UN MODELO Y LENGUAJE DE CONSULTA GENÉRICO PARA EL PROCESAMIENTO ANALÍTICO ONLINE Y SU APLICACIÓN A CAMPOS DE DATOS CONTINUOS

by

Silvia Alicia Gómez

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

Major Subject: Ingeniería en Informática

at

INSTITUTO TECNOLÓGICO DE BUENOS AIRES

Buenos Aires, Argentina 2 de Junio, 2014

© Copyright by Silvia Alicia Gómez, 2014

Instituto Tecnológico de Buenos Aires

Faculty of Ingeniería

Department of Ingeniería en Informática

The undersigned hereby certify that they have examined, and recommend to the Faculty of Graduate Studies for acceptance, the thesis entitled "**Un Modelo y Lenguaje de Consulta Genérico para el Procesamiento Analítico Online y su Aplicación a Campos de Datos Continuos**" by **Silvia Alicia Gómez** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy** .

Dated: _____

Supervisor:

_____

Ph.D. Alejandro Vaisman

Examiners:

_____

Ph.D. Mónica Alejandra Caniupán Marileo

_____

Ph.D. Adriana Marotta

_____

Ph.D. Oscar Romero Moral

Instituto Tecnológico de Buenos Aires

Faculty of Ingeniería

DATE: _____

AUTHOR:	Silvia Alicia Gómez
TITLE:	**Un Modelo y Lenguaje de Consulta Genérico para el Procesamiento Analítico Online y su Aplicación a Campos de Datos Continuos**
MAJOR SUBJECT:	Ingeniería en Informática
DEGREE:	Doctor of Philosophy
CONVOCATION:	Junio, 2014

Permission is herewith granted to Instituto Tecnológico de Buenos Aires to circulate and to have copied for non-commercial purposes, at its discretion, the above thesis upon the request of individuals or institutions.

_____

Signature of Author

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

The author attests that permission has been obtained for the use of any copyrighted material appearing in the thesis (other than brief excerpts requiring only proper acknowledgement in scholarly writing), and that all such use is clearly acknowledged.

Table of Contents

Abstract **ix**

Abstract **xi**

Acknowledgements **xiii**

List of Figures **xviii**

Chapter 1 Introduction **1**

1.1 On Line Analytical Processing (OLAP) 2

1.2 Spatial Data and Geographic Information Systems (GIS) 4

1.3 SOLAP: Putting Together Spatial and OLAP Data 6

1.4 Running Example 8

1.5 Thesis Contribution 12

1.6 Thesis Organization 14

Chapter 2 Related Work **16**

2.1 Cube-Based OLAP Models 17

2.1.1 Query Languages for OLAP 21

2.2 SOLAP: Spatial OLAP 23

2.3 Continuous Fields 24

2.4 Summary 27

Chapter 3 A Formal Model for Data Cubes **28**

3.1 Dimension Schema and Instance 28

3.2 Data Cube Schema and Instances 35

3.3 Data Types for Level Descriptors and Measures 41

3.4 Summary 43

Chapter 4 The Cube Algebra 44

4.1 Instance Preserving Operations 44

4.1.1 Roll-up Operator 45

4.1.2 Drill-down Operator 46

4.2 Instance Generating Operators 47

4.2.1 Dice Operator 47

4.2.2 Slice Operator 53

4.2.3 Drill-across Operator 56

4.3 Summary 62

Chapter 5 Extending the Drill-across Operator 63

5.1 Semantic Mapping between Cubes 64

5.2 Semantic Compatibility of Cuboids 69

5.3 Extended Drill-Across Operation 72

5.4 Discussion 78

5.5 Semantic Mapping and OLAP operations 79

5.5.1 Semantic Mapping and Roll-Up 80

5.5.2 Semantic Mapping and Drill-down 80

5.5.3 Semantic Mapping and Dice 80

5.5.4 Semantic Mapping and Slice 80

5.5.5 Semantic Mapping and Drill-across 81

5.6 Summary 81

Chapter 6 A Case Study 82

6.1 Cubes of the Case Study 82

6.2 Queries over Individual Cubes 87

6.3 Integrated Queries 90

6.4 Continuous Data and Cubes 98

6.5 Summary 99

Chapter 7 Continuous Fields: Algebra and Data Model 100
7.1 Continuous Fields . . . 100
7.2 Tesselations . . . 104
7.2.1 Voronoi Tessellation . . . 105
7.2.2 Raster Tesselation . . . 106
7.2.3 TIN (Triangulated Irregular Networks) . . . 108
7.3 Spatial and Spatio-Temporal DFields . . . 109
7.4 A Closed Generic Map Algebra for DFields . . . 111
7.4.1 Generic Local Operator . . . 115
7.4.2 Generic Focal Operator . . . 116
7.4.3 Generic Zonal Operator . . . 119
7.5 Summary . . . 122

Chapter 8 Modeling Continuous Fields as Data Cubes 124
8.1 Conceptual Representation of Fields as OLAP Cubes . . . 125
8.2 Operations over FOLAP Cuboids at the Logical Level . . . 131
8.2.1 Roll-up over FOLAP Cuboids . . . 134
8.2.2 Dice over FOLAP Cuboids . . . 145
8.2.3 Slice over FOLAP Cuboids . . . 149
8.2.4 Drill-down over FOLAP Cuboids . . . 152
8.2.5 Drill-Across over FOLAP Cuboids . . . 153
8.3 Summary . . . 161

Chapter 9 GOLAP-QL: A Generic OLAP Query Language 162
9.1 GOLAP-QL Syntax . . . 162
9.2 GOLAP-QL Semantics . . . 165
9.3 GOLAP Engine Architecture . . . 170
9.3.1 Lexical and Syntactic Analyzers . . . 172
9.3.2 Semantic Analyzer . . . 175
9.3.3 GOLAP Optimizer . . . 178
9.3.4 GOLAP Engine . . . 181

9.4 Example GOLAP-QL Queries 183
9.5 Summary 194

Chapter 10 Model Implementation 197
10.1 Schema Implementation 197
10.1.1 DField Metadata 199
10.2 Data Preparation 202
10.2.1 DFields 202
10.2.2 OLAP Cubes 203
10.3 GOLAP Implementation 204
10.4 Experimental Results 207
10.5 Summary 211

Chapter 11 Conclusions and Open Research Directions 212
11.1 Conclusions 212
11.2 Open Research Directions 214

Appendices 216

Appendix A Adapter Evaluator Algorithm for Semantic Checking 217

Appendices 217

Bibliography 221

Abstract

El análisis de los datos históricos es crucial para la gestión estratégica y la toma de decisiones en diferentes tipos de organizaciones, desde empresas comerciales hasta entidades gubernamentales o civiles. A diferencia de los inicios, la proliferación actual de datos útiles supera los límites de las propias organizaciones. Por otra parte, los datos que se incorporan al análisis organizacional son muy complejos , involucrando imágenes, funcionalidades geográficas, mapas satelitales, web logs, información de redes sociales y datos de bioinformática, entre otros.

Para la toma de decisiones, los datos son tradicionalmente consolidados y almacenados en grandes repositorios, generalmente denominados *data warehouses*. Los mismos están organizados siguiendo el modelo multidimensional, lo cual permite percibir los datos como cubos, en los cuales cada eje corresponde a una dimensión de análisis (que proporciona el contexto) y cada elemento, denominado celda, contiene uno o varios valores representando las métricas para las coordenadas correspondientes. Cada dimensión es organizada como una jerarquía de niveles, para permitir trabajar los datos en diferentes granularidades. Las herramientas OLAP (On Line Analytical Processing) permiten realizar consultas sobre estas bases consolidadas de manera eficiente, con la posibilidad de agregar y desagregar las métricas a lo largo de las dimensiones. Aunque en la actualidad, el acceso a datos se ve facilitado por el avance de Internet y de las redes sociales colaborativas, la gran cantidad, variedad y complejidad de estos datos hace muy difícil su manipulación por parte de los usuarios finales. Los enfoques actuales requieren consultar los diferentes tipos de datos utilizando herramientas y lenguajes apropiados para cada tipo, y luego integrar la información de manera ad hoc, con el fin de obtener el resultado deseado. De esto se desprende que, el analista debe ser capaz de manipular datos complejos y combinar los resultados parciales obtenidos en cada fuente con la información almacenada en los repositorios relacionales. Teniendo en

cuenta que los analistas están muy familiarizados con los cubos OLAP, su tarea se simplificaría notablemente si todos los tipos de datos a manipular pudieran ser percibidos como cubos de datos, sin importar su verdadera naturaleza.

A la luz de lo anterior, proponemos un modelo genérico de datos, junto con un lenguaje de alto nivel, que permite realizar consultas en un nivel conceptual, utilizando solamente los típicos operadores OLAP, bien conocidos por los analistas: roll-up, drill-down, slice, dice y drill-across.

Así, el problema que abordamos en esta tesis es la falta de un único lenguaje de consulta OLAP de alto nivel, que le permita a los usuarios finales manipular cubos (agregar y desagregar métricas, seleccionar, proyectar y combinar datos) independientemente de su contenido a nivel lógico o físico. Para resolver este problema proponemos un marco, en el cual, el usuario final percibe todo tipo de datos (es decir, datos discretos tradicionales y datos continuos) como un cubo OLAP tradicional. Además de definir formalmente el modelo conceptual multidimensional y sus operadores, también definimos un lenguaje de consulta, que denominamos GOLAP-QL , para manipular los cubos de datos con foco en las operaciones tradicionales a nivel conceptual (agregación, desagregación , selección, proyección y cruce de datos multidimensionales), sin tener en cuenta la implementación de dichos datos en los niveles lógico y físico. De esta forma, nuestro enfoque puede soportar otros tipos de datos, más allá de los tratados en esta tesis. Por ejemplo, aunque en este trabajo nos centramos en los datos espaciales, también pueden ser tratados de la misma forma datos de redes sociales. La aplicabilidad del modelo se demuestra a través del tratamiento de datos continuos espaciales, presentando un modelo discreto para los llamados *campos contínuos*, permitiendo verlos a nivel conceptual como cubos de datos tradicionales. Una colección de operadores genéricos a nivel lógico sobre este modelo discreto, brinda el apoyo necesario a la propuesta. También detallamos la implementación del modelo completo y desarrollamos un proceso de optimización de consultas basado en reglas. Como aporte final, presentamos un estudio de caso con los tipos de datos que se mencionaron anteriormente.

Abstract

The analysis of historical data is crucial for strategic management and decision making in most organizations, from commercial companies to governmental or civil entities. Unlike in the early days of decision-making systems, the proliferation of data nowadays exceeds the boundaries of the organizations themselves. Moreover, the data to be incorporated into organizational analysis are very complex, involving images, geographic features, satellite maps, web logs, social network information, bioinformatics data and so on.

For decision making, data are traditionally consolidated and stored in usually large repositories denoted data warehouses, which are organized following the multidimensional model, which allows data to be perceived as a *cube* where each axis corresponds to a dimension of analysis (which provides context) and each element, named cell, contains one or several values representing the measures for the corresponding coordinates. Each dimension is organized as a hierarchy of levels allowing to see data at different levels of aggregation. On Line Analytical Processing (OLAP) tools allow efficiently querying and navigating data in these consolidated databases, with the possibility of aggregating and deaggregating measures along the dimensions. Although at present time, data access is facilitated by the advance of the Internet and of the collaborative networks, the amount, variety and complexity of such data makes their manipulation very difficult for end users. Current approaches require querying different kinds of data using tools and languages appropriate for each kind, and integrating the information in an ad hoc manner in order to obtain the desired result. It follows that the analyst must be able to manipulate complex data and use the partial results to combine these data with information stored in relational repositories. Taking into account that analysts are very familiar with OLAP cubes, their task would be simplified if all kinds of data they manipulate can be perceived as a data cube, regardless their actual nature.

In light of the above, we propose a generic data model, together with a high-level language that allows querying data at a conceptual level using just the typical OLAP operators analyst know very well: roll-up, drill-down, slice, dice and drill-across.

The problem we address in this thesis is the lack of a single high-level OLAP query language that can allow end-users to manipulate cubes (aggregate, deaggregate, combine, slice, dice) independently of their content at the logical or physical level. To solve this problem we propose a framework where the end user perceives all kinds of data (i.e., traditional discrete data and continuous data) as a data cube. We formally define the multidimensional conceptual model and its operators. We also define a query language denoted GOLAP-QL, to manipulate the data cubes, with focus in the traditional cube operations at the conceptual level (aggregation, disaggregation, selecting, projecting and crossing of multidimensional data), without caring of how they are implemented at the logical and physical levels. Thus, our approach can support also other kinds of data, beyond the ones addressed in this thesis. For example, although in this work we focus on spatial data, social network data can also be addressed. We show the applicability of the model for addressing spatial continuous data, presenting a discrete model for so-called continuous fields, where at the conceptual level we see such data as standard data cubes. A collection of generic operators at the logical level over this model gives support to our proposal. We detail the implementation of the model and develop a process of rule-based query optimization. As a final contribution, we present a case study in the presence of the kinds of data previously mentioned.

Acknowledgements

My acknowledgment aims to consider all the people who have contributed with my thesis, one way or another. I apologize in advance for any involuntary omission.

I am deeply grateful to God for having given me a wonderful family who has taught me respect for values and appreciation for effort. I thank my parents, Carlos and Ofelia, for their permanent encouragement and pride in the gains of their daughters. I also want to thank, especially, my aunt Nélida, for her unconditional love and warmth, and her concern for my progress in my thesis.

I am enormously grateful for the love and limitless patience of my husband, Luis, who has accompanied me in every step of the way never letting me give up in my attempt. His family saw me through this task as well.

To my sister Leticia, unconditional friend throughout my life, without whose priceless collaboration I would have never achieved my objectives, my most heartfelt thanks.

I also thank my colleages/friends who stood by me during the process, giving me their support in all this time.

My appreciation to the ITBA authorities for motivating me, in all these years, to carry on with this doctorate research.

Specifically related to my doctoral activity, I want to thank Roberto Perazzo and the members of the Doctoral Committee for their constant encouragement. Thanks to Eduardo Bonelli for sharing his experiences and helping me zoom in on the initial objectives in my beginnings. I am also grateful to Esteban Zumanyi for his cordiality during my stay in Belgium.

Finally, with great gratitude I want to acknowledge the commitment, dedication and patience of my Director, Alejandro Vaisman. I have no doubt that without his guidance and advice it would have been impossible for me to conclude this thesis.

List of Figures

1.1 Flora and Fauna cube from Example 1. 3
1.2 Flora and Fauna cube with aggregation of Species dimension. 4
1.3 Gis layers 5
1.4 Air route polyline 7
1.5 Convective activity satellite image 7
1.6 Available data for harvest analysis 10
1.7 The analyst view of data with our approach 13

3.1 TimeDim dimension lattice. 30
3.2 BlockDim dimension lattice. 31
3.3 Members of a TimeDim Dimension instance. 33
3.4 Brabant province overlapping three regions. 33
3.5 Members of a BlockDim Dimension instance. 35
3.6 Two user visualizations of cuboid 37
3.7 Lattice of the cuboids in the Vineyard instance. 40
3.8 Three cuboids of the Vineyard instance. 41

4.1 Cuboid $V_{month-grape}$ of Vineyard. 45
4.2 ROLL-UP($V_{month-grape}$, TimeDim, Year) 46
4.3 ROLL-UP($V_{year-grape}$, BlockDim, GrapeType), ... 46
4.4 V_{top} after rolling-up to All in every dimension. 46
4.5 Dimension lattices of Drinks cube schema. 47
4.6 Dimensions lattices of Vineyard-short. 51
4.7 The cuboids of Vineyard-short 52
4.8 Dice($V_{month-grape}$, harvest $\geq$ 7.500) 53
4.9 The cuboids of Vineyard-short after dicing. 54
4.10 SLICE($V_{year-grape}$, BlockDim) 56

4.11 The cuboids of Vineyard-short instance after slicing. 57
4.12 Cuboid $\mathsf{D}_{\mathsf{year-country-product}}$ of Drinks 58
4.13 SLICE($\mathsf{D}_{\mathsf{year-country-product}}$, sales) 59
4.14 DRILL-ACROSS($\mathsf{Sliced\_V}_{\mathsf{year-grape}}$, $\mathsf{Sliced\_D}_{\mathsf{year-country-product}}$) . 61
4.15 The cuboids of Vineyard-short after drillind-across. 61

5.1 Dimension lattices of C_1 68
5.2 Dimension lattices of C_2 68
5.3 Two equivalent cuboids and their matching cells. 71
5.4 Applying the extended DRILL-ACROSS operator. 75
5.5 Possible ROLL-UP operations after drilling-across. 76
5.6 Extended DRILL-ACROSS with new Measures. 77

6.1 Two views of a trajectory in Fumigation 83
6.2 TemporalDim Dimension. 84
6.3 SpatialDim Dimension. 84
6.4 DateDim Dimension. 85
6.5 GeoDim Dimension. 85
6.6 PestDim Dimension. 85
6.7 Vineyard bottom cuboid 86
6.8 Temperature bottom cuboid 86
6.9 Precipitation bottom cuboid 87
6.10 Altitude bottom cuboid 87
6.11 Fumigation bottom cuboid 87
6.12 Resulting cuboid of Query 1. 88
6.13 Resulting cuboid for Query 2. 89
6.14 Resulting cuboid of Query 3. 90
6.15 Semantic mapping between temporal levels. 91
6.16 Semantic mapping between spatial levels. 91
6.17 Intermediate cuboid Q4-Vin3. 94
6.18 Intermediate cuboid Q4-Temp3. 94

6.19 Intermediate cuboid Q4-Fug4. 94
6.20 Intermediate cuboid Q4-Aux1. 95
6.21 Resulting cuboid for Query 4. 95
6.22 Intermediate cuboid Q5-Vin5. 97
6.23 Intermediate cuboid Q5-Alt2. 97
6.24 Intermediate cuboid Q5-Pre3. 97
6.25 Intermediate cuboid Q5-Aux1. 97
6.26 Resulting cuboid for Query 5. 98
6.27 Resulting cuboid for Query 5 with only the new measure. . 98

7.1 Graphical Representation of the BField Elevation 103
7.2 Inferring the value for a point p using a Voronoi Tesselation. 106
7.3 Example of Raster Tessellation 107
7.4 Example of TIN diagram 109
7.5 A Spatio-Temporal Discretized Field 112
7.6 Examples of DFields. 112
7.7 Star notation for swapped coordinates 115
7.8 Generic Local operator. 116
7.9 Example of Neighborhoods. 118
7.10 Generic Focal operator. 119
7.11 Example of Isopartition. 120
7.12 Generic Zonal Operator. 121
7.13 Integrated Example of Generic Map Algebra 123

8.1 FOLAP Dimension Lattices - Option 1 126
8.2 FOLAP Dimension Lattices - Option 2 126
8.3 FOLAP bottom cuboid associated to the pH SDField 129
8.4 FOLAP cube instance associated to the pH SDField 130
8.5 FOLAP bottom cuboid associated to Temperature STDField 132
8.6 FOLAP cube associated to Temperature STDField 133
8.7 GeomToField(pH,$\mathcal{G}$). 136

8.8 ROLL-UP($\mathsf{pHF}_{\mathsf{bottom}}$, SpatialDim, All) . . . 137
8.9 ROLL-UP($\mathsf{pHF}_{\mathsf{bottom}}$, SpatialDim, Region) . . . 139
8.10 ROLL-UP($\mathsf{TempF}_{\mathsf{bottom}}$, SpatialDim, All) . . . 140
8.11 ROLL-UP($\mathsf{TempF}_{\mathsf{bottom}}$, SpatialDim, Region) . . . 141
8.12 TIMETOFIELD(Temperature, $\mathcal{I}_{\mathsf{Quarter}}$). . . . 143
8.13 ROLL-UP($\mathsf{TempF}_{\mathsf{bottom}}$, TemporalDim, All) . . . 144
8.14 ROLL-UP($\mathsf{TempF}_{\mathsf{bottom}}$, TemporalDim, Quarter) . . . 146
8.15 DICE($\mathsf{pHF}_{\mathsf{bottom}}$, x < 50 and value > 6.7) . . . 147
8.16 DICE($\mathsf{pHF}_{\mathsf{bottom}}$, DISTANCE(point, POINT2D(10, 10)) < 30) . 148
8.17 DICE($\mathsf{pHF}_{\mathsf{Region}}$, AREA(rGeom) < 100) . . . 149
8.18 SLICE($\mathsf{TempF}_{\mathsf{bottom}}$, TemporalDim) . . . 151
8.19 Slice and Dice over a bottom FOLAP cube . . . 152
8.20 DRILL-DOWN($\mathsf{pHF}_{\mathsf{Region}}$, SpatialDim, Point) . . . 154
8.21 DRILL-ACROSS($\mathsf{NDVIF}_{\mathsf{bottom}}$, $\mathsf{pHF}_{\mathsf{bottom}}$) . . . 156
8.22 NDVIF_3 = DRILL-ACROSS($\mathsf{NDVIF}_{\mathsf{Region}}$, $\mathsf{pHF}_{\mathsf{Region}}$). . . . 157
8.23 DRILL-ACROSS($\mathsf{BaseStation}_{\mathsf{aux2}}$, $\mathsf{TempF}_{\mathsf{aux2}}$) . . . 159
8.24 DRILL-ACROSS($\mathsf{Vineyard}_{\mathsf{Region}}$, $\mathsf{pHF}_{\mathsf{Region}}$) . . . 160

9.1 Cuboid C_1. . . . 170
9.2 ROLLUP(C_1, BlockDim, GrapeType) . . . 170
9.3 DICE(C_1, BlockDim, GrapeType), gType = 'white') . . . 171
9.4 DRILLDOWN(DICE(ROLLUP(C_1, BlockDim, . . . , Grape) . . . 171
9.5 GOLAP Engine Architecture. . . . 172
9.6 Q1 Parse Tree. . . . 173
9.7 Q2 Parse Tree. . . . 174
9.8 Q3 Parse Tree. . . . 174
9.9 Q2 Parse Tree Navigation. . . . 177
9.10 Q3 Parse Tree Navigation. . . . 177
9.11 Parse tree for Q4. . . . 187
9.12 Optimized parse tree for Q4. . . . 188

9.13 SDField October for QbE1 189
9.14 Result cuboid for Q4 in the output console 189
9.15 Parse tree for Q5. 192
9.16 Optimized parse tree for Q5. 193
9.17 Result cuboid for Q5 in the output console 194
9.18 Parse tree for Q6 195
9.19 Result cuboid for Q6 in the output console. 196
9.20 Associated SDFields for the periods requested in Q6. 196

10.1 DFieldMetadata table 202
10.2 Rule 2 optimization over FOLAP cuboids 208
10.3 Rule 3 Optimization on FOLAP cuboids 209
10.4 Rule 4 Optimization on FOLAP cuboids 210

Chapter 1

Introduction

The analysis of historical data is crucial for strategic management and decision making in different kinds of organizations, from commercial companies to governmental or civil entities. Unlike in the early days of decision-making systems, the proliferation of useful data nowadays exceeds the boundaries of the organizations. Moreover, the data to be incorporated into organizational analysis are very complex, involving images, geographic features, satellite maps, web logs, social network information, and so on. Although at present time data access is facilitated by the advance of the Internet and of the collaborative networks, the great amount and high complexity of such data makes their manipulation very difficult for end users.

As an example, consider the management on the evolution of geographical reserves of a country over time. This task requires the integration of alphanumeric records containing data about each zone (e.g., date of creation of the reserve, description of flora and fauna), geometric data (i.e., polygons that define each zone), satellite images of environmental pollution, and so on.

As another example, the comprehensive management of airports requires information about the flight plan of the different airlines, which includes air routes (represented as geometric elements), weather conditions (represented as maps containing data about pressure, humidity, convective activity, visibility, among other ones), records of estimated and actual departure and arrival times, etc.

As a last example, the study of the correlation between a certain disease and environmental factors, such as air pollution and electromagnetic radiation emitted by telecommunication antennas, requires working with data from hospital records,

social networks, pollution data and maps reporting intensity distribution of radio-frequency fields.

In the examples above, the sources of the data involved can be of very different kinds, for which there exist very different possibilities of representation. The typical sources of alphanumerical data are the well-known relational databases. For decision making, data are consolidated and stored in usually large repositories denoted *data warehouses* [26], which are organized according the multidimensional model. Specific tools and algorithms allow efficiently querying these multidimensional databases that contain historical data. These tools conform what is usually known as On Line Analytical Processing (OLAP).

1.1 On Line Analytical Processing (OLAP)

The multidimensional model organizes data as a set of *dimensions* and *measures*. Dimensions are categories (aspects) of the analysis context according to the business perspective, and the measures represent factual data. In this model, data can be perceived as a *cube* where each axis corresponds to a dimension of analysis and each element, named cell, contains one or several values representing measures for the corresponding coordinates. Each dimension is organized as a hierarchy of levels allowing to see data at different levels of aggregation.

For instance, the flora and fauna data in our first example above, can be modeled as an OLAP cube with the dimensions Species, Geography, and Time, as shown in Figure 1.1. Dimension instances are composed of *members*. For example, 'squirrels' is a member of the dimension Species. A cell of this cube, representing a so-called *fact*, is of the form ('squirrels', 'forest reserve D', 2010,75), meaning that there are 75 squirrels in the forest reserve denoted 'D'. The first three elements in the tuple represent the dimension members, while the last one is the measure that quantifies the fact. This kind of representation allows the end user to analyze data in a very simple way. Thus, we can see in Figure 1.1 that the population of squirrels in the forest reserve 'D' has decreased from 75 to 52 during the period 2010-2013, meanwhile the ducks have decreased from 92 to 70 in the same zone and

time interval. Aggregations can be performed along dimension hierarchies. For example, species can be classified in categories, like 'flora' and 'fauna', such that, for instance, 'squirrels' and 'ducks' are at a lower level than 'fauna'. Finally, factual data in Figure 1.1 corresponding to 'squirrels' and 'ducks' for 2010, aggregated over categories, amount to 167.

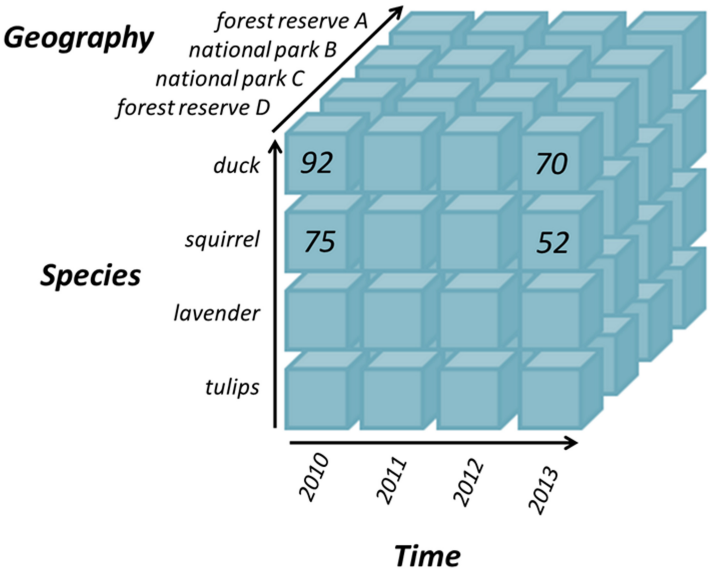


Figure 1.1: Flora and Fauna cube from Example 1.

OLAP cubes allow easily aggregating and disaggregating data. For example, if the analyst aggregates 'squirrels' and 'ducks' to the 'animal' category, as a result she detects that the animals in the forest reserve 'D' have decreased from 167 to 122. Figure 1.2 shows the user's visualization of the aggregated cube. To operate with cubes, a collection of typical OLAP operations are defined. The usual ones are:

- Roll-up: aggregates measures along a dimension hierarchy (using an aggregate function) to obtain measures at a coarser granularity.

- Drill-down: performs the operation opposite to roll-up, that is, it moves from a more general level to a more detailed level in a dimension hierarchy.

- Slice: removes a dimension in a cube, that is, a cube of $n-1$ dimensions is obtained from a cube of n dimensions.
- Dice: keeps the cells in a cube that satisfy a Boolean condition Φ.
- Drill-across: is analogous to a full outer join in the relational algebra. If the condition is not stated, it corresponds to an outer equijoin. Informally, this operation allows to combine two or more cubes, in a way that measures corresponding to the same coordinates are included in the same cell.

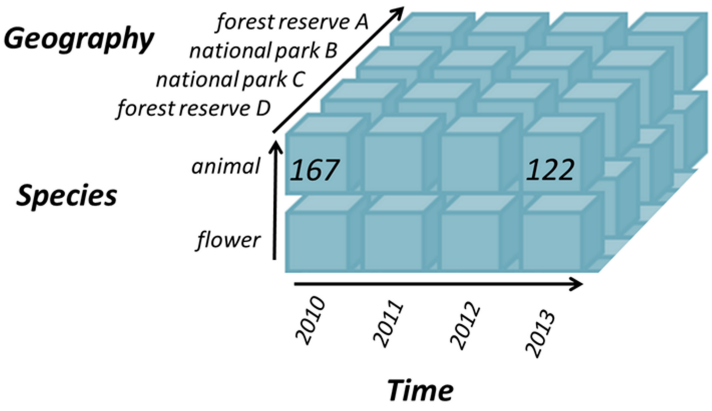


Figure 1.2: Flora and Fauna cube with aggregation of Species dimension.

The incorporation of geographical data to decision-support systems is increasingly being needed, and also facilitated by, for example, Global Position Systems (GPSs) present in any mobile device, the enormous amount of easily accessible satellite images, maps, and so on. We discuss next how this information can enhance the power of decision-support systems.

1.2 Spatial Data and Geographic Information Systems (GIS)

Geographic Information Systems (GIS) are systems designed to capture, store, display and manipulate all kinds of geographical data.

Spatial information in a GIS is typically stored in different thematic layers. For example, one layer can represent geographical accidents such as mountains, rivers,

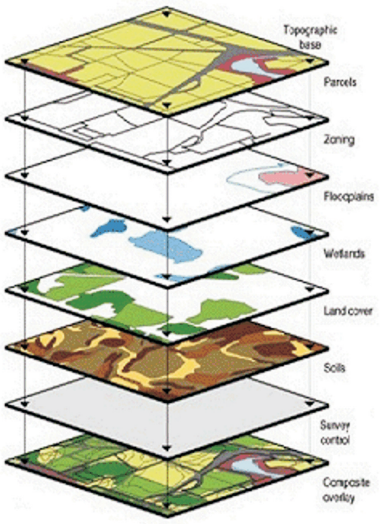


Figure 1.3: GIS layers (from `http://www.oilandgasbmps.org/resources/gis.php`)

etc., and another one can model cities and streets (See Figure 1.3). Information in themes can be stored in data structures according to different data models, the most usual ones being the *raster model* and the *vector model*. In the vector model, infinite sets of points in space are represented as finite geometric structures, or geometries, like, for example, points, polylines and polygons. There are several possible data structures to actually store these geometries. In a thematic layer, spatial data are annotated with classical relational attribute information, of (in general) numeric or string type. While spatial data are stored in data structures suitable for these kinds of data, associated attributes are usually stored in conventional relational databases. Spatial data in the different thematic layers of a GIS system can be mapped univocally to each other using a common frame of reference, like a coordinate system. These layers can be overlapped to obtain an integrated spatial view. On the other hand, in the raster model, the space is sampled into pixels or cells, each one having an associated attribute or set of attributes. In other words, spatial elements that a GIS can represent as discrete objets follow the vector model, for example, cities can be represented as points, and annotated with classical information, such as strings or numbers, for example, the surface

and population of a city. On the contrary, the raster model is used to represent continuous data, like for example, *continuous fields*, which describe phenomena that change continuously in time and/or space, like temperature, land elevation, radio-frequency intensity emission, land use, frequently used in human geography. Note that continuous fields are represented usually using the raster model, other models can also be used, as we will study later. Layers of different kinds (discrete and continuous) can be overlapped for analysis, as Figure 1.3 shows.

1.3 SOLAP: Putting Together Spatial and OLAP Data

The notion of OLAP can be extended to spatial data to allow exploring such data by drilling on maps in the same way as OLAP operates over tables and charts. In our first example example above, we could enhance the cube containing data about species, geography, and time, with spatial data containing the polygons that define each reserve zone. These polygons can also be organized into hierarchies, and therefore measures can be aggregated also along spatial hierarchies. For example, reserve areas can be aggregated into regions. Thus, we could be able to obtain the number of species by region and year, or solve spatial geometric queries, such as 'find the reserves that have decreased their areas in 10% during the period 2005-2010'. All of these can be nicely shown in a map. Another example of discrete spatial data is the 'air route' in the second example above. In this case, the route is represented as a three-dimensional (3-D) polyline, as we can see in Figure 1.4. Note however that in the former cases, the geographic (spatial) elements were defined as dimensions in the cube, although in the latter case it would be more natural to consider the route as a measure to be aggregated (although this is a design decision: representing a geometry as a dimension or as a measure has, as always, pros and cons that must be considered by the analyst). This combination of discrete spatial data and OLAP tools is denoted SOLAP (standing for Spatial OLAP), and we call the cubes containing (discrete) spatial and alphanumerical data as SOLAP cubes. In the next chapter we will describe SOLAP briefly, and provide references.

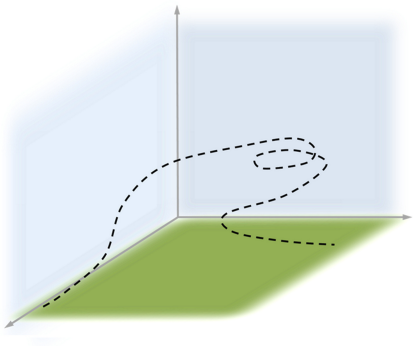


Figure 1.4: Air route polyline

SOLAP analysis can also be extended with continuous field data. For example, "convective activity" can be recorded through a succession of satellite imagery, as it is shown in Figure 1.5. Another example of continuous fields are the pollution and electromagnetic radiation used in our third example above.

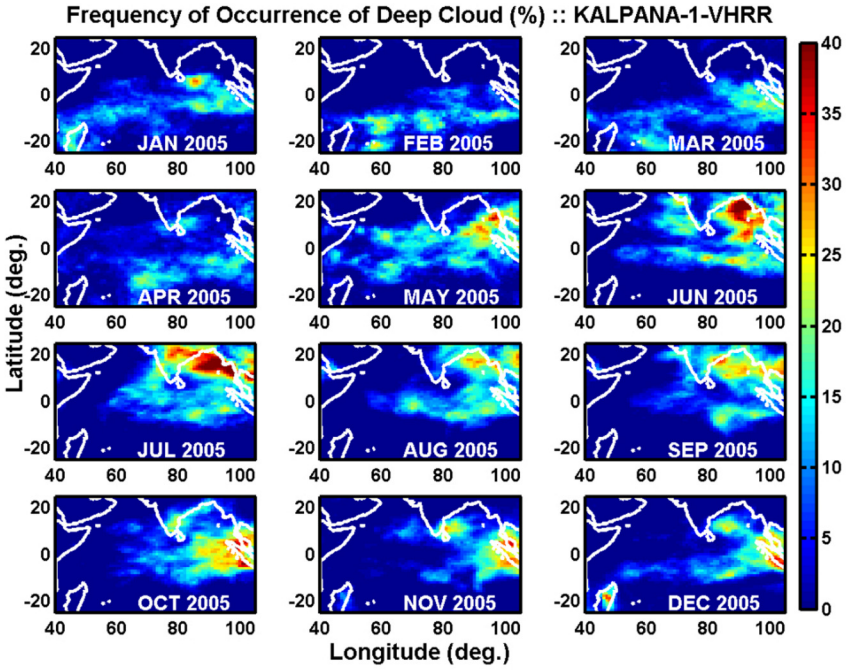


Figure 1.5: Convective activity satellite image from `http://www.intechopen.com/source/html/38768/media/image4.jpg`

Although GIS tools allow the user to manipulate spatial objects, manipulation of both geometric polygons and satellite maps, demands certain specific skills from the user. Consequently, analysts and managers should learn such complex operations, to be able to integrate spatial information in the decision-making process.

From the above, it follows clearly that in modern decision-making analysis the user must manipulate different types of data, including traditional relational databases, OLAP cubes containing alphanumerical information, discrete and continuous spatial data included in maps, data from social networks, and many more. Solving this problem is the main goal of this thesis.

1.4 Running Example

We motivate our work with an example from the wine industry that we use throughout the thesis. A vineyard producer in Belgium has developed a business analytics system in order to be able to take well-informed decisions about grape production. For example, she needs to define the more adequate times of the year for each crop, analyze the lands that historically have delivered the best production, and so on. Information is stored in SOLAP cubes, that means, geographic zones are stored as spatial dimensions in these cubes. To enhance analysis our producer wants to incorporate external information about precipitation, temperature, altitude and soil type (i.e., physical phenomena stored as continuous fields). More specifically, she knows that climate changes impact directly on wine production, and metrics such as average growing season temperatures can be used for establishing optimum regions for wine production, for each wine variety [24, 25]. For example, Cabernet Sauvignon is produced in regions that span from intermediate to hot climates, with temperatures ranging from 16.50° C to 19.50° C in the growing season.

As a concrete example, let us consider the following analysis scenario. The producer knows that the quality of the wine for each type of grape depends on the specific climatic conditions for periods such as bud break (February, March), flowering (April), ripeness (May to August) and harvest (September, October),

the conditions of altitude, and the soil type (acid, alkaline). Using this knowledge, the producer first analyzes three years of production data by zone, for each kind of grape, with respect to the altitude, temperature amplitude, average precipitation during the ripeness period, and soil information.

To perform the analysis above, the following data are needed (see Figure 1.6):

- TemperatureMin: A collection of satellite images with monthly minimum temperatures during 2010-2013
- TemperatureMax: A collection of satellite images with monthly maximum temperatures during 2010-2013
- Precipitation: A collection of satellite images with monthly precipitation (mm) during 2010-2013
- Altitude: A map with image altitudes (with respect to sea level) as of 2012
- PH: A collection of satellite images with quarterly pH value of soil during 2010-2013
- Vineyard: A SOLAP cube containing historical alphanumerical information about vineyard harvest classified by grape type and by zones of Belgium during 2010-2012

We can see that different kinds of data (alphanumerical, discrete spatial, and spatio-temporal fields) are involved in this analysis. Current approaches require querying these data using different tools and languages and integrating the information in an ad hoc manner, in order to obtain the desired result. For example, to compute the average monthly precipitation during 2012 classified by districts of Belgium, the analyst has to apply map algebra operators [48] over the precipitation images corresponding to 2012. We explain map algebra later in the thesis. In this case the operator will compute the average temperature (from all the available maps) at each point in space. After this average is computed, the analyst can compute the average temperature by district. This is performed using SOLAP operators which compute the average temperature overlapping the previous result,

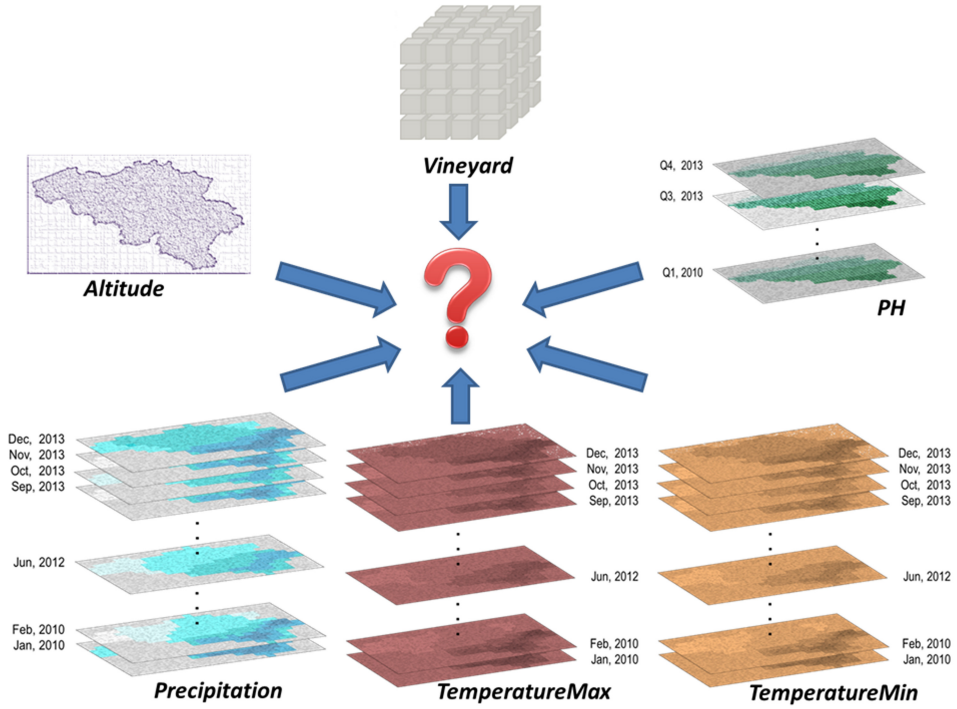


Figure 1.6: Available data for harvest analysis

and the map of Belgium containing the polygons that represent the districts. The process above must be repeated for each continuous data (e.g., precipitation).

It follows that the analyst must be able to manipulate complex data (i.e., satellite maps, geographic elements) and use the partial results to combine these data with information stored in relational repositories. All of these imply complex ad hoc manipulation for each analysis task, which turns out to be awkward and inefficient.

On the other hand, analysts are very familiar with OLAP cubes. Therefore, their task would be simplified if all kinds of data they manipulate can be perceived as a data cube, regardless their nature. To achieve this, we need a generic data model, together with a high-level language that allows querying data at a conceptual level, independently of their kinds, using just the typical OLAP operators she knows very well: roll-up, drill-down, slice, dice and drill-across. These operators can of course be enhanced with other functions that can extend their functionality.

A language like the one we propose would allow, for example, to compute the

maximum temperature by district for the period March-August, 2012, as:

```
Tmin=Slice(
       Rollup(
         Rollup(
           Dice(TemperatureMin, Month>=Mar-2012 and Month<=Aug-2012),
           Time, Year),
         Space, District),
       Time)
```

Note that even though this appears to be a simple expression, many kinds of data are involved. TemperatureMin is a *continuous* spatiotemporal field containing the minimum temperatures in space across time. The DICE operation obtains, from this field, the minimum temperatures for the desired period. Then, a ROLL-UP operation aggregates these data along the Time dimension to the level Year, and another ROLL-UP operation aggregates the result over the Space dimension and its level District (a *discrete* spatial dimension level). Finally, the Time dimension is dropped. Note that here we have mentioned three kinds of data: alphanumerical, discrete spatial, and continuous spatiotemporal. However, the language is agnostic of such kinds of data, and treats all of them in the same way: all of them *are seen as data cubes*.

Analogously, we can aggregate Vineyard cube data for 2012, according a discrete spatial dimension hierarchy, as follows:

```
V= Slice(
     Rollup(
        Dice(
           Rollup(Vineyard, Time, Year),
           Year=2012),
        Space, District),
     Time)
```

Analogously we proceed with other kinds of data like precipitation, altitude, and soil PH. The graphical schema of the problem is shown in Figure 1.7. Finally, we put together in the same cube all these data using the DRILL-ACROSS operation, as shown next.

```
Q= Drillacross(
     Drillacross(
       Drillacross(
         Drillacross(
           Drillacross(
             Drillacross(Tmin, Tmax),
             Precipitation),
           Altitude),
       PH),
   Vineyard)
```

In summary, the vineyard producer only sees cubes and does not care about the representation of the data contained in these cubes. The distinction of data types and data representations is performed at the logical and physical levels (e.g., temperature can be a field type at the logical level, and stored as a rasterized grid at the physical level). As far as we are aware of, this is a completely novel approach in the OLAP field.

1.5 Thesis Contribution

The problem we address in this thesis is the lack of a single high-level OLAP query language that can allow end-users to manipulate cubes (aggregate, disaggregate, combine, slice, dice) independently of their content at the logical or physical level.

The *first contribution* of this thesis is a framework where the end user (analyst, manager) perceives all kinds of data (i.e., traditional discrete data and continuous data) as a data cube. We formally define the multidimensional conceptual model, along with its operators. In this model, the user only sees the typical OLAP

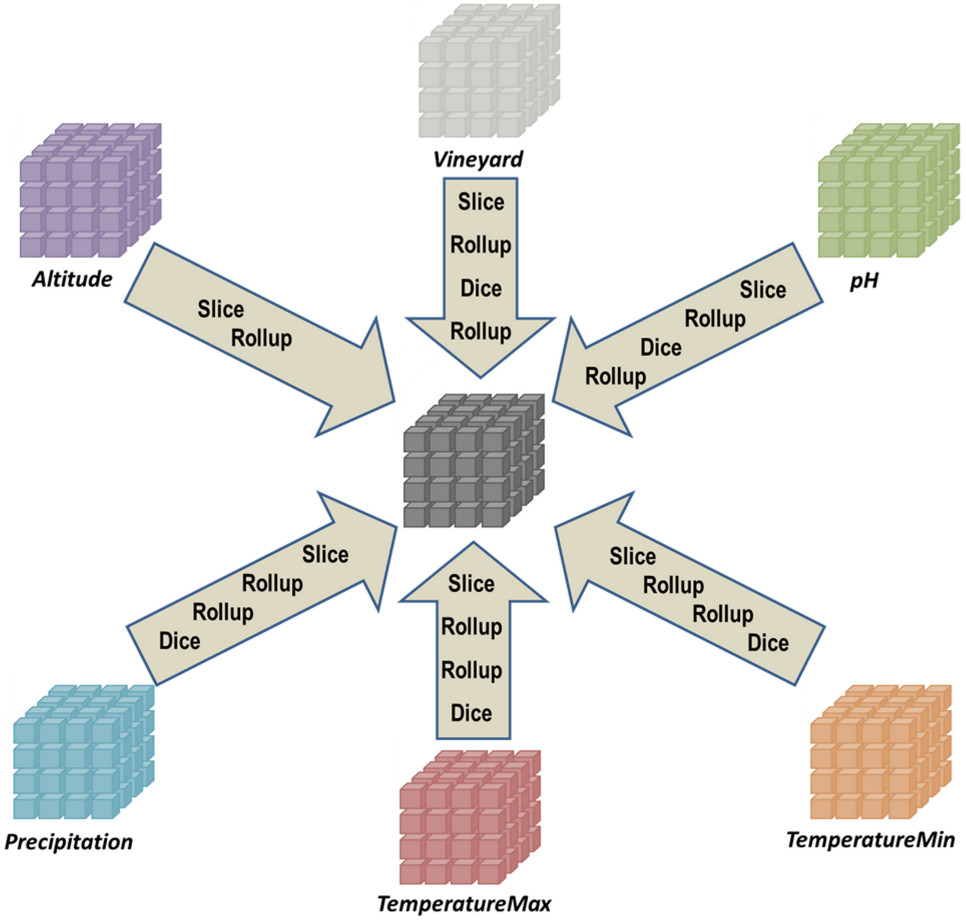


Figure 1.7: The analyst view of data with our approach

cubes and its operators without caring about the kinds of data and how they are represented.

As a *second contribution*, we define a procedural language, denoted GOLAP-QL, to manipulate the cubes above. Our approach focuses in the traditional OLAP cube operators at the conceptual level (aggregation, disaggregation, selecting, projecting and crossing of multidimensional data), defining polymorphic operators at the logical level, and only caring about the data representation at the physical level. In this sense, our approach can support also other kinds of data, beyond the ones addressed in this thesis. For example, although in this work we focus on spatial data, social network data can also be addressed. Since our approach is strongly based on the integration of all kinds of cubes, the drill-across operator is

a key component of the language, given that it is the only OLAP operator that allows combining cubes, in a way analogous to what is done in relational databases by the join operator. A typical problem with this operator is that the dimensions of the cubes involved must be identical, at the schema and instance levels. Since this is very difficult to accomplish, especially if we want to manipulate many kinds of data, we build on the work of Abelló et al. [2] to define semantic mapping mechanisms that allow overriding the limitations above. This mapping is a key component of our second contribution.

As a *third contribution* of this thesis, we show the applicability of the model for addressing spatial continuous data. We present a discrete model for continuous fields, where at the conceptual level we see such data as standard cubes with one-level dimensions. A collection of generic operators at the logical level over this model gives support to our proposal.

As a *fourth contribution* we detail the implementation of the model and develop a rule-based query optimization process.

Finally, we illustrate how this proposal can be used by means of a case study, in which we present a comprehensive set of decision-support queries in the presence of the three kinds of data previously mentioned.

As a final remark, this thesis consolidates and expands the work previously published in [16, 17, 18].

1.6 Thesis Organization

This thesis is organized as follows: In Chapter 2 we review related work in the field of OLAP modeling and operations, and Continuous Fields manipulation. In Chapter 3 we introduce the multidimensional data model over which we will build our proposal. In Chapter 4 we present an algebra whose operands are data cubes, and explain the syntax and semantics of each operator. Chapter 5 studies in detail the Drill-Across operator (which is key for our proposal) and presents mapping technique that allows to apply the operation even when the dimensions and instances of the cubes involved do not coincide. In Chapter 6 we illustrate with

an example based on our running example, the concepts studied in the previous chapters. Chapter 7 presents a discrete data model for representing continuous fields. This data model is general enough to support any kind of underlying data representation (e.g., Voronoi, Raster, TIN). In Chapter 8 we show that actually fields can be considered as another kind of data cube, and detail how each typical OLAP operation can be applied over field cubes. In Chapter 9 we propose a procedural query language, named GOLAP, that operates on data cubes. We describe query processing in GOLAP, and present a set of optimization rules. Chapter 10 discusses the implementation and presents preliminary experimental results. Finally, in Chapter 11 we summarize the main features of our contribution and we analyze open research directions.

Chapter 2

Related Work

Online Analytical Processing (OLAP) techniques allow performing complex analysis over the information stored in data warehouses. A review of the evolution of data warehouse technology reveals that research and development has mainly focused on aspects such as the construction of data warehouses, materialization, indexing, and the implementation of OLAP functionality. This view has been called system-centric [13], and led to well-established and commercialized technologies such as relational OLAP (ROLAP), multidimensional OLAP (MOLAP), and hybrid OLAP (HOLAP), at the logical and the physical abstraction levels.

However, most of the time, non-expert, unskilled users, such as managers and analysts find out that handling data warehouses and OLAP systems requires expert knowledge due to complicated data warehouse structures and the complexity of OLAP systems and query languages. This occurs for two main reassons: (a) a lack of a generic, user-friendly, and comprehensible conceptual data model; (b) available OLAP query and analysis languages such as MDX and SQL OLAP operate at the logical level and require the user's deep understanding of the data warehouse structure in order to be able to formulate queries. In a ROLAP environment, for example, the user is faced with the logical design of relational tables in terms of star, snowflake, or fact constellation schemas.

As a consequence, a generic, conceptual, and user-centric data warehouse model that focuses on user requirements is still needed, in spite of the more than twenty years that data warehouse technology has been around. Such a model must: (a)

be located above the logical level; (b) be independent of the models and technologies (ROLAP, MOLAP, HOLAP) at the logical level; (c) enable the user to generically and abstractly represent and query hierarchical multidimensional data; (d) have an associated query language based exclusively on the conceptual level, thus providing high-level query operations for the user; (e) support any kind of underlying data. The focus of this thesis is mainly on this last point. The current data explosion (a phenomena generically called *Big Data*) requires systems that can seamlessly incorporate different kinds of data: discrete, continuous, alphanumerical, geographic, textual, and so on. Typical OLAP systems support just alphanumerical data, while all other kinds of data must be incorporated ad hoc.

This thesis proposes a conceptual and user-centric data warehouse model and query language that satisfies these design criteria. In this model, the user just perceives a data cube which she can create, manipulate, update, and query, although we focus on the latter aspect. The cube is used as the user concept that completely abstracts from any logical and physical implementation details. Technically, this implies that cubes can be regarded as an abstract data type that provides cubes as the only kind of values (objects), offers high-level operations on cubes or between cubes such as slice, dice, drill-down, roll-up, and drill-across as the only available access methods, and hides any data representation and algorithmic details from the user, who can concentrate on her main interest, namely to analyze large volumes of data. In Chapter 4 we define an algebra with cubes as the only sort and a collection of unary and binary operations on cubes.

In what follows, we describe the efforts for achieving a consensus on a conceptual model for OLAP and data warehousing, as well as a standard and well defined algebra for OLAP. Since we will present geographic data as a particular case for an underlying data type, we also review research in this topic.

2.1 Cube-Based OLAP Models

Unlike the case of relational databases, where there is a difference between the conceptual design and the logical and physical implementations, multidimensional

models do not present this differentiation so clearly [13]. In fact, in many cases, the modeling of cubes involves deciding about issues at the logical level (for example, deciding star topology or snowflake for ROLAP design). This lack of identification of a true conceptual level, in which the user should only define dimensions, hierarchies, and measures, regardless of logic or implementation details, unnecessarily complicates the task of the analyst. Data models and query languages should remain simple, conceptual and at a very high level.

Vassiliadis and Sellis [54] classify models according to a collection of dimensions, like implementation (relational-oriented, cube-oriented..), language type (procedural, declarative, visual) and physical representation. Another classification is found in [34], where a complete description of the user's actual needs is given, regarding restructuring operations (handling and visualization the cube), granularity operations (aggregation and de-aggregation along an aggregation hierarchy), and data manipulation (taking into account the standard relational database operations, such as selection, projection or join).

The multidimensional model introduced by Agrawal et al. [4] is characterized by its symmetric treatment of dimensions and measures, the support of multiple hierarchies along each dimension and the possibility of performing ad hoc aggregations. A set of minimal operators is also introduced with its corresponding mapping to the SQL language. Despite the authors' attempt to provide a conceptual model, it is mainly oriented to an SQL implementation into a relational database. To address symmetrical treatment of dimensions and measures, although the authors propose the conversion from one to the other by using the operators *Push* and *Pull*, this not always works properly, because of their different nature. That is, measures are not associated with granularity levels, and aggregation functions are not applicable to them.

Cabbibo and Torlone [11] propose a formal model for dimensions which are constructed from hierarchies of dimension levels, where fact tables are functions from particular combinations of levels to measures. In this model, data are characterized from a set of so-called rollup functions, each of which maps the instances of a dimension level to instances of another dimension level, according to the desired

aggregation. The approach is completed with a multidimensional relational calculus, basically oriented to the formulation of aggregate queries. They extend their approach in [12] by introducing a graphical query language based on an extension to the relational algebra. It is worth noting that this model is closer to the logical than to the conceptual level.

Gyssens and Lakshmanan [22] present a conceptual model for OLAP-based applications by differentiating structural aspects from content. They define an algebra and its equivalent calculus, with a set of operators including selection, projection, cartesian product, summarization and restructuring operators (folding and unfolding). Similarly to [4], they present two kinds of attributes, parameters and measures, with no distinction between them. Despite providing implementation-independent algebra, the formulation is far from what a user analyst is ready to use.

Vassiliadis et al. [53] present a formal model that defines the concepts of dimensions, hierarchies and multidimensional cube. They introduce a set of cube operators and provide a mapping of the proposed model to both the relational model and the multidimensional arrays. This work includes for the first time a reference to a *Drill-down* operator which de-aggregates cube data obtained through previous aggregations. In spite of being basically a conceptual cube model, it incorporates some logic aspects, needed to maintain a base-cube (that means, a cube at the lowest granularity level) in order to ensure the correct aggregation results.

Golfarelli et al. [14, 15] present a graphical conceptual model for data warehouses, called *Dimensional Fact Model*, and propose a methodology to build the model from an Entity-Relationship schema. Their model represents the typical data warehouse concepts (dimensions, measures, hierarchies) by using a graphical notation, as an abstraction of the star schema, with a central fact entity and one graph per dimension representing attribute hierarchies. However, the authors do not identify OLAP operators, expressing queries through complex predicates that either select a subset of the aggregated fact instances or affect the way the fact instances are aggregated.

Nguyen et al. [39] use UML to map their proposed conceptual multidimensional data model to an object-oriented data model. The data model is based on the concepts of dimensions, dimension members, dimension levels, dimension schemas, dimension paths and hierarchies, dimension operators, measures, and data cubes. The authors also define the following cube operators: groupBy, jumping, rollingUp, and drilling down. However, again, this model is defined almost at the logical level, and does not introduce a full set of high-level operations, missing important operations, such as slicíng and drilling across.

Tsois et al. [50] propose a multidimensional aggregation cube based on an end-user's point of view. They describe a set of requirements for the conceptual modeling of real-world OLAP scenarios. However, the authors do not develop a formal model, nor a language supporting their model.

On of the few proposals including a query language at a conceptual level is the one by Abelló et al. [1]. Here, the data cube is defined in terms of a set of well-known concepts, such as measures and cells, dimensions and aggregation levels, and facts. An algebra is defined as a set of multidimensional operations, such as base changes, dice, slice, drill-across, roll-up and drill-down. However, this query language is not intuitive for unskilled end users, and the operations defined over the cube (change base, generalization, specialization, and derivation) are far from the knowledge and interests of managers and analysts in an OLAP scenario. In subsequent efforts, the authors present YAM^2, a multidimensional object-oriented model based on an extension of UML [3]. This model is very close to logical level, since the authors refer to fact tables and star schemas, which assume a ROLAP representation.

Malinowsky and Zimanyi [33] introduce MultiDim, a model based on the E/R model, with an intuitive graphical notation that allows representing several kinds of hierarchies (i.e., balanced, unbalanced, exclusive, alternativa, and parallel). The approach also resembles the star and snowflake schema, thus close to a relational representation.

Pardillo et al. [40, 41] specify an OLAP algebra at the conceptual level by using the object-constraint language (OCL). They also propose an automatic tracing

from the conceptual OLAP queries to a particular OLAP system by using a model-driven architecture. Although the authors argue as an advantage that analysts can use this algebra to query data warehouses without being aware of logical details, being based on OCL, the syntax is quite complex for unskilled end users.

Recently, Ciferri et al. [13] proposed a cube-centric query language, denoted Cube Algebra, which is defined at the conceptual level, and where the only sort is the data cube. Cube Algebra defines a set of operators based on the needs for cube manipulation: the well-known slice, dice, roll-up (and its inverse, drill-down) and drill-across. Note however that in [13], Cube Algebra is just sketched at a very high level of detail. This proposal, and the work of Gómez et al [16, 17, 18] are the only ones that actually consider the data cube as a first-class citizen of an OLAP data model. Moreover, the work by Gómez et al. goes further, since it addresses underlying data types other than alphanumerical, and includes a prototype implementation over continuous spatial data. This thesis builds on this work, as we will see later.

2.1.1 Query Languages for OLAP

From the study above we can conclude that none of the models discussed can be considered to operate purely at a conceptual level, hiding all implementation details. Moreover, only three of them, namely [4], [22], and [50], can be considered as based on the cube metaphor, and the latter does not even include an associated query language. The only exception is the Cube Algebra [13], which applies to the conceptual level.

Unlike it occurs in relational databases and SQL, there is no consensus on a standard query language for OLAP. Therefore, each model proposed its own languages, leading to a proliferation of operators often performing almost the same function with different names (for example, 'Destroy' in [4] and 'Slice' in [53]).

With respect to commercial tools, OLAP query and analysis languages such as MDX and SQL OLAP operate at the logical level and require the user's deep understanding of the data warehouse structure in order to be able to formulate queries. These languages are quite complex, overwhelm the unskilled user, and are

therefore inappropriate as end-user languages. Compared with languages like Cube Algebra, the MDX query looks cryptic and unintuitive, since the former operates at the conceptual level. In addition, MDX has not a clearly defined semantics. On the contrary, in this thesis we will define a very precise semantics and execution mechanism for our generic algebra.

Abelló and Romero [44] provide a summary and a comparative table of the OLAP operators available in different proposals. The authors take as a basis for the comparison the functionalities provided by natural operations like selection, roll-up, projection, drill-across, union, difference and intersection. The authors use this analysis to advocate for a multidimensional algebra allowing satisfactory navigation and querying of data contained in a warehouse.

An important issue for devising a query language for OLAP is how to combine different data cubes. Most of the operators in the many proposals are unary, that means, they receive a data cube and return a cube (roll-up. drill-down, slice, etc.). The only operation allowing to combine data cubes (analogously to the join operation in the relational model) is the drill-across. The original conception of Kimball [26] restricts its applicability, since it requires the cubes involved in the operation to have the same dimensions and identical instances of level members, which is hardly the case in real life situations. Abelló et al. [2] address this problem introducing a list of rules that allows crossing cubes, even if they do not share dimensions or members at first sight. However, it is worth noticing that no solution for mapping cube cells has been proposed. One of our contributions is, precisely, the definition of such a mapping. We present an in-depth study of the drill-across operation, because we do not only use this operation to combine typical alpanumerical data cubes, but also data cubes containing data of different kinds (e.g., cubes with spatial data and cubes with alphanumerical data).

2.2 SOLAP: Spatial OLAP

We already introduced the notion of spatial information and Geographic Information Systems (GIS). Geographic Information Systems (GIS) have been extensively used in various application domains, ranging from economical, ecological and demographic analysis, to city and route planning [55]. Rigaux et al. [42] survey various techniques, such as spatial data models, algorithms, and indexing methods, developed to address specific features of spatial data that are not adequately handled by mainstream database technology.

Modern organizations need sophisticated GIS-based Decision Support System (DSS) to analyze their data with respect to geographic information, represented not only as attribute data, but also in maps. Thus, OLAP and GIS vendors are increasingly integrating their products. In this scenario, classical aggregate queries (like "total sales of cars in California"), and aggregation combined with complex queries involving geometric components ("total sales in all villages crossed by the Schelde river and within a radius of 100 km around Antwerp") must be efficiently supported. Navigation of the results using typical OLAP operations like roll-up or drill-down is also required.

Rivest et al. [43] introduced the concept of SOLAP (standing for Spatial OLAP), a paradigm aimed at being able to explore spatial data by drilling on maps, in a way analogous to what is performed in OLAP with tables and charts. Also, Shekhar et al. [46] proposed MapCube, a visualization tool for spatial data cubes. MapCube is basically an operator that, given a so-called base map, cartographic preferences and an aggregation hierarchy, produces an album of maps that can be navigated via roll-up and drill-down operations.

An important issue in spatial OLAP refers to the treatment of alphanumeric attributes that represent spatial data. Although it is usual to consider only geometric attributes to be spatial, Bédard et al. [8] define that any OLAP dimension that refers to a geo-referenced object should be considered spatial. For instance, the string "Buenos Aires" must be considered spatial data, although it is not represented as a geometry. Moreover, we can define a hierarchy with spatial data, i.e., City $\rightarrow$ District $\rightarrow$ Region, but if the members of these levels do not have a

geometric domain, we cannot express queries involving spatial operators, such as area, intersects, includes, covered-by, etc. Instead, if we define geometries for one or more attributes of a dimension, they become *geometric spatial data* which allow powerful querying by using these operators.

Bédard et al. [9] present a review of the efforts for integrating OLAP and GIS, and describe the required characteristics for SOLAP.

In a more recent work, Gomez et al. [19] propose a formal data model and language for SOLAP, denoted, respectively, Piet and Piet-QL. Piet-QL allows to mention in the same query, operations over cubes and spatial operators, like intersection, union, and so on, allowing to integrate in the same query, alphanumerical cube data with spatial data, represented by geometries. In this proposal, the cube contains non-geometric spatial dimensions but the queries may combine spatial elements with external geometric elements of different GIS layers (rivers, cities, districts, provinces and regions) through a binding mechanism. As a result, queries like "products manufactured in cities crossed by at least two rivers" are possible.

Zimanyi et al. [51] go a step further and allow geometric spatial data to be represented as measures, providing greater flexibility in the modeling of complex situations.

2.3 Continuous Fields

Continuous Fields (from now on, fields) describe physical phenomena that change continuously in time and/or space, like temperature, pressure, and land elevation. Fields can be of different dimensionality, e.g., elevation in a two-dimensional (2D) spatial domain, pollution defined in a three-dimensional (3D) spatial domain, or temperature in a 4D spatiotemporal domain. More generally, the N-dimensional domain can be any continuous one. For example, we can even model time series as a unidimensional field. In real-world practice, scientists and practitioners register the value of a field taking samples at (generally) fixed locations, and inferring the values at other points in space using some interpolation

method. Different discrete data models have been proposed to represent continuous fields, based on sampling and interpolation. The most popular one is the already mentioned raster model, where the 2D space is divided into regular squares. We will study fields extensively in this thesis.

The joint contribution of the GIS and database communities to the problem of analyzing fields using OLAP models has been limited. Among this limited work, Shanmugasundaram et al. [45] propose a data cube representation that deals with continuous dimensions not needing a predefined discrete hierarchy. They focus on using the data density to calculate aggregate queries without accessing the data. This representation reduces the storage requirements, but continuity is addressed in a limited way.

In another effort, Ahmed and Miquel [6] discuss the importance of modeling multidimensional structures for field-based data and analyze how either cell values or interpolation methods can be used for inferring values at non-sampled points. Nevertheless, neither formal definitions are provided nor methods for calculating aggregating functions in the continuous cube are described. Sequels of this proposal introduce a SOLAP application supporting some form of continuous data [5, 7].

Vaisman and Zimanyi [51] present a conceptual model for SOLAP that supports dimensions and measures representing continuous fields, and characterize multidimensional queries over fields. They define a field data type, a set of associated operations, and a multidimensional calculus supporting this data type. Gomez et al. [20] proposed physical data structures for implementing this set of operators. Therefore, these proposals operate at the logical and physical levels, respectively. In another contribution, Bimonte et al. [10] recently introduced a multidimensional model that supports measures and dimension as continuous field data.

Regarding algebras for field analysis, Tomlin [48] proposed *Map Algebra*, an informal language for manipulating two-dimensional fields, represented as raster data. This language basically consists in three operators denoted *Local*, *Focal* and *Zonal*. For instance, given a collection of grids, and an aggregate function,

the *Local* operator returns a grid such that the value in each cell is the result of applying the aggregate function to values of the cell at the same location in the input grids. For example, the aggregate function MAX, applied as a local operator takes several input grids and generates a new one where the value of each cell `c` is calculated as the maximum value at the same cell position, among all the input grids. *Focal operators* receive a field and a region (represented as a contiguous group of cells), and generate a new grid where the value at each cell is calculated by summarizing with some function the values of the neighbor cells in the input grid. For example, the MAX function applied as a focal operator takes an input grid and generates a new one where the value at each cell `c` is the maximum value among the neighboring cells of `c` in the input grid. The neighborhood can take any form, such as a square, rectangle, circle, surrounding each cell. Finally, *Zonal operators* receive two input raster data fields, such that one of them plays a reference role, and produce a *table* where all the values of the cells in the non-referential grid are summarized (taking into account the values in the referential field) using some function. That means, those cells with the same value determine a zone. Note that this operator causes the algebra to be not closed, since it receives fields but produces a table.

For example, the MAX function could be applied as a zonal operator summarizing values in cells of the non-referential grid that are not necessary contiguous, but which have the same characteristic (e.g., they belong to the same country).

Map Algebra operators have been implemented in different GIS (Geographic Information Systems), like ArcGIS[1].

Mennis and Viger [37] generalized the concept of Map Algebra, adding the temporal dimension. More recently, Mennis [36] proposed the Multidimensional Map Algebra, an extension of Map Algebra that allows working with raster data of different dimensionality. The proposal also discusses the local, focal, and zonal operators in this mixing of raster data and how to define neighborhoods and lags. For example, a Grid and a 4-D HyperCube can participate of operations. However, the proposal is restricted to raster data.

[1] `http://www.esri.com/software/arcgis/index.html`

Regarding continuous data representation, the main drawback of the raster model is that the value assigned to a cell may represent *many* sampled points (i.e., the actual sampled values are lost). That means, the space is arbitrary partitioned (technically, tessellated) without considering the sampled points. Therefore, alternatives to raster data have been discussed in the literature. For example, Ledoux and Gold [30] argue that using *Voronoi diagrams* for representing fields has several advantages, and they redefine the local, focal and zonal operations when space is tessellated using Voronoi diagrams.

Later, Ledoux [52] proposed a solution for creating a three-dimensional discrete Voronoi diagram and discussed its implementation. A Voronoi based Map Algebra is defined *only* for this field representation, not allowing mixing fields of different kind or dimensionality. Other discrete representation exist (like TIN, mentioned in Section 1). We are not aware of any algebra supporting TIN.

2.4 Summary

In this section we have given a brief overview of the main topics addressed in this thesis: Online Analytical Processing (OLAP) and the conceptual and logical data models and query language for it. We also quickly reviewed spatial databases, and spatial OLAP. Finally, we provided a quick overview of continuous fields and the map algebra. In the next section, we introduce our conceptual model, which views a data warehouse as a data cube, agnostic of any kind of data type.

Chapter 3

A Formal Model for Data Cubes

We have already explained that in the multidimensional model, data are perceived as a data cube whose axes are *dimensions* along which facts are analyzed. Each coordinate in this cube is associated with one or more measures, altogether representing a fact. In addition, each dimension is organized as a hierarchy of levels, which allows to visualize data at different levels of aggregation. In this chapter we formalize the notion of a data cube, and give a precise semantics for it. Like in database theory, and following the usual approach in data warehouse literature, we define cubes as composed of a schema, which defines the structure, and instances, which represent the actual data.

3.1 Dimension Schema and Instance

We start by defining the notion of dimensions.

Definition 1 (Dimension Schema)**.** A *dimension schema* is a tuple $\langle \mathsf{nameD}, \mathcal{L}, \rightarrow \rangle$ where:

(a) nameD is a literal that identifies the dimension

(b) $\mathcal{L}$ is a finite set of tuples $\langle \mathsf{name}_l, \mathsf{A}_l \rangle$, called levels, where name_l identifies univocally a level in $\mathcal{L}$, and $\mathsf{A}_l = \langle \mathsf{a}_1, \ldots, \mathsf{a}_m \rangle$ is a tuple of attribute names of the level, called level descriptors, each one of them having domain $\mathrm{Dom}(\mathsf{a}_j)$, $1 \leq j \leq m$

(c) $\langle \mathcal{L}, \rightarrow \rangle$ denotes a bounded lattice (that is, a lattice with a unique bottom level and a unique top level) where $\rightarrow$ is a partial order that defines the aggregation relation (from now, rollup relation) between pairs of levels of $\mathcal{L}$

(d) The top level of $\mathcal{L}$ is $\langle \mathsf{All}, \langle \mathsf{all} \rangle \rangle$.

□

Remark 1. *For simplicity, we refer to a level* $\langle \mathsf{name}_l, \mathsf{A}_l \rangle$ *using only its name* name_l. *For example, the level* $\langle \mathsf{Day}, \langle \mathsf{date}, \mathsf{dayOfWeek} \rangle \rangle$ *is named* Day. □

Remark 2. *To avoid ambiguity, each attribute is prefixed with the name of its corresponding dimension and its corresponding level. For example, if the descriptor* name *is used for both* District *and* Province *levels in a* BlockDim *dimension, when it appears in expressions its complete path must be expressed as, i.e.,* BlockDim.Province.name *and* BlockDim.District.name. *Otherwise, if the descriptors for* Province *and* District *are* provName *and* distName, *respectively, they are unique and it is not necessary to add a prefix. In what follows, we will use the full path format when needed.* □

For the rest of the document, the following sets apply: $\mathbb{S}$ is the set of Strings, $\mathbb{I}$ is the set of Integers, $\mathbb{R}$ is the set of Reals, $\mathbb{G}$ is the set of Degrees, $\mathbb{D}$ is the set of Dates, $\mathbb{T}$ is the set of Timestamps, $\mathbb{P}_{2D}$ is the set of de 2D polygons, $\mathbb{P}_{3D}$ is the set of de 3D polygons.

Example 1 (TimeDim Dimension Schema)**.** The schema of the TimeDim dimension is defined next.
$\langle \mathsf{TimeDim}, \{\langle \mathsf{Day}, \langle \mathsf{date}, \mathsf{dayOfWeek} \rangle \rangle, \langle \mathsf{Month}, \langle \mathsf{month} \rangle \rangle, \langle \mathsf{Year}, \langle \mathsf{year} \rangle \rangle, \langle \mathsf{All}, \langle \mathsf{all} \rangle \rangle\}, \rightarrow \rangle$, where:

- The order of the lattice is given by Day$\rightarrow$Month, Month $\rightarrow$Year, Year$\rightarrow$All
- $\mathsf{Dom}(\mathsf{date}) = \mathbb{S}$, $\mathsf{Dom}(\mathsf{dayOfWeek}) = \mathbb{S}$, $\mathsf{Dom}(\mathsf{month}) = \mathbb{S}$, $\mathsf{Dom}(\mathsf{year}) = \mathbb{I}$

Figure 3.1 shows the lattice graphically. □

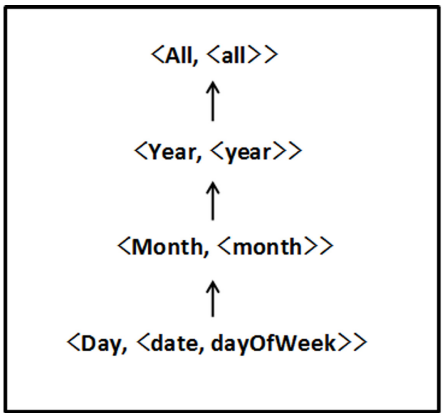


Figure 3.1: TimeDim dimension lattice.

Example 2 (BlockDim Dimension Schema)**.** We next describe the design of the BlockDim dimension.
$\langle \mathsf{BlockDim}, \{\langle \mathsf{Block}, \langle \mathsf{idBlock}, \mathsf{bGeom}\rangle\rangle, \langle \mathsf{District}, \langle \mathsf{dName}, \mathsf{dGeom}\rangle\rangle,$
$\langle \mathsf{Province}, \langle \mathsf{pName}, \mathsf{pGeom}\rangle\rangle, \langle \mathsf{Region}, \langle \mathsf{rName}, \mathsf{rGeom}\rangle\rangle,$
$\langle \mathsf{Grape}, \langle \mathsf{gName}\rangle\rangle, \langle \mathsf{GrapeType}, \langle \mathsf{gType}\rangle\rangle, \langle \mathsf{All}, \langle \mathsf{all}\rangle\rangle\}, \rightarrow\rangle$, where

- The lattice is Block→District, District→Province, Province→Region, Region→All, Block→Grape, Grape→GrapeType, GrapeType→All.

- $\mathrm{Dom}(\mathsf{idBlock}) = \mathbb{I}$, $\mathrm{Dom}(\mathsf{dName})=\mathbb{S}$, $\mathrm{Dom}(\mathsf{pName})=\mathbb{S}$, $\mathrm{Dom}(\mathsf{rName})=\mathbb{S}$, $\mathrm{Dom}(\mathsf{bGeom})=\mathbb{P}_{2D}$, $\mathrm{Dom}(\mathsf{dGeom})=\mathbb{P}_{2D}$, $\mathrm{Dom}(\mathsf{pGeom})=\mathbb{P}_{2D}$, $\mathrm{Dom}(\mathsf{rGeom})=\mathbb{P}_{2D}$, $\mathrm{Dom}(\mathsf{gName})=\mathbb{S}$, $\mathrm{Dom}(\mathsf{gType})=\mathbb{S}$.

Figure 3.2 shows the lattice graphically. □

In order to instantiate the dimension schema, each level descriptor is populated with actual elements belonging to its domain. Besides, the relations among the members must respect the partial order given by the lattice.

Definition 2 (Dimension Instance)**.** A *dimension instance* of a dimension schema $\langle \mathsf{nameDS}, \mathcal{L}, \rightarrow\rangle$ is a tuple of the form $\langle\langle \mathsf{nameDS}, \mathcal{L}, \rightarrow\rangle, \mathcal{T}_{\mathsf{L}}, \mathcal{R}\rangle$ such that:

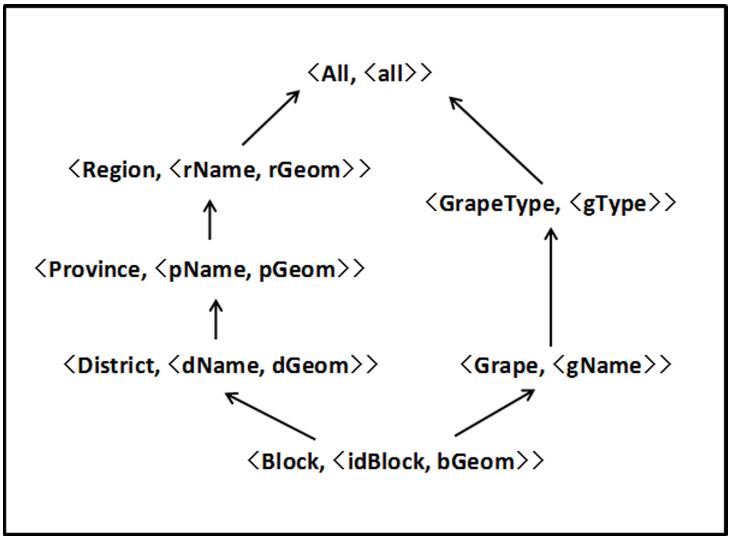


Figure 3.2: BlockDim dimension lattice.

(a) A finite set $\mathcal{T}_{\mathsf{L}}$ of tuples of the form $\langle \mathsf{v}_1, \mathsf{v}_2, \ldots, \mathsf{v}_{n_l} \rangle$ $\forall \mathsf{L} = \langle \mathsf{L}, \langle \mathsf{a}_i \ldots \mathsf{a}_n \rangle \rangle \in \mathcal{L}$ such that $\forall i, i = 1 \ldots n, \mathsf{v}_i \in Dom(\mathsf{a}_i)$. The tuples of each level are called members of this level. The level All has the unique member $\langle \mathsf{all} \rangle$

(b) A set $\mathcal{R}$ of functions $\text{RUP}_{\mathsf{L}_i}^{\mathsf{L}_j} : \mathcal{T}_{Li} \rightarrow \mathcal{T}_{Lj}$, $\forall \mathsf{L}_i, \mathsf{L}_j \in \mathcal{L}$, such that $\mathsf{L}_i \rightarrow \mathsf{L}_j$ in the lattice $\langle \mathcal{L}, \rightarrow \rangle$, i.e. each $\text{RUP}_{\mathsf{L}_i}^{\mathsf{L}_j}$ maps members of L_i to members of L_j.

□

Each $\text{RUP}_{L_i}^{L_j}$ must be a total function, in order to satisfy the summarizability condition [32, 35]:

1. Many-to-many associations must not be used.

2. Existing many-to-one associations among levels must be total, i.e., all values contributing to the coarser level must be assigned to some member at the higher level. This condition can be enforced by including an additional value 'dummy member' which assigns a father to members lacking one at a coarser level.

3. There must be no missing values.

Remark 3. *Although at the conceptual level a level member is identified with all of its attributes, in an actual implementation a memberID can be used.*

Example 3 (TimeDim Dimension Instance)**.** A possible instance for the TimeDim Dimension schema given in Example 1 is:

- $\mathcal{T}_{\mathsf{Day}} = \{\langle \text{'2007-01-01'}, \text{'Mon'} \rangle, \ldots, \langle \text{'2011-12-31'}, \text{'Sat'} \rangle\}$
- $\mathcal{T}_{\mathsf{Month}} = \{\langle \text{'Jan-2007'} \rangle, \langle \text{'Feb-2007'} \rangle, \ldots, \langle \text{'Dec-2011'} \rangle\}$
- $\mathcal{T}_{\mathsf{Year}} = \{\langle 2007 \rangle, \langle 2008 \rangle, \langle 2009 \rangle, \langle 2010 \rangle, \langle 2011 \rangle\}$
- $\mathcal{T}_{\mathsf{All}} = \{\langle \mathsf{all} \rangle\}$
- $\mathcal{R} = \{\text{RUP}_{\mathsf{Day}}^{\mathsf{Month}}, \text{RUP}_{\mathsf{Month}}^{\mathsf{Year}}, \text{RUP}_{\mathsf{Year}}^{\mathsf{All}}\}$, with
  - $\text{RUP}_{\mathsf{Day}}^{\mathsf{Month}} = \{(\langle \text{'2007-01-01'}, \text{'Mon'} \rangle, \langle \text{'Jan-2007'} \rangle), \ldots$
    $\ldots, (\langle \text{'2011-12-31'}, \text{'Sat'} \rangle, \langle \text{'Dec-2011'} \rangle)\}$
  - $\text{RUP}_{\mathsf{Month}}^{\mathsf{Year}} = \{(\langle \text{'Jan-2007'} \rangle, \langle 2007 \rangle), \ldots, (\langle \text{'Dec-2011'} \rangle, \langle 2011 \rangle)\}$
  - $\text{RUP}_{\mathsf{Year}}^{\mathsf{All}} = \{(x, \langle \mathsf{all} \rangle) | x \in \mathcal{T}_{\mathsf{Year}})\}$

Figure 3.3 shows part of the members and their rollup relation. □

Example 4 (BlockDim Dimension Instance)**.** At the time of the independence of Belgium from the Netherlands in 1830, the Belgian territory was composed of nine provinces: Hainaut, Namur, Luxenbourg, Liege, West-Vlaanderen, Oost-Vlaanderen, Antwerpen, Limburg and Brabant. In 1920, Belgium was transformed from a unitary state to a federal state with regions and finally, in 1955, Brabant was split into two provinces (Vlaams Brabant and Wallon Brabant), and the Brussels-Capital region, avoiding the inclusion of a province in more than one region (see Figure 3.4). Thus, Hainaut, Namur, Luxenbourg, Liege and Vlaams Brabant belong to Region-Wallon, and West-Vlaanderen, Oost-Vlaanderen, Antwerpen, Limburg and Wallon Brabant belong to region Vlaams-Gewest. Brussels-Capital region is not a province, it is a region without provinces and with exactly one district that coincides with the region.

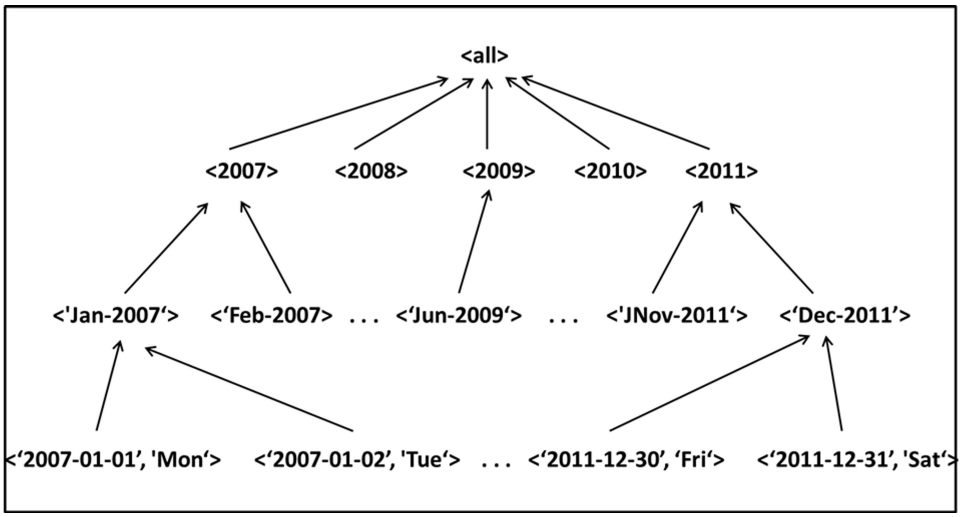


Figure 3.3: Members of a TimeDim Dimension instance.

In the case of the Brussels-Capital region, which has no provinces, we guarantee the summarizability conditions by introducing a dummy province 'Virtual Brussels' such that the Brussels district rolls-up to this 'Virtual Brussels' province and this dummy province rolls-up to Brussels Capital region. This is to avoid having lagged hierarchies.

Figure 3.4: Brabant province overlapping three regions.

In this context, a possible instance for the BlockDim Dimension given in Example 2 may be:

- $\mathcal{T}_{\mathsf{Block}} = \{\langle 35001, \mathsf{g}35001\rangle, \ldots, \langle 38543, \mathsf{g}38543\rangle\}$

- $\mathcal{T}_{\mathsf{District}} = \{\langle$'Arlon', g101$\rangle$, $\langle$'Louvain', g102$\rangle$,..., $\langle$'Brussel', g127$\rangle\}$
- $\mathcal{T}_{\mathsf{Province}} = \{\langle$'Antwerp', g11$\rangle$, ..., $\langle$'Virtual Brussel', g00$\rangle\}$
- $\mathcal{T}_{\mathsf{Region}} = \{\langle$'VlaamsGewest', g1$\rangle$, $\langle$'Brussel-Capital', g2$\rangle$, $\langle$'Wallone', g3$\rangle\}$
- $\mathcal{T}_{\mathsf{Grape}} = \{\langle$'cabernet sauvignon'$\rangle$, $\langle$'chardonnay'$\rangle$, ..., $\langle$'riesling'$\rangle\}$
- $\mathcal{T}_{\mathsf{GrapeType}} = \{\langle$'red'$\rangle$, $\langle$'white'$\rangle\}$
- $\mathcal{T}_{\mathsf{All}} = \{\langle$all$\rangle\}$
- $\mathcal{R} = \{\text{RUP}^{\mathsf{District}}_{\mathsf{Block}}$, $\text{RUP}^{\mathsf{Province}}_{\mathsf{District}}$, $\text{RUP}^{\mathsf{All}}_{\mathsf{Province}}$, $\text{RUP}^{\mathsf{Grape}}_{\mathsf{Block}}$, $\text{RUP}^{\mathsf{GrapeType}}_{\mathsf{Grape}}$, $\text{RUP}^{\mathsf{All}}_{\mathsf{GrapeType}}\}$, with
  - $\text{RUP}^{\mathsf{District}}_{\mathsf{Block}} = \{(\langle 35001, \mathsf{g1}\rangle, \langle$'Arlon', g101$\rangle), \ldots$
    $\ldots, (\langle 38543, \mathsf{g3543}\rangle, \langle$'Mechelen', g127$\rangle)\}$
  - $\text{RUP}^{\mathsf{Province}}_{\mathsf{District}} = \{(\langle$'Louvain', g102$\rangle, \langle$'Brabant', g12$\rangle), \ldots$
    $\ldots, (\langle$'Brussel', g127$\rangle, \langle$'Virtual Brussel'$\rangle)\}$
  - $\text{RUP}^{\mathsf{Region}}_{\mathsf{Province}} = \{(\langle$'Antwerp', g11$\rangle, \langle$'VlaamsGewest', g1$\rangle), \ldots$
    $\ldots, (\langle$'Wallon Brabant', g12$\rangle, \langle$'Wallone', g1$\rangle), \ldots$
    $\ldots, (\langle$'Virtual Brussel', g00$\rangle, \langle$'Brussel-Capital', g2$\rangle)\}$
  - $\text{RUP}^{\mathsf{All}}_{\mathsf{Region}} = \{(x, \langle \mathsf{all}\rangle) | x \in \mathcal{T}_{\mathsf{Region}})\}$
  - $\text{RUP}^{\mathsf{Grape}}_{\mathsf{Block}} = \{(\langle 35001, \mathsf{g1}\rangle, \langle$'cabernet sauvignon'$\rangle), \ldots, \}$
  - $\text{RUP}^{\mathsf{GrapeType}}_{\mathsf{Grape}} = \{(\langle$'riesling'$\rangle, \langle$'white'$\rangle), (\langle$'malbec'$\rangle, \langle$'red'$\rangle), \ldots, \}$
  - $\text{RUP}^{\mathsf{All}}_{\mathsf{GrapeType}} = \{(x, \langle \mathsf{all}\rangle) | x \in \mathcal{T}_{\mathsf{GrapeType}})\}$

Figure 3.5 shows part of the members and their rollup relation graphically. □

From now, $\mathcal{A}$ is a set of aggregation functions that includes the typical SQL aggregation functions, i.e. MAX, MIN, SUM, AVG, COUNT, plus possible user-defined functions.

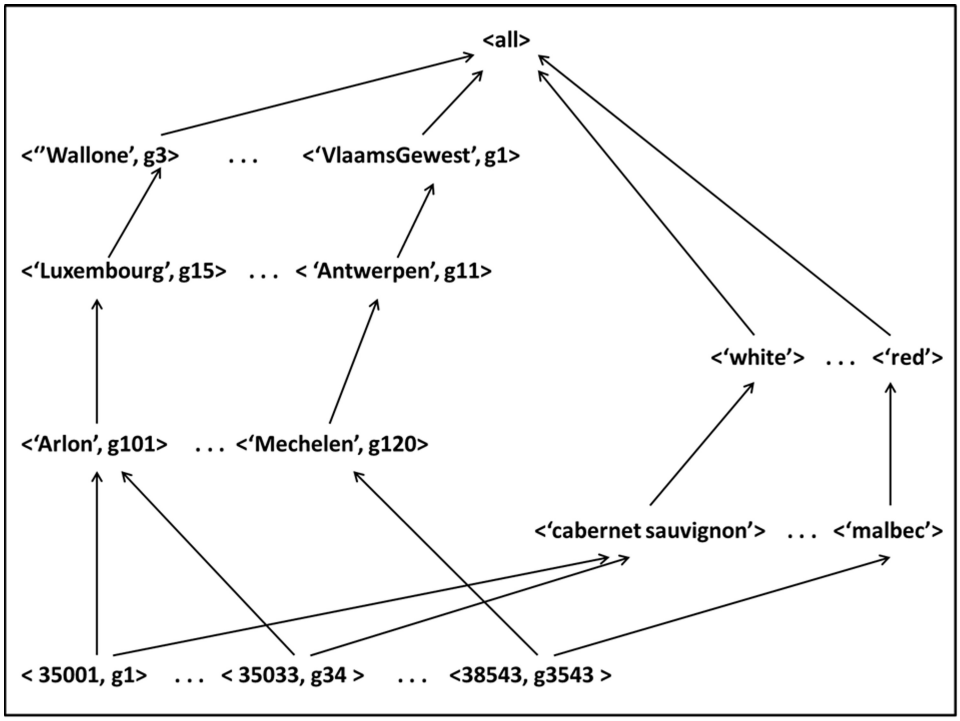


Figure 3.5: Members of a BlockDim Dimension instance.

3.2 Data Cube Schema and Instances

We are now ready to define the notion of data cube.

Definition 3 (Cube Schema)**.** A *cube schema* is a tuple $\langle \mathsf{nameC}, \mathcal{D}, \mathcal{M} \rangle$ such that:

(a) nameC is the name of the Cube

(b) $\mathcal{D}$ is a finite set of dimension schemas (see Definition 1)

(c) $\mathcal{M}$ is a finite set of attributes, where each element, called *measure*, $\mathsf{m} \in \mathcal{M}$ has domain Dom(m)

(d) A function $\mathcal{F} : \mathcal{M} \rightarrow \mathcal{A}$ maps each measure in $\mathcal{M}$ to an aggregate function in $\mathcal{A}$, i.e., each measure has exactly one aggregate function associated to it.

□

Remark 4. *The aggregate function associated to a measure must be selected according to the domain of the measure. For example, it makes no sense to associate the* AVG *function to a measure with domain String. We explain this in Section 3.3.*

Example 5 (Vineyard Cube schema)**.** We define the cube schema $\langle \mathsf{Vineyard}, \{\mathsf{TimeDim}, \mathsf{BlockDim}\}, \{\mathsf{Harvest}\}\rangle$, by considering the dimension in Examples 1 and 2, and the measure harvest with Dom(harvest)$= \mathbb{I}$ and $\textsc{Sum} \in \mathcal{A}$ as its associated aggregate function. □

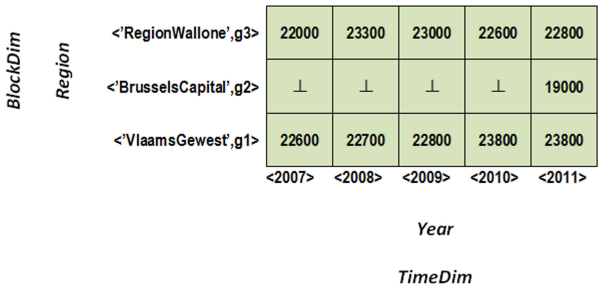
In order to define a cube instance, we need first to introduce some concepts. Intuitively, a cube instance contains measure values for all the possible combinations of the members corresponding to the levels of the dimensions included in the cube schema. That means, if we have a cube with dimensions D1 (with levels A and B), and D2 (with levels C and D), and a measure M, for which an aggregate function f applies, we can consider this cube as the union of smaller cubes, whose axes are AC, AD, BC, BD, and so on. Each of these cubes are called *cuboids*. The following definition spells out the above, and shows how the cuboids are populated.

Definition 4 (Cuboid Instance)**.** Given a cube schema $\langle \mathsf{C}, \mathcal{D}, \mathcal{M}\rangle$ where $|\mathcal{D}| = D$ and $|\mathcal{M}| = M$, and a dimension instance l_i for each $\mathsf{D}_i \in \mathcal{D}$, $\forall i, i = 1 \ldots D$, and a set $\mathcal{V}_{\mathsf{Cb}}$=$\{\mathsf{l}_1, \mathsf{l}_2, \ldots, \mathsf{l}_D\}$ where l_i is a level $\in \mathcal{L}_i$ of D_i, $\forall i, i = 1 \ldots D$, such that there are not two levels belonging to the same dimension, a *cuboid instance* Cb is a partial function Cb:$\mathcal{T}_{l_1} \times \cdots \times \mathcal{T}_{l_D} \rightarrow \mathrm{Dom}(\mathsf{m}_1) \times \cdots \times \mathrm{Dom}(\mathsf{m}_M)$ where $\mathsf{m}_k \in \mathcal{M}$, $\forall k, k = 1 \ldots M$. The elements in the domain of Cb are named *cells*, and $\mathcal{V}_{\mathsf{Cb}}$ is called the *set of levels* of the cuboid. □

Definition 4 allows to instantiate the cuboids with dimension members and measure values. For example, if we consider a cuboid composed by the bottom levels of all the dimensions in $\mathcal{D}$, a valid instantiation would assign a value of each measure to any combination of the members in the bottom level of the dimension, as the next example shows.

Example 6 (Cuboid Instance)**.** Consider the cube schema Vineyard defined in Example 5. Using the dimension instances of Examples 3 and 4 and the fact values of the grape harvest in Belgium from 2007 to 2011, we instantiate the cuboid $\mathsf{V}_{\mathsf{y-r}}$ with $\mathcal{V}_{\mathsf{V}_{\mathsf{y-r}}}$={Year, Region}. Figure 3.6 shows the two most common visualization of this cuboid, namely spatial and tabular. □

For the rest of this work, we will use the tabular representation with the following notation: the first row lists the schema dimensions, the second row indicates each dimension level and the aggregation functions for measures, and the third row gives the level descriptors and the measures. The rest of the rows lists the cells of the cuboid.



(a) Spatial representation

TimeDim	BlockDim	Measures
Year	Region	(SUM)
year	rName, rGeom	harvest
⟨2007⟩	⟨'RegionWallone', g3⟩	22000
⟨2007⟩	⟨'VlaamsGewest', g1⟩	22600
⟨2008⟩	⟨'RegionWallone', g3⟩	23300
⟨2008⟩	⟨'VlaamsGewest', g1⟩	22700
⟨2009⟩	⟨'RegionWallone', g3⟩	23000
⟨2009⟩	⟨'VlaamsGewest', g1⟩	22800
⟨2010⟩	⟨'RegionWallone', g3⟩	22600
⟨2010⟩	⟨'VlaamsGewest', g1⟩	23800
⟨2011⟩	⟨'RegionWallone', g3⟩	22800
⟨2011⟩	⟨'VlaamsGewest', g1⟩	23800
⟨2011⟩	⟨'BrusselsCapital', g2⟩	19000

(b) Tabular representation

Figure 3.6: Two user visualizations of cuboid

Definition 5 (Adjacent Cuboids)**.** Two cuboids Cb_1 and Cb_2, responding to the same cube schema, are *adjacent* if their set of levels $\mathcal{V}_{\mathsf{Cb}_1}$ and $\mathcal{V}_{\mathsf{Cb}_2}$ differ in exactly one element, i.e. they differ in exactly one level. □

Example 7 (Adjacent Cuboids)**.** Consider the cube schema given in Example 5, and the cuboids Cb_1, Cb_2 and Cb_3 given by $\mathcal{V}_{\mathsf{Cb}_1}$ ={Day, District}, $\mathcal{V}_{\mathsf{Cb}_2}$ ={Day, Region} and $\mathcal{V}_{\mathsf{Cb}_3}$ ={Month , Region}, respectively. According to the definition 5, Cb_1 is adjacent to Cb_2 and Cb_2 is adjacent to Cb_3, but Cb_1 is not adjacent to Cb_3. □

Definition 6 (Order between Adjacent Cuboids)**.** Given two adjacent cuboids Cb_1 and Cb_2 such that the difference between their set of levels is given by $l_p \in \mathcal{V}_{\mathsf{Cb}_1}$ and $l_q \in \mathcal{V}_{\mathsf{Cb}_2}$ where l_p and l_q are levels of the lattice $\langle \mathcal{L}, \rightarrow \rangle$ of the dimension D_k such that $l_p \rightarrow l_q$, then $\mathsf{Cb}_1 \preceq \mathsf{Cb}_2$.

Moreover, for each pair of adjacent cuboids $\mathsf{Cb}_1 \preceq \mathsf{Cb}_2$, a cell c in Cb_2, $c = (c_1, \ldots, c_{k-1}, c_k, c_{k+1}, \ldots, c_n, m_1, m_2, \ldots, m_s)$, is obtained from all the cells in Cb_1 as follows: $(c_1, \ldots, c_{k-1}, b_{k1}, c_{k+1}, \ldots, c_n, m_{1,1}, m_{1,2} \ldots, m_{1,s}), \ldots, (c_1, \ldots, c_{k-1}, b_{k2}, c_{k+1}, \ldots, c_n, m_{2,1}, m_{2,2}, \ldots, m_{2,s}), \ldots, (c_1, \ldots, c_{k-1}, b_{kp}, c_{k+1}, \ldots, c_n, m_{p,1}, m_{p,2}, \ldots, m_{p,s})$; where $\textsc{Rup}_{l_{k_1}}^{l_{k_2}}(b_{ki}) = c_k$, $i = 1..p$, that means, all the elements b_{ki} in level l_{k_1} in dimension k roll-up to c_k in level l_{k_2} in the same dimension, and the measures in the cell c in Cb_2 are obtained as $m_i = \textsc{Fagg}^i(m_{i,1}, \ldots, m_{i,p})$, where $\textsc{Fagg}^i$ is the aggregate function associated to measure m_i. □

Example 8 (Order between Cuboids)**.** Consider the cuboids in Example 7, $\mathsf{Cb}_1 \preceq \mathsf{Cb}_2$ because District $\rightarrow$ Region holds, and $\mathsf{Cb}_2 \preceq \mathsf{Cb}_3$ because Day $\rightarrow$ Month holds. □

Example 9 (Order between Cuboids: instances)**.** Assume a two-dimensional data cube with dimensions Geom and Time. Consider also a cuboid Cb_1, with dimension levels (axes) Region and Day, and cells representing production at region r_i at day d_i. The cells are as follows: $(r_1, d_1, 10)$, $(r_1, d_2, 10)$, $(r_1, d_3, 10)$, $(r_1, d_4, 10)$. The

other cells are similar, that means, $(r_2, d_1, 10)$, $(r_2, d_2, 10)$, $(r_2, d_3, 10)$, $(r_2, d_4, 10)$, and so on. Consider now a cuboid Cb_2, with axes Region and Month, that means, Cb_1 and Cb_2 are adjacent, and $\mathsf{Cb}_1 \preceq \mathsf{Cb}_2$. Assume also that d_1 and d_2 roll-up to m_1 and d_3 and d_4 roll-up to m_2, respectively. In the cuboid Cb_2, the cells will be $(r_1, m_1, 20)$, $(r_1, m_2, 20)$, $(r_2, m_1, 20)$, $(r_2, m_2, 20)$, and so on. □

Definition 7 (Bottom Cuboid)**.** We denote *bottom cuboid* the cuboid whose set of levels is composed of the bottom levels of each participating dimension. □

In general, when facts are defined at the finest granularity for all participating dimensions, the bottom cuboid will contain the facts.

Definition 8 (Top Cuboid)**.** We denote *top cuboid* the cuboid whose set of levels is composed of the level All of each participating dimension. That means, the instance of the top cuboid will contain the all element in all of its dimensions. □

Example 10 (Bottom and Top Cuboids)**.** Given the cube schema defined in Example 5, its bottom cuboid is defined by the set {Day, Block} and its Top cuboid is defined by the set {All, All}. □

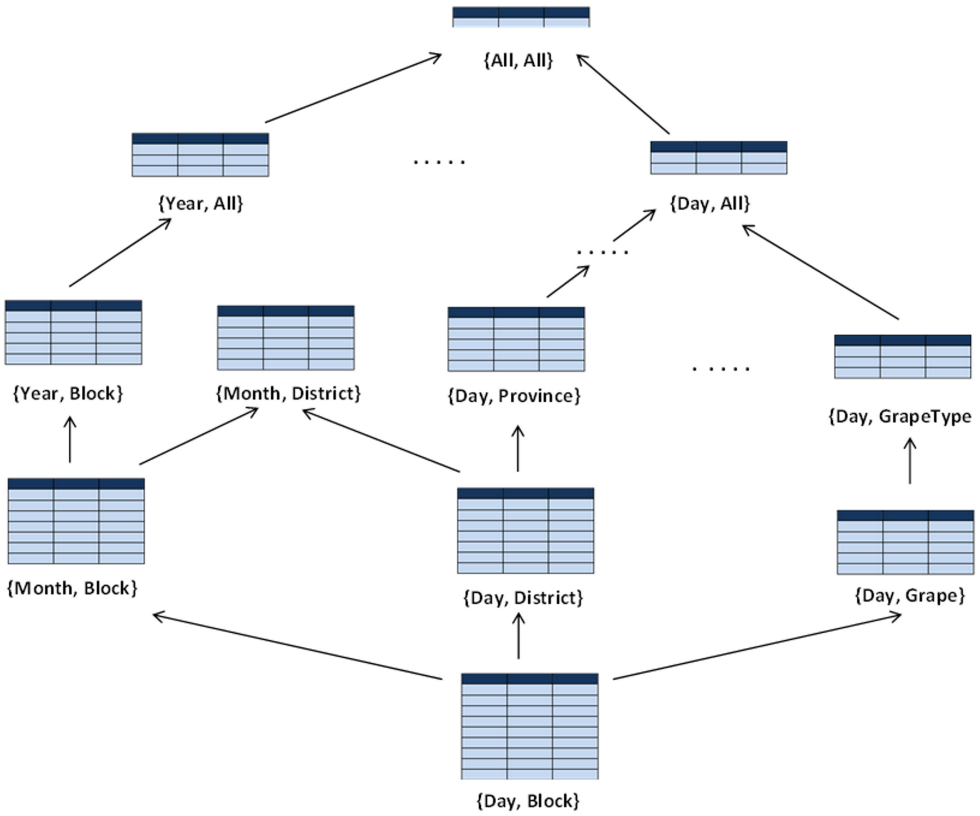
Given the former definitions, the semantics of a data cube is given as follows.

Definition 9 (Cube Instance)**.** Given a cube schema $\langle \mathsf{C}, \mathcal{D}, \mathcal{M} \rangle$ such that $|\mathcal{D}| = D$ and $|\mathcal{M}| = M$, and a dimension instance I_i for each $\mathsf{D}_i \in \mathcal{D}$, $i = 1 \dots D$, a *cube instance* CI is the lattice $\langle \mathsf{CB}, \preceq \rangle$ where CB is the set of all its possible cuboids and $\preceq$ is the order between adjacent cuboids in CB. □

Property 1 (Bounded Lattice)**.** *The lattice of cuboids $\langle \mathsf{CB}, \preceq \rangle$ of a cube instance CI, is a bounded lattice (i.e., a lattice with an unique bottom level and a unique top level) where the unique bottom element is the bottom cuboid of CI, and the unique top element is the top cuboid of CI.* □

Given a cube instance CI responding to the schema $\langle \mathsf{CI}, \mathcal{D}, \mathcal{M} \rangle$ such that $|\mathcal{D}| = D$, the total number of cuboids in CI is given by $\prod_{i=1 \dots D}(\#D_i)$ where $\#D_i$ is the number of levels of the dimension $\mathsf{D}_i \in \mathcal{D}$.

Example 11 (Cuboids of Vineyard)**.** Consider the cube schema Vineyard defined in Example 5. We instantiate its cuboids with the dimension instances of Examples 3 and 4, and the fact values of the grape harvest in Belgium from 2007 to 2011. Figure 3.7 shows part of the lattice of its twenty eight cuboids. Figures 3.8(a), 3.8(b) and 3.8(c) show a detailed tabular visualization of the cuboids $\mathsf{V}_{\mathsf{month-block}}$, $\mathsf{V}_{\mathsf{month-grape}}$ and $\mathsf{V}_{\mathsf{year-region}}$ of the cube instance Vineyard corresponding to the sets of levels $\mathcal{V}_{\mathsf{V}_{\mathsf{month-block}}}$={Month,Block}, $\mathcal{V}_{\mathsf{V}_{\mathsf{month-grape}}}$={Month,Grape} and $\mathcal{V}_{\mathsf{V}_{\mathsf{year-region}}}$={Year,Region}, respectively. □



(a)

Figure 3.7: Lattice of the cuboids in the Vineyard instance.

TimeDim Month month	BlockDim Block idBlock, bGeom	Measures (SUM) harvest
⟨Aug-07⟩	⟨35001, g35001⟩	7200
⟨Aug-07⟩	⟨35002, g35002⟩	7500
⟨Sep-07⟩	⟨35003, g35003⟩	7300
⟨Sep-07⟩	⟨35004, g35004⟩	7200
⟨Sep-07⟩	⟨35005, g35005⟩	7400
⟨Oct-07⟩	⟨35006, g35006⟩	8000
⟨Aug-08⟩	⟨35001, g35001⟩	7800
⟨Aug-08⟩	⟨35002, g35002⟩	7600
⟨Aug-08⟩	⟨35003, g35003⟩	7900
⟨Sep-08⟩	⟨35004, g35004⟩	8100
⟨Sep-08⟩	⟨35005, g35005⟩	7500
⟨Nov-08⟩	⟨35006, g35006⟩	7100
⟨Aug-09⟩	⟨35001, g35001⟩	7200
⟨Aug-09⟩	⟨35002, g35002⟩	7800
⟨Aug-09⟩	⟨35003, g35003⟩	8000
⟨Sep-09⟩	⟨35004, g35004⟩	7800
⟨Sep-09⟩	⟨35005, g35005⟩	7400
⟨Oct-09⟩	⟨35006, g35006⟩	7600
⟨Aug-10⟩	⟨35077, g35001⟩	7500
⟨Aug-10⟩	⟨35078, g35001⟩	7400
⟨Sep-10⟩	⟨35079, g35001⟩	7700
⟨Sep-10⟩	⟨35084, g35004⟩	8100
⟨Oct-10⟩	⟨35085, g35005⟩	7900
⟨Oct-10⟩	⟨35086, g35006⟩	7800
⟨Aug-11⟩	⟨35077, g35001⟩	7500
⟨Aug-11⟩	⟨35078, g35001⟩	7500
⟨Aug-11⟩	⟨35079, g35001⟩	7800
⟨Sep-11⟩	⟨35084, g35004⟩	7900
⟨Sep-11⟩	⟨35085, g35005⟩	8000
⟨Nov-11⟩	⟨35086, g35006⟩	7900

(a) Cuboid $\mathsf{V}_{\mathsf{month-block}}$

TimeDim Month month	BlockDim Grape gName	Measures (SUM) harvest
⟨Aug-07⟩	⟨kerner⟩	14700
⟨Sep-07⟩	⟨kerner ⟩	7300
⟨Sep-07⟩	⟨chardonnay⟩	14600
⟨Oct-07⟩	⟨chardonnay⟩	8000
⟨Aug-08⟩	⟨kerner⟩	23300
⟨Sep-08⟩	⟨chardonnay⟩	15600
⟨Nov-08⟩	⟨chardonnay⟩	7100
⟨Aug-09⟩	⟨kerner⟩	23000
⟨Sep-09⟩	⟨chardonnay⟩	15200
⟨Oct-09⟩	⟨chardonnay⟩	7600
⟨Aug-10⟩	⟨pinot blanc⟩	14900
⟨Sep-10⟩	⟨pinot blanc⟩	7700
⟨Sep-10⟩	⟨pinot noir⟩	8100
⟨Oct-10⟩	⟨pinot noir⟩	15700
⟨Aug-11⟩	⟨pinot blanc⟩	22800
⟨Sep-11⟩	⟨pinot noir⟩	15900
⟨Nov-11⟩	⟨pinot noir⟩	7900

(b) Cuboid $\mathsf{V}_{\mathsf{month-grape}}$

TimeDim Year year	BlockDim Region rName, rGeom	Measures (SUM) harvest
⟨2007⟩	⟨Region-Wall one, g3⟩	22000
⟨2007⟩	⟨VlaamsGewest, g1⟩	22600
⟨2008⟩	⟨Region-Wall one, g3⟩	23300
⟨2008⟩	⟨VlaamsGewest, g1⟩	22700
⟨2009⟩	⟨Region-Wall one, g3⟩	23000
⟨2009⟩	⟨VlaamsGewest, g1⟩	22800
⟨2010⟩	⟨Region-Wall one, g3⟩	22600
⟨2010⟩	⟨VlaamsGewest, g1⟩	23800
⟨2011⟩	⟨Region-Wall one, g3⟩	22800
⟨2011⟩	⟨VlaamsGewest, g1⟩	23800

(c) Cuboid $\mathsf{V}_{\mathsf{year-region}}$

Figure 3.8: Three cuboids of the Vineyard instance.

3.3 Data Types for Level Descriptors and Measures

The definition of the domain for measures and level descriptors is very important because it restricts the operations that can be applied over the instances. In our model, besides the typical well known data types (integer, float, string, date), more complex data types may be used, like geometries, moving objects, etc., with their corresponding functions. For example, if the domain of a measure or level descriptor is float, all the operations over real numbers can be applied. Otherwise, if the domain of a measure or level descriptor are the strings, the subtract operation

is not allowed. Moreover, if a moving object[1] type [21] is defined for the domain of a measure or level descriptor M, queries like M.trajectory.projectionXY.length() $\leq$ 400 are possible.

It is very important to point out that if a function F with signature F : A $\rightarrow$ B is applied to a measure or a member of a descriptor level with domain A, the possible operations over the element obtained are given by the data type of B. See Example 12.

We considered above the data type of the domain. However, there exists another point worth considering, namely the order of that data into the domain. Domains can be classified into *ordinal* and *nominal*. This restricts the possible operations on the data belonging to these kinds of domains. An ordinal domain is composed of values with an inherent order among them, although the distance between pairs of them may be unknown. For instance, the domain {Low, Medium, High} represents the grading scale of a course. A nominal domain is composed of arbitrary values and there is no specific order among them, i.e., only a distinctive categorization is given. For example, the domain {male, female}. See Example 13.

Example 12 (Domains and Functions)**.** We know the level descriptor 'rGeom' in the Region level of BlockDim of Example 4 is not ordinal, but if the function Area, with signature Area : $\mathbb{P}_{2D} \rightarrow \mathbb{R}$ is applied, we can ask relational queries over the resulting data, as Real is an ordinal domain. Thus, although we cannot ask rGeom $\geq$ Paris, the query Area(rGeom) $\geq$ Area(Paris) is possible. □

Example 13 (Nominal and Ordinal Domains)**.** The descriptor date in the Day level of TimeDim in Example 3 is ordinal, we can ask for date $\geq$ 2011-07-18 and date $\leq$ 2011-07-24. On the contrary, if p is a level descriptor with domain Point2D, we can not ask for p $\geq$ Point(1, 5), since Point2D is not an ordinal domain, but we can ask for p= Point(1, 5) or p IN {Point(3, 6), Point(1, 5), Point(8,12)}. □

In the present model, we consider the ordinal domains Integer, Real, Date and String, and the non ordinal domains Vector and Geometry. These data types can

[1] A moving object is an object that changes its position (in general, continuously) in time and space (for example a car).

be used for both, measure domains and level descriptor domains. In the case of Vector, each component may be treated according to its domain. To denote the component i of a vector V we write V[i].

3.4 Summary

In this chapter we have presented the data model we will use in the remainder, following the usual approach of considering database objects as composed of a schema and instances. Thus, we consider a data cube as composed by a schema and instances. Further, each dimension has in turn a schema and instances. Importantly, to define the semantics of the data cube we considered a data cube instance as a collection of its subcubes, denoted cuboids. There is an order between cuboids, defined by the hierarchies of the dimensions that compose the cube. This conforms a lattice of cuboids, which in the next chapter we will use to provide a precise semantics of the operations that we will define over the cubes.

Chapter 4

The Cube Algebra

In this chapter, we formally define the algebra operations supporting the data model introduced in Chapter 3. We identify two classes of operations: (a) operations that preserve the cube instance of the cuboids where they are applied, which we denote *instance preserving operations*; (b) operations that generate a new cube instance, which we denote *instance generating operations*. The Roll-up and Drill-down operations belong to the first class, whereas Dice, Slice and Drill-across belong to the second one. We detail each of them in this chapter and provide a precise semantics using the theoretical concepts introduced in the previous chapter, particularly the notion of cube lattice. In addition, we provide algorithms to produce the new instances induced by the instance generating operations.

4.1 Instance Preserving Operations

The Instance Preserving Operations (IPO) preserve the cube instance of the cuboid received as an input parameter. They act like 'navigators' within the lattice of cuboids. Roll-Up and Drill-down are IPO. These operators receive a cuboid belonging to an instance, and return another cuboid in the same instance.

In what follows we use the following sets: **C** is the set of all the cuboids, **D** is the set of all the dimensions, **M** is the set of all the measures, **L** is the set of all the dimension levels, **B** is the set of all the boolean expressions over level descriptors and measures.

4.1.1 Roll-up Operator

The ROLL-UP operator is a function with signature ROLL-UP : $\mathbf{C} \times \mathbf{D} \times \mathbf{L} \rightarrow \mathbf{C}$, that aggregates measures up to a given level within a dimension hierarchy by using the corresponding RUP function given in the instance of the input dimension.

Definition 10 (Roll-up Operator)**.** Given a cube instance CI with schema $\langle \mathsf{CS}, \mathcal{D}_{in}, \mathcal{M}_{in} \rangle$, a cuboid $\mathsf{C}_{in} \in \mathsf{CI}$, a dimension $\mathsf{D} \in \mathcal{D}_{in}$, two levels l_{in} and L of D such that $\mathsf{l}_{in} \in \mathcal{V}_{\mathsf{C}_{in}}$ and $\mathsf{l}_{in} \rightarrow^{*} \mathsf{L}$ in $\langle \mathcal{L}, \rightarrow \rangle$, ROLL-UP$(\mathsf{C}_{in}, \mathsf{D}, \mathsf{L})$ returns a cuboid $\mathsf{C}_{out} \in \mathsf{CI}$ such that $\mathcal{V}_{\mathsf{C}_{out}} = (\mathcal{V}_{\mathsf{C}_{in}} - \{\mathsf{l}_{in}\}) \cup \{\mathsf{L}\}$. Notice that $\mathsf{C}_{in} \preceq \mathsf{C}_{out}$ in the lattice CI. □

Example 14 (Roll-up operator)**.** Consider the cuboid $\mathsf{V}_{\mathsf{month-grape}}$ of the Vineyard instance given in Example 11, which is depicted in Figure 4.1. If we apply ROLL-UP$(\mathsf{V}_{\mathsf{month-grape}}, \mathsf{TimeDim}, \mathsf{Year})$ the resulting cuboid $\mathsf{V}_{\mathsf{year-grape}}$ is shown in Figure 4.2. Notice that the value of harvest has been aggregated using the function SUM associated to the measure. Figure 4.3 depicts the cuboid that results of applying ROLL-UP$(\mathsf{V}_{\mathsf{year-grape}}, \mathsf{BlockDim}, \mathsf{GrapeType})$ (a roll-up to GrapeType). Finally, in Figure 4.4 we can see the top cuboid resulting of rolling-up to level All in every dimension. □

TimeDim	BlockDim	Measures
Month	Grape	(SUM)
month	gName	harvest
⟨Aug-07⟩	⟨kerner⟩	14700
⟨Sep-07⟩	⟨kerner ⟩	7300
⟨Sep-07⟩	⟨chardonnay⟩	14600
⟨Oct-07⟩	⟨chardonnay⟩	8000
⟨Aug-08⟩	⟨kerner⟩	23300
⟨Sep-08⟩	⟨chardonnay⟩	15600
⟨Nov-08⟩	⟨chardonnay⟩	7100
⟨Aug-09⟩	⟨kerner⟩	23000
⟨Sep-09⟩	⟨chardonnay⟩	15200
⟨Oct-09⟩	⟨chardonnay⟩	7600
⟨Aug-10⟩	⟨pinot blanc⟩	14900
⟨Sep-10⟩	⟨pinot blanc⟩	7700
⟨Sep-10⟩	⟨pinot noir⟩	8100
⟨Oct-10⟩	⟨pinot noir⟩	15700
⟨Aug-11⟩	⟨pinot blanc⟩	22800
⟨Sep-11⟩	⟨pinot noir⟩	15900
⟨Nov-11⟩	⟨pinot noir⟩	7900

Figure 4.1: Cuboid $\mathsf{V}_{\mathsf{month-grape}}$ of Vineyard.

TimeDim	BlockDim	Measures
Year	Grape	(Sum)
year	gName	harvest
⟨2007⟩	⟨kerner⟩	22000
⟨2007⟩	⟨chardonnay⟩	22600
⟨2008⟩	⟨kerner⟩	23300
⟨2008⟩	⟨chardonnay⟩	22700
⟨2009⟩	⟨kerner⟩	23000
⟨2009⟩	⟨chardonnay⟩	22800
⟨2010⟩	⟨pinot blanc⟩	22600
⟨2010⟩	⟨pinot noir⟩	23800
⟨2011⟩	⟨pinot blanc⟩	22800
⟨2011⟩	⟨pinot noir⟩	23800

Figure 4.2: ROLL-UP($\mathsf{V}_{\mathsf{month-grape}}$, TimeDim, Year)

TimeDim	BlockDim	Measures
Year	GrapeType	(Sum)
year	gType	harvest
⟨2007⟩	⟨white⟩	44600
⟨2008⟩	⟨white⟩	46000
⟨2009⟩	⟨white⟩	45800
⟨2010⟩	⟨white⟩	22600
⟨2010⟩	⟨red⟩	23800
⟨2011⟩	⟨white⟩	22800
⟨2011⟩	⟨red⟩	23800

Figure 4.3: ROLL-UP($\mathsf{V}_{\mathsf{year-grape}}$, BlockDim, GrapeType),

TimeDim	BlockDim	Measures
All	All	(Sum)
all	all	harvest
⟨all ⟩	⟨all ⟩	229400

Figure 4.4: $\mathsf{V}_{\mathsf{top}}$ after rolling-up to All in every dimension.

4.1.2 Drill-down Operator

The DRILL-DOWN operator is a function with signature DRILL-DOWN : $\mathbf{C} \times \mathbf{D} \times \mathbf{L} \rightarrow \mathbf{C}$, that disaggregates measures down to a given level within a dimension hierarchy by using the corresponding RUP function given in the instance of the input dimension.

Definition 11 (Drill-down)**.** Given a cube instance CI with schema $\langle \mathsf{CS}, \mathcal{D}_{in}, \mathcal{M}_{in} \rangle$, a cuboid $\mathsf{C}_{in} \in \mathsf{CI}$, a dimension $\mathsf{D} \in \mathcal{D}_{in}$, two levels $\mathsf{l}_{in}, \mathsf{L} \in \mathsf{D}$ such that $\mathsf{l}_{in} \in \mathcal{V}_{\mathsf{C}_{in}}$ and $\mathsf{L} \rightarrow^{*} \mathsf{l}_{in}$ in $\langle \mathcal{L}, \rightarrow \rangle$, DRILL-DOWN($\mathsf{C}_{in}, \mathsf{D}, \mathsf{L}$) returns a cuboid $\mathsf{C}_{out} \in \mathsf{CI}$ such that $\mathcal{V}_{\mathsf{C}_{out}} = (\mathcal{V}_{\mathsf{C}_{in}} - \{\mathsf{l}_{in}\}) \cup \{\mathsf{L}\}$. Notice that $C_{out} \preceq C_{in}$ in the lattice CI. □

Example 15 (Drill-down operator)**.** Given the cuboid $\mathsf{V}_{\mathsf{year-grape}}$ of Figure 4.2, the result of DRILL-DOWN($\mathsf{V}_{\mathsf{aux}}$, TimeDim, Month) is the cuboid in Figure 4.1. □

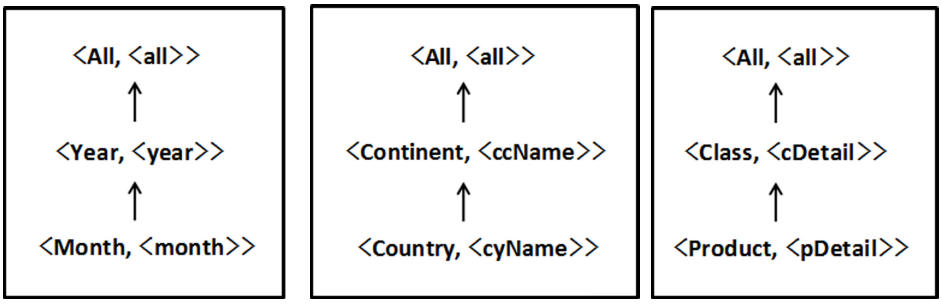
It is important to point out that, unlike the approach of Agrawal et al. [4], the Drill-down operation of Definition 11 is not a single undo from a previous Roll-up, since our Algebra allows nested operators. We explain this in detail in Chapter 9.

4.2 Instance Generating Operators

The Instance Generating Operators (IGO) build a new cube instance to which the resulting cuboid belongs to. DICE, SLICE and DRILL-ACROSS are IGO. In other words, the result of an IGO is a cuboid which induces a new cube instance.

In what follows we be also using in our examples a cube denoted Drinks, representing the consumption of drinks in bars and the sales of drinks in markets between 2008 and 2011, both in litres per person.

The schema of the Drinks cube is ⟨Drinks, {TimeDim, ZoneDim, ProductDim}, {consumption, sales}⟩. Both measures, consumption and sales have associated the aggregate function AVG $\in \mathcal{A}$. The dimension lattices are shown in Figure 4.5.



(a) TimeDim lattice (b) ZoneDim lattice (c) ProductDim lattice

Figure 4.5: Dimension lattices of Drinks cube schema.

4.2.1 Dice Operator

The DICE operator is a function with signature DICE : $\mathbf{C} \times \mathbf{B} \rightarrow \mathbf{C}$ that selects the values in dimensions or measures that satisfy a boolean condition.

Definition 12 (Dice)**.** Given a cube instance CI with schema $\langle \mathsf{CS}, \mathcal{D}_{in}, \mathcal{M}_{in} \rangle$, a cuboid $\mathsf{C}_{in} \in \mathsf{CI}$, and a boolean condition ϕ over the measures of $\mathcal{M}_{in}$, and/or the level descriptors of the levels in $\mathcal{V}_{\mathsf{C}_{in}}$, DICE$(\mathsf{C}_{in}, \phi)$ returns a new cuboid $\mathsf{C}_{out} \in \mathbf{C}$, as follows:

(a) $c_i = (c_{i_1}, \ldots, c_{i_n}, m_{i_1}, \ldots, m_{i_s}) \in \mathsf{C}_{out}$ if $\exists c_j = (c_{j_1}, \ldots, c_{j_n}, m_{j_1}, \ldots, m_{j_s}) \in \mathsf{C}_{in}$ and $c_{i_p} = c_{j_p} \forall p = 1 \ldots n$, and $m_{i_q} = m_{j_q} \forall q = 1 \ldots s$, and c_j satisfies ϕ

(b) $\mathcal{V}_{\mathsf{C}_{out}} = \mathcal{V}_{\mathsf{C}_{in}}$

The boolean condition ϕ must satisfy certain syntax rules (according to Extended Backus-Naur Form) and it must also be checked semantically in order to detect domain consistency. That is, ϕ must be coherent with the data type domains of the measures and descriptors. These syntax and semantic aspects will be explained in Chapter 9. Example 16 shows some possible boolean conditions.

Since DICE is an IGO, C_{out} induces a new cube instance CI_{out} such that $\mathsf{C}_{out} \in \mathsf{CI}_{out}$. Algorithm 1 computes this instance. It starts cloning the lattice of cuboids CI_{in}, i.e., creating a lattice of cuboids with the same schema and instance. Since the algorithm needs to remember the collection of recalculated nodes, it marks all the cuboids of C_{out} as 'not visited', and during the process each recalculated node is marked as 'visited'. First, the bottom cuboid of CI_{out} is recalculated by eliminating all the cells which aggregate to the removed cells in C_{in}. Then, all the cuboids are recalculated by propagating the aggregation of the cells of each cuboid. This aggregation is computed based on the RUP function between the members of levels for each pair of adjacent cuboids (see Definition 2), such that the preceding cuboid has been calculated previously ('visited'). The algorithm stops when all the cuboids are marked as 'visited'. □

Remark 5. *When* DICE *is applied to a cuboid, it is very important that all the cuboids of the lattice* CI_{out} *change globally, in order to ensure consistency in future operations. Since both, the bottom cuboid and the top cuboid are reachable from*

any cuboid and they ensure navigability towards any other cuboid, dicing a cuboid induces a changes in many cuboids in the lattice. For example, if DICE *eliminates the province of 'Antwerpen' in the level Province of the dimension BlockDim in a cuboid, then the districts of 'Antwerpen' cannot appear when performing* DRILL-DOWN *to the District level in the same dimension; analogously, 'Antwerpen' cannot appear when applying a* ROLL-UP *to Region. Example 17 illustrates this concept.*

Algorithm 1 New cube instance for DICE(Cb_{in}, ϕ), with $\mathsf{Cb}_{in} \in \mathsf{Cl}$ instance

Cl_{out} *schema* $\leftarrow$ Cl *schema*

Cl_{out} *instance* $\leftarrow$ Cl *instance* (i.e., clone Cl as C_{out})

for all cuboid $\mathsf{Cb}_i \in \mathsf{C}_{out}$ **do**

Set Cb_i as Not Visited

In $\mathsf{Cb}_{bottom} \in \mathsf{C}_{out}$ eliminate all cells $c_k = (c_{k1}, \ldots, c_{kn}, m_{k1}, \ldots, m_{ks})$ such that $\exists c_j = (c_{j1}, \ldots, c_{jn}, m_{j1}, \ldots, m_{js}) \in \mathsf{Cb}_{in}$ and ROLL-UP$^*(c_{kv}) = (c_{jv})$ $\forall v, v = 1 \ldots n$

Set Cb_{bottom} as Visited

repeat

if $\exists \mathsf{Cb}_j$ Not Visited $\in \mathsf{Cl}_{out}$ **then**

if $\exists \mathsf{Cb}_i$ Visited $\in \mathsf{Cl}_{out}$, adjacent to Cb_j and $\mathsf{Cb}_i \preceq \mathsf{Cb}_j$ **then**

Recalculate the cells of Cb_j as follows:

$\forall c_j = (c_{j_1}, \ldots, c_{j_n}, m_{j_1}, \ldots, m_{j_s}) \in \mathsf{Cb}_j$, $m_{j_v} = f_{agg_v}(m_{i_{1_v}}, \ldots, m_{ip_v})$, $\forall v, v = 1 \ldots s$, where f_{agg_v} is the aggregate function associated to measure m_{jv}, and $\exists\ c_{i1} = (c_{i1_1}, \ldots, c_{i1_n}, m_{i1_1}, \ldots, m_{i1_s})$, $c_{i2} = (c_{i2_1}, \ldots, c_{i2_n}, m_{i2_1}, \ldots, m_{i2_s}), \ldots, c_{ip} = (c_{ip_1}, \ldots, c_{ip_n}, m_{ip_1}, \ldots, m_{ip_s}) \in \mathsf{Cb}_i$ such that ROLL-UP$(c_{ik_r}) = c_{j_r}$ $\forall k, k = 1 \ldots p$, $\forall r, r = 1 \ldots n$

if $c_j \in \mathsf{Cb}_j$ aggregates only eliminated tuples of Cb_i **then**

c_j must be eliminated too from Cb_j

Set Cb_j as Visited

until all cuboids of C_{out} are set as Visited

Example 16 (Dice conditions)**.** Consider the cube instance Vineyard of Example 11. The following Dice conditions are possible:

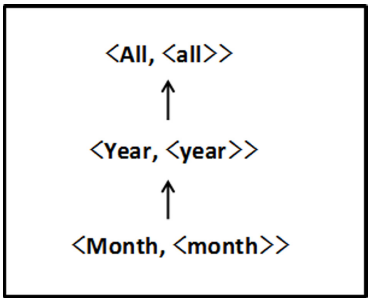
- ϕ = harvest $\geq$ 7000 AND harvest $\leq$ 8000
- ϕ = harvest $\leq$ 7500 OR gType = 'red'
- ϕ = TimeDim.Year.year = 2007
- ϕ = TimeDim.Year.year $>$ 2006 AND TimeDim.Year.year $<$ 2010
- ϕ = BlockDim.District.distName = 'Mechelen' OR BlockDim.District.distName = 'Leuven'
- ϕ = SubString(GrapeDim.Variety.gVariety, 1, 1) = 'p'
- ϕ = Area(BlockDim.Block.bGeom) $\geq$ 15000
- ϕ = MyFun(BlockDim.Block.bGeom, harvest) $<$ 1.8, where MyFun is the user-defined function MyFun(a, b) = 0.2 $*$ b/Area(a).

□

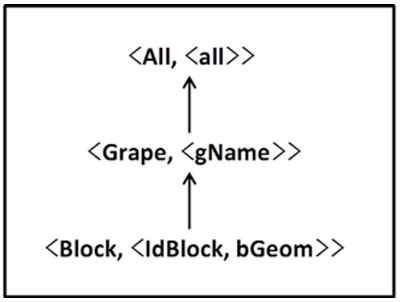
Example 17 (Dice Operator)**.** In this example we consider the cube instance Vineyard-short, a simplified version of the cube Vineyard, in order to reduce the number of cuboids and make the example more readable. In this simplified cube, the TimeDim dimension only has the levels Month, Year and All, and its BlockDim dimension only contains the levels Block, Grape and All. Figure 4.6 shows the new lattices and Figure 4.7 shows the nine cuboids of Vineyard-short instance.

Consider the cuboid $V_{month-grape}$ in Figure 4.1. Dice($V_{month-grape}$, harvest $\geq$ 7.500) returns the cuboid shown in Figure 4.8. Note that cells (⟨'Sep-07'⟩,⟨'kerner'⟩) and (⟨'Nov-08'⟩,⟨'chardonnay'⟩) disappear because their harvests are less than 7.500 (7.300 and 7.100, respectively).

Additionally, a new cube instance $Vineyard - short_{new}$ is built following Algorithm 1. Figure 4.9 shows the new cube instance.



(a) Short TimeDim Dimension



(b) Short BlockDim dimension

Figure 4.6: Dimensions lattices of Vineyard-short.

We can see that the new bottom cuboid does not contain the cells (⟨'Sep-07'⟩,⟨35003, g35003⟩) and (⟨'Nov-08'⟩,⟨35006, g35006⟩), because they agregate to the eliminated cells in $\mathsf{V}_{\mathsf{month-grape}}$.

As a consequence, in the new $\mathsf{V}_{\mathsf{year-block}}$ the cells (⟨2007⟩,⟨35003, g35003⟩) and (⟨2008⟩,⟨35006, g35006⟩) have been removed.

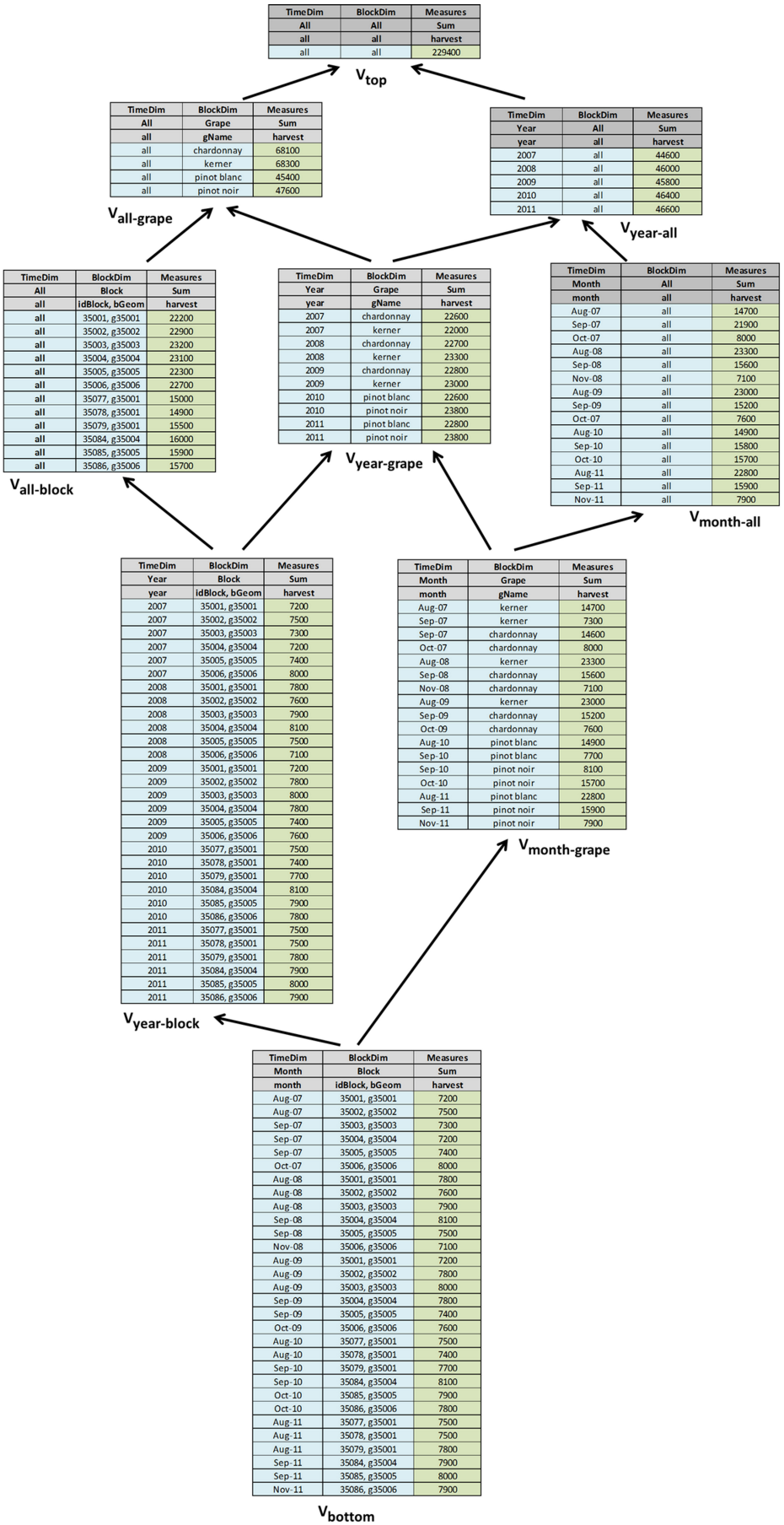
On the other hand, in the new $\mathsf{V}_{\mathsf{year-grape}}$ the cells (⟨2007⟩,⟨'kerner'⟩) and (⟨2008⟩,⟨'chardonnay'⟩) have changed their measures to 14.700 and 15.600 because the values 7300 and 7300 (corresponding to the deleted cells) are no more aggregated, respectively.

In particular, in the new $\mathsf{V}_{\mathsf{month-all}}$ the cell (⟨'Nov-08'⟩,⟨all⟩) is deleted because the eliminated cell (⟨'Nov-08'⟩,⟨'chardonnay'⟩) in $\mathsf{V}_{\mathsf{month-grape}}$ was the only one aggregating to it.

Finally, the total measure of the new cuboid $\mathsf{V}_{\mathsf{top}}$ changes its value from 229.400 to 215.000, because the eliminated cells in $\mathsf{V}_{\mathsf{bottom}}$ aggregate transitively to it.

In Figure 4.9 the measures with changes respect to the original values are highlighted in rose.

Notice that it is impossible to apply DICE($\mathsf{V}_{\mathsf{month-grape}}$, year = 2007) because the descriptor year does not belong to $\mathcal{V}_{\mathsf{V}_{\mathsf{month-grape}}}$. In order to apply this condition, a ROLL-UP to Year level in TimeDim dimension must be applied previously. □



TimeDim	BlockDim	Measures
All	All	Sum
all	all	harvest
all	all	229400

V_{top}

TimeDim	BlockDim	Measures
All	Grape	Sum
all	gName	harvest
all	chardonnay	68100
all	kerner	68300
all	pinot blanc	45400
all	pinot noir	47600

$V_{all\text{-}grape}$

TimeDim	BlockDim	Measures
Year	All	Sum
year	all	harvest
2007	all	44600
2008	all	46000
2009	all	45800
2010	all	46400
2011	all	46600

$V_{year\text{-}all}$

TimeDim	BlockDim	Measures
All	Block	Sum
all	idBlock, bGeom	harvest
all	35001, g35001	22200
all	35002, g35002	22900
all	35003, g35003	23200
all	35004, g35004	23100
all	35005, g35005	22300
all	35006, g35006	22700
all	35077, g35001	15000
all	35078, g35001	14900
all	35079, g35001	15500
all	35084, g35004	16000
all	35085, g35005	15900
all	35086, g35006	15700

$V_{all\text{-}block}$

TimeDim	BlockDim	Measures
Year	Grape	Sum
year	gName	harvest
2007	chardonnay	22600
2007	kerner	22000
2008	chardonnay	22700
2008	kerner	23300
2009	chardonnay	22800
2009	kerner	23000
2010	pinot blanc	22600
2010	pinot noir	23800
2011	pinot blanc	22800
2011	pinot noir	23800

$V_{year\text{-}grape}$

TimeDim	BlockDim	Measures
Month	All	Sum
month	all	harvest
Aug-07	all	14700
Sep-07	all	21900
Oct-07	all	8000
Aug-08	all	23300
Sep-08	all	15600
Nov-08	all	7100
Aug-09	all	23000
Sep-09	all	15200
Oct-07	all	7600
Aug-10	all	14900
Sep-10	all	15800
Oct-10	all	15700
Aug-11	all	22800
Sep-11	all	15900
Nov-11	all	7900

$V_{month\text{-}all}$

TimeDim	BlockDim	Measures
Year	Block	Sum
year	idBlock, bGeom	harvest
2007	35001, g35001	7200
2007	35002, g35002	7500
2007	35003, g35003	7300
2007	35004, g35004	7200
2007	35005, g35005	7400
2007	35006, g35006	8000
2008	35001, g35001	7800
2008	35002, g35002	7600
2008	35003, g35003	7900
2008	35004, g35004	8100
2008	35005, g35005	7500
2008	35006, g35006	7100
2009	35001, g35001	7200
2009	35002, g35002	7800
2009	35003, g35003	8000
2009	35004, g35004	7800
2009	35005, g35005	7400
2009	35006, g35006	7600
2010	35077, g35001	7500
2010	35078, g35001	7400
2010	35079, g35001	7700
2010	35084, g35004	8100
2010	35085, g35005	7900
2010	35086, g35006	7800
2011	35077, g35001	7500
2011	35078, g35001	7500
2011	35079, g35001	7800
2011	35084, g35004	7900
2011	35085, g35005	8000
2011	35086, g35006	7900

$V_{year\text{-}block}$

TimeDim	BlockDim	Measures
Month	Grape	Sum
month	gName	harvest
Aug-07	kerner	14700
Sep-07	kerner	7300
Sep-07	chardonnay	14600
Oct-07	chardonnay	8000
Aug-08	kerner	23300
Sep-08	chardonnay	15600
Nov-08	chardonnay	7100
Aug-09	kerner	23000
Sep-09	chardonnay	15200
Oct-09	chardonnay	7600
Aug-10	pinot blanc	14900
Sep-10	pinot blanc	7700
Sep-10	pinot noir	8100
Oct-10	pinot noir	15700
Aug-11	pinot blanc	22800
Sep-11	pinot noir	15900
Nov-11	pinot noir	7900

$V_{month\text{-}grape}$

TimeDim	BlockDim	Measures
Month	Block	Sum
month	idBlock, bGeom	harvest
Aug-07	35001, g35001	7200
Aug-07	35002, g35002	7500
Sep-07	35003, g35003	7300
Sep-07	35004, g35004	7200
Sep-07	35005, g35005	7400
Oct-07	35006, g35006	8000
Aug-08	35001, g35001	7800
Aug-08	35002, g35002	7600
Aug-08	35003, g35003	7900
Sep-08	35004, g35004	8100
Sep-08	35005, g35005	7500
Nov-08	35006, g35006	7100
Aug-09	35001, g35001	7200
Aug-09	35002, g35002	7800
Aug-09	35003, g35003	8000
Sep-09	35004, g35004	7800
Sep-09	35005, g35005	7400
Oct-09	35006, g35006	7600
Aug-10	35077, g35001	7500
Aug-10	35078, g35001	7400
Sep-10	35079, g35001	7700
Sep-10	35084, g35004	8100
Oct-10	35085, g35005	7900
Oct-10	35086, g35006	7800
Aug-11	35077, g35001	7500
Aug-11	35078, g35001	7500
Aug-11	35079, g35001	7800
Sep-11	35084, g35004	7900
Sep-11	35085, g35005	8000
Nov-11	35086, g35006	7900

V_{bottom}

Figure 4.7: The cuboids of Vineyard-short instance.

TimeDim	BlockDim	Measures
Month	Grape	(Sum)
month	gName	harvest
⟨ Aug-07 ⟩	⟨ kerner ⟩	14700
⟨ Sep-07 ⟩	⟨ chardonnay ⟩	14600
⟨ Oct-07 ⟩	⟨ chardonnay ⟩	8000
⟨ Aug-08 ⟩	⟨ kerner ⟩	23300
⟨ Sep-08 ⟩	⟨ chardonnay ⟩	15600
⟨ Aug-09 ⟩	⟨ kerner ⟩	23000
⟨ Sep-09 ⟩	⟨ chardonnay ⟩	15200
⟨ Oct-09 ⟩	⟨ chardonnay ⟩	7600
⟨ Aug-10 ⟩	⟨ pinot blanc ⟩	14900
⟨ Sep-10 ⟩	⟨ pinot blanc ⟩	7700
⟨ Sep-10 ⟩	⟨ pinot noir ⟩	8100
⟨ Oct-10 ⟩	⟨ pinot noir ⟩	15700
⟨ Aug-11 ⟩	⟨ pinot blanc ⟩	22800
⟨ Sep-11 ⟩	⟨ pinot noir ⟩	15900
⟨ Nov-11 ⟩	⟨ pinot noir ⟩	7900

Figure 4.8: Dice($\mathsf{V}_{\mathsf{month-grape}}$, $\mathsf{harvest} \geq 7.500$)

4.2.2 Slice Operator

The SLICE operator is a function with signature SLICE : $\mathbf{C} \times (\mathbf{D} \cup \mathbf{M}) \rightarrow \mathbf{C}$, that reduces the dimensionality of a cube by removing one of its dimensions or measures. Following Agrawal et al. [4], the dimensions and the measures of a cube are interchangeable. Thus, the SLICE operator is applicable for eliminating a dimension when $|\mathcal{D}_{in}| > 1$ or a measure when $|\mathcal{M}_{in}| > 1$.

In the case of eliminating a dimension, the input dimension must contain just one value in its domain. Therefore, this operator applies a ROLL-UP to All in the input dimension prior to slicing it. Notice that if in the dimension to be eliminated there is just a single value, the application of the ROLL-UP operator would introduce no change.

Definition 13 (Slice)**.** Given a cube instance CI with schema $\langle \mathsf{CI}, \mathcal{D}_{in}, \mathcal{M}_{in} \rangle$, a cuboid $\mathsf{C}_{in} \in \mathsf{CI}$, and a dimension $\mathsf{D} \in \mathcal{D}_{in}$ such that $|\mathcal{D}_{in}| > 1$ or a measure $\mathsf{M} \in \mathcal{M}_{in}$ such that $|\mathcal{M}_{in}| > 1$, according to the input parameters:

(1) SLICE($\mathsf{C}_{in}, \mathsf{D}$) returns a new cuboid $\mathsf{C}_{out} \in \mathbf{C}$ obtained by removing the dimension D in C_{in} as follows:

(a) $c_i = (c_{i1}, \ldots, c_{ik-1}, c_{ik+1} \ldots, c_{in}, m_{i1}, \ldots, m_{is}) \in \mathsf{C}_{out}$ if $\exists c_j = (c_{j1}, \ldots, c_{ik-1}, all, c_{ik+1} \ldots, c_{jn}, m_{j1}, \ldots, m_{js}) \in$ ROLL-UP($\mathsf{C}_{in}, \mathsf{D}, \mathsf{All}$) and $c_{ip} =$

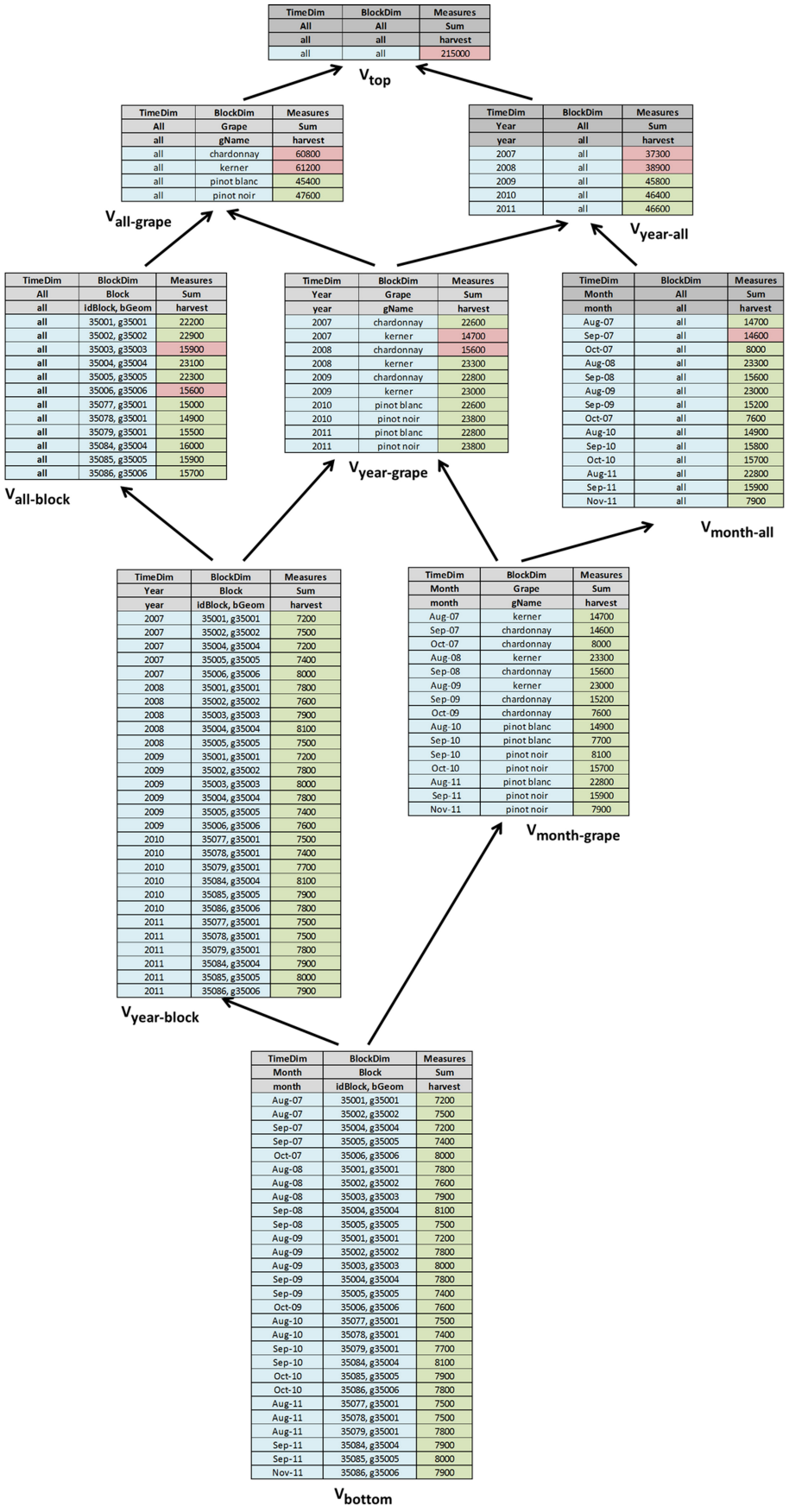


Figure 4.9: The cuboids of Vineyard-short after dicing.

$c_{jp} \forall p = 1 \ldots n, p \neq k$ and $m_{iq} = m_{jq} \forall q = 1 \ldots s$

(b) $\mathcal{V}_{\mathsf{C}_{out}} = \mathcal{V}_{\mathsf{C}_{in}} - \{\mathsf{l}_d\}$, where l_d is the level in the cuboid corresponding to the removed dimension D.

(2) SLICE$(\mathsf{C}_{in}, \mathsf{M})$ returns a new cuboid $\mathsf{C}_{out} \in \mathbf{C}$ obtained by removing the measure M in C_{in} as follows:

(a) $c_i = (c_{i1}, \ldots, c_{in}, m_{i1}, \ldots, m_{ik-1}, m_{ik+1}, \ldots, m_{is}) \in \mathsf{C}_{out}$ if $\exists c_j = (c_{j1}, \ldots, c_{jn}, m_{j1}, \ldots, m_{jk-1}, m_{jk}, m_{jk+1}, \ldots, m_{js}) \in \mathsf{C}_{in}$ and m_{jk} corresponds to M and $c_{ip} = c_{jp} \forall p = 1 \ldots n$ and $m_{iq} = m_{jq} \forall q = 1 \ldots s, q \neq k$

(b) $\mathcal{V}_{\mathsf{C}_{out}} = \mathcal{V}_{\mathsf{C}_{in}}$

Since SLICE is an IGO, C_{out} induces a new cube instance CI_{out} such that $\mathsf{C}_{out} \in \mathsf{CI}_{out}$. Algorithm 2 computes the instance CI_{out}. It first clones the lattice of cuboids CI, i.e., it creates a lattice of cuboids with the same schema and instance. Then, the deleted dimension in $\mathcal{D}$ or the deleted measure in $\mathcal{M}$ are removed from the schema, and the SLICE operator is applied to all the cuboids of CI_{out}. □

Algorithm 2 New cube instance for SLICE$(\mathsf{Cb}_{in}, \mathsf{S})$, with $\mathsf{Cb}_{in} \in \mathsf{CI}$ instance

CI_{out} *schema* $\leftarrow$ CI *schema*
if S is a dimension **then**
  $\mathcal{D}_{out} \leftarrow \mathcal{D}_{in} - \{S\}$
else
  $\mathcal{M}_{out} \leftarrow \mathcal{M}_{in} - \{S\}$
CI_{out} *instance* $\leftarrow$ CI *instance* (i.e., clone CI as C_{out})
for all cuboid $\mathsf{Cb}_j \in \mathsf{C}_{out}$ **do**
  SLICE$(\mathsf{Cb}_j, \mathsf{S})$

Remark 6. *After a applying the* SLICE *operation to a dimension, the number of cuboids in* CI_{out} *is N times lower than the number of cuboids in* CI_{in}*, where N is*

the number of levels in the lattice $\mathcal{L}$ of the dimension D. On the contrary, there is no change in the number of cuboids after applying SLICE *to a measure.*

Example 18 (Slicing a Dimension)**.** Consider the simplified version of Vineyard-short given in Example 17. SLICE($V_{year-grape}$, BlockDim), returns the cuboid shown in Figure 4.10. First, a ROLL-UP to All over the BlockDim dimension is applied (obtaining a unique value for the cuboid instance) and then the dimension is eliminated. This operation yields a new cube instance, built applying the Algorithm 2. Figure 4.11 shows this new cube instance. □

TimeDim	Measures
Year	(Sum)
year	harvest
⟨2007⟩	44600
⟨2008⟩	46000
⟨2009⟩	45800
⟨2010⟩	46400
⟨2011⟩	46600

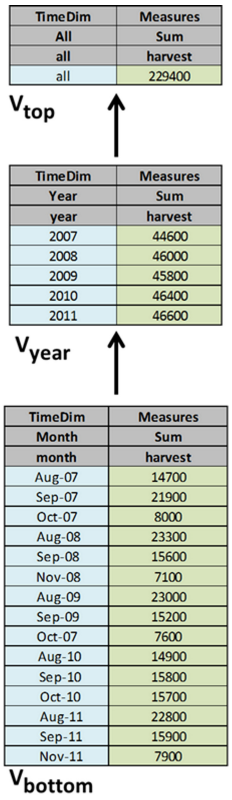
Figure 4.10: SLICE($V_{year-grape}$, BlockDim)

Example 19 (Slicing a Measure)**.** Consider the cuboid $D_{year-country-product} \in$ Drinks, shown in Figure 4.12. If we perform SLICE($D_{year-country-product}$, sales), the resulting cuboid is shown in Figure 4.13.

According to Algorithm 2, SLICE induces a new cube instance where the sliced measure has been removed from all its cuboids. □

4.2.3 Drill-across Operator

The traditional DRILL-ACROSS operator (based on the work of Kimball and Ross [27]) is a function with signature DRILL-ACROSS: $\mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}$ that basically performs a join of two cubes in order to give a unified view of their measures. In simple terms, given two cuboids that are defined over the same dimensions and contain the same instances (but with different measures), the Drill-across operation returns a single cuboid with the same dimensions and instances, containing the union of the measures of both cuboids. As a result, measures coming from different cuboids can be easily compared because they are displayed together.



TimeDim	Measures
All	Sum
all	harvest
all	229400

V_{top}

TimeDim	Measures
Year	Sum
year	harvest
2007	44600
2008	46000
2009	45800
2010	46400
2011	46600

V_{year}

TimeDim	Measures
Month	Sum
month	harvest
Aug-07	14700
Sep-07	21900
Oct-07	8000
Aug-08	23300
Sep-08	15600
Nov-08	7100
Aug-09	23000
Sep-09	15200
Oct-07	7600
Aug-10	14900
Sep-10	15800
Oct-10	15700
Aug-11	22800
Sep-11	15900
Nov-11	7900

V_{bottom}

Figure 4.11: The cuboids of Vineyard-short instance after slicing.

Definition 14 (Drill-across)**.** Given two cubes instances CI_1 and CI_2 with schemas $\langle \mathsf{CI}_1, \mathcal{D}, \mathcal{M}_1 \rangle$ and $\langle \mathsf{CI}_2, \mathcal{D}, \mathcal{M}_2 \rangle$, respectively, a cuboid $\mathsf{C}_{1_{in}} \in \mathsf{CI}_1$ and a cuboid $\mathsf{C}_{2_{in}} \in \mathsf{CI}_2$ such that $\mathcal{V}_{\mathsf{C}_{1_{in}}} = \mathcal{V}_{\mathsf{C}_{2_{in}}}$, DRILL-ACROSS$(\mathsf{C}_{1_{in}}, \mathsf{C}_{2_{in}})$ returns a new cuboid C_{out} with the dimensions of $\mathcal{D}$ and all the measures of $\mathsf{C}_{1_{in}}$ and $\mathsf{C}_{2_{in}}$, as follows:

(a) $\mathsf{c}_i = (c_{i1}, \ldots, c_{in}, \ldots, m_{i1}, \ldots, m_{ir}, m_{ir+1}, \ldots, m_{ir+s}) \in \mathsf{C}_{out}$ where $\mathsf{c}_p = (c_{p1}, \ldots, c_{pn}, \ldots, m_{p1}, \ldots, m_{pr}) \in \mathsf{C}_{1_{in}}$ and $\mathsf{c}_q = (c_{q1}, \ldots, c_{qn}, \ldots, m_{q1}, \ldots, m_{qs}) \in \mathsf{C}_{2_{in}}$ and $m_{iu} = m_{pu} \forall u, u = 1 \ldots r$ and $m_{ir+v} = m_{qv} \forall v, v = 1 \ldots s$

(b) $\mathcal{V}_{\mathsf{C}_{out}} = \mathcal{V}_{\mathsf{C}_{1_{in}}}$

Since DRILL-ACROSS is and IGO, C_{out} induces a new cube instance CI_{out} such that $\mathsf{C}_{out} \in \mathsf{CI}_{out}$. Algortihm 3 builds CI_{out}. It first clones the lattice of cuboids CI_1, i.e., creates a lattice of cuboids with the same schema and instance. Then,

TimeDim	ZoneDim	ProductDim	Measures	
Yearh	Country	Product	(Avg)	(Avg)
year	cyName	pDetail	consumption	sales
⟨2008⟩	⟨Italy⟩	⟨lager beer⟩	0.6	0.5
⟨2008⟩	⟨Italy⟩	⟨stout beer⟩	0.4	0.6
⟨2008⟩	⟨Italy⟩	⟨pale beer⟩	0.1	0.8
⟨2008⟩	⟨Italy⟩	⟨chardonnay wine⟩	2.1	1.6
⟨2008⟩	⟨Italy⟩	...	...	...
⟨2008⟩	⟨Italy⟩	⟨ron⟩	1.9	1.6
⟨2008⟩	⟨Belgium⟩	⟨lager beer⟩	1.2	0.9
⟨2008⟩	⟨Belgium⟩	⟨stout beer⟩	1.0	0.9
⟨2008⟩	⟨Belgium⟩	⟨pale beer⟩	0.6	0.7
⟨2008⟩	⟨Belgium⟩	⟨chardonnay wine⟩	0.7	0.6
⟨2008⟩	⟨Belgium⟩	⟨malbec wine⟩	0.1	0.2
⟨2008⟩	⟨Belgium⟩	⟨kerner wine⟩	0.6	0.7
⟨2008⟩	⟨Belgium⟩	⟨whiskey⟩	0.2	0.4
⟨2008⟩	⟨Belgium⟩	⟨vodska⟩	1.9	1.6
⟨2008⟩	⟨Belgium⟩	⟨ron⟩	0.6	0.5
⟨2008⟩	⟨Greece⟩	⟨lager beer⟩	1.5	1.6
⟨2008⟩	⟨Greece⟩	⟨stout beer⟩	1.2	1.4
⟨2008⟩	⟨Greece⟩	...	...	...
⟨2008⟩	⟨Greece⟩	⟨ron⟩	1.0	0.9
...	...	...	...	...
⟨2011⟩	⟨Italy⟩	⟨lager beer⟩	0.7	0.6
⟨2011⟩	⟨Italy⟩	⟨stout beer⟩	0.4	0.5
⟨2011⟩	⟨Italy⟩	⟨pale beer⟩	0.2	0.3
⟨2011⟩	⟨Italy⟩	⟨chardonnay wine⟩	1.9	1.6
⟨2011⟩	⟨Italy⟩	...	...	...
⟨2011⟩	⟨Belgium⟩	⟨lager beer⟩	1.1	0.9
⟨2011⟩	⟨Belgium⟩	⟨stout beer⟩	1.1	1.0
⟨2011⟩	⟨Belgium⟩	⟨pale beer⟩	0.6	0.6
⟨2011⟩	⟨Belgium⟩	⟨chardonnay wine⟩	0.9	0.9
⟨2011⟩	⟨Belgium⟩	⟨malbec wine⟩	0.3	0.6
⟨2011⟩	⟨Belgium⟩	⟨pinot noir wine⟩	0.5	0.6
⟨2011⟩	⟨Belgium⟩	⟨whiskey⟩	0.2	0.1
⟨2011⟩	⟨Belgium⟩	⟨cognac⟩	0.5	0.6
⟨2011⟩	⟨Belgium⟩	⟨vodska⟩	1.6	1.7
⟨2011⟩	⟨Belgium⟩	⟨ron⟩	1.0	0.9
⟨2011⟩	⟨Greece⟩	⟨lager beer⟩	1.3	1.2
...	...	...	...	...
⟨2011⟩	⟨Greece⟩	⟨stout beer⟩	1.2	0.9

Figure 4.12: Cuboid $\mathsf{D}_{\mathsf{year-country-product}}$ of Drinks

the algorithm modifies the set $\mathcal{M}$ in the cube schema, with the union of the sets of measures of both cuboids, and the DRILL-ACROSS operation is applied to each cuboid of CI_{out} and the cuboid of CI_2 with the same set levels (i.e., all the cuboids in CI_{out} are crossed with the corresponding cuboid in CI_2).

Example 20 (Drill-across operator)**.** Consider the cube instances Vineyard-short and Drinks of Examples 17 and 19, respectively.

We would like to put together information about wine sales and grape production bt year. To implement this, we need to peform a Drill-across operation

TimeDim	ZoneDim	ProductDim	Measures
Yearh	Country	Product	(Avg)
year	cyName	pDetail	consumption
⟨2008⟩	⟨Italy⟩	⟨lager beer⟩	0.6
⟨2008⟩	⟨Italy⟩	⟨stout beer⟩	0.4
⟨2008⟩	⟨Italy⟩	⟨pale beer⟩	0.1
⟨2008⟩	⟨Italy⟩	⟨chardonnay wine⟩	2.1
⟨2008⟩	⟨Italy⟩	...	...
⟨2008⟩	⟨Italy⟩	⟨ron⟩	1.9
⟨2008⟩	⟨Belgium⟩	⟨lager beer⟩	1.2
⟨2008⟩	⟨Belgium⟩	⟨stout beer⟩	1.0
⟨2008⟩	⟨Belgium⟩	⟨pale beer⟩	0.6
⟨2008⟩	⟨Belgium⟩	⟨chardonnay wine⟩	0.7
⟨2008⟩	⟨Belgium⟩	⟨malbec wine⟩	0.1
⟨2008⟩	⟨Belgium⟩	⟨kerner wine⟩	0.6
⟨2008⟩	⟨Belgium⟩	⟨whiskey⟩	0.2
⟨2008⟩	⟨Belgium⟩	⟨vodska⟩	1.9
⟨2008⟩	⟨Belgium⟩	⟨ron⟩	0.6
⟨2008⟩	⟨Greece⟩	⟨lager beer⟩	1.5
⟨2008⟩	⟨Greece⟩	⟨stout beer⟩	1.2
⟨2008⟩	⟨Greece⟩	...	...
⟨2008⟩	⟨Greece⟩	⟨ron⟩	1.0
⟨2008⟩	⟨Greece⟩	...	...
...	...	...	...
⟨2011⟩	⟨Italy⟩	⟨lager beer⟩	0.7
⟨2011⟩	⟨Italy⟩	⟨stout beer⟩	0.4
⟨2011⟩	⟨Italy⟩	⟨pale beer⟩	0.2
⟨2011⟩	⟨Italy⟩	⟨chardonnay wine⟩	1.9
⟨2011⟩	⟨Italy⟩	...	...
⟨2011⟩	⟨Belgium⟩	⟨lager beer⟩	1.1
⟨2011⟩	⟨Belgium⟩	⟨stout beer⟩	1.1
⟨2011⟩	⟨Belgium⟩	⟨pale beer⟩	0.6
⟨2011⟩	⟨Belgium⟩	⟨chardonnay wine⟩	0.9
⟨2011⟩	⟨Belgium⟩	⟨malbec wine⟩	0.3
⟨2011⟩	⟨Belgium⟩	⟨pinot noir⟩	0.5
⟨2011⟩	⟨Belgium⟩	⟨whiskey⟩	0.2
⟨2011⟩	⟨Belgium⟩	⟨cognac⟩	0.5
⟨2011⟩	⟨Belgium⟩	⟨vodska⟩	1.6
⟨2011⟩	⟨Belgium⟩	⟨ron⟩	1.0
⟨2011⟩	⟨Greece⟩	⟨lager beer⟩	1.3
...	...	...	...
⟨2011⟩	⟨Greece⟩	⟨stout beer⟩	1.2

Figure 4.13: SLICE($\mathsf{D}_{\mathsf{year-country-product}}$, sales)

between $\mathsf{V}_{\mathsf{year-grape}} \in$ Vineyard-short and $\mathsf{D}_{\mathsf{year-country-product}} \in$ Drinks. However, the traditional definition of DRILL-ACROSS does not allow this. Since the cube instances do not have the same schema, slicing and dicing must be applied before that. Only the TimeDim dimension is shared by both schemas. Thus, we must eliminate the dimensions ProductDim and ZoneDim in Drinks and the dimension BlockDim in Vineyard-short.

On the one hand, since the facts in Vineyard-short only correspond to data from

Algorithm 3 New cube instance for DRILL-ACROSS($\mathsf{Cb1}_{in}, \mathsf{Cb2}_{in}$), with $\mathsf{Cb1}_{in} \in \mathsf{CI}_1$ and $\mathsf{Cb2}_{in} \in \mathsf{CI}_2$ instance

CI_{out} *schema* $\leftarrow \mathsf{CI}_1$*schema*

$\mathcal{M}_{out} \leftarrow \mathcal{M}_{1_{in}} \bigcup \mathcal{M}_{2_{in}}$

CI_{out} *instance* $\leftarrow \mathsf{CI}_1$ *instance* (i.e., clone CI_1 as C_{out})

for all cuboid $\mathsf{Cb}_j \in \mathsf{C}_{out}$ **do**

DRILL-ACROSS($\mathsf{Cb}_j, \mathsf{Cb}_i$) where $\mathsf{Cb}_i \in \mathsf{Cb2}_{in}$ and $\mathcal{V}_{\mathsf{Cb}_i} = \mathcal{V}_{\mathsf{Cb}_j}$

'Belgium', we must only keep this country in Drinks, otherwise the aggregate measures will present non-coherent values. On the other hand, only the products in Drinks representing names of wines must be kept. Thus, previous to the slicing, we must select the particular shared values (by applying a dicing over the cuboids) to ensure the consistence of the resulting cuboid. The following steps prepare the two cuboids for the DRILL-ACROSS operator:

$\mathsf{Sliced\_V}_{\mathsf{year-grape}} = \text{SLICE}(\mathsf{V}_{\mathsf{year-grape}}, \mathsf{BlockDim})$

$\mathsf{Aux1} = \text{DICE}(\mathsf{D}_{\mathsf{year-country-product}}, \mathsf{cyName} = \text{'Belgium'} \wedge (\mathsf{pDetail} = \text{'malbec wine'} \vee \cdots \vee \mathsf{pDetail} = \text{'pinot noir wine'})$

$\mathsf{Aux2} = \text{SLICE}(\mathsf{Aux1}, \mathsf{ZoneDim})$

$\mathsf{Sliced\_D}_{\mathsf{year-country-product}} = \text{SLICE}(\mathsf{Aux2}, \mathsf{ProductDim})$

Finally, $\mathsf{Sliced\_V}_{\mathsf{year-grape}}$ and $\mathsf{Sliced\_D}_{\mathsf{year-country-product}}$ have the values related to grapes in Belgium per year, and we can apply DRILL-ACROSS($\mathsf{Sliced\_V}_{\mathsf{year-grape}}$, $\mathsf{Sliced\_D}_{\mathsf{year-country-product}}$), whose result is shown in Figure 4.14. The complete output instance is depicted in Figure 4.15. Notice that the total aggregation in harvest is 184800 (instead of 229400) because the harvest value corresponding to year 2007 has no match in the DRILL-ACROSS operation.

Intuitively one might think in crossing detailed information for grapes. However, because of the restriction of strict equality in the instances, we could only

combine the summarized information at the year level since the dimensions BlockDim and ProductDim need to be sliced. The solution we present in the next chapter overcomes this limitation. □

TimeDim	Measures		
Year	(Sum)	(Avg)	(Avg)
year	harvest	consumption	sales
⟨2008⟩	46000	0.46	0.5
⟨2009⟩	45800	0.91	0.9
⟨2010⟩	46400	0.7	0.63
⟨2011⟩	46600	0.57	0.7

Figure 4.14: DRILL-ACROSS(Sliced_$V_{year-grape}$, Sliced_$D_{year-country-product}$)

TimeDim	Measures		
All	Sum	Avg	Avg
all	harvest	consumption	sales
all	184800	0.66	0.68

VD_{top}

TimeDim	Measures		
Year	Sum	Avg	Avg
year	harvest	consumption	sales
2008	46000	0.46	0.5
2009	45800	0.91	0.9
2010	46400	0.7	0.63
2011	46600	0.57	0.7

VD_{year}

TimeDim	Measures		
Month	Sum	Avg	Avg
month	harvest	consumption	sales
Aug-08	23300	0.3	0.36
Sep-08	15600	0.55	0.57
Nov-08	7100	0.53	0.57
Aug-09	23000	0.64	0.64
Sep-09	15200	0.86	0.84
Oct-07	7600	1.23	1.22
Aug-10	14900	0.63	0.61
Sep-10	15800	0.72	0.65
Oct-10	15700	0.75	0.63
Aug-11	22800	0.47	0.62
Sep-11	15900	0.59	0.74
Nov-11	7900	0.65	0.74

VD_{bottom}

Figure 4.15: The cuboids of Vineyard-short after drillind-across.

4.3 Summary

In this chapter we presented an OLAP algebra supporting our conceptual data model. This algebra ignores implementation aspects and only manipulates data cubes, regardless the kinds of data contained in such cubes. We will use this feature of the algebra in the remainder of this thesis. We classified the algebra operations into two groups: instance preserving and instance generating operations. Operators in the first group (e.g., Roll-up and Drill-down) preserve the cube instance of the cuboids over which they are applied, while operations in the second group (e.g., Dice, Slice and Drill-across) generate a new cube instance. Finally, we introduced an example to show the limitations of the traditional Drill-across operator, which we will study in the next chapter, where we will also propose a solution to overcome those limitations.

Chapter 5

Extending the Drill-across Operator

As we mentioned in Chapter 4, the traditional DRILL-ACROSS operator based on the definition given by Kimball and Ross [27] has strong limitations due to the restriction about the input cuboids, namely that both cuboids must have the same schema dimensions and instances. This prevents the application of the operator in many real-world scenarios. For example, if the dimensions are identical but the instance members have different representations, the drill-across operation could not be applied, although the different representations are conceptually equivalent.

In order to relax the above requirements, Abelló et al. [2] introduced a set of rules based on semantic relationships. Building on this idea, we extend the DRILL-ACROSS operation to allow identifying equivalent dimensions in both input data cubes, through two kinds of semantic relationships: dimension-dimension derivation and dimension-dimension association as follows:

- Dimension-dimension derivation: If two dimensions come from a common concept although their structures differ, they can be mapped to each other. For example, if two dimensions contain the same instances but different names, these names can be matched through a mapping function. Another case occurs when two dimensions that represent the same concept are defined at different levels of granularity or detail. This would be the case of two spatial dimensions with granularities 'point', and 'polygon', respectively. A solution consists in rolling the former up to the closest common level. For

example, 'point' could be rolled-up to 'polygon'. If necessary, even a new level could be introduced.

- Dimension-dimension association: Corresponds to the case in which two cubes have different dimension lattices, but a set of dimension levels from one of them could be considered equivalent to a set of dimension levels from the other. For example, in one cube we define the dimensions LatDim and LongDim, containing the levels Latitude and Longitude, respectively, and in another cube the dimension GeomDim containing the level Point. A mapping function can deal with the problem of different representation by identifying the set {LatDim.Latitude, LongDim.Longitude} as equivalent to the set {GeomDim.Point}, together with the mapping of the corresponding members. For example the member $\langle\langle 30^o\rangle, \langle 20^o\rangle\rangle$ in the first set with the member $\langle(1030, 5020)\rangle$ in the second one.

In this chapter, we show how to apply the DRILL-ACROSS operation between cuboids belonging to cubes that do not share identical dimensions and instances, introducing the concept of *semantic mapping between cubes*.

5.1 Semantic Mapping between Cubes

In order to perform a DRILL-ACROSS operation between two cubes that do not share dimensions or instances, we need to define a semantic mapping between them. This mapping aims at solving the differences between dimension names, level names, member representations or even structure of dimension lattices.

The first step to semantically relate two cubes consists in choosing each set of dimension levels from one cube that are semantically equivalent to a set of levels from the other one, applying the notions of dimension-dimension derivation and dimension-dimension association. For this, we must identify all possible equivalent sets of levels, with the restriction of that two levels from the same dimension must not coexist in the same set. In the second step, we define a mapping of level members for each pair of semantically equivalent sets of levels.

For example, the set {LatDim.Latitude, LongDim.Longitude} in a cube C1 may be identified as equivalent to the set {GeoDim.Point} in another cube C2. Then, a correspondence among the members from the levels of the two sets must be built, based on the level descriptors. In this case, the composed member $\langle\langle 55^o\rangle, \langle 43^o\rangle\rangle$ of {LatDim.Latitude, LongDim.Longitude} semantically indicates the same that the member $\langle(6105, 4773)\rangle$ of {GeoDim.Point}, and viceversa.

When a set s of dimension levels is identified as candidate for a semantic mapping, the elements of the cartesian product between the level members of the levels of s are consider the members in s. Formally:

Definition 15 (Members of a Set of Levels)**.** Given $\mathsf{s} = \{\mathsf{l}_1, \ldots, \mathsf{l}_k\}$ a set of k dimension levels, we denote $\mathcal{T}_\mathsf{s}$ the *members of* s, which are given by the cartesian product between the members of each level, i.e., $\mathcal{T}_\mathsf{s} = \mathcal{T}_{\mathsf{l}_1} \times \mathcal{T}_{\mathsf{l}_2} \ldots \times \mathcal{T}_{\mathsf{l}_k}$, with members $\mathcal{T}_{\mathsf{l}_i}, \forall i = 1 \ldots k$ □

Example 21 (Members of a Set of Levels)**.** Consider the set of dimension levels s={LatDim.Latitude, LongDim.Longitude}. If $\langle 55^o\rangle \in \mathcal{T}_{LatDim.Latitude}$ and $\langle 43^o\rangle \in \mathcal{T}_{LongDim.Longitude}$, then the tuple $\langle\langle 55^o\rangle, \langle 43^o\rangle\rangle \in \mathcal{T}_s$. □

Before we define the extended version of the DRILL-ACROSS operator, we need to formalize the notion of semantic mapping.

Definition 16 (Semantic Mapping of Cubes)**.** Given two cube instances C_1 and C_2 with schemas $\langle \mathsf{C}_1, \mathcal{D}_1, \mathcal{M}_1\rangle$ and $\langle \mathsf{C}_2, \mathcal{D}_2, \mathcal{M}_2\rangle$, respectively, with $N_1 = |\mathcal{D}_1|$ and $N_2 = |\mathcal{D}_2|$, a semantic mapping between them, denoted $\mathsf{SM}(\mathsf{C}_1, \mathsf{C}_2)$ is the tuple $\langle \mathcal{P}, \mathcal{S}\rangle$, such that:

(a) $\mathcal{P}$ is a set of pairs of sets of levels, $\mathcal{P} = \{(\mathsf{s}_1, \mathsf{s}_2) | \mathsf{s}_1 \in 2^{\mathcal{L}_{11} \cup \mathcal{L}_{12} \cup \cdots \cup \mathcal{L}_{1N_1}} \wedge \mathsf{s}_2 \in 2^{\mathcal{L}_{21} \cup \mathcal{L}_{22} \cup \cdots \cup \mathcal{L}_{2N_2}}\}$, where $\mathcal{L}_{1i}$ is the set of levels of dimension $\mathsf{D}_i \in \mathcal{D}_1$, $\forall i, i = 1 \ldots N_1$ and $\mathcal{L}_{2j}$ is the set of levels of dimension $\mathsf{D}_j \in \mathcal{D}_2$, $\forall j, j = 1 \ldots N_2$, with the conditions that $\mathsf{s}_1 \neq \phi$, $\mathsf{s}_2 \neq \phi$ and $\forall \mathsf{a} \in \mathsf{s}_\mathsf{i}, \forall \mathsf{b} \in \mathsf{s}_\mathsf{i}, \mathsf{a} \in \mathcal{L}_{ip} \wedge \mathsf{b} \in \mathcal{L}_{iq} \Rightarrow \mathsf{p} \neq \mathsf{q}, \forall i, i = 1 \ldots 2$ (i.e., s_1 and s_2 are sets of dimension levels of C_1

and C_2, respectively, but neither of them can contain more than one level from the same dimension).

(b) $\mathcal{S}$ is the set containing all the relations SMAP defined as follows. There is one relation $\text{SMAP}_{\mathsf{s}_1}^{\mathsf{s}_2}$ for each pair $(\mathsf{s}_1, \mathsf{s}_2) \in \mathcal{P}$ such that: $\text{SMAP}_{\mathsf{s}_1}^{\mathsf{s}_2} \subseteq \mathcal{T}_{\mathsf{s}_1} \times \mathcal{T}_{\mathsf{s}_2}$ and $\forall \mathsf{a}, \mathsf{b} \in \mathcal{T}_{\mathsf{s}_1}, \forall \mathsf{c}, \mathsf{d} \in \mathcal{T}_{\mathsf{s}_2} : ((\mathsf{a}, \mathsf{c}) \in \text{SMAP}_{\mathsf{s}_1}^{\mathsf{s}_2} \wedge (\mathsf{b}, \mathsf{c}) \in \text{SMAP}_{\mathsf{s}_1}^{\mathsf{s}_2} \Rightarrow \mathsf{a} = \mathsf{b}) \wedge ((\mathsf{a}, \mathsf{c}) \in \text{SMAP}_{\mathsf{s}_1}^{\mathsf{s}_2} \wedge (\mathsf{a}, \mathsf{d}) \in \text{SMAP}_{\mathsf{s}_1}^{\mathsf{s}_2} \Rightarrow \mathsf{c} = \mathsf{d})$.

□

Remark 7. *Definition 16 indicates that if a semantic equivalence between two members exists, this must be one-to-one. However, members without correspondence may exist.* □

Remark 8. *The mapping relation between sets of levels is defined based on their level descriptors, and it can be done by intension (with an expression through the level descriptors) or by extension (exhaustively element by element).* □

We also remark that the concept of semantic equivalence between dimension levels and the correspondence between level members must be based on the point of view of the user. Examples 22 and 23 illustrate the above.

Example 22 (Dimension-Dimension Derivation)**.** Consider the cube instances Vineyard and Drinks given in Sections 3.2 and 4.2, abbreviated in the sequel as V and D, respectively. The analyst has identified the following semantic mapping between these two cubes, by applying dimension-dimension derivation relationships:

$\mathsf{SM}(\mathsf{V}, \mathsf{D}) = \langle \mathcal{P}, \mathcal{S} \rangle$, where

$\mathcal{P}$ = { ({V.TimeDim.Month}, {D.TimeDim.Month}), ({V.TimeDim.Year}, {D.TimeDim.Year}), ({V.BlockDim.Grape}, {D.ProdDim.Product}) }

$\mathcal{S}$ = { $\text{SMAP}^{\{\text{D.TimeDim.Month}\}}_{\{\text{V.TimeDim.Month}\}}$, $\text{SMAP}^{\{\text{D.TimeDim.Year}\}}_{\{\text{V.TimeDim.Year}\}}$, $\text{SMAP}^{\{\text{D.ProdDim.Product}\}}_{\{\text{V.BlockDim.Grape}\}}$ }

The member mappings are given by intension as:

$\text{SMAP}^{\{\text{D.TimeDim.Month}\}}_{\{\text{V.TimeDim.Month}\}}\{(\langle \text{month1} \rangle, \langle \text{month2} \rangle) | \text{month1=month2}\}$

$\text{SMAP}^{\{\text{D.TimeDim.Year}\}}_{\{\text{V.TimeDim.Year}\}} = \{(\langle \text{year1} \rangle, \langle \text{year2} \rangle) | \text{year1=year2}\}$

$\text{SMAP}^{\{\text{D.ProdDim.Product}\}}_{\{\text{V.BlockDim.Grape}\}} = \{(\langle \text{gName} \rangle, \langle \text{pDetail} \rangle) \mid \text{CONTAINS}(\text{pDetail}, \text{gName}) = \text{true}\}$

Another alternative for the last mapping may be the extension form, as follows:

$\text{SMAP}^{\{\text{D.ProdDim.Product}\}}_{\{\text{V.BlockDim.Grape}\}} = \{(\langle \text{'malbec'} \rangle, \langle \text{'malbec wine'} \rangle), \ldots, (\langle \text{'pinot blanc'} \rangle,$
$\langle \text{'pinot blanc wine'} \rangle)\}$

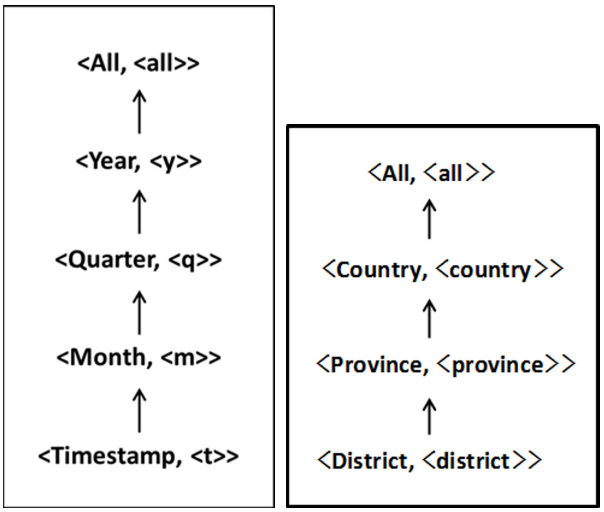
No level of dimension ZoneDim of Drinks is present in the mapping, because they have no conceptual equivalence with any level of Vineyard. Also note that, although the levels Grape and Product have been identified as conceptually equivalent, the levels GrapeType and Class (the levels to which the former roll-up, respectively) are not considered as equivalent and therefore they are not present in $\mathcal{P}$. This aspect will be treated in detail in Section 5.3. Thus, the member $\langle$'malbec'$\rangle$ of Grape is considered equivalent to the member $\langle$'malbec wine'$\rangle$ of Product. However, the member $\langle$'stout beer'$\rangle$ of Product has no equivalence in Grape (see Definition 18 below). □

Example 23 (Dimension-Dimension Association)**.** Consider the cubes C_1 and C_2, with schemas $\langle \mathsf{C}_1, \{\text{TemporalDim}, \text{GeoDim}\}, \{\text{sales}\}\rangle$ and $\langle \mathsf{C}_2, \{\text{DateDim}, \text{HourDim}, \text{SpaceDim}\}, \{\text{cost}\}\rangle$, respectively, whose lattices are shown in Figures 5.1 and 5.2.

In this case we identify that the dimensions GeoDim and SpaceDim represent the same concept with different name and different granularity in the lattice levels. Then, a dimension-dimension derivation relationship between the levels District and State, and between the two levels Country may be defined.

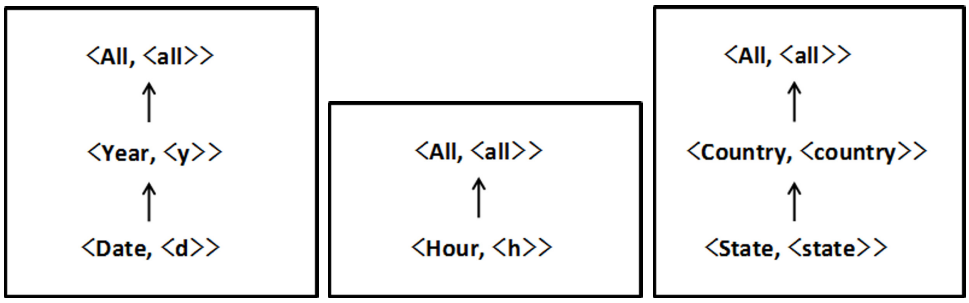
With respect to the temporal aspect, although C_1 has only one temporal dimension and C_2 has two, the combination of Date and Hour has the same meaning as Timestamp. According to this, and applying dimension-dimension association, the set of levels {DateDim.Date, TimeDim.Hour} is equivalent to the set {TemporalDim.Timestamp}.

We can thus define the semantic mapping:



(a) TemporalDim lattice (b) GeoDim lattice

Figure 5.1: Dimension lattices of C_1 (Example 23).



(a) DateDim lattice (b) HourDim lattice (c) SpaceDim Dimension

Figure 5.2: Dimension lattices of C_2 (Example 23).

$\mathsf{SM}(\mathsf{C}_1, \mathsf{C}_2) = \langle \mathcal{P}, \mathcal{S} \rangle$, where

$\mathcal{P} = \{(\{\mathsf{TemporalDim.Timestamp}\}, \{\mathsf{DateDim.Date}, \mathsf{HourDim.Hour}\}),$

$(\{\mathsf{TemporalDim.Year}\}, \{\mathsf{DateDim.Year}\}), (\{\mathsf{GeoDim.District}\}, \{\mathsf{SpaceDim.Sate}\}),$

$(\{\mathsf{GeoDim.Country}\}, \{\mathsf{SpaceDim.Country}\})\}$

and

$\mathcal{S} = \{\textsc{Smap}^{\{\mathsf{DateDim.Date},\mathsf{HourDim.Hour}\}}_{\mathsf{TemporalDim.Timestamp}\}}, \textsc{Smap}^{\{\mathsf{DateDim.Year}\}}_{\{\mathsf{TemporalDim.Year}\}}, \textsc{Smap}^{\{\mathsf{SpaceDim.Sate}\}}_{\{\mathsf{GeoDim.District}\}},$

$\textsc{Smap}^{\{\mathsf{SpaceDim.Country}\}}_{\{\mathsf{GeoDim.Country}\}}\}$

To map the members, suppose that all the domains are $\mathbb{S}$, and the format for the members of Date is 'mmm,dd-yyyy', for the members of Hour is 'hh:mm' and for the members of Timestamp is 'dd-mmm-yyyy,hh:mm'. Then, the four member mappings of $\mathcal{S}$ may be expressed by intension as follows:

$\text{SMAP}^{\{\text{SpaceDim.Sate}\}}_{\{\text{GeoDim.District}\}} = \{(\langle \text{district} \rangle, \langle \text{state} \rangle) \mid \text{district=state}\}$

$\text{SMAP}^{\{\text{SpaceDim.Country}\}}_{\{\text{GeoDim.Country}\}} = \{(\langle \text{country1} \rangle, \langle \text{country2} \rangle) \mid \text{country1=country2}\}$

$\text{SMAP}^{\{\text{DateDim.Year}\}}_{\{\text{TemporalDim.Year}\}} = \{(\langle \text{year1} \rangle, \langle \text{year2} \rangle) \mid \text{year1=year2}\}$

$\text{SMAP}^{\{\text{DateDim.Date,HourDim.Hour}\}}_{\{\text{TemporalDim.Timestamp}\}} = \{(\langle \langle \text{d} \rangle, \langle \text{h} \rangle \rangle, \langle \text{t} \rangle) \mid \text{SUB}(\text{d},1,3) = \text{SUB}(\text{t},4,3) \wedge \text{SUB}(\text{d},5,2) = \text{SUB}(\text{t},1,2) \wedge \text{SUB}(\text{d},8,4) = \text{SUB}(\text{t},8,4) \wedge \text{SUB}(\text{h},1,5) = \text{SUB}(\text{t},13,5)\}$

where SUB(str,i,s) returns the substring of the string str composed by the s characters following the position i in the string.

Thus, the member $\langle$'05-Feb-2010,15:30'$\rangle$ of {TemporalDim.Timestamp} is equivalent to the composed member $\langle\langle$'Feb,05-2010'$\rangle, \langle$'15:30'$\rangle\rangle$ of {DateDim.Date, HourDim.Hour}. □

5.2 Semantic Compatibility of Cuboids

The semantic mapping between two cubes allows us to introduce the concept of semantically compatible cuboids. Intuitively two cuboids $\text{Cb}_1 \in \text{C1}$ and $\text{Cb}_2 \in \text{C2}$ are semantically compatible if every combination of their levels is included in $\mathcal{P} \in \text{SM}(\text{C1}, \text{C2})$. Formally,

Definition 17 (Semantic Compatibility of Cuboids)**.** Two cuboids Cb_1 and Cb_2, belonging to the cube instances C_1 and C_2, respectively, are *semantically compatible*, denoted $\text{Cb}_1 \simeq \text{Cb}_2$, if $\forall \text{l}_\text{i} \in \mathcal{V}_{\text{Cb}_1} \wedge \forall \text{l}_\text{j} \in \mathcal{V}_{\text{Cb}_2}, \exists(\text{s}_1, \text{s}_2) \mid \text{l}_\text{i} \in \text{s}_1 \wedge \text{l}_\text{j} \in \text{s}_2 \Rightarrow \text{s}_1 \subseteq \mathcal{V}_{\text{Cb}_1} \wedge \text{s}_2 \subseteq \mathcal{V}_{\text{Cb}_2}$ That is, there is a semantic mapping for all levels in the two cubouids. □

Example 24 (Semantic Compatibility of Cuboids)**.** Consider the semantic mapping SM(Vineyard,Drinks) given in Example 22. No semantic mapping can be defined between any pair of cuboids in the cube instances of Vineyard and Drinks,

because of the Country and Continent levels in the ZoneDim dimension in Drinks, which have no equivalent in the other cube. In order to find semantically compatible cuboids between the two cube instances, we must drop the dimension ZoneDim, which can be done through a SLICE operation, as follows.

$\mathsf{Drinks}_{\mathsf{month-product}} = \textsc{Slice}(\mathsf{Drinks}_{\mathsf{month-country-product}}, \mathsf{ZoneDim})$

$\mathsf{Drinks}_{\mathsf{year-product}} = \textsc{Slice}(\mathsf{Drinks}_{\mathsf{year-country-product}}, \mathsf{ZoneDim}).$

Thus the following cuboids are semantically compatible:

$\mathsf{Vineyard}_{\mathsf{month-grape}} \simeq \mathsf{Drinks}_{\mathsf{month-product}}$

$\mathsf{Vineyard}_{\mathsf{year-grape}} \simeq \mathsf{Drinks}_{\mathsf{year-product}}$ □

When two cuboids are semantically compatible, we must check if their coordinates can be considered equivalent, and if so, perform the mapping.

Definition 18 (Semantic Equivalence of Coordinates)**.** Let be $(\mathsf{s}_{11}, \mathsf{s}_{12}), \ldots, (\mathsf{s}_{k1}, \mathsf{s}_{k2})$ the pairs of $\mathsf{SM}(\mathsf{C1}, \mathsf{C2})$ that make $\mathsf{Cb}_1 \simeq \mathsf{Cb}_2$, with $\mathsf{Cb}_1 \in \mathsf{C1}, \mathsf{Cb}_2 \in \mathsf{C2}$. The coordinate $c_1 = (c_{11}, c_{12}, \ldots, c_{1N}) \in \mathsf{Cb}_1$, is semantically equivalent to $c_2 = (c_{21}, c_{22}, \ldots, c_{2N}) \in \mathsf{Cb}_2$, denoted $c_1 \cong c_2$, iff $\forall\ i = 1 \ldots\ k\ \exists\ \textsc{Smap}_{\mathsf{s_{i1}}}^{\mathsf{s_{i2}}}$ and $(\pi_{\mathsf{s_{i1}}}(c_1), \pi_{\mathsf{s_{i2}}}(c_2)) \in \textsc{Smap}_{\mathsf{s_{i1}}}^{\mathsf{s_{i2}}}$.

We denote $\pi_{\mathsf{s_j}}(c_{i1}, c_{i2}, \ldots, c_{iN})$ the sub-tuple $(c_{ip}, \ldots, c_{iq})$ such that its components are members of the levels in the set $\mathsf{s_j}$. □

Example 25 (Semantic Equivalence of Coordinates)**.** Consider now the cubes C_1 and C_2 of Example 23, and the associated cuboids $\mathsf{C}_{\mathsf{district\text{-}timestamp}} \simeq \mathsf{C}_{\mathsf{state\text{-}date\text{-}hour}}$. Figure 5.3 shows one of the possible equivalence between cells. Other semantic equivalent cuboids are $\mathsf{C}_{\mathsf{country\text{-}timestamp}} \simeq \mathsf{C}_{\mathsf{country\text{-}date\text{-}hour}}$, and $\mathsf{C}_{\mathsf{all\text{-}timestamp}} \simeq \mathsf{C}_{\mathsf{all\text{-}date\text{-}hour}}$.

Note that the cuboids $\mathsf{Cb1}_{\mathsf{province\text{-}timestamp}}$ and $\mathsf{Cb2}_{\mathsf{state\text{-}date\text{-}hour}}$ are not equivalent and thus can not be used in a DRILL-ACROSS. In fact, Province and State have different granularity, and a member from Province may be related to more than one member from State. Therefore, a ROLL-UP must be applied before performing a DRILL-ACROSS. □

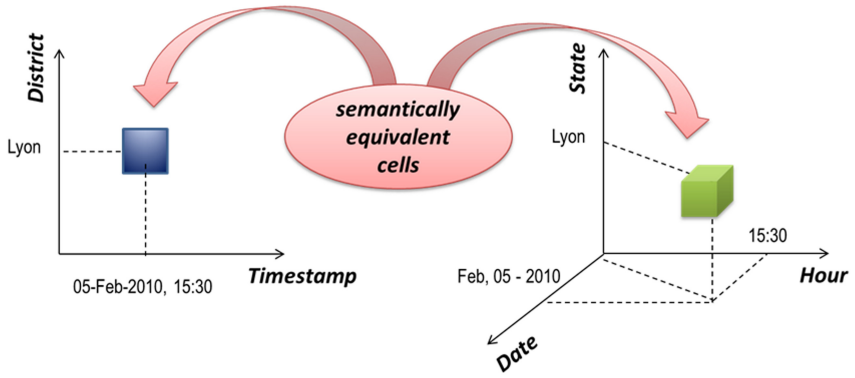


Figure 5.3: Two equivalent cuboids and their matching cells.

Example 26 (Semantic Equivalence of Coordinates)**.** We show now a mapping of the coordinates of the cuboids of Example 24, $\mathsf{Vineyard}_{\mathsf{year-grape}} \simeq \mathsf{Drinks}_{\mathsf{year-product}}$. In $\text{SMAP}^{\mathsf{Year}}_{\mathsf{Year}}$ and $\text{SMAP}^{\mathsf{Product}}_{\mathsf{Grape}}$ it holds: $\langle \pi_{\mathsf{Year}}(\langle 2008, \text{'malbec'} \rangle), \pi_{\mathsf{Year}}(\langle 2008, \text{'malbec wine'} \rangle) \rangle = \langle 2008, 2008 \rangle \in \text{SMAP}^{\mathsf{Year}}_{\mathsf{Year}}$; and $\langle \pi_{\mathsf{Grape}}(\langle 2008, \text{'malbec'} \rangle), \pi_{\mathsf{Product}}(\langle 2008, \text{'malbec wine'} \rangle) \rangle = \langle \text{'malbec'}, \text{'malbec wine'} \rangle \in \text{SMAP}^{\mathsf{Product}}_{\mathsf{Grape}}$. Thus, $\langle 2008, \text{'malbec'} \rangle \cong \langle 2008, \text{'malbec wine'} \rangle$.

On the contrary, $\langle 2008, \text{'malbec'} \rangle$ and $\langle 2008, \text{'stout beer'} \rangle$ are not semantically equivalent, because although $\langle \pi_{\mathsf{Year}}(\langle 2008, \text{'malbec'} \rangle), \pi_{\mathsf{Year}}(\langle 2008, \text{'malbec wine'} \rangle) \rangle \in \text{SMAP}^{\mathsf{Year}}_{\mathsf{Year}}$, there is no $(\mathsf{s1}, \mathsf{s2}) \in \mathcal{S}$ such that $\langle \pi_{\mathsf{s1}}(\langle 2008, \text{'malbec'} \rangle), \pi_{\mathsf{s2}}(\langle 2008, \text{'stout beer'} \rangle) \rangle = \langle \text{'malbec'}, \text{'stout beer'} \rangle \in \text{SMAP}^{\mathsf{s2}}_{\mathsf{s1}}$ in SM(Vineyard,Drinks). □

In many scenarios, two dimensions of different cubes can represent the same concept but with different lattice structure, thus only a few levels can be mapped. In these cases we can modify the dimension lattices by inserting a new level. For instance, the cube schema and its instance must be modified using the dimension update operators defined in Hurtado et al. [23]. For example, given two cubes C1 and C2 that contain two Time dimensions with lattices Month → Quarter → Year → All and Bimester → Year → All, respectively, instead of mapping only the levels Year and All, we may want to insert the Bimester level between Month and Quarter in the Time dimension lattice of C1, and then define the semantic mapping by including this finest commnon level.

The insertion of a new level B between two levels A and C of a dimension, not only involves modifying the dimension lattice, but also the definition of the roll-up functions between members, i.e., $\text{RUP}^{\mathsf{B}}_{\mathsf{A}}$ and $\text{RUP}^{\mathsf{C}}_{\mathsf{B}}$ functions. Example 27 shows in detail this process.

Example 27 (Level Insertion)**.** Consider the cube Vineyard from Example 11. In order to insert the level Quarter between Month and Year in DateDim, we modify the schema with $\langle \mathsf{TimeDim}, \mathcal{L}_{new}, \rightarrow \rangle$ where $\mathcal{L}_{new} = \mathcal{L}_{DateDim} \cup \{\langle \mathsf{Quarter}, \langle \mathsf{quarter} \rangle \rangle\}$ and define the new following RUP functions:
$\text{RUP}^{\mathsf{Quarter}}_{\mathsf{Month}} = \{(\langle \mathsf{m} \rangle, \langle \mathsf{q} \rangle) | ((\text{PRE}(\mathsf{m}) = \text{'Jan'} \vee \text{PRE}(\mathsf{m}) = \text{'Feb'} \vee \text{PRE}(\mathsf{m}) = \text{'Mar'} \vee \text{PRE}(\mathsf{m}) = \text{'Apr'}) \wedge \text{PRE}(\mathsf{q}) = \text{'Q1'}) \vee \ldots \vee ((\text{PRE}(\mathsf{m}) = \text{'Sep'}, \ldots, \text{PRE}(\mathsf{m}) = \text{'Dec'}) \wedge \text{PRE}(\mathsf{q}) = \text{'Q4'})\}$

where PRE is a function that returns the substring previous to the '-' in the input string.
$\text{RUP}^{\mathsf{Year}}_{\mathsf{Quarter}} = \{(\langle \mathsf{q} \rangle, \langle \mathsf{y} \rangle) \mid \text{POST}(\mathsf{q}) = \mathsf{y}\}$ where POST is a function that returns the substring posterior to the '-' in the input string. □

5.3 Extended Drill-Across Operation

Based on Definitions 17 and 18, we extend the DRILL-ACROSS operation, where the input of the operation are two semantically equivalent cuboids. The output is a cuboid whose dimension set is, by convention, the dimension set of the first cuboid. After performing the DRILL-ACROSS, the output cuboid becomes the bottom cuboid of the induced cube instance, and thus each current level in the first input cuboid becomes the new bottom level of its corresponding dimension lattice.

Definition 19 (Extended Drill-across)**.** Let CI_1 and CI_2 be two cube instances with schemas $\langle \mathsf{CI}_1, \mathcal{D}_1, \mathcal{M}_1 \rangle$ and $\langle \mathsf{CI}_2, \mathcal{D}_2, \mathcal{M}_2 \rangle$, respectively; there are also two cuboids $\mathsf{C1}_{\mathsf{in}} \in \mathsf{CI}_1$ and $\mathsf{C2}_{\mathsf{in}} \in \mathsf{CI}_2$ such that $\mathsf{C1}_{\mathsf{in}} \simeq \mathsf{C2}_{\mathsf{in}}$. Finally, $|\mathcal{D}_1| = D_1, |\mathcal{D}_2| = D_2, |\mathcal{M}_1| = M_1, |\mathcal{M}_2| = M_2$ hold. Then, DRILL-ACROSS($\mathsf{C1}_{\mathsf{in}}, \mathsf{C2}_{\mathsf{in}}$) returns a new cuboid $\mathsf{C}_{\mathsf{out}}$ as follows:

(a) $\mathcal{V}_{\mathsf{C}_{\mathsf{out}}} = \mathcal{V}_{\mathsf{C1}_{\mathsf{in}}}$ where each level in $\mathcal{V}_{C_{out}}$ becomes the new bottom level of its corresponding dimension lattice; $\mathcal{M} = \mathcal{M}_1 \cup \mathcal{M}_2$, that is, the union of the measures of both cuboids.

(b) $\mathsf{c} = (c_1, c_2, \ldots, c_{D_1}, m_1, \ldots, m_{M1}, m_{M1+1}, \ldots, m_{M_1+M_2}) \in C_{out}$ such that $\exists c_p = (c_{p_1}, \ldots, c_{p_{D_1}}, m_1, \ldots, m_{M_1}) \in \mathsf{C}_{\mathsf{1}_{\mathsf{in}}}$ and $\exists c_q = (c_{q1}, \ldots, c_{q_{D_2}}, m_{M1+1}, \ldots, m_{M_1+M_2}) \in \mathsf{C}_{\mathsf{2}_{\mathsf{in}}}$ and $(c_{p_1}, \ldots, c_{p_{D_1}}) \cong (c_{q_1}, \ldots, c_{q_{D_2}})$, and $c_{p_i} = c_i, \forall i = 1..N$. Aggregate functions associated with each measure in the input cuboids are kept.

Since DRILL-ACROSS is an IGO, $\mathsf{C}_{\mathsf{out}}$ induces a new cube instance $\mathsf{CI}_{\mathsf{out}}$ such that $\mathsf{C}_{\mathsf{out}} \in \mathsf{CI}_{\mathsf{out}}$. Algortihm 4 builds $\mathsf{CI}_{\mathsf{out}}$. It first clones the lattice of cuboids CI_1, i.e., it creates a lattice of cuboids with the same schema and instance. Then, the algorithm modifies the set $\mathcal{M}$ in the cube schema, with the union of the sets of measures of both cuboids, and the set $\mathcal{D}$, by eliminating the levels that are not reachable from the current levels. Then, the DRILL-ACROSS operation is applied to each cuboid of $\mathsf{CI}_{\mathsf{out}}$ with the cuboid of CI_2 which is semantically compatible with it (i.e., all the semantically compatible cuboids are crossed). □

The extended DRILL-ACROSS operator requires either input cuboids with exactly the same schemas and instances or a defined semantic mapping between them. In the later case, all the dimensions of each cuboid must be involved in the mapping. Thus, the dimensions not included in any semantic mapping must be dropped before performing the DRILL-ACROSS, using a SLICE operation. □

Example 28 (Drill-across operator)**.** Consider the SM(Vineyard, Drinks) given in Example 22 and the cuboids $\mathsf{V}_{\mathsf{year-grape}}$ and $\mathsf{D}_{\mathsf{year-country-product}}$, belonging to Vineyard and Drinks, respectively. In order to perform a DRILL-ACROSS between these cuboids we must drop the ZoneDim dimension in the Drinks cuboid, since there is no semantic mapping for it. Before performing the SLICE, we select the country of interest in Drinks, in this case 'Belgium', by applying a DICE operation. All of the above is performed as follows:

Algorithm 4 New cube instance for DRILL-ACROSS($\mathsf{Cb1_{in}}, \mathsf{Cb2_{in}}$), with $\mathsf{Cb1_{in}} \in \mathsf{Cl_1}$, $\mathsf{Cb2_{in}} \in \mathsf{Cl_2}$ instance, where $\mathcal{D}_1 = \{\langle \mathsf{D}_{1_1}, \mathcal{L}_{1_1}, \rightarrow_{1_1}\rangle, \ldots, \langle \mathsf{D}_{1_N}, \mathcal{L}_{1_N}, \rightarrow_{1_N}\rangle\}$

$\mathsf{Cl_{out}}$ *schema* $\leftarrow \mathsf{Cl_1}$*schema*

$\mathcal{M}_{out} \leftarrow \mathcal{M}_{1_{in}} \bigcup \mathcal{M}_{2_{in}}$

$\mathcal{D}_{out} \leftarrow \{\langle \mathsf{D}_{\mathsf{out}_1}, \mathcal{L}_{out_1}, \rightarrow_{out_1}\rangle, \ldots, \langle \mathsf{D}_{\mathsf{out}_N}, \mathcal{L}_{out_N}, \rightarrow_{out_N}\rangle\}$ such that level $\mathsf{l} \in \mathcal{L}_{out_i}$ if $\mathsf{l} \in \mathcal{L}_{1_i}$ and $\langle \mathcal{L}_{out_i}, \rightarrow_{out_i}\rangle$ is a sublattice of $\langle \mathcal{L}_{1_i}, \rightarrow_{1_i}\rangle$ and $\mathsf{cl} \rightarrow^*_{out_i} \mathsf{l}$, with $\mathsf{cl} \in \mathcal{L}_{out_i} \cap \mathcal{V}_{\mathsf{C1_{in}}}, \forall i = 1..N$

$\mathsf{Cl_{out}}$ *instance* $\leftarrow \mathsf{Cl_1}$ *instance* (i.e., clone $\mathsf{Cl_1}$ as $\mathsf{C_{out}}$)

for all cuboid $\mathsf{Cb_j} \in \mathsf{Cl_{out}}$ **do**

  if $\mathsf{Cb1_{in}} \preceq \mathsf{Cb_j}$ **then**

    DRILL-ACROSS($\mathsf{Cb_j}, \mathsf{Cb_i}$) where $\mathsf{Cb_i} \in \mathsf{Cl_2}$ and $\mathsf{Cb_j} \simeq \mathsf{Cb_i}$

  else

    Eliminate $\mathsf{Cb_j}$ from $\mathsf{Cl_{out}}$

$\mathsf{Aux1} = \textsc{Dice}(\mathsf{D_{year-country-product}}, \mathsf{cyName} = \text{`Belgium'})$

$\mathsf{Aux2} = \textsc{Slice}(\mathsf{Aux1}, \mathsf{ZoneDim})$

$\mathsf{V}_{out} = \textsc{Drill-across}(\mathsf{V_{year-grape}}, \mathsf{Aux2})$

Figure 5.4 depicts the partial and final resulting cuboids. Notice that the grapes which are not indicated as wines in Drinks (for example, 'pinot blanc'), are not present in the resulting cuboid. Also notice that, opposite to the resulting cuboid in Example 20, the extended DRILL-ACROSS allows us to obtain the measures discriminated by year, and also by grape name. □

According to Definition 19, when a cuboid $\mathsf{C_{out}}$ is obtained from an extended DRILL-ACROSS, a new cube instance is induced. The dimension lattices of the cube scheme are modified and the current levels of $\mathsf{C_{out}}$ become the new bottom levels of each corresponding dimension lattice. In consequence, after performing a DRILL-ACROSS, the current levels become the bottom levels of the resulting cuboid, and therefore a ROLL-UP operation can be applied to levels *above* the current levels, but no DRILL-DOWN can be applied to levels *below* them. We explain this in Example 29 below. □

TimeDim	ZoneDim	ProductDim	Measures	
Yearh	Country	Product	(Avg)	(Avg)
year	cyName	pDetail	consumption	sales
⟨2008⟩	⟨Belgium⟩	⟨lager beer⟩	1.2	0.9
⟨2008⟩	⟨Belgium⟩	⟨stout beer⟩	1.0	0.9
⟨2008⟩	⟨Belgium⟩	⟨pale beer⟩	0.6	0.7
⟨2008⟩	⟨Belgium⟩	⟨chardonnay wine⟩	0.7	0.6
⟨2008⟩	⟨Belgium⟩	⟨malbec wine⟩	0.1	0.2
⟨2008⟩	⟨Belgium⟩	⟨kerner wine⟩	0.6	0.7
⟨2008⟩	⟨Belgium⟩	⟨whiskey⟩	0.2	0.4
⟨2008⟩	⟨Belgium⟩	⟨vodska⟩	1.9	1.6
⟨2008⟩	⟨Belgium⟩	⟨ron⟩	0.6	0.5
...	...	...	...	...
⟨2011⟩	⟨Belgium⟩	⟨lager beer⟩	1.1	0.9
⟨2011⟩	⟨Belgium⟩	⟨stout beer⟩	1.1	1.0
⟨2011⟩	⟨Belgium⟩	⟨pale beer⟩	0.6	0.6
⟨2011⟩	⟨Belgium⟩	⟨chardonnay wine⟩	0.9	0.9
⟨2011⟩	⟨Belgium⟩	⟨malbec wine⟩	0.3	0.6
⟨2011⟩	⟨Belgium⟩	⟨pinot noir wine⟩	0.5	0.6
⟨2011⟩	⟨Belgium⟩	⟨whiskey⟩	0.2	0.1
⟨2011⟩	⟨Belgium⟩	⟨cognac⟩	0.5	0.6
⟨2011⟩	⟨Belgium⟩	⟨vodska⟩	1.6	1.7
⟨2011⟩	⟨Belgium⟩	⟨ron⟩	1.0	0.9

(a) $\mathsf{Aux1} = \textsc{Dice}(\mathsf{D}_{\mathsf{year-country-product}}, \mathsf{cyName} = \text{`}\mathsf{Belgium}\text{'})$

TimeDim	ProductDim	Measures	
Year	Product	(Avg)	(Avg)
year	pDetail	consumption	sales
⟨2008⟩	⟨lager beer⟩	1.2	0.9
⟨2008⟩	⟨stout beer⟩	1.0	0.9
⟨2008⟩	⟨pale beer⟩	0.6	0.7
⟨2008⟩	⟨chardonnay wine⟩	0.7	0.6
⟨2008⟩	⟨malbec wine⟩	0.1	0.2
⟨2008⟩	⟨kerner wine⟩	0.6	0.7
⟨2008⟩	⟨whiskey⟩	0.2	0.4
⟨2008⟩	⟨vodska⟩	1.9	1.6
⟨2008⟩	⟨ron⟩	0.6	0.5
...	...	...	...
⟨2011⟩	⟨lager beer⟩	1.1	0.9
⟨2011⟩	⟨stout beer⟩	1.1	1.0
⟨2011⟩	⟨pale beer⟩	0.6	0.6
⟨2011⟩	⟨chardonnay wine⟩	0.9	0.9
⟨2011⟩	⟨malbec wine⟩	0.3	0.6
⟨2011⟩	⟨pinot noir wine⟩	0.5	0.6
⟨2011⟩	⟨whiskey⟩	0.2	0.1
⟨2011⟩	⟨cognac⟩	0.5	0.6
⟨2011⟩	⟨vodska⟩	1.6	1.7
⟨2011⟩	⟨ron⟩	1.0	0.9

(b) $\mathsf{Aux2} = \textsc{Slice}(\mathsf{Aux1}, \mathsf{ZoneDim})$

TimeDim	BlockDim	Measures		
Year	Grape	(Sum)	(Avg)	(Avg)
year	gName	harvest	consumption	sales
⟨2008⟩	⟨malbec⟩	25000	0.1	0.2
⟨2008⟩	⟨kerner⟩	22900	0.6	0.7
...	...	...	...	...
⟨2011⟩	⟨chardonnay⟩	22100	0.9	0.9
⟨2011⟩	⟨malbec⟩	24000	0.3	0.6
⟨2011⟩	⟨pinot noir⟩	23800	0.5	0.6

(c) $\mathsf{V}_{\mathsf{out}} = \textsc{Drill-across}(\mathsf{V}_{\mathsf{year-grape}}, \mathsf{Aux2})$

Figure 5.4: Applying the extended Drill-across operator.

Example 29 (Drill-across and Drill-down)**.** Consider the cuboid $\mathsf{V_{out}}$ obtained in Example 28 (Figure 5.4(c)).

We cannot apply DRILL-DOWN($\mathsf{V_{out}}$, BlockDim, Block), because the values of the measure consumption have been imported from another cuboid and they are not defined for the members of the level Block; thus, there is no way to precisely compute how to disaggregate this measure into members that had never stored their values. This can be seen for example in the first tuple of $\mathsf{V_{out}}$ in Figure 5.4(c): the value 0.1 for the measure consumption was computed as a result of a slice operation, and is not actually associated to a mamber in Block.

On the contrary, it is possible to apply ROLL-UP($\mathsf{V_{out}}$, BlockDim, GrapeType) or ROLL-UP($\mathsf{V_{out}}$, BlockDim, All), using the same aggregate function AVG associated to the measure consumption in the original cuboid. The resulting cuboids of these two possible ROLL-UP operations are shown in Figure 5.5. □

TimeDim	BlockDim	Measures		
Year	GrapeType	(Sum)	(Avg)	(Avg)
year	gType	harvest	consumption	sales
⟨2008⟩	⟨red⟩	25000	0.5	0.48
⟨2008⟩	⟨white⟩	22900	0.42	0.52
⟨2009⟩	⟨red⟩	25600	0.9	0.87
⟨2009⟩	⟨white⟩	20200	0.92	0.93
⟨2010⟩	⟨red⟩	33800	0.66	0.69
⟨2010⟩	⟨white⟩	22600	0.74	0.57
⟨2011⟩	⟨red⟩	23800	0.5	0.68
⟨2011⟩	⟨white⟩	47800	0.64	0.72

(a) ROLL-UP($\mathsf{V_{year-grape}}$, BlockDim), GrapeType

TimeDim	BlockDim	Measures		
Year	All	(Sum)	(Avg)	(Avg)
year	all	harvest	consumption	sales
⟨2008⟩	⟨all⟩	47900	0.46	0.5
⟨2009⟩	⟨all⟩	45800	0.91	0.9
⟨2010⟩	⟨all⟩	56400	0.7	0.63
⟨2011⟩	⟨all⟩	71600	0.57	0.7

(b) ROLL-UP($\mathsf{V_{year-grape}}$, BlockDim), All

Figure 5.5: Possible ROLL-UP operations after drilling-across.

In many cases, the goal of crossing two cubes is not only to easily compare the measures from different cuboids (because they can take together), but also to operate with such measures. For example, suppose that, in addition

to drilling-across Vineyard and Drinks, we want to obtain a new measure, like $0.01 \star \mathsf{harvest/consumption}$. That is, we want to allow to add new measures to the resulting cube, through functions over input measures and literals. In this way, Definition 19 must be modified to add as an arguments the expressions that compute the new measures, together with the corresponding aggregation function. Then, the operator would be invoked as: DRILL-ACROSS($\mathsf{C1}_{in}, \mathsf{C2}_{in}, \mathsf{NewMeasures}$), where $\mathsf{NewMeasures} = \{\langle \mathsf{fn}_1, \mathsf{name}_1, \mathsf{agg}_1 \rangle, \ldots, \langle \mathsf{fn}_\mathsf{K}, \mathsf{name}_\mathsf{K}, \mathsf{agg}_\mathsf{K} \rangle\}, \mathsf{K} \geq 0$, where each fn_i is a function over elements $\mathsf{m} \in \mathcal{M}_1 \cup \mathcal{M}_2 \cup \mathsf{Literals}$, name_i is the name of a new measure, and $\mathsf{agg}_\mathsf{i} \in \mathcal{A}$ is the aggregate function associate to the new measure $\mathsf{name}_\mathsf{i}, \forall i = 1 \ldots K$. Each coordinate of the cuboid will be as in Definition 19, with the addition of the new calculated measures, as follows: $\mathsf{c} \in C_{out} = (c_1, c_2, \ldots, c_N, m_1, \ldots, m_{M1}, m_{M1+1}, \ldots, m_{M_1+M_2}, m_{M_1+M_2+1}, \ldots, m_{M_1+M_2+K})$.

Since DRILL-ACROSS is an IGO, $\mathsf{C_{out}}$ induces a new cube instance $\mathsf{CI_{out}}$ such that $\mathsf{C_{out}} \in \mathsf{CI_{out}}$. Algorithm 5 shows how the new cube instance CI_{out} is built when new measures are defined in DRILL-ACROSS.

Example 30 (Drill-across operator and New Measures)**.** Consider the same cubes and cuboids $\mathsf{V_{year-grape}}$ and Aux2 of Example 28, and the set $\mathsf{NewMeasures} = \{\langle \text{FM}, \mathsf{relation}, \text{AVG} \rangle\}$, where $\text{FM} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, with $\text{FM}(\mathsf{harvest}, \mathsf{consumption}) = 0.01 * \mathsf{harvest}/(\mathsf{consumption} * 100)$. Figure 5.6 depicts the resulting cuboid of DRILL-ACROSS($\mathsf{V_{year-grape}}, \mathsf{Aux2}, \mathsf{NewMeasures}$). □

TimeDim	BlockDim	Measures			
Year	Grape	(Sum)	(Avg)	(Avg)	(Avg)
year	gName	harvest	consumption	sales	relation
⟨2008⟩	⟨malbec⟩	25000	0.1	0.2	25.0
⟨2008⟩	⟨kerner⟩	22900	0.6	0.7	3.8
...	...	...	...	...	...
⟨2011⟩	⟨chardonnay⟩	22100	0.9	0.9	2.5
⟨2011⟩	⟨malbec⟩	24000	0.3	0.6	8.0
⟨2011⟩	⟨pinot noir⟩	23800	0.5	0.6	4.8

Figure 5.6: Extended DRILL-ACROSS with new Measures.

Algorithm 5 New cube instance for DRILL-ACROSS($\mathsf{Cb1_{in}}, \mathsf{Cb2_{in}}, \mathsf{NewMeasures}$), with $\mathsf{Cb1_{in}} \in \mathsf{Cl_1}$, $\mathsf{Cb2_{in}} \in \mathsf{Cl_2}$ instance, where $\mathcal{D}_1 = \{\langle \mathsf{D}_{1_1}, \mathcal{L}_{1_1}, \rightarrow_{1_1}\rangle, \ldots, \langle \mathsf{D}_{1_N}, \mathcal{L}_{1_N}, \rightarrow_{1_N}\rangle\}$

$\mathsf{Cl_{out}}$ *schema* $\leftarrow$ $\mathsf{Cl_1}$*schema*

$\mathcal{M}_{out} \leftarrow \mathcal{M}_1 \bigcup \mathcal{M}_2 \bigcup \mathcal{M}_{new}\rangle$ where $\mathcal{M}_{new} = \{\mathsf{name_1}, \ldots, \mathsf{name_K}\}$

$\mathcal{D}_{out} \leftarrow \{\langle \mathsf{D}_{\mathsf{out}_1}, \mathcal{L}_{out_1}, \rightarrow_{out_1}\rangle, \ldots, \langle \mathsf{D}_{\mathsf{out}_N}, \mathcal{L}_{out_N}, \rightarrow_{out_N}\rangle\}$ such that level $\mathsf{l} \in \mathcal{L}_{out_i}$ if $\mathsf{l} \in \mathcal{L}_{1_i}$ and $\langle \mathcal{L}_{out_i}, \rightarrow_{out_i}\rangle$ is a sublattice of $\langle \mathcal{L}_{1_i}, \rightarrow_{1_i}\rangle$ and $\mathsf{cl} \rightarrow^*_{out_i} \mathsf{l}$, with $\mathsf{cl} \in \mathcal{L}_{out_i} \cap \mathcal{V}_{\mathsf{C1_{in}}}, \forall i = 1..N$

$\mathsf{Cl_{out}}$ *instance* $\leftarrow$ $\mathsf{Cl_1}$ *instance* (i.e., clone $\mathsf{Cl_1}$ as $\mathsf{C_{out}}$)

for all cuboid $\mathsf{Cb_j} \in \mathsf{Cl_{out}}$ **do**

  if $\mathsf{Cb1_{in}} \preceq \mathsf{Cb_j}$ **then**

    DRILL-ACROSS($\mathsf{Cb_j}, \mathsf{Cb_i}$) where $\mathsf{Cb_i} \in \mathsf{Cl_2}$ and $\mathsf{Cb}_j \simeq \mathsf{Cb_i}$

  else

    Eliminate $\mathsf{Cb_j}$ from $\mathsf{Cl_{out}}$

5.4 Discussion

Consider two cubes $\mathsf{C_1}$ and $\mathsf{C_2}$, both of them with a dimension GeomDim, and hierarchy District $\rightarrow$ Province $\rightarrow$ All, where the measure of $\mathsf{C_1}$ is radiation, and the measure of $\mathsf{C_2}$ is pollution (with the function SUM associated to both of them).

Now, suppose an instance of $\mathsf{C_1}$ containing two districts, Arlon and Bastogne, with radiation values 20 and 18, respectively, such that both of them roll-up to the Luxembourg province. Also, and instance of $\mathsf{C_2}$ contains districts Neufchateau and Virton, with pollution values 6 and 11, also rolling-up to Luxembourg.

Although a semantic mapping $\mathsf{SM}(\mathsf{C_1}, \mathsf{C_2})$ may identify as equivalent both District levels, and both Province levels, it is clear that if District is the current level of GeomDim in both cubes, then DRILL-ACROSS($\mathsf{C_1}$, $\mathsf{C_2}$) is empty, because there is not match between the District levels.

In spite of the above, if we apply $\mathsf{D_1}$ = ROLL-UP($\mathsf{C_1}$, GeomDim, Province), and $\mathsf{D_2}$ = ROLL-UP($\mathsf{C_2}$, GeomDim, Province), then DRILL-ACROSS($\mathsf{D_1}, \mathsf{D_2}$) results in a cube containing the Luxembourg province, and values 38 and 17 for measures

radiation and pollution, respectively. As a drawback of this, once DRILL-ACROSS is applied we will not be able to navigate the hierarchy downwards (that is, we will not be able to drill-down).

This raises the question: would this be correct? Let us imagine that temperature and humidity are measured in regions, but represented as single values for a whole region, although measured using different numbers of sensors located in different geographical points, registering values at different instants. In fact, we could find many few matches at the point and timestamp granularity, but it would make sense to cross such information at a region level by summarizing the behavior of regions as a whole.

Note that this problem is also present when drilling across traditional cubes although they share dimensions. For example, if we have two cubes with data about sales and factories. Both of them record the information in the operational systems at the timestamp granularity, but in the data warehouses this information is probably summarized at weekly granularity. Although no match may occur at the timestamp granularity, we could drill-across both cubes at a coarser level, say week, i.e., a comparison can be done when analyzing what happened during weeks.

We remark that only the user can decide which levels are really semantically equivalent, and define the correct semantic mapping according to this decision.

5.5 Semantic Mapping and OLAP operations

When a semantic mapping is defined between two cubes, the cuboid resulting from the OLAP operations applied to cuboids belonging to those cubes, induces a new semantic mapping over the output cube produced by such operator.

For example, consider cuboids Cb_1 and Cb_2 belonging to cubes C1 and C2, respectively, a semantic mapping SM(C1,C2), and a DICE operation of the form DICE(Cb_1, ϕ) which produces cuboid Cb_3. This cuboid induces a cube C3 such that there is an induced semantic mapping SM(C3,C2) whose sets $\mathcal{P}$ and $\mathcal{S}$ are subsets of the ones in SM(C1,C2). We next study how the mappings induced by the OLAP operations are defined by means of a set of rules. For simplicity, we describe the

rules by using the operation over the first cube of the semantic mapping, but the same applies for the second one. We use these rules later in Section 6.3.

5.5.1 Semantic Mapping and Roll-Up

Given two cubes $\mathsf{C1}$ and $\mathsf{C2}$ with $\mathsf{SM(C1,C2)}=\langle\mathcal{P},\mathcal{S}\rangle$, a cuboid $\mathsf{Cb}_1 \in \mathsf{C1}$ and a cuboid $\mathsf{Cb}_2 \in \mathsf{C2}$, since the ROLL-UP operator is an IPO, the cube corresponding to the cuboid $\mathsf{Cb}_3 = \text{ROLL-UP}(\mathsf{Cb}_1, \mathsf{D}, \mathsf{L})$ is $\mathsf{C1}$ itself. Thus, the semantic mapping is preserved for further operations (e.g., a DRILL-ACROSS).

5.5.2 Semantic Mapping and Drill-down

Given two cubes $\mathsf{C1}$ and $\mathsf{C2}$ with $\mathsf{SM(C1,C2)}=\langle\mathcal{P},\mathcal{S}\rangle$, a cuboid $\mathsf{Cb}_1 \in \mathsf{C1}$ and a cuboid $\mathsf{Cb}_2 \in \mathsf{C2}$, since the DRILL-DOWN operator is an IPO, the cube corresponding to the cuboid $\mathsf{Cb}_3 = \text{DRILL-DOWN}(\mathsf{Cb}_1, \mathsf{D}, \mathsf{L})$ is, like above, $\mathsf{C1}$ itself, and the mapping is preserved.

5.5.3 Semantic Mapping and Dice

Consider two cubes $\mathsf{C1}$ and $\mathsf{C2}$, a semantic mapping $\mathsf{SM(C1,C2)}=\langle\mathcal{P},\mathcal{S}\rangle$, a cuboid $\mathsf{Cb}_1 \in \mathsf{C1}$ and a cuboid $\mathsf{Cb}_2 \in \mathsf{C2}$. The operation $\text{DICE}(\mathsf{Cb}_1, \phi)$ produces a cuboid Cb_3. In addition, the operation induces a new cube $\mathsf{C3}$, and a new mapping $\mathsf{SM(C3, C2)} = \langle\mathcal{P},\mathcal{S}'\rangle$ such that $\forall\text{SMAP}^{\mathsf{s}_2}_{\mathsf{s}_1} \in \mathcal{S}$, $\text{SMAP'}^{\mathsf{s}_2}_{\mathsf{s}_1} \in \mathcal{S}'$ is given by $\text{SMAP'}^{\mathsf{s}_2}_{\mathsf{s}_1} = \text{SMAP}^{\mathsf{s}_2}_{\mathsf{s}_1} - \{(\mathsf{m}_1, \mathsf{m}_2) | \mathsf{m}_1 \text{ is a member deleted from the } \mathsf{C3} \text{ instance}\}$.

Notice that the pairs of $\mathcal{P}$ are the same of the original semantic mapping, because the DICE operator only introduces changes in the cube instance, without changing the cube schema.

5.5.4 Semantic Mapping and Slice

Given two cubes $\mathsf{C1}$ and $\mathsf{C2}$ with $\mathsf{SM(C1,C2)}=\langle\mathcal{P},\mathcal{S}\rangle$, a cuboid $\mathsf{Cb}_1 \in \mathsf{C1}$ and a cuboid $\mathsf{Cb}_2 \in \mathsf{C2}$, the operation $\text{SLICE}(\mathsf{Cb}_1, \mathsf{D})$ produces a cuboid Cb_3 which induces the cube $\mathsf{C3}$ and a mapping $\mathsf{SM(C3, C2)} = \langle\mathcal{P}',\mathcal{S}'\rangle$ such that $\mathcal{P}' = \mathcal{P} -$

$\{(\mathsf{s}_1, \mathsf{s}_2)|\exists \mathsf{l}_i \in \mathsf{s}_1$ where l_i is a level of $\mathsf{D}, \forall i = 1..|\mathsf{D}|\}$ and $\mathcal{S}' = \mathcal{S} - \{(\text{SMAP}^{\mathsf{s}_2}_{\mathsf{s}_1}|\exists \mathsf{l}_i \in \mathsf{s}_1$ such that l_i is a level of $\mathsf{D}, \forall i = 1..|\mathsf{D}|\}$

In simple words, all the pairs containing levels of the eliminated dimension, are eliminated from $\mathcal{P}$. The member mappings of $\mathcal{S}$ are modified accordingly.

5.5.5 Semantic Mapping and Drill-across

Given two cubes C1 and C2 with SM(C1,C2)=$\langle \mathcal{P}, \mathcal{S} \rangle$, a cuboid $\mathsf{Cb}_1 \in$ C1 and a cuboid $\mathsf{Cb}_2 \in$ C2, the operation $\mathsf{Cb}_3 = \text{DRILL-ACROSS}(\mathsf{Cb}_1, \mathsf{Cb}_2)$ induces a new cube C3 and a mapping $\mathsf{SM}(\mathsf{C3}, \mathsf{C2}) = \langle \mathcal{P}', \mathcal{S}' \rangle$ such that $\mathcal{P}' = \mathcal{P} - \{(\mathsf{s}_1, \mathsf{s}_2)|\exists \mathsf{l}_i \in \mathsf{s}_1$ where $\mathsf{l}_1 \in \mathsf{D}_i$ in Cb_1, and l_1 cannot reach a level in $\mathcal{V}_{Cb_3}\}$ and $\mathcal{S}' = \mathcal{S} - \{(\text{SMAP}^{\mathsf{s}_2}_{\mathsf{s}_1}|\exists \mathsf{l}_i \in \mathsf{s}_1$ such that $\mathsf{l}_1 \in \mathsf{D}_i$ dimension of Cb_1 and l_1 is not reachable from $\mathcal{V}_{Cb_3}\}$. Since the drill-across operation introduces a cut in the dimension lattice, all the pairs containing deleted levels, are removed from $\mathcal{P}$. Thus, the member mappings of $\mathcal{S}$ are modified accordingly.

5.6 Summary

In this chapter we have extended the DRILL-ACROSS operation introducing the concept of semantic mapping between cubes, allowing us to solve the differences between dimension names, level names, member representations or even the structure of dimension lattices. The Semantic Mapping between cubes is based on the semantic relationships dimension-dimension derivation and dimension-dimension association, and consists in two steps: choosing each set of dimension levels from in cube that are semantically equivalent to a set of levels in the other cube, and then defining a mapping of level members for each pair of semantically equivalent sets of levels. In addition, we introduced the concepts of semantically compatible cuboids and semantically equivalent coordinates. We further extended the DRILL-ACROSS operator with the possibility of adding new measures to the resulting cuboid applying functions over the measures of the input ones. Finally, we showed that when a semantic mapping is defined between two cubes, the result of each OLAP operation (i.e., a cuboid) also induces a new semantic mapping.

Chapter 6

A Case Study

In this chapter we integrate the concepts studied in the previous chapters, and show how they can be applied to a complex real-world case study. This case study is based on the running example introduced in Chapter 1, that is, the system for the analysis of wine production in Belgium. In order to be even more comprehensive, we also include a cube containing data about the trajectories of field fumigation flight trajectories. All in all, we consider five non-homogeneous cubes, i.e., cubes that do not have the same dimensions and instances. We show how queries over these cubes can be addressed using the OLAP operators we have introduced in Chapters 4 and 5, regardless the data actually contained in these cubes. We first show sample queries over individual cubes, and then we integrate them in global queries, that is, queries that combine different kinds of cubes.

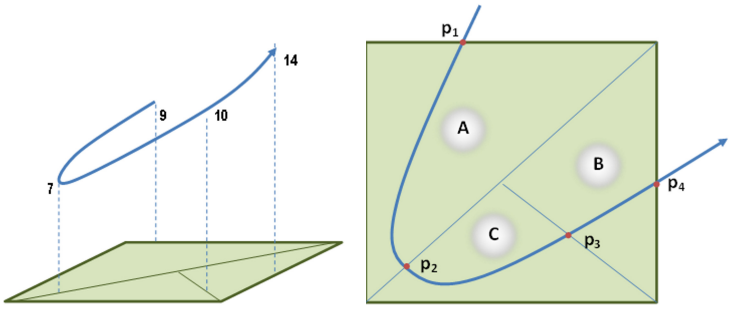
6.1 Cubes of the Case Study

Since our case study is based on vineyard production and how climate changes impact directly on it, information about the environment is relevant for analysis. Thus, besides the Vineyard cube given in Example 11, we will introduce three cubes with the following environmental information: Temperature, Precipitation, and Altitude. A fourth cube, named Fumigation, with information about pesticide spraying, is also included for analysis.

We remark that we model the BlockDim dimension using the GS1 numbering

system for the wine supply chain traceability,[1] as we explain next. The key data required for traceability purposes is the identification of the Block from which the grape comes. Each block is identified with a Global Location Number (GLN). If the grape grower changes the type of its grape planting, the GLN is not further used. Thus, a GLN or idBlock can be associated with only one type of grape. Our design of the BlockDim dimension responds to this restriction and guarantees traceability, at the expense of some complexity in query formulation, as we will see in Section 6.2.

The Temperature and Precipitation cubes register, respectively, temperature and precipitation data at predefined points of Belgium during years 2006 through 2010. The Altitude cube stores terrain altitude with respect to the sea level. No temporal information is recorded in this cube, since we consider that altitude does not change over time. Finally, the cube Fumigation contains information about the flight trajectory reported by each pilot after the fumigation over a plantation. In this sense, the measure of this cube is not a traditional discrete one, but a 3-D geometry whose aggregate function is the geometric UNION. Each reported 3-D trajectory is partitioned into the sub-trajectories that correspond to teh geometry of each plantation. For this, we take into account the first and last point whose XY-projections fall into the geometry area. Figure 6.1 shows a 3-D trajectory partitioned into three 2-D sub-trajectories over zones A, B and C.

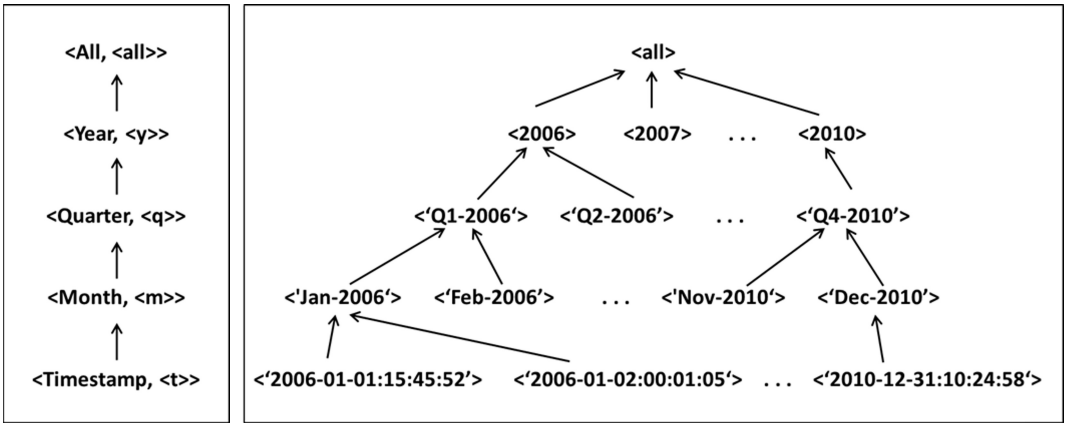


(a) A spatial view (b) A projection over the plane

Figure 6.1: Two views of a fumigation trajectory. The points p_1, p_2, p_3 and p_4 are the limits of each sub-trajectory over zones A, B and C.

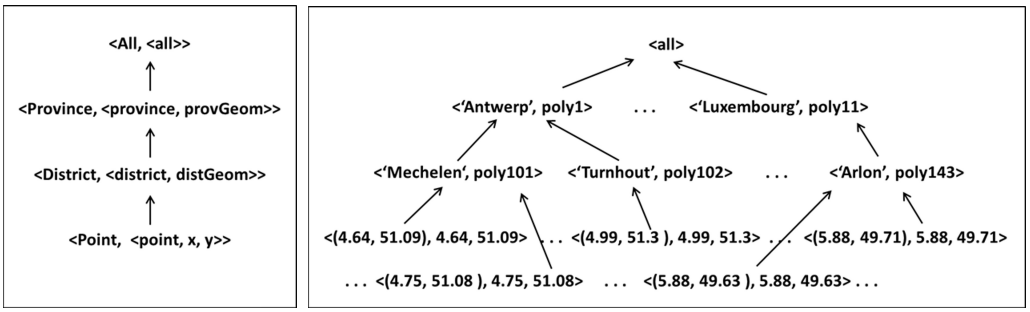
[1] http://www.gs1.org

The Temperature and Precipitation cube schemas share the dimensions TemporalDim and SpatialDim. The latter is also shared by the Altitude cube. Figures 6.2 and 6.3 depict the TemporalDim and SpatialDim dimensions, respectively.



(a) Dimension lattice (b) Dimension instance

Figure 6.2: TemporalDim Dimension.

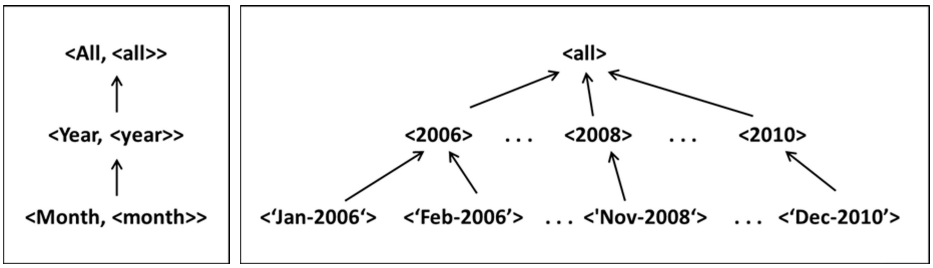


(a) Dimension lattice (b) Dimension instance

Figure 6.3: SpatialDim Dimension.

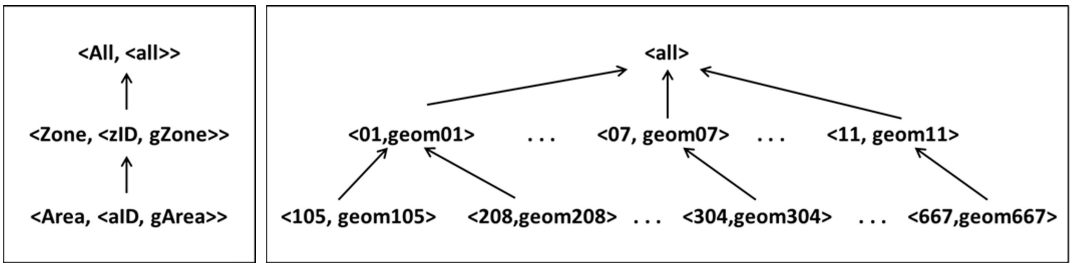
The Fumigation cube schema contains the dimensions DateDim, GeoDim and PestDim, whose schemas and instances can be seen in Figures 6.4, 6.5 and 6.6, respectively.

Figures 6.7 through 6.11 show the bottom cuboids of the five cube instances.



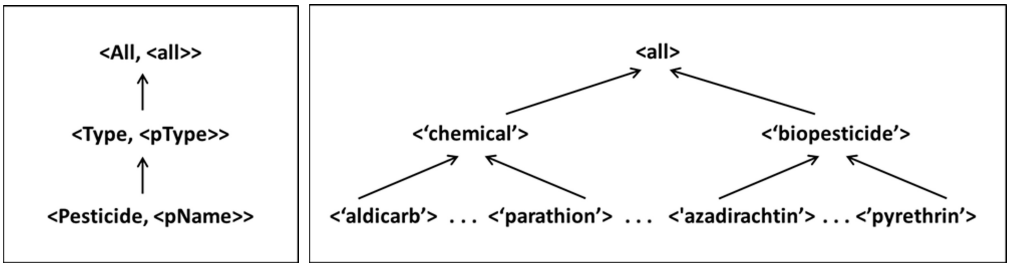
(a) Dimension lattice (b) Dimension instance

Figure 6.4: DateDim Dimension.



(a) Dimension lattice (b) Dimension instance

Figure 6.5: GeoDim Dimension.



(a) Dimension lattice (b) Dimension instance

Figure 6.6: PestDim Dimension.

Remark 9. *For the sake of space, the geometric elements in dimensions and measures are shown through identifier names instead of their real geometry. For example, in Figure 6.11 we simply show the trajectory given by the geometry Linestring(4.48141 51.37759 11.55681, 4.65486 51.30325 11.58643, 4.81180 51.35281 11.9762,*

4.93982 51.27848 12.00780, 4.86136 51.19174 12.15689, 4.47728 51.26195 12.34782) as 't12'. □

TimeDim Day date, dayOfWeek	BlockDim Block idBlock, bGeom	Measures (Sum) harvest
⟨'2007-08-06','Mon'⟩	⟨35001, g35001⟩	7200
⟨'2007-08-06','Mon'⟩	⟨35002, g35002⟩	7500
⟨'2007-09-13','Thu'⟩	⟨35003, g35003⟩	7300
⟨'2007-09-14','Fri'⟩	⟨35004, g35004⟩	7200
⟨'2007-09-14','Fri'⟩	⟨35005, g35005⟩	7400
⟨'2007-10-10','Wed'⟩	⟨35006, g35006⟩	8000
⟨'2008-08-08','Fri'⟩	⟨35001, g35001⟩	7800
⟨'2008-08-11','Mon'⟩	⟨35002, g35002⟩	7600
⟨'2008-08-12','Tue'⟩	⟨35003, g35003⟩	7900
⟨'2008-09-16','Tue'⟩	⟨35004, g35004⟩	8100
⟨'2008-09-17','Wed'⟩	⟨35005, g35005⟩	7500
⟨'2008-11-12','Wed'⟩	⟨35006, g35006⟩	7100
⟨'2009-08-04','Tue'⟩	⟨35001, g35001⟩	7200
⟨'2009-08-04','Tue'⟩	⟨35002, g35002⟩	7800
⟨'2009-08-06','Thu'⟩	⟨35003, g35003⟩	8000
⟨'2009-09-08','Tue'⟩	⟨35004, g35004⟩	7800
⟨'2009-09-09','Wed'⟩	⟨35005, g35005⟩	7400
⟨'2009-10-15','Thu'⟩	⟨35006, g35006⟩	7600
⟨'2010-08-12','Thu'⟩	⟨35077, g35001⟩	7500
⟨'2010-08-13','Fri'⟩	⟨35078, g35001⟩	7400
⟨'2010-09-16','Thu'⟩	⟨35079, g35001⟩	7700
⟨'2010-09-16','Thu'⟩	⟨35084, g35004⟩	8100
⟨'2010-10-11','Tue'⟩	⟨35085, g35005⟩	7900
...	...	...
⟨'2010-10-12','Wed'⟩	⟨35086, g35006⟩	7800
⟨'2011-08-11','Thu'⟩	⟨35077, g35001⟩	7500
⟨'2011-08-12','Fri'⟩	⟨35078, g35001⟩	7500
⟨'2011-08-12','Fri'⟩	⟨35079, g35001⟩	7800
⟨'2011-09-06','Tue'⟩	⟨35084, g35004⟩	7900
⟨'2011-09-07','Wed'⟩	⟨35085, g35005⟩	8000
...	...	...
⟨'2011-11-10','Thu'⟩	⟨35086, g35006⟩	7900

Figure 6.7: Vineyard bottom cuboid

TemporalDim Timestamp t	SpaceDim Point point, x, y	Measures (Avg) temperature
⟨'2007-08-06:01:07:54'⟩	⟨(2.58,49.42), 2.58, 49.42⟩	17.8
...	...	...
⟨'2009-09-11:10:52:02'⟩	⟨(4.75,51.08), 4.75, 51.08⟩	15.2
⟨'2009-09-11:12:05:46'⟩	⟨(4.99,51.30), 4.99,51.30⟩	14.6
⟨'2009-09-11:36:10:32'⟩	⟨(5.88,49.63), 5.88, 49.63⟩	13.9
...	...	...
⟨'2010-12-26:00:12:48'⟩	⟨(6.42,51.58), 6.42, 51.58⟩	3.1

Figure 6.8: Temperature bottom cuboid

TemporalDim Timestamp t	SpaceDim Point point, x, y	Measures (Avg) precipitation
⟨'2007-08-06:01:07:54'⟩	⟨(2.58,50.75), 2.58, 50.75⟩	56
...	...	...
⟨'2009-09-11:10:52:02'⟩	⟨(4.75,51.08), 4.75,51.08⟩	66
⟨'2009-09-11:12:05:46'⟩	⟨(4.92,51.42), 4.92, 51.42⟩	69
⟨'2009-09-11:36:10:32'⟩	⟨(5.92,50.25), 5.92, 50.25⟩	78
...	...	...
⟨'2010-12-26:00:12:48'⟩	⟨(6.42,50.42), 6.42, 50.42⟩	122

Figure 6.9: Precipitation bottom cuboid

SpaceDim Point point, x, y	Measures (Max) altitude
⟨(2.55,49.47), 2.55, 49.47⟩	114
...	...
⟨(3.79,50.55), 3.79, 50.55⟩	65
⟨(4.19,50.64), 4.19, 50.64⟩	96
⟨(4.28,50.63), 4.28, 50.63⟩	121
...	...
⟨(6.37,51.57), 6.37, 51.57⟩	25

Figure 6.10: Altitude bottom cuboid

DateDim Month month	GeoDim Area aID, gArea	PestDim Pesticide pName	Measures (Union) trajectory
⟨Jan-2007⟩	⟨105, geom105⟩	⟨'aldicarb'⟩	t12
⟨Feb-2007⟩	⟨208, geom208⟩	⟨'parathion'⟩	t22
...	...	...	...
⟨Jan-2010⟩	⟨105, geom105⟩	⟨'aldicarb'⟩	t104
⟨Jan-2010⟩	⟨304, geom304⟩	⟨'aldicarb'⟩	t107
⟨Jan-2010⟩	⟨105, geom105⟩	⟨'parathion'⟩	t118
⟨Jan-2010⟩	⟨304, geom304⟩	⟨'aldicarb'⟩	t132
⟨Jan-2010⟩	⟨667 ,geom667⟩	⟨'aldicarb'⟩	t162
⟨Jul-2010⟩	⟨667 ,geom667⟩	⟨'azadirachtin'⟩	t162
...	...	...	...
⟨Sep-2010⟩	⟨208, geom208 ⟩	⟨'pyrethrin'⟩	t205

Figure 6.11: Fumigation bottom cuboid

We next address a collection a collection of queries to the cubes described above, using the cube algebra operations defined in the previous chapters.

6.2 Queries over Individual Cubes

We start with some example queries posed over individual cubes (that is, queries using unary OLAP operators).

Query 1

"*Average temperature at the center of Brussels (Long 4.37°, Lat 50.84°) during June of 2010*". This query is applied over the Temperature cube. The following expressions answer the query:

Q1-Aux1 = ROLL-UP(Temperature, TemporalDim, Month)
Q1 = DICE(Q1-Aux1, point = (4.37,50.84) and m = 'Jun-2010')

We first aggregate the measures applying a ROLL-UP to the Month level in order to be able to select a particular month and coordinate point. Figure 6.12 shows the resulting cuboid.

Notice that, according to the modeled dimension level, another option for the second operation (i.e., DICE) would be to ask for the two coordinates separately, as follows:

Q1 = DICE(Q1-Aux1, x = 4.37 and y = 50.84 and m = 'Jn-2010')

It is important to point out that if we applied a DRILL-DOWN operation over Q1 to the Timestamp level in the dimension TemporalDim, the resulting cuboid would only contain timestamps corresponding to June 2010.

TemporalDim Month m	SpaceDim Point point, x, y	Measures (Avg) temperature
⟨'Jun-2010'⟩	⟨(4.37,50.84), 4.37, 50.84⟩	18.0

Figure 6.12: Resulting cuboid of Query 1.

Remark 10. *According to the TemporalDim schema in Figure 5.1(a), if the values of the members in the Month instance did not have the year concatenated, a* ROLL-UP *to Year level would be needed to select the year 2010. Then a* DRILL-DOWN *to the Month level would be required to choose June. In this case, although the result is the same that with Q1, the query becomes more complex.* □

Query 2

"*Districts where cabernet sauvignon was produced during 2010, with a harvest greater than 34000*". This query operates over the Vineyard cube.

We cannot ask simultaneously for grapes and districts because both of them have been modeled as different levels of the same spatial dimension (see Figure 3.2). Thus, we need to perform ROLL-UP and DRILL-DOWN operations as follows:

Q2-Aux1 = ROLL-UP(Vineyard, TimeDim, Year)
Q2-Aux2 = DICE(Q2-Aux1, year = 2010)
Q2-Aux3 = ROLL-UP(Q2-Aux2, BlockDim, Grape)
Q2-Aux4 = DICE(Q2-Aux3, gName = 'cavernet sauvignon')
Q2-Aux5 = DRILL-DOWN(Q2-Aux4, BlockDim, Block)
Q2-Aux6 = ROLL-UP(Q2-Aux5, BlockDim, District)
Q2 = DICE(Q2-Aux6, harvest >= 34000)

In Q2-Aux1 we aggregate time values up to Year level, in order to select the elements corresponding to the year 2010 in Q2-Aux2. Since we need to select an specific grape, in Q2-Aux3 we aggregate to the Grape level and then select 'cabernet sauvignon' in Q2-Aux4. Thus, since the dice operation induced a new cube instance according to the DRILL-DOWN semantics already explained, Q2-Aux5 will only contain blocks whose planting during 2010 is 'cabernet sauvignon'. Thus, in Q2-Aux6 we aggregate these blocks to the District level and finally obtain the desired cuboid Q2 by dicing the correct harvest value. Figure 6.13 shows the resulting cuboid.

TimeDim Year year	BlockDim District dName, dGeom	Measures (Sum) harvest
⟨2010⟩	⟨Antwerp, g55⟩	36600
⟨2010⟩	⟨Brabant, g80⟩	34800
⟨2010⟩	⟨Leuven, g18⟩	46800

Figure 6.13: Resulting cuboid for Query 2.

It is important to point out that, although the current levels of Q2 are Year

and District, if we now roll-up to Province level, only values corresponding to the selected grape 'cabernet sauvignon' will be aggregated.

Query 3

The next query operates over the Fumigation cube. "*Zones where the length of fumigation trajectories with 'chemical' pesticide during January 2010 is larger than 100 km*". The following expressions solve the query:

Q3-Aux1 = ROLL-UP(Fumigation, GeoDim, Zone)
Q3-Aux2 = ROLL-UP(Q3-Aux1, PestDim, Type)
Q3 = DICE(Q3-Aux2, month = 'Jan-2010'' and pType = 'chemical' and LENGTH(trajectory) > 100)

For this query, we used the dimension instances depicted in Figures 6.4(b), 6.5(b) and 6.6(b). The measure is a 3-D trajectory over which geometric functions can be applied, e.g., LENGTH, UNION, CONTAINS, etc. The two ROLL-UP operations compute the union (the aggregation function associated to the measure) of all the 3-D trajectories by month, zone and pesticide type. The length of the union is the sum of the lengths of all of the trajectories that compose it. Thus, a dice condition over such length be applied. Figure 6.14 shows the resulting cuboid.

DateDim Month month	GeoDim Zone zID, gZone	PestDim Type pType	Measures (Union) trajectory
⟨Jan-2010⟩	⟨01, geom01⟩	⟨chemical⟩	t104 ∪ t118
⟨Jan-2010⟩	⟨07, geom07⟩	⟨chemical⟩	t107 ∪ t132
⟨Jan-2010⟩	⟨11 ,geom11⟩	⟨chemical⟩	t162

Figure 6.14: Resulting cuboid of Query 3.

6.3 Integrated Queries

We will now show queries that combine information in the Vineyard cube with the rest of the cubes (Precipitation, Temperature, etc.). For this, we must define a

semantic mapping between the corresponding cubes, since, although the temporal and spatial dimensions conceptually express the same concepts, there are some differences between them that must be addressed prior to apply a drill-across operation. Figures 6.15 and 6.16 graphically show the corresponding pairs of semantically equivalent levels of $\mathcal{P}$ for SM(Vineyard, Temperature), SM(Vineyard, Precipitation), SM(Vineyard, Altitude) and SM(Vineyard, Fumigation).

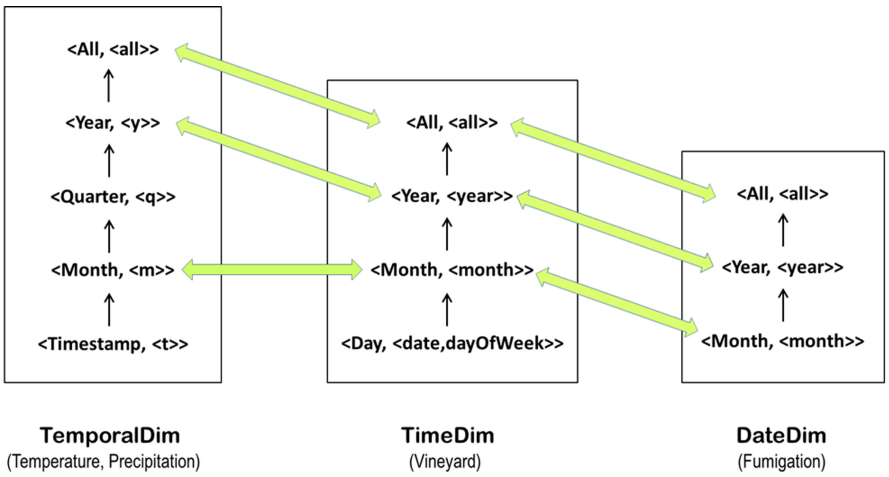


Figure 6.15: Semantic mapping between temporal levels.

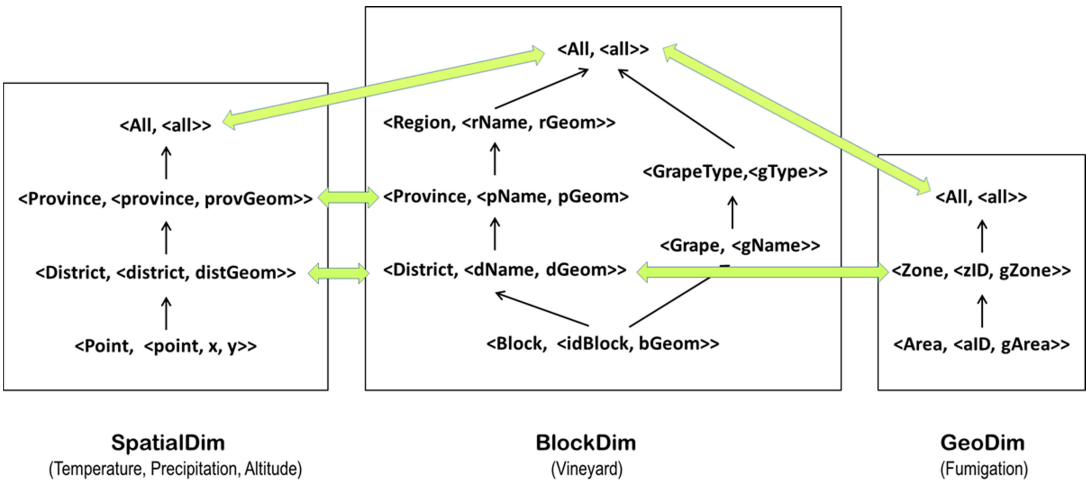


Figure 6.16: Semantic mapping between spatial levels.

It is important to point out that the equality of names of two level descriptors

is not enough to define the equivalence between their members. That is the case of the descriptor pName in level Province of the BlockDim dimension, and pName in level Pesticide of the PestDim dimension. The semantic mapping must be identified by a curator with a full knowledge of the user domain.

The member mappings between TimeDim.Month and DateDim.Month is given by the homonymous level descriptors, i.e., month of TimeDim.Month with month of DateDim.Month. The same occurs with TimeDim.Year and DateDim.Year. Below, the rest of the member mappings are defined by intension:

$\text{SMAP}^{\{\text{TemporalDim.Month}\}}_{\{\text{TimeDim.Month}\}} = \{(\langle\text{month}\rangle, \langle\text{m}\rangle)|\text{month=m}\}$

$\text{SMAP}^{\{\text{TemporalDim.Year}\}}_{\{\text{TimeDim.Year}\}} = \{(\langle\text{year}\rangle, \langle\text{y}\rangle)|\text{year=y}\}$

$\text{SMAP}^{\{\text{SpatialDim.District}\}}_{\{\text{BlockDim.District}\}} = \{(\langle\text{dName, dGeom}\rangle, \langle\text{district,distGeom}\rangle)|\text{dName=district}\}$

$\text{SMAP}^{\{\text{SpatialDim.Province}\}}_{\{\text{BlockDim.Province}\}} = \{(\langle\text{pName, pGeom}\rangle, \langle\text{province,provGeom}\rangle)|$
$\text{pName=province}\}$

$\text{SMAP}^{\{\text{GeoDim.Zone}\}}_{\{\text{BlockDim.District}\}} = \{(\langle\text{dName, dGeom}\rangle, \langle\text{zID,gZone}\rangle)|\text{dGeom=gZone}\}$.

Notice that the member mapping may not involve all level descriptors, since some of them can be irrelevant to define the equivalence, e.g., for $\text{SMAP}^{\{\text{SpatialDim.Province}\}}_{\{\text{BlockDim.Province}\}}$, the level descriptors pGeom and provGeom are not taken into account for the member equivalence.

Once the semantic mapping is defined, we can formulate queries involving several cubes.

Query 4

"*Districts sprayed with chemical substances at a height lower than 12 meters, and where the temperatures during January 2010 were greater than 2.0 Celsius degrees*". This can be solved as follows.

$\text{Q4-Vin1} = \text{ROLL-UP}(\text{Vineyard}, \text{BlockDim}, \text{District})$

$\text{Q4-Vin2} = \text{ROLL-UP}(\text{Q4-Vin1}, \text{TimeDim}, \text{Month})$

$\text{Q4-Vin3} = \text{DICE}(\text{Q4-Vin2}, \text{month} = \text{'Jan-2010'})$

The cuboid Q4-Vin3 selects the month and year after rolling-up to the corresponding levels in Vineyard. The resulting cuboid with the harvest per district during January 2010 is shown in Figure 6.17. Let us call $\mathsf{Vineyard}_{\mathsf{Q4-Vin3}}$ the cube induced by cuboid Q4-Vin3. According to the rules of Section 5.5, the following mappings are induced: SM($\mathsf{Vineyard}_{\mathsf{Q4-Vin3}}$, Temperature), SM($\mathsf{Vineyard}_{\mathsf{Q4-Vin3}}$, Precipitation), SM($\mathsf{Vineyard}_{\mathsf{Q4-Vin3}}$, Altitude), SM($\mathsf{Vineyard}_{\mathsf{Q4-Vin3}}$, Fumigation).

Q4-Tem1 = ROLL-UP(Temperature, SpatialDim, District)
Q4-Tem2 = ROLL-UP(Q4-Tem1, TemporalDim, Month)
Q4-Tem3 = DICE(Q4-Tem2, temperature $>$ 2.0 and m = 'Jan-2010'

The cuboid Q4-Tem3 selects the month January 2010 in Temperature after rolling-up to the Month and District levels. The resulting cuboid with temperatures higher than 2.0°C by district during January 2010 is depicted in Figure6.18. Let us call $\mathsf{Temperature}_{\mathsf{Q4-Tem3}}$ the cube induced by Q4-Tem3. New semantic mappings are induced, like SM($\mathsf{Vineyard}_{\mathsf{Q4-Vin3}}$, $\mathsf{Temperature}_{\mathsf{Q4-Tem3}}$) among other ones.

Q4-Fug1 = ROLL-UP(Fumigation, PestDim, Type)
Q4-Fug2 = ROLL-UP(Q4-Fug1, GeoDim, Zone)
Q4-Fug3 = DICE(Q4-Fug2, month = 'Jan-2010' and pType = 'chemical' and ZMIN(trajectory) <= 12) (ZMIN returns the minimum height of a 3D geometry)
Q4-Fug4 = SLICE(Q4-Fug3, PestDim)

The cuboid Q4-Fug4 contains the aggregated trajectories during January 2010 only for the chemical pesticide category, after rolling-up to Month, Type, and Zone levels. The resulting cuboid is shown in Figure 6.19.

The output cuboid Q4-Fug3 induces the cube $\mathsf{Fumigation}_{\mathsf{Q4-Fug3}}$. According to the rules of Section 5.5, a semantic mapping SM($\mathsf{Vineyard}_{\mathsf{Q4-Vin3}}$, $\mathsf{Fumigation}_{\mathsf{Q4-Fug3}}$) is also induced. However, no cuboid of $\mathsf{Vineyard}_{\mathsf{Q4-Vin3}}$ is semantically compatible with any cuboid of $\mathsf{Fumigation}_{\mathsf{Q4-Fug3}}$, because of the PestDim dimension. Thus,

TimeDim Month month	BlockDim District dName, dGeom	Measures (Sum) harvest
⟨'Jan-2010'⟩	⟨'Nijvel', g118⟩	29200
⟨'Jan-2010'⟩	⟨'Hasselt', g127⟩	39400
⟨'Jan-2010'⟩	⟨'Antwerp', g100⟩	25400
⟨'Jan-2010'⟩	⟨'Louvain', g132⟩	31400
⟨'Jan-2010'⟩	⟨'Arlon', g101⟩	32100
⟨'Jan-2010'⟩	⟨'Oostende', g135⟩	22600
...	...	...
⟨'Jan-2010'⟩	⟨'Mechelen', g120⟩	35400

Figure 6.17: Intermediate cuboid Q4-Vin3.

TemporalDim Month m	SpaceDim District district, distGeom	Measures (Avg) temperature
⟨'Jan-2010'⟩	⟨'Antwerp', poly200⟩	2.3
⟨'Jan-2010'⟩	⟨'Hasselt', poly123⟩	2.1
⟨'Jan-2010'⟩	⟨'Louvain', poly132⟩	2.2
⟨'Jan-2010'⟩	⟨'Mechelen', poly207⟩	2.3
...	...	...
⟨'Jan-2010'⟩	⟨'Oostende', poly315⟩	2.9

Figure 6.18: Intermediate cuboid Q4-Temp3.

the last SLICE operation induces a new cube $\mathsf{Fumigation}_{\mathsf{Q4-Fug4}}$ and the semantic mapping $\mathsf{SM}(\mathsf{Vineyard}_{\mathsf{Q4-Vin3}}, \mathsf{Fumigation}_{\mathsf{Q4-Fug4}})$, with six semantically compatible cuboids, like Q4-Vin3 and Q4-Fug4.

DateDim Month month	GeoDim Zone zID, gZone	PestDim Type pType	Measures (Union) trajectory
⟨Jan-2010⟩	⟨01, geom01⟩	⟨chemical⟩	t104 ∪ t118
⟨Jan-2010⟩	⟨07, geom07⟩	⟨chemical⟩	t107 ∪ t132
⟨Jan-2010⟩	⟨11, geom11⟩	⟨chemical⟩	t162
⟨Jan-2010⟩	⟨14, geom280⟩	⟨'pyrethrin'⟩	t205 ∪ t262 ∪ t301

Figure 6.19: Intermediate cuboid Q4-Fug4.

We can now compute the DRILL-ACROSS between the semantically compatible cuboids Q4-Vin3 and Q4-Tem3:

Q4-Aux1 = DRILL-ACROSS(Q4-Vin3, Q4-Tem3)

Q4 = DRILL-ACROSS(Q4-Aux1, Q4-Fug4)

The result is the cuboid Q4-Aux1, containing measures harvest and temperature,

and it is shown in Figure 6.20. Notice that the districts of 'Nijvel' and 'Arlon' are not present in Q4-Aux1, since they have no semantically equivalent members in the cuboid Q4-Tem3. The schema of the output cube $\mathsf{Vineyard}_{\mathsf{Q4-Aux1}}$ neither contains the levels Block, Grape and GrapeType in the BlockDim dimension, nor the level Day in the dimension TimeDim because they are below the Month and District levels, which are the current levels of the input cuboids.

Q4-Aux1 induces new semantic mappings, in particular $\mathsf{SM}(\mathsf{Vineyard}_{\mathsf{Q4-Aux1}}, \mathsf{Fumigation}_{\mathsf{Q4-Fug4}})$. Because of this mapping, the cuboids Q4-Aux1 and Q4-Fug4 are semantically compatible and can be drilled-across, so the desired result can be computed. This result contains three measures, namely harvest, temperature and trajectory. Figure 6.21 depicts the final cuboid.

TimeDim	BlockDim	Measures	
Month	District	(Sum)	(Avg)
month	dName, dGeom	harvest	temperature
⟨'Jan-2010'⟩	⟨'Hasselt', g127⟩	39400	2.1
⟨'Jan-2010'⟩	⟨'Antwerp', g100⟩	25400	2.3
⟨'Jan-2010'⟩	⟨'Louvain', g132⟩	31400	2.2
⟨'Jan-2010'⟩	⟨'Oostende', g135⟩	22600	2.9
...	...	...	...
⟨'Jan-2010'⟩	⟨'Mechelen', g120⟩	35400	2.3

Figure 6.20: Intermediate cuboid Q4-Aux1.

TimeDim	BlockDim	Measures		
Day	Block	(Sum)	(Avg)	(Union)
date,dayofWeek	idBlock, bGeom	harvest	temperature	trajectory
⟨'Jan-2010'⟩	⟨'Hasselt', g127⟩	39400	2.1	t205 ∪ t262 ∪ t301
⟨'Jan-2010'⟩	⟨'Mechelen', g120⟩	35400	2.3	t107 ∪ t132

Figure 6.21: Resulting cuboid for Query 4.

Query 5

"*Ratio between harvest and total area of provinces during 2010 for 'pinot noir' in provinces where the maximum altitude is greater than 200 meters and the average precipitation is greater than 95 mm*". The following groups of OLAP expressions solve the query.

Q5-Vin1 = ROLL-UP(Vineyard, BlockDim, Grape)
Q5-Vin2 = ROLL-UP(Q5-Vin1, TimeDim, Year)
Q5-Vin3 = DICE(Q5-Vin2, year = 2010 and gName = 'pinot noir')
Q5-Vin4 = DRILL-DOWN(Q5-Vin3, BlockDim, Block)
Q5-Vin5 = ROLL-UP(Q5-Vin4, BlockDim, Province)

The cuboid Q5-Vin3 selects the desired grape and year, after rolling-up to the corresponding levels. Then, a DRILL-DOWN and a new ROLL-UP result in a cuboid with the provinces that correspond to the previously selected members, shown in Figure 6.22.

Q5-Alt1 = ROLL-UP(Altitude, SpatialDim, Province)
Q5-Alt2 = DICE(Q5-Alt1, altitude >= 200)

The cuboid Q5-Alt2 contains the desired altitude per province, after rolling-up to the corresponding level. This cuboid is shown in Figure 6.23.

Q5-Pre1 = ROLL-UP(Precipitation, SpatialDim, Province)
Q5-Pre2 = ROLL-UP(Q5-Pre1, TemporalDim, Year)
Q5-Pre3 = DICE(Q5-Pre2, y = 2010 and precipitation >= 95)

Q5-Pre3 contains the desired precipitation per province during year 2010, after rolling-up to the corresponding levels. Figure 6.24 depicts this intermediate cuboid.

Q5-Aux1 = DRILL-ACROSS(Q5-Vin5, Q5-Alt2)
Q5 = DRILL-ACROSS(Q5-Aux1, Q5-Pre3, {FR, harvest-per-area, SUM})
where $\text{FR} : \mathbb{R} \times \mathbb{P}_{2D} \rightarrow \mathbb{R}$, with FR(harvest, pGeom) = harvest/AREA(pGeom).

In order to cross the information between the three intermediate cuboids above, two DRILL-ACROSS operations are applied. The first one between Q5-Vin5 and Q5-Alt2, and the second one between this partial result and Q5-Pre3. The latter includes the creation of a new measure named 'harvest-per-area' (see Example 30

TimeDim Year year	BlockDim Province pName, pGeom	Measures (Sum) harvest
⟨2010⟩	⟨'East Flanders', g10⟩	89200
⟨2010⟩	⟨'Flemish Brabant', g12⟩	79400
⟨2010⟩	⟨'Hainaut', g11⟩	85400
⟨2010⟩	⟨'Liege', g14⟩	78400
⟨2010⟩	⟨'Limburg', g15⟩	82100
⟨2010⟩	⟨'Luxembourg', g13⟩	92600
⟨2010⟩	⟨'Namur', g17⟩	81400
⟨2010⟩	⟨'West Flanders', g19⟩	92600

Figure 6.22: Intermediate cuboid Q5-Vin5.

SpatialDim Province province, provGeom, population	Measures (Max) altitude
⟨'Liege', poly34, 1067685⟩	385
⟨'Luxembourg', poly43, 269023⟩	630
⟨'Namur', poly27, 472281⟩	280

Figure 6.23: Intermediate cuboid Q5-Alt2.

TemporalDim Year y	SpatialDim Province province, provGeom, population	Measures (Avg) precipitation
⟨2010⟩	⟨'Liege', poly34, 1067685⟩	107
⟨2010⟩	⟨'Luxembourg', poly43, 269023⟩	99

Figure 6.24: Intermediate cuboid Q5-Pre3.

about including new measures in a DRILL.ACROSS operation). Figures 6.25 and 6.26 shown these last resulting cuboids. The cuboid Q5 contains the result.

TimeDim Year year	BlockDim Province pName, pGeom	Measures (Sum) harvest	 (Max) altitude
⟨2010⟩	⟨'Liege', g14⟩	78400	385
⟨2010⟩	⟨'Luxembourg', g13⟩	92600	630
⟨2010⟩	⟨'Namur', g17⟩	81400	280

Figure 6.25: Intermediate cuboid Q5-Aux1.

To display only the new metric introduced, we must apply three SLICE operations over the measures harvest, altitude and precipitation as follows, and the final resulting cuboid is shown in Figure 6.27:

TimeDim	BlockDim	Measures			
Year	Province	(Sum)	(Max)	(Avg)	(Sum)
year	pName, pGeom	harvest	altitude	precipitation	harvest-per-area
⟨2010⟩	⟨'Liege', g14⟩	78400	385	107	20.4
⟨2010⟩	⟨'Luxembourg', g13⟩	92600	630	99	20.8

Figure 6.26: Resulting cuboid for Query 5.

Q5-Aux2 = SLICE(Q5, harvest)

Q5-Aux3 = SLICE(Q5-Aux2, altitude)

Q5short = SLICE(Q5-Aux3, precipitation)

TimeDim	BlockDim	Measures
Year	Province	(Sum)
year	pName, pGeom	harvest-per-area
⟨2010⟩	⟨'Liege', g14⟩	20.4
⟨2010⟩	⟨'Luxembourg', g13⟩	20.8

Figure 6.27: Resulting cuboid for Query 5 with only the new measure.

6.4 Continuous Data and Cubes

At first sight, the reader may think that the five cubes presented in this section are typical OLAP cubes containing alphanumerical information, or SOLAP cubes because of the geographic dimensions or measures. However, actually Temperature, Precipitation and Altitude had been obtained by discretizing information from continuous sources. The difference between these discretized cubes and the typical SOLAP cubes lies in the possibility of asking for values that are not present in the stored data but which can be inferred for example, using interpolation methods. That is the case of Query 1. In a traditional SOLAP cube, if the value corresponding to the coordinate (Long $4.37°$, Lat $50.84°$) is not exactly stored in the cube instance, it will never appear in any resulting cuboid. On the contrary, if the cube represents a discretized continuous function, although the data are not stored in the cube, it can be calculated based on the interpolation of other values actually stored in the cube.

In summary, the five queries above have given a comprehensive idea of the power of our proposal: we have shown that these complex queries have been addressed being unaware of the kinds of data represented in the five cubes, which are actually of completely different kinds: Temperature and Precipitation actually come from spatiotemporal continuous data, Altitude comes from spatial continuous data, Fumigation represents moving object data, and Vineyard is a standard SOLAP cube. However, we were able to express the five queries without caring about these details: we have just manipulated cubes. We provide an in-depth discussion in Chapters 7 and 8.

6.5 Summary

In this chapter we have shown how the concepts studied in the previous ones can be applied to solve complex queries over heterogeneous data cubes. We also illustrated the power provided by the extension of the Drill-across operator. Without such a mapping, these queries could not have been solved. Through these examples we have shown that the algebra proposed over our conceptual model can solve complex queries just using the basic operators, hiding from the user implementation details such as whether the underlying nature of a cube is discrete or continuous. In the next chapter we apply these concepts to the particular case of continuous field data, showing how the operations are defined at lower abstraction levels.

Chapter 7

Continuous Fields: Algebra and Data Model

In Chapter 6 we have suggested that in some cases it is possible to treat continuous information as an OLAP cube, in order to seamlessly combine this information with other discrete OLAP cubes, applying the typical OLAP operations at a conceptual level. In particular, in this thesis we will study how we can represent continuous field data as OLAP cubes, although we remark that the procedure could be applied to any kind of data, provided these data fulfill some conditions.

In this chapter we present a model that allows transforming continuous fields into discrete spatial and spatiotemporal data structures that can then be represented as OLAP cubes (this representation will be explained in detail in Chapter 8). Continuous fields are typically queried using a language denoted Map Algebra, which is only applicable to raster data. Therefore we also extend Map Algebra to support any kind of field representation (raster, Voronoi, TIN, etc.). In this way we will be able to show that not only we can treat continuous fields as cubes, but also to support different and even mixed kinds of representations of such fields.

7.1 Continuous Fields

A Continuous Field (from now, Field) can be expressed as a function that describes the variation of the values of a phenomena/feature (for example, real numbers for precipitation or temperature, strings for countries) at every point of

a continuous field domain. Thus, this function has a continuous N-dimensional domain (spatial and/or temporal) and its range is included in the values of the represented phenomena/feature [49]. The formal definition of Field is given below.

Definition 20 (Continuous Field)**.** A Continuous Field is composed of:

(a) A domain D which is a continuous set

(b) A range of values R

(c) A mapping function Fn: $\mathsf{D} \rightarrow \mathsf{R}$

□

Example 31. The Gravitational field around a mass M located at $(\mathsf{x}_0, \mathsf{y}_0, \mathsf{z}_0)$ has its domain in $\mathbb{R}^3$, its range of values in $\mathbb{R}^3$ (it is a vectorial field), and its function is defined point by point by the following expression (G is the universal gravitational constant): Fn$(\mathsf{x}, \mathsf{y}, \mathsf{z})$= $(\mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3)$ where

$$\mathsf{a}_1 = -\mathsf{x} * \frac{\mathsf{G} * \mathsf{M}}{\sqrt{((\mathsf{x} - \mathsf{x}_0)^2 + (\mathsf{y} - \mathsf{y}_0)^2 + (\mathsf{z} - \mathsf{z}_0)^2)^3}}$$

$$\mathsf{a}_2 = -\mathsf{y} * \frac{\mathsf{G} * \mathsf{M}}{\sqrt{((\mathsf{x} - \mathsf{x}_0)^2 + (\mathsf{y} - \mathsf{y}_0)^2 + (\mathsf{z} - \mathsf{z}_0)^2)^3}}$$

$$\mathsf{a}_3 = -\mathsf{z} * \frac{\mathsf{G} * \mathsf{M}}{\sqrt{((\mathsf{x} - \mathsf{x}_0)^2 + (\mathsf{y} - \mathsf{y}_0)^2 + (\mathsf{z} - \mathsf{z}_0)^2)^3}}$$

Notice that this field contains a vector in each point of its domain. □

In practice, operations are performed over discrete representations that must satisfy certain constraints, and over a bounded domain. Moreover, the values of the function are restricted to some specific area of interest, and outside this area the function may be unknown. This leads to the notion of Bounded Field.

Definition 21 (Bounded Continuous Field)**.** An N-dimensional Bounded Continuous Field (BField) F is defined by:

(a) A domain $\mathsf{Dom}(\mathsf{F}) = \mathsf{Dom_1}(\mathsf{F}) \times \cdots \times \mathsf{Dom_N}(\mathsf{F})$, where $\mathsf{Dom_i}$ is a closed interval in $\mathbb{R}$, $\forall i, i = 1..n$

(b) A list of N labels $\mathsf{Labels(F)} = \langle \mathsf{l}_1, \ldots, \mathsf{l}_\mathsf{N} \rangle$, to describe semantically each dimension in the domain, where $\mathsf{l}_i \neq \mathsf{l}_j,\ \forall i, i = 1 \ldots \mathsf{N},\ \forall j, j = 1 \ldots \mathsf{N}$. (We denote $\mathsf{L}_i(\mathsf{F})$ the i^{th} component of $\mathsf{Labels(F)}$)

(c) A set of values $\mathsf{R} \cup \{\perp\}$, denoted $\mathsf{Range(F)}$, where $\perp$ is a distinguished value that does not belong to R

(d) A mapping function $\textsc{Fn} : \mathsf{Dom(F)} \rightarrow \mathsf{Range(F)}$ □

Remark 11. *Notice that '$\perp$' can be represented in different ways. For an application that uses databases we can use the* null *value. For an application that uses a file this value can be* NaN *in the case of numbers. The range of values for a field can be constants (such as numbers, categorical names) or vectors (for defining its magnitude and direction).* □

Example 32 (Bounded Continuous Field)**.** We define the BField Elevation as a 2-D domain limited by a square of 6 km × 6 km, Labels(Elevation)=<X,Y> and Range(Elevation)={1, 2, 3, 4, $\perp$}.

The mapping function is defined point by point in the spatial domain as it is graphically depicted in Figure 7.1. It is important to point out that the continuous nature of Fields guarantees that we can obtain a value of Range(Elevation) for every point in the domain. □

When a BField corresponds to a phenomenon, it is usually measured only at some finite number of points of its N-dimensional domain. Thus, in practice, the values of the function are restricted to some specific area of interest, and only a finite set of sample points are known (also, we actually do not know the value of a phenomenon, but only an approximation with its respective error range). To estimate the values at non-sampled points, different interpolation functions can be used. Based on this, we model a continuous field as a discrete field that contains only sampled values that have been measured in a finite set of points. Thus, we define a Discretized Field as follows.

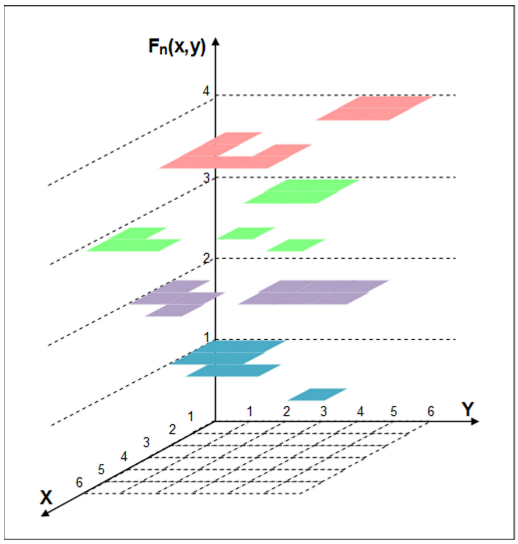


Figure 7.1: Graphical Representation of the BField Elevation

Definition 22 (Discretized Field)**.** An N-dimensional Discretized Field (DField) F is a BField (see Definition 21) with:

(a) A non-empty set of K tuples of N+1 dimensionality, Samples(F) = $\{(x_{11}, \ldots, x_{1N}, v_1), \ldots, (x_{K1}, \ldots, x_{KN}, v_K)\}$, where $\forall i, \forall j, i = 1..K, j = 1..N, x_{ij} \in \text{Dom}_j(F)$, and $v_i \in$ Range(F)

(b) An interpolation function F_S over Samples(F), used for defining F_N (Definition 21) □

Definition 23 (Sampled Tuple, Sampled Point, Sampled Value)**.** Given an N-dimensional DField F, we denote:

- *Sampled tuple* each element in Samples(F).
- *Sampled point* the first n components of a sampled tuple. We denote sp(s) the sampled point of the sample tuple s ∈ Samples(F).
- *Sampled value* the last component of a sampled tuple. We denote sv(s) the sampled point of the sample tuple s ∈ Samples(F). □

7.2 Tesselations

Since DFields only contain a finite set of sampled values, we need to be able to infer the value of the function at each point in space. To do this, the space is normally partitioned using several possible methods we discuss next.

Computationally, an N-dimensional *continuous* spatial domain can be *partitioned* or *discretized* by grouping points into N-dimensional polytopes (e.g., polygons in 2D, polyhedra in 3D, polytopes in high order dimensions). The partition of the spatial domain is denoted a *tessellation*. This concept is used for representing infinite points in the continuous domain in a finite way. Formally:

Definition 24 (Tessellation)**.** A *Tessellation* of a finite N-dimensional Euclidean space S, denoted $\mathcal{Ts}(\mathsf{S})$, is a non-empty set $\mathcal{Ts}(\mathsf{S})=\{\mathsf{g}_1, \mathsf{g}_2, \ldots, \mathsf{g}_n\}$, such that the following conditions hold:

1. g_i is a N-dimensional polytope, $\forall i, i = 1..n$

2. $\mathsf{S} = \bigcup_{i=1..n} \mathsf{g}_i$

3. $\forall i, i = 1..n \; \forall j, j = 1..n \;\; \mathsf{g}_i \bigcap \mathsf{g}_j = \phi.$ □

There are regular and irregular tessellations. A regular tessellation can be defined with the Schläfli symbol, which consists of the form {p,q,r,...}. In two dimensions, {p} is used to describe a regular tessellation. For example, {3} for an equilateral triangle, {4} for a square. In three dimensions, {p, q} is used to describe a regular tessellation with q regular p faces around each vertex. For example, {4, 3} for a cube (i.e., three squares around each vertex). Analogously, in four dimensions, {p, q, r} is used.

In practice, different tessellations are used to represent fields. For example, raster data, TIN (Triangulated Irregular Network) and Voronoi among other ones. Moreover, different interpolation functions are used to define Fn over non-sampled points. In the following subsections we discuss some of them.

7.2.1 Voronoi Tessellation

This section introduces the use of Voronoi diagram to define a domain tessellation. For an in-depth discussion on how to build Voronoi diagrams, we refer the reader to the work by Ledoux and Gold [29].

Given a finite set of sampled points $\mathcal{Ps}$, each $\mathsf{sp} \in \mathcal{Ps}$ has an associated Voronoi cell consisting of all the points of the space closer to sp than to any other sample. As a consequence, the segments in a Voronoi diagram are composed by all the points equidistant to the two nearest samples and the nodes are the points equidistant to three or more samples (see Figure 7.2(a)).

The use of a Voronoi tessellation guarantees that the actual samples (which represent measured values) are preserved in the discrete representation, i.e., no sample is lost or modified, and only non-sampled points are inferred. To estimate the values of the function at non-sampled points, different interpolation functions can be used. For example the simplest one is the *Constant function* which assigns each point inside a Voronoi cell the value of the sampled value contained in it.

The algorithm to find out the value of a query point p, can be summarized as follows: if p is a sample, the algorithm returns its measured value. Otherwise. it computes the Voronoi cell corresponding to p, and returns the value of the sample of this cell. The constant function method is based on the idea that all points in a Voronoi cell are closer to the sample inside it than any other sample. Thus, the closer sample is enough for approximate its value, i.e. its nearest neighbor sample.

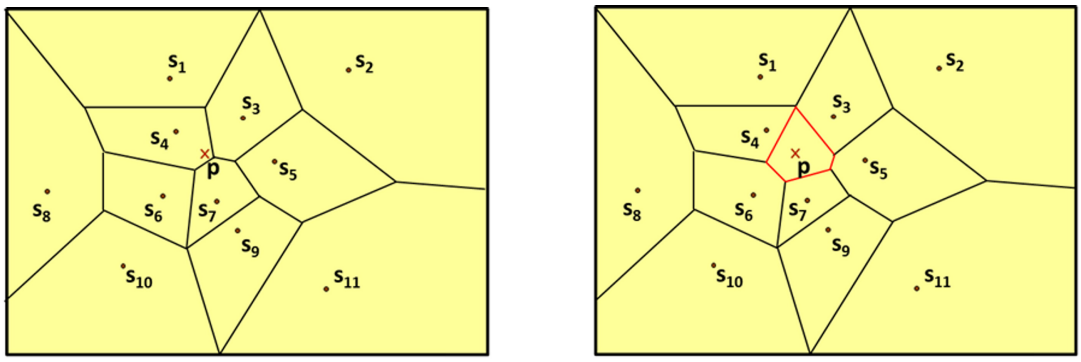
Other interpolation methods have been proposed. For example, the *Natural Neighbor Interpolation* ([47]) uses several samples. The value of these samples are weighted by a formula that estimates the unknown value of the query point. Algorithm 6 details how values are inferred with this method.

Example 33 (Voronoi tesselation)**.** Figure 7.2(a) shows a Voronoi tesselation built using sampled points. We need to estimate the value of a non-sampled point p. According to Algorithm 6, p is inserted and a new cell is induced, as shown in Figure 7.2(b). The estimated value of p is calculated by considering the weighted contribution of the parts of the old cells that build the new cell. A zoomed view of

the local cells that have been split when adding p is shown in Figure 7.2(c). The estimated value for p is calculated as:

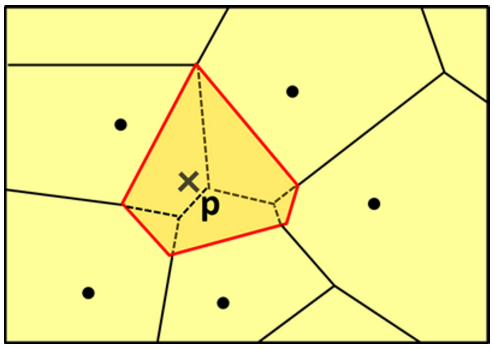
$$\mathsf{value(p)} = \frac{\sum_{i=3..7} \textsc{Area}(\mathsf{c_p} \cap \mathsf{c}_{si}) * \mathsf{sv}(\mathsf{s}_i)}{\textsc{Area}(\mathsf{c_p})}$$

where c_{si} is the cell corresponding to sample s_i and $\mathsf{c_p}$ is the new cell induced by the insertion of p. □



(a) Original Voronoi Tesselation

(b) New cell after inserting p



(c) Zoom of the new cell

Figure 7.2: Inferring the value for a point p using a Voronoi Tesselation.

7.2.2 Raster Tesselation

In GIS, raster data (gridded data) have been extensively used. A raster tesselation is a regular partition of a 2-D or 3-D domain. In practice, samples are used to assign each cell a single value, for example all sampled values corresponding to

Algorithm 6 Natural Neighbor Interpolation

CellsV$\leftarrow$ {c|c is a cell of the original Voronoi Tesselation}

newCell$\leftarrow$ new cell resulting when p point is inserted

S$\leftarrow$0

for all $\text{cell}_{s_i} \in$ CellsV **do**

  if $\text{cell}_{s_i} \bigcap$ newCell $\neq \phi$ **then**

    S $\leftarrow$ S + AREA($\text{cell}_{s_i} \cap$ newCell) $*$ sv(s_i)

val(p)$\leftarrow$S/AREA(newCell)

sampled points inside each cell are summarized into a single one. Thus, the discretized field uses a function that assigns a constant value to each point belonging to the same cell. It can be perceived as a matrix with cells of some extent, such that the tessellated domain is stored in metadata (cell size and extent) and the matrix stores inferred values.

Example 34 (Raster tesselation)**.** Figure 7.3 depicts a raster tessellation. Each cell contains a sampled value, representing the value of all the points within it. □

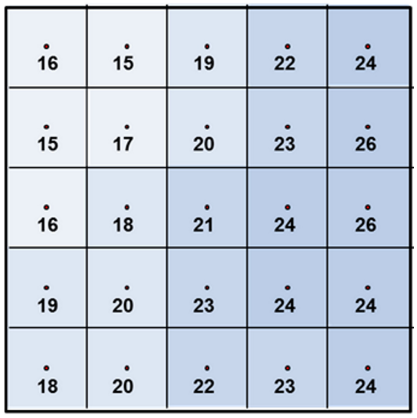


Figure 7.3: Example of Raster Tessellation

7.2.3 TIN (Triangulated Irregular Networks)

A Triangulated Irregular Network (TIN) is a data structure for the representation of 2-D surfaces. TINs are used in GIS to represent altitude, and they are derived from the digital elevation model (DEM). The TIN Model comprises a triangular network whose vertices (corresponding to sampled points), connected by edges, form a triangular irregular tessellation typically based on a Delaunay triangulation. In TIN, in regions where there is little variation in the surface height, the points may be widely spaced whereas in areas of more intense variation in height, the point density is increased [28]. Algorithms to build the Delaunay triangulation can be found in the work of Lee and Schachter [31].

The value z at each point of the plane XY may be calculated in different ways. *Linear interpolation* is the simplest method, and it may be achieved by using the plane equation of the surface given by the three vertices of the triangle containing the point to be interpolated. Given three points (x1, y1, z1), (x2, y2, z2) and (x3, y3, z3) a plane surface has a formula:

$$\begin{pmatrix} x & y & z & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{pmatrix}$$

Thus, the plane equation is:

$$\begin{aligned} z \;=\; & \frac{(x_3y_2z_1 + x_2y_1z_3 + x_1y_3z_2 - x_1y_2z_3 - x_2y_3z_1 - x_3y_1z_2)}{x_3y_2 + x_1y_3 + x_2y_1 - x_1y_2 - x_2y_3 - y_1x_3} + \\ & + \frac{(y_1z_2 + z_1y_2 + y_3z_3 - y_3z_2 - y_2z_1 - y_1z_3)x}{x_3y_2 + x_1y_3 + x_2y_1 - x_1y_2 - x_2y_3 - y_1x_3} + \\ & + \frac{(x_3z_2 + x_1y_3 + x_2z_1 - x_1z_2 - x_2z_3 - x_3z_1)y}{x_3y_2 + x_1y_3 + x_2y_1 - x_1y_2 - x_2y_3 - y_1x_3} \end{aligned}$$

Example 35 (TIN tesselation)**.** Consider a fragment of a TIN diagram, given in Figure 7.4, and the samples $s_1 = (20, 60, 10)$, $s_2 = (10, 20, 0)$ and $s_3 = (40, 25, 5)$.

The z value (altitude) of each point inside the triangle (s_1, s_2, s_3) using linear interpolation is given by:

$$z = \frac{3x + 5y - 130}{23}$$

Thus, the z value inferred for the point p=(20,40) is $130/23$. □

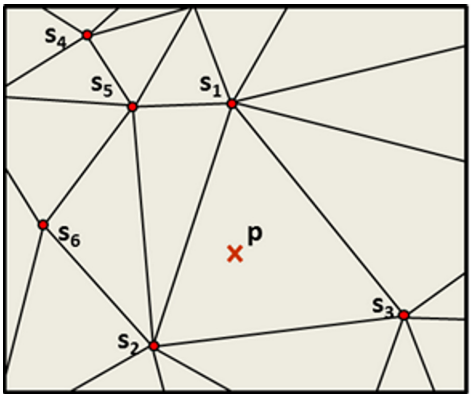


Figure 7.4: Example of TIN diagram

Remark 12. *Notice that, like Voronoi tesselations, a TIN tesselation preserves the sampled values. This is not necessary the case in a raster tesselation.*

7.3 Spatial and Spatio-Temporal DFields

Independently of how they are implemented, DFields can be classified into *spatial discretized* (SDField) and *spatio-temporal discretized* (STDField) ones. We next formalize the concepts of SDField and STDField, providing a conceptual model for fields.

Spatial DFields have just a spatial domain which is tessellated in a way such that each partition contains at least one sample.

Definition 25 (Spatial Discretized Field)**.** A Spatial Discretized Field (SDField) F is a DField (see Definition 22) such that:

(a) Its domain $\mathsf{Dom(F)}$ is spatial

(b) $\mathcal{T}\mathit{s}(\mathsf{Dom(F)}) = \{\mathsf{g}_1, \ldots, \mathsf{g}_\mathsf{P}\}$, and $\forall i, i = 1..\mathsf{P} \;\; \exists\; s \in \mathsf{Samples(F)}$ such that $\mathsf{sp(s)} \in \mathsf{g}_i$ where P is the number of cells of the tessellation $\mathcal{T}\mathit{s}(\mathsf{Dom(F)})$.

□

Borrowing ideas from Cubic Map Algebra [38], we model a *spatio-temporal DField* as a list of snapshots of SDFields across time, such that each snapshot is valid during a certain time interval. These time intervals induce a temporal partition of the time dimension. Snapshots can be of different kinds, meaning that the first one could be represented using a raster tessellation, and the next one could be a Voronoi partition. For all these snapshots only one interpolation function FS can be used.

Definition 26 (Spatio-Temporal DFields)**.** A Spatio-temporal Discretized Field (SDTField) F is a time-ordered sequence of N-dimensional SDFields, $\mathsf{Seq(F)}=\{\mathsf{F}_1, \mathsf{F}_2, \ldots \mathsf{F}_K\}$, such that:

(a) $\mathsf{Dom}(\mathsf{F}_i)$ is the same $\forall i, i = 1..K$, denoted $\mathsf{Dom}_s(\mathsf{F})$

(b) Each F_i is a snapshot of the field taken at time $\mathsf{t}_{\mathsf{F}i}$, and has an associated time interval $\mathsf{I}_{\mathsf{F}i} \subset \mathbb{R}$ representing its interval of validity, such that:

1. $\mathsf{Dom}_t(\mathsf{F}) = \bigcup_{i=1..K} \mathsf{I}_{\mathsf{F}i}$
2. $\mathsf{I}_{\mathsf{F}_i} = [\mathsf{s}_{\mathsf{F}_i}, \mathsf{e}_{\mathsf{F}_i})$ where $\mathsf{s}_{\mathsf{F}_i} = \mathsf{t}_{\mathsf{F}_i} - (\mathsf{t}_{\mathsf{F}_{i+1}} - \mathsf{t}_{\mathsf{F}_i})/2$ and $\mathsf{e}_{\mathsf{F}_i} = \mathsf{t}_{\mathsf{F}_i} + (\mathsf{t}_{\mathsf{F}_{i+1}} - \mathsf{t}_{\mathsf{F}_i})/2$, except $\mathsf{I}_{\mathsf{F}_K}$ which is closed (i.e., $[\mathsf{s}_{\mathsf{F}_k}, \mathsf{e}_{\mathsf{F}_k}]$). In consequence, $\forall i, i = 1..K$ $\forall j, j = 1..K$ $\mathsf{I}_{\mathsf{F}_i} \bigcap \mathsf{I}_{\mathsf{F}_j} = \phi$),

(c) $\mathsf{Dom(F)} = \mathsf{Dom}_s(\mathsf{F}) \times \mathsf{Dom}_t(\mathsf{F})$

(d) $\mathsf{Labels}(\mathsf{F}_i) = \langle \mathsf{l}_1, \ldots, \mathsf{l}_\mathsf{k} \rangle$ is the same $\forall i, i = 1 \ldots \mathsf{K}$. Then, $\mathsf{Labels(F)} = \langle \mathsf{l}_1, \ldots \mathsf{l}_\mathsf{n}, \mathsf{Time} \rangle$

(e) $\mathsf{Range}(\mathsf{F}_i)$ is the same $\forall i, i = 1..\mathsf{K}$, denoted $\mathsf{Range(F)}$

(f) $\mathsf{Samples(F)} = \{(\mathsf{sp(s)}, \mathsf{sv(s)}, \mathsf{t_{F_i}}) | \ \exists \mathsf{F}_i \in \mathsf{Seq(F)} \ \exists s \ \in \mathsf{Samples(F}_i)\}$ where $\mathsf{t_{F_i}} = \mathsf{s_{F_i}} + (\mathsf{e_{F_i}} - \mathsf{s_{F_i}})/2$ (i.e., each sample in F_i contains the middle point of the time interval $\mathsf{I_{F_i}}$ as its time value).

(g) Fs is the same $\forall i, i = 1..K$

(h) $\text{Fn}(x_1, \ldots x_n, t) = \text{Fn}_i(x_1, \ldots x_n)$ where Fn_i is the Fn corresponding to the snapshot F_i such that $t \ \in \ \mathsf{I_{F_i}}$ □

Example 36 (STDField)**.** Figure 7.5 shows a STDField with two snapshots. The intervals of validity of these snapshots, according with Definition 26, are such that the snapshot falls in the middle of the intervals. Then, $\mathsf{I}_{F_1} = [2,10)$ and $\mathsf{I}_{F_2} = [10,14]$, as can be seen in the figure. Also, Fs is the constant function. Fn(2.5, 3.5, 9) has the same value of Fn(2.5, 3.5) corresponding to the SDField of I_{F_1}, since time instant 9 belongs to the interval I_{F_1}. That means, during the whole interval we assume that the value of the function is the same for the same point. □

Example 37 (SDFields and STDFields)**.** The soil pH measures the acidity or basicity in soils. The Normalized Difference Vegetation Index (NDVI) is used to analyze the quality and development of vegetation based on remote sensing measurements of the radiation intensity emitted by plants. Figure 7.6 shows a Voronoi SDField pH, a raster SDField NDVI and an STDField Temperature, where the first two snapshots correspond to a Voronoi representation, and the other ones, to rasterized partitions. □

7.4 A Closed Generic Map Algebra for DFields

For manipulating Fields, several algebras have been proposed, like the map algebras presented in Chapter 2. However, these approaches cannot handle fields with different kinds of tessellations. That means, for instance, that for the *Local* function, all fields (input and output) must have the same kind of tessellation.

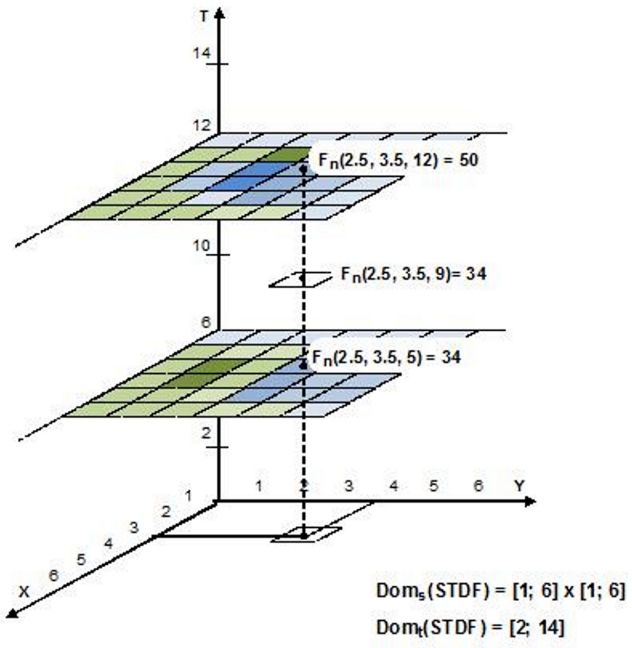
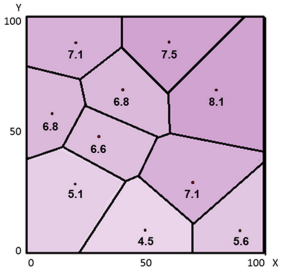
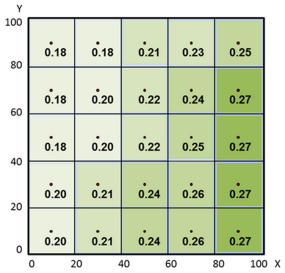


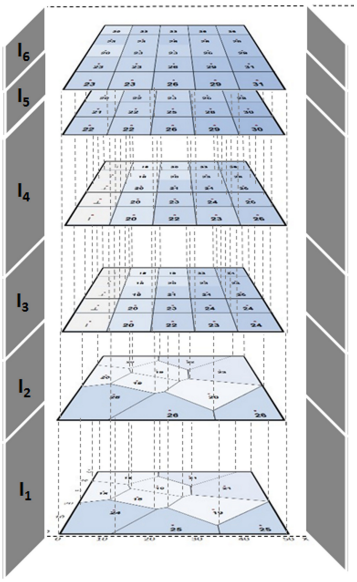
Figure 7.5: A Spatio-Temporal Discretized Field



(a) pH SDField



(b) NDVI SDField



(c) Temperature STDField

Figure 7.6: Examples of DFields.

In this section we propose an algebra that overcomes this limitation. Therefore, this algebra operates at a conceptual level through a set of meta-operators that are agnostic of the tesselations of the involved fields. In addition, we show that, opposite to map algebra, our algebra is closed (all of them receive fields and return fields), and therefore they can be nested in combined expressions. In short, we are providing an additional abstraction level. At a lower level, our definitions will allow to implement each operation according with the corresponding tesselation.

DFields may come from different sources, and their dimensions can be labeled differently from each other, even if they refer to the same semantic concept and even if they are represented in a same way (i.e., raster, Voronoi, etc.). In order to operate with them, they must be *domain compatible*, i.e., they have the same dimensionality (2D, 3D) and there must exist a semantic mapping (analogous to the one presented in Chapter 5) between the labels belonging to same domain. For example, let us consider two DFields F_1 and F_2 with $\mathsf{Labels}(\mathsf{F}_1) = \langle \mathsf{X}, \mathsf{Y} \rangle$, $\mathsf{Dom}(\mathsf{X})$=[10, 30], $\mathsf{Dom}(\mathsf{Y})$=[50, 120] and $\mathsf{Labels}(\mathsf{F}_2) = \langle \mathsf{coordY}, \mathsf{coordX} \rangle$, $\mathsf{Dom}(\mathsf{coordY})$= [50, 120], $\mathsf{Dom}(\mathsf{coorX})$=[10, 30]. F_1 and F_2 are domain compatible because we can define a semantic mapping of labels by matching X with coordX and Y with coordY.

Definition 27 (Semantic Mapping of Labels)**.** Consider a set of k DFields $\{\mathsf{F}_1, .., \mathsf{F}_k\}$ such that each F_i belongs to an N_i dimensional domain. Let $\mathsf{p} = \textsc{Min}(\mathsf{N}_i)$, $\forall\, i, i = 1..k$; A *semantic mapping* of Labels between the F_i's is a set of tuples of dimension k: $\mathsf{SML}(\mathsf{F}_1, \ldots, \mathsf{F}_k) = \{t | t = (\mathsf{l}_1, \mathsf{l}_2, \ldots, \mathsf{l}_k),\ \mathsf{l}_i \in \mathsf{Labels}(\mathsf{F}_i)\ \forall i, i = 1..k,$ and l_i represents the same concept$\}$. Moreover, (a) $1 \leq |\mathsf{SML}(\mathsf{F}_1, \ldots, \mathsf{F}_k)| \leq \mathsf{p}$; (b) $\forall \mathsf{t1}, \mathsf{t2} \in \mathsf{SML}(\mathsf{F}_1, \ldots, \mathsf{F}_k)$, $\mathsf{t1} = (\mathsf{l}_{11}, \mathsf{l}_{12}, \ldots, \mathsf{l}_{1k})$ and $\mathsf{t2} = (\mathsf{l}_{21}, \mathsf{l}_{22}, \ldots, \mathsf{l}_{2k})$ $\mathsf{l}_{1j} \neq l_{2j} \forall j, j = 1..k$. □

Definition 28 (Semantic Compatibility)**.** A set of DFields is *semantically compatible* iff there exists a Semantic Mapping of Labels between the fields in it, containing N tuples. □

Semantic compatibility does not imply that fields are defined over compatible

domains. In consequence, we introduce a stronger condition in order to guarantee a correct DField manipulation.

Definition 29 (Domain Compatibility)**.** Let $\mathsf{F} = \{\mathsf{F}_1, \mathsf{F}_2, ..., \mathsf{F}_k\}$ a set of k N-dimensional semantically compatible DFields. The DFields in F are *domain compatible* iff: $\forall(\mathsf{l}_{i1}, \mathsf{l}_{i2}, ...\mathsf{l}_{ik}) \in \mathsf{SML}(\mathsf{F}_1, \mathsf{F}_2, ..., \mathsf{F}_k), i = 1..N$, $\mathsf{Dom}_{\mathsf{l}_{i1}}(\mathsf{F}_1) = \mathsf{Dom}_{\mathsf{l}_{i2}}(\mathsf{F}_2) = ... = \mathsf{Dom}_{\mathsf{l}_{ik}}(\mathsf{F}_k)$, where $\mathsf{Dom}_{\mathsf{l}_{ij}}(\mathsf{F}_j)$ is the domain corresponding to the label l_{ij}, with $j = 1..\mathsf{k}$.

That means that the domains are the same for all the involved fields. □

Remark 13. *Note that if two N-dimensional DFields F_1 and F_2 are semantically compatible, they are domain-compatible in the intersection of their domains.* □

When two domain-compatible DFields F_1 and F_2 participate in some operation, it may be necessary to swap the coordinates of a point p belonging to $\mathsf{Dom}(\mathsf{F}_1)$ according with the semantic mapping of labels, when referring to an analogous point in $\mathsf{Dom}(\mathsf{F}_2)$. For this, we define the '*' notation, such that p^* is the point $\mathsf{p} \in \mathsf{F}_2$ with its coordinates swapped according to the order of the levels of F_1 (see Example 38) .

Example 38 (Star Notation)**.** F_1 and F_2 are DFields such that, $\mathsf{Labels}(\mathsf{F}_1) = \langle \mathsf{x}, \mathsf{y} \rangle$ and $\mathsf{Dom}(\mathsf{F}_1) = [1, 10] \times [3, 6]$; and $\mathsf{Labels}(\mathsf{F}_2) = \langle \mathsf{Y}, \mathsf{X} \rangle$ and $\mathsf{Dom}(\mathsf{F}_2) = [3, 6] \times [1, 10]$. Also, there exists $\mathsf{SML}(\mathsf{F}_1, \mathsf{F}_2) = \{(\mathsf{x}, \mathsf{X}), (\mathsf{y}, \mathsf{Y})\}$. Thus, for $\mathsf{p} = (3,5) \in \mathsf{Dom}(\mathsf{F}_1)$, the corresponding point in $\mathsf{Dom}(\mathsf{F}_2)$ is $\mathsf{p}^*=(5,3)$. Figure 7.7 illustrates this. □

As we already said, the traditional map algebra operators assume that the tessellations of all the fields are the same (i.e., all raster, all Voronoi, etc.). Since we want to allow different tessellations, we need to generalize map algebra, and redefine the traditional operators (i.e., local, focal and zonal). For this, we will make use of the notion of Discretized Field introduced above. For each operator we next provide the informal intuition, followed by the formal definition.

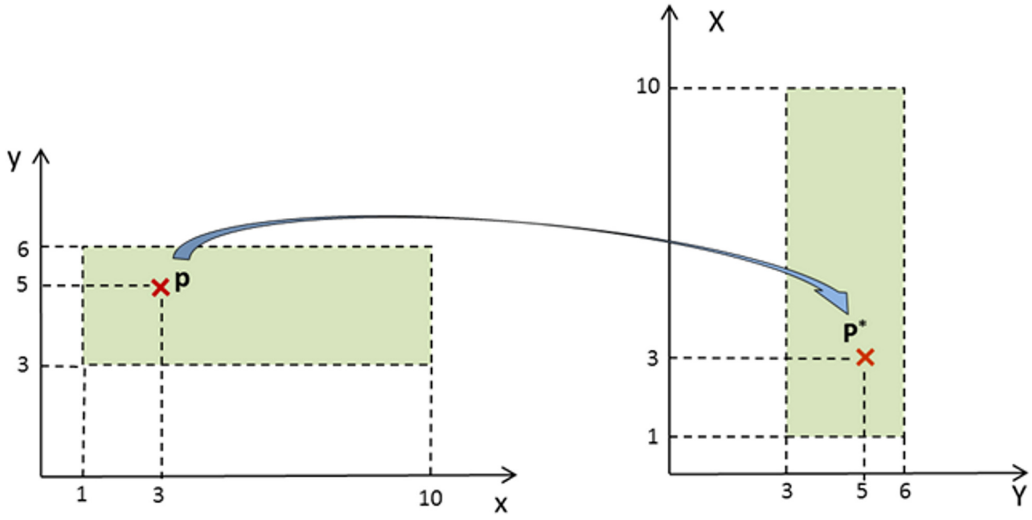


Figure 7.7: Star notation for swapped coordinates

7.4.1 Generic Local Operator

The *Generic Local Operator* receives a collection of domain-compatible DFields, and produces a new one whose values at each location p are computed applying an input function Fl to the values at the same location in all the input fields. In fact, this operator performs this computation for the samples of the first input DField and searches in each one of the rest of the input DFields the position of these points to infer their values before applying the Fl function. The choice of the first input field is arbitrary, it could have been any other one. In consequence, the output DField contains the same sampled points of the first input DField but with the new calculated values, and keeps the same interpolation function for future inferred calculus.

Definition 30 (Generic Local Operator)**.** Given a set of k domain-compatible DFields F_1, F_2, ..., F_k, and a function $\text{Fl} : \mathsf{Range}(\mathsf{F}_1) \times .. \times \mathsf{Range}(\mathsf{F}_k) \rightarrow \mathsf{R} \cup \{\perp\}$, $\text{Local}(\text{Fl}, \mathsf{F}_1, \ldots, \mathsf{F}_k)$ builds a new DField $\mathsf{F}_{\mathsf{out}}$ such that:

(a) $\mathsf{Dom}(\mathsf{F}_{\mathsf{out}}) = \mathsf{Dom}(\mathsf{F}_1)$ (by convention, actually, it could be any other one)

(b) $\mathsf{Labels}(\mathsf{F}_{\mathsf{out}}) = \mathsf{Labels}(\mathsf{F}_1)$

(c) $\mathcal{T}s(\mathsf{Dom}(\mathsf{F_{out}})) = \mathcal{T}s(\mathsf{Dom}(\mathsf{F_1}))$

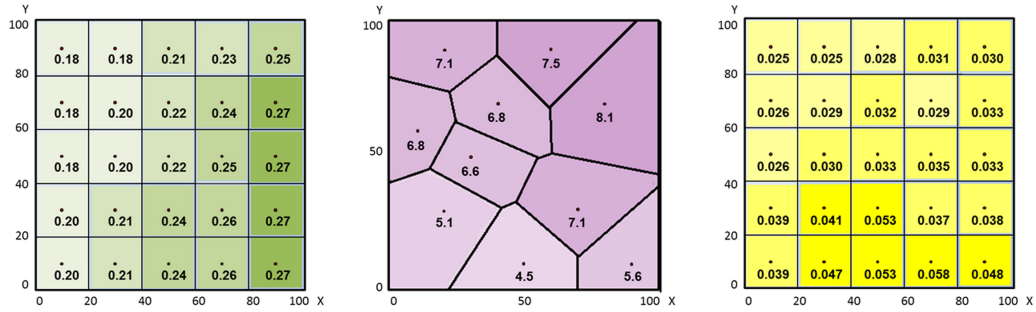
(d) $\mathsf{Samples}\ (\mathsf{F_{out}}) = \{\mathsf{s'} | \mathsf{s'} = \langle \mathsf{sp(s)}, \mathsf{v'} \rangle \wedge \mathsf{s} \in \mathsf{Samples}(\mathsf{F_1}) \wedge \mathsf{v'} = \textsc{Fl}(\textsc{Fn}_1(\mathsf{sp(s)}), \textsc{Fn}_2(\mathsf{sp(s)}^*), \ldots, \textsc{Fn}_k(\mathsf{sp(s)}^*))$ where $\textsc{Fn}_i$ is the function of $\mathsf{F}_i\ \forall i, i = 1..k\ \}$

(e) $\mathsf{Range}(\mathsf{F_{out}}) = \mathsf{Range}(\textsc{Fl})$

(f) Fs of $\mathsf{F_{out}}$ is Fs of F_1 □

Example 39 (Generic Local operator)**.** Consider the domain compatible SDFields NDVI (raster) and pH (Voronoi) of Example 37. We want to relate these two aspects by applying a function $\textsc{Fl} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ such that $\textsc{Fl}(\mathsf{ndvi},\mathsf{ph}) = \mathsf{ndvi}/\mathsf{ph}$.

Thus, we apply Local(Fl,NDVI,pH) and obtain the resulting SDField shown in Figure 7.8(c). Notice that, for example, the value 0.025 of the left upper sampled point is the quotient between the NDVI value 0.18 and the pH value 7.1. We chose the raster field as output, following the operator's definition. Also note that the definition of the operator does not mention the kind of tessellation, which is left to the implementation stage. □



(a) NDVI SDField (b) pH SDField (c) Local(Fl,NDVI,pH)

Figure 7.8: Generic Local operator.

7.4.2 Generic Focal Operator

The Focal operator aggregates values over a region. When an aggregate function (like Sum, Avg) is applied over a field, we must define the region of the

domain where the values will be aggregated. Given an N-dimensional DField F, an aggregation region, denoted AggReg, is an M-dimensional region with M≤N such that $\mathsf{AggReg} \subseteq \mathsf{Dom(F)}$. To approximate aggregates in this region, we need the following definition.

Definition 31 (Representative Points)**.** Given a DField F where $\mathcal{T}\mathsf{s}(\mathsf{Dom(F)}) = \{\mathsf{g}_1, \mathsf{g}_2, ...\mathsf{g}_n\}$, we associate a unique point to each geometry, and denote it a *representative point*, $\mathsf{rep}(\mathsf{g}_i)$, $\forall i, i = 1..n$. The value of the function at $\mathsf{rep}(\mathsf{g}_i)$ is called a *representative value* $\mathsf{vrep}(\mathsf{g}_i)$ □

Remark 14. *In a Voronoi Tessellation there is exactly one sample for each geometry, defined as the representative point of such geometry. Analogously in the raster model (although the representative point is not necessarily the original sample). In TIN, the centroid of the geometry could be used.* □

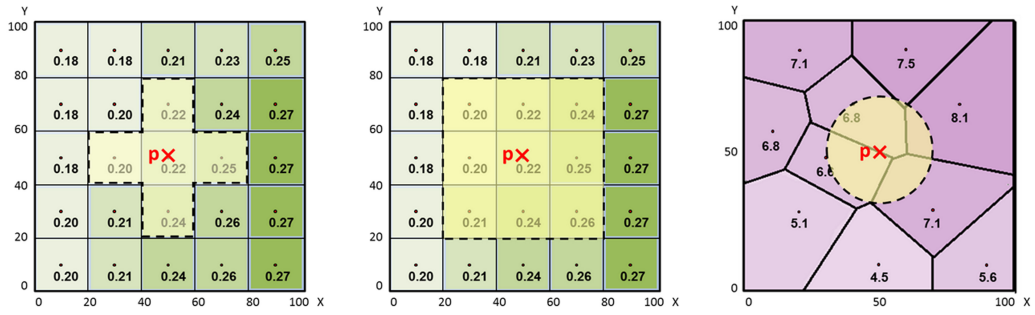
Definition 32 (Neighborhood)**.** Given a DField F, a point $\mathsf{p} \in \mathsf{Dom(F)}$, and a formula defining a criterion, denoted γ, we define the *neighborhood* of a point p according to γ, denoted $\mathsf{NbReg(p, \gamma)}$, the region around p that results from applying the criterion γ to p.

The criterion γ can be simply the well-known $\mathsf{5\text{-}cells}$ neighbors or $\mathsf{9\text{-}cells}$ neighbors, an expression (e.g., $(\mathsf{x} - \mathsf{p_x})^2 + (\mathsf{y} - \mathsf{p_y})^2 = \mathsf{r}^2$), or it can be defined algorithmically. □

Example 40 (Neighborhood)**.** Figures 7.9(a), 7.9(b) and 7.9(c), depict three options of Neighborhood for p, based on $\gamma = \mathsf{5\text{-}cells}$, $\gamma = \mathsf{9\text{-}cells}$, and $\gamma = (\mathsf{x} - \mathsf{p_x})^2 + (\mathsf{y} - \mathsf{p_y})^2 = \mathsf{r}^2$, respectively. □

The *Generic Focal Operator* receives a DField F and produces a new one such that at each location p its value is calculated aggregating the values of F in the neighborhood of p (according to a criterion γ) using an aggregate function. The new DField keeps the representation and labels of the input parameter.

Let us call $\mathsf{AggRegSet}$ a set of aggregation regions and $\mathsf{DFieldSet}$ a set of Discretized Fields.



(a) 5-cell (b) 9-cell (c) Circle with radius 2)

Figure 7.9: Example of Neighborhoods.

Definition 33 (Generic Focal Operator)**.** Given an N-dimensional DField F, a neighborhood criterion γ, and an aggregation function $\textsc{Fag} : \mathsf{AggRegionSet} \times \mathsf{DFieldSet} \rightarrow \mathsf{R}_n \cup \{\perp\}$, $\textsc{Focal}(\mathsf{F}, \gamma, \textsc{Fag})$ returns a DField $\mathsf{F}_{\mathsf{out}}$ where:

(a) $\mathsf{Dom}(\mathsf{F}_{\mathsf{out}}) = \mathsf{Dom}(\mathsf{F})$

(b) $\mathsf{Labels}(\mathsf{F}_{\mathsf{out}}) = \mathsf{Labels}(\mathsf{F})$

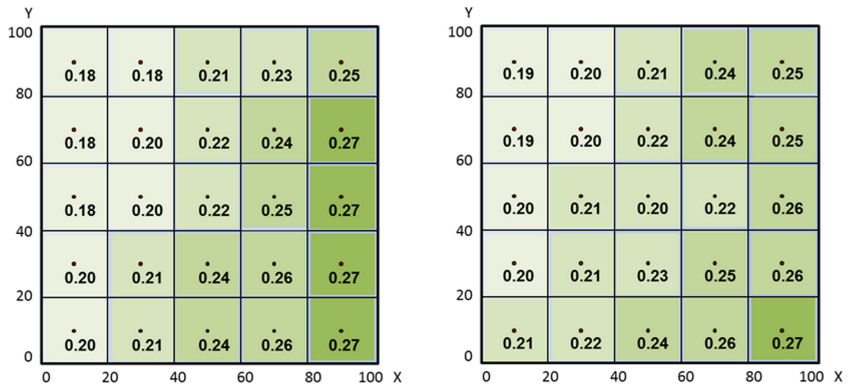
(c) $\mathcal{T}s(\mathsf{Dom}(\mathsf{F}_{\mathsf{out}})) = \mathcal{T}s(\mathsf{Dom}(\mathsf{F}))$

(d) $\mathsf{Samples}(_{\mathsf{out}}) = \{\mathsf{s'}|\mathsf{s'} = \langle \mathsf{sp(s)}, \mathsf{v'} \rangle \wedge s \in \mathsf{Samples(F)} \wedge \mathsf{v'} = \textsc{Fag}(\mathsf{NbReg(sp(s)}, \gamma), \mathsf{F})\}$

(e) $\mathsf{Range}(\mathsf{F}_{\mathsf{out}}) = \mathsf{R_n} \cup \{\perp\}$

(f) Fs of $\mathsf{F}_{\mathsf{out}}$ is Fs of F □

Example 41 (Generic Focal Operator)**.** Consider the SDField NDVI of Example 37. We smooth the sample values by applying a Focal operation over each sampled point with the 9-cell criterion and the Avg function. The resulting SDField is shown in 7.10(b). For example, the value 0.19 of the left upper sampled point is computed as the rounded average between its value 0.18 and the values 0.18, 0.18 and 0.2 because it has only three neighbors instead of 9 (because it is in a corner). □



(a) NDVI SDField (b) Focal(NDVI,9-cell,AVG)

Figure 7.10: Generic Focal operator.

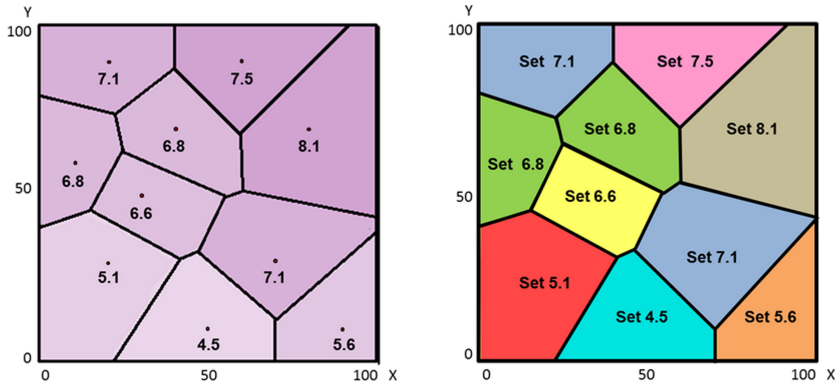
7.4.3 Generic Zonal Operator

Before giving a formal definition of this operator, we need the notion of *isopartition*, intuitively, the partition of the domain into sets of cells such that all the cells into a set have the same representative value.

Definition 34 (Isopartition of a DField)**.** Given a DField F, and the set of existing representative values $\mathsf{V} = \{\mathsf{r} | \mathsf{r} = \mathsf{vrep}(\mathsf{g_i}) \wedge \mathsf{g}_i \in \mathcal{T}s(\mathsf{Dom}(\mathsf{F}))\}$, an *isopartition* of F with respect to V, denoted as $\mathsf{IP}(\mathsf{F}, \mathsf{V})$ is the set $\{t \mid t = \langle \mathsf{v}, \{\mathsf{g}_1, \mathsf{g}_2, \ldots \mathsf{g}_k\}\rangle \wedge \mathsf{v} \in \mathsf{V} \wedge \forall i, i = 1..k, \mathsf{vrep}(\mathsf{g_i}) = \mathsf{v} \wedge (\nexists\ \mathsf{g} \text{ with } \mathsf{vrep}(\mathsf{g}) = \mathsf{v} \wedge \mathsf{g} \notin \{\mathsf{g}_1, \mathsf{g}_2, \ldots \mathsf{g}_k\})\}$. □

Example 42 (Isopartition)**.** Given the SDField pH of Example 37, the isopartition respect to its sample values is shown in Figure 7.11. Notice that the result contains as many sets as different representative values exist. Besides, notice that the set corresponding to 6.8 is a convex polygon, but the set corresponding to 7.1 is composed of two no contiguous cells, which is composed of all the partitions with the same value. □

The *Generic Zonal Operator* receives two domain-compatible DFields, F and Ref (a reference field), and an aggregate function FAG. First, it generates an isopartition of Ref. Then, it builds an output DField with the sampled points of



(a) pH SDField (b) IP(pH, $\text{Values}_{\text{pH}}$)

Figure 7.11: Example of Isopartition.

Ref and for each zone (set) of this isopartiton, it computes the sample values by aggregating the values of F that correspond to this zone. Formally:

Definition 35 (Generic Zonal Operator)**.** Let Ref and F two domain compatible N-dimensional DFields, and an aggregate function FAG : $\text{AggRegSet} \times \text{DFieldSet} \rightarrow \text{R}_\text{n} \cup \{\perp\}$, ZONAL(Ref, F, FAG) builds a new DField F_{out} such that:

(a) $\text{Dom}(\text{F}_{\text{out}}) = \text{Dom}(\text{Ref})$

(b) $\text{Labels}(\text{F}_{\text{out}}) = \text{Labels}(\text{Ref})$

(c) $\mathcal{T}s(\text{Dom}(\text{F}_{\text{out}})) = \mathcal{T}s(\text{Dom}(\text{Ref}))$

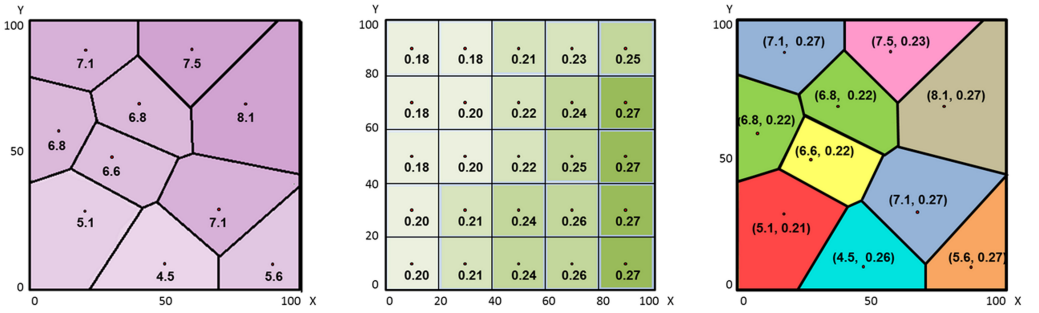
(d) $\text{Samples}(\text{F}_{\text{out}}) = \{\text{s'} | \text{s'} = \langle \text{sp(s)}, \text{v'} \rangle \wedge \text{s} \in \text{Samples(Ref)} \wedge \text{v'} = \text{FAG(Reg, F)} \wedge \exists \langle \text{sv(s)}, \{\text{g}_1, \ldots \text{g}_k\} \rangle \in \text{IP(Ref, V)} \wedge \text{V} = \{\text{vrep}(\text{g}_i) \mid g_i \in \mathcal{T}s(\text{Dom(Ref)})\} \wedge \text{Reg} = \cup_{i=1..k}\ \text{g}_i\}$

(e) $\text{Range}(\text{F}_{\text{out}}) = \text{R}_\text{n} \cup \{\perp\}$

(f) FS of F_{out} is FS of Ref □

Remark 15. *Note that, opposite to the case of traditional Map Algebra c[38], in our generic field algebra, the* ZONAL *operator returns a field, thus, the algebra is closed.* □

Example 43 (Generic Zonal Operator)**.** Consider the SDfields pH and NDVI of Example 37. We apply the ZONAL operator using pH as Ref field, and NDVI as the F field in the definition above, and the aggregate MAX function. The resulting SDField of ZONAL(pH, NDVI, MAX) is depicted in Figure 7.12(c). □



(a) pH SDField (b) NDVI SDField (c) ZONAL(pH, NDVI, MAX)

Figure 7.12: Generic Zonal Operator.

We conclude the chapter with an integrated example that illustrates the advantage of defining a closed algebra, since each resulting DField can be used as input to the next operator.

Example 44 (Integrated Example)**.** The Natural Resources Conservation Service (NRCS) classifies a pH level of a soil as 'acidic' for pH < 6.6, 'neutral' for pH between 6.6 and 7.3, and 'alkaline' for pH > 7.3. We want to generate a new DField with the average NDVI index for each zone defined by this pH classification, using the Voronoi SDField pH in Figure 7.6(a), and the raster SDField NDVI of Figure 7.6(b).

To compare a DField against a constant value we need to generate a constant DField (i.e., one such that all of its samples have the same value), and then apply a LOCAL operator. In our example, we first build the SDField Const (Figure 7.13(c)), with the lower and upper values of the pH limit classification, and then we apply LOCAL(FNRANGE, pH, Cte) where the function FNRANGE : $\mathbb{R} \times \mathbb{R}^2 \to \mathbb{S}$ is defined as follows:

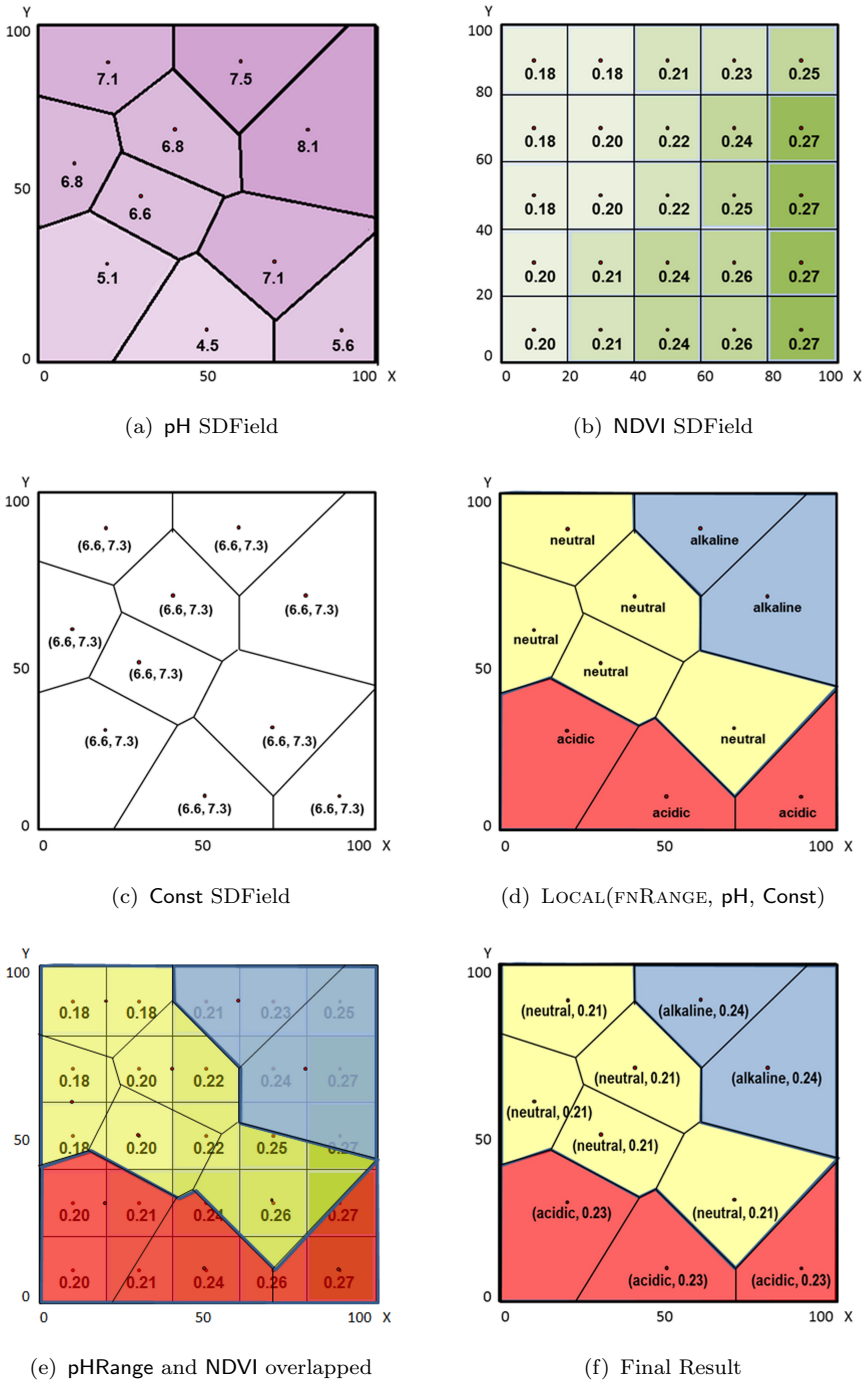
FNRANGE(v1,v2)={*if* v1 < v2[0] *return* 'acidic' *else if* v1 < v2[1] *return* 'neutral' *else return* 'alkaline'}. In the expression above, v2[0] and v2[1] are the boundaries of the pH intervals.

Figure 7.13(d) depicts this intermediate result. Finally, to obtain the average by zone we apply a ZONAL operator between this partial resulting DField and NDVI SDField. Figure 7.13(e) shows the overlapping between this intermediate result and NDVI SDField previous to the final step. The final resulting SDField can be shown in Figure 7.13(f)

Since the Generic Map Algebra is closed, the query can be expressed as: ZONAL(LOCAL(fnRange, pH, cte), NDVI, AVG). □

7.5 Summary

In this chapter we showed that continuous fields can be represented in a discrete form, and that this discretization can be performed in many ways. Based on this discrete model we generalized the traditional map algebra operations redefining them to support different kinds of representations (the original map algebra operations only work over a raster tesselation). In summary, our generalized map algebra operators receive a field, and return a field, although these fields can be represented in different ways. We will use this model in the next chapter, to show that the generic cubes introduced in the previous chapters can be instantiated with continuous data, and that we can perform OLAP operations over continuous fields as if they were just standard data cubes, provided that the former can be represented in a discrete way. Therefore, the algebra introduced in this chapter can somehow be considered the link between continuous fields and OLAP.



(a) pH SDField

(b) NDVI SDField

(c) Const SDField

(d) Local(fnRange, pH, Const)

(e) pHRange and NDVI overlapped

(f) Final Result

Figure 7.13: Integrated Example: Zonal(Local(fnRange, pH,cte), NDVI, Avg).

Chapter 8

Modeling Continuous Fields as Data Cubes

Nowadays, a large number of data sources need to be integrated to operate with traditional OLAP and SOLAP cubes, ideally in a way such that end users do not care about the logical and/or representation of the underlying data. Further, it would be desirable that the end user can manipulate all the available information for decision making through a generic high-level language. In this chapter we make use of the machinery developed in previous chapters in order to solve this problem. In simple words, we integrate cubes of different types and provide the end user with a language that only manipulates data cubes, using the OLAP operators formally defined in Chapter 4. Thus, the language operates at the conceptual level. At the logical level, a layer with the actual data types is defined. Yet below this layer there can be many ways of implementing these logical operations, depending on how data are actually represented. For example, we have seen that continuous field data can be represented in a discrete way in many forms, namely raster, Voronoi, and TIN, among other ones. At the physical level, the OLAP operations can also be implemented in several ways.

In Chapter 7 we have studied how continuous fields can be represented in a discrete way yet allowing the user to perceive these data as continuous, by means of interpolation functions. In this chapter we use these concepts to represent continuous data as a data cube, and therefore seamlessly combine continuous data perceived in this way, with other OLAP and SOLAP (that is, discrete) cubes

applying the typical OLAP operations. For this, we must:

1. Identify the source (continuous) data. In the case of continuous fields, for example, they can be represented as satellite images.

2. Transform these continuous data into discrete data structures (in our case, SDFields, and STDFields)

3. Detect if it is possible to map the cube data structure of Definition 3 in the source data

4. Represent the discretized data as a data cube

5. Define the OLAP operations at the logical level over the discretized data.

Simply stated, once the cube is identified and defined in a discrete way, the user can address queries to a collection of cubes of different kinds, as shown in Chapter 6. At a lower abstraction level, these operations are translated to the underlying data model, for example, using the map algebra operations. The implementation of the operations varies according to the discrete data representation. Specifically, for continuous fields, a ROLL-UP operation at the conceptual level can be defined as a ZONAL map algebra operation which is finally implemented according with the actual discretized representation (raster, Voronoi, etc.).

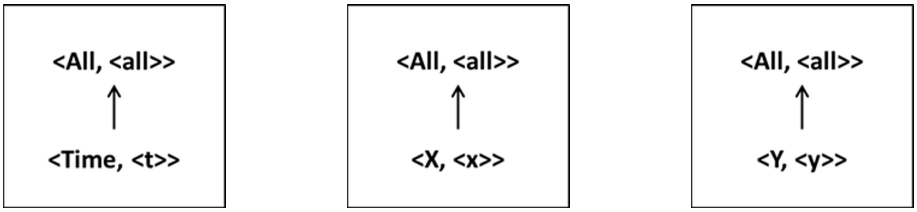
Note that although in this chapter we address the problem of continuous fields, this procedure can be applied to any kind of data that can be represented in a discrete multidimensional way.

8.1 Conceptual Representation of Fields as OLAP Cubes

The usual OLAP concepts of measures and dimensions apply straightforwardly to the notion of field: the measure is precisely what the field represents (e.g., temperature values, NDVI index, etc.) and the cube dimensions can be the spatial and/or the temporal dimension. Thus, we can define a middleware structure that represents a field as a cube, which we denote a FOLAP cube. However, since cubes

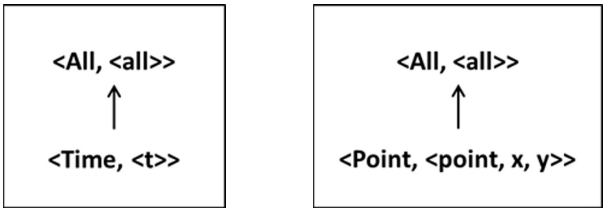
are discrete structures, prior to define the field as a cube we need to discretize such field, using the notion of DField explained before in this thesis.

For example, we can model temperature facts in three dimensions based on the STDField Temperature in Figure 7.6(c). In this case, the dimension lattice has exactly two levels, the bottom one representing the DField data, and the distinguished level All. Figures 8.1 and 8.2 graphically show both dimensions: the time dimension lattice (TemporalDim), and two possible options for modeling the bottom level in the spatial dimension lattice. The first one models space coordinates as two different dimensions X and Y; the second one models the space as a unique Point level.



(a) TemporalDim (b) XDim (c) YDim

Figure 8.1: FOLAP Dimension Lattices - Option 1



(a) TemporalDim (b) SpatialDim

Figure 8.2: FOLAP Dimension Lattices - Option 2

From now on, we will use the second option. Notice that we model the level descriptors of Point as ⟨point, x, y⟩ so we can apply conditions over the geometry point(x, y), or over its coordinates x and y, indistinctly.

Starting from the intuition that our data model allows us to perceive a DField as an OLAP Cube and apply the traditional OLAP operators on it, in what follows we

formally define a FOLAP cube associated to spatial and spatio-temporal DFields. To be concise, we give the definition for a spatio-temporal field. The definition for a spatial field is obtained just omitting the temporal part.

Definition 36 (FOLAP Cube Schema associated to a STDField)**.** Given a spatiotemporal DField F with N spatial dimensions and one temporal dimension, its associated FOLAP cube schema is defined as the tuple $\langle \mathsf{nameF}, \mathcal{D}, \mathcal{M} \rangle$, such that:

(a) nameF is the name of the STDField;

(b) $\mathcal{D} = \{\mathsf{SpatialDim}, \mathsf{TemporalDim}\}$ where the dimensions are given by $\langle \mathsf{SpatialDim}, \{\langle \mathsf{Point}, \langle \mathsf{point}, \mathsf{x}_1, \ldots, \mathsf{x}_N \rangle \rangle, \langle \mathsf{All}, \langle \mathsf{all} \rangle \rangle, \rightarrow_s \rangle\}$, with lattice $\{\mathsf{Point} \rightarrow_s \mathsf{All}\}$, and $\langle \mathsf{TemporalDim}, \{\langle \mathsf{Time}, \langle \mathsf{t} \rangle \rangle, \langle \mathsf{All}, \langle \mathsf{all} \rangle \rangle\}, \rightarrow_t \rangle$ with lattice $\{\mathsf{Time} \rightarrow_t \mathsf{All}\}$, with $\mathsf{Dom}(\mathsf{x_i}) = \mathsf{Dom_i}(\mathsf{F})\ \forall i, i = 1..(\mathsf{N})$, $\mathsf{Dom}(\mathsf{point}) = \mathsf{Dom_s}(\mathsf{F})$, and and $\mathsf{Dom}(\mathsf{t}) = \mathsf{Dom_t}(\mathsf{F})$;

(c) $\mathcal{M} = \{\mathsf{value_F}\}$ with $\mathsf{Dom}(\mathsf{value_F}) = \mathsf{Range}(\mathsf{F})$, where $\mathsf{value_F}$ is the value of F which is associated with an aggregate function in $\mathcal{A}$.

□

Note that the level descriptors for the level Point include an N-dimensional point and also their isolated coordinates, in order to allow a more versatile analysis. Also note that although at the conceptual level we propose the dimension names $\mathsf{TemporalDim}$ and $\mathsf{SpatialDim}$, at the implementation level any name may be given to such dimensions.

Example 45 (FOLAP Cube Schema associated to the pH SDField)**.** Given the pH SDField in Example 37, its corresponding FOLAP cube schema is defined as: $\langle \mathsf{pHF}, \{\mathsf{SpatialDim}\}, \{\mathsf{value_{pH}}\} \rangle$ where $\mathsf{SpatialDim}$ is given by the lattice shown in Figure 8.2(b), with the following level descriptor domains: $\mathsf{Dom}(\mathsf{point})$=[0, 100]×[0, 100] and $\mathsf{Dom}(\mathsf{x})$=$\mathsf{Dom}(\mathsf{y})$=[0, 100]. The measure has $\mathsf{Dom}(\mathsf{value_{pH}})$=[3, 9] and its aggregate function is $\textsc{Avg} \in \mathcal{A}$. □

Example 46 (FOLAP Cube Schema associated to Temperature STDField)**.** Given the STDField Temperature in Example 37, its corresponding FOLAP cube schema is defined by $\langle \mathsf{TempF}, \{\mathsf{TemporalDim}, \mathsf{SpatialDim}\}, \{\mathsf{value}_{\mathsf{temp}}\}\rangle$, where the dimension lattices are shown in Figure 8.2. The level descriptor domains are $\mathsf{Dom}(\mathsf{point}) = [0, 100] \times [0, 100], \mathsf{Dom}(\mathsf{x}) = \mathsf{Dom}(\mathsf{y}) = [0, 100]$, $\mathsf{Dom}(\mathsf{t}) =$ [2009-01-01:00:00:00, 2009-12-31:59:59:59]. The domain of the measure is $\mathsf{Dom}(\mathsf{value}_{\mathsf{temp}}) = [-10, 35]$; the associated aggregate function in this case is AVG. □

In order to define a FOLAP cube instance we need to identify the dimension instance for each dimension schema. Actually, the members of the FOLAP cube bottom level are the infinite elements of the continuous domain of the associated DField. However, given that we must work with a finite number of members (Definition 2), we take advantage of the discretized field model, and define the members of the bottom levels as the *sampled points* of the DField. Since the spatial and temporal lattices have only two levels (i.e., the bottom one and the All level), there will be a single RUP function for each dimension, i.e., $\text{RUP}^{\mathsf{All}}_{\mathsf{Time}}$ for TemporalDim, and $\text{RUP}^{\mathsf{All}}_{\mathsf{Point}}$ for SpatialDim. Finally, each FOLAP instance has either two cuboids (for a spatial FOLAP cube), or four cuboids (for a spatiotemporal FOLAP cube). The following definition formalizes this concept (we define spatial and spatio-temporal instances separately):

Definition 37 (FOLAP Cube Instance associated to an SDField)**.** Given a FOLAP cube schema $\langle \mathsf{F}, \mathcal{D}, \mathcal{M}\rangle$ corresponding to an N-dimensional SDField F, its dimension instance is built (according to Definition 2) as follows:

The SpatialDim instance I_s, is composed of $\mathcal{T}_{\mathsf{Point}} = \{\langle \mathsf{point}, \mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{N}\rangle | (\mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{N}, \mathsf{v}) \in \mathsf{Samples}(\mathsf{F}) \wedge \mathsf{point} = (\mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{N})\}$ and $\mathcal{T}_{\mathsf{All}} = \{\langle \mathsf{all}\rangle\}$, and $\mathcal{R} = \{\text{RUP}^{\mathsf{All}}_{\mathsf{Point}}\}$ such that $\text{RUP}^{\mathsf{All}}_{\mathsf{Point}} = \{(\langle \mathsf{m}\rangle, \langle \mathsf{all}\rangle) | \mathsf{m} \in \mathcal{T}_{\mathsf{Point}})\}$

Then the FOLAP cube instance is composed of the following two cuboids:

$\mathsf{C}_{\mathsf{Point}} : \mathcal{T}_{\mathsf{Point}} \rightarrow \mathsf{Dom}(\mathsf{value}_\mathsf{F})$;

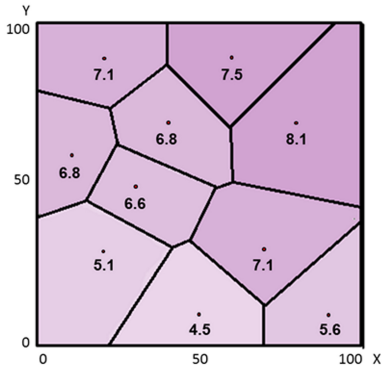
$\mathsf{C}_{\mathsf{All}} : \mathcal{T}_{\mathsf{All}} \rightarrow \mathsf{Dom}(\mathsf{value}_\mathsf{F})$,

where if (c, v) belongs to the cuboid $\mathsf{C}_{\mathsf{Point}}$, then $\mathsf{c} = \langle \mathsf{point}, \mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{N}\rangle \in \mathcal{T}_{\mathsf{Point}}$ and

$(\mathsf{sp}(\mathsf{point}), \mathsf{v}) \in \mathsf{Samples}(\mathsf{F})$. The cuboid $\mathsf{C}_{\mathsf{All}}$ is computed applying the $\text{RUP}^{\mathsf{All}}_{\mathsf{Point}}$ function. □

The key difference between the FOLAP cube instance and a regular OLAP cube instance is that, although an element is not present in the dimension instance of a FOLAP cube (since the domain has infinite values), it can always be inferred. However, the end-user only works with the finite set of tuples that she sees in the FOLAP cube, although all values can be computed if necessary, as explained in Section 8.2.

Example 47 (FOLAP Cube Instance associated to the pH SDField)**.** Consider the FOLAP cube schema pHF of Example 45. Figure 8.3 shows the pH SDField with the corresponding bottom cuboid of its associated pHF FOLAP cube, and Figure 8.4 depicts its full two-cuboids lattice. Notice that the FOLAP bottom cuboid has exactly ten cells, since the associated SDField has ten samples whose values are not $\perp$. □



(a) pH SDField

SpatialDim Point point, x, y	Measures (Avg) $value_{pH}$
⟨(11, 59), 11, 59⟩	6.8
⟨(21, 30), 21, 30⟩	5.1
⟨(21, 88), 21, 88⟩	7.1
⟨(30, 48), 30, 48⟩	6.6
⟨(40, 69), 40, 69⟩	6.8
⟨(50, 10), 50, 10⟩	4.5
⟨(60, 88), 60, 88⟩	7.5
⟨(70, 30), 70, 30⟩	7.1
⟨(79, 69), 79, 69⟩	8.1
⟨(89, 11), 89, 11⟩	5.6

(b) pHF bottom cuboid

Figure 8.3: FOLAP bottom cuboid associated to the pH SDField

Definition 38 (FOLAP Cube Instance associated to an STDField)**.** Given a FOLAP cube schema $\langle \mathsf{F}, \mathcal{D}, \mathcal{M} \rangle$ corresponding to a SDTField F with an N-dimensional spatial dimension, its dimension instances are built according to Definition 2 as follows:

F_{All}

SpatialDim	Measures
All	Avg
all	$value_{pH}$
<all>	6,6

F_{Point}

SpatialDim	Measures
Point	Avg
point, x, y	$value_{pH}$
<(11, 59), 11, 59>	6.8
<(21, 30), 21, 30>	5.1
<(21, 88), 21, 88>	7.1
<(30, 48), 30, 48>	6,6
<(40, 69), 40, 69>	6.8
<(50, 10), 50, 10>	4.5
<(60, 88), 60, 88>	7.5
<(70, 30), 70, 30>	7.1
<(79, 69), 79, 69>	8.1
<(89, 11), 89, 11>	5.6

Figure 8.4: FOLAP cube instance associated to the pH SDField

The SpatialDim instance $\mathsf{I_s}$, is composed of $\mathcal{T}_{\mathsf{Point}} = \{\langle \mathsf{point}, \mathsf{x_1}, \ldots, \mathsf{x_N}\rangle | (\mathsf{x_1}, \ldots, \mathsf{x_n}, \mathsf{v}) \in \mathsf{Samples(F)} \wedge \mathsf{point} = (\mathsf{x_1}, \ldots, \mathsf{x_N})\}$ and $\mathcal{T}_{\mathsf{All}} = \{\langle \mathsf{all}\rangle\}$, and $\mathcal{R} = \{\textsc{Rup}^{\mathsf{All}}_{\mathsf{Point}}\}$ such that $\textsc{Rup}^{\mathsf{All}}_{\mathsf{Point}} = \{(\langle \mathsf{m}\rangle, \langle \mathsf{all}\rangle) | \mathsf{m} \in \mathcal{T}_{\mathsf{Point}})\}$; the TemporalDim instance $\mathsf{I_t}$, is composed of $\mathcal{T}_{\mathsf{Time}} = \{\langle \mathsf{x_t}\rangle | (\mathsf{x_1}, \ldots, \mathsf{x_i}, \ldots, \mathsf{x_N}, \mathsf{x_t}, \mathsf{v}) \in \mathsf{Samples(F)}\}$, and $\mathcal{T}_{\mathsf{All}} = \{\langle \mathsf{all}\rangle\}$, and $\mathcal{R} = \{\textsc{Rup}^{\mathsf{All}}_{\mathsf{Time}}\}$ such that $\textsc{Rup}^{\mathsf{All}}_{\mathsf{Time}} = \{(\langle \mathsf{m}\rangle, \langle \mathsf{all}\rangle) | \mathsf{m} \in \mathcal{T}_{\mathsf{Time}}\}$.

Then the FOLAP cube instance is composed of the following four cuboids:

$\mathsf{C_{Time-Point}} : \mathcal{T}_{\mathsf{Time}} \times \mathcal{T}_{\mathsf{Point}} \rightarrow \mathsf{Dom(value_F)}$,

$\mathsf{C_{Time-All}} : \mathcal{T}_{\mathsf{Time}} \times \mathcal{T}_{\mathsf{All}} \rightarrow \mathsf{Dom(value_F)}$,

$\mathsf{C_{All-Point}} : \mathcal{T}_{\mathsf{All}} \times \mathcal{T}_{\mathsf{Point}} \rightarrow \mathsf{Dom(value_F)}$,

$\mathsf{C_{All-All}} : \mathcal{T}_{\mathsf{All}} \times \mathcal{T}_{\mathsf{All}} \rightarrow \mathsf{Dom(value_F)}$

where if $(\mathsf{c_1}, \mathsf{c_2}, \mathsf{v})$ belongs to the cuboid $\mathsf{C_{Time-Point}}$, then $\mathsf{c_1} = \langle \mathsf{point}, \mathsf{x_1}, \ldots, \mathsf{x_N}\rangle \in \mathcal{T}_{\mathsf{Point}}$, $\mathsf{c_2} = \langle \mathsf{x_t}\rangle \in \mathcal{T}_{\mathsf{Time}}$, and $(\mathsf{sp(point)}, \mathsf{c_2}, \mathsf{v}) \in \mathsf{Samples(F)}$. The rest of the cuboids are computed by applying the corresponding Rup function. □

Example 48 (FOLAP Cube Instance associated to the Temperature STDField)**.**

Consider the FOLAP cube schema TempF given in Example 46. Figure 8.5 depicts the Temperature STDField with the corresponding bottom cuboid of its associated FOLAP cube, and Figure 8.6 depicts its full four-cuboids lattice. Notice that in the FOLAP bottom cuboid the number of cells is less than the number of samples in Temperature, since there are $\perp$ values in the samples in Temp_3 and Temp_4 of Seq(Temperature). For example, the cell ($\langle$2009-06-30:12:05:46$\rangle$, $\langle$(10.0,10.0),10.0, 10.0$\rangle$,$\perp$) is present in the STDField but not in the FOLAP bottom cuboid. □

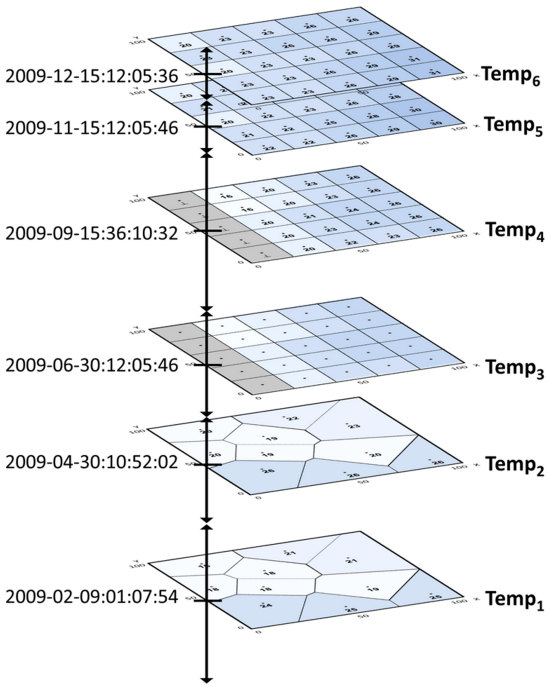
Remark 16. *In what follows, when we refer to a FOLAP cuboid, for brevity we will say, in general, "DField associated to the FOLAP cuboid F" instead of "DField associated to the FOLAP cube containing the FOLAP cuboid F".* □

When OLAP operators are applied to a FOLAP cuboid, the user can treat them as typical OLAP cuboids. However, we will see that the operations induce changes into the corresponding FOLAP Cube and its associated DField, to keep the underlying field consistent with the chained operations. In consequence, at a conceptual level, the OLAP operators on FOLAP cuboids have the same syntax and semantics defined in Chapters 4 and 5. We discuss next how these operators can be implemented for FOLAP cuboids at the Logical Level.

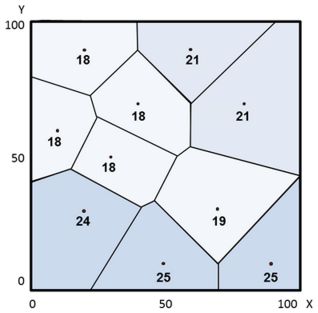
8.2 Operations over FOLAP Cuboids at the Logical Level

Several well-known implementations exist for typical OLAP cubes: MOLAP (Multidimensional OLAP), where data are stored in a multidimensional structure; ROLAP (Relational OLAP), that manipulates the data stored in the relational database to give the appearance of traditional OLAP cube functionality but performing SQL statements internally; HOLAP (Hybrid OnlineAnalytical), that combines MOLAP and ROLAP. We will use a ROLAP implementation where only the tables corresponding to the bottom cuboids are materialized.

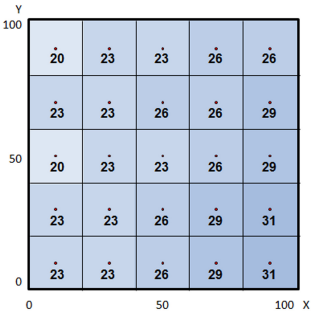
In this section we will explain how the OLAP operations over FOLAP cuboids are implemented individually, applying the operations of the generic algebra for fields explained in Chapter 7. In the next chapter we will explain how they can



(a) Temperature STDField



(b) Temp_1 detail



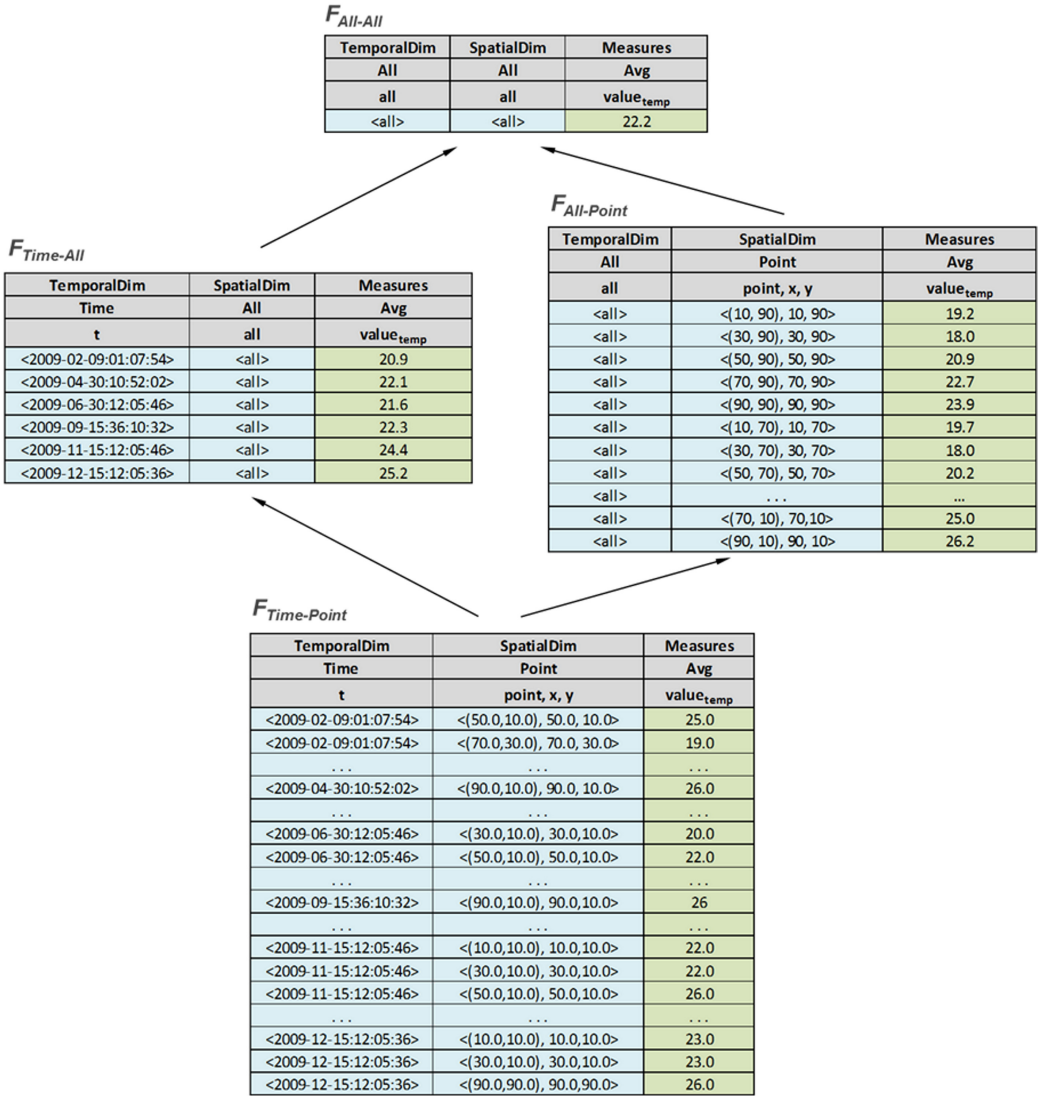
(c) Temp_6 detail

TemporalDim Timestamp t	SpaceDim Point point, x, y	Measures (Avg) $value_{temp}$
⟨2009-02-09:01:07:54⟩	⟨(50.0,10.0), 50.0, 10.0⟩	25.0
⟨2009-02-09:01:07:54⟩	⟨(70.0,30.0), 70.0, 30.0⟩	19.0
...	...	...
⟨2009-04-30:10:52:02⟩	⟨(90.0,10.0), 90.0, 10.0⟩	26.0
...	...	...
⟨2009-06-30:12:05:46⟩	⟨(30.0,10.0), 30.0,10.0⟩	20.0
⟨2009-06-30:12:05:46⟩	⟨(50.0,10.0), 50.0,10.0⟩	22.0
...	...	...
⟨2009-09-15:36:10:32⟩	⟨(90.0,10.0), 90.0,10.0⟩	26
...	...	...
⟨2009-11-15:12:05:46⟩	⟨(10.0,10.0), 10.0,10.0⟩	22.0
⟨2009-11-15:12:05:46⟩	⟨(30.0,10.0), 30.0,10.0⟩	22.0
⟨2009-11-15:12:05:46⟩	⟨(50.0,10.0), 50.0,10.0⟩	26.0
...	...	...
⟨2009-12-15:12:05:36⟩	⟨(10.0,10.0), 10.0,10.0⟩	23.0
⟨2009-12-15:12:05:36⟩	⟨(30.0,10.0), 30.0,10.0⟩	23.0
⟨2009-12-15:12:05:36⟩	⟨(90.0,90.0), 90.0,90.0⟩	26.0

(d) TempF bottom cuboid

Figure 8.5: FOLAP bottom cuboid associated to Temperature STDField

be put together in a generic algebra expression. This distinction is relevant, since the some operators may have different requirements depending on where they are



$F_{All\text{-}All}$

TemporalDim	SpatialDim	Measures
All	All	Avg
all	all	$value_{temp}$
<all>	<all>	22.2

$F_{All\text{-}Point}$

TemporalDim	SpatialDim	Measures
All	Point	Avg
all	point, x, y	$value_{temp}$
<all>	<(10, 90), 10, 90>	19.2
<all>	<(30, 90), 30, 90>	18.0
<all>	<(50, 90), 50, 90>	20.9
<all>	<(70, 90), 70, 90>	22.7
<all>	<(90, 90), 90, 90>	23.9
<all>	<(10, 70), 10, 70>	19.7
<all>	<(30, 70), 30, 70>	18.0
<all>	<(50, 70), 50, 70>	20.2
<all>	...	...
<all>	<(70, 10), 70,10>	25.0
<all>	<(90, 10), 90, 10>	26.2

$F_{Time\text{-}All}$

TemporalDim	SpatialDim	Measures
Time	All	Avg
t	all	$value_{temp}$
<2009-02-09:01:07:54>	<all>	20.9
<2009-04-30:10:52:02>	<all>	22.1
<2009-06-30:12:05:46>	<all>	21.6
<2009-09-15:36:10:32>	<all>	22.3
<2009-11-15:12:05:46>	<all>	24.4
<2009-12-15:12:05:36>	<all>	25.2

$F_{Time\text{-}Point}$

TemporalDim	SpatialDim	Measures
Time	Point	Avg
t	point, x, y	$value_{temp}$
<2009-02-09:01:07:54>	<(50.0,10.0), 50.0, 10.0>	25.0
<2009-02-09:01:07:54>	<(70.0,30.0), 70.0, 30.0>	19.0
...	...	...
<2009-04-30:10:52:02>	<(90.0,10.0), 90.0, 10.0>	26.0
...	...	...
<2009-06-30:12:05:46>	<(30.0,10.0), 30.0,10.0>	20.0
<2009-06-30:12:05:46>	<(50.0,10.0), 50.0,10.0>	22.0
...	...	...
<2009-09-15:36:10:32>	<(90.0,10.0), 90.0,10.0>	26
...	...	...
<2009-11-15:12:05:46>	<(10.0,10.0), 10.0,10.0>	22.0
<2009-11-15:12:05:46>	<(30.0,10.0), 30.0,10.0>	22.0
<2009-11-15:12:05:46>	<(50.0,10.0), 50.0,10.0>	26.0
...	...	...
<2009-12-15:12:05:36>	<(10.0,10.0), 10.0,10.0>	23.0
<2009-12-15:12:05:36>	<(30.0,10.0), 30.0,10.0>	23.0
<2009-12-15:12:05:36>	<(90.0,90.0), 90.0,90.0>	26.0

Figure 8.6: FOLAP cube associated to Temperature STDField

located in a query evaluation tree. For example, according to the semantics given in Section 4.1, the IPO operators do not introduce changes in the cuboids (i.e., in the tables representing the cuboids) that compose a cube instance. However, the IGO operators can eliminate tuples or columns in the cuboids of such instance (see Chapter 9). For example, we have seen that if in a DICE operation (an IGO one) some of the values in a non-bottom cuboid C do not satisfy the dicing conditions, all the cuboids in the path from the bottom to C must be modified to keep the

instance consistent during navigation. In addition, we associate just one DField to each FOLAP cube, namely the DField corresponding to the bottom cuboid. Since a DICE operation may change this DField, to keep this bottom cuboid for future queries we need to perform some additional operations at the physical level, for example, we may need to take a copy of the cuboid. All of these, however, will occur behind the scenes, and will not be seen by the end-user.

8.2.1 Roll-up over FOLAP Cuboids

Although at first sight we can only define a ROLL-UP from the bottom level to the All level in a FOLAP cuboid (since all its dimensions have only two levels), it is also possible to aggregate field data over other spatial and temporal levels. For example, we can take the average altitude over a region or country in the spatial dimension, or the average temperature at different points, by month, quarter, or year, in the temporal dimension. For this, we need to insert new levels into the FOLAP cube dimensions. For instance, a level L can be inserted between two existing levels L_1 and L_2 of the D dimension in a FOLAP cube; not only the schema must be modified with this new level, but also the instance. The latter is performed populating the new level L with its new members, and adding the corresponding functions $\text{RUP}_{\mathsf{L}_1}^{\mathsf{L}}$ and $\text{RUP}_{\mathsf{L}}^{\mathsf{L}_2}$ (see Example 27). For the spatial dimension, the new level must have exactly one level descriptor with geometric domain. Furthermore, when the new level L is introduced between the bottom level L_b and another level L_u, the RUP function is applied to the continuous data in the DField domain. For example, if we insert the City level between Point and All in a FOLAP cube C associated to an SDfield F, each cell c_i representing a city at the City level in C aggregates all the points of the continuous domain of F whose locations are contained into the geometry of c_i.

Since a FOLAP cube may have spatial and temporal dimensions, there are basically three possibilities for applying a roll-up operation:

- Spatial ROLL-UP on a FOLAP cuboid associated to an SDField
- Spatial ROLL-UP on a FOLAP cuboid associated to an STDField

- Temporal ROLL-UP on a FOLAP cuboid associated to an STDField

Spatial Roll-up on a FOLAP Cuboid Associated to an SDField

In this case, the roll-up is performed along the spatial dimension D with schema $\langle \mathsf{D}, \mathcal{L}, \rightarrow \rangle$, up to a level $\mathsf{L} \in \mathcal{L}$ containing exactly one level descriptor whose domain type is an N-dimensional geometry. For example, we may have a geometry defining regions and compute the average altitude, aggregating elevation measures from points to regions. This aggregation can be performed through the ZONAL operator (see Section 7.4.3). The input is a FOLAP cuboid $\mathsf{C_{in}}$ associated to an N-dimensional SDField $\mathsf{F_{in}}$, and the values of $\mathsf{F_{in}}$ are aggregated over the zones defined by the geometries of $\mathcal{T}_{\mathsf{L}}$. In other words, the spatial dimension acts like the reference field. This is achieved transforming the spatial discrete geometries of $\mathcal{T}_{\mathsf{L}}$, into equivalent spatial fields, because the Zonal operation requires two *fields* as input. This transformation from geometries to fields is performed by the GEOMTOFIELD operator, which creates a temporary DField tessellated using the geometries of $\mathcal{T}_{\mathsf{L}}$, and such that the domain and labels are the ones of the input DField. In this tesselation, the sampled point of the sampled tuple corresponding to each geometry is any point belonging to the geometry; its sampled value is the label of the geometry in $\mathcal{T}_{\mathsf{L}}$. In order to complete the tessellation, the geometry of the DField domain that is not covered by any geometry of $\mathcal{T}_{\mathsf{L}}$, has a sampled value '$\perp$'.

Definition 39 (GeomToField operator)**.** Given an N-dimensional SDField $\mathsf{F_{in}}$, a set $\mathcal{G}$ of disjoint labeled geometries over an N-dimensional space, the operation $\textsc{GeomToField}(\mathsf{F_{in}}, \mathcal{G})$ returns a new N-dimensional DField $\mathsf{F_o}$ as follows:

(a) $\mathsf{Dom}(\mathsf{F_o}) = \mathsf{Dom}(\mathsf{F_{in}})$

(b) $\mathsf{Labels}(\mathsf{F_o}) = \mathsf{Labels}(\mathsf{F_{in}})$

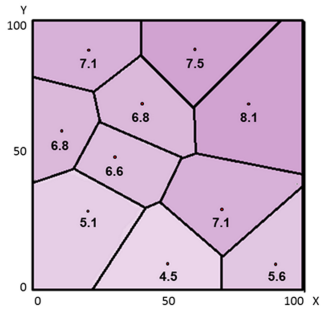
(c) $\mathcal{T}s(\mathsf{Dom}(\mathsf{F_o})) = ((\mathcal{G} \cap \mathsf{GDom}) \cup (\mathsf{GDom} - \mathsf{UG}))$ where GDom is the geometry of $\mathsf{Dom}(\mathsf{F_{in}})$, and UG is the union of all the geometries of $\mathcal{G}$

(d) $\mathsf{Samples}(\mathsf{F_o}) = \{(\mathsf{p},\mathsf{v})|(\exists \mathsf{g} \in (\mathcal{G} \cap \mathsf{GDom}) \wedge \mathsf{p} \in \mathsf{g} \wedge \mathsf{v} = \mathsf{label}(\mathsf{g})) \vee (\exists \mathsf{g} \in (\mathsf{GDom} - \mathsf{UG}) \wedge \mathsf{p} \in (\mathsf{GDom} - \mathsf{UG}) \wedge \mathsf{v} = \perp)\}$

(e) $\mathsf{Range}(\mathsf{F_o}) = \mathsf{Labels}(\mathcal{G}) \cup \{\perp\}$

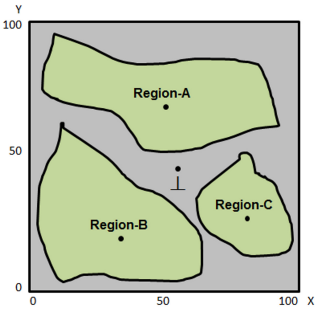
(f) Fs is the constant function. □

Example 49 (GeomToField operator)**.** Consider the pH SDField given in Example 37, and a set $\mathcal{G}$ of geometries corresponding to the rGeom level descriptor of the Region level in Example 4. Figure 8.7 shows the set of geometries and the SDField that results of applying GeomToField(pH,$\mathcal{G}$). □



(a) Input pH SDField

(b) Geometries of $\mathcal{G}$



(c) SDField with geometries as cells

Figure 8.7: GeomToField(pH,$\mathcal{G}$).

It is important to point out that the representative sampled point of each cell may be any point in its geometry. In the case of concave geometries its centroid could be chosen.

Then, ROLL-UP($\mathsf{C_{in}}, \mathsf{D}, \mathsf{L}$), where $\mathsf{C_{in}}$ is a FOLAP cuboid associated to an N-dimensional SDField $\mathsf{F_{in}}$, D is a spatial dimension, and L is a level in the spatial dimension hierarchy, returns a cuboid $\mathsf{C_{out}}$ corresponding to an N-dimensional SDField $\mathsf{F_{out}}$, created by applying the ZONAL operator as follows:

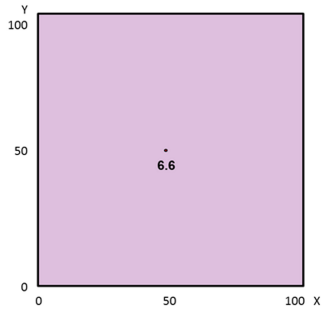
$\mathsf{F_{out}}$ = ZONAL($\mathsf{Zones}, \mathsf{F_{in}}$, FAG), where Zones = GEOMTOFIELD($\mathsf{F_{in}}, \mathcal{G}$) and $\mathcal{G}$ is the set of geometries in $\mathcal{T}_{\mathsf{L}}$.

In particular, if a ROLL-UP is invoked with target level All, then Zones will be a single rectangular zone embedded in the spatial domain of the $\mathsf{F_{in}}$ SDField. This temporary SDField is then represented as a FOLAP output cuboid.

Example 50 (Spatial Roll-up to All on the pHF FOLAP cuboid)**.** Consider the FOLAP cuboid $\mathsf{pHF_{bottom}}$ given in Example 47. Figure 8.8 shows the output FOLAP cuboid $\mathsf{pHF_{All}}$ and its associated SDField after applying ROLL-UP($\mathsf{pHF_{bottom}}$, SpatialDim,All). In this case, as the associated aggregate function for the measure is AVG, our approach computes a weighted average, taking into account the size of the area of each geometry. Since in this case, All represents the complete spatial domain, the final value is 6.6, which is the weighted average of the sampled values in the geometries of Figure 8.7(a). □

SpatialDim All all	Measures (Avg) $value_{pH}$
$\langle all \rangle$	6.6

(a) $\mathsf{pHF_{All}}$ FOLAP cuboid



(b) $\mathsf{pH_{All}}$ SDField

Figure 8.8: $\mathsf{pHF_{All}}$ = ROLL-UP($\mathsf{pHF_{bottom}}$, SpatialDim, All).

Example 51 (Spatial Roll-up to Region on the pHF FOLAP cuboid)**.** Consider the $\mathsf{pHF_{bottom}}$ SDField given in Example 47. We insert a new level Region, between

the levels Point and All, where the new level is given by $\langle \mathsf{Region}, \langle \mathsf{rName}, \mathsf{rGeom} \rangle \rangle$, with $\mathsf{Dom}(\mathsf{rName}) = \mathbb{S}$, $\mathsf{Dom}(\mathsf{rGeom}) = \mathbb{P}_{2D}$, and $\mathcal{T}_{\mathsf{Region}} = \{\langle$'Region-A', geom506$\rangle$, $\langle$'Region-B', geom701$\rangle$, $\langle$'Region-C', geom703$\rangle\}$.

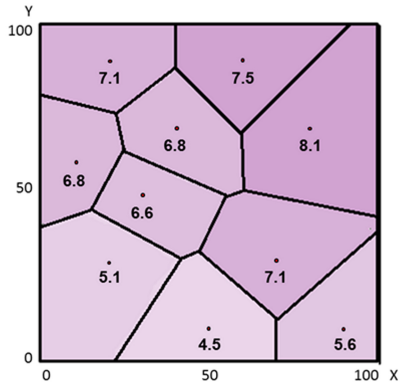
Figure 8.9 shows the output FOLAP cuboid $\mathsf{pHF}_{\mathsf{Region}}$ and its associated SD-Field after applying ROLL-UP($\mathsf{pHF}_{\mathsf{bottom}}$,SpatialDim,Region). Note that actually not only the sampled points are contributing values, but also all the infinite points that fall inside each geometry G over which the aggregation is performed (due to the continuous nature of the domain). Thus, outside the intersection with the geometries, the field takes the value '$\perp$'. Inside each intersection, the value associated with G is given by the weighted average of the values of all the areas in the tesselation of the field pHF that intersect G. For example, in Region-C the value we would obtain if we compute the simple average of the values associated to the areas intersected by Region-C would be $\frac{8.1+7.1+5.6}{3} = 6.9$. Instead, we apply the AVG function weighted with the percentage of each tessellated cell area with respect to the area of the geometry, i.e., the value associated to Region-C is obtained as $0.06 \times 8.1 + 0.65 \times 5.6 + 0.29 \times 7.1 = 6.7$. □

Spatial Roll-up on a FOLAP Cuboid Associated to an STDField

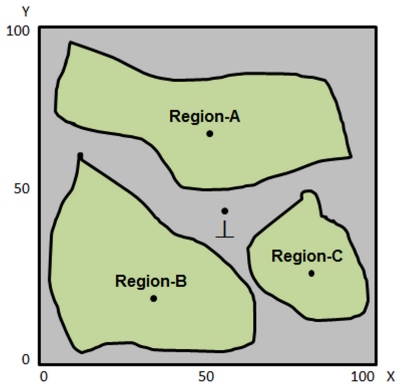
ROLL-UP($\mathsf{C}_{\mathsf{in}}, \mathsf{D}, \mathsf{L}$), where C_{in} is a FOLAP cuboid associated to an STDField F_{in} (with N spatial dimensions), D is a spatial dimension, and L is a level in the spatial dimension hierarchy, returns a cuboid $\mathsf{C}_{\mathsf{out}}$ corresponding to an N-dimensional SDField $\mathsf{F}_{\mathsf{out}}$, created by applying the ZONAL operator to aggregate the values of each SDField of $\mathsf{Seq}(\mathsf{F}_{\mathsf{in}})$ over the zones defined by the geometries in $\mathcal{T}_{\mathsf{L}}$. That means, given a temporal sequence of K spatial discretized fields, a zonal operation is applied to each field in the sequence using as reference field the geometries in L converted into a field by means of a GEOMTOFIELD operation. $\mathsf{F}_{\mathsf{out}}$ is built as follows:

(a) $\mathsf{Dom}(\mathsf{F}_{\mathsf{out}}) = \mathsf{Dom}(\mathsf{F}_{\mathsf{in}}), \forall i = 1..K$

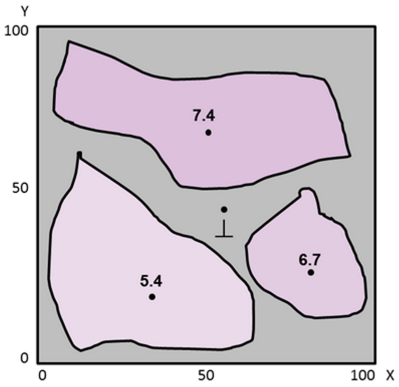
(b) $\mathsf{Labels}(\mathsf{F}_{\mathsf{out}}) = \mathsf{Labels}(\mathsf{F}_{\mathsf{in}}) \forall i = 1..K$



(a) pH SDField



(b) Zones SDField



(c) $\mathsf{pH}_{\mathsf{Region}}$ SDField

SpatialDim Region rName, rGeom	Measures (Avg) $value_{pH}$
$\langle$'$Region-A$', $geom506\rangle$	7.4
$\langle$'$Region-B$', $geom701\rangle$	5.4
$\langle$'$Region-C$', $geom703\rangle$	6.7

(d) $\mathsf{pHF}_{\mathsf{Region}}$ FOLAP cuboid

Figure 8.9: $\mathsf{pHF}_{\mathsf{Region}} = \textsc{Roll-up}(\mathsf{pHF}_{\mathsf{bottom}}, \mathsf{SpatialDim}, \mathsf{Region})$.

(c) $\mathsf{F}_{\mathsf{out}i} = \{\textsc{Zonal}(\textsc{GeomToField}(\mathsf{F}_{\mathsf{in}i}, \mathcal{G}), \mathsf{F}_{\mathsf{in}i}, \textsc{Fag}) \wedge \mathsf{F}_{\mathsf{in}i} \in \mathsf{Seq}(\mathsf{F}_{\mathsf{in}})\}\ \forall i = 1..K$, where $\mathcal{G}$ is the set of geometries of $\mathcal{T}_{\mathsf{L}}$. That is, the same set $\mathcal{G}$ of geometries of $\mathcal{T}_{\mathsf{L}}$ is used for rolling-up all the SDFields in the Sequence.

(d) $\mathsf{Range}(\mathsf{F}_{\mathsf{out}}) = \mathsf{Range}(\textsc{Fag}) \cup \{\perp\}$ where Fag is the aggregate function associated to the measure in C_{in}, and is the same $\forall i = 1..K$

(e) Fs of $\mathsf{F}_{\mathsf{out}}$ is Fs of F_{in}, and is the same $\forall i = 1..K$

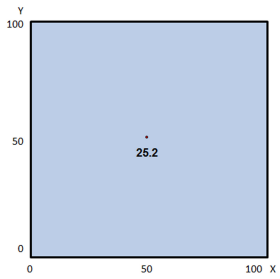
Note that $\mathsf{Dom}(\mathsf{F}_{\mathsf{out}})$, $\mathsf{Labels}(\mathsf{F}_{\mathsf{out}})$ and $\mathsf{Range}(\mathsf{F}_{\mathsf{out}})$ are the same for all the snapshots, following the STDField definition (Definition 26).

Example 52 (Spatial Roll-up to All on the TempF FOLAP cuboid)**.** Consider the

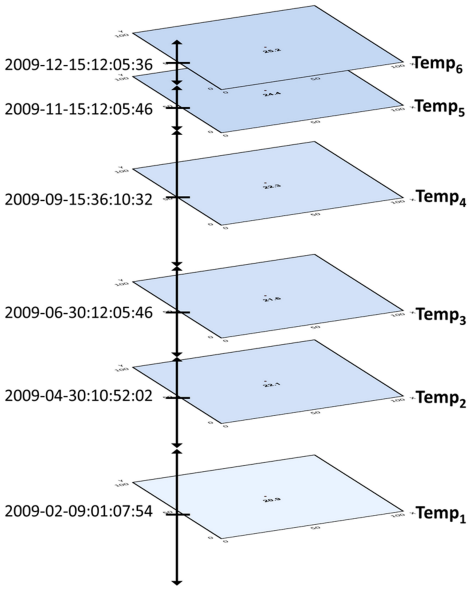
$\mathsf{TempF}_{\mathsf{bottom}}$ FOLAP cuboid given in Example 48. Figure 8.10 shows the output FOLAP cuboid $\mathsf{TempF}_{\mathsf{Time-All}}$ and its associated SDfield after applying ROLL-UP($\mathsf{TempF}_{\mathsf{bottom}}$,SpatialDim,All). Notice that a spatial ROLL-UP is applied to each SDField of $\mathsf{Seq}(\mathsf{TempF}_{\mathsf{bottom}})$. □

TemporalDim Timestamp t	SpaceDim All all	Measures (Avg) $value_{temp}$
⟨2009-02-09:01:07:54⟩	⟨all⟩	20.9
⟨2009-04-30:10:52:02⟩	⟨all⟩	22.1
⟨2009-06-30:12:05:46⟩	⟨all⟩	21.6
⟨2009-09-15:36:10:32⟩	⟨all⟩	22.3
⟨2009-11-15:12:05:46⟩	⟨all⟩	24.4
⟨2009-12-15:12:05:36⟩	⟨all⟩	25.2

(a) $\mathsf{TempF}_{\mathsf{Time-All}}$ FOLAP cuboid



(b) Temp_6 detail



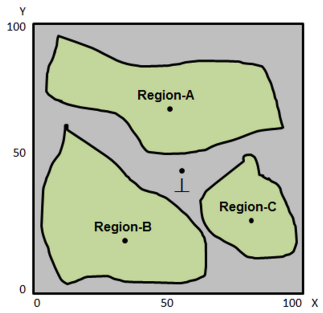
(c) $\mathsf{Temperature}_{\mathsf{Time-All}}$ STDField

Figure 8.10: $\mathsf{TempF}_{\mathsf{Time-All}}$ = ROLL-UP($\mathsf{TempF}_{\mathsf{bottom}}$, SpatialDim, All).

Example 53 (Spatial Roll-up to Region on the TempF FOLAP cuboid)**.** Consider now the $\mathsf{TempF}_{\mathsf{bottom}}$ STDField given in Example 48. To aggregate temperatures by region, we need to insert the level Region given in Example 51. Figure 8.11 shows the output FOLAP cuboid $\mathsf{TempF}_{\mathsf{Time-Region}}$ which is the result of the ROLL-UP($\mathsf{TempF}_{\mathsf{bottom}}$,SpatialDim,Region) operation, obtained from the STDField $\mathsf{TempF}_{\mathsf{out}}$. □

Temporal Roll-up on a FOLAP Cuboid Associated to an STDField

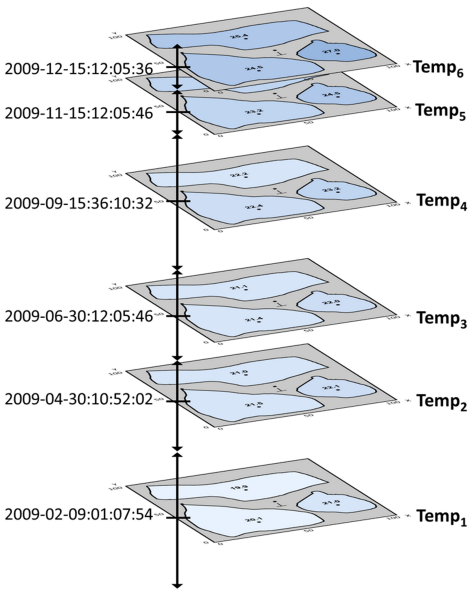
In this case, ROLL-UP($\mathsf{C}_{\mathsf{in}}, \mathsf{D}, \mathsf{L}$) receives a FOLAP cuboid C_{in} associated to an N-dimensional STDField F_{in}, a temporal dimension D with schema $\langle \mathsf{D}, \mathcal{L}, \rightarrow \rangle$, and



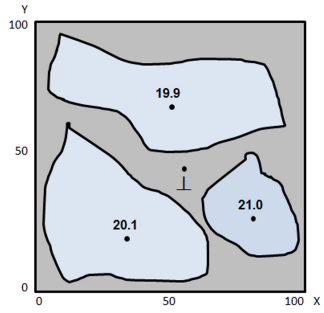
(a) Zones SDField

TemporalDim Timestamp t	SpaceDim Region rName, rGeom	Measures (Avg) $value_{temp}$
⟨2009-02-09:01:07:54⟩	⟨'$Region-A'$, $geom506$⟩	19.9
⟨2009-02-09:01:07:54⟩	⟨'$Region-B'$, $geom701$⟩	20.1
⟨2009-02-09:01:07:54⟩	⟨'$Region-C'$, $geom703$⟩	21.0
⟨2009-04-30:10:52:02⟩	⟨'$Region-A'$, $geom506$⟩	21.0
⟨2009-04-30:10:52:02⟩	⟨'$Region-B'$, $geom701$⟩	21.5
⟨2009-04-30:10:52:02⟩	⟨'$Region-C'$, $geom703$⟩	22.1
⟨2009-06-30:12:05:46⟩	⟨'$Region-A'$, $geom506$⟩	21.1
⟨2009-06-30:12:05:46⟩	⟨'$Region-B'$, $geom701$⟩	21.4
⟨2009-06-30:12:05:46⟩	⟨'$Region-C'$, $geom703$⟩	22.0
⟨2009-09-15:36:10:32⟩	⟨'$Region-A'$, $geom506$⟩	22.2
⟨2009-09-15:36:10:32⟩	⟨'$Region-B'$, $geom701$⟩	22.4
⟨2009-09-15:36:10:32⟩	⟨'$Region-C'$, $geom703$⟩	23.2
⟨2009-11-15:12:05:46⟩	⟨'$Region-A'$, $geom506$⟩	23.2
⟨2009-11-15:12:05:46⟩	⟨'$Region-B'$, $geom701$⟩	23.2
⟨2009-11-15:12:05:46⟩	⟨'$Region-C'$, $geom703$⟩	24.5
⟨2009-12-15:12:05:36⟩	⟨'$Region-A'$, $geom506$⟩	25.4
⟨2009-12-15:12:05:36⟩	⟨'$Region-B'$, $geom701$⟩	24.5
⟨2009-12-15:12:05:36⟩	⟨'$Region-C'$, $geom703$⟩	27.0

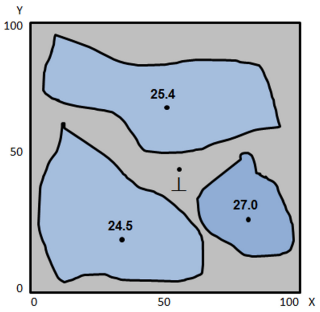
(b) $\mathsf{TempF}_{\mathsf{Time-Region}}$ FOLAP cuboid



(c) $\mathsf{Temperature}_{\mathsf{Time-Region}}$ STDField



(d) Temp_1 detail



(e) Temp_6 detail

Figure 8.11: $\mathsf{TempF}_{\mathsf{Time-Region}} = \textsc{Roll-up}(\mathsf{TempF}_{\mathsf{bottom}}, \mathsf{SpatialDim}, \mathsf{Region})$.

a level $\mathsf{L} \in \mathcal{L}$. The result is a cuboid $\mathsf{C_{out}}$ associated to an STDField $\mathsf{F_{out}}$ built by applying the ZONAL operator which aggregates the values of $\mathsf{F_{in}}$ over the temporal intervals of $\mathcal{T}_\mathsf{L}$.

Since ZONAL operates with two DFields, we must produce an STDField by means of the TIMETOFIELD operation, which is analogous to the GEOMTOFIELD operation previously introduced. This operation creates an STDField as follows. For each temporal interval of $\mathcal{T}_\mathsf{L}$ the operation takes the most recent snapshot (an SDField) in the interval and defines an SDField with the same tesselation and domain. For example, in Figure 8.12, we have three SDFields spanning the fourth quarter: $\mathsf{temp_4}$, $\mathsf{temp_5}$ and $\mathsf{temp_6}$. The tesselation of the latter will be selected since it is the most recent one in the interval. This will be the reference DField for the Zonal operation that will compute the average between all the SDFields in the fourth quarter.

Definition 40 (TimeToField operator)**.** Given an SDTField $\mathsf{F_{in}}$, a set $\mathcal{I}$ of contiguous temporal intervals, TIMETOFIELD$(\mathsf{F_{in}}, \mathcal{I})$ returns a new N-dimensional STDField $\mathsf{F_o}$ as follows:

(a) $\mathsf{Dom_s}(\mathsf{F_o}) = \mathsf{Dom_s}(\mathsf{F_{in}})$

(b) $\mathsf{Dom_t}(\mathsf{F_o}) = ((\mathcal{I} \cap \mathsf{Dom_t}(\mathsf{F_{in}})) \cup (\mathsf{Dom_t}(\mathsf{F_{in}}) - [\mathsf{t_{final}}, \mathsf{t_{init}}]))$ where $\mathsf{t_{init}}$ and $\mathsf{t_{final}}$ are the start of the first interval and the final of the last interval of $\mathcal{I}$, respectively.

(c) $\forall i, i = 1..|\mathcal{I}|, \mathsf{F}_i$ is the most recent snapshot of $\mathsf{F_{in}}$ among the snapshots that have non-empty temporal intersection with $I_i \in \mathcal{I}$.

(d) $\mathsf{Labels}(\mathsf{F_o}) = \mathsf{Labels}(\mathsf{F_{in}})$

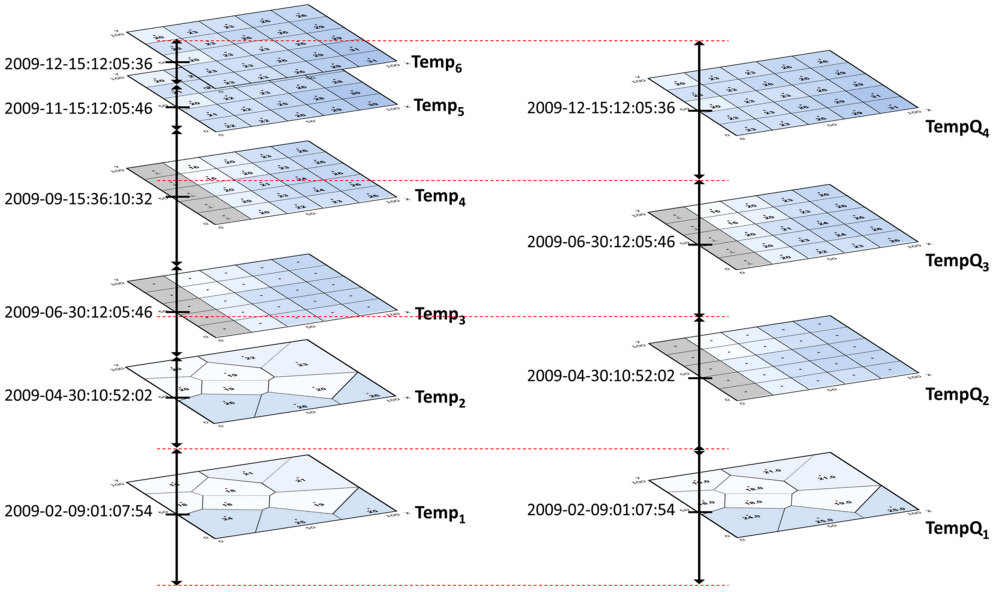
(e) $\mathsf{Range}(\mathsf{F_o}) = \mathsf{Range}(\mathsf{F_{in}}) \cup \{\perp\}$

(f) FS is the constant function.

□

Simply stated, TIMETOFIELD prepares the reference field with the temporal granularity corresponding to the temporal level to which we want to roll-up. Note

that (c) and (e) address the cases where the intervals in the output field do not cover the temporal interval of the input field (for example, there can be an input snapshot taken after the last quarter that we are considering for the aggregation). In this case, initial and final intervals with sampled values $\perp$ are added.



(a)

Figure 8.12: TimeToField(Temperature, $\mathcal{I}_{\mathsf{Quarter}}$).

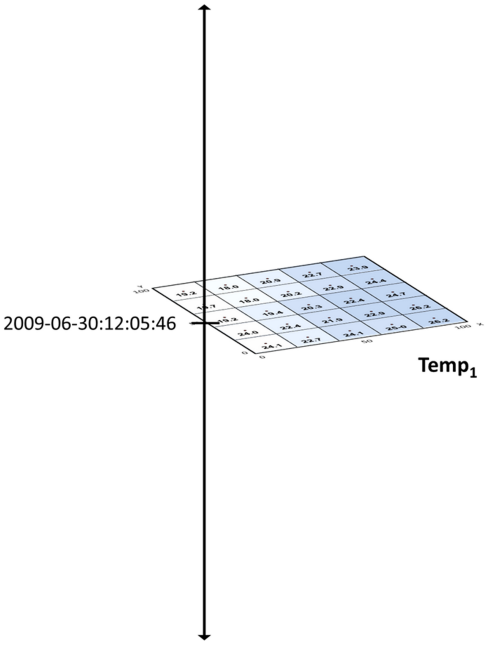
Finally, the STDField $\mathsf{F}_{\mathsf{out}}$ is obtained with $\mathsf{F}_{\mathsf{out}}$ = Zonal(Times, F_{in}, Fag), where Times = TimeToField($\mathsf{F}_{\mathsf{in}}, \mathcal{I}$) and $\mathcal{I}$ is the set of temporal intervals of $\mathcal{T}_{\mathsf{L}}$. In particular, if Roll-up is invoked with target level All, Times will be a single temporal interval embedded in the temporal domain of the F_{in} DField.

Example 54 (Temporal Roll-up to All on the Temperature STDField)**.** Consider the FOLAP cuboid $\mathsf{TempF}_{\mathsf{bottom}}$ given in Example 48. Figure 8.13 shows the output FOLAP cuboid $\mathsf{TempF}_{\mathsf{All-Point}}$ and its associated STDField after applying Roll-up($\mathsf{TempF}_{\mathsf{bottom}}$,TemporalDim,All). Note that the figure shows a representative field located in the middle of the temporal domain (in this case 2009-06-30:12:05:46). This field has the tesselation of the most recent SDField snapshot in the input (the rasterized tesselation corresponding to temp_6) in Example 48, and the sampled

values in each cell are computed as usual in the Zonal operator. □

TemporalDim All all	SpaceDim Point point, x,y	Measures (Avg) $value_{temp}$
⟨all⟩	⟨(10, 90), 10, 90⟩	19.2
⟨all⟩	⟨(30, 90), 30, 90⟩	18.0
⟨all⟩	⟨(50, 90), 50, 90⟩	20.9
⟨all⟩	⟨(70, 90), 70, 90⟩	22.7
⟨all⟩	⟨(90, 90), 90, 90⟩	23.9
⟨all⟩	⟨(10, 70), 10, 70⟩	19.7
⟨all⟩	⟨(30, 70), 30, 70⟩	18.0
⟨all⟩	⟨(50, 70), 50, 70⟩	20.2
⟨all⟩	⟨(70, 70), 70, 70⟩	22.9
⟨all⟩	⟨(90, 70), 90, 70⟩	24.4
⟨all⟩	⟨(10, 50), 10, 50⟩	19.2
⟨all⟩	⟨(30, 50), 30, 50⟩	19.4
⟨all⟩	⟨(50, 50), 50, 50⟩	20.3
⟨all⟩	⟨(70, 50), 70, 50⟩	22.4
⟨all⟩	⟨(90, 50), 90, 50⟩	24.7
⟨all⟩	⟨(10, 30), 10, 30⟩	24.0
⟨all⟩	⟨(30, 30), 30, 30⟩	22.4
⟨all⟩	⟨(50, 30), 50, 30⟩	21.9
⟨all⟩	⟨(70, 30), 70, 30⟩	22.9
⟨all⟩	⟨(90, 30), 90, 30⟩	26.2
⟨all⟩	⟨(10, 10), 10, 10⟩	24.1
⟨all⟩	⟨(30, 10), 30, 10⟩	22.7
⟨all⟩	⟨(50, 10), 50, 10⟩	24.1
⟨all⟩	⟨(70, 10), 70, 10⟩	25.0
⟨all⟩	⟨(90, 10), 90, 10⟩	26.2

(a) $\mathsf{TempF}_{\mathsf{All-Point}}$ FOLAP cuboid



(b) $\mathsf{Temperature}_{\mathsf{All-Point}}$ SDTField

Figure 8.13: $\mathsf{TempF}_{\mathsf{All-Point}} = \text{ROLL-UP}(\mathsf{TempF}_{\mathsf{bottom}}, \mathsf{TemporalDim}, \mathsf{All})$.

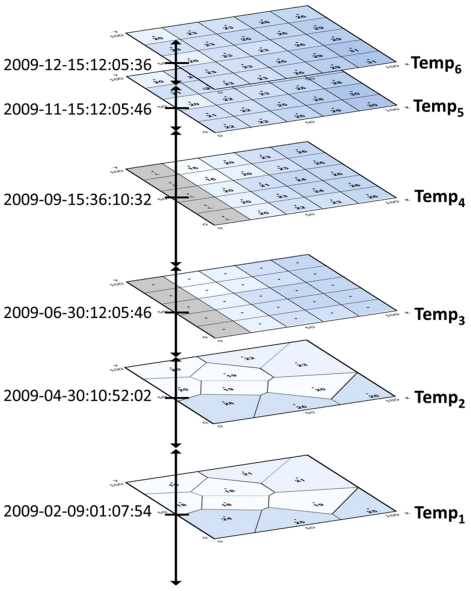
Example 55 (Temporal Roll-up to Quarter on the Temperature STDField)**.** Consider the FOLAP cuboid $\mathsf{TempF}_{\mathsf{bottom}}$ given in Example 48. To aggregate temperatures by quarter, we need to insert the Quarter level between Time and All levels in the TemporalDim dimension. The aggregation is performed by the operation ROLL-UP($\mathsf{TempF}_{\mathsf{bottom}}$,TemporalDim,Quarter). Figure 8.14 shows the resulting output FOLAP cuboid and the associated STDField. We can see that for each quarter, the TIMETOFIELD operation selected the tesselation of the most recent field (e.g., temp_6 for the last quarter). The average between all the SDFields in each quarter is computed as usual. Note that there are four representative samples,

one for each quarter, whose time instant is set to the middle point of the interval. Also, note that in the first quarter, the most recent snapshot has a Voronoi tesselation, and this was chosen as tesselation for the aggregated field in the such quarter. □

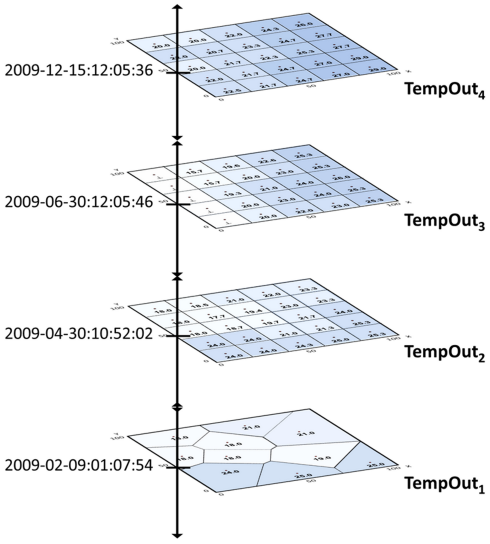
8.2.2 Dice over FOLAP Cuboids

Recall that DICE is an IGO operator (Section 4.2), therefore, a new FOLAP cube instance can be induced by the operation, and in this case, a new associated DField must be built. According to Definition 4.2.1, the DICE condition can only involve literals, measures and descriptors belonging to the input cuboid levels. In consequence, only t, point, x, y descriptors can be used to express conditions over the bottom cuboid. Dicing a bottom cuboid may have two kinds of consequences: first, the tuples in the FOLAP cube that do not satisfy the condition are removed; second, the corresponding samples in the associated DField must be set to $\perp$.

We have also seen that if we want to enhance analysis, levels must be inserted into the cube dimensions. In order to obtain a cuboid containing one of these inserted levels, a ROLL-UP operation must be performed. If we apply a DICE condition over this new cuboid C, the removed cells in C (i.e., the ones that do not satisfy the condition) must be, again, propagated to the bottom cuboid (and its associated DField) in the cube instance in order to guarantee consistency, for example, in future DRILL-DOWN operations. Also, since we do not generate a DField associated to non-bottom cuboids resulting from a roll-up, dicing on such non-bottom cuboid requires computing the intersection between the geometric and/or temporal properties of the cuboid (i.e., the tuples that are not removed) and the cells of the DField associated to the bottom cuboid, in order to induce a new tessellation, if necessary. The cells in the new tessellated DField that do not have spatio and/or temporal intersection with the upper cuboid are set to $\perp$, while the other ones keep their values. This new tessellation is necessary when at least one cell in the associated DField is not totally spatially or temporally contained in the spatio and/or temporal part of a tuple of the upper cuboid that does not satisfy the dicing condition. We illustrate this in Example 58.



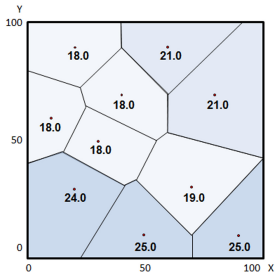
(a) Temperature STDField



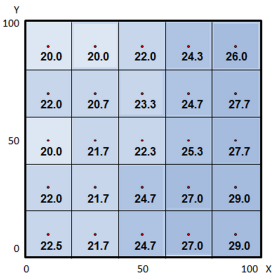
(b) $\mathsf{Temperature}_{\mathsf{Quarter-Point}}$ STDField

TemporalDim Quarter quarter	SpaceDim Point point, x, y	Measures (Avg) $value_{temp}$
⟨Q1-2009⟩	⟨(21, 88), 21, 88⟩	18.0
⟨Q1-2009⟩	⟨(60, 88), 60, 88⟩	21.0
...	...	...
⟨Q1-2009⟩	⟨(89, 11), 89, 11⟩	25.0
⟨Q2-2009⟩	⟨(10, 90), 10, 90⟩	18.0
⟨Q2-2009⟩	⟨(20, 70), 20, 70⟩	18.5
...	...	...
⟨Q2-2009⟩	⟨(90, 10), 90, 10⟩	25.2
...	...	...
⟨Q3-2009⟩	⟨(20, 90), 20, 90⟩	15.7
⟨Q3-2009⟩	⟨(30, 90), 30, 90⟩	19.6
...	...	...
⟨Q3-2009⟩	⟨(90, 10), 90, 10⟩	25.3
⟨Q4-2009⟩	⟨(10, 90), 10, 90⟩	20.0
⟨Q4-2009⟩	⟨(20, 90), 20, 90⟩	20.0
...	...	...
⟨Q4-2009⟩	⟨(90, 10), 90, 10⟩	29.0

(c) $\mathsf{TempF}_{\mathsf{Quarter-Point}}$ FOLAP cuboid



(d) $\mathsf{TempOut}_1$ detail



(e) $\mathsf{TempOut}_4$ detail

Figure 8.14: $\mathsf{TempF}_{\mathsf{Quarter-Point}} = \textsc{Roll-up}(\mathsf{TempF}_{\mathsf{bottom}}, \mathsf{TemporalDim}, \mathsf{Quarter})$.

In consequence, when $\textsc{Dice}(\mathsf{C}_{\mathsf{in}}, \phi)$ receives a FOLAP cuboid C_{in} associated to a DField F_{in}, besides returning the new cuboid $\mathsf{C}_{\mathsf{out}}$ and its new cube instance (see Definition 12), the associated DField becomes a new DField $\mathsf{F}_{\mathsf{out}}$ built as follows

(for clarity we assume that a cell $\mathsf{c_i}$ is divided into $\mathsf{c_{i1}}, \mathsf{c_{i2}}$ although more complex tesselations can occur for irregular regions):

(a) $\mathsf{Dom}(\mathsf{F_{out}}) = \mathsf{Dom}(\mathsf{F_{in}})$

(b) $\mathsf{Labels}(\mathsf{F_{out}}) = \mathsf{Labels}(\mathsf{F_{in}})$

(c) if $\mathsf{C_{in}}$ is the bottom cuboid then $\mathcal{T}s(\mathsf{Dom}(\mathsf{F_{out}})) = \mathcal{T}s(\mathsf{Dom}(\mathsf{F_{in}}))$
otherwise $\mathcal{T}s(\mathsf{Dom}(\mathsf{F_{out}})) = \{\mathsf{c_i} \mid (\mathsf{c_i} \in \mathcal{T}s(\mathsf{Dom}(\mathsf{F_{in}})) \wedge (\forall \mathsf{e} \in \mathsf{C_{out}}(\mathsf{c_i} \subseteq \mathsf{e} \vee \mathsf{c_i} \cap \mathsf{e} = \phi))) \vee (\exists \mathsf{c_{i1}}, \mathsf{c_{i2}} \wedge \mathsf{c_i} = \mathsf{c_{i1}} \cup \mathsf{c_{i2}} \wedge \forall \mathsf{e} \in \mathsf{C_{out}}\ \mathsf{c_{i1}} \cap \mathsf{e} = \phi \wedge \exists \mathsf{e} \in \mathsf{C_{out}}\ \mathsf{c_{i2}} \subseteq \mathsf{e})\}$

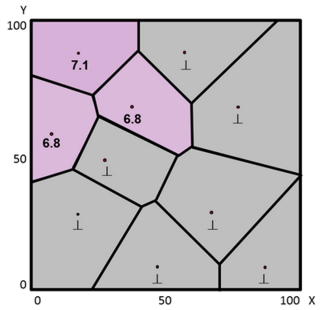
(d) $\mathsf{Samples}(\mathsf{F_{out}}) = \{\langle \mathsf{sp(s)}, \mathsf{v'} \rangle \mid \mathsf{s} \in \mathsf{Samples}(\mathsf{F_{in}}), \mathsf{v'} = \mathsf{sv(s)}$ if ϕ is satisfied, or $\perp$ otherwise$\}$

(e) $\mathsf{Range}(\mathsf{F_{out}}) = \mathsf{Range}(\mathsf{F_{in}}) \cup \{\perp\}$

(f) Fs of $\mathsf{F_{out}}$ is Fs of $\mathsf{F_{in}}$

SpatialDim Point point, x, y	Measures (Avg) $value_{pH}$
$\langle(21, 88), 21, 88\rangle$	7.1
$\langle(40, 69), 40, 69\rangle$	6.8
$\langle(11, 59), 11, 59\rangle$	6.8

(a) $\mathsf{pHF_2}$ FOLAP cuboid



(b) $\mathsf{pH_2}$ SDField

Figure 8.15: $\mathsf{pHF_2} = \textsc{Dice}(\mathsf{pHF_{bottom}}, \mathsf{x} < 50$ and $\mathsf{value} > 6.7)$.

Example 56 (Dice on a bottom FOLAP cuboid)**.** Consider the cuboid $\mathsf{pHF_{bottom}}$ of Example 47. Figure 8.15 shows the output FOLAP cuboid $\mathsf{pHF_2}$ and its associated SDfield after applying $\textsc{Dice}(\mathsf{pHF_{bottom}}, \mathsf{x} < 50$ and $\mathsf{value_{pH}} > 6.7)$.

Notice that all the samples that do not satisfy the condition set the value to $\perp$ in the DField but their corresponding cells are not present in the associated

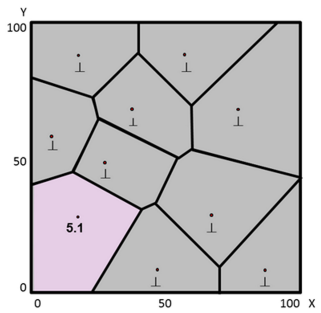
FOLAP cuboid. Since the input cuboid is the bottom cuboid of pHF FOLAP cube, the new output DField keeps its tessellation and only sets with '⊥' the seven cells corresponding to the seven sampled points that do not satisfy the condition. We emphasize that modeling the level descriptors of Point level with both point and *also* the x and y coordinates allows to easily express DICE condition without applying functions to retrieve the coordinates of a sampled point. □

Example 57 (Dice with geometric conditions over a bottom cuboid)**.** Now we show how geometric functions in the DICE condition can be used. In this case we use the *distance* function. Consider the FOLAP cuboid $\mathsf{pHF}_{\mathsf{bottom}}$ given in Example 47 and the operation DICE($\mathsf{pHF}_{\mathsf{bottom}}$,DISTANCE(point, Point2D(10,10)) < 30). Figure 8.16 shows the resulting FOLAP cuboid pHF_3 and its associated DField. Notice that the only sampled tuple in the pH SDField with a sampled point at less than 30 km from the point (10,10) is $\langle((21, 30), 5.1\rangle$ (see Figure 8.16(b)).

Since the input cuboid is the bottom cuboid of pHF FOLAP cube, the new output DField keeps its tessellation and sets with ⊥ the nine cells corresponding to the nine sampled points that do not satisfy the condition. □

SpatialDim Point point, x, y	Measures (Avg) $value_{pH}$
$\langle(21, 30), 21, 30\rangle$	5.1

(a) pHF_3 FOLAP cuboid

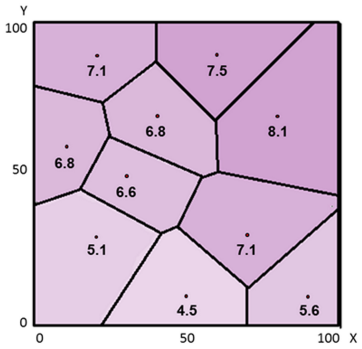


(b) pH_3 SDField

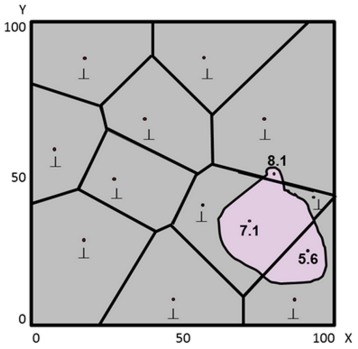
Figure 8.16: pHF_3 = DICE($\mathsf{pHF}_{\mathsf{bottom}}$, DISTANCE(point, POINT2D(10, 10)) < 30).

Example 58 (Dice on a non-bottom FOLAP cuboid)**.** Consider the FOLAP cuboid $\mathsf{pHF}_{\mathsf{Region}}$ obtained in Example 51 through a ROLL-UP to the Region level,

and the operation DICE($\mathsf{pHF}_{\mathsf{Region}}$, AREA($\mathsf{rGeom} < 100$)). Figure 8.17 shows the output FOLAP cuboid pHF_4 and its new associated DField. Since $\mathsf{pHF}_{\mathsf{Region}}$ is not a bottom cuboid of pHF FOLAP cube, a new DField with a new Tessellation is induced by the operation. Notice that the three cells on the right have non-empty intersection with the only member that satisfies the condition, namely 'Region-C'. Thus, the new Tessellation will contain 14 cells instead of 10, because of the splitting of these three cells. In consequence, if after this DICE a DRILL-DOWN is invoked with target level Point, no cell in the bottom cuboid that has not fulfilled the condition in the previous DICE appears in the resulting cube. □



(a) pH SDField



(b) pH_4 SDField

SpatialDim Region rName, rGeom	Measures (Avg) $value_{pH}$
$\langle$'$Region - C'$, $geom703\rangle$	6.2

(c) pHF_4 FOLAP cuboid

Figure 8.17: pHF_4 = DICE($\mathsf{pHF}_{\mathsf{Region}}$, AREA($\mathsf{rGeom}$) < 100).

8.2.3 Slice over FOLAP Cuboids

To define a slice operation, we have two possibilities:

- Spatial or temporal slicing on a FOLAP cuboid associated to an STDField
- Slicing measures on a FOLAP cuboid associated to a SDField or a STDField

This follows from the constraints we defined for the SLICE operator: it is not possible to slice a dimension on a FOLAP cuboid associated to an SDField, since $|\mathcal{D}| = 1$. Also, to slice a measure, the FOLAP cuboid must have at least two measures, i.e., no slice on measures is allowed when $|\mathcal{M}| = 1$.

Spatial or Temporal Slice on a FOLAP Cuboid Associated to an STDField

Since SLICE is an IGO operator, a new FOLAP cube instance is induced, and thus, a new associated STDField is built. Basically, when slicing dimensions, the new STDField is obtained through a previous ROLL-UP to ALL in the dimension to be removed, after which the dimension is eliminated.

When slicing a spatial dimension in a FOLAP cuboid associated to an STDField, the latter loses its SpatialDim dimension, becoming a time series (composed of temporal intervals). Since time series satisfy neither our definition of SDField, nor the STDField one, we do not address this case of the SLICE operation in this use case, although time series (or unidimensional fields) can be supported by the model.

In the case of temporal slicing, the TemporalDim dimension is dropped, and the field becomes an SDField, where each sampled tuple aggregates the sampled values at the same location for all the SDFields of the Sequence.

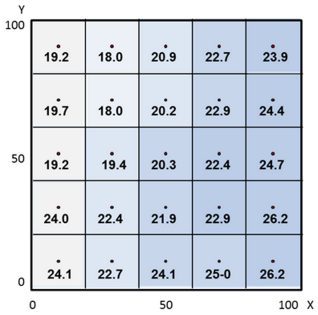
Example 59 (Slice of TemporalDim on a FOLAP cuboid)**.** Consider the FOLAP cuboid $\mathsf{TempF}_{\mathsf{bottom}}$ given in Example 48. Figure 8.18 shows the output FOLAP cuboid and its associated DField after applying SLICE($\mathsf{TempF}_{\mathsf{bottom}}$,TemporalDim).

Notice that, the sampled points in the resulting DField are those that belong to Temp_6 SDField (the most recent SDField in the sequence), and the TemporalDim dimension is eliminated. In consequence, the input spatio-temporal DField becomes a spatial DField. □

Example 60 (Dice and Slice over a FOLAP cuboid)**.** Given the FOLAP cuboid $\mathsf{TempF}_{\mathsf{bottom}}$ of Example 48, we need to compute its values independently of the

SpatialDim Point point, x, y	Measures (Avg) $value_{temp}$
⟨(10, 90), 10, 90⟩	19.2
⟨(30, 90), 30, 90⟩	18.0
⟨(50, 90), 50, 90⟩	20.9
⟨(70, 90), 70, 90⟩	22.7
⟨(90, 90), 90, 90⟩	23.9
⟨(10, 70), 10, 70⟩	19.7
⟨(30, 70), 30, 70⟩	18.0
⟨(50, 70), 50, 70⟩	20.2
⟨(70, 70), 70, 70⟩	22.9
⟨(90, 70), 90, 70⟩	24.4
⟨(10, 50), 10, 50⟩	19.2
⟨(30, 50), 30, 50⟩	19.4
⟨(50, 50), 50, 50⟩	20.3
⟨(70, 50), 70, 50⟩	22.4
⟨(90, 50), 90, 50⟩	24.7
⟨(10, 30), 10, 30⟩	24.0
⟨(30, 30), 30, 30⟩	22.4
⟨(50, 30), 50, 30⟩	21.9
⟨(70, 30), 70, 30⟩	22.9
⟨(90, 30), 90, 30⟩	26.2
⟨(10, 10), 10, 10⟩	24.1
⟨(30, 10), 30, 10⟩	22.7
⟨(50, 10), 50, 10⟩	24.1
⟨(70, 10), 70, 10⟩	25.0
⟨(90, 10), 90, 10⟩	26.2

(a) TempF_2 FOLAP cuboid



(b) $\mathsf{Temperature}_2$ SDField

Figure 8.18: $\mathsf{TempF}_2 = \textsc{Slice}(\mathsf{TempF}_{\mathsf{bottom}}, \mathsf{TemporalDim})$.

time dimension such that we want only to keep those samples located at an x coordinate greater than or equal to 20 km and with temperature less than 24^oC. To remove the temporal dimension we apply the Slice operator and to select the samples that satisfy the required condition, we apply the Dice operator. The query reads:

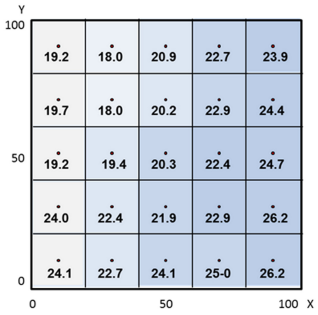
$\mathsf{TempF}_4 = \textsc{Slice}(\mathsf{TempF}_{\mathsf{bottom}}, \mathsf{TemporalDim})$

$\mathsf{TempF}_5 = \textsc{Dice}(\mathsf{TempF}_4, \mathsf{x} \geq 20 \text{ and } \mathsf{v} < 24)$.

Figure 8.19 depicts the partial and final results. □

SpatialDim Point point, x, y	Measures (Avg) $value_{temp}$
⟨(50.0,10.0), 50.0, 10.0⟩	25.0
⟨(70.0,30.0), 70.0, 30.0⟩	19.0
...	...
⟨(90.0,10.0), 90.0, 10.0⟩	26.0
...	...
⟨(30.0,10.0), 30.0,10.0⟩	20.0
⟨(50.0,10.0), 50.0,10.0⟩	22.0
...	...
⟨(90.0,10.0), 90.0,10.0⟩	26.0
...	...
⟨(10.0,10.0), 10.0,10.0⟩	22.0
⟨(30.0,10.0), 30.0,10.0⟩	22.0
⟨(50.0,10.0), 50.0,10.0⟩	26.0
...	...
⟨(10.0,10.0), 10.0,10.0⟩	23.0
⟨(30.0,10.0), 30.0,10.0⟩	23.0
⟨(90.0,90.0), 90.0,90.0⟩	26.0

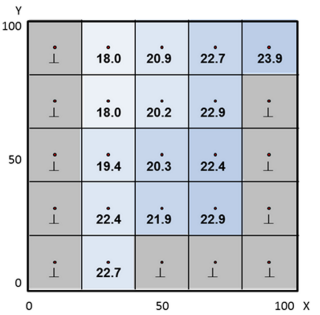
(a) TempF_4 FOLAP cuboid



(b) $\mathsf{Temperature}_4$ SDField after slicing

SpatialDim Point point, x, y	Measures (Avg) $value_{temp}$
⟨(70.0,30.0), 70.0, 30.0⟩	19.0
...	...
⟨(30.0,10.0), 30.0,10.0⟩	20.0
⟨(50.0,10.0), 50.0,10.0⟩	22.0
...	...
⟨(30.0,10.0), 30.0,10.0⟩	22.0
...	...
⟨(30.0,10.0), 30.0,10.0⟩	23.0

(c) TempF_5 FOLAP cuboid



(d) $\mathsf{Temperature}_5$ SDField after dicing

Figure 8.19: DICE and SLICE over $\mathsf{TempF}_{\mathsf{bottom}}$ (Example 60).

Slicing Measures

When a measure is sliced, besides removing it from each cuboid of the FOLAP cube of the input cuboid, a new output associated DField is built by simply eliminating the sampled value corresponding to this measure in each sampled tuple.

8.2.4 Drill-down over FOLAP Cuboids

As explained in previous sections, when an IGO operation is applied over a FOLAP cuboid, the DField associated to the bottom cuboid of the cube needs to be updated. This requirement is needed to keep consistency in the cube navigation

when a drill-down operation is called. For example, if a dice operation only keeps temperatures in Belgium, when the user drills down to find out temperatures in cities she must only see the temperatures in cities in Belgium.

DRILL-DOWN($\mathsf{C_{in}}, \mathsf{D}, \mathsf{L}$), where $\mathsf{C_{in}}$ is a FOLAP cuboid associated to a DField $\mathsf{F_{in}}$, returns a cuboid $\mathsf{C_{out}}$ by means of computing ROLL-UP($\mathsf{C_{bottom}}, \mathsf{D}, \mathsf{L}$), where $\mathsf{C_{bottom}}$ is the bottom cuboid of the cube instance corresponding to $\mathsf{C_{in}}$.

Example 61 (Drill-down on pH FOLAP Cuboid)**.** Consider the FOLAP cuboid $\mathsf{pHF_4}$ obtained in Example 58, after rolling-up to Region and then dicing areas less than 100 $\mathsf{km^2}$ on $\mathsf{pHF_{bottom}}$.

Figure 8.20 shows the output FOLAP cuboid $\mathsf{pHF_5}$ and its associated DField after applying DRILL-DOWN($\mathsf{pHF_4}$,SpatialDim,Point).

The cube semantics we defined (that is, to perceive the cube as a collection of cuboids), allows a DRILL-DOWN to be computed as a ROLL-UP from the bottom cuboid to the operation's input level. In this case the target level is Point and it coincides with the bottom level, thus, in this case there is nothing else to do.

Notice that the cells in the output FOLAP cuboid correspond to the new tessellation induced by the DICE on $\mathsf{pHF_{Region}}$, each preserving its original value from pH DField. Thus, if we roll-up to Region again, the value of 'Region-C' computed based on the DField associated to the diced $\mathsf{pHF_5}$ FOLAP cube is 6.7 (the same value obtained for this region in $\mathsf{pHF_4}$). □

8.2.5 Drill-Across over FOLAP Cuboids

We have two main possibilities for the Drill-across operations, each one having in turn, other variants:

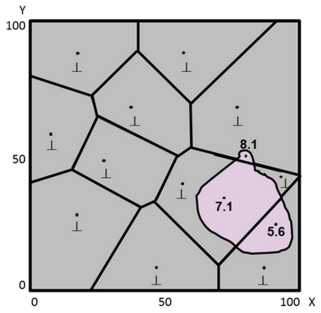
1. Both input cuboids are FOLAP cuboids
   (a) Both of them are bottom cuboids
   (b) Both of them are non-bottom cuboids, i.e., they come from a previous roll-up operation

SpatialDim Region rName, rGeom	Measures (Avg) $value_{pH}$
$\langle$'$Region - C'$, $geom703\rangle$	6.7

(a) $\mathsf{pHF}_{\mathsf{Region}}$ FOLAP cuboid

SpatialDim Point point, x, y	Measures (Avg) $value_{pH}$
$\langle(70, 30), 70, 30\rangle$	7.1
$\langle(80, 50), 80, 50\rangle$	8.1
$\langle(90, 25), 90, 25\rangle$	5.6

(b) pHF_5 FOLAP cuboid



(c) $\mathsf{pH}_4\mathsf{Region}$ SDField

Figure 8.20: $\mathsf{pHF}_5 = \textsc{Drill-down}(\mathsf{pHF}_{\mathsf{Region}}, \mathsf{SpatialDim}, \mathsf{Point})$.

2. One input cuboid is a typical OLAP cuboid, and the other one is a FOLAP cuboid

   (a) The first parameter is an OLAP cuboid and the second parameter is a bottom FOLAP cuboid

   (b) The first parameter is an OLAP cuboid and the second parameter is a non-bottom FOLAP cuboid

   (c) The first parameter is a FOLAP cuboid and the second parameter is an OLAP cuboid

Since Drill-across is an IGO operator, a new OLAP cube or FOLAP cube instance is induced, and a new associated DField may be produced. We next detail the implementation of the operator at the logical level for each situation.

Drill-Across between two Bottom FOLAP Cuboids

This is the simplest case. The result is computed as a generic Local operation (Section 7.4.1) as follows. $\textsc{Drill-across}(\mathsf{C}_1, \mathsf{C}_2)$, where C_1 and C_2 are

two (bottom) FOLAP cuboids such that their associated DFields F_1 and F_2 are domain compatible, returns a FOLAP cuboid $\mathsf{C}_{\mathsf{out}}$ which induces a new cube instance according to the semantics of Definition 19. $\mathsf{C}_{\mathsf{out}}$ is computed from a DField obtained as $\mathsf{F}_{\mathsf{out}} = \textsc{Local}(\textsc{Fl}, \mathsf{F}_1, \mathsf{F}_2)$, where $\textsc{Fl} : \mathsf{Range}(\mathsf{F}_1) \times \mathsf{Range}(\mathsf{F}_2) \rightarrow \mathsf{Range}(\mathsf{F}_1) \times \mathsf{Range}(\mathsf{F}_2)$ such that $\textsc{Fl}(\mathsf{a},\mathsf{b}) = (\mathsf{a},\mathsf{b})$ (i.e., a vector composed of both values). According to the definition of the generic operation, the tesselation of the result is, by convention, the tesselation of C_1. The same occurs with the measures: the first component of the DField vector value is the one of C_1, and second measure is the one of C_2.

Example 62 (Drill-across between two bottom FOLAP cuboids)**.** Given the SDFields of Example 37, NDVI and pH, and their corresponding FOLAP cubes, Figure 8.21 depicts the result of $\textsc{Drill-across}(\mathsf{NDVIF}_{\mathsf{bottom}}, \mathsf{pHF}_{\mathsf{bottom}})$, and its associated DField. □

Drill-Across between Two Non-bottom FOLAP Cuboids

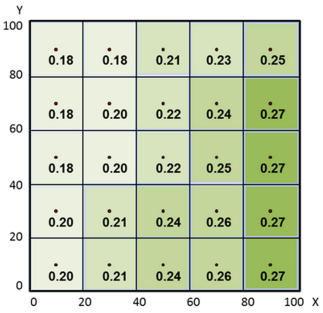
In this case, both input FOLAP cuboids come from a previous Roll-up. $\textsc{Drill-across}(\mathsf{C}_1, \mathsf{C}_2)$, where C_1 and C_2 are two non-bottom FOLAP cuboids such that $\mathsf{C}_1 \simeq \mathsf{C}_2$, returns a cuboid $\mathsf{C}_{\mathsf{out}}$ according to Definition 19. Note that this is an OLAP cuboid without an associated DField.

Example 63 (Drill-across between two non-bottom FOLAP cuboids)**.** Consider the NDVI and pH SDFields of the Example 37 and their corresponding FOLAP cubes, such that the level Region of Example 51 has been inserted in both FOLAP cuboids. Assume that we first roll-up to the Region level on the bottom cuboids of those cubes, namely, $\mathsf{NDVIF}_{\mathsf{bottom}}$ and $\mathsf{pHF}_{\mathsf{bottom}}$. Then, we can drill-across the resulting cuboids as follows:

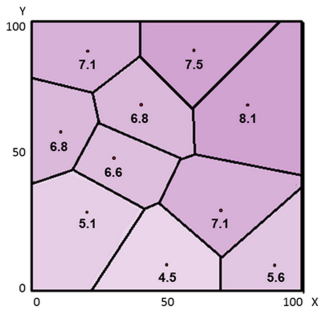
$\mathsf{NDVIF}_{\mathsf{Region}} = \textsc{Roll-up}(\mathsf{NDVIF}_{\mathsf{bottom}}, \mathsf{SpatialDim}, \mathsf{Region})$

$\mathsf{pHF}_{\mathsf{Region}} = \textsc{Roll-up}(\mathsf{pHF}_{\mathsf{bottom}}, \mathsf{SpatialDim}, \mathsf{Region})$

$\textsc{Drill-across}(\mathsf{NDVIF}_{\mathsf{Region}}, \mathsf{pHF}_{\mathsf{Region}})$.



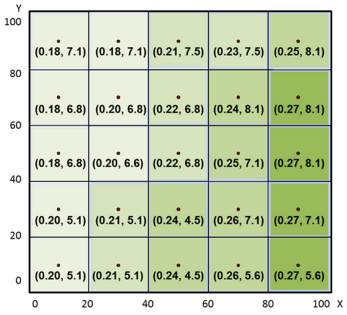
(a) NDVI SDField



(b) pH SDField

SpaceDim Point point, x, y	Measures (Avg) $value_{ndvi}$	(Avg) $value_{pH}$
⟨(10.0,90.0), 10.0, 90.0⟩	0.18	7.1
⟨(30.0,90.0), 30.0, 90.0⟩	0.18	7.1
⟨(50.0,90.0), 50.0, 90.0⟩	0.21	7.5
...	...	...
⟨(10.0,70.0), 10.0, 70.0⟩	0.18	6.8
⟨(30.0,70.0), 30.0, 70.0⟩	0.20	6.8
⟨(50.0,70.0), 50.0, 70.0⟩	0.22	6.8
...	...	...
⟨(10.0,30.0), 10.0, 30.0⟩	0.20	5.1
⟨(30.0,30.0), 30.0, 30.0⟩	0.21	5.1
⟨(50.0,30.0), 50.0, 30.0⟩	0.24	4.5
...	...	...
⟨(50.0,10.0), 50.0, 10.0⟩	0.24	4.5
⟨(70.0,10.0), 70.0, 10.0⟩	0.26	5.6
⟨(90.0,10.0), 90.0, 10.0⟩	0.27	5.6

(c) NDVIF_2 FOLAP cuboid



(d) NDVI_2 SDField

Figure 8.21: $\mathsf{NDVIF}_2 = \textsc{Drill-Across}(\mathsf{NDVIF}_{\mathsf{bottom}}, \mathsf{pHF}_{\mathsf{bottom}})$.

After drilling-across, the SpatialDim dimension lattice in the output cuboid is Region→All. Figure 8.22 shows the output FOLAP cuboids of the rolling-up operations and the resulting cuboid of the final Drill-across. □

Drill-Across between an OLAP Cuboid and a Bottom FOLAP Cuboid

When $\textsc{Drill-across}(\mathsf{C}_1, \mathsf{C}_2)$ receives a typical OLAP as first parameter and a bottom FOLAP cuboid as second parameter, and there is a semantic mapping between their associated cubes, such that $\mathsf{C}_1 \simeq \mathsf{C}_2$, the output cuboid is an OLAP cuboid with the union of the measures of the FOLAP cuboid, according to Definition 19.

SpatialDim Region rName, rGeom	Measures (Avg) $value_{ndvi}$
$\langle$'$Region-A'$, $geom506\rangle$	0.22
$\langle$'$Region-B'$, $geom701\rangle$	0.21
$\langle$'$Region-C'$, $geom703\rangle$	0.26

(a) $\mathsf{NDVIF}_{\mathsf{Region}}$ FOLAP cuboid

SpatialDim Region rName, rGeom	Measures (Avg) $value_{pH}$
$\langle$'$Region-A'$, $geom506\rangle$	7.4
$\langle$'$Region-B'$, $geom701\rangle$	5.4
$\langle$'$Region-C'$, $geom703\rangle$	6.7

(b) $\mathsf{pHF}_{\mathsf{Region}}$ FOLAP cuboid

SpatialDim Region rName, rGeom	Measures (Avg) $value_{ndvi}$	(Avg) $value_{pH}$
$\langle$'$Region-A'$, $geom506\rangle$	0.22	7.4
$\langle$'$Region-B'$, $geom701\rangle$	0.21	5.4
$\langle$'$Region-C'$, $geom703\rangle$	0.26	6.7

(c) NDVIF_3 OLAP cuboid

$NDVI_{All}$

SpatialDim	Measures	
All	Avg	Avg
all	$value_{ndvi}$	$value_{pH}$
<all>	0.22	6,5

$NDVI_{Region}$

SpatialDim	Measures	
Region	Avg	Avg
rName, gName	$value_{ndvi}$	$value_{pH}$
<'Region-A', geom506>	0.22	7.4
<'Region-B', geom701>	0.21	5.4
<'Region-C', geom703>	0.26	6.7

(d) Output OLAP cube instance

Figure 8.22: $\mathsf{NDVIF}_3 = \textsc{Drill-Across}(\mathsf{NDVIF}_{\mathsf{Region}}, \mathsf{pHF}_{\mathsf{Region}})$.

Since the bottom FOLAP cuboid has levels Point and Time, if $\mathsf{C}_1 \simeq \mathsf{C}_2$ then the OLAP cubes must have data representing isolated spatial points or timestamps (call them p_i and t_i, respectively. In consequence, to drill-across the two cuboids, the values of the points in the OLAP cuboid C_1 (e.g., p_i) are matched against the sampled points of the associated DField in the FOLAP cuboid C_2. If they do not match, they can be inferred by taking advantage of the continuous nature of its domain.

Example 64 (Drill-across between an OLAP and a bottom FOLAP cuboid)**.** A cube with schema $\langle$BaseStation, {TimeDim, AntennaDim}, {radiatedPower}$\rangle$ registers the radiated power of base stations' mobile antennas in a city. Assume that the TimeDim dimension is the one in Example 1, and the AntennaDim dimension has the following schema: $\langle$AntennaDim, {$\langle$AM-Antenna, $\langle$x, y, h, azimut, tilt, model$\rangle\rangle$, $\langle$All, $\langle$all$\rangle\rangle\}, \rightarrow\rangle$, where the lattice is AM-Antenna$\rightarrow$All, with Dom(x) = Dom(y) = Dom(h) = $\mathbb{R}$, Dom(azimut) = Dom(tilt) = $\mathbb{G}$ and Dom(model) = $\mathbb{S}$. The measure radiatedPower is given in dBm units and its associated aggregate function is Sum. Also consider the TempF FOLAP cuboid given in Example 48. Besides,

there exists a semantic mapping based on the matching of cells at the AM-Antenna and Point levels (given by the equivalence of coordinates x and y in both cubes).

We want to detect a possible correlation between temperatures and the accumulation of radiated power. For this, we aggregate the data of both cubes by month, slice the temporal dimension and then cross the cubes, as follows:

$\mathsf{BaseStation}_{\mathsf{aux1}} = \textsc{Roll-up}(\mathsf{BaseStation}_{\mathsf{bottom}}, \mathsf{TimeDim}, \mathsf{Month})$

$\mathsf{BaseStation}_{\mathsf{aux2}} = \textsc{Slice}(\mathsf{BaseStation}_{\mathsf{aux1}}, \mathsf{TimeDim})$

$\mathsf{TempF}_{\mathsf{aux1}} = \textsc{Roll-up}(\mathsf{TempF}_{\mathsf{bottom}}, \mathsf{TemporalDim}, \mathsf{Month})$

$\mathsf{TempF}_{\mathsf{aux2}} = \textsc{Slice}(\mathsf{TempF}_{\mathsf{aux1}}, \mathsf{TemporalDim})$

Since we have rolled-up and sliced along the temporal dimension, the resulting cuboids remain being bottom cuboids with respect to the spatial dimension, but now there is an SDField associated to $\mathsf{TempF}_{\mathsf{aux2}}$.

Finally, we apply the Drill-across operation:

$\mathsf{BaseStation}_{\mathsf{mix}} = \textsc{Drill-across}(\mathsf{BaseStation}_{\mathsf{aux2}}, \mathsf{TempF}_{\mathsf{aux2}})$.

Figure 8.23 shows the final output OLAP cuboid. Notice that when an antenna location (x,y) does not match any sampled point in $\mathsf{TempF}_{\mathsf{aux2}}$, we infer its temperature value by using the Fn function of the associated SDField, e.g., the location (25.8, 35.3) matches the cell (30, 30), so we can put together the pair of measures (44, 22.4), in the first tuple of the $\mathsf{BaseStation}_{\mathsf{mix}}$ cuboid. □

Drill-Across between an OLAP Cuboid and a Non-Bottom FOLAP Cuboid

When $\textsc{Drill-across}(\mathsf{C}_1, \mathsf{C}_2)$ receives a typical OLAP cuboid as its first parameter, and a non-bottom FOLAP cuboid as its second parameter, with $\mathsf{C}_1 \simeq \mathsf{C}_2$, when there is a Semantic Mapping between their cubes, the output cuboid is an OLAP cuboid that puts together the measures of both cuboids according to the semantics of Definition 19.

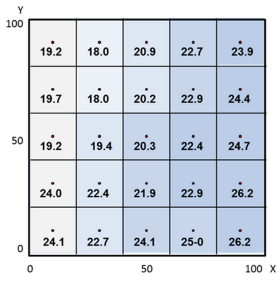
Example 65 (Mixed Drill-across)**.** Consider the $\mathsf{Vineyard}_{\mathsf{bottom}}$ OLAP cuboid given in Example 11 and the $\mathsf{pHF}_{\mathsf{bottom}}$ FOLAP cuboid of Example 62.

AntennaDim AM-Antenna x, y, h, azimut, tilt, model	Measures (Sum) radPower
$\langle 25.8, 35.3, 45, 35, 15, \text{`}multiband'\rangle$	44
$\langle 30.0, 70.0, 45, 35, 15, \text{`}multiband'\rangle$	45
$\langle 45.8, 73.0, 45, 35, 15, \text{`}monoband'\rangle$	43
...	...
$\langle 95.8, 41.3, 45, 35, 15, \text{`}multiband'\rangle$	46

(a) $\mathsf{BaseStation}_{\mathsf{aux2}}$ OLAP cuboid

SpatialDim Point point, x, y	Measures (Avg) $value_{temp}$
⟨(30, 90), 30, 90⟩	18.0
...	...
⟨(30, 70), 30, 70⟩	18.0
⟨(50, 70), 50, 70⟩	20.2
...	...
⟨(30, 30), 30, 30⟩	22.4
⟨(50, 30), 50, 30⟩	21.9
⟨(70, 30), 70, 30⟩	22.9
⟨(30, 10), 30, 10⟩	22.7

(b) $\mathsf{TempF}_{\mathsf{aux2}}$ FOLAP cuboid



(c) $\mathsf{Temperature}_{\mathsf{aux2}}$ SDField

AntennaDim AM-Antenna x, y, h, azimut, tilt, model	Measures (Sum) radPower	Measures (Avg) $value_{temp}$
$\langle 25.8, 35.3, 45, 35, 15, \text{`}multiband'\rangle$	44	22.4
$\langle 30.0, 70.0, 45, 35, 15, \text{`}multiband'\rangle$	45	18.0
$\langle 45.8, 73.0, 45, 35, 15, \text{`}monoband'\rangle$	43	20.2
...	...	...
$\langle 95.8, 41.3, 45, 35, 15, \text{`}multiband'\rangle$	46	24.7

(d) $\mathsf{BaseStation}_{\mathsf{mix}}$ OLAP cuboid

Figure 8.23: $\mathsf{BaseStation}_{\mathsf{mix}} = \textsc{Drill-across}(\mathsf{BaseStation}_{\mathsf{aux2}}, \mathsf{TempF}_{\mathsf{aux2}})$.

Suppose an $\text{SM}(\mathsf{Vineyard}_{\mathsf{bottom}}, \mathsf{pHF}_{\mathsf{bottom}})$ is defined, where $(\{\mathsf{BlockDim.Region}\}, \{\mathsf{SpatialDim.region}\}) \in \mathcal{P}$ and $\textsc{Smap}^{\{\mathsf{SpatialDim.region}\}}_{\{\mathsf{BlockDim.Region}\}} = \{(\langle \text{`Flanders'}\rangle, \langle \text{`Region-A'}\rangle), (\langle \text{`Wallonia'}\rangle, \langle \text{`Region-B'}\rangle), (\langle \text{`Brussels Capital'}\rangle, \langle \text{`Region-C'}\rangle)\}$.

That means, we have a mapping at the level of regions. We can then look for a correlation between the production and the pH of the regions. We first slice temporal dimension in Vineyard because this dimension is not present in the Semantic Mapping. Then we roll-up both cuboids to the level Region, and finally we drill-across the resulting cuboids, as follows:

$\mathsf{Vineyard}_{\mathsf{aux}} = \textsc{Slice}(\mathsf{Vineyard}_{\mathsf{bottom}}, \mathsf{TimeDim})$

$\mathsf{Vineyard}_{\mathsf{Region}} = \textsc{Roll-up}(\mathsf{Vineyard}_{\mathsf{aux}}, \mathsf{BlockDim}, \mathsf{Region})$

$\mathsf{pHF}_{\mathsf{Region}} = \textsc{Roll-up}(\mathsf{pHF}_{\mathsf{bottom}}, \mathsf{SpatialDim}, \mathsf{Region})$

$\mathsf{Vineyard}_{\mathsf{mix}} = \textsc{Drill-across}(\mathsf{Vineyard}_{\mathsf{Region}}, \mathsf{pHF}_{\mathsf{Region}})$.

Figure 8.24 shows the output cuboids of the ROLL-UP operations and the resulting cuboid of the final DRILL-ACROSS. Note for example, that the matching regions 'Flanders' and 'A' have their measures (37,200,7.4) together in the same cell of the output cuboid. □

BlockDim Block idBlock, bGeom	Measures (Sum) harvest
⟨35001, g35001⟩	7200
⟨35002, g35002⟩	7500
⟨35003, g35003⟩	7300
⟨35004, g35004⟩	7200
⟨35005, g35005⟩	7400
...	...
⟨35086, g35006⟩	7900

(a) $\mathsf{Vineyard}_{\mathsf{aux}}$ OLAP cuboid

BlockDim Region rName, rGeom	Measures (Sum) harvest
⟨'Flanders'⟩	372000
⟨'Wallonia'⟩	275000
⟨'Brussels Capital'⟩	138000

(b) $\mathsf{Vineyard}_{\mathsf{Region}}$ OLAP cuboid

SpatialDim Region rName, rGeom	Measures (Avg) $value_{pH}$
$\langle$'$Region - A'$, $geom506\rangle$	7.4
$\langle$'$Region - B'$, $geom701\rangle$	5.4
$\langle$'$Region - C'$, $geom703\rangle$	6.7

(c) $\mathsf{pHF}_{\mathsf{Region}}$ FOLAP cuboid

BlockDim Region rName, rGeom	Measures (Sum) harvest	Measures (Avg) $value_{pH}$
⟨'Flanders'⟩	372000	7.4
⟨'Wallonia'⟩	275000	5.4
⟨'Brussels Capital'⟩	138000	6.7

(d) $\mathsf{Vineyard}_{\mathsf{mix}}$ OLAP cuboid

Figure 8.24: $\mathsf{Vineyard}_{\mathsf{mix}}$ = DRILL-ACROSS($\mathsf{Vineyard}_{\mathsf{Region}}$, $\mathsf{pHF}_{\mathsf{Region}}$).

Drill-Across between a FOLAP Cuboid and a OLAP Cuboid

We consider that when DRILL-ACROSS($\mathsf{C}_1, \mathsf{C}_2$) receives a FOLAP cuboid as its first parameter and an OLAP cuboid as its second parameter, it is not possible to cross the cuboids because, since DRILL-ACROSS is an IGO operator, if a measure is added to a FOLAP cube, the associated DField should incorporate this measure into the corresponding sampled tuples. Although this appears to be simple, the problem is that as a DField has the ability to infer values that are not present among the sampled tuples, interpolate values that do not come from a continuous domain will induce errors.

8.3 Summary

We have described how the concepts studied in previous chapters are applied to continuous fields, allowing us to manipulate such fields in the same way as we do with OLAP cuboids. For this, we made use of the notion of discretized field, which allowed us to implement at the logical level, the OLAP operators defined at a higher abstraction level. We called the data structure we used for this, a FOLAP cube, which we defined as a lattice of cuboids such that the bottom cuboid in the lattice contains the sampled tuples of an associated DField. At the conceptual level, the FOLAP cube behaves like a traditional OLAP cube and supports all the OLAP operations. Finally, we have detailed the computation of each OLAP operator over FOLAP cuboids at the logical level. The basic idea on which we built our solution is that continuous data can be discretized and represented as a cube in a seamless fashion.

Chapter 9

GOLAP-QL: A Generic OLAP Query Language

In this chapter we present a query language based on the algebra presented in Chapter 4, which allows us to manipulate cubes regardless their underlying representation. We call this language GOLAP-QL. GOLAP-QL is a procedural scripting language equipped with the necessary data types that the user needs to define the domains of the level descriptors and measures. That means, the language can be easily extended with the necessary data types. We describe the query processing of a GOLAP-QL query, including syntactic and semantic analysis, as well as the query optimization process, which includes a set of rules that the optimizer can apply to make query execution more efficient.

9.1 GOLAP-QL Syntax

The syntax of GOLAP-QL query language is depicted in Listing 9.1, using Backus-Naur Form with regular expressions. The non-terminal start symbol is <Pgm>. The language includes the following *terminal symbols*:

- QUOTEDLITERAL$: Text enclosed in single quotes. For Geometric Data we use the Well-Known Text (WKT) notation of the OpenGIS specification. For example, 'POINT(50 0)', 'LINESTRING(30 0,10 1)' and 'POLYGON((20 10,4 0,4 4,0 4,20 10),(12 1, 2 1, 2 2, 1 2,12 1))' are WKT valid

expressions. Similarly, MULTIPOINT, MULTILINESTRING and MULTIPOLYGON can be expressed as quoted literals.

- INTEGERLITERAL$: Integer numbers.
- DECIMALLITERAL$: Real numbers expressed in decimal notation.
- OPENBRACKET$ and CLOSEBRACKET$: Tokens for brackets.
- COMMA$, SEMICOLON$: Token for comma and semicolon separator.
- SINGLEQUOTE$: The ‘’’ token.
- Tokens for GOLAP Operators: DRILLACROSS, ROLLUP, DRILLDOWN, SLICE and DICE.
- Tokens for boolean operators: AND, OR, NOT.
- Tokens for relational operators: <, <=, >, >=, =, ! =.
- ASSIGNMENT$: The ‘:=’ token.
- IDENTIFIER$: Non-quoted text that must be different from any of the previous tokens. For example, SpatialDim is an identifier used for referring to the name of a dimension.

The system was implemented over the Mondrian[1] engine. The implementation is described in Chapter 10. We extended the Mondrian specification to declare the extra features required by SOLAP and field data cubes which are not supported in Mondrian:

1. The Geometry data type for members and/or measures (for SOLAP cubes);
2. The underlying DField (for field cubes).

[1] MondrianOLAP,http://community.pentaho.com/projects/mondrian

The syntax of the cube schemas (including dimension schemas) is the one of MultiDimensional eXpressions (MDX)[2]. There is an initial XML document containing the cube schemas and the semantic mappings between cubes. The syntax of Semantic Mappings between cubes, expressed as a DTD, is shown in Listing 9.2.

Listing 9.1: GOLAP QL Syntax

```
1 <Pgm> := <AssignStm> ( SEMICOLON <AssignStm> )*
2 <AssignStm> := IDENTIFIER ASSIGNMENT$ <OLAPExpression>
3 <OLAPExpression> := <DRILLACROSSExpression> | <ROLLUPExpression> |
4              <DRILLDOWNExpression> | <SLICEExpression> | <DICEExpression>
5 <Literal> := QUOTEDLITERAL | INTEGERLITERAL | DECIMALLITERAL
6 <Parameter> := <IDExp> | QUOTEDLITERAL | <OLAPExpression>
7 <IDExp> := IDENTIFIER
8 <DRILLACROSSExpression> := DRILLACROSS OPENBRACKET$
9       <Parameter> COMMA$ <Parameter> ( COMMA$ SINGLEQUOTE$ ExtraParam
            SINGLEQUOTE$ )? CLOSEBRACKET$
10 <ROLLUPExpression> := ROLLUP OPENBRACKET$ <Parameter> COMMA$
       IDENTIFIER COMMA$ IDENTIFIER CLOSEBRACKET$
11 <DRILLDOWNExpression> := DRILLDOWN OPENBRACKET$ <Parameter> COMMA$
       IDENTIFIER COMMA$ IDENTIFIER CLOSEBRACKET$
12 <SLICEExpression> := SLICE OPENBRACKET$ <Parameter> COMMA$ IDENTIFIER
       CLOSEBRACKET$
13 <DICEExpression> := DICE OPENBRACKET$ <Parameter> COMMA$ <
       BooleanExpression> CLOSEBRACKET$
14 <BooleanExpression> := <BooleanTerm> ( <BinaryOp> <BooleanTerm>)*
15 <BinaryOp> := AND | OR
16 <BooleanTerm> := NOT ? ( IDENTIFIER <RelOperator> <Literal>
17      | NOT ? OPENBRACKET$ <BooleanExpression> CLOSEBRACKET$
18      | NOT ? ( IDENTIFIER OPENBRACKET$ IDENTIFIER CLOSEBRACKET$ <
            RelOperator> <Literal>
19 <ExtraParam> := <FirstPart> | <SecondPart> | <FirstPart><SecondPart>
20 <FirstPart> := <RenamingRule> ( COMMA$ <RenamingRule> )*
21 <RenamingRule> := (1. | 2.) IDENTIFIER$ AS IDENTIFIER$
22 <SecondPart> := <AddingMeasure> ( COMMA$ <AddingMeasure> )*
23 <AddingMeasure> := fn:IDENTIFIER$ OPENBRACKET$ IDENTIFIER ( COMMA$
       IDENTIFIER )* CLOSEBRACKET$ AS IDENTIFIER$ WITH IDENTIFIER$
```

[2] http://mondrian.pentaho.com/documentation/mdx.php

Listing 9.2: Semantic Mapping Syntax

```
1 <!ELEMENT SemanticMapping EMPTY>
2 <!ELEMENT LevelMapping (#PCDATA)>
3 <!ATTLIST SemanticMapping  cube1 #REQUIRED  cube2 #REQUIRED>
4 <!ATTLIST LevelMapping  <BooleanExpression> #REQUIRED>
5 <BooleanExpression> is the same as in Lines 14 trough 18 of Listing 9.1
```

9.2 GOLAP-QL Semantics

A GOLAP-QL query (or program, which we denote a sequence of GOLAP-QL expressions) may be composed of one or more expressions, separated by a semicolon. Identifiers are used to represent cuboids, which can be obtained from a previous operation, while quoted literals are used to indicate *stored* cuboids (see Section 9.1). An assignment "A:=B" allows binding the cuboid B to an identifier A in order to be reused. Each operation returns a cuboid, allowing nested expressions. When several operators are nested, the invocation is applied inside out, that means, starting from the most internal operator. Each resulting cuboid is the input cuboid for the next operator. For example, the nested expression:

$\mathsf{cuboid}_{\mathsf{out}} := \mathsf{Operator}_{\mathsf{n}}(\mathsf{Operator}_{\mathsf{n-1}}(\ldots \mathsf{Operator}_{1}(\mathsf{cuboid}, \mathsf{args}_{1}) \ldots, \mathsf{args}_{\mathsf{n-1}}), \mathsf{args}_{\mathsf{n}});$

is equivalent to the GOLAP-QL program:

$\mathsf{aux}_{1} := \mathsf{Operator}_{1}(\mathsf{cuboid}, \mathsf{args}_{1});$

$\mathsf{aux}_{2} := \mathsf{Operator}_{2}(\mathsf{aux}_{1}, \mathsf{args}_{2});$

...

$\mathsf{aux}_{\mathsf{n-1}} := \mathsf{Operator}_{\mathsf{n-1}}(\mathsf{aux}_{\mathsf{n-2}}, \mathsf{args}_{\mathsf{n-1}});$

$\mathsf{cuboid}_{\mathsf{out}} := \mathsf{Operator}_{\mathsf{n}}(\mathsf{aux}_{\mathsf{n-1}}, \mathsf{args}_{\mathsf{n}});$

where $\mathsf{args}_{\mathsf{i}}$ are the arguments corresponding to $\mathsf{Operator}_{\mathsf{i}}$.

When evaluating nesting operations, each intermediate resulting cuboid is stored in a temporary cuboid $\mathsf{aux}_{\mathsf{i}}$, which is automatically assigned to internal variables to be used as input parameter in further invocations. These auxiliary cuboids $\mathsf{aux}_{\mathsf{i}}$ are temporary and they are eliminated by the garbage collector once

the expression is fully evaluated. Further, if the user needs to keep some intermediate result for future operations, she must store the operation result in a variable by assigning the resulting cuboid to an identifier.

We next describe how each basic operator is evaluated by the GOLAP engine.

The DICE operator selects values in dimensions or measures that satisfy a boolean condition (see Definition 12). When invoking DICE(Cb,ϕ), the first parameter may be an identifier representing a cuboid (possibly obtained from a previous operation), a quoted literal indicating an stored cuboid, or another nested expression. The second parameter is a boolean expression that may include relational operations between a level descriptor of the current levels and a constant, between two level descriptors of the current levels, between a measure descriptor and a constant, an user-function over level descriptors and/or measures. If the condition of the DICE operator contains level descriptors not present in the set $\mathcal{V}_{\mathsf{Cb}}$, it produces an error that must be treated accordingly.

The SLICE operator reduces the dimensionality of a cube by removing one of its dimensions or measures (see Definition 13). When invoking SLICE(Cb,D), the first parameter may be an identifier representing a cuboid (possibly obtained from a previous operation), a quoted literal indicating an stored cuboid, or another nested expression. The second parameter must be an identifier indicating the dimension or measure to be eliminated. If the dimension or measure does not belong to the first parameter schema, it produces an error. The language does not allow eliminating all the dimensions. Thus, if a SLICE is applied to a cube with one dimension, it throws an exception. Finally, note that after slicing a cuboid, it is not possible to use the eliminated dimension or measure in further operations on the resulting cuboid.

The ROLL-UP operator aggregates measures up to a given level into a dimension hierarchy by using the corresponding RUP function given in the instance of the input dimension (see Definition 10). When invoking ROLLUP(Cb, D, L), the first parameter may be an identifier representing a cuboid (perhaps obtained from a previous operation), a quoted literal indicating an stored cuboid, or another

nested expression. The second parameter must be an identifier indicating a selected dimension and the third parameter must be an identifier indicating the target level for the aggregation. This operator checks both, that the input level L belongs to the input dimension D, and the condition $\mathsf{Lc} \preceq \mathsf{L}$, where Lc is the level corresponding to D in the set of levels $\mathcal{V}_{\mathsf{Cb}}$ of the input cuboid.

The DRILL-DOWN operator disaggregates measures down to a given level within a dimension hierarchy by using the corresponding RUP function given in the instance of the input dimension (see Definition 11). When invoking DRILLDOWN(Cb, D, L, the first parameter may be an identifier representing a cuboid (maybe obtained from a previous operation), a quoted literal indicating an stored cuboid, or another nested expression. The second parameter must be an identifier indicating a selected dimension and the third parameter must be an identifier indicating the target level for the disaggregation. This operator checks both that input level L belongs to the input dimension D, and the condition $\mathsf{L} \preceq \mathsf{Lc}$, where Lc is the level corresponding to D in the set of levels $\mathcal{V}_{\mathsf{Cb}}$ of the input cuboid.

As discussed in Chapter 4, DRILL-DOWN does not just undo a previous ROLL-UP (as considered in many proposals reviewed in Chapter 2), due the possibility of nesting operators that may eliminate members or dimensions from the input cuboid during a ROLL-UP from level L1 up to L2 and its corresponding DRILL-DOWN from level L2 down to L1. If we just undo this roll-up without considering the intermediate changes that may have occurred, after the DRILL-DOWN the user will see members that have been eliminated in previous steps. For example, if the user applies a ROLL-UP to the Region level, then eliminates the Central region, and finally applies a DRILL-DOWN to the Province level, the semantics that we consider is that she no longer wants to see the provinces belonging to the Central region. Example 66 illustrates this in detail.

The DRILL-ACROSS operator performs a join between two cuboids in order to give a unified view of their measures (see Definition 19). When invoking DRILLACROSS(Cb1, Cb2, [NewMeasures]), the first two parameters may be an identifier representing a cuboid (perhaps obtained from a previous operation), a quoted

literal indicating an stored cuboid, or another nested expression. The last parameter is optional but if present, it must be an special quoted literal (see the syntax of ExtraParam in Listing 9.1) representing the operations to be performed to rename measures or to obtain new ones (with their names and aggregate functions). We need this optional parameter because it may occur that some measures in both cubes have the same denomination. To avoid ambiguity, the user must express how the original measures are named in the resulting cube. This strategy can be used just for changing the names of measures. These measures that do not appear in this expression keep their original names. Finally, this expression can be also used to add new measures to the output cuboid. In fact, the complete expression is composed of two (implicit) parts: the first part is used to rename variables, and the second part is used to add new measures. The definition of a new measure is expressed by a function (prefixed by the symbol fn:) whose parameter can be the name of any measure in the resulting cube. The name of the measure being defined and the aggregate function associated to it need also to be expressed. To refer to a measure in order to use it as parameter in the function expression, it must be prefixed with number 1 or number 2, to indicate if it belongs to the first cuboid or to the second cuboid, respectively. If the same measure is renamed multiple times, only the last denomination is considered. The expression can contain as many renaming rules as the user wants. If duplicating measures appear in the resulting cube, the DRILLACROSS throws an exception. Example 67 shows possible expressions for the third optional parameter of the DRILL-ACROSS operation.

The DRILLACROSS operation requires that a semantic mapping exists between both input cuboids involved. This semantic mapping must be defined in the initial XML document, after the creation of the cube schemas. If there is not a previously-defined semantic mapping between the cubes corresponding to the input cuboids, the operation produces an error. In the case of an existing semantic mapping, the operator checks that all the dimensions of each cuboid are involved in the mapping, and if this condition is not satisfied, it produces an error. To enforce the input conditions, the user may previously navigate on one or both cuboids, in order to obtain the semantically compatible cuboids (see Definition 18). Moreover,

the dimensions not included in the mapping must be dropped applying a SLICE operation before performing a DRILLACROSS (see Example 28). The DRILLACROSS operator works like a natural join, i.e., if a cell in the first cuboid does not have a semantic equivalent one in the second cuboid, it will not be included in the output cuboid. Thus, if the input cuboids do not share instances members, the resulting cuboid would be empty, as explained in Section 5.4.

It is important to remark that, after applying DRILLACROSS, it will not be possible to perform a DRILLDOWN operation along the levels that precede the ones in the resulting cuboid (i.e., that precede levels in $\mathsf{L}_{\mathsf{C}_{out}}$), because these levels are not present in the dimensions of the output cube. That is, DRILL-ACROSS induces a new bottom cuboid, as showed in Example 29.

Example 66 (DRILLDOWN and DICE)**.** Consider the cuboid C_1 in Figure 9.1 and the query DRILLDOWN(DICE(ROLLUP(C_1,BlockDim,GrapeType),gType = 'white'), BlockDim,Grape). The inner ROLLUP returns the cuboid C_2 given in Figure 9.2; then, the DICE operation keeps the tuples with grape type white (see Figure 9.3); finally, DRILLDOWN disaggregates the level GrapeType, but since the grape type 'red' has been eliminated, it only returns the corresponding members in Grape. As we can see in Figure 9.4, the resulting cuboid has removed the cells containing white grapes, for example (⟨2011⟩,⟨'pinot noir'⟩) is not present any more. Thus, the DRILLDOWN operation is not a just an undo of the previous ROLLUP, since the final result is differs from C_1. □

Example 67 (DRILL-ACROSS operation)**.** The expression C5 := DRILLACROSS(SLICE('mytempcube', TemporalDim), 'altitudecube', '2.value AS altvalue') indicates that in the resulting cube the measures are value and altvalue. The former belongs to mytempcube, and altvalue belongs to altitudecube.

Also, C6 := DRILLACROSS(SLICE('mytempcube', TemporalDim), 'altitudecube', '2.value AS altvalue, 1.value AS valuetemp; 2.value AS valuealt') indicates that in the resulting cube the measures are valuetemp and valuealt. The former is the measure that belongs to mytempcube and valuealt is the one that belongs to altitudecube.

TimeDim	BlockDim	Measures
Year	Grape	(SUM)
year	gName	harvest
⟨2007⟩	⟨kerner⟩	22000
⟨2007⟩	⟨chardonnay⟩	22600
⟨2008⟩	⟨kerner⟩	23300
⟨2008⟩	⟨chardonnay⟩	22700
⟨2009⟩	⟨kerner⟩	23000
⟨2009⟩	⟨chardonnay⟩	22800
⟨2010⟩	⟨pinot blanc⟩	22600
⟨2010⟩	⟨pinot noir⟩	23800
⟨2011⟩	⟨pinot blanc⟩	22800
⟨2011⟩	⟨pinot noir⟩	23800

Figure 9.1: Cuboid C_1.

TimeDim	BlockDim	Measures
Year	GrapeType	(SUM)
year	gType	harvest
⟨2007⟩	⟨white⟩	44600
⟨2008⟩	⟨white⟩	46000
⟨2009⟩	⟨white⟩	45800
⟨2010⟩	⟨white⟩	22600
⟨2010⟩	⟨red⟩	23800
⟨2011⟩	⟨white⟩	22800
⟨2011⟩	⟨red⟩	23800

Figure 9.2: C_2=ROLLUP(C_1,BlockDim,GrapeType).

Finally, consider C7 := DRILLACROSS(ROLLUP(ROLLUP('Vineyard', BlockDim, Grape), TimeDim, Month), SLICE('Drinks', ZoneDim), '1.harvest AS myharvest, 2.consumption AS sales, fn : percent(myharvest, sales) AS mypercent WITH Sum').

Here we use a third (optional) parameter to add a new measure. We define the percent function, taking as parameters the measures of the input cuboids. The above expression indicates that in the resulting cube the measures are myharvest, consumption and mypercent. The last one is a new measure calculated by the function percent, which is applied over the measures myharvest and sales of the resulting cube, and that will be aggregated with the SUM function. □

9.3 GOLAP Engine Architecture

We now discuss the strategies used by the GOLAP engine to optimize and execute queries expressed in GOLAP-QL.

TimeDim	BlockDim	Measures
Year	GrapeType	(SUM)
year	gType	harvest
⟨2007⟩	⟨white⟩	44600
⟨2008⟩	⟨white⟩	46000
⟨2009⟩	⟨white⟩	45800
⟨2010⟩	⟨white⟩	22600
⟨2011⟩	⟨white⟩	22800

Figure 9.3: C_3=DICE(ROLLUP(C_1,BlockDim,GrapeType),gType='white').

TimeDim	BlockDim	Measures
Year	Grape	(SUM)
year	gName	harvest
⟨2007⟩	⟨kerner⟩	22000
⟨2007⟩	⟨chardonnay⟩	22600
⟨2008⟩	⟨kerner⟩	23300
⟨2008⟩	⟨chardonnay⟩	22700
⟨2009⟩	⟨kerner⟩	23000
⟨2009⟩	⟨chardonnay⟩	22800
⟨2010⟩	⟨pinot blanc⟩	22600
⟨2011⟩	⟨pinot blanc⟩	22800

Figure 9.4: C_4 = DRILLDOWN(DICE(ROLLUP(C_1 , BlockDim , GrapeType) , gType='white') , BlockDim , Grape).

Proposals for OLAP query processing are based on the assumption that data cubes are implemented in relational databases (ROLAP). Typically, OLAP queries are defined at the logical level (i.e., SQL or MDX expressions), rather than at a conceptual level. On the contrary, GOLAP-QL is a high-level language. The engine includes an optimization module that works at the conceptual level. The optimizer can change the canonical GOLAP-QL expression that the user poses. Once this expression is translated from the conceptual to the logical level, the usual optimization strategies are applied by the underlying relational engine. Note that in Examples 66 and 67 the query evaluation process ignored the kinds of underlying data present in the cubes. This is handled at lower levels of abstraction.

The GOLAP architecture is composed of the following modules, shown in Figure 9.5: the Lexical Analyzer, the Syntactic Analyzer, the Semantic Analyzer, the Query Optimizer and the GOLAP Engine.

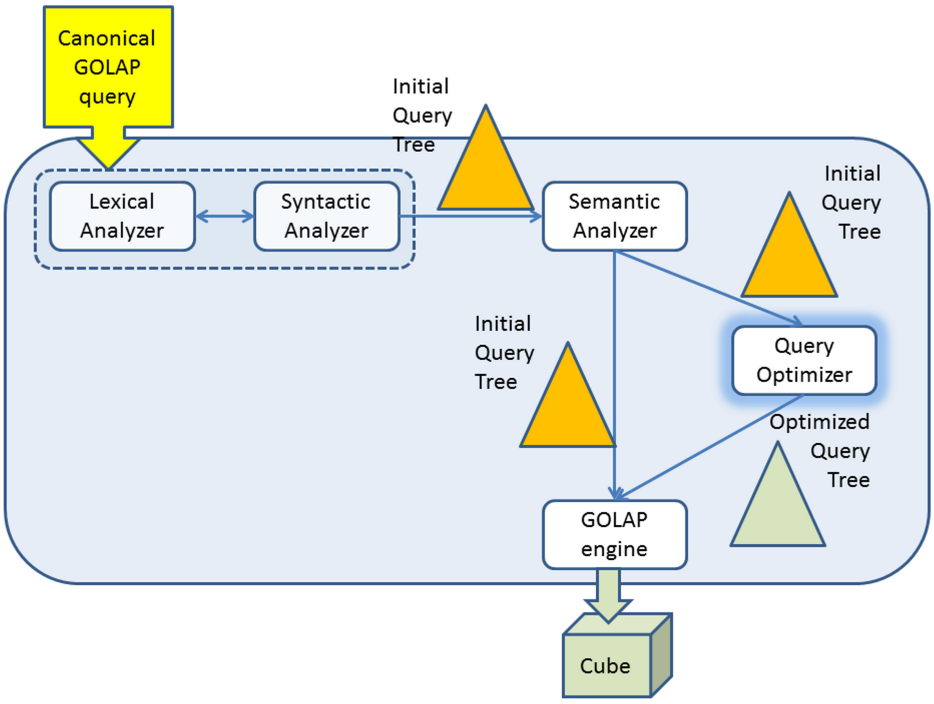


Figure 9.5: GOLAP Engine Architecture.

9.3.1 Lexical and Syntactic Analyzers

The Lexical Analyzer is responsible of identifying the language tokens, while the Syntactic Analyzer checks the syntax according to the grammar described in Section 9.1. If no syntactic errors are detected, a tree data structure (a parse tree) representing the query is built. The implementation details of both analyzers are described in Chapter 10.

Example 68 (Q1-Valid Syntactic GOLAP-QL query)**.** The lexical and syntactic analyzers, applied to the following query, generate the parse tree shown in Figure 9.6.

Result:= SLICE(ROLLUP(DICE(ROLLUP('Vineyard', TimeDim, Quarter), TimeDim. Quarter.quarter = 'Q1-2008'), TimeDim, Year), TimeDim) □

Example 69 (Q2-Valid Syntactic GOLAP-QL query)**.** Given that we can define variables to store cubes that can be used latter in the same script expression, query

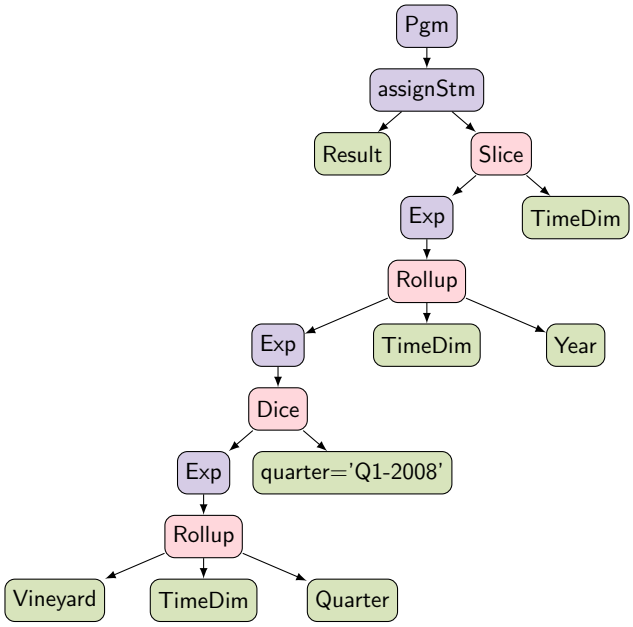


Figure 9.6: Q1 Parse Tree.

Q1 could also be expressed as:
Cube1 :=ROLLUP(Vineyard, TimeDim, Quarter);
Result := SLICE(ROLLUP(DICE(Cube1, TimeDim.Quarter.quarter = 'Q1-2008'), TimeDim, Year), TimeDim)

The lexical and syntactic analyzers generate the parse tree shown in Figure 9.7. □

Example 70 (Q3-Valid Syntactic GOLAP-QL Query)**.** The lexical and syntactic analyzers detect that the following query is a valid syntactic query and builds the Tree shown in Figure 9.8.
Result:= DICE(SLICE('Vineyard',TimeDim),TimeDim.Day.date ='22/12/2008') □

Example 71 (Q4-Invalid Syntactic GOLAP Query)**.** The syntactic analyzer detects that the following query is invalid, since the DICE operator requires two

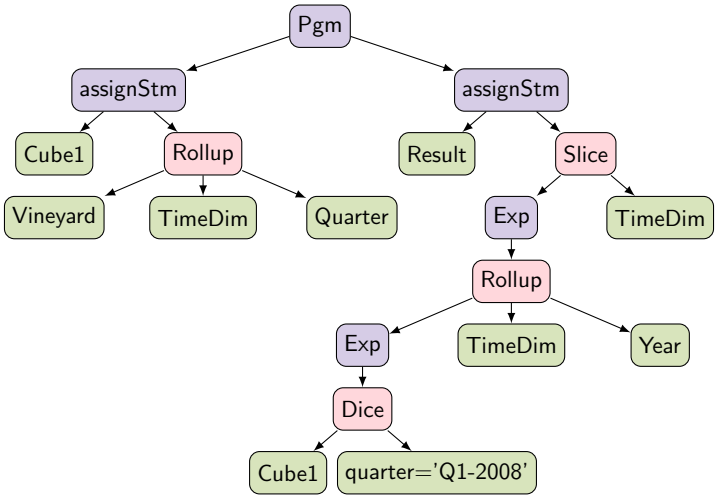


Figure 9.7: Q2 Parse Tree.

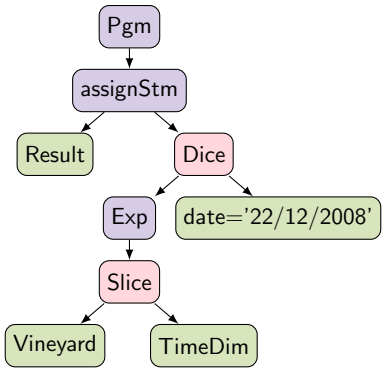


Figure 9.8: Q3 Parse Tree.

parameters: the first one could be a literal or an expression, the second one should be a boolean expression.

Result:=DICE(SLICE('Vineyard',TimeDim),TimeDim.Quarter.quarter)

In this case the second expression is not well-formed, thus, no parse tree is generated and the process finishes throwing an exception. □

9.3.2 Semantic Analyzer

The GOLAP semantic analyzer detects if the query is semantically correct according to the semantics described in Section 9.2. It navigates the parse tree generated in the previous step and determines if the query can be executed. For instance, for each query of Examples 68 through 70, a parse tree was generated, although only the queries of Examples 68 and 69 can be executed, because for the query of Example 70 the semantic analyzer will detect that DICE refers to a dimension (TimeDim) that was dropped by a SLICE operation. To perform this analysis, the schema of the data cubes is used and the parse tree is navigated in a read-only fashion, to detect the order of the operations used in the query. We next explain how the schemas of the data cubes are managed.

Each data cube has a schema that describes its structure, i.e., dimensions, levels, and measures. Some data cubes declare their schema explicitly, and others do not. The former are those cubes that are declared in the XML data file mentioned at the beginning of the chapter. The latter are the cubes that are intermediate expressions generated by the GOLAP operations, i.e., ROLLUP, DRILLDOWN, SLICE, DICE, and DRILLACROSS. For instance, in Example 68, four intermediate data cubes are generated. Three are anonymous, and one is assigned to a variable called Result. In Example 69, also four intermediate data cubes are generated, but only two of them are anonymous (the other two are assigned to variables Cube1 and Result). For the cubes obtained by applying the GOLAP operations, the analyzer needs to infer their structures prior to execute the query. This step is done by navigating the (read-only) parse tree in a bottom-up fashion. Example 72 explains a tree navigation in order to check the semantics of a GOLAP query.

Example 72 (Q2-Semantic Analysis)**.** We next show how the tree generated in Example 69 is Navigated by the semantic analyzer. For clarity, we labeled the nodes with numbers (see Figure 9.9)

First, the ROLLUP node labeled with number 1 is visited, and its structure is inferred using the structure of its parameter (Vineyard), which is defined in the

XML document. Since this operator does not change the structure, the intermediate data cube has the same schema of Vineyard. At this point, it is important to remark that the set of levels $\mathcal{V}_{\mathsf{Vineyard}}$ has been recorded and semantically validated, i.e., it is valid to roll-up from Day (current level of Vineyard) to Year, since a hierarchy Day $\rightarrow$ Month $\rightarrow$ Quarter $\rightarrow$ Year was defined in the schema of the Vineyard data cube. This intermediate data cube is assigned to a variable named Cube1.

Then, the DICE node, labeled with number 2, is visited and its structure is inferred using the schema of Cube1. The DICE operator does not change the schema. The semantic analyzer checks if the DICE expression is semantically correct, i.e., if it uses the levels in $\mathcal{V}_{\mathsf{Cube1}}$.

The third intermediate data cube schema is generated by visiting the ROLLUP node labeled with number 3, and its structure is inferred using the structure of the data cube obtained in node 2. The analyzer checks if the new current level is valid, i.e., if it is possible to roll-up from Quarter to Year.

Finally, the SLICE node labeled with number 4 is visited. The schema of the output data cube to be produced here does no longer contain the sliced dimension.

□

Example 73 (Q3-Semantic Analysis)**.** The query in Example 70 is syntactically correct but semantically incorrect. Figure 9.10 shows the order in which the parse tree is navigated.

First, the SLICE node labeled with number 1 is visited and its structure is inferred by using the structure of its parameter (Vineyard). Since this operator drops the TimeDim dimension, the intermediate resulting data cube does not contain this dimension.

Then, the DICE node labeled with number 2 is visited and its structure is inferred by using the previous schema. The DICE operator does not change the

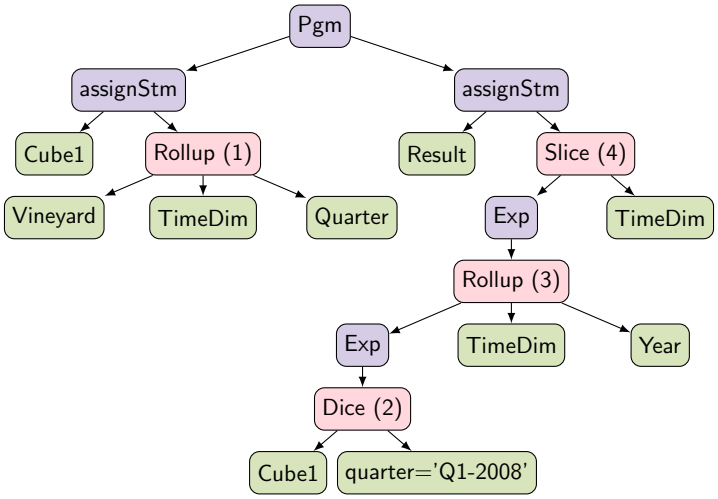


Figure 9.9: Q2 Parse Tree Navigation.

schema. The semantic analyzer checks if the dice expression is semantically correct, i.e., if it uses expression uses current levels. In this case the expression *is not semantically correct* since the expression refers to a level of a non-existent dimension. The analyzer thus throws an exception with the details of the detected problem, and the process ends. □

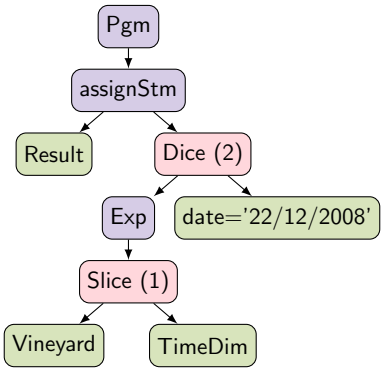


Figure 9.10: Q3 Parse Tree Navigation.

If a query is correct, two phases can follow the semantic analysis, as shown in Figure 9.5: the OLAP Engine or the Query Optimizer can be invoked. The latter transforms the canonical parse tree into another one, that is likely to be more

efficient when the query is executed. We discuss query optimization in the next section.

9.3.3 GOLAP Optimizer

Once the semantic analyzer navigates and accepts the parse tree, the query optimizer can be ran. The GOLAP optimizer takes the parse tree, and, if necessary, modifies it. The canonical tree represents several GOLAP expressions that were submitted by the user in a single script. The optimizer works at the conceptual level, i.e., it modifies the tree which contains internal nodes with GOLAP operators. Since we have implemented the operators over a relational database, traditional strategies can be used by the relational database engine for optimizing the SQL queries that will be eventually executed. The optimizer follows a rule-based approach, consisting of the following rules.

Optimization Rule 1: Commutativity of IPO Operations

Nesting two consecutive IPO operators based on a cuboid Cb *over different dimensions is commutative.* □

Proof. Let us denote as NAV, generically the ROLL-UP and DRILL-DOWN operations:

(1) $\mathsf{A}{:=}\textsc{Nav}_2(\textsc{Nav}_1(\mathsf{Cb},\mathsf{D_p},\mathsf{L_{p_{new}}}),\ \mathsf{D_q},\mathsf{L_{q_{new}}})$

(1) $\mathsf{B}{:=}\textsc{Nav}_1(\textsc{Nav}_2(\mathsf{Cb},\mathsf{D_q},\mathsf{L_{q_{new}}}),\ \mathsf{D_p},\mathsf{L_{p_{new}}})$

Consider that $\mathcal{V}_{\mathsf{Cb}}{=}\{\mathsf{L_1},\ldots,\mathsf{L_p},\ldots,\mathsf{L_q},\ldots,\ \mathsf{L_n}\}$, where $\mathsf{L_p}$ and $\mathsf{L_q}$ are the levels corresponding to the dimensions $\mathsf{D_p}$ and $\mathsf{D_q}$, respectively. Also, consider C the cube containing the cuboid Cb.

The IPO operators only change in the set $\mathcal{V}_{\mathsf{Cb}}$ the level corresponding to the input dimension. Thus, in expression (1), the $\textsc{Nav}_1$ operation sets the level $\mathsf{L_{p_{new}}}$ as the current level, and then $\textsc{Nav}_2$ sets the level $\mathsf{L_{q_{new}}}$. Accordingly, A is the cuboid in C with $\mathcal{V}_A{=}\{\mathsf{L_1},\ldots,\mathsf{L_{p_{new}}},\ldots,\mathsf{L_{q_{new}}},\ \ldots,\ \mathsf{L_n}\}$.

On the other hand, in expression (2), the NAV_2 operation sets the level $\mathsf{L}_{\mathsf{q_{new}}}$ as the current level, and then NAV_1 sets the level $\mathsf{L}_{\mathsf{p_{new}}}$. In consequence, B is the cuboid in C with $\mathcal{V}_B$=$\{\mathsf{L}_1,\ldots,\mathsf{L}_{\mathsf{p_{new}}},\ldots,\mathsf{L}_{\mathsf{q_{new}}},\ \ldots,\ \mathsf{L}_\mathsf{n}\}$.

Since it is not possible to have two cuboids A and B in C cube with the same set of levels, then A=B must hold. □

Optimization Rule 2: Nested IPO Operations

Nesting a series of consecutive IPO operators based on a cuboid Cb *over the same dimension* D *is equivalent to apply a single* ROLL-UP(Cb, D, L) *if* $\mathsf{Cb} \preceq \mathsf{Cb_{out}}$, *or a single* DRILL-DOWN(Cb, D, L) *if* $\mathsf{Cb_{out}} \preceq \mathsf{Cb}$, *where* L *is the level parameter of the last IPO operation.* □

Proof. Let us denote as NAV, generically the ROLL-UP and DRILL-DOWN operations:

(1) A:=NAV(...,NAV(NAV(Cb,D,L_1),D,L_2), ...,D,$\mathsf{L_{last}}$)

(2) B:=ROLL-UP(Cb,D,$\mathsf{L_{last}}$) if $\mathsf{Cb} \preceq \mathsf{A}$ or B:=DRILL-DOWN(Cb,D,$\mathsf{L_{last}}$) if $\mathsf{A} \preceq \mathsf{Cb}$

Consider that $\mathcal{V}_{\mathsf{Cb}}$=$\{\mathsf{L}_1,\ldots,\mathsf{L}_\mathsf{k},\ldots,\ \mathsf{L}_\mathsf{n}\}$, where L_k is the level corresponding to the dimension D, and C is the cube containing the cuboid Cb.

The IPO operators only change the level in $\mathcal{V}_{\mathsf{Cb}}$ corresponding to the input dimension. Thus, in the ith NAV operation, the level L_i is set as the current level. Since the evaluation in nested operations is done from the innermost operator, after successive changes the last change in expression (1) sets the $\mathsf{L_{last}}$ level in $\mathcal{V}_\mathsf{A}$. In consequence, A is the cuboid of C with $\mathcal{V}_A$=$\{\mathsf{L}_1,\ldots,\mathsf{L_{last}},\ldots,\mathsf{L}_\mathsf{n}\}$.

Taking into account if in the lattice of dimension D $\mathsf{L}_\mathsf{k} \rightarrow^* \mathsf{L_{last}}$ or $\mathsf{L_{last}} \rightarrow^* \mathsf{L}_\mathsf{k}$ hold, a ROLL-UP or a DRILL-DOWN over Cb to $\mathsf{L_{last}}$, respectively, sets the $\mathsf{L_{last}}$ level in $\mathcal{V}_\mathsf{B}$. Thus, B is also the cuboid of C with $\mathcal{V}_B$=$\{\mathsf{L}_1,\ldots,\mathsf{L_{last}},\ldots,\mathsf{L}_\mathsf{n}\}$.

Since it is not possible to have two cuboids A and B in C cube with the same set of levels, then A=B holds. □

Optimization Rule 3: Slicing of Nested IPO Operations

Performing a SLICE *operation over a nesting sequence of only IPO operations based on a cuboid* Cb *over the dimension to be removed, is equivalent to apply only the* SLICE *operation over that dimension.* □

Proof. We denote generically NAV the ROLL-UP and DRILL-DOWN operations:
(1) A:=SLICE(NAV(...,NAV(Cb,D,L_1),...,D,L_2),D)
(2) B:=SLICE(Cb,D)

Since SLICE applies an internal ROLL-UP to All in the dimension to be eliminated, we rewrite the expression by making this operation explicit, as follows:
(1) A:=SLICE(ROLL-UP(NAV(...,NAV(Cb,D,L_1), ...,D,L_2), D, All), D)
(2) B:=SLICE(ROLL-UP(Cb,D,All), D)

According to Rule 2, we replace the nesting IPO operations in (1) with a single ROLL-UP, and we obtain: B:=SLICE(ROLL-UP(Cb, D, All), D).

In consequence, the expression B is identical to A. □

Optimization Rule 4: Nesting Dice Operations

Nesting several consecutive DICE *operators based on a cuboid is equivalent to apply a single* DICE *whose condition is the conjunction of all the separate conditions on the same cuboid.* □

Proof. We prove the rule for a sequence of two DICE operations; the same can be applied associatively for the rest.
(1) A:=DICE(DICE(Cb, ϕ_1), ϕ_2)
(2) B:=DICE(Cb, ϕ_1 AND ϕ_2)

Let us call Aux the intermediate cuboid of DICE(Cb, ϕ_1) in expression (1).

If $c_i = (c_{i1}, c_{i2}, \ldots, c_{in}, m_{i1}, \ldots, m_{ik}) \in A$, then c_i satisfies ϕ_2 in the last DICE operation. This implies that c_i was present in Aux, and thus c_i satisfies ϕ_1. Therefore, c_i satisfies both ϕ_1 and ϕ_2, and then c_i satisfies (ϕ_1 AND ϕ_2). In consequence, $c_i \in B$.

Conversely, if $\mathsf{c_i} = (\mathsf{c_{i1}}, \mathsf{c_{i2}}, \ldots, \mathsf{c_{in}}, \mathsf{m_{i1}}, \ldots, \mathsf{m_{ik}}) \in \mathsf{B}$, $\mathsf{c_i}$ satisfies both conditions ϕ_1 and ϕ_2. Accordingly, $\mathsf{c_i} \in \mathsf{Aux}$ because it satisfies ϕ_1, and then $\mathsf{c_i} \in \mathsf{A}$ because it satisfies ϕ_2.

Finally, $\forall \mathsf{c_i}, \mathsf{c_i} \in \mathsf{A} \Leftrightarrow \mathsf{c_i} \in \mathsf{B}$, implies A=B. □

9.3.4 GOLAP Engine

The Query Engine receives the parse tree (optimized or not) for evaluation. The tree is navigated, and for each visited node the corresponding operation is performed. More specifically, for ROLLUPExpression, DRILLSOWNExpression, SLICEExpression, DICEExpression and DRILLACROSSExpression nodes, a polymorphic function is applied, depending of the type of the input parameter, i.e., OLAP or FOLAP cuboids.

The Query Engine strategy consists in:

- For OLAP cubes, only the bottom cuboid of each cube instance is materialized, and each resulting cuboid is computed by applying SQL queries, as the cubes are implemented in ROLAP databases.

- For FOLAP cubes, the Generic Map Algebra operations are applied over the associated DField (corresponding to the bottom cuboid) and the FOLAP cuboid is populated with that result.

When a ROLLUPExpression or a DRILLDOWNExpression node is visited, the new level given by the second parameter (the right child) is added to the set of current levels. If the first parameter (left child) is an OLAP cuboid, no changes are made to the corresponding fact table. In consequence, since these IPO operations are not materialized, when IGO operations are applied the corresponding GROUP BY must be performed to summarize the measures in the fact table. If the first parameter (left child) is a FOLAP cuboid, operations based on the ZONAL operator are temporarily performed on the associated DField, and the output cuboid is updated with the values obtained. If the DField is an STDField and none of the current levels is a bottom level, two Zonal operations are performed. However, the transactions are not materialized in the bottom DField, which remains unchanged.

When a SLICEExpression node is visited, a new cuboid without the dimension or measure given by the second parameter (right child) is returned. If the first parameter (left child) is an OLAP cuboid, the dimension or measure is actually removed from the fact table. If the first parameter is a FOLAP cuboid, the fact table and the associated bottom DFiled are updated.

When a DICEExpression node is visited, a new cuboid without the cells that do not satisfy the condition given by the second parameter (right child) is returned. If the first parameter (left child) is an OLAP cube, the tuples that do not satisfy the condition are actually eliminated from the fact table. If the first parameter is a FOLAP cuboid, besides updating the fact table, the cells of the associated DField that do not satisfy the condition are set to '$\perp$', and the tessellation is changed, if necessary (see Example 58).

When a DRILLACROSSExpression node is visited, a new cuboid with the measures of the first and second parameters (left and middle childs), and optionally the new measures given by the third parameter (right child) are returned. The matching between cells is given by the semantic mapping between the cubes. If the first parameter is an OLAP cube, the measures of the second parameter are actually added to the fact table and, if the right child exists, the renamed measures and/or the new measures are also added. In the case the second parameter is a bottom FOLAP cuboid, the measures are inferred from the associated DField. If the first and second parameters are non-bottom FOLAP cuboids, the fact table is actually updated, and the output cuboid becomes an OLAP cuboid. If the first and second parameters are bottom FOLAP cuboids, a generic LOCAL operation is performed between them, and the ouput FOLAP cuboid is built with the resulting DField.

Remark 17. *Note that the semantic mappings that are modified by the operations as explained in Section 5.5, are performed automatically in our implementation. The user only needs to indicate the semantic mappings for the input cubes, in the XML document. The mappings induced by the OLAP operations are computed by the engine.*

9.4 Example GOLAP-QL Queries

So far we have assumed that all the queries we have shown have been written by experts, and in a concise way. However, since our model, and its associated language GOLAP-QL were designed to work at the conceptual level, non-expert, naive users may be able to express queries over cubes, and these queries may not always be written in the best way. For instance, a ROLLUP can be followed by a DRILLDOWN operator, while, as we have seen, this sequence can be replaced by just the latter operation, and the result would be the same. Next, we show some queries assuming that are not written in the most concise way, and show how the GOLAP optimizer deals with them. Note that non-optimal sequences of operations can also be obtained when a graphic OLAP-style navigation using a graphic tool gets translated into GOLAP-QL commands, which is also a possible scenario for our proposal.

As we explained in Section 1.4 and Chapter 6, in our running example, a wine producer in Belgium has developed a business analytics system in order to be able to take well-informed decisions about grape production in her vineyard. Two SOLAP cubes were produced: a Vineyard cube, which contains data about grape harvest involving time periods, geographic zones and grape variety, and a Fumigation cube containing fumigation information that includes fumigated zones, time periods and factual 3D trajectories of the spraying planes (see Chapter 6). To enhance analysis possibilities, our producer has incorporated external information about precipitation, temperature and altitude, represented as continuous field data.

The FOLAP cubes TempF and PrecipF, associated to the STDFields Temperature and Precipitation, were built with the AVG function ssocited to their measures. Also, the FOLAP cube AltitudeF has been created, associated to the SDField Altitude, with the MAX function as its measure. Finally, the necessary semantic mappings between these cubes have been defined (see Section 6.3).

Using the scenario above, we illustrate the capabilities of the language and the use of the optimizer module, by means of three comprehensive example queries.

Example 74 (Q4: Using a SOLAP and a FOLAP cuboid)**.** The user wants to study the relationship between grape production, average temperature and altitude, regardless the time periods, for those provinces with temperatures between 11 and 15 Celsius degrees. To answer this query, the user needs the Vineyard SOLAP cube, containing grape production information, and the TempF and AltitudeF FOLAP cubes for the rest of the data.

First, we roll-up the Vineyard cube to the levels Province and Month resulting in a cuboid Aux1. Then, we do the same with the bottom cuboid of TempF (associated to the STDField Temperature), and select the cells with the desired temperature range in TempF, resulting in a cuboid Aux2. Note that in this case, the user has first rolled-up TempF to the District level, and then to Province, which is not the best way to write this query. She has also performed two consecutive dicing operations, while they may have been written as a single one. We show next how the optimizer module handles this.

Aux1 := ROLLUP(ROLLUP('Vineyard', TimeDim, Month), BlockDim, Province);

Aux2 := DICE(DICE(ROLLUP(ROLLUP(ROLLUP('TempF', SpatialDim, District), TemporalDim, Month), SpatialDim, Province), value $>$ 11), value $<$ 15);

A drill-across operation is then performed between Aux1 and Aux2, to be able to put together all the information, and data are aggregated over time, that means, the TemporalDim dimension is removed, to be able to further combine the result with the Altitude cuboid, which is spatial. This results in the cuboid Aux3

Aux3 := SLICE(ROLLUP(DRILLACROSS(Aux1, Aux2, '2.value AS temperature'), TimeDim, All), TimeDim);

So, basically, Aux3 contains the combination between an alphanumerical discrete cube, and the temperature cube aggregated over time. Finally, we perform a drill-across between Aux3 and Altitude, which is previously rolled-up to the Province level.

Result := DRILLACROSS(Aux3, ROLLUP('AltitudeF', SpatialDim, Province), '2.value AS altitude')

Figure 9.11 depicts the corresponding parse tree. The optimized query is shown in Figure 9.12. We can observe that the two DICE operations have been simplified into a single one, and the ROLLUP to All previous to slicing the TimeDim dimension is eliminated, and the double ROLL-UP to District and to Province has been replaced for a single Roll-up to Province.

The GOLAP Engine executes the optimized parse tree as follows. First, the cube resulting from the two ROLLUP operations over Vineyard is stored in cuboid Aux1. Previous to the DICE operation, the GOLAP Engine applies two ROLLUP operations up to Month and Province levels. Then, the Temperature STDField associated to the TempF FOLAP cube is modified by the DICE operation, so samples that do not satisfy the condition are set to '$\perp$', and the corresponding tuples in TempF are eliminated. In summary, Aux2 is a cuboid with an associated STDField, namely $\mathsf{Temperature}_{\mathsf{out}}$. Notice that, if an SDField of the $\mathsf{Seq}(\mathsf{Temperature}_{\mathsf{out}})$ has a tessellation that contains some cells straddling two provinces such that one of them satisfies the dicing condition and the other not, its tessellation is changed (see Example 58). For instance, the SDField corresponding to the October snapshot in $\mathsf{Seq}(\mathsf{Temperature}_{\mathsf{out}})$ is shown in Figure 9.13.

Taking advantage of the automatically inferred semantic mapping between Aux1 and Aux 2 (derived from SM(Vineyard, TempF), the DRILL-ACROSS between these partial resulting cuboids is stored in the Aux3 cuboid, also adding the altitude measure to the original harvest measure.

Finally, the Result cuboid has the two measures of Aux3 plus the altitude measure summarized by province. This cuboid, as displayed in the output console, is shown in Figure 9.14. □

Example 75 (Q5: Using two SOLAP cuboids)**.** Now, our user wants to find out the production of the districts that have been fumigated with chemical products at a height less than 12 meters during 2009.

The user first computes a cuboid over Vineyard at the District level, selecting

the cells corresponding to the year 2009. However, she first rolls-up to the Province level, and from there she drills-down one level to reach the districts. The optimizer will take care of fixing this flaw in the query formulation.

Aux1 := DICE(DRILLDOWN(ROLLUP(ROLLUP('Vineyard', BlockDim, Province), TimeDim, Year), BlockDim, District), year = 2009);

Then, the query prepares the second cuboid, the one containing fumigation data. The measure of the Fumigation cube is a 3D geometry (a trajectory), which allows applying the spatial function ZMIN, which returns the minimum height of the trajectory.

Aux2 := SLICE(ROLLUP(DRILLDOWN(DICE(DICE(DICE(ROLLUP(ROLLUP('Fumigation', PestDim, Type), DateDim, Year), PestDim.Type.ptype = 'chemical'), ZMIN(trajectory) < 12), year = 2009), PestDim, Pesticide), GeoDim, Zone), PestDim);

Notice that the members of the Month level in the Vineyard and Fumigation cubes will never match, because fumigation always precedes picking (that is, we consider that those will be concurrent facts at the month granularity). On the contrary, after positioning both cuboids at the Year level and after applying the dicing operation, the join in the final DRILLACROSS operation will give a nonempty result.

Result := ROLLUP(SLICE(DRILLACROSS(Aux1, Aux2, 'fn : Zmin(trajectory) AS heightFumig WITH Min'), trajectory), BlockDim, Province)

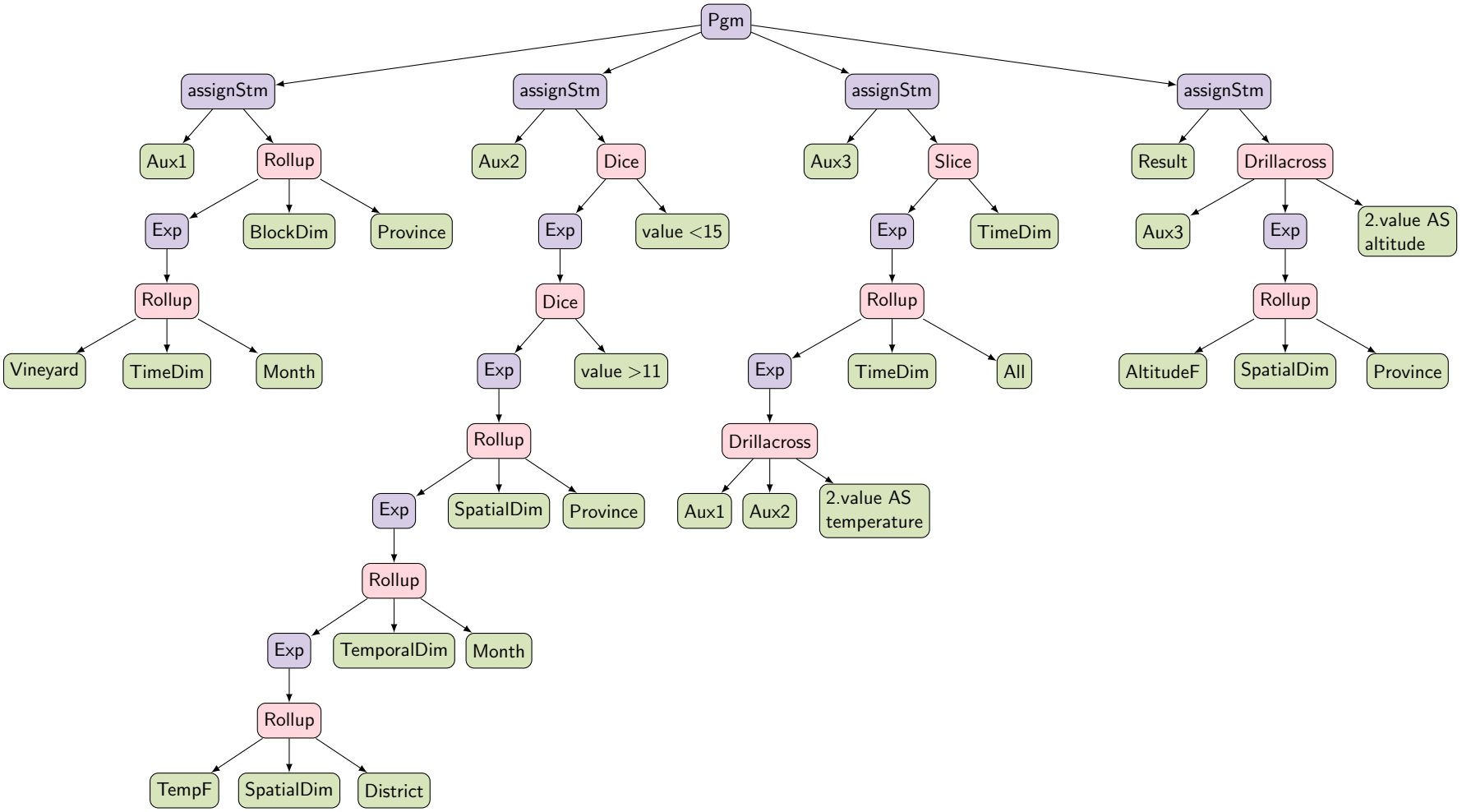


Figure 9.11: Parse tree for Q4.

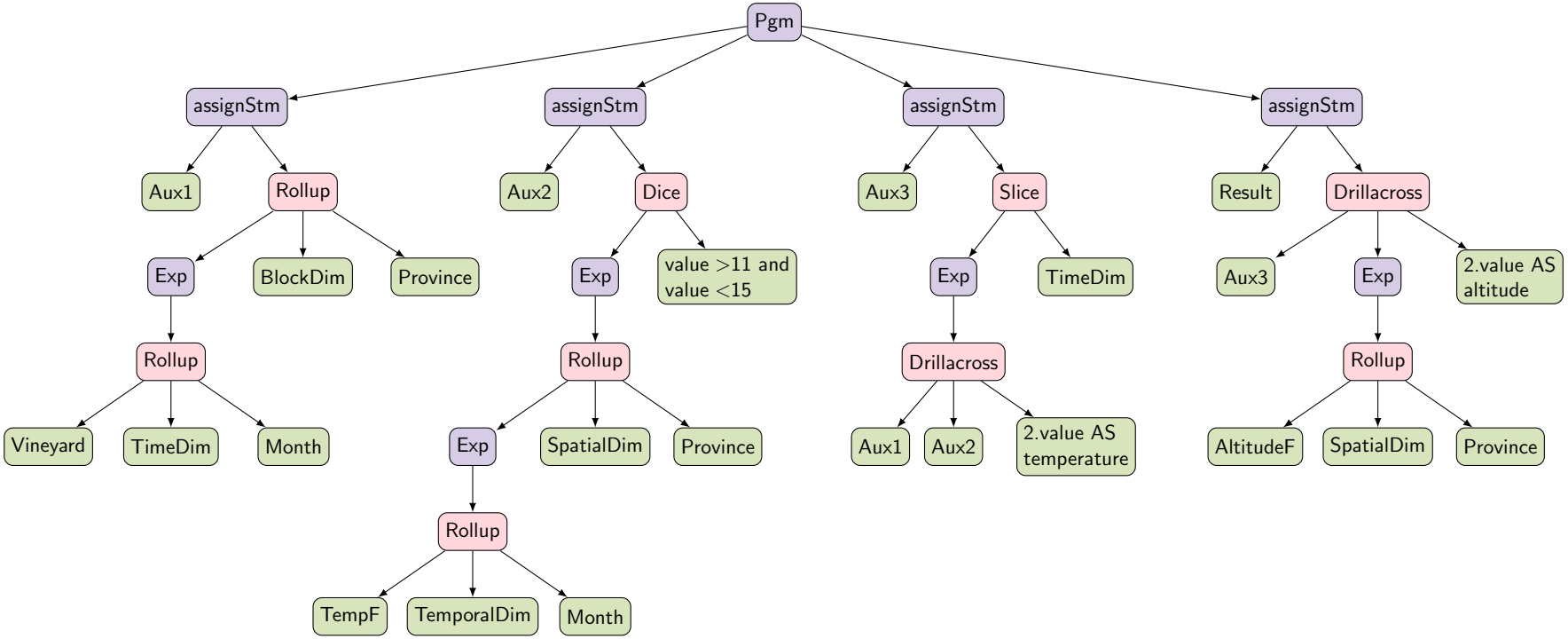


Figure 9.12: Optimized parse tree for Q4.

Figure 9.13: SDField corresponding to October in $\mathsf{Seq}(\mathsf{Temperature}_{\mathsf{out}})$ for Q4

Panel de Salida

Salida de datos | Comentar | Mensajes | Historial

	pname character varying(50)	pgeom geometry	harvest text	temperature numeric(6,2)	altitude text
1	Antwerp		18913	11.19	54
2	Brussels	01060000	7689	11.38	105
3	East Flanders		53212	11.58	85
4	Flemish Brabant		110024	11.23	117
5	Liege		46957	13.59	608
6	Limburg		35242	14.59	236
7	Luxembourg		27048	13.38	540
8	Namur		12807	14.34	431
9	West Flanders		6800	11.51	53

Figure 9.14: Result cuboid for Q4 in the output console

Figure 9.15 depicts the parse tree that corresponds to the original query, that means, without optimization. The parse tree of the optimized query is shown in Figure 9.16. We can observe that the optimizer has eliminated the first ROLLUP to Province in Vineyard, has replaced the three DICE operations in Fumigation by only one operation, and has eliminated the DRILLDOWN to Pesticide level in PestDim, before slicing this dimension.

The GOLAP Engine executes the optimized parse tree as follows. First, the

cube resulting from the ROLLUP and DICE operations over Vineyard is stored in Aux1. Then, in Aux2 we keep the zones with their trajectories summarized using UNION function, such that these trajectories satisfy the dicing condition. Notice that Aux2 does not contain PestDim dimension. Taking advantage of the semantic mapping between Vineyard and Fumigation (studied in Chapter 6), which is propagated to Aux1 and Aux2, a DRILL-ACROSS operation can be applied between the previous partial resulting cuboids. Finally, the SLICE of the measures allows keeping only the desired ones in the Result cuboid, i.e., harvest, and fumigation minimum height. The final Result cuboid in the output console is shown in Figure 9.17. □

Example 76 (Q6: Using two FOLAP cuboids)**.** Our final example looks for all the districts in Belgium that present the ideal conditions for the Malbec crop, which are related to the range of temperature and average precipitation during specific periods. For example, the best conditions for the picking period are temperatures between 18 and 22 Celsius degrees, with average precipitations less than 40 mm. In consequence, the user needs to put together cubes representing fields containing minimum and maximum temperatures, and fields containing average precipitation summarized by district for the following periods: bud break (February-March), flowering (first part of April), ripeness (from the second half of April to the end of August) and picking (September).

In the XML cube schema file we declare two new FOLAP cubes associated to the Temperature STDField: TempMax and TempMin, with MAX and MIN functions for the measure $\mathsf{value}_{\mathsf{temp}}$, respectively. We also build the PrecipF FOLAP cube, associated with Precipitation STDField. Finally, the necessary semantic mappings are defined between them.

Furthermore, we insert the new Period level in the TemporalDim dimension between Timestamp and Year with the ad-hoc members: budbreak (February and March), flowering (April 1st to 15th), ripeness (April 16th to August 31th), picking

(September), nothing1 (January) and nothing (October, November and December). The members nothing and nothing1 are defined in order to complete the year interval and guarantee summarizability.

Finally, the query reads:

Aux1 := RollUp(RollUp('TempMin', TemporalDim, Period), SpatialDim, District);
Aux2 := RollUp(RollUp('TempMax', TemporalDim, Period), SpatialDim, District);
Aux3 := RollUp(RollUp('PrecipF', TemporalDim, Period), SpatialDim, District);
Result := Drillacross(Drillacross(Aux1, Aux2, '1.value AS minTemp
2.value AS maxTemp'), Aux3, '2.value AS avgPrecip')

Figure 9.18 depicts the parse tree that corresponds to the original query. In this case, the optimizer module does not change the parse tree. The GOLAP engine executes the parse tree as follows. First, it applies two Rollup operations on the TempMin FOLAP cuboid, up to the Period and District levels, and stores the resulting cuboid into Aux1. Then, it repeats the same two operations over the TempMax and PrecipF FOLAP cuboids, and stores the partial result into cuboids Aux2 and Aux3, respectively.

Notice that all these operations do not introduce changes into the STDFields associated to the three FOLAP cubes, because they only take the cubes at the desired levels. Finally, to apply the Drill-across operation between Aux1 and Aux2, the engine detects that these FOLAP cuboids are non-bottom cuboids on their respective cubes, and so it applies the corresponding Zonal operations over the Temperature and Precipitation associated STDFields prior to crossing the cuboids. The final cuboid, Result, as seen on the output console, is shown in Figure 9.19. Figure 9.20 depicts the associated SDFields corresponding to the four desired periods for analysis. □

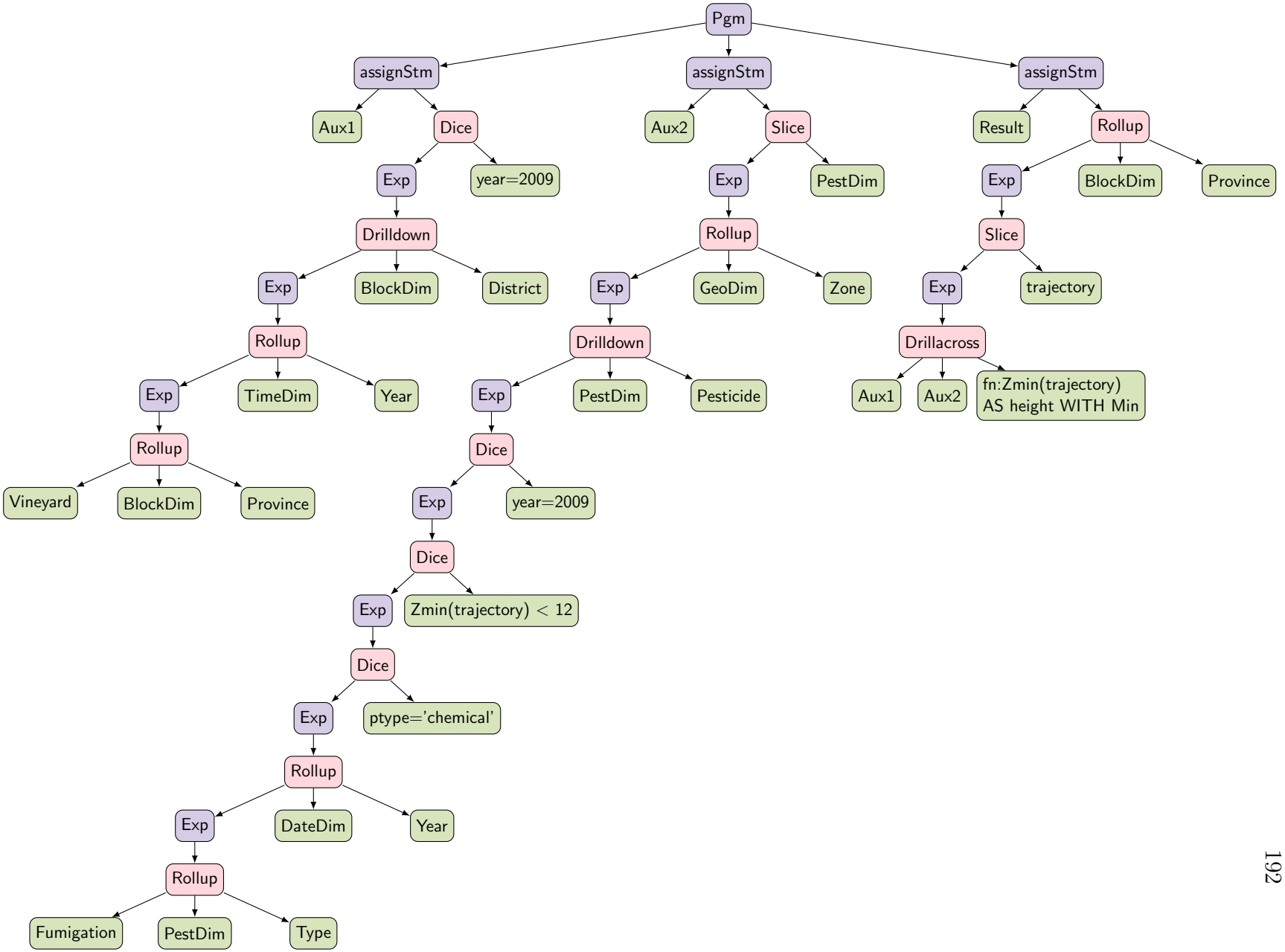


Figure 9.15: Parse tree for Q5.

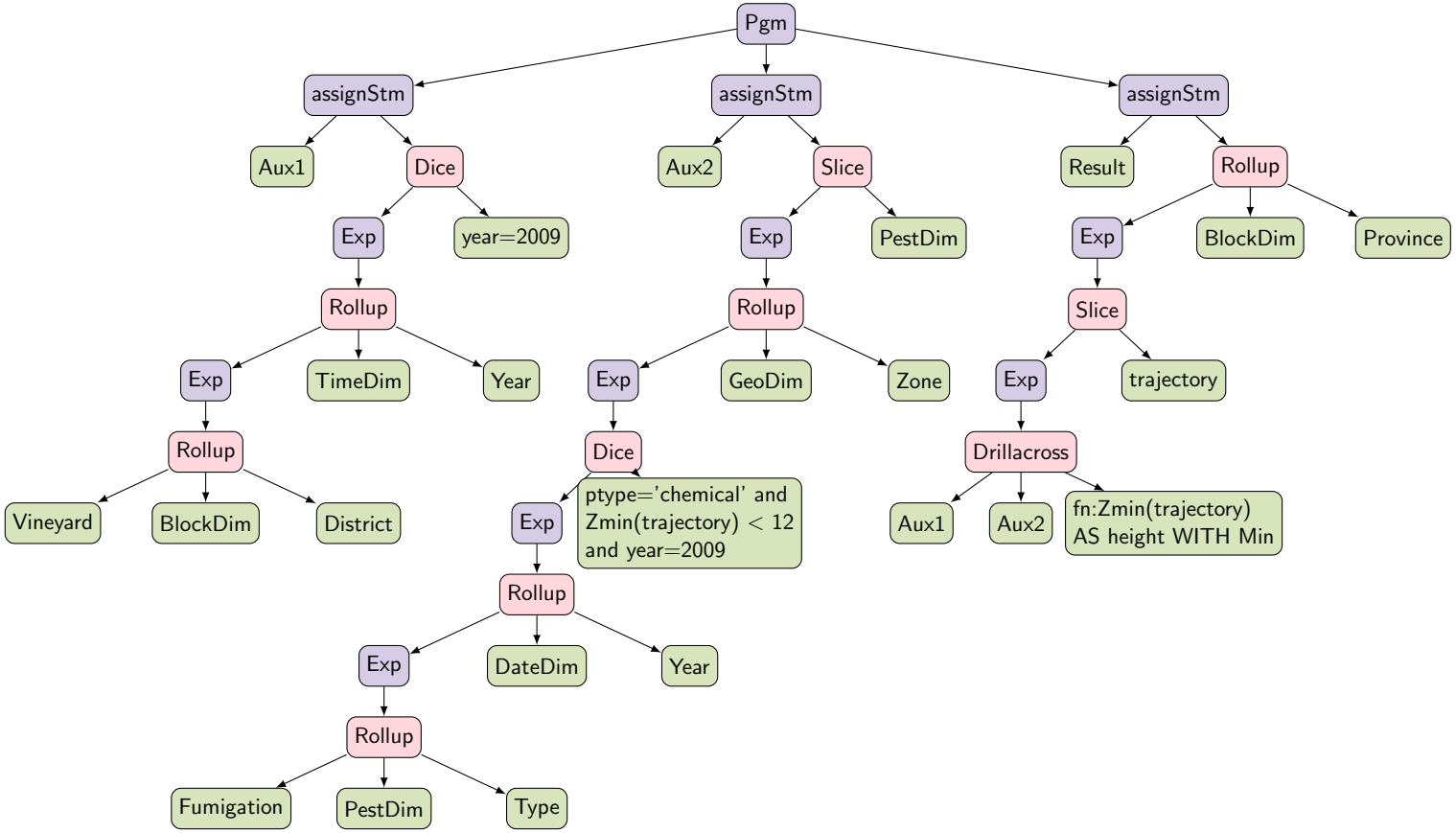
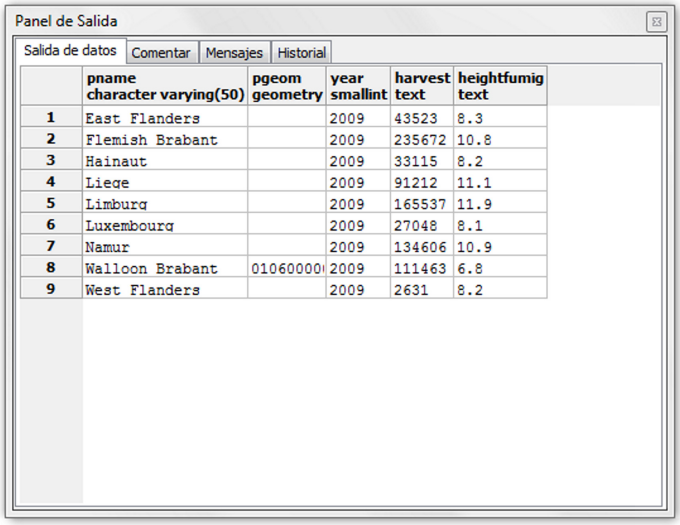


Figure 9.16: Optimized parse tree for Q5.



	pname character varying(50)	pgeom geometry	year smallint	harvest text	heightfumig text
1	East Flanders		2009	43523	8.3
2	Flemish Brabant		2009	235672	10.8
3	Hainaut		2009	33115	8.2
4	Liege		2009	91212	11.1
5	Limburg		2009	165537	11.9
6	Luxembourg		2009	27048	8.1
7	Namur		2009	134606	10.9
8	Walloon Brabant	01060000	2009	111463	6.8
9	West Flanders		2009	2631	8.2

Figure 9.17: Result cuboid for Q5 in the output console

9.5 Summary

In this chapter we have introduced a scripting query language, denoted GOLAP QL, that allows manipulating cubes based on the OLAP algebra defined in previous chapters. The GOLAP architecture is composed of five modules, namely Lexical Analyzer, Syntactic Analyzer, Semantic Analyzer, Query Optimizer and the GOLAP Engine, allowing lexical, syntactic, and semantics analysis. The query optimizer is based on a set of transformation rules that allow rewriting a GOLAP-QL expression in a way such that the GOLAP engine can apply those rules to produce a more efficient execution tree.

The design of the GOLAP Language is prepared to incorporate new data types that may be perceived as OLAP cubes. The first step is to generate a correct definition of a discrete cube associated to the data type to be added. Then, we could extend the GOLAP engine with a new algorithm for each node representing an OLAP expression (i.e., one new polymorphic function for each operator over the new added type).

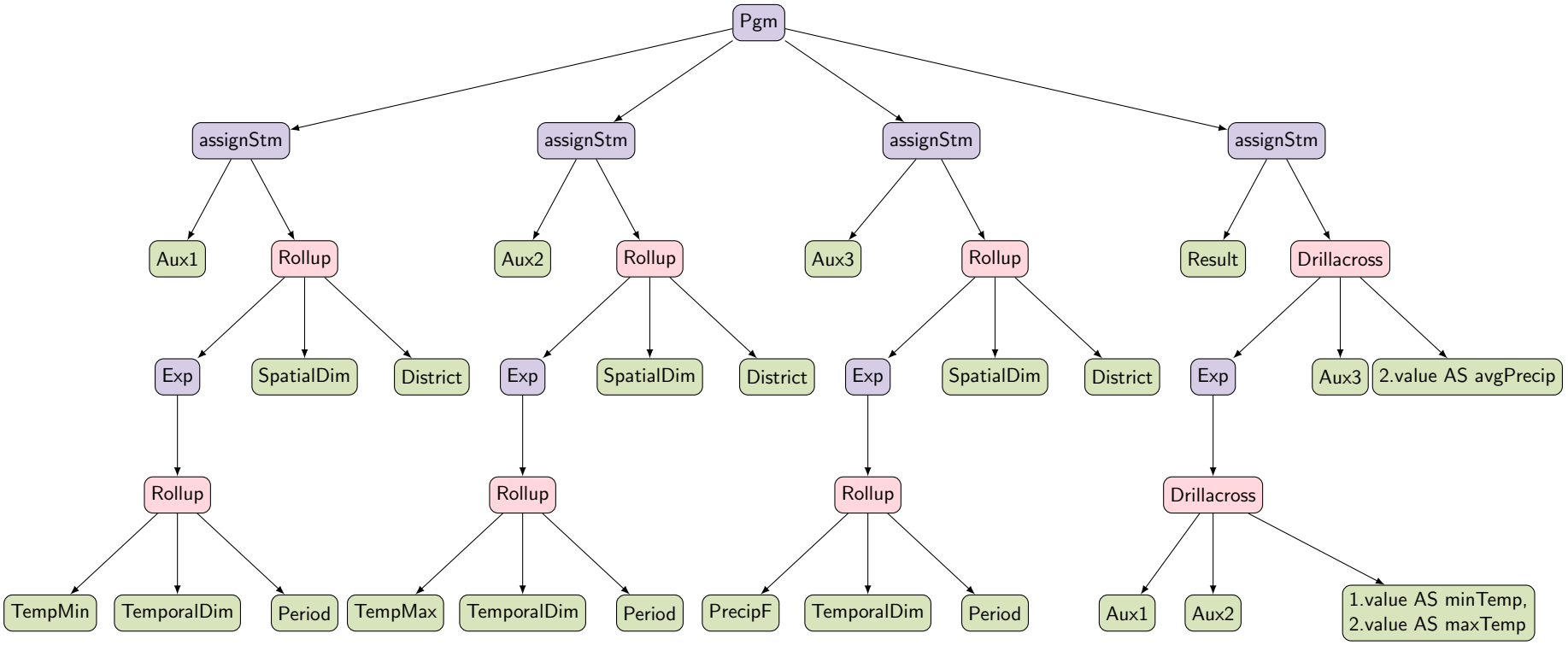


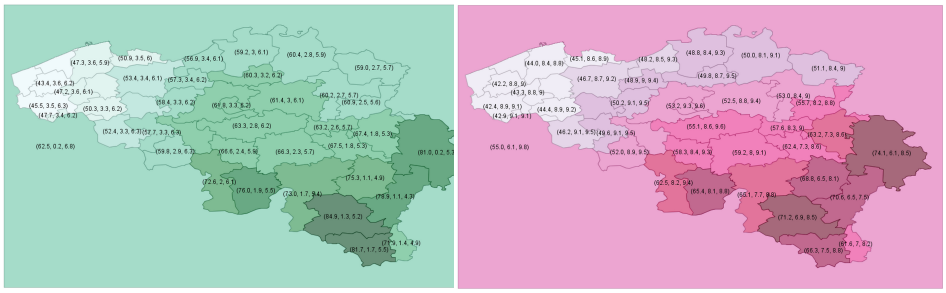
Figure 9.18: Parse tree for Q6

Panel de Salida

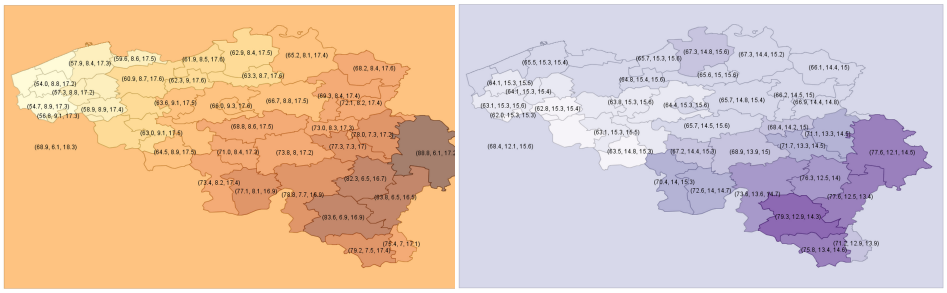
Salida de datos | Comentar | Mensajes | Historial

	dname character varying(75)	dgeom geometry	period character varying(30)	mintemp text	maxtemp text	avgprecip numeric(7,1)
1	Aalst	01060000(	BudBreak-2009	3.3	6.2	58.4
2	Aalst	01060000(	Flowering-2009	9.1	9.5	50.2
3	Aalst	01060000(	Picking-2009	15.3	15.6	63.8
4	Aalst	01060000(	Ripeness-2009	9.1	17.5	63.6
5	Antwerpen	01060000(	BudBreak-2009	3	6.1	59.2
6	Antwerpen	01060000(	Flowering-2009	8.4	9.3	48.8
7	Antwerpen	01060000(	Picking-2009	14.8	15.6	67.3
8	Antwerpen	01060000(	Ripeness-2009	8.4	17.5	62.9
9	Arlon	01060000(	BudBreak-2009	1.4	4.9	71.9
10	Arlon	01060000(	Flowering-2009	7	8.2	61.6
11	Arlon	01060000(	Picking-2009	12.9	13.9	71.2
12	Arlon	01060000(	Ripeness-2009	7	17.1	75.4
13	Ath	01060000(	BudBreak-2009	3.3	6.3	57.7
14	Ath	01060000(	Flowering-2009	9.1	9.5	49.6
15	Ath	01060000(	Picking-2009	15.3	15.5	63.1
16	Ath	01060000(	Ripeness-2009	9.1	17.5	63.0
17	Bastogne	01060000(	BudBreak-2009	1.1	4.3	78.9
18	Bastogne	01060000(	Flowering-2009	6.5	7.5	70.6
19	Bastogne	01060000(	Picking-2009	12.5	13.4	77.6
20	Bastogne	01060000(	Ripeness-2009	6.5	16.5	83.8
21	Borgworm	01060000(	BudBreak-2009	2.6	5.7	63.2
22	Borgworm	01060000(	Flowering-2009	8.3	9	57.6
23	Borgworm	01060000(	Picking-2009	14.2	15	68.4
24	Borgworm	01060000(	Ripeness-2009	8.3	17.3	73.0

Figure 9.19: Result cuboid for Q6 in the output console.



(a) Bud Break Period (b) Flowering Period



(c) Ripeness Period (d) Picking Period

Figure 9.20: Associated SDFields for the periods requested in Q6.

Chapter 10

Model Implementation

In this chapter we describe the prototype we have implemented as a proof-of-concept of the plausibility of our proposal. The queries of Section 9.4 have been executed using this implementation. We also report preliminary experimental results over real and synthetic data.

10.1 Schema Implementation

Mondrian is an open source OLAP engine written in Java.[1] It can execute MultiDimensional eXpressions (MDX), a query language introduced by Microsoft[2] for querying OLAP cubes. Although it is not an open standard, several OLAP vendors adopted it for their products. GeoMondrian is an open source SOLAP server that extends Mondrian to work with geometries, i.e., it provides spatial extensions to MDX expressions.

Mondrian describes cube schemas in an XML file which is read by the OLAP engine. Its basic XML elements are cubes, dimensions, hierarchies, levels, properties, measures and annotations. The structure of this XML is very simple and contains all the information necessary for parsing MDX expressions. GeoMondrian extends this cube schema in order to support geometry data types in properties and measures.

The main reason for not using the MDX query language is that MDX expresses OLAP queries at the logical level, and we focused in defining basic but powerful

[1] Java Standard Edition, `http://www.oracle.com/technetwork/java/javase`
[2] Microsoft, `http://www.microsoft.com`

operators to be used at the conceptual level. We also decided not to use MDX inside our Algebra operators because in fact MDX is not executed directly in OLAP engines, it is translated into SQL in order to be executed in a RDBMS. Nevertheless, we decided to use the existing GeoMondrian cube schema for declaring cube elements instead of creating our own schema syntax, given its simplicity and to allow Mondrian/GeoMondrian users to reuse their existing files.

An "annotation element" can be used inside any element in GeoMondrian cube schema for adding comments that are ignored by the server. Taking advantage of this feature we used this element for declaring 'field extensions'. Thus, any cube schema could be used indistinctly in our engine (which interprets same kind of annotations as we explain later) or in the GeoMondrian OLAP server. More precisely, we use this kind of element inside a FOLAP cube for declaring the name of the DField that is associated to it. The attribute called 'name' indicates the name of this DField. Other kinds of annotations are ignored, as usual.

For example, the AltitudeF FOLAP cube has associated the DField named 'altitudefield' and it is declared as follows:

```
<Cube name="AltitudeF">
  <Annotations >
    <Annotation name="field" >altitudefield</Annotation>
  </Annotations>
  <Table name="altitudecube" schema="public" />
  <DimensionUsage source="newspatialdim"
     name="SpatialDim" visible="true" foreignKey="sample"/>
  <Measure name="value" column="value_1" datatype="Numeric"
     formatString="string" aggregator="max" formatter="string"
     caption="string" description="string" visible="true"/>
</Cube>
```

10.1.1 DField Metadata

As discussed previously, the GOLAP Engine actually performs OLAP operations over DFields. For example, evaluation of the expression DICE(SLICE(TempF, TemporalDim), $\text{value}_{\text{temp}} > 20$) first eliminates the TemporalDim dimension of the TempF FOLAP cube which has associated the temperature STDField. Thus, this STDField must exist in the system. Moreover, the temporary resulting cube obtained after slicing the dimension has also an associated SDField. Finally, the DICE operator is applied over this spatial cube (and its associated SDField).

In our implementation, all DFields, temporary or not, persist in ROLAP tables. Given a DField, its samples are stored in a separate table whose name coincides with the name of the DField. For example, the samples of the Altitude SDField are stored in the following table:

```
CREATE TABLE altitudefield
(
  id integer NOT NULL DEFAULT nextval('fieldseq'::regclass),
  sample geometry,
  cell geometry,
  x double precision,
  y double precision,
  value_1 double precision,
  PRIMARY KEY (id)
)
```

Each row in this table describes a sampled tuple. Attributes x and y store the 2-D spatial information. For efficiency reasons, instead of calculating on-the-fly the spatial sample point through the expression Point(x, y), we precalculate and store it in the attribute sample. Besides, the attribute cell stores the geometry of the tessellated cell where the sample belongs. The sampled value can be a single value or an N-dimensional vector. For simplicity, we treat a single value as a one-dimensional vector. Thus, we use as many attributes of the form value_i

as dimensionality this vector has. Finally, the attribute id is used to univocally identify a tuple in this table.

In the case where an SDField is one of the components of an STDField, we add the extra attribute t as its temporal component. For instance, the samples of the ninth SDField component of the Temperature STDField (i.e., the ninth snapshot) are stored in the following table:

```
CREATE TABLE temp_9
(
  id integer NOT NULL DEFAULT nextval('fieldseq'::regclass),
  sample geometry,
  cell geometry,
  x double precision,
  y double precision,
  t timestamp without time zone,
  value_1 double precision,
  PRIMARY KEY (id)
)
```

The rest of the information of each DField is stored in a common table called DFieldMetadata. In the case of a SDField it is enough to store: 1) The name of the DField (which coincides with the name of the table that stores its samples); 2) Its domain represented by close intervals (l1min, l1max, l2min, l2max); 3) Its labels (label1, label2); 4) The name of tessellation type (tesstype); 5) The name of the interpolation function (fninterfn); 6) The dimensionality of the vector for its sampled value (numbervalues).

In the case of an STFDield, all of its SDField components are stored as explained before. For example, the Temperature STDField spanning one year is composed of twelve SDFields, one for each month. Thus, there are twelve tables for the samples exist, and twelve rows in the Dfieldmetadata table. Besides, an extra row in the DFieldmetadata table describes its overall structure. More precisely, we

must add attributes l3min, l3max, label3, fieldcomponents and seq. The attribute fieldcomponents stores an array with the name of its SDFields components. The attribute seq stores an array with the intervals of validity of each snapshot.

Finally, the DFieldmetadata table used for storing all kinds of DFields is:

```
CREATE TABLE dfieldmetadata
(
   name character varying(100) NOT NULL,
   l1min double precision,
   l1max double precision,
   l2min double precision,
   l2max double precision,
   l3min double precision,
   l3max double precision,
   label1 character varying(100),
   label2 character varying(100),
   label3 character varying(100),
   tesstype character varying(10),
   fninterfn character varying(10) NOT NULL,
   numbervalues integer,
   fieldcomponents character varying[],
   seq bigint[],
   PRIMARY KEY (name)
)
```

Some attributes may contain null values. For instance, fieldcomponents and seq for an SDField. Moreover, the tesstype attribute for an STDField contains null since each of its components can be tessellated in a different way. Figure 10.1 depicts an actual snapshot of this table.

	name [PK] characte	label1 character var	l1min double precis	l1max double precis	label2 character var	l2min double precis	l2max double precis	lab cha	l3min bigint	l3max bigint	numl integ	tesstype character var	fninterfn character var	fieldcomponen character vary	seq bigint[]
107	temp	x	2.5	6.500000476	y	49.33333206	51.66666793	t	12307752000	12623112000	1		core.CteFn	{temp_0,temp_	{1230775200000,123
108	temp_0	x	2.5	6.500000476	y	49.33333206	51.66666793	t	12307752000	12334536000	1	RASTER	core.CteFn		
109	temp_1	x	2.5	6.500000476	y	49.33333206	51.66666793	t	12334536000	12358728000	1	RASTER	core.CteFn		
110	temp_10	x	2.5	6.500000476	y	49.33333206	51.66666793	t	12570444000	12596364000	1	RASTER	core.CteFn		
111	temp_11	x	2.5	6.500000476	y	49.33333206	51.66666793	t	12596364000	12623112000	1	RASTER	core.CteFn		
112	temp_2	x	2.5	6.500000476	y	49.33333206	51.66666793	t	12358728000	12385548000	1	RASTER	core.CteFn		
113	temp_3	x	2.5	6.500000476	y	49.33333206	51.66666793	t	12385548000	12411468000	1	RASTER	core.CteFn		
114	temp_4	x	2.5	6.500000476	y	49.33333206	51.66666793	t	12411468000	12438252000	1	RASTER	core.CteFn		
115	temp_5	x	2.5	6.500000476	y	49.33333206	51.66666793	t	12438252000	12464172000	1	RASTER	core.CteFn		
116	temp_6	x	2.5	6.500000476	y	49.33333206	51.66666793	t	12464172000	12490956000	1	RASTER	core.CteFn		
117	temp_7	x	2.5	6.500000476	y	49.33333206	51.66666793	t	12490956000	12517740000	1	RASTER	core.CteFn		
118	temp_8	x	2.5	6.500000476	y	49.33333206	51.66666793	t	12517740000	12543660000	1	RASTER	core.CteFn		
119	temp_9	x	2.5	6.500000476	y	49.33333206	51.66666793	t	12543660000	12570444000	1	RASTER	core.CteFn		
120	tt	x	2.499999761	6.416666984	y	49.41666412	51.58333587	t	10227600000	10280304000	2		core.CteFn	{tt_0,tt_1}	{1022760000000,102
121	tt_0	x	2.499999761	6.416666984	y	49.41666412	51.58333587	t	10227600000	10254384000	2	RASTER	core.CteFn		
122	tt_1	x	2.499999761	6.416666984	y	49.41666412	51.58333587	t	10254384000	10280304000	2	RASTER	core.CteFn		
123	voroproof	dimx	2.583333253	6.416666984	dimy	49.41666412	51.58333587				1	VORONOI	core.CteFn		
124	altitudefie	x	2.499999999	6.416666666	y	49.41666666	51.58333333				2	RASTER	core.CteFn		

Figure 10.1: DFieldMetadata table

10.2 Data Preparation

We next describe the process of preparing the data for our experiments. We focused on data variety rather that in data volume, and prioritized the fact of dealing with real-world data.

10.2.1 DFields

Data for building Altitude, Temperature and Precipitation DFields were obtained from WorldClim[3], which provides a set of global climate layers freely available for academic purposes. Each layer is stored in grid format (generic raster data) and consists of two files: a sequential binary file where each cell is a signed 2-byte integer value (.bil) and a header with text description (.hdr).

Altitude data represent elevation above sea level, expressed in meters and corresponds to a non-temporal layer. Temperature and Precipitation data represent monthly mean temperature and average precipitation from 1950 to 2000, i.e., the files contain 12 layers representing different months (since we wanted to cross information with other cubes that contain information during the year 2009, we assumed that these data represented values sampled during year 2009).

[3]WorldClim dataset, `http://www.worldclim.org`

The spatial components are in the latitude/longitude coordinate reference system datum World Geodetic System 84 (WGS84). Altitude and climate data have a 5 arc-minutes and 10 arc-minutes of spatial resolution, respectively.

The Temperature values were expressed in Celsius degrees and values were multiplied by 10 in order to be stored as signed 2 byte integers. We converted those values to real numbers (divided by 10) when importing them into our database. The unit used for the Precipitation data is millimeter, and for Altitude data, meter.

Raster data was downloaded in a generic grid format and imported into a PostgreSQL database equipped with the PostGIS plugin for handling spatial data types. This generates polygons with their associated values.

At the moment of implementing the algorithms proposed in this work we used PostgreSQL V9.0[4] object-relational database with the PostGIS V1.5[5] plug-in for enabling spatial capabilities. PostGIS had an ongoing project aiming at developing raster support which was not part of the plug-in. Moreover, nowadays PostGIS does not provide any capability to perform map algebra operations. Since our FOLAP operators are based on map algebra operators, and we do not focus on visualization, we used DIVA-GIS V7.5[6] for cropping world maps to the boundaries of our country of interest, i.e., Belgium, and then exporting the map to a vector format which can be imported into the spatial-enabled PostgreSQL database.

Once this transformation was made and data were stored in the database, the Altitude SField was stored in a table with 655 tuples. The information of each climate STField was stored in 4032 tuples, i.e. 336 tuples in a separate table for each month. Note however that to perform the actual OLAP operations, interpolation functions must be applied, since these fields represent continuous data.

10.2.2 OLAP Cubes

For the Vineyard cube, we used a map of Belgium containing geographic, demographic and economic information about provinces and districts (represented

[4]PostgreSQL ORDBMS, `http://www.postgresql.org`
[5]PostGIS spatial database extender for PostgreSQL, `http://www.postgis.org`
[6]`http://www.diva-gis.org/`

as polygons). The maps were obtained from the spatial library of the GIS Center3. With this geographic information we built the spatial hierarchy dimension described in Chapter 2. The fact table was populated with synthetic data consisting of 5136 tuples. These data reflects the real distribution of vine harvest over provinces and the real proportion of blocks according to wine production in districts of Belgium, according to Vinogusto SPRL[7] information. On the other hand, dates were generated randomly, although keeping unchanged the actual harvest time of each kind of grape, since each type has an annual grape harvest period, and a block can not be harvested more than once a year.

A Fumigation cube instance was populated with random dates and synthetic 3-D trajectories based on the geometries of the Belgian districts. According to its GeoDim dimension (Figure 6.5, the Area descriptor polygons were generated arbitrarily but they are completely contained in the Zone descriptor geometries (which correspond to the districts of Belgium as indicated in the semantic mapping described in Chapter 6) to satisfy summarizability.

The FOLAP cubes corresponding to each DField were built with all the above information.

10.3 GOLAP Implementation

Once a user poses a GOLAP query, the Lexical Analyzer and the Syntactic Analyzer check the syntax according to the rules of grammar of GOLAP query language described in Chapter 9. The parser is equipped with a tree-building preprocessor.

The implementation of both analyzers was developed with JavaCC[8] which is a top-down (recursive descent) parser. If no syntactic errors are detected, a tree data structure representing the query is built. Thus, a JJTree which represents the expression is built, and it can be navigated through Java language.

The JJTree (a data type) provides some basic support for the Visitor design pattern. In our case, this strategy allows us to separate the JJTree (the data type)

[7] http://www.vinogusto.com/es/bodegas/belgica
[8] https://javacc.java.net/

from the algorithms which visit its nodes. We implemented as many algorithms as different ways of visiting the nodes we needed, i.e. we provide different kinds of visitors to interact with the nodes, depending on the goal.

More precisely, JavaCC generates (using the grammar rules already introduced) the FCQLVisitor java interface, which contains one visit function for each kind of node in the JJTree:

```
public interface FCQLVisitor
{
  ...
  public Object visit(AssignStm node, Object data);
  public Object visit(DrillAcrossExpression node, Object data);
  public Object visit(RollupExpression node, Object data);
  public Object visit(DrillDownExpression node, Object data);
  public Object visit(SliceExpression node, Object data);
  public Object visit(DiceExpression node, Object data);
  public Object visit(QuotedLiteral node, Object data);
  public Object visit(BooleanExpression node, Object data);
  ...
}
```

JavaCC also generates an Accept() method for each node. Then, an algorithm that needs to solve a specific problem through navigating the tree, might belong to a class Adapter, which implements this FCQLVisitor interface.

The first visitor algorithm is implemented in the semantic analyzer. The goal consists in navigating the JJTree parsing generated in the previous step and determine if the query can be executed. For this purpose we created the FCQLVisitorAdapterEvaluator_SemanticChecking class, which contains a hash structure for storing pairs of objects: the first component is the name of a GOLAP variable used in assignments, and the second component is the schema of the cube. Each time an assignment appears, the semantic evaluator infers the cube schema on the right-hand side of the expression without evaluating the expression. This phase

only needs to detect semantic errors, not to evaluate the query. For example, if a DICE condition appears on the right-hand side of an assignment, the algorithm needs to analyze if the condition refers to existing measures and/or current levels of existing dimensions. If not, an exception is thrown. Otherwise, the left-hand side (variable) of the assignment is stored in the hash table with the inferred cube schema. If later, in another expression, a variable is used as parameter then the hash structure is looked-up. If this variable has been stored, its associated value (cube schema) is recovered and used to detect semantic correctness. If not, an exception is thrown due to the attempt to use a variable not previously defined.

Notice that for storing the cube schema in the hash structure it is necessary to infer it, and this process is solved recursively since the operators can be nested as many times as we want. Moreover, each time a DRILL-ACROSS operator is used where new measures are added or renamed, the algorithm needs to take this features into consideration to infer the correct cube schema. We show a piece of the algorithm in Appendix A.

Once the Semantic Analyzer accepts the JJTree, the following step (optional) is to run the query optimizer. The same strategy was used, i.e., a new visitor algorithm was designed inside the query optimizer with a different goal: to apply the optimizing rules explained in Section 9.3.3. We created the FCQLVisitorAdapter_RulesRewriting class and when a node that corresponds to a ROLLUPExpression, DRILLDOWNExpression, DICEExpression or SLICEExpression is reached, the algorithm modifies the tree according to those rules.

The final step takes place when the query engine receive a parsing tree (optimized or not) for evaluating. Again, a new visitor algorithm was implemented inside the query engine with the aim of applying the operators over cubes. This algorithm differs from the one in the semantic analyzer because it works with cubes. Nevertheless it also needs to use a hash structure, but in this case instead of storing the cube schema it stores the cubes that are obtained after applying the GOLAP operators visited.

10.4 Experimental Results

We now describe some preliminary experimental results, performed with two goals in mind: first, we wanted to evaluate the impact of the optimization rules presented in Section 9.3.3; second, we wanted to have an idea of the overall performance of the implemented prototype.

Our experiments showed that the time overhead introduced by the optimization step (the time consumed for navigating and rewriting canonical trees) ranges from 4 ms to 9 ms, i.e., it is negligible. Consequently, the total execution time is not affected by the introduction of this module, even when it finds no rule to apply. On the other hand, when at least one of the rules can be applied, we will show that overall query execution time decreases substantially. Furthermore, as we will see, this performance improvement is specially important in FOLAP cuboids due to the overhead introduced by the Generic Map Algebra operators on their associated DFields.

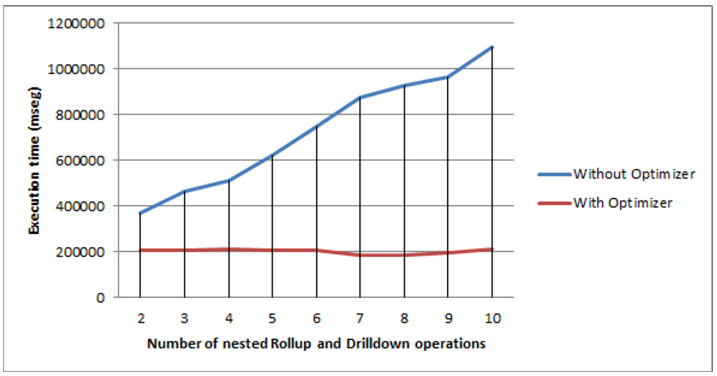
We next describe the experiments, and report the results. Our tests run on a non-dedicated Pentium Dual Core CPU 2.3 GHz notebook with total free RAM of 4.0 GB, and a 300 GB hard disk.

Tests for Rule 2

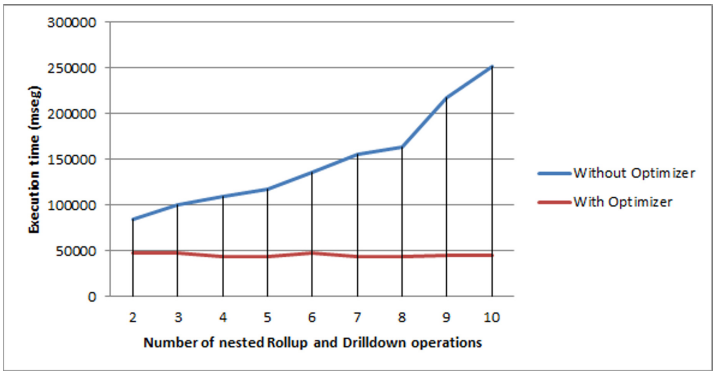
In order to measure the impact of applying Rule 2 (Nested IPO Operations) alone, we designed nested queries of variable length that range from two to ten operations which involving sequences of ROLLUP and DRILLDOWN operations over the same dimension. Queries were executed over both kinds of FOLAP cuboids, that is, cuboids associated to SDFields and to STDFields.

The queries executed over the spatial field AltitudeF were:
ROLLUP(AltitudeF, SpatialDim, District);
DRILLDOWN(ROLLUP(AltitudeF, SpatialDim, Province), SpatialDim, Point);
...
ROLLUP(DRILLDOWN(ROLLUP(... ROLLUP(AltitudeF, SpatialDim, District)))...))).

Analogous queries were executed over the spatiotemporal field TempF. We repeated each execution three times and registered the average execution time. Figure 10.2 shows the experimental results over the TempF and AltitudeF bottom FOLAP cuboids. We can see that while execution times increases linearly with the number of nested operations, the optimized queries run in an almost constant time of about 200 seconds in the case o a spatiotemporal field, and 50 seconds in the case of a spatial field.



(a) Over an STDField



(b) Over an SDField

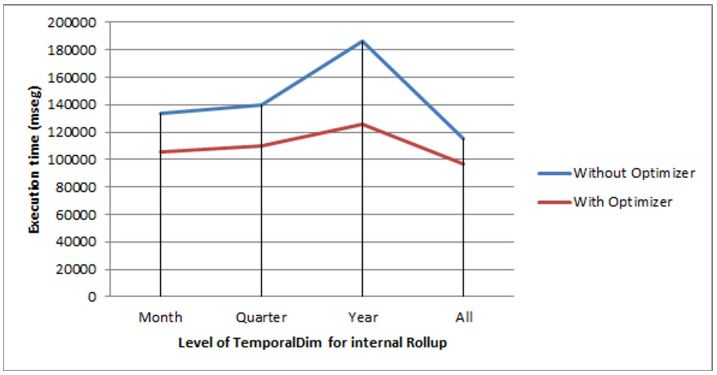
Figure 10.2: Rule 2 optimization over FOLAP cuboids

Tests for Rule 3

To measure the impact of Rule 3 over query execution, we use a nested query consisting in a a SLICE operation over the result of a previous ROLLUP operation on the temporal dimension over a FOLAP cube associated to an STDField. The

queries executed over TempF were:
SLICE(ROLLUP('TempF', TemporalDim, Month),TemporalDim);
SLICE(ROLLUP('TempF', TemporalDim, Quarter),TemporalDim);
SLICE(ROLLUP('TempF', TemporalDim, Year), TemporalDim);
SLICE(ROLLUP('TempF', TemporalDim, All),TemporalDim).

We repeated each query evaluation three times and report the average time. Figure 10.3 shows the experimental results for the TempF bottom FOLAP cuboid. Notice that expressions containing a rollup to All take less time to run because they do not need to perform the TimeToField operation.



(a) Associated to an STDField

Figure 10.3: Rule 3 Optimization on FOLAP cuboids

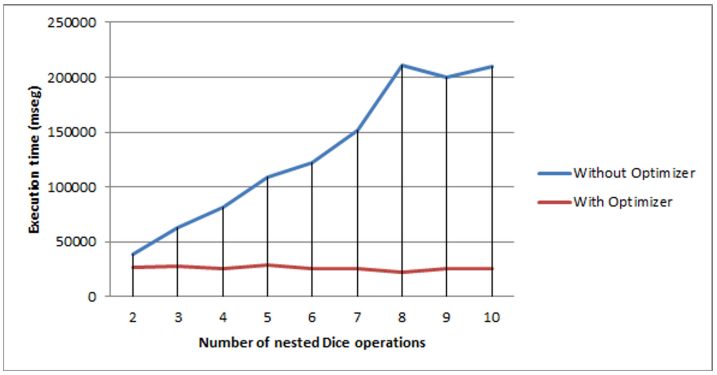
Tests for Rule 4

To measure the impact of applying Rule 4 (Nested Dice Operations) alone, we designed nested queries of variable length that range from two to ten operations which only involve DICE operations. The dicing condition was chosen so that a non empty result set were obtained. Queries were executed over both kinds of FOLAP cuboids, that is, cuboids associated to SDFields and to STDFields. The queries executed over AltitudeF were:
DICE(AltitudeF, value > 10);
DICE(DICE(AltitudeF, value>30), value>30);
...

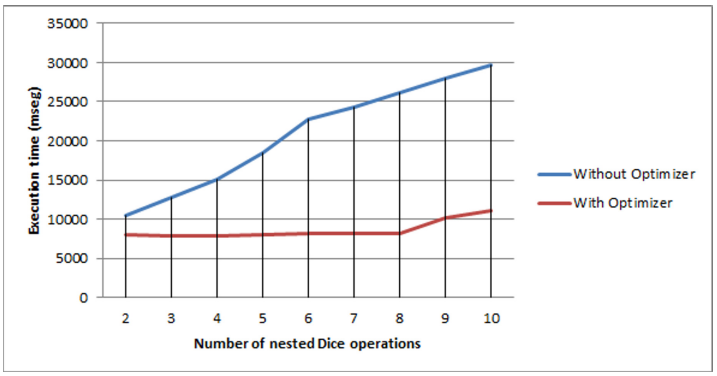
DICE(DICE(... DICE(AltitudeF, value>30))))))))))).

The same queries were executed over TempF.

We repeated each query evaluation three times and registered the average execution time. Figure 10.4 shows the experimental results for TempF and AltitudeF bottom FOLAP cuboids. We can see that the results follow the same pattern than in the case of the Rule 2 testing, but obviously with better execution times, since the query complexity is lower than in that case.



(a) Associated to an STDField



(b) Associated to an SDField

Figure 10.4: Rule 4 Optimization on FOLAP cuboids

In summary, we can observe that the overall performance of the query executions decrease substantially when the parse trees are optimized.

10.5 Summary

In this chapter we described the details of our implementation, as well as the technology we used. The development involved the following technologies: the Java Language, the GeoMondrian spatial OLAP engine, the PostgreSQL database, PostGIS plug-in, and the OpenJump technology. The extension of the GeoMondrian cube schema definition was designed to keep compatibility, i.e., OLAP servers can read the schema definition files, and skip specific DField annotations. Besides taking advantage of the Visitor/Adapter design patterns, a decoupled implementation was used to enable three kinds of visitors that can process the parse tree generated by the syntactic analyzer, depending on their specific goals. Finally, preliminary experimental results over the implemented prototype showed the plausibility of our proposal.

Chapter 11

Conclusions and Open Research Directions

Modern decision-making systems do not only use data internal to an organization, consolidated in a data warehouse, but also require complex data, i.e., images, geographic features, satellite maps, web logs, social networks information, normally external to the organization. More than often, analysts cannot manipulate all these kinds of data, since they are trained in the use of traditional OLAP systems that only handle alphanumerical data. In this thesis we addressed this problem, providing a high-level OLAP query language that allows end-users to manipulate cubes (aggregate, disaggregate, combine, slice, dice) independently of their content at the logical or physical levels. As a particular case, we show how discrete and continuous spatial data can be treated seamlessly, and combined with traditional OLAP cubes to enhance data analysis. We next summarize our contributions and suggest future research directions.

11.1 Conclusions

In this thesis we have first defined a formal framework which supports the cube metaphor, that is, allows different kinds of data to be perceived as data cubes. Therefore, the data cube is the only data structure the user manipulates in this framework. For this, we proposed both a generic conceptual definition of the multidimensional conceptual model and a conceptual algebra for cubes. In this

model, the data cube is composed of a schema and instances, and a data cube instance is defined as a collection of all its subcubes, denoted cuboids. There is an order between cuboids, defined by the hierarchies of the dimensions that compose the cube. This order induces a lattice of cuboids, a formalism which allows us to define a precise semantics for the operations over data cubes.

Further, the OLAP algebra we proposed is defined at the conceptual level, so it ignores implementation aspects, allowing to conceptually define any kind of data cube. We classified the operation into two groups, Instance Preserving Operators (IPOs) and Instance Generating Operations (IGOs). The former include the operations that preserve the cube instance, that means, the cuboids are not modified by the operations, and the semantics is defined by the input and output cuboids. Examples of IPOs are the Roll-up and Drill-down operations. The latter contains the operations that generate a new cube instance as a consequence of their application, i.e., Dice, Slice and Drill-across. Finally, we proposed an extension to the Drill-across operation based on the notion of semantic mapping between cubes, which allows us to solve the differences between dimension names, level names, members representations or even structure of dimension lattices. The semantic mapping between cubes is based on the semantic relationships dimension-dimension derivation and dimension-dimension association introduced by Abelló et al. [2]. We further extended the operator with the possibility of building new measures in the resulting cuboid by applying functions over the measures of the two input cuboids.

We also defined GOLAP-QL, a query language to manipulate the cubes, based on the OLAP algebra we proposed. The GOLAP architecture also includes a query optimization module, based on a set of rules that allows to automatically rewrite queries, merging and dropping operators to provide an efficient implementation. Importantly, the GOLAP engine is prepared to incorporate new data types that may be perceived as OLAP cubes. The first step is to generate a correct definition of the cube associated to the new data type, and then incorporate into the GOLAP engine a new polymorphic function for each operator over this new added type.

Our approach can thus support several kinds of data representations. As a

case study we showed the applicability of the model addressing spatial continuous data, namely continuous fields. In order to deal with continuous fields, we presented a discrete model for them at the logical level and a generalization of the well-known map algebra operations defined by Tomlin, redefining them in order to define a closed algebra, and to support different kinds of representations, since traditional map algebra operations only support raster representations. We modeled continuous fields as standard cubes with single-level dimensions which allows to manipulate such fields in the same way as we do with OLAP cuboids. We called these cubes FOLAP cubes. At the conceptual level, FOLAP cubes behave like a traditional OLAP cube and support all OLAP operations, which we also detailed in this thesis. In summary, a continuous field has an associated equivalent FOLAP cube, that can be seamlessly manipulated as any other traditional cube.

As a final contribution, we implemented a prototype which we used to run several queries over non homogeneous cubes, in the framework of a case study. Our proposal (as far as we are aware of, the first one of its kind) provides a very general framework for spatiotemporal data analysis and it can solve complex queries implementation by only manipulating discrete data cubes, regardless their underlying implementation, which remains hidden from the user.

11.2 Open Research Directions

In addition to the obvious direction which would be to solve typical implementation issues that can lead to an efficient system running the proposed solution, we believe that our proposal opens interesting possibilities in the emerging field of big data. In effect, we have used continuous field data as an example of the data types to be supported by our model. Many other kinds of complex data can be incorporated incrementally, each one having many issues to be solved. For example, semantic web data is an immediate candidate for this, but also image, video and music data warehouses can be developed based on our proposal. If all of these kinds of data could be combined in the same framework, and treated just as standard data cubes, we will be providing user a very powerful tool for data

analysis.

With respect to the GOLAP engine, the optimizer module can be extended to include not only rule-based optimization, but also cost-based optimization, in order to improve the time execution of queries and trend to a real time processing.

Along a different research line, we believe that the proposal also opens interesting possibilities for developing new extensible OLAP visualization tools, for example, to design visual interactive frameworks with usability design patterns. In this way, new visualization techniques appropriate for new data types that our framework allows to add, can be incorporated incrementally.

Appendices

Appendix A

Adapter Evaluator Algorithm for Semantic Checking

```
class FCQLVisitorAdapterEvaluator_SemanticChecking
        extends FCQLVisitorAdapter_SemanticChecking
{
    HashMap(Object, OLAPSchema> table= HashMap<Object, OLAPSchema > ();

    Object eval(Node node)
    {
        if (node is of type AssignStm)
        {
            // two childs: the name of the vble and the cube
            Object cubeSchema= lookForSchema(node.getRightChild()) ;
            table.put( node.getIzqChild(), cubeSchema );
            return cubeSchema;
        }
        // Pgm  (root)
     Node[] ch = node.children;
        Object result= null;
     for (Node aChild : ch)
     result= eval( aChild );  // an AssignStm

     return result;  // returns the last result
    }
```

```
Object lookForSchema( Node aNode)
{
    if (node is of type DICEExpression)
    {
        Node cubeSchema= getSchemaFromParameter(aNode.getLeftChild());
        Node diceExpression= aNode.getRightChild();
        // validDice checks if the dice expression use only measures,
        // valid functions, and current levels of dimensions of cubeSchema.
        // If not throws an exception.
        Node newCubeSchema= validDice (cubeSchema, diceExpression);
        return newCubeSchema;
    }

    if (node is of type SLICEExpression)
    {
        Node cubeSchema= getSchemaFromParameter(aNode.getLeftChild());
        String dimension= aNode.getRightChild();
        // validSlice checks if slicing a valid dimension or measure.
        // Checks also if the number of dimension is at least 1 and
        // the number of measures is at least 1.
        // If not throws an exception.
        Node newCubeSchema= validSlice( cubeSchema, dimension);
        return newCubeSchema;
    }

    if (node is of type ROLLUPExpression)
    {
        Node cubeSchema= getSchemaFromParameter(aNode.getLeftChild());
        String dimension= aNode.getMiddleChild();
        String level= aNode.getRightChild();
```

```
        // validRollup checks if the dimension exists and if it is
        // possible to rollup from current level to the target.
        // If not throws an exception.
        Node newCubeSchema= validRollup( cubeSchema, dimension, level);
        return newCubeSchema;
    }

    if (node is of type DRILLDOWNExpression)
    {
        // similarly to ROLLUPExpression
    }

    if (node is of type DRILLACROSSExpression)
    {
        Node cubeSchema1= getSchemaFromParameter(aNode.getLeftChild());
        Node cubeSchema2= getSchemaFromParameter(aNode.getMiddleChild());
        // optional parameter
        Node newCubeSchema= null;

        // Checks is both cube Schema corresponds to bottom DFileds...
        // If not throws an exception.
        if ( aNode.getRightChild() != null )
            newCubeSchema= validDA( cubeSchema1, cubeSchema2,
                                                aNode.getRightChild() );
        else
            newCubeSchema= validDA( cubeSchema1, cubeSchema2);

        return newCubeSchema;
    }
}

// Analyzes the rule :
// <Parameter> :=  <IDExp> | QUOTEDLITERAL | <OLAPExpression>
Node getSchemaFromParamter( Node paramater )
```

```
    {
        if (paramater is of type IDExp)  // name of a vble ?
        {
            // Given the vble, get the corresponding cubeschema
            // previously stored from the hash table
            Object cubeSchema= table.get(paramater);
            if (cubeSchema == null)
                throw Exception: trying to use a vble not previously initialized.
            return cubeSchema;
        }

        if (parameter is of type QUOTEDLITERAL)
        {
            //reads its schema from the XML cube schema.
            // If it does not exists, an exception is thrown
            Object cubeSchema= ReadXMLCubeSchema( parameter );
            return cubeSchema;
        }

        if (parameter is of type OLAPExpression)
        {
            // find out its schema by discovering
            // the schema of the nested expressions.
            return lookForSchema( parameter );
        }
    }
}
```

Bibliography

[1] Alberto Abelló, José Samos, and Fèlix Saltor, *Understanding facts in a multidimensional object-oriented model*, Proceedings of the 4th ACM International Workshop on Data Warehousing and OLAP (DOLAP 2001) (Atlanta, USA), 2001, pp. 32–39.

[2] Alberto Abelló, José Samos, and Fèlix Saltor, *On relationships offering new drill-across possibilities*, Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP (DOLAP 2002) (Virginia, USA), 2002, pp. 7–13.

[3] Alberto Abelló, José Samos, and Fèlix Saltor, *YAM2: A multidimensional conceptual model extending UML*, Information Systems **31** (2006), no. 6, 541–567.

[4] Rakesh Agrawal, Ashish Gupta, and Sunita Sarawagi, *Modeling multidimensional databases*, Proceedings of the 15th International Conference on Data Engineering (ICDE'97) (Birmingham, UK), IEEE Computer Society, 1997, pp. 232–243.

[5] Taher Ahmed and Maryvonne Miquel, *Multidimensional structures dedicated to continuous spatiotemporal phenomena*, Proceedings of the 22nd British National Conference on Databases (BNCOD 22) (Sunderland, England, UK), Springer, 2005, pp. 29–40.

[6] Taher Ahmed, Maryvonne Miquel, and Robert Laurini, *Continuous data warehouse: Concepts, challenges and potentials*, Proceedings of the 12th International Conference on Geoinformatics - Geospatial Information Research: Bridging the Pacific and Atlantic (Geoinformatics 2004) (Gävle, Sweden), 2004, pp. 157–164.

[7] Taher Omran Ahmed, *Continuous spatial data warehousing*, Proceedings of 9th International Arab Conference on Information Technology (Hammamet, Tunisia), 2008.

[8] Yvan Bédard, Tim Merret, and Jiawei Han, *Fundamentals of spatial data warehousing for geographic knowledge discovery*, Geographic data mining and knowledge discovery (2001), 53–73.

[9] Yvan Bédard, Sonia Rivest, and Marie-Josée Proulx, *Spatial online analytical processing (SOLAP): Concepts, architectures, and solutions from a geomatics*

engineering perspective, Data Warehouses and OLAP: Concepts, Architectures and Solutions (Wrembel-Koncilia, ed.), IRM Press, 2007, pp. 298–319.

[10] Sandro Bimonte and Myoung-Ah Kang, *Towards a model for the multidimensional analysis of field data*, Proceedings of the 14th East-European Conference on Advances in Databases and Information System (ADBIS'10) (Novi Sad, Serbia), Springer, 2010, pp. 58–72.

[11] Luca Cabibbo and Riccardo Torlone, *Querying multidimensional databases*, Proceedings of the 6th International Workshop on Database Programming Languages (Colorado, USA), Springer, 1997, pp. 253–269.

[12] Luca Cabibbo and Riccardo Torlone, *From a procedural to a visual query language for OLAP*, Proceedings of the 10th International Conference on Scientific and Statistical Database Management (Capri, Italy), IEEE Computer Society, 1998, pp. 74–83.

[13] Cristina Dutra de Aguiar Ciferri, Ricardo Rodrigues Ciferri, Leticia Gómez, Markus Schneider, Alejandro Vaisman, and Esteban Zimányi, *Cube algebra: A generic user-centric model and query language for OLAP cubes*, International Journal of Data Warehousing and Mining (IJDWM) **9** (2013), no. 2, 39–65.

[14] Matteo Golfarelli, Dario Maio, and Stefano Rizzi, *Conceptual design of data warehouses from E/R schema*, Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences (HICSS '98) (Washington DC, USA), IEEE Computer Society, 1998, pp. 334–343.

[15] Matteo Golfarelli and Stefano Rizzi, *A methodological framework for data warehouse design*, Proceedings of the 1st ACM International Workshop on Data Warehousing and OLAP (DOLAP'98) (New York, USA), ACM, 1998, pp. 3–9.

[16] Leticia Gómez, Silvia Gómez, and Alejandro Vaisman, *Analyzing continuous fields with OLAP cubes*, Proceedings of the 14th International Workshop on Data Warehousing and OLAP (DOLAP'11) (Glasgow, UK), ACM, 2011, pp. 89–94.

[17] Leticia Gómez, Silvia Gómez, and Alejandro Vaisman, *A generic data model and query language for spatiotemporal OLAP cube analysis*, Proceedings of the 15th International Conference on Extending Database Technology (EDBT '12) (Berlin, Germany), ACM, 2012, pp. 300–311.

[18] Leticia Gómez, Silvia Gómez, and Alejandro Vaisman, *Modeling and querying continuous fields with OLAP cubes*, International Journal of Data Warehousing and Mining (IJDWM) **9** (2013), no. 3, 22–45.

[19] Leticia Gómez, Alejandro Vaisman, and Sebastián Zich, *Piet-QL: a query language for GIS-OLAP integration*, Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems(GIS'08) (New York, NY, USA), ACM, 2008.

[20] Leticia Gómez, Alejandro Vaisman, and Esteban Zimányi, *Physical design and implementation of spatial data warehouses supporting continuous fields*, Proceedings of the 12th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'10) (Bilbao, Spain), Springer, 2010, pp. 25–39.

[21] Ralf Güting, Michael Böhlen, Martin Erwig, Christian Jensen, Nikos Lorentzos, Markus Schneider, and Michalis Vazirgiannis, *A foundation for representing and quering moving objects*, ACM Transactions on Database Systems **25** (2000), no. 1, 1–42.

[22] Marc Gyssens and Laks Lakshmanan, *A foundation for multi-dimensional databases*, Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB '97) (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 1997, pp. 106–115.

[23] Carlos Hurtado, Alberto Mendelzon, and Alejandro Vaisman, *Maintaining data cubes under dimension updates*, Proceedings of the 15th International Conference on Data Engineering (ICDE '99) (Washington, DC, USA), IEEE Computer Society, 1999, pp. 346–355.

[24] Gregory Jones, *Climate and terroir: Impacts of climate variability and change on wine*, Fine Wine and Terroir-The Geoscience Perspective **9** (2006), 1–14.

[25] Gregory Jones, Michael White, Owen Cooper, and Karl Storchmann, *Climate change and global wine quality*, Climatic Change **73** (2005), no. 3, 319–343.

[26] Ralph Kimball, *The data warehouse toolkit*, J. Wiley and Sons, Inc., 1996.

[27] Ralph Kimball and Margy Ross, *The data warehouse toolkit: The complete guide to dimensional modeling*, 2nd ed., John Wiley & Sons, Inc., 2002.

[28] Mark Kumler, *An intensive comparison of triangulated irregular networks (TINs) and digital elevation models (DEMs)*, Cartographica: The International Journal for Geographic Information and Geovisualization **31** (1994), 45.

[29] Hugo Ledoux and Christopher Gold, *Interpolation as a tool for the modelling of three-dimensional geoscientific datasets*, Proceedings 4th ISPRS Workshop on Dynamic and Multi-dimensional GIS (DMGIS'05) (Wales, UK), 2005, pp. 79–84.

[30] Hugo Ledoux and Christopher Gold, *A voronoi-based map algebra*, Progress in Spatial Data Handling (Andreas Riedl, Wolfgang Kainz, and Gregory A. Elmes, eds.), Springer Berlin Heidelberg, 2006, pp. 117–131.

[31] Der-Tsai Lee and Brian Schachter, *Two algorithms for constructing a delaunay triangulation*, International Journal of Computer and Information Sciences **9** (1980), no. 3, 219–242.

[32] Hans Lenz and Arie Shoshani, *Summarizability in OLAP and statistical data bases*, Proceedings of the 9th International Conference on Scientific and Statistical Database Management (Washington, USA), IEEE Computer Society, 1997, pp. 132–143.

[33] Elzbieta Malinowski and Esteban Zimányi, *Advanced data warehouse design: From conventional to spatial and temporal applications*, Springer, 2008.

[34] Patrick Marcel, *Modeling and querying multidimensional databases: An overview*, Networking and Information Systems Journal **2** (1999), 515–548.

[35] Jose-Norberto Mazón, Jens Lechtenbörger, and Juan Trujillo, *A survey on summarizability issues in multidimensional modeling*, Data & Knowledge Engineering **68** (2009), no. 12, 1452–1469.

[36] Jeremy Mennis, *Multidimensional map algebra: Design and implementation of a spatio-temporal GIS processing language*, Transactions in GIS **14** (2010), no. 1, 1–21.

[37] Jeremy Mennis and Roland Viger, *Analyzing time series of satellite imagery using temporal map algebra*, Proceedings of the American Society for Photogrammetry and Remote Sensing Annual Conference (ASPRS) (Denver, USA), 2004.

[38] Jeremy Mennis, Roland Viger, and Dana Tomlin, *Cubic map algebra functions for spatio-temporal analysis*, Cartography and Geographic Information Science **32** (2005), no. 1, 17–32.

[39] Thanh Nguyen and A Min Tjoa, *An object oriented multidimensional data model for OLAP*, Proceedings of 1st International Conference on Web-Age Information Management (WAIM), number 1846 in LNCS (Shangai, China), Springer, 2000, pp. 69–82.

[40] Jesús Pardillo, Jose-Norberto Mazón, and Juan Trujillo, *Bridging the semantic gap in OLAP models: platform-independent queries*, Proceeding of the ACM 11th international workshop on Datawarehousing and OLAP (DOLAP '08) (California, USA), ACM, 2008, pp. 89–96.

[41] Jesús Pardillo, Jose-Norberto Mazón, and Juan Trujillo, *Extending OCL for OLAP querying on conceptual multidimensional models of data warehouses*, Information Sciences **180** (2010), no. 5, 584–601.

[42] Philippe Rigaux, Michel Scholl, and Agnés Voisard, *Spatial databases*, Morgan Kaufmann, 2002.

[43] Sonia Rivest, Yvan Bédard, and Pierre Marchand, *Toward better suppport for spatial decision making: Defining the characteristics of spatial on-line analytical processing (SOLAP)*, Geomatica **55** (2001), 539–555.

[44] Oscar Romero and Alberto Abelló, *On the need of a reference algebra for OLAP*, Proceedings of the 12th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'07) (Regensburg, Germany), Springer, 2007, pp. 99–110.

[45] Jayavel Shanmugasundaram, Usama Fayyad, and Paul Bradley, *Compressed data cubes for OLAP aggregate query approximation on continuous dimensions*, Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99) (San Diego, California, USA), ACM, 1999, pp. 223–232.

[46] Sashi Shekhar, Chang tien Lu, Xinhong Tan, Sanjai Chawla, and Ranga Vatsavai, *MapCube: A visualization tool for spatial data warehouses*, Geographic data mining and Knowledge Discovery (GKD) (Harvey Miller and Jiawei Han, eds.), Taylor and Francis, 2001, pp. 74–109.

[47] Robert Sibson, *Interpreting multivariate data*, ch. A brief description of natural neighbour interpolation, Wiley, New York, USA, 1981.

[48] Dana Tomlin, *Geographic information systems and cartographic modelling*, Prentice-Hall, 1990.

[49] Nectaria Tryfona, *Modeling phenomena in spatiotemporal databases: Desiderata and solutions*, Proceedings of the 9th Database and Expert Systems Applications (DEXA) (Vienna, Austria), Springer, 1998, pp. 155–165.

[50] Aris Tsois, Nikos Karayannidis, and Timos Sellis, *MAC: Conceptual data modeling for OLAP*, Porceedings of the 3rd International Workshop on Design and Management of Data Warehouses (DMDW 2001) (Interlaken, Switzerland), CEUR-WS, 2001.

[51] Alejandro Vaisman and Esteban Zimányi, *A multidimensional model representing continuous fields in spatial data warehouses*, Proceedings of the 17th ACM International Conference on Advances in Geographic Information Systems (GIS'09) (Seattle, Washington), ACM, 2009, pp. 168–177.

[52] Tom van der Putte and Hugo Ledoux, *Modelling three-dimensional geoscientific datasets with the discrete voronoi diagram*, Advances in 3D Geo-Information Sciences (Thomas H. Kolbe, Gerhard Köning, and Claus Nagel, eds.), Lecture Notes in Geoinformation and Cartography, Springer Berlin Heidelberg, 2011, pp. 227–242.

[53] Panos Vassiliadis, *Modeling multidimensional databases, cubes and cube operations*, Proceedings of the 10th International Conference on Scientific and Statistical Database Management (SSDBM '98) (Capri, Italy), IEEE Computer Society, 1998, pp. 53–62.

[54] Panos Vassiliadis and Timos Sellis, *A survey of logical models for OLAP databases*, ACM SIGMOD Record (ACM Special Interest Group on Management of Data) **28** (1999), no. 4, 64–69.

[55] Michael Worboys, *GIS: A computing perspective*, Taylor & Francis, 1995.