟨`groupDimRole`⟩. Finally, `Average Milk Yield` is bound to the unary calculated measure ⟨`cubeMeasure`⟩ indicating the measure to be calculated, while `High Average Monthly Milk Yield` is bound to the unary cube predicate ⟨`aggrCond`⟩ referring to the restrictions to be applied to the result cube.

Figure B.4: Example for non-comparative pattern (continued from Figure B.3)

With parameters for the non-comparative pattern bound as before, pattern grounding over the Happy Milk eMDM yields an applicable ground pattern that is executed resulting in the SQL query shown below, which returns an average milk yield per farm considering only milking events in September with a high fat content, showing only result of farms with a high monthly average milk yield.

```
1  WITH resultCube AS (
2    SELECT groupDim."Farm Id",
3           AVG(c."Milk Yield") AS "Average Milk Yield"
4    FROM   "Milking" c
5           JOIN "Time" sliceDim ON
6              c."Milking Time" = sliceDim."Date"
7           JOIN "Farm" groupDim ON
8              c."Farm" = groupDim."Farm Id"
9    WHERE  c."Fat Content" > 4.5 AND
10          sliceDim."Month Label" = "September"
11   GROUP BY groupDim."Farm Id"
12 )
13
14 SELECT  *
15 FROM    resultCube rc
16 WHERE   rc."Average Milk Yield" > 250
```

B.2 Homogeneous Subset-Baseset Comparison

Figure B.5: Aliases, problem, solution, and context of the homogeneous subset-baseset comparison pattern

Homogeneous Subset-Baseset Comparison

Also Known As

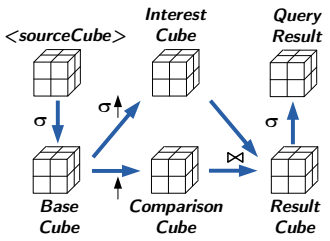
Group-Rest Comparison, Part-Whole Comparison

Problem

Aggregated measure values for one specific group of facts from one cube should be compared to aggregated measure values of another specific group of facts – of which it is a subset – by calculating a comparative measure.

Solution

From the ⟨sourceCube⟩, select the set of relevant facts using the unary cube predicate ⟨baseCubeSlice⟩ and the unary dimension predicate ⟨baseDimSlice⟩, which selects over the ⟨baseDim⟩ dimension referenced by the cube's dimension role ⟨dimRole⟩ – the result serves as the *base cube* for further analysis. From that base cube, select the facts of interest using conditions over the dimension role ⟨iDimRole⟩ according to the unary dimension predicate ⟨iDimSlice⟩, yielding the *interest cube*. The *comparison cube* is represented by the *base cube*, without further fact restrictions, hence, it includes the facts of the interest cube as well. Perform a roll-up for *interest cube* and *comparison cube* according to the ⟨groupCond⟩ dimension grouping over the ⟨joinDim⟩ dimension referenced by the dimension role ⟨joinDimRole⟩ and compute a unary calculated measure ⟨cubeMeasure⟩. To obtain the *result cube*, join the interest cube and comparison cube over the ⟨groupCond⟩ dimension grouping and compute a binary calculated measure ⟨compMeasure⟩, then return only the results satisfying the unary cube predicate ⟨compAggrCond⟩.



Context

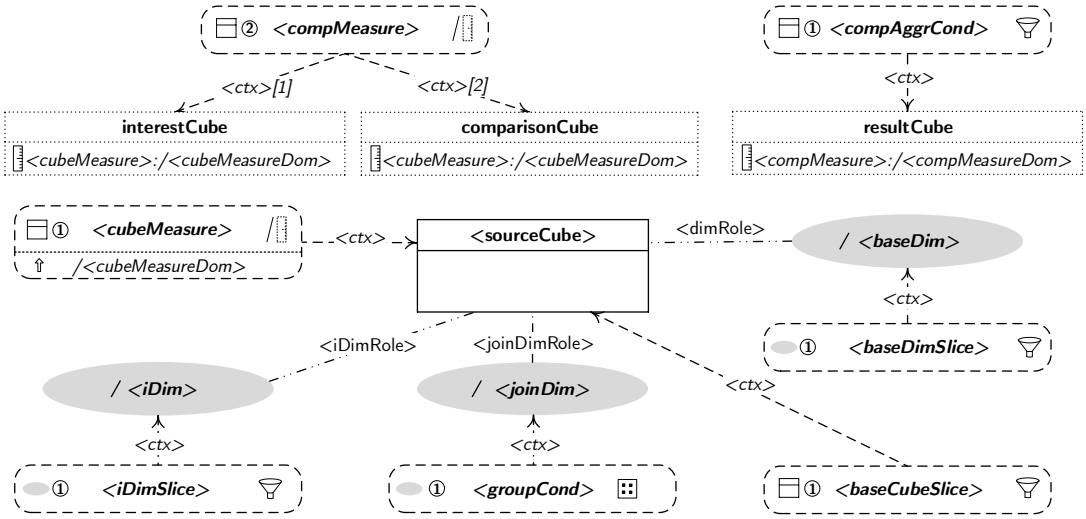


Figure B.6: A template and related patterns for subset-baseset comparison (continued from Figure B.5)

Template (Data Model: Relational, Variant: Star Schema, Language: SQL, Dialect: ORACLEv11)

```
WITH baseCube AS (
  SELECT  *
  FROM    <sourceCube> sc JOIN
          <baseDim> bd ON
             sc.<dimRole> = bd.$dimKey(<baseDim>)
  WHERE   $expr(<baseCubeSlice>, sc) AND
          $expr(<baseDimSlice>, bd)
),
interestCube AS (
  SELECT $expr(<groupCond>, jd),
         $expr(<cubeMeasure>, bc) AS <cubeMeasure>
  FROM   baseCube bc JOIN
         <iDim> cd ON
            bc.<iDimRole>=cd.$dimKey(<iDim>) JOIN
         <joinDim> jd ON
            bc.<joinDimRole>=jd.$dimKey(<joinDim>)
  WHERE  $expr(<iDimSlice>, cd)
  GROUP BY $expr(<groupCond>, jd)
),
comparisonCube AS (
  SELECT $expr(<groupCond>, jd),
         $expr(<cubeMeasure>, bc) AS <cubeMeasure>
  FROM   baseCube bc JOIN
         <joinDim> jd ON
            bc.<joinDimRole>=jd.$dimKey(<joinDim>)
  GROUP BY $expr(<groupCond>, jd)
),
resultCube AS (
  SELECT $expr(<groupCond>, ic),
         ic.<cubeMeasure> AS "Group of Interest",
         cc.<cubeMeasure> AS "Group of Comparison",
         $expr(<compMeasure>, ic, cc) AS <compMeasure>
  FROM   interestCube ic JOIN
         comparisonCube cc ON
            $expr(<groupCond>, ic) = $expr(<groupCond>, cc)
)

SELECT *
FROM   resultCube rc
WHERE  $expr(<compAggrCond>, rc)
```

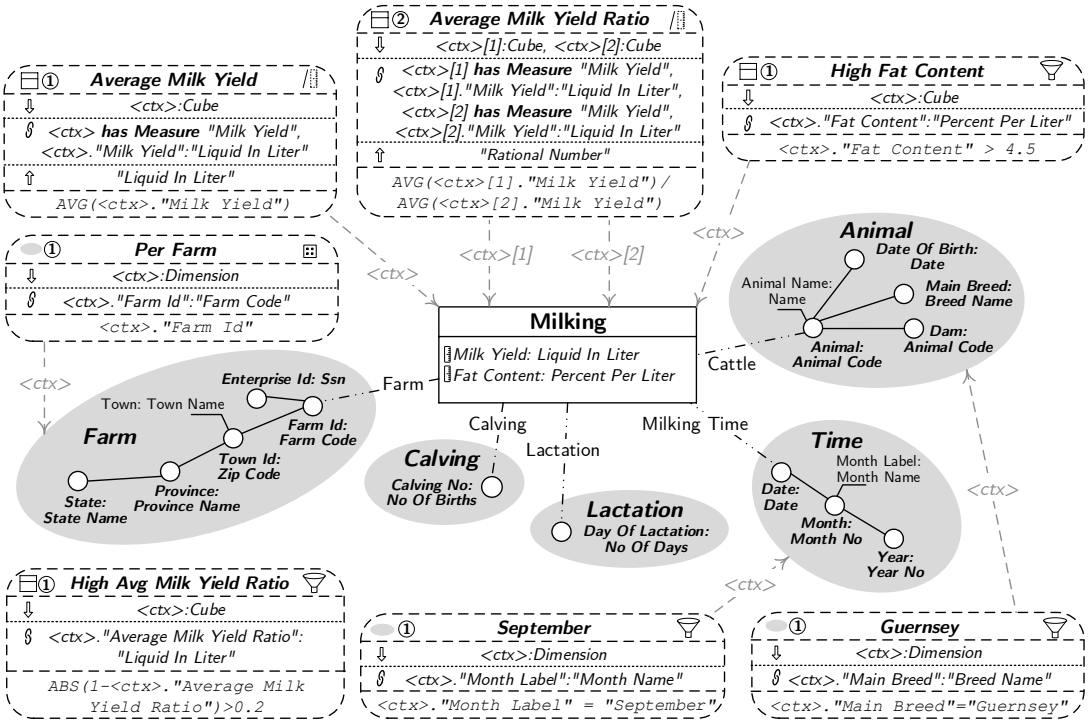
Related Patterns

Subset-Complement Comparison, Subset-Baseset Comparison of Milkings

Figure B.7: Example for subset-baseset comparison (continued from Figure B.6)

Example

Consider dairy company *Happy Milk*, which consists of three farms located at different sites, tending to a herd of about a thousand animals in total, half of which are Holstein breed, the other half Jersey breed. Happy Milk runs its farms as precision dairy farming operations aiming to increase efficiency and reduce losses due to animal illness through monitoring the animals' health status and proactively induce necessary treatments. In precision dairy farming, animals are typically monitored using a wide range of sensors capturing a multitude of data concerning, e.g., microclimate in the barns (temperature and moisture, among others), animal movement within farms, food consumption, milk yield and composition of the produced milk, and information about calvings. Data gathered by various sensors are integrated into a relational data warehouse system realized using a star schema which is conceptually represented using entities and properties in a multidimensional model.



Happy Milk's `Milking` cube captures for analysis purposes aggregated data about milking events quantified by the measures `Milk Yield` (in liters) and `Fat Content` (as percentage); value sets `Liquid In Liter` and `Percent Per Liter` are the types of `Milk Yield` and `Fat Content`, respectively. Hence, the cube records milk yield and fat content on a daily basis per animal and farm as well as calving period and phase in the animal's lactation cycle.

Figure B.8: Example for subset-baseset comparison (continued from Figure B.7)

Happy Milk added Guernsey cattle to the farm herds in August to increase milk production, especially the production of high-fat milk. As the Guernsey breed of cattle was bred to achieve a high milk yield with a high fat concentration, the newly acquired cattle should outperform the existing herd in terms of milk yield with high fat content. To verify this assumption, the ratio between the average milk yield of Guernsey cattle and the average milk yield of all cattle has to be calculated, considering only milking events with a high fat content in September. On a more abstract level, that query corresponds to subset-baseset comparison, i.e., the computation of a ratio between the measure values of a subset of facts and its baseset of facts, referring to a certain milking time – in this case, September –, the group of interest is distinguished according to the main breed – in this case, Guernsey.

To obtain an executable OLAP query based on the subset-baseset comparison pattern, a pattern user (BI user) simply binds names of available eMDM elements for each pattern parameter during pattern instantiation. In the Happy Milk scenario, the cube name `Milking`, which refers to the cube that tracks milk yield of farms in the Happy Milk eMDM, is bound to ⟨`sourceCube`⟩ – it is restricted by the unary cube predicate name `High Fat Content` that is bound to the parameter ⟨`baseCubeSlice`⟩. The unary dimension predicate name `September` is bound to the parameter ⟨`baseDimSlice`⟩ to restrict the dimension referenced by ⟨`dimRole`⟩ dimension role named `Milking Time`. The ⟨`iDimRole`⟩ is set to the name `Cattle` since it refers to a dimension which can be used for further restrictions. The group of fact that is to be compared to its baseset is specified by the unary dimension predicate `Guernsey` for the ⟨`iDimSlice`⟩. As the groups should be compared per farm the `Farm` dimension role is specified as the ⟨`joinDimRole`⟩. The ⟨`groupCond`⟩ parameter is assigned the `Per Farm` dimension grouping while ⟨`cubeMeasure`⟩ is assigned the `Average Milk Yield` derived cube measure. Finally, `Average Milk Yield` is bound to the unary calculated measure ⟨`cubeMeasure`⟩ indicating the measure to be calculated, while `High Average Monthly Milk Yield` is bound to the unary cube predicate ⟨`compAggrCond`⟩ referring to the restrictions to be applied to the result cube.

Figure B.9: Example for subset-baseset comparison (continued from Figure B.8)

With parameters for the subset-baseset comparison bound as before, pattern grounding over the Happy Milk eMDM yields an applicable ground pattern that is executed resulting in the SQL query shown below, which compares per farm the average milk yield with a high fat content of Guernsey cattle in September with the average milk yield with a high fat content of all cattle in September, showing only results where the difference in milk yields between the groups exceeds 20 percent.

```
WITH baseCube AS (
  SELECT *
  FROM   "Milking" sc JOIN
         "Time" bd ON
            sc."Milking Time" = bd."Date"
  WHERE  sc."Fat Content" > 4.5 AND
         bd."Month Label" = "September"
),
interestCube AS (
  SELECT jd."Farm Id",
         AVG(bc,"Milk Yield") AS "Average Milk Yield"
  FROM   baseCube bc JOIN
         "Farm" jd ON
            bc."Farm" = jd."Farm Id" JOIN
         "Animal" cd ON
            bc."Cattle" = cd."Animal"
  WHERE  cd."Main Breed" = "Guernsey"
  GROUP BY jd."Farm Id"
),
comparisonCube AS (
  SELECT jd."Farm Id",
         AVG(bc,"Milk Yield") AS "Average Milk Yield"
  FROM   baseCube bc JOIN
         "Farm" jd ON
            bc."Farm" = jd."Farm Id" JOIN
  GROUP BY jd."Farm Id"
),
resultCube AS (
  SELECT ic."Farm Id",
         ic."Average Milk Yield" AS "Group of Interest",
         cc."Average Milk Yield" AS "Group of Comparison",
         (ic."Average Milk Yield"/cc."Average Milk Yield") AS "Average
             Milk Yield Ratio"
  FROM   interestCube ic JOIN
         comparisonCube cc ON
            ic."Farm Id" = cc."Farm Id"
)

SELECT *
FROM   resultCube rc
WHERE  ABS(1-rc."Average Milk Yield Ratio")>0.2
```

B.3 Homogeneous Subset-Complement Comparison

Figure B.10: Aliases, problem, solution, and context of the homogeneous subset-complement comparison

Homogeneous Subset-Complement Comparison

Also Known As

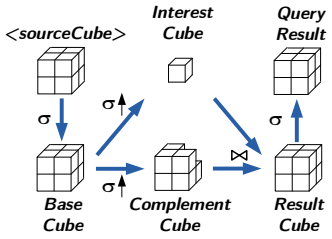
Group to Complement Comparison, Set to Other Comparison

Problem

Aggregated measure values of a particular group of facts should be compared with its complement group of facts, with both group of facts coming from one cube.

Solution

From the ⟨sourceCube⟩, select the set of relevant facts using the unary cube predicate ⟨baseCubeSlice⟩ and the unary dimension predicate ⟨baseDimSlice⟩, which selects over the ⟨baseDim⟩ dimension referenced by the cube's dimension role ⟨dimRole⟩ – the result serves as the *base cube* for further analysis. From that base cube, select *interest cube* using conditions over the dimension role ⟨iDimRole⟩ according to the dimension predicate ⟨iDimSlice⟩, yielding the *interest cube*. The *complement cube* is represented by the *base cube*, without including facts from the *interest cube*. Perform a roll-up for *interest cube* and *complement cube* according to the ⟨groupCond⟩ dimension grouping over the ⟨joinDim⟩ dimension referenced by the dimension role ⟨joinDimRole⟩ and compute a unary calculated measure ⟨cubeMeasure⟩. To obtain the *result cube*, join the *interest cube* and *complement cube* over the ⟨groupCond⟩ dimension grouping and compute a binary derived comparative measure ⟨compMeasure⟩, then return only the results satisfying the unary cube predicate ⟨compAggrCond⟩.



Context

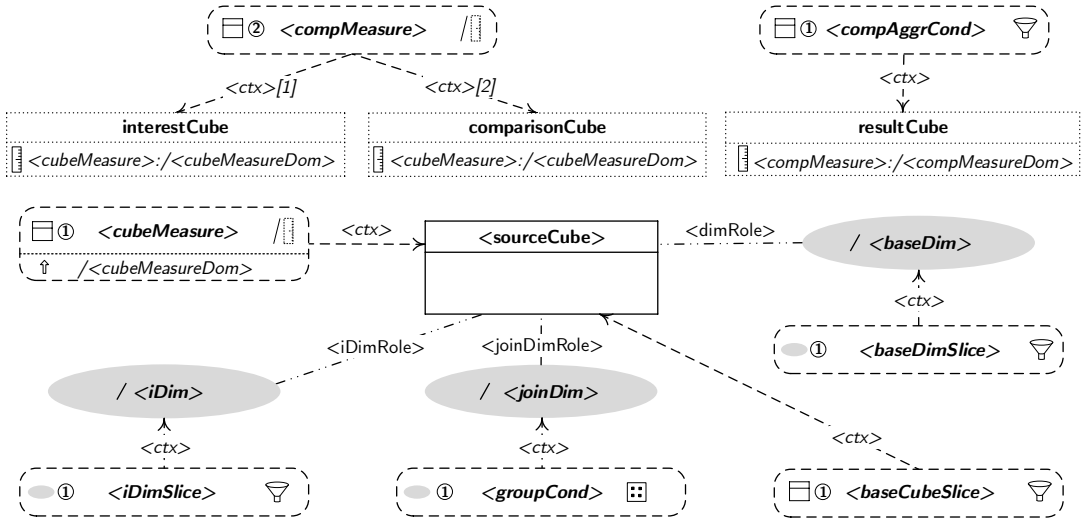


Figure B.11: A template and related patterns for subset-complement comparison (continued from Figure B.10)

Template (Data Model: Relational, Variant: Star Schema, Language: SQL, Dialect: ORACLEv11)

```
WITH baseCube AS (
  SELECT  *
  FROM    <sourceCube> sc
          JOIN <baseDim> bd ON
             sc.<dimRole> = bd.$dimKey(<baseDim>)
  WHERE   $expr(<baseCubeSlice>, sc) AND
          $expr(<baseDimSlice>, bd)
),
interestCube AS (
  SELECT $expr(<groupCond>, jd),
         $expr(<cubeMeasure>, bc) AS <cubeMeasure>
  FROM   baseCube bc
         JOIN <iDim> cd ON
            bc.<iDimRole>=cd.$dimKey(<iDim>)
         JOIN <joinDim> jd ON
            bc.<joinDimRole>=jd.$dimKey(<joinDim>)
  WHERE  $expr(<iDimSlice>, cd)
  GROUP BY $expr(<groupCond>, jd)
),
complementCube AS (
  SELECT $expr(<groupCond>, jd),
         $expr(<cubeMeasure>, bc) AS <cubeMeasure>
  FROM   baseCube bc
         JOIN <iDim> cd ON
            bc.<iDimRole>=cd.$dimKey(<iDim>)
         JOIN <joinDim> jd ON
            bc.<joinDimRole>=jd.$dimKey(<joinDim>)
  WHERE  bc.<iDimRole> NOT IN
            (SELECT $dimKey(<iDim>)
             FROM   <iDim>
             WHERE  $expr(<iDimSlice>, <iDim>))
  GROUP BY $expr(<groupCond>, jd)
),
resultCube AS (
  SELECT $expr(<groupCond>, ic),
         ic.<cubeMeasure> AS "Group of Interest",
         cc.<cubeMeasure> AS "Group of Comparison",
         $expr(<compMeasure>, ic, cc) AS <compMeasure>
  FROM   interestCube ic
         JOIN complementCube cc ON
            $expr(<groupCond>, ic) = $expr(<groupCond>, cc)
)
SELECT *
FROM   resultCube rc
WHERE  $expr(<compAggrCond>, rc)
```

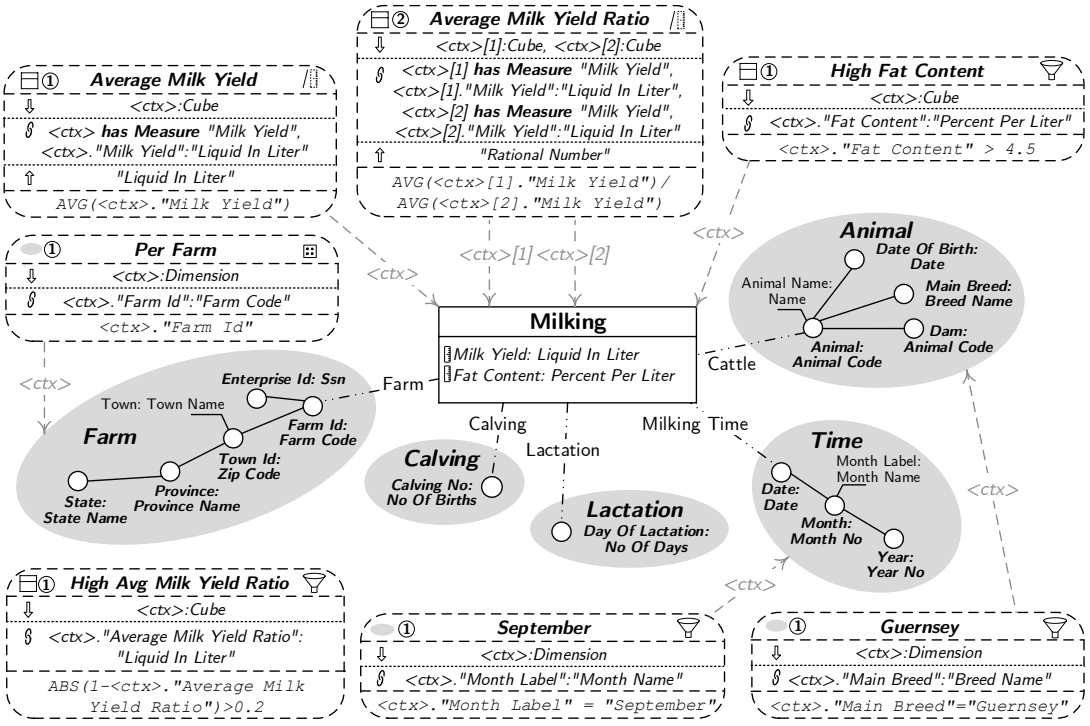
Related Patterns

Homogeneous Subset-Baseset Comparison, Subset-Complement Comparison of Milkings

Figure B.12: Example for subset-complement comparison (continued from Figure B.11)

Example

Consider dairy company *Happy Milk*, which consists of three farms located at different sites, tending to a herd of about a thousand animals in total, half of which are Holstein breed, the other half Jersey breed. Happy Milk runs its farms as precision dairy farming operations aiming to increase efficiency and reduce losses due to animal illness through monitoring the animals' health status and proactively induce necessary treatments. In precision dairy farming, animals are typically monitored using a wide range of sensors capturing a multitude of data concerning, e.g., microclimate in the barns (temperature and moisture, among others), animal movement within farms, food consumption, milk yield and composition of the produced milk, and information about calvings. Data gathered by various sensors are integrated into a relational data warehouse system realized using a star schema which is conceptually represented using entities and properties in a multidimensional model.



Happy Milk's `Milking` cube captures for analysis purposes aggregated data about milking events quantified by the measures `Milk Yield` (in liters) and `Fat Content` (as percentage); value sets `Liquid In Liter` and `Percent Per Liter` are the types of `Milk Yield` and `Fat Content`, respectively. Hence, the cube records milk yield and fat content on a daily basis per animal and farm as well as calving period and phase in the animal's lactation cycle.

Figure B.13: Example for subset-complement comparison (continued from Figure B.12)

Happy Milk added Guernsey cattle to the farm herds in August to increase milk production, especially the production of high-fat milk. As the Guernsey breed of cattle was bred to achieve a high milk yield with a high fat concentration, the newly acquired cattle should outperform the existing herd in terms of milk yield with high fat content. To verify this assumption, the ratio between the average milk yield of Guernsey cattle and the average milk yield of all other cattle has to be calculated, considering only milking events with a high fat content in September. On a more abstract level, that query corresponds to subset-complement comparison, i.e., the computation of a ratio between the measure values of a subset of facts and its complement set of facts, referring to a certain milking time – in this case, September –, the group of interest is distinguished according to the main breed – in this case, Guernsey.

To obtain an executable OLAP query based on the subset-complement comparison pattern, a pattern user (BI user) simply binds names of available eMDM elements for each pattern parameter during pattern instantiation. In the Happy Milk scenario, the cube name `Milking`, which refers to the cube that tracks milk yield of farms in the Happy Milk eMDM, is bound to ⟨`sourceCube`⟩ – it is restricted by the unary cube predicate name `High Fat Content` that is bound to the parameter ⟨`baseCubeSlice`⟩. The unary dimension predicate name `September` is bound to the parameter ⟨`baseDimSlice`⟩ to restrict the dimension referenced by ⟨`dimRole`⟩ dimension role named `Milking Time`. The ⟨`iDimRole`⟩ is set to the name `Cattle` since it refers to a dimension which can be used for further restrictions. The group of fact that is to be compared to its baseset is specified by the `Guernsey` dimension predicate for the ⟨`iDimSlice`⟩. As the groups should be compared per farm the `Farm` dimension role is specified as the ⟨`joinDimRole`⟩. The ⟨`groupCond`⟩ parameter is assigned the `Per Farm` dimension grouping while ⟨`cubeMeasure`⟩ is assigned the `Average Milk Yield` derived cube measure. Finally, `Average Milk Yield` is bound to the unary calculated measure ⟨`cubeMeasure`⟩ indicating the measure to be calculated, while `High Average Monthly Milk Yield` is bound to the unary cube predicate ⟨`compAggrCond`⟩ referring to the restrictions to be applied to the result cube.

With parameters for the subset-complement comparison bound as before, pattern grounding over the Happy Milk eMDM yields an applicable ground pattern that is executed resulting in the SQL query shown below, which compares per farm the average milk yield with a high fat content of Guernsey cattle in September with the average milk yield with a high fat content of all cattle other than Guernsey in September, showing only results where the difference in milk yields between the groups exceeds 20 percent.

Figure B.14: Example for subset-complement comparison (continued from Figure B.13)

```
WITH baseCube AS (
  SELECT *
  FROM   "Milking" sc
         JOIN "Time" sd ON
            sc."Milking Time" = sd."Date"
  WHERE  sc."Fat Content" > 4.5 AND
         sd."Month Label" = "September"
),
interestCube AS (
  SELECT jd."Farm Id",
         AVG(bc,"Milk Yield") AS "Average Milk Yield"
  FROM   baseCube bc
         JOIN "Farm" jd ON
            bc."Farm" = jd."Farm Id"
         JOIN "Animal" cd ON
            bc."Cattle" = cd."Animal"
  WHERE  cd."Main Breed" = "Guernsey"
  GROUP BY jd."Farm Id"
),
complementCube AS (
  SELECT jd."Farm Id",
         AVG(bc,"Milk Yield") AS "Average Milk Yield"
  FROM   baseCube bc
         JOIN "Farm" jd ON
            bc."Farm" = jd."Farm Id"
         JOIN "Animal" cd ON
            bc."Cattle" = cd."Animal"
  WHERE  bc."Cattle" NOT IN
            (SELECT "Animal"
             FROM   "Animal"
             WHERE  "Animal"."Main Breed"="Guernsey")
  GROUP BY jd."Farm Id"
),
resultCube AS (
  SELECT ic."Farm Id",
         ic."Average Milk Yield" AS "Group of Interest",
         cc."Average Milk Yield" AS "Group of Comparison",
         (ic."Average Milk Yield"/cc."Average Milk Yield") AS "Average
             Milk Yield Ratio"
  FROM   interestCube ic
         JOIN complementCube cc ON
            ic."Farm Id" = cc."Farm Id"
)

SELECT *
FROM   resultCube rc
WHERE  ABS(1-rc."Average Milk Yield Ratio")>0.2
```

B.4 Homogeneous Subset-Subset Comparison

Figure B.15: Aliases, problem, solution, and context of the homogeneous subset-subset comparison

Homogeneous Subset-Subset Comparison

Also Known As

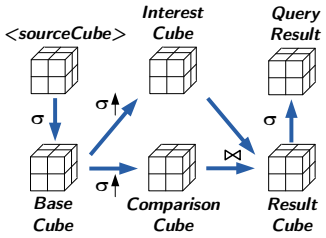
Group-Group Comparison, Set to Set Comparison

Problem

Retrieve aggregated measure values for two specified groups of facts from a single source cube, which should be compared by calculating a comparative measure.

Solution

From the ⟨sourceCube⟩, select the set of relevant facts using the unary cube predicate ⟨baseCubeSlice⟩ and the unary dimension predicate ⟨baseDimSlice⟩, which selects over the dimension referenced by the ⟨dimRole⟩ – the result serves as the *base cube* for further analysis. From that base cube, select *interest cube* and *comparison cube* using conditions over the dimension role ⟨compDimRole⟩ according to the unary dimension predicates ⟨iDimSlice⟩ and ⟨cDimSlice⟩, respectively. Perform a roll-up for *interest cube* and *comparison cube* according to the ⟨groupCond⟩ dimension grouping over the ⟨joinDim⟩ dimension referenced by the dimension role ⟨joinDimRole⟩ and compute a unary calculated measure ⟨cubeMeasure⟩. To obtain the *result cube*, join the *interest cube* and *comparison cube* over the ⟨groupCond⟩ dimension grouping and compute a binary calculated measure ⟨compMeasure⟩, then return only the results satisfying the unary cube predicate ⟨compAggrCond⟩.



Context

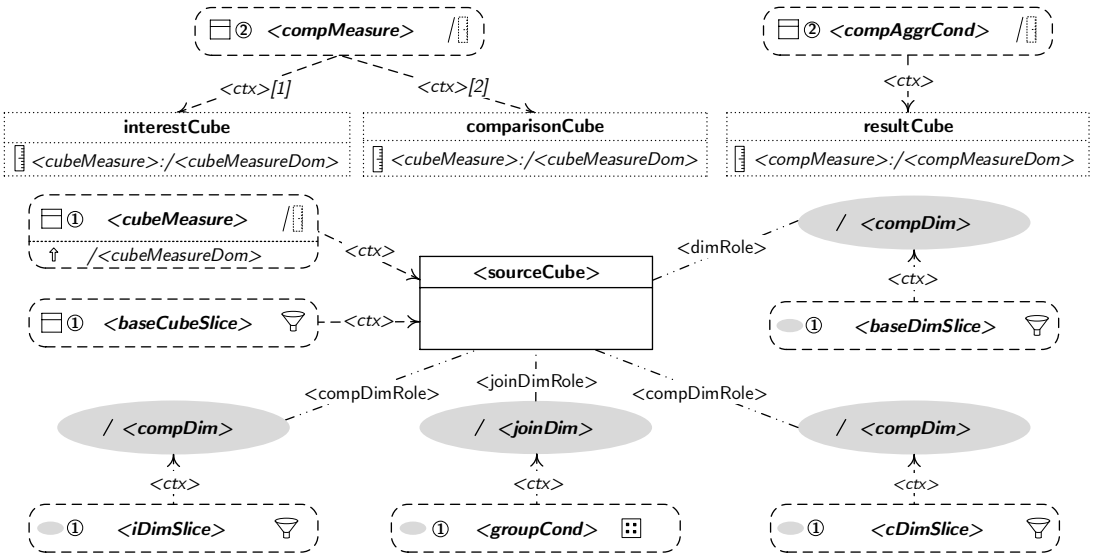


Figure B.16: A template and related patterns for homogeneous subset-subset comparison (continued from Figure B.15)

Template (Data Model: Relational, Variant: Star Schema, Language: SQL, Dialect: ORACLEv11)

```
WITH baseCube AS (
  SELECT  *
  FROM    <sourceCube> sc
          JOIN <baseDim> bd ON
             sc.<dimRole> = bd.$dimKey(<baseDim>)
  WHERE   $expr(<baseCubeSlice>, sc) AND
          $expr(<baseDimSlice>, bd)
),
interestCube AS (
  SELECT $expr(<groupCond>, jd),
         $expr(<cubeMeasure>, bc) AS <cubeMeasure>
  FROM   baseCube bc
         JOIN <compDim> cd ON
            bc.<compDimRole>=cd.$dimKey(<compDim>)
         JOIN <joinDim> jd ON
            bc.<joinDimRole>=jd.$dimKey(<joinDim>)
  WHERE  $expr(<iDimSlice>, cd)
  GROUP BY $expr(<groupCond>, jd)
),
comparisonCube AS (
  SELECT $expr(<groupCond>, jd),
         $expr(<cubeMeasure>, bc) AS <cubeMeasure>
  FROM   baseCube bc
         JOIN <compDim> cd ON
            bc.<compDimRole>=cd.$dimKey(<compDim>)
         JOIN <joinDim> jd ON
            bc.<joinDimRole>=jd.$dimKey(<joinDim>)
  WHERE  $expr(<cDimSlice>, cd)
  GROUP BY $expr(<groupCond>, jd)
),
resultCube AS (
  SELECT $expr(<groupCond>, ic),
         ic.<cubeMeasure> AS "Group of Interest",
         cc.<cubeMeasure> AS "Group of Comparison",
         $expr(<compMeasure>, ic, cc) AS <compMeasure>
  FROM   interestCube ic
         JOIN comparisonCube cc ON
            $expr(<groupCond>, ic) = $expr(<groupCond>, cc)
)
SELECT *
FROM   resultCube rc
WHERE  $expr(<compAggrCond>, rc)
```

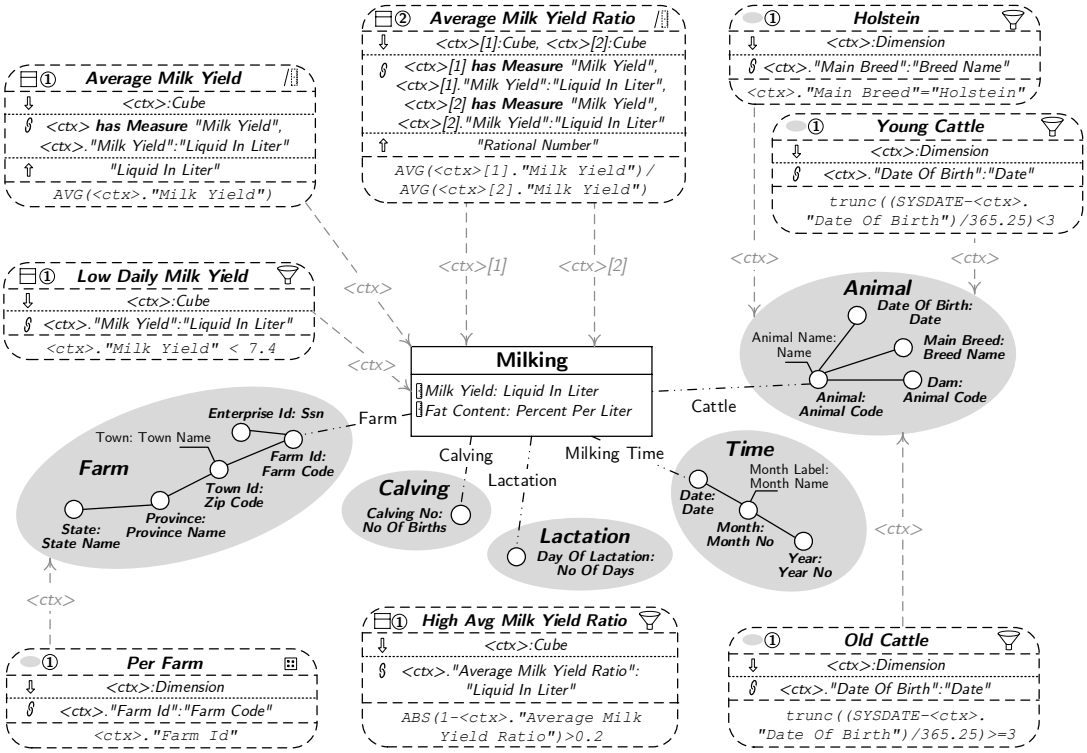
Related Patterns

Breed-Specific Subset-Subset Comparison

Figure B.17: Example for homogeneous subset-subset comparison (continued from Figure B.16)

Example

Consider dairy company *Happy Milk*, which consists of three farms located at different sites, tending to a herd of about a thousand animals in total, half of which are Holstein breed, the other half Jersey breed. Happy Milk runs its farms as precision dairy farming operations aiming to increase efficiency and reduce losses due to animal illness through monitoring the animals' health status and proactively induce necessary treatments.



In precision dairy farming, animals are typically monitored using a wide range of sensors capturing a multitude of data concerning, e.g., microclimate in the barns (temperature and moisture, among others), animal movement within farms, food consumption, milk yield and composition of the produced milk, and information about calvings. Data gathered by various sensors are integrated into a relational data warehouse system realized using a star schema which is conceptually represented using entities and properties in a multidimensional model.

Happy Milk's `Milking` cube captures for analysis purposes aggregated data about milking events quantified by the measures `Milk Yield` (in liters) and `Fat Content` (as percentage); value sets `Liquid In Liter` and `Percent Per Liter` are the types of `Milk Yield` and `Fat Content`, respectively. Hence, the cube records milk yield and fat content on a daily basis per animal and farm as well as calving period and phase in the animal's lactation cycle.

Figure B.18: Example for homogeneous subset-subset comparison (continued from Figure B.17)

Happy Milk detects a drop in the milk yield of Holstein cattle. Multiple causes can lead to a decreased milk production but, first of all, the affected Holstein population needs to be identified. A BI user may start the investigation with the computation of the ratio between the average milk yield of young Holstein cattle and the average milk yield of old Holstein cattle, considering only animals that have and low daily milk yield. On a more abstract level, that query corresponds to subset-subset comparison, i.e., the computation of a ratio between measure values for two subsets of facts, referring to a certain breed – in this case, Holstein –, each group distinguished according to specified characteristics – in this case, the date of birth.

To obtain an executable OLAP query based on the subset-subset comparison pattern, the pattern user (BI user) simply binds names of available eMDM elements for each pattern parameter during pattern instantiation. In the Happy Milk scenario, the cube name `Milking`, which refers to the cube that tracks milk yield of farms in the Happy Milk eMDM, is bound to ⟨sourceCube⟩. The `Milking` cube in the Happy Milk eMDM has a dimension role `Cattle` that references a dimension `Animal`, i.e., `Cattle` is bound to ⟨dimRole⟩. The unary dimension predicate name `Holstein` is bound to the parameter ⟨baseDimSlice⟩ whereas the uanry cube predicate name `Low Daily Milk Yield` is bound to the parameter ⟨baseCubeSlice⟩. The ⟨compDimRole⟩ is set to the name `Cattle` since it refers to a dimension which can be used for further restrictions. The groups of facts that are to be compared are specified by the unary dimension predicate `Young Cattle` for the ⟨iDimSlice⟩ and the unary dimension predicate `Old Cattle` for the ⟨cDimSlice⟩. As the groups should be compared per farm the `Farm` dimension role is specified as the ⟨joinDimRole⟩. The ⟨groupCond⟩ parameter is assigned the `Per Farm` dimension grouping while ⟨cubeMeasure⟩ is assigned the unary calculated measure `Average Milk Yield`. Finally, `Average Milk Yield Ratio` is bound to the binary calculated measure ⟨compMeasure⟩ indicating the measure to be calculated, while `High Avg Milk Yield Ratio` is bound to the unary cube predicate ⟨compAggrCond⟩ referring to the restrictions to be applied to the result cube.

Figure B.19: Example for homogeneous subset-subset comparison (continued from Figure B.18)

With parameters for the breed-specific subset-subset comparison bound as before, pattern grounding over the Happy Milk eMDM yields an applicable ground pattern that is executed resulting in the SQL query shown below, which compares per farm the average milk yield of cattle younger than three years that have a low daily milk yield with cattle older than three years that have a low daily milk yield, showing only results where the difference in milk yields between the groups exceeds 20 percent.

```
WITH baseCube AS (
  SELECT *
  FROM   "Milking" sc
         JOIN "Animal" sd ON
            sc."Cattle" = sd."Animal"
  WHERE  sc."Milk Yield" < 7.4 AND
         sd."Main Breed" = "Holstein"
),
interestCube AS (
  SELECT jd."Farm Id",
         AVG(bc,"Milk Yield") AS "Average Milk Yield"
  FROM   baseCube bc
         JOIN "Farm" jd ON
            bc."Farm" = jd."Farm Id"
         JOIN "Animal" cd ON
            bc."Cattle" = cd."Animal"
  WHERE  trunc((SYSDATE-cd."Date Of Birth")/365.25)<3
  GROUP BY jd."Farm Id"
),
comparisonCube AS (
  SELECT jd."Farm Id",
         AVG(bc,"Milk Yield") AS "Average Milk Yield"
  FROM   baseCube bc
         JOIN "Farm" jd ON
            bc."Farm" = jd."Farm Id"
         JOIN "Animal" cd ON
            bc."Cattle" = cd."Animal"
  WHERE  trunc((SYSDATE-cd."Date Of Birth")/365.25)>=3
  GROUP BY jd."Farm Id"
),
resultCube AS (
  SELECT ic."Farm Id",
         ic."Average Milk Yield" AS "Group of Interest",
         cc."Average Milk Yield" AS "Group of Comparison",
         (ic."Average Milk Yield"/cc."Average Milk Yield") AS "Average
             Milk Yield Ratio"
  FROM   interestCube ic
         JOIN comparisonCube cc ON
            ic."Farm Id" = cc."Farm Id"
)
SELECT *
FROM   resultCube rc
WHERE  ABS(1-rc."Average Milk Yield Ratio")>0.2
```

B.5 Heterogeneous Subset-Subset Comparison

Figure B.20: Aliases, problem, and solution of the heterogeneous subset-subset comparison pattern

Heterogeneous Subset-Subset Comparison

Also Known As

Heterogeneous Group-Group Comparison

Problem

Retrieve aggregated measure values for two specified groups of facts from two different source cubes sharing a common dimension and granularity, which should be compared by calculating a comparative measure.

Solution

From the ⟨iSourceCube⟩, select the relevant facts using the unary dimension predicate ⟨iDimSlice⟩ over the dimension referenced by the dimension role ⟨iCompDimRole⟩ followed by performing a roll-up according to the ⟨groupCond⟩ dimension grouping over the dimension referenced by the ⟨iJoinDimRole⟩ dimension role and compute a unary calculated measure ⟨iCubeMeasure⟩ – the result serves as the *interest cube* for further analysis. Analogously, from the ⟨cSourceCube⟩, select the relevant facts using the unary dimension predicate ⟨cDimSlice⟩ over the dimension referenced by the dimension role ⟨cCompDimRole⟩ followed by performing a roll-up according to the ⟨groupCond⟩ dimension grouping over the dimension referenced by the ⟨cJoinDimRole⟩ dimension role and compute a unary calculated measure ⟨cCubeMeasure⟩ – the result serves as the *comparison cube* for further analysis. Although, the dimension role names of ⟨iJoinDimRole⟩ and ⟨cJoinDimRole⟩ can differ, they should reference the same dimension allowing a reasonable comparison. To obtain the *result cube*, join the *interest cube* and *comparison cube* over the ⟨groupCond⟩ combining the ⟨iJoinDim⟩ and ⟨cJoinDim⟩ dimension referenced by the ⟨iJoinDimRole⟩ and the ⟨cJoinDimRole⟩ dimension roles. Finally, compute a binary calculated measure ⟨compMeasure⟩, then return only the results satisfying the unary cube predicate ⟨compAggrCond⟩ cube predicate.

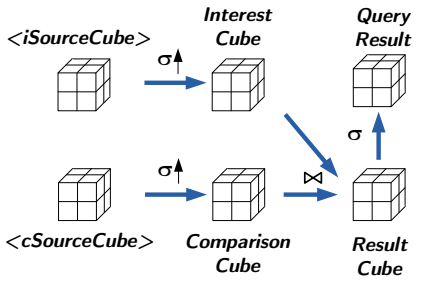
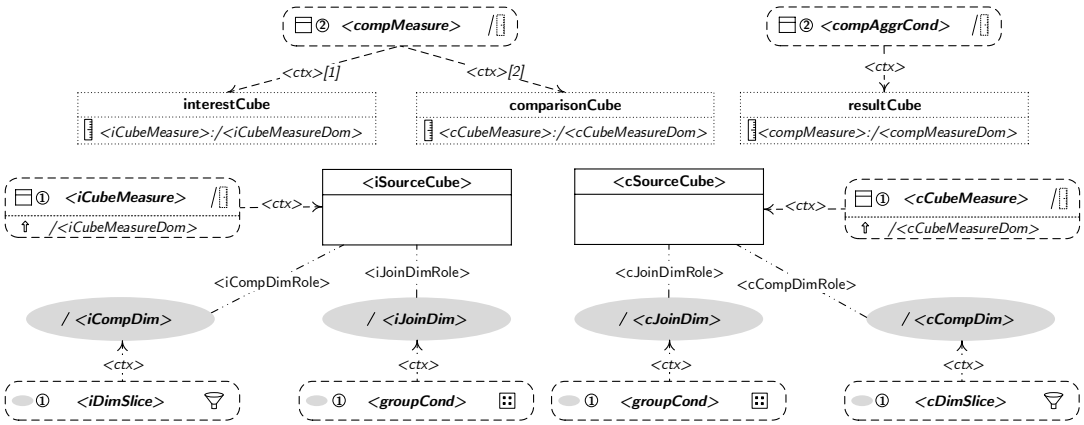


Figure B.21: Context and a template for heterogeneous subset-subset comparison (continued from Figure B.20)

Context



Template (Data Model: Relational, Variant: Star Schema, Language: SQL, Dialect: ORACLEv11)

```
1  WITH interestCube AS (
2    SELECT $expr(<groupCond>, ijd),
3           $expr(<iCubeMeasure>, isc) AS <iCubeMeasure>
4    FROM   <iSourceCube> isc
5           JOIN <iCompDim> icd ON
6              isc.<iCompDimRole>=icd.$dimKey(<iCompDim>)
7           JOIN <iJoinDim> ijd ON
8              isc.<iJoinDimRole>=ijd.$dimKey(<iJoinDim>)
9    WHERE  $expr(<iDimSlice>, icd)
10   GROUP BY $expr(<groupCond>, ijd)
11 ),
12 comparisonCube AS (
13   SELECT $expr(<groupCond>, cjd),
14          $expr(<cCubeMeasure>, csc) AS <cCubeMeasure>
15   FROM   <cSourceCube> csc
16          JOIN <cCompDim> ccd ON
17             csc.<cCompDimRole>=ccd.$dimKey(<cCompDim>)
18          JOIN <cJoinDim> cjd ON
19             csc.<cJoinDimRole>=cjd.$dimKey(<cJoinDim>)
20   WHERE  $expr(<cDimSlice>, ccd)
21   GROUP BY $expr(<groupCond>, cjd)
22 ),
23 resultCube AS (
24   SELECT $expr(<groupCond>, ic),
25          ic.<cubeMeasure> AS "Group of Interest",
26          cc.<cubeMeasure> AS "Group of Comparison",
27          $expr(<compMeasure>, ic, cc) AS <compMeasure>
28   FROM   interestCube ic
29          JOIN comparisonCube cc ON
30             $expr(<groupCond>, ic) = $expr(<groupCond>, cc)
31 )
32
33 SELECT *
34 FROM   resultCube rc
35 WHERE  $expr(<compAggrCond>, rc)
```

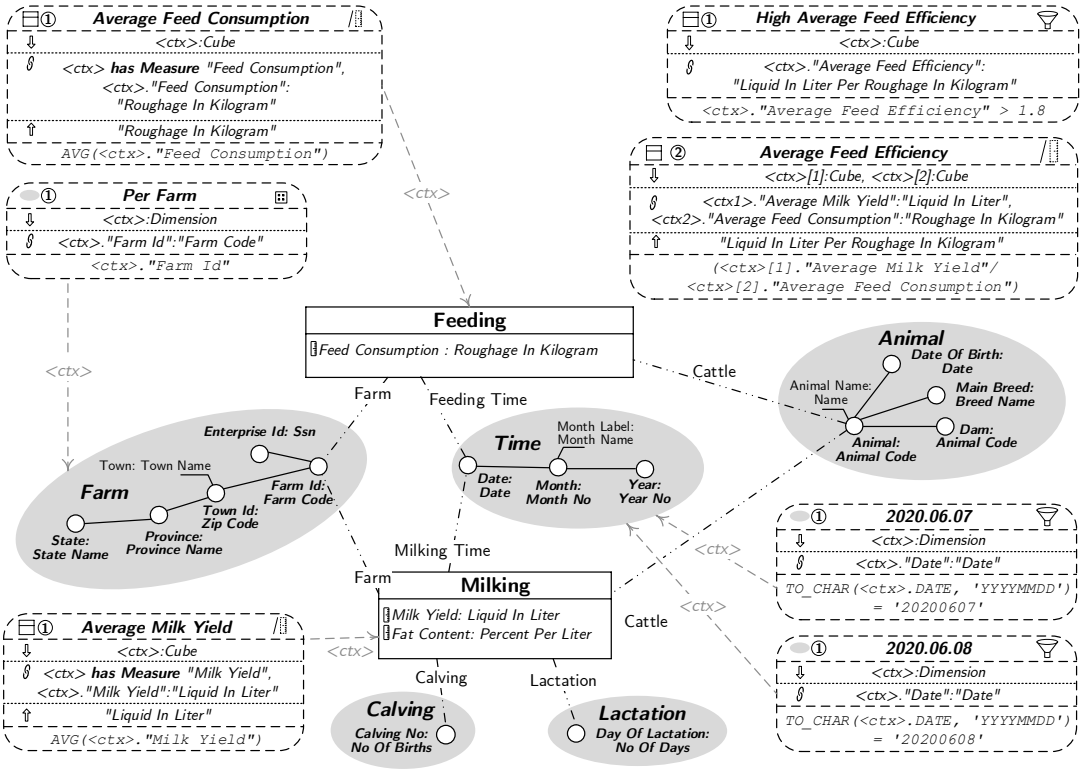
Figure B.22: Related patterns and an example for heterogeneous subset-subset comparison (continued from Figure B.21)

Related Patterns

Homogeneous Subset-Subset Comparison

Example

Consider dairy company *Happy Milk*, which consists of three farms located at different sites, tending to a herd of about a thousand animals in total, half of which are Holstein breed, the other half Jersey breed. Happy Milk runs its farms as precision dairy farming operations aiming to increase efficiency and reduce losses due to animal illness through monitoring the animals' health status and proactively induce necessary treatments.



In precision dairy farming, animals are typically monitored using a wide range of sensors capturing a multitude of data concerning, e.g., microclimate in the barns (temperature and moisture, among others), animal movement within farms, food consumption, milk yield and composition of the produced milk, and information about calvings. Data gathered by various sensors are integrated into a relational data warehouse system realized using a star schema which is conceptually represented using entities and properties in a multidimensional model.

Figure B.23: Example for heterogeneous subset-subset comparison (continued from Figure B.22)

Happy Milk's Milking cube captures for analysis purposes aggregated data about milking events quantified by the measures Milk Yield (in liters) and Fat Content (as percentage); value sets Liquid In Liter and Percent Per Liter are the types of Milk Yield and Fat Content, respectively. Hence, the cube records milk yield and fat content on a daily basis per animal and farm as well as calving period and phase in the animal's lactation cycle. In addition, Happy Milk's Feeding cube captures for analysis purposes aggregated data about feeding events quantified by the measure Feed Consumption (in kilogram); value set Roughage In Kilogram is the type Feed Consumption, respectively. Hence, the cube records feeding on a daily basis per animal and farm.

Happy Milk wants to detect farms that are highly efficient in terms of produced amount of milk and the used resources. A BI user may start the identification of highly efficient farms with the computation of the ratio between the average milk yield of one day and the average consumed feed the day before. On a more abstract level, that query corresponds to heterogeneous subset-subset comparison, i.e., the computation of a ratio between measure values for two subsets of facts originating from two different cubes, each group distinguished according to specified characteristics – in this case, the date of milk production and feed consumption.

To obtain an executable OLAP query based on the heterogeneous subset-subset comparison pattern, the pattern user (BI user) simply binds names of available eMDM elements for each pattern parameter during pattern instantiation. In the Happy Milk scenario, the cube name Milking, which refers to the cube that tracks milk yield of farms in the Happy Milk eMDM, is bound to ⟨iSourceCube⟩ – it is restricted by the unary dimension predicate name 2020.06.08 bound to the parameter ⟨iDimSlice⟩. The unary calculated measure Average Milk Yield is bound to ⟨iCubeMeasure⟩ and aggregated by binding Per Farm to ⟨groupCond⟩ that applies to a dimension referred to by the dimension role Farm bound to ⟨iJoinDimRole⟩. Analogously, the cube name Feeding, which refers to the cube that tracks feeding of farms in the Happy Milk eMDM, is bound to ⟨cSourceCube⟩ – it is restricted by the unary dimension predicate name 2020.06.07 bound to the parameter ⟨cDimSlice⟩. The unary calculated measure Average Feed Consumption is bound to ⟨cCubeMeasure⟩ and aggregated by binding Per Farm to ⟨groupCond⟩ that applies to a dimension referred to by the dimension role Farm bound to ⟨cJoinDimRole⟩. Finally, Average Feed Efficiency is bound to the binary calculated measure ⟨compMeasure⟩, while High Average Feed Efficiency is bound to ⟨compAggrCond⟩.

Figure B.24: Example for heterogeneous subset-subset comparison (continued from Figure B.23)

With parameters for the heterogeneous subset-subset comparison bound as before, pattern grounding over the Happy Milk eMDM yields an applicable ground pattern that is executed resulting in the SQL query shown below, which compares per farm the average milk yield on June 8th, 2020 with the average food consumed the day before, showing only results of a high feeding efficiency.

```
WITH interestCube AS (
  SELECT ijd."Farm Id",
         AVG(isc."Milk Yield") AS "Average Milk Yield"
  FROM   "Milking" isc
         JOIN "Time" icd ON
            isc."Milking Time"=icd."Date"
         JOIN "Animal" ijd ON
            isc."Cattle"=ijd."Animal"
  WHERE  TO_DATE(icd."Date",'YYYYMMDD') = '20200608'
  GROUP BY ijd."Farm Id"
),
comparisonCube AS (
  SELECT cjd."Farm Id",
         AVG(csc."Feed Consumption") AS "Average Feed Consumption"
  FROM   "Feeding" csc
         JOIN "Time" ccd ON
            csc."Feeding Time"=ccd."Date"
         JOIN "Animal" cjd ON
            csc."Cattle"=cjd."Animal"
  WHERE  TO_DATE(ccd."Date",'YYYYMMDD') = '20200607'
  GROUP BY cjd."Farm Id",
),
resultCube AS (
  SELECT ic."Farm Id",
         ic."Average Milk Yield" AS "Group of Interest",
         cc."Average Feed Consumption" AS "Group of Comparison",
         (ic."Average Milk Yield"/cc."Average Feed Consumption") AS
             "Average Feed Efficiency"
  FROM   interestCube ic
         JOIN comparisonCube cc ON
            ic."Farm Id" = cc."Farm Id"
)

SELECT *
FROM   resultCube rc
WHERE  rc."Average Feed Efficiency" > 1.8
```

Appendix C

Domain-Specific Sample Pattern From the AgriProKnow Use Case

The running example of the *Happy Milk* used in this thesis is inspired by the experience we gained during the agriProKnow project, which was funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the "Production of the Future" program between September 2015 and January 2018 (Grant 848610). The OLAP pattern used in the running example describes a race-specific subset-to-subset comparison and is described in detail in Listings Figure C.1 through Figure C.5. These figures comprise the aliases, the problem considered, the solution to be followed, the context defining the contract to be fulfilled, a template linking the pattern to its implementation, related patterns, and a detailed example.

Figure C.1: Aliases, problem, solution, and context of the breed-specific subset-subset comparison pattern

Breed-Specific Subset-Subset Comparison

Also Known As

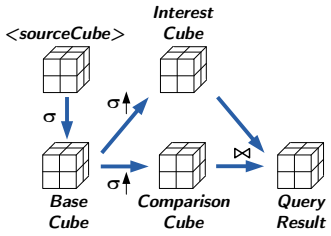
Breed-Specific Cattle Group to Cattle Group Comparison

Problem

Retrieve aggregated measure values for two specified groups of facts about cattle relating to a specific breed from a single source cube, which should be compared by calculating a comparative measure.

Solution

From the ⟨sourceCube⟩, select the set of relevant facts using the unary cube predicate ⟨baseCubeSlice⟩ and dimension predicate ⟨baseDimSlice⟩, which selects over the `Main Breed` level of a dimension ⟨baseDim⟩ referenced by the dimension role `Cattle` – the result serves as the *base cube* for further analysis. From that base cube, select *interest cube* and *comparison cube* using conditions over the dimension role ⟨compDimRole⟩ according to the unary dimension predicates ⟨iDimSlice⟩ and ⟨cDimSlice⟩, respectively. Perform a roll-up for interest and comparison cube according to the ⟨groupCond⟩ dimension grouping over the ⟨joinDim⟩ dimension referenced by the dimension role ⟨joinDimRole⟩ and compute a unary calculated measure ⟨cubeMeasure⟩. To obtain the query result cube, join the interest cube and comparison cube over the ⟨groupCond⟩ dimension grouping and compute a binary calculated comparative measure ⟨compMeasure⟩.



Context

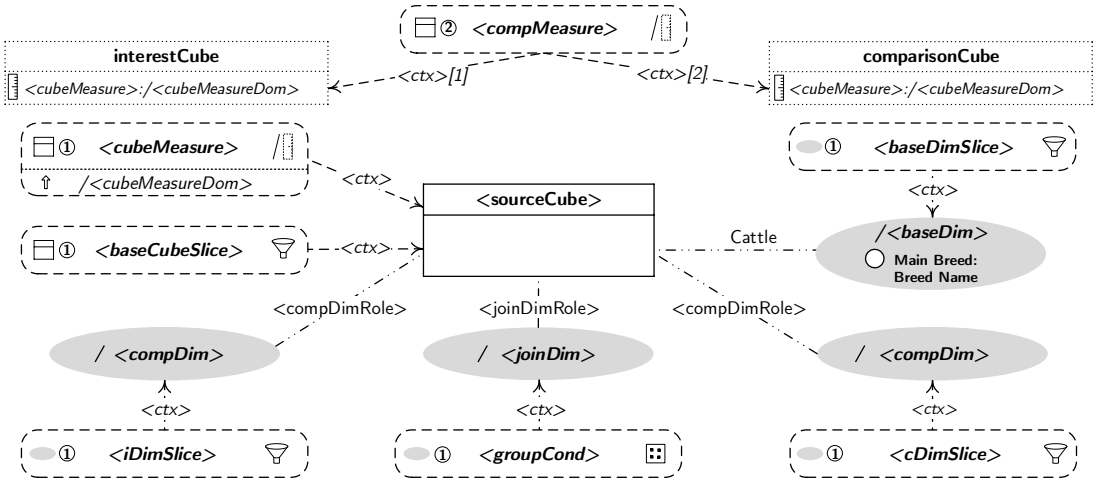


Figure C.2: A template and related patterns for breed-specific subset-subset comparison (continued from Figure C.1)

Template (Data Model: Relational, Variant: Star Schema, Language: SQL, Dialect: ORACLEv11)

```
WITH baseCube AS (
  SELECT *
  FROM   <sourceCube> sc
         JOIN <baseDim> a ON
            sc."Cattle"=a.$dimKey(<baseDim>)
  WHERE  $expr(<baseCubeSlice>, sc) AND
         $expr(<baseDimSlice>, a)
),
interestCube AS (
  SELECT $expr(<groupCond>, jd),
         $expr(<cubeMeasure>, bc) AS <cubeMeasure>
  FROM   baseCube bc
         JOIN <joinDim> jd ON
            bc.<joinDimRole>=jd.$dimKey(<joinDim>)
         JOIN <compDim> cd ON
            bc.<compDimRole>=cd.$dimKey(<compDim>)
  WHERE  $expr(<iDimSlice>, cd)
  GROUP BY $expr(<groupCond>, jd)
),
comparisonCube AS (
  SELECT $expr(<groupCond>, jd),
         $expr(<cubeMeasure>, bc) AS <cubeMeasure>
  FROM   baseCube bc
         JOIN <joinDim> jd ON
            bc.<joinDimRole>=jd.$dimKey(<joinDim>)
         JOIN <compDim> cd ON
            bc.<compDimRole>=cd.$dimKey(<compDim>)
  WHERE  $expr(<cDimSlice>, cd)
  GROUP BY $expr(<groupCond>, jd)
)
SELECT $expr(<groupCond>, ic),
       ic.<cubeMeasure> AS "Group of Interest",
       cc.<cubeMeasure> AS "Group of Comparison",
       $expr(<compMeasure>, ic, cc) AS <compMeasure>
FROM   interestCube ic
       JOIN comparisonCube cc ON
          $expr(<groupCond>, ic)=$expr(<groupCond>, cc)
```

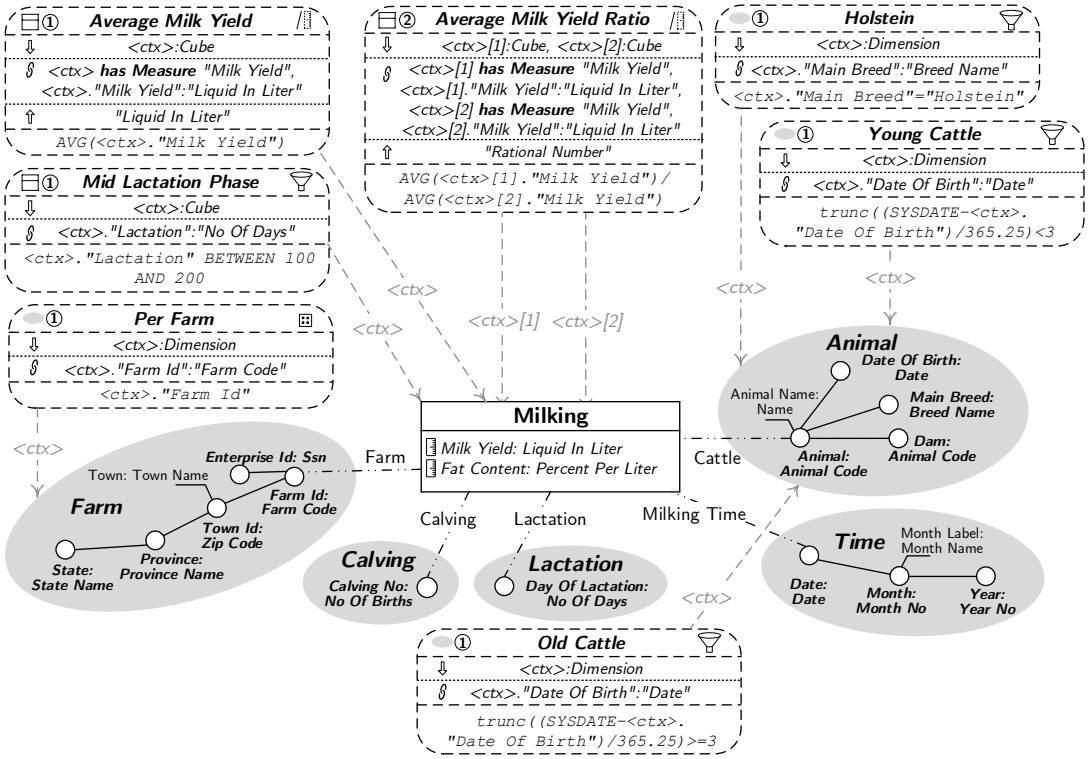
Related Patterns

Diary- and Breed-Specific Subset-Subset Comparison, Homogeneous Subset-Subset Comparison

Figure C.3: Example for breed-specific subset-subset comparison (continued from Figure C.2)

Example

Consider dairy company *Happy Milk*, which consists of three farms located at different sites, tending to a herd of about a thousand animals in total, half of which are Holstein breed, the other half Jersey breed. Happy Milk runs its farms as precision dairy farming operations aiming to increase efficiency and reduce losses due to animal illness through monitoring the animals' health status and proactively induce necessary treatments. In precision dairy farming, animals are typically monitored using a wide range of sensors capturing a multitude of data concerning, e.g., microclimate in the barns (temperature and moisture, among others), animal movement within farms, food consumption, milk yield and composition of the produced milk, and information about calvings. Data gathered by various sensors are integrated into a relational data warehouse system realized using a star schema which is conceptually represented using entities and properties in a multidimensional model.



Happy Milk's `Milking` cube captures for analysis purposes aggregated data about milking events quantified by the measures `Milk Yield` (in liters) and `Fat Content` (as percentage); value sets `Liquid In Liter` and `Percent Per Liter` are the types of `Milk Yield` and `Fat Content`, respectively. Hence, the cube records milk yield and fat content on a daily basis per animal and farm as well as calving period and phase in the animal's lactation cycle.

Figure C.4: Example for breed-specific subset-subset comparison (continued from Figure C.3)

Happy Milk detects a drop in the milk yield of Holstein cattle. A decrease in milk yield may have many reasons but, first of all, the affected Holstein population must be identified. A BI user may start the investigation by calculating the ratio between the average milk yield of young Holstein cattle and the average milk yield of old Holstein cattle, considering only animals with in their mid lactation phase. On a more abstract level, that query corresponds to breed-specific subset-subset comparison: computation of a ratio between the average measure values for each subset of facts referring to a certain breed – in this case, Holstein – where each group is distinguished according to specified characteristics – in this case, young and old.

To obtain an executable OLAP query using the breed-specific subset-subset comparison pattern, the pattern user (BI user) simply binds names of available eMDM elements for each pattern parameter during pattern instantiation. In the Happy Milk scenario, the cube name `Milking`, which in the Happy Milk eMDM refers to a cube that captures milk yield at farms, is bound to ⟨`sourceCube`⟩. The `Milking` cube in the Happy Milk eMDM has a dimension role `Cattle` that references a dimension `Animal` as its domain and thus satisfies the domain and property constraints defined for the cube to be bound to ⟨`sourceCube`⟩. The name `Holstein`, which refers to a unary dimension predicate, is bound to the parameter ⟨`baseDimSlice`⟩ whereas the unary cube predicate name `Mid Lactation Phase` is bound to the parameter ⟨`baseCubeSlice`⟩. The ⟨`compDimRole`⟩ parameter is set to the name `Cattle` since that role refers to a dimension which can be used for further restriction. The groups of facts that are to be compared are specified by the `Young Cattle` unary dimension predicate for the ⟨`iDimSlice`⟩ and the `Old Cattle` unary dimension predicate for the ⟨`cDimSlice`⟩. Since the groups should be compared per farm the `Farm` dimension role is specified as the ⟨`joinDimRole`⟩ while the ⟨`groupCond`⟩ parameter is assigned the `Per Farm` dimension grouping predicate. The `Average Milk Yield` for the ⟨`cubeMeasure`⟩ indicates the unary calculated measure to be calculated per group. Finally, the `Average Milk Yield Ratio` for the ⟨`compMeasure`⟩ indicates the comparative binary calculated measure.

With parameters for the breed-specific subset-subset comparison bound as in the previous example, pattern grounding over the Happy Milk eMDM yields an applicable ground pattern that is executed resulting in the SQL query, which compares per farm the average milk yield of cattle younger than three years that are in their mid lactation phase with cattle older than three years that are in their mid lactation phase, by calculating the ratio over the average milk yield per group.

Figure C.5: Example for breed-specific subset-subset comparison (continued from Figure C.4)

```
WITH baseCube AS (
  SELECT *
  FROM   "Milking" sc
         JOIN "Animal" a ON
            sc."Cattle" = a."Animal"
  WHERE  sc."Lactation" BETWEEN 100 AND 200 AND
         a."Main Breed" = "Holstein"
),
interestCube AS (
  SELECT jd."Farm Id",
         AVG(bc."Milk Yield") AS "Average Milk Yield"
  FROM   baseCube bc
         JOIN "Farm" jd ON
            bc."Farm" = jd."Farm Id"
         JOIN "Animal" cd ON
            bc."Cattle" = cd."Animal"
  WHERE  trunc((SYSDATE-cd."Date Of Birth")/365.25)<3
  GROUP BY jd."Farm Id"
),
comparisonCube AS (
  SELECT jd."Farm Id",
         AVG(bc."Milk Yield") AS "Average Milk Yield"
  FROM   baseCube bc
         JOIN "Farm" jd ON
            bc."Farm" = jd."Farm Id"
         JOIN "Animal" cd ON
            bc."Cattle" = cd."Animal"
  WHERE  trunc((SYSDATE-cd."Date Of Birth")/365.25)>=3
  GROUP BY jd."Farm Id"
),
SELECT ic."Farm Id",
       ic."Average Milk Yield" AS "Group of Interest",
       cc."Average Milk Yield" AS "Group of Comparison",
       (ic."Average Milk Yield"/cc."Average Milk Yield") AS "Average
           Milk Yield Ratio"
FROM   interestCube ic
       JOIN comparisonCube cc ON
          ic."Farm Id" = cc."Farm Id"
```

Appendix D

Fully-Worked Organization-Specific Running Example of the Thesis

The usage scenario described in Section 8.4 shows only a few statements required to define the cubes, dimensions, and business terms for subsequent pattern definition and usage. A complete set of the required statements to be executed is shown as follows: Listing D.1 represents the statements to define the organization of the repository; Listing D.2 represents the statements to define the dimensions and cubes; Listing D.3 represents the statements for defining the required business terms; and Listing D.4 represents the definition of the pattern to be used. It should be noted that a full definition is provided for business terms and the pattern, that is, both a description and a template are defined.

Listing D.1: Happy Milk's repository organization

```
CREATE REPOSITORY "Happy Milk";
CREATE CATALOGUE "Happy Milk"/"Happy Milk Catalog";
CREATE GLOSSARY "Happy Milk"/"Happy Milk Vocabulary";
CREATE MULTIDIMENSIONAL_MODEL "Happy Milk"/"Happy Milk MDM";
```

Listing D.2: Happy Milk's dimension and cube definitions

```
-- ANIMAL DIMENSION
CREATE DIMENSION "Happy Milk"/"Happy Milk MDM"/"Animal" WITH
   LEVEL PROPERTIES
      "Animal":"Animal Code";
      "Date Of Birth":"Date";
      "Main Breed":"Breed Name";
      "Dam":"Animal Code";
   END LEVEL PROPERTIES;

   ATTRIBUTE PROPERTIES
      "Animal Name":"Name";
   END ATTRIBUTE PROPERTIES;

   CONSTRAINTS
      "Animal" ROLLS_UP_TO "Date Of Birth";
      "Animal" ROLLS_UP_TO "Main Breed";
      "Animal" ROLLS_UP_TO "Dam";
      "Animal" DESCRIBED_BY "Animal Name";
   END CONSTRAINTS;
END DIMENSION;

-- TIME DIMENSION
CREATE DIMENSION "Happy Milk"/"Happy Milk MDM"/"Time" WITH
   LEVEL PROPERTIES
      "Date":"Date";
      "Month":"Month No";
      "Year":"Year No";
   END LEVEL PROPERTIES;

   ATTRIBUTE PROPERTIES
      "Month Label":"Month Name";
   END ATTRIBUTE PROPERTIES;

   CONSTRAINTS
      "Date" ROLLS_UP_TO "Month";
      "Month" ROLLS_UP_TO "Year";
      "Month" DESCRIBED_BY "Month Label";
   END CONSTRAINTS;
```

```
39 END DIMENSION;
40
41 -- LACTATION DIMENSION
42 CREATE DIMENSION "Happy Milk"/"Happy Milk MDM"/"Lactation" WITH
43    LEVEL PROPERTIES
44       "Day Of Lactation":"No Of Days";
45    END LEVEL PROPERTIES;
46 END DIMENSION;
47
48 -- CALVING DIMENSION
49 CREATE DIMENSION "Happy Milk"/"Happy Milk MDM"/"Calving" WITH
50    LEVEL PROPERTIES
51       "Calving No":"No Of Births";
52    END LEVEL PROPERTIES;
53 END DIMENSION;
54
55 -- FARM DIMENSION
56 CREATE DIMENSION "Happy Milk"/"Happy Milk MDM"/"Farm" WITH
57    LEVEL PROPERTIES
58       "Farm Id":"Farm Code";
59       "Enterprise Id":"Ssn";
60       "Town Id":"Zip Code";
61       "Province":"Province Name";
62     "State":"State Name";
63    END LEVEL PROPERTIES;
64
65    ATTRIBUTE PROPERTIES
66       "Town":"Town Name";
67    END ATTRIBUTE PROPERTIES;
68
69    CONSTRAINTS
70       "Farm Id" ROLLS_UP_TO "Enterprise Id";
71       "Farm Id" ROLLS_UP_TO "Town Id";
72       "Town Id" ROLLS_UP_TO "Province";
73     "Province" ROLLS_UP_TO "State";
74       "Town Id" DESCRIBED_BY "Town";
75    END CONSTRAINTS;
76 END DIMENSION;
77
78 -- MILKING CUBE
79 CREATE CUBE "Happy Milk"/"Happy Milk MDM"/"Milking" WITH
80    MEASURE PROPERTIES
81       "Milk Yield":"Liquid In Liter";
82       "Fat Content":"Percent Per Liter";
83    END MEASURE PROPERTIES;
84
```

```
85    DIMENSION_ROLE PROPERTIES
86       "Farm":"Farm";
87       "Calving":"Calving";
88       "Lactation":"Lactation";
89       "Milking Time":"Time";
90       "Cattle":"Animal";
91    END DIMENSION_ROLE PROPERTIES;
92 END CUBE;
```

Listing D.3: Happy Milk's business term definitions

```
1  -- LOW DAILY MILK YIELD BUSINESS TERM
2  CREATE UNARY_CUBE_PREDICATE "Happy Milk"/"Happy Milk Vocabulary"/"Low
       Daily Milk Yield" WITH
3    CONSTRAINTS
4       <ctx>."Milk Yield":"Liquid In Liter";
5    END CONSTRAINTS;
6  END UNARY_CUBE_PREDICATE;
7
8  CREATE TERM DESCRIPTION FOR "Happy Milk"/"Happy Milk Vocabulary"/"Low
       Daily Milk Yield" WITH
9    LANGUAGE = "English";
10   ALIAS = "Daily Milk Yield Below Average";
11   DESCRIPTION = "A cattle has a low dailymilk yield if its milk yield for
        a specific day is lower than 7.4";
12 END TERM DESCRIPTION;
13
14 CREATE TERM TEMPLATE FOR "Happy Milk"/"Happy Milk Vocabulary"/"Low Daily
       Milk Yield" WITH
15   LANGUAGE = "SQL" ;
16   DIALECT = "ORACLEv11" ;
17   EXPRESSION = "*{ <ctx>.""Milk Yield"" < 7.4 }*";
18 END TERM TEMPLATE;
19
20
21 -- AVERAGE MILK YIELD BUSINESS TERM
22 CREATE UNARY_CALCULATED_MEASURE "Happy Milk"/"Happy Milk
       Vocabulary"/"Average Milk Yield" WITH
23
24   CONSTRAINTS
25      <ctx> HAS MEASURE "Milk Yield";
26      <ctx>."Milk Yield":"Liquid In Liter";
27   END CONSTRAINTS;
28
29   RETURNS "Liquid In Liter";
30
```

```
31 END UNARY_CALCULATED_MEASURE;
32
33 CREATE TERM DESCRIPTION FOR "Happy Milk"/"Happy Milk Vocabulary"/"Average
       Milk Yield" WITH
34   LANGUAGE = "English";
35   ALIAS = "Average Milk Production";
36   DESCRIPTION = "The average milk yield production";
37 END TERM DESCRIPTION;
38
39 CREATE TERM TEMPLATE FOR "Happy Milk"/"Happy Milk Vocabulary"/"Average
       Milk Yield" WITH
40   LANGUAGE = "SQL" ;
41   DIALECT = "ORACLEv11" ;
42   EXPRESSION = "*{ AVG(<ctx>.""Milk Yield"") }*";
43 END TERM TEMPLATE;
44
45
46 -- AVERAGE MILK YIELD RATIO BUSINESS TERM
47 CREATE BINARY_CALCULATED_MEASURE  "Happy Milk"/"Happy Milk
       Vocabulary"/"Average Milk Yield Ratio" WITH
48   CONSTRAINTS
49      <ctx>[1] HAS MEASURE "Average Milk Yield";
50      <ctx>[1]."Average Milk Yield":"Liquid In Liter";
51      <ctx>[2] HAS MEASURE "Average Milk Yield";
52      <ctx>[2]."Average Milk Yield":"Liquid In Liter";
53   END CONSTRAINTS;
54
55   RETURNS "Rational Number";
56 END BINARY_CALCULATED_MEASURE;
57
58 CREATE TERM DESCRIPTION FOR "Happy Milk"/"Happy Milk Vocabulary"/"Average
       Milk Yield Ratio" WITH
59   LANGUAGE = "English";
60   ALIAS = "Average Milk Production Ratio";
61   DESCRIPTION = "The ratio of the average milk yield production";
62 END TERM DESCRIPTION;
63
64 CREATE TERM TEMPLATE FOR "Happy Milk"/"Happy Milk Vocabulary"/"Average
       Milk Yield Ratio" WITH
65   LANGUAGE = "SQL";
66   DIALECT = "ORACLEv11";
67   EXPRESSION = "*{ <ctx>[1].""Average Milk Yield"")/<ctx>[2].""Average
         Milk Yield"" }*";
68 END TERM TEMPLATE;
69
70
```

```
71 -- HIGH AVERAGE MILK YIELD RATIO BUSINESS TERM
72 CREATE UNARY_CUBE_PREDICATE "Happy Milk"/"Happy Milk Vocabulary"/"High
       Average Milk Yield Ratio" WITH
73   CONSTRAINTS
74      <ctx>."Average Milk Yield Ratio":"Liquid In Liter";
75   END CONSTRAINTS;
76 END UNARY_CUBE_PREDICATE;
77
78 CREATE TERM DESCRIPTION FOR "Happy Milk"/"Happy Milk Vocabulary"/"High
       Average Milk Yield Ratio" WITH
79   LANGUAGE = "English";
80   ALIAS = "Average Milk Production Ratio Over 20%";
81   DESCRIPTION = "The ratio of the average milk yield production exceeds
         20%";
82 END TERM DESCRIPTION;
83
84 CREATE TERM TEMPLATE FOR "Happy Milk"/"Happy Milk Vocabulary"/"High
       Average Milk Yield Ratio" WITH
85   LANGUAGE = "SQL" ;
86   DIALECT = "ORACLEv11" ;
87   EXPRESSION = "*{ <ctx>.""Average Milk Yield Ratio""=>0.2 }*";
88 END TERM TEMPLATE;
89
90 -- YOUNG CATTLE BUSINESS TERM
91 CREATE UNARY_DIMENSION_PREDICATE "Happy Milk"/"Happy Milk
       Vocabulary"/"Young Cattle" WITH
92   CONSTRAINTS
93      <ctx>."Date Of Birth":"Date";
94   END CONSTRAINTS;
95 END UNARY_DIMENSION_PREDICATE;
96
97 CREATE TERM DESCRIPTION FOR "Happy Milk"/"Happy Milk Vocabulary"/"Young
       Cattle" WITH
98   LANGUAGE = "English";
99   ALIAS = "Young Cow";
100  DESCRIPTION = "A cow is young if its age in years is lower than three";
101 END TERM DESCRIPTION;
102
103 CREATE TERM TEMPLATE FOR "Happy Milk"/"Happy Milk Vocabulary"/"Young
       Cattle" WITH
104  LANGUAGE = "SQL" ;
105  DIALECT = "ORACLEv11" ;
106  EXPRESSION = "*{ trunc((SYSDATE - <ctx>.""Date Of Birth"")/365.25) < 3
         }*";
107 END TERM TEMPLATE;
108
```

```
109 -- OLD CATTLE BUSINESS TERM
110 CREATE UNARY_DIMENSION_PREDICATE "Happy Milk"/"Happy Milk Vocabulary"/"Old
       Cattle" WITH
111   CONSTRAINTS
112     <ctx>."Date Of Birth":"Date";
113   END CONSTRAINTS;
114 END UNARY_DIMENSION_PREDICATE;
115
116 CREATE TERM DESCRIPTION FOR "Happy Milk"/"Happy Milk Vocabulary"/"Old
       Cattle" WITH
117   LANGUAGE = "English";
118   ALIAS = "Old Cow";
119   DESCRIPTION = "A cow is old if its age in years is greater equals than
        three";
120 END TERM DESCRIPTION;
121
122 CREATE TERM TEMPLATE FOR "Happy Milk"/"Happy Milk Vocabulary"/"Old Cattle"
       WITH
123   LANGUAGE = "SQL" ;
124   DIALECT = "ORACLEv11" ;
125   EXPRESSION = "*{ trunc((SYSDATE - <ctx>.""Date Of Birth"")/365.25) >= 3
        }*";
126 END TERM TEMPLATE;
127
128
129 -- HOLSTEIN BUSINESS TERM
130 CREATE UNARY_DIMENSION_PREDICATE "Happy Milk"/"Happy Milk
       Vocabulary"/"Holstein" WITH
131   CONSTRAINTS
132     <ctx>."Main Breed":"Breed Name";
133   END CONSTRAINTS;
134 END UNARY_DIMENSION_PREDICATE;
135
136 CREATE TERM DESCRIPTION FOR "Happy Milk"/"Happy Milk
       Vocabulary"/"Holstein" WITH
137   LANGUAGE = "English";
138   ALIAS = "Cattle Breed Holstein";
139   DESCRIPTION = "Restriction of result to cattle of main breed holstein";
140 END TERM DESCRIPTION;
141
142 CREATE TERM TEMPLATE FOR "Happy Milk"/"Happy Milk Vocabulary"/"Holstein"
       WITH
143   LANGUAGE = "SQL" ;
144   DIALECT = "ORACLEv11" ;
145   EXPRESSION = "*{ <ctx>.""Main Breed"" = ""Holstein"" }*";
146 END TERM TEMPLATE;
```

```
147
148
149 -- PER FARM BUSINESS TERM
150 CREATE DIMENSION_GROUPING "Happy Milk"/"Happy Milk Vocabulary"/"Per Farm"
        WITH
151   CONSTRAINTS
152      <ctx>."Farm Id":"Farm Code";
153   END CONSTRAINTS;
154 END DIMENSION_GROUPING;
155
156 CREATE TERM DESCRIPTION FOR "Happy Milk"/"Happy Milk Vocabulary"/"Per
        Farm" WITH
157   LANGUAGE = "English";
158   ALIAS = "Aggregate At Farm Level";
159   DESCRIPTION = "Rollup to the Farm level";
160 END TERM DESCRIPTION;
161
162 CREATE TERM TEMPLATE FOR "Happy Milk"/"Happy Milk Vocabulary"/"Per Farm"
        WITH
163   LANGUAGE = "SQL" ;
164   DIALECT = "ORACLEv11" ;
165   EXPRESSION = "*{ <ctx>.""Farm Id"" }*";
166 END TERM TEMPLATE;
167
168
169 -- SAME FARM BUSINESS TERM
170 CREATE BINARY_DIMENSION_PREDICATE "Happy Milk"/"Happy Milk
        Vocabulary"/"Same Farm" WITH
171   CONSTRAINTS
172      <ctx>[1] HAS LEVEL "Farm Id";
173      <ctx>[1]."Farm Id":"Farm Code";
174    <ctx>[2] HAS LEVEL "Farm Id";
175      <ctx>[2]."Farm Id":"Farm Code";
176   END CONSTRAINTS;
177 END BINARY_DIMENSION_PREDICATE;
178
179 CREATE TERM DESCRIPTION FOR "Happy Milk"/"Happy Milk Vocabulary"/"Same
        Farm" WITH
180   LANGUAGE = "English";
181   ALIAS = "Join Over Farm", "Join By Farm";
182   DESCRIPTION = "Combine same farm levels";
183 END TERM DESCRIPTION;
184
185 CREATE TERM TEMPLATE  FOR "Happy Milk"/"Happy Milk Vocabulary"/"Same Farm"
        WITH
186   LANGUAGE = "SQL" ;
```

```
187   DIALECT = "ORACLEv11" ;
188   EXPRESSION = "*{ <ctx>[1].""Farm Id"" = <ctx>[2].""Farm Id"" }*";
189 END TERM TEMPLATE;
190
191
192 -- TOWN ASC BUSINESS TERM
193 CREATE DIMENSION_ORDERING "Happy Milk"/"Happy Milk Vocabulary"/"Town ASC"
        WITH
194   CONSTRAINTS
195      <ctx>."Town":"Town Name";
196   END CONSTRAINTS;
197 END DIMENSION_ORDERING;
198
199 CREATE TERM DESCRIPTION FOR "Happy Milk"/"Happy Milk Vocabulary"/"Town
        ASC" WITH
200   LANGUAGE = "English";
201   ALIAS = "Town Ascending";
202   DESCRIPTION = "Order result according to the town name in an ascending
         way";
203 END TERM DESCRIPTION;
204
205 CREATE TERM TEMPLATE FOR "Happy Milk"/"Happy Milk Vocabulary"/"Town ASC"
        WITH
206   LANGUAGE = "SQL" ;
207   DIALECT = "ORACLEv11" ;
208   EXPRESSION = "*{ <ctx>.""Town"" }*";
209 END TERM TEMPLATE;
210
211
212 -- SEPTEMBER BUSINESS TERM
213 CREATE UNARY_DIMENSION_PREDICATE "Happy Milk"/"Happy Milk
        Vocabulary"/"September" WITH
214   CONSTRAINTS
215      <ctx>."Month Label":"Month Name";
216   END CONSTRAINTS;
217 END UNARY_DIMENSION_PREDICATE;
218
219 CREATE TERM DESCRIPTION FOR "Happy Milk"/"Happy Milk
        Vocabulary"/"September" WITH
220   LANGUAGE = "English";
221   ALIAS = "Month September";
222   DESCRIPTION = "Restrict result to the month September";
223 END TERM DESCRIPTION;
224
225 CREATE TERM TEMPLATE FOR "Happy Milk"/"Happy Milk Vocabulary"/"September"
        WITH
```

```
226   LANGUAGE = "SQL" ;
227   DIALECT = "ORACLEv11" ;
228   EXPRESSION = "*{ <ctx>.""Month Label"" = ""September"" }*";
229 END TERM TEMPLATE;
230
231
232 -- UNDER HEAT STRESS BUSINESS TERM
233 CREATE BINARY_DIMENSION_PREDICATE "Happy Milk"/"Happy Milk
      Vocabulary"/"Under Heat Stress" WITH
234   CONSTRAINTS
235       <ctx>[1]."Animal":"Animal Code";
236       <ctx>[2]."Date":"Date";
237   END CONSTRAINTS;
238 END BINARY_DIMENSION_PREDICATE;
239
240 CREATE TERM DESCRIPTION FOR "Happy Milk"/"Happy Milk Vocabulary"/"Under
      Heat Stress" WITH
241   LANGUAGE = "English";
242   ALIAS = "Heat Stress";
243   DESCRIPTION = "Any time the Temperature-Humidity Index (THI) is above 80
        cattle will be under heat stress";
244 END TERM DESCRIPTION;
245
246 CREATE TERM TEMPLATE  FOR "Happy Milk"/"Happy Milk Vocabulary"/"Under Heat
      Stress" WITH
247   LANGUAGE = "SQL" ;
248   DIALECT = "ORACLEv11" ;
249   EXPRESSION = "*{ (<ctx>[1].""Animal"", <ctx>[2].""Date"") IN SELECT *
        FROM ""Under Heat Stress"") }*";
250 END TERM TEMPLATE;
```

Listing D.4: Happy Milk's pattern definitions

```
1 CREATE PATTERN "Happy Milk"/"Happy Milk Catalog"/"Breed-Specific
      Subset-Subset Comparison" WITH
2   PARAMETERS
3       <sourceCube>:CUBE;
4       <baseCubeSlice>:UNARY_CUBE_PREDICATE;
5       <animalBreedSlice>:UNARY_DIMENSION_PREDICATE;
6       <compDimRole>:DIMENSION_ROLE;
7       <iDimSlice>:UNARY_DIMENSION_PREDICATE;
8       <cDimSlice>:UNARY_DIMENSION_PREDICATE;
9       <joinDimRole>:DIMENSION_ROLE;
10      <groupCond>:DIMENSION_GROUPING;
11      <cubeMeasure>:UNARY_CALCULATED_MEASURE;
12      <compMeasure>:BINARY_CALCULATED_MEASURE;
```

```
13   END PARAMETERS;
14
15   DERIVED ELEMENTS
16      <compDim>:DIMENSION <= <sourceCube>.<compDimRole>;
17      <joinDim>:DIMENSION <= <sourceCube>.<joinDimRole>;
18        <cubeMeasureDom>:NUMBER_VALUE_SET <= <cubeMeasure>.RETURNS;
19   END DERIVED ELEMENTS;
20
21   LOCAL CUBES
22     "interestCube":CUBE;
23     "interestCube" HAS MEASURE <cubeMeasure>;
24     "interestCube".<cubeMeasure>:<cubeMeasureDom>;
25     "comparisonCube":CUBE;
26     "comparisonCube" HAS MEASURE <cubeMeasure>;
27     "comparisonCube".<cubeMeasure>:<cubeMeasureDom>;
28   END LOCAL CUBES;
29
30   CONSTRAINTS
31      <sourceCube> HAS DIMENSION_ROLE "Cattle";
32      <sourceCube> HAS DIMENSION_ROLE <compDimRole>;
33      <sourceCube>."Cattle":"Animal";
34      "Animal" HAS LEVEL "Main Breed";
35      "Animal"."Main Breed":"Breed Name";
36
37      <sourceCube>.<compDimRole>:<compDim>;
38      <sourceCube>.<joinDimRole>:<joinDim>;
39      <cubeMeasure> RETURNS <cubeMeasureDom>;
40
41      <animalBreedSlice> IS_APPLICABLE_TO "Animal";
42      <baseCubeSlice> IS_APPLICABLE_TO <sourceCube>;
43      <iDimSlice> IS_APPLICABLE_TO <compDim>;
44      <cDimSlice> IS_APPLICABLE_TO <compDim>;
45      <groupCond> IS_APPLICABLE_TO <joinDim>;
46      <cubeMeasure> IS_APPLICABLE_TO <sourceCube>;
47      <compMeasure> IS_APPLICABLE_TO ("interestCube","comparisonCube");
48   END CONSTRAINTS;
49 END PATTERN;
50
51 CREATE PATTERN DESCRIPTION FOR "Happy Milk"/"Happy Milk
       Catalog"/"Breed-Specific Subset-Subset Comparison" WITH
52   LANGUAGE = "English";
53   ALIAS = "Breed-Specific Comparison";
54   PROBLEM = "From the <sourceCube>, select the set of relevant facts using
         the unary cube predicate <baseCubeSlice> and dimension predicate
         <baseDimSlice>, which selects over the ""Main Breed"" level of a
         dimension <baseDim> -- the result serves as the ""base cube"" for
```

```
      further analysis.  From that base cube, select ""interest cube"" and
      ""comparison cube"" using conditions over the dimension role
      <compDimRole> according to the unary dimension predicates <iDimSlice>
      and <cDimSlice>, respectively. Perform a roll-up for interest and
      comparison cube according to the <groupCond> dimension grouping over
      the <joinDim> dimension referenced by the dimension role
      <joinDimRole> and compute a unary calculated measure <cubeMeasure>.
      To obtain the query result cube, join the interest cube and
      comparison cube over the <groupCond> dimension grouping and compute a
      binary calculated comparative measure <compMeasure>.";
  RELATED = "Breed-Specific Subset-Subset Side-By-Side Comparison";
  EXAMPLE = "....";
END PATTERN DESCRIPTION;

CREATE PATTERN TEMPLATE FOR "Happy Milk"/"Happy Milk
    Catalog"/"Breed-Specific Subset-Subset Comparison" WITH
  LANGUAGE = "SQL" ;
  DIALECT = "ORACLEv11" ;
  EXPRESSION = "
*{ WITH baseCube AS (
   SELECT *
   FROM   <sourceCube> sc JOIN
          ""Animal"" a ON sc.""Cattle"" = a.}*$dimKey(""Animal"")*{
  WHERE  }* $expr(<baseCubeSlice>, sc) *{ AND }*
         $expr(<animalBreedSlice>, a)*{),

interestCube AS (
  SELECT }* $expr(<groupCond>, jd)*{,
         }* $expr(<cubeMeasure>, bc)*{ AS <cubeMeasure>
  FROM   baseCube bc JOIN
         <joinDim> jd ON bc.<joinDimRole>=jd.}*$dimKey(<joinDim>)*{ JOIN
         <compDim> cd ON bc.<compDimRole>=cd.}*$dimKey(<compDim>)*{
  WHERE  }*$expr(<iDimSlice>, cd)*{
  GROUP BY }*$expr(<groupCond>, jd)*{),

comparisonCube AS (
  SELECT }*$expr(<groupCond>, jd)*{,
         }*$expr(<cubeMeasure>, bc)*{ AS <cubeMeasure>
  FROM   baseCube bc JOIN
         <joinDim> jd ON bc.<joinDimRole>=jd.}*$dimKey(<joinDim>)*{ JOIN
         <compDim> cd ON bc.<compDimRole>=cd.}*$dimKey(<compDim>)*{
  WHERE  }*$expr(<cDimSlice>, cd)*{
  GROUP BY }*$expr(<groupCond>, jd)*{)

SELECT }*$expr(<groupCond>, ic)*{,
       ic.<cubeMeasure> AS ""Group of Interest"",
```

```
90          cc.<cubeMeasure> AS ""Group of Comparison"",
91          }*$expr(<compMeasure>, ic, cc)*{ AS <compMeasure>
92  FROM    interestCube ic JOIN
93          comparisonCube cc ON }*$expr(<groupCond>, ic)*{ = }*
                $expr(<groupCond>, cc)";
94  END PATTERN TEMPLATE;
```

Appendix E

Curriculum Vitæ

Ilko Kovačić, MSc.

Curriculum Vitæ

Personal Information

Name	Ilko Kovačić
Address	Mariahilfer Straße 49/1/10, 1060 Wien
Phone	+43 699 11013738
E-Mail	ilko.kovacic@gmail.com
Birthday	October 3, 1990

Work Experience

Co-Founder & Executive Director

since 02/2021 — Finspacer GmbH

Sales and Product Development.

Lecturer for Bachelor Study Business Informatics

since 01/2021 — Johannes Kepler University, Linz

Teaching class Data Modeling and Data & Knowledge Engineering.

Lecturer for Bachelor Study Information, Media and Communication

since 02/2020 — Fachhochschule Burgenland, Pinkafeld

Teaching class Database Systems.

Lecturer for Master Study E-Commerce

since 09/2018 — Fachhochschule Wiener Neustadt, Campus Wieselburg

Teaching class Databases & Information Management.

University Assistant

09/2015–12/2020 — Johannes Kepler University, Linz - Data & Knowledge Engineering

University teaching, co-supervision of theses, and participation in research and development projects.

Project Member

08/2015–09/2015 — Johannes Kepler University, Linz - Data & Knowledge Engineering

Participation in the SemNOTAM project.

Tutor

03/2014–06/2015 Johannes Kepler University, Linz - Data & Knowledge Engineering

Grading excercises and supporting students in class Data & Knowledge Engineering and Data Modeling.

Produkt- & Content-Manager

02/2012–12/2013 3p+ projekt produkt präsentation GmbH, Holzhausen

Management of data preparation, product development, and project execution within the development of the 3p+ profiler.

Education

Doctoral Program in Social Sciences, Economics & Business

2015–2021 Johannes Kepler University, Linz

Specialization in Business Informatics

Master's degree in Business Informatics (with distinction)

2013–2015 Johannes Kepler University, Linz

Specialization in Business Intelligence & Data Science

Bachelor's degree in Business Informatics

2010–2013 Johannes Kepler University, Linz

Matura

2005–2010 Higher Technical College for Information Technology, Wels

Reseach & Development Projects

BEST - Achieving the Benefits of SWIM by making smart use of Semantic Technologies

2016–2018 SESAR Joint Undertaking, under European Union's Horizon 2020 Research and Innovation programme by EU Horizon 2020 (Grant Agreement No. 699298)

AgriProKnow - Information Management in Precision Dairy Farming

2015–2018 Austrian Research Promotion Agency (FFG - Production of the Future) (Grant Agreement No. 848610)

SemNOTAM: Ontology-based representation and semantic querying of Digital Notices to Airman

2014–2017 Austrian Research Promotion Agency (FFG - TAKE OFF) (Grant Agreement No. 83990)

Theses

OLAP Patterns: A Pattern-Based Approach to Multidimensional Data Analysis

Department Department for Business Informatics - Data & Knowledge Engineering

Supervisor o. Univ.-Prof. Dipl.-Ing. Dr. techn. Michael Schrefl

Implementation of an Extensible Mapper of Aeronautical Information Exchange Model Data to ObjectLogic

Department: Department for Business Informatics - Data & Knowledge Engineering

Supervisors: o. Univ.-Prof. Dipl.-Ing. Dr. techn. Michael Schrefl

Entwicklung von Interaktionskonzepten für die gestenbasierte und kollaborative Modellierung - Implementierung des konzeptuellen Entwurfs anhand eines Prototyps

Department: Department for Business Informatics - Communications Engineering

Supervisors: o. Univ.-Prof. Dipl.-Ing. Dr. Christian Stary

Publications

2021 I. Kovacic, C. Schütz, B. Neumayr, M. Schrefl: *OLAP Patterns: A Pattern-Based Approach to Multidimensional Data Analysis*. In: Data & Knowledge Engineering Journal, peer reviewed, 2021. (accepted subject to minor revision)

2018 I. Kovacic, C. Schütz, S. Schausberger, R. Sumereder, M. Schrefl: *Guided Query Composition with Semantic OLAP Patterns.* In: Proceedings of the 2nd International Workshop on Data Analytics Solutions for Real-Life Applications (DARLI-AP 2018), EDBT/ICDT 2018 Joint Conference, CEUR Workshop Proceedings, 8 pages, 2018.

2017 C. Schütz, S. Schausberger, I. Kovacic, M. Schrefl: *Semantic OLAP Patterns: Elements of Reusable Business Analytics*. In: Proceedings of the Confederated International Conferences On-the-Move 2017 (OTM 2017), October 23-27, 2017, Rhodes, Greece, Springer International Publishing, Lecture Notes in Computer Science (LNCS Vol. 10574), Print ISBN 978-3-319-69458-0, Online ISBN 978-3-319-69459-7, peer reviewed, pp. 318-336, 2017.

2017 I. Kovacic, D. Steiner, C. Schütz, B. Neumayr, F. Burgstaller, M. Schrefl, S. Wilson: *Ontology-Based Data Description and Discovery in a SWIM Environment.* In: Proceedings of the 2017 Integrated Communications, Navigation and Surveillance Conference (ICNS 2017), April 18-20, 2017, Washington D.C., USA, IEEE Computer Society Press, 13 pages, peer reviewed, Best Paper in Track 5 - Special Topics/Other, 2017.

2016 D. Steiner, F. Burgstaller, E. Gringinger, M. Schrefl, I. Kovacic: *In-flight Provisioning and Distribution of ATM Information.* In: Proceedings of the 30th Congress of the International Council of the Aeronautical Sciences (ICAS 2016), September 25-30, 2016, Daejeon, Korea, 2016.

2016 D. Steiner, I. Kovacic, F. Burgstaller, M. Schrefl, T. Friesacher, E. Gringinger: *Semantic Enrichment of DNOTAMs to Reduce Information Overload in Pilot Briefings.* In: Proceedings of the Integrated Communications, Navigation and Surveillance Conference (ICNS), April 19-21, 2016, Washington D.C., USA, IEEE Publications, 2016.

Talks

06/2021 Austrian Computer Science Day 2021 in Klagenfurt, Austria, Nominated as Young Expert

10/2018 2nd International Workshop on Data Analytics Solutions for Real-Life Applications (DARLI-AP 2018) in Vienna, Austria, workshop participation and presentation of the paper *Guided Query Composition with Semantic OLAP Patterns*

10/2017 Confederated International Conferences On-the-Move 2017 (OTM 2017) in Rhodes, Greece, conference participation and presentation of the paper *Semantic OLAP Patterns: Elements of Reusable Business Analytics*

04/2017 Integrated Communication, Navigation, and Surveillance Conference (ICNS) in Washington D.C., USA, conference participation and presentation of the paper *Ontology-Based Data Description and Discovery in a SWIM Environment*

04/2016 Integrated Communication, Navigation, and Surveillance Conference (ICNS) in Washington D.C., USA, conference participation and presentation of the paper *Semantic Enrichment of DNOTAMs to Reduce Information Overload in Pilot Briefings*

Languages

German	Native
Serbo-Croatian	Native
English	C1

IT-Skills

Programming	Java, C, Python
Semantic technologies	OWL, RDFs, SPARQL, ObjectLogic, datalog, F-Logic
Web Development	HTML, CSS, JavaScript, PHP, jQuery, AJAX
Databases	SQL (Oracle), PL/SQL, XQuery, XPath
Text processing	OpenOffice, LaTeX, Microsoft Office
Imaging	Adobe Photoshop, GIMP
Administration	Knowledge in handling Linux and Windows systems
	Knowledge of network administration

Awards & Certificates

12/2015 Merit scholarship for the master's program

02/2015 ERP4 Students, Integrated Business Processes with SAP ERP (TERP10)

01/2015 Uni Management Club Linz, Management Acadmia

03/2014 IBM DB2 Academic Associate, DB2 Database and Application Fundamentals

06/2009 Cambridge ESOL Level 1 Certificate in ESOL International