Universidade Estadual de Campinas
Instituto de Computação

Jaudete Daltio

Views over Graph Databases: A Multifocus Approach for Heterogeneous Data

Visões em Bancos de Dados de Grafos: Uma Abordagem Multifoco para Dados Heterogêneos

CAMPINAS
2017

Jaudete Daltio

Views over Graph Databases: A Multifocus Approach for Heterogeneous Data

Visões em Bancos de Dados de Grafos: Uma Abordagem Multifoco para Dados Heterogêneos

Tese apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Doutora em Ciência da Computação.

Thesis presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Supervisor/Orientadora: Profa. Dra. Claudia Maria Bauzer Medeiros

Este exemplar corresponde à versão final da Tese defendida por Jaudete Daltio e orientada pela Profa. Dra. Claudia Maria Bauzer Medeiros.

CAMPINAS
2017

Agência(s) de fomento e nº(s) de processo(s): Não se aplica.

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

D17v

Daltio, Jaudete, 1983-
Views over graph databases : a multifocus approach for heterogeneous data / Jaudete Daltio. – Campinas, SP : [s.n.], 2017.

Orientador: Claudia Maria Bauzer Medeiros.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Banco de dados. 2. Grafo (Sistema de computador). 3. Modelagem de dados. 4. Gerenciamento da informação. I. Medeiros, Claudia Maria Bauzer,1954-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Visões em bancos de dados de grafos : uma abordagem multifoco para dados heterogêneos
Palavras-chave em inglês:
Databases
Graph (Computer system)
Data modeling
Information management
Área de concentração: Ciência da Computação
Titulação: Doutora em Ciência da Computação
Banca examinadora:
Claudia Maria Bauzer Medeiros [Orientador]
Ana Carolina Brandão Salgado
Ricardo Rodrigues Ciferri
André Santanchè
Guilherme Pimentel Telles
Data de defesa: 04-09-2017
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação

Jaudete Daltio

Views over Graph Databases: A Multifocus Approach for Heterogeneous Data

Visões em Bancos de Dados de Grafos: Uma Abordagem Multifoco para Dados Heterogêneos

Banca Examinadora:

- Profa. Dra. Claudia Maria Bauzer Medeiros
  Instituto de Computação/ Universidade Estadual de Campinas
- Profa. Dra. Ana Carolina Brandão Salgado
  Centro de Informática/ Universidade Federal de Pernambuco
- Prof. Dr. Ricardo Rodrigues Ciferri
  Departamento de Computação/ Universidade Federal de São Carlos
- Prof. Dr. André Santanchè
  Instituto de Computação/ Universidade Estadual de Campinas
- Prof. Dr. Guilherme Pimentel Telles
  Instituto de Computação/ Universidade Estadual de Campinas

A ata da defesa com as respectivas assinaturas dos membros da banca encontra-se no processo de vida acadêmica do aluno.

Campinas, 04 de setembro de 2017

Two roads diverged in a wood, and I-
I took the one less traveled by,
And that has made all the difference.

(The Road Not Taken, Robert Frost)

Agradecimentos

A professora Claudia, por ter sido tudo o que eu precisei ao longo desse processo. Muito mais do que uma professora. Muito mais do que uma orientadora. Por acreditar que eu era capaz. Pela confiança, pela atenção, pela paciência e pelas críticas. É um grande previlégio tê-la em minha vida.

Ao Cristiano, por sempre estar ao lado. Pelo incentivo e pela compreensão. Infinita compreensão. Por me ajudar a enfrentar todos os desafios, principalmente aqueles que achei que não seria capaz de superar. Por fazer de mim uma pessoa melhor a cada dia.

A minha família, pelo apoio e pela confiança. Aos meus queridos irmãos, pela constante preocupação. Aos meus pais, Hermes e Bernadete, lembrados com muito carinho e sempre presentes em meu coração. E Dante e Fernanda, meus pequenos amores, pelos grandes sorrisos.

Aos amigos que estiveram sempre comigo ao longo dessa jornada. É uma dádiva ter tantas pessoas maravilhosas em minha vida com quem compartilhar minhas conquistas. Em especial aos amigos do Laboratório de Sistemas de Informação – LIS, pelas opniões, discussões e contribuições neste trabalho.

Ao especialista em geoprocessamento da Agência Nacional de Águas (ANA), Alexandre de Amorim Teixeira, pelo auxílio no acesso e na interpretação dos dados utilizados na pesquisa.

Aos membros da banca, pelas sugestões e contribuições no trabalho.

À Empresa Brasileira de Pesquisa Agropecuária (EMBRAPA) e às agências de fomento CAPES, CNPq e projetos, FAPESP/CEPID in Computational Engineering and Sciences (2013/08293-7), INCT in Web Science pelo apoio direto ou indireto na realização deste trabalho.

Resumo

A pesquisa científica tornou-se cada vez mais dependente de dados. Esse novo paradigma de pesquisa demanda técnicas e tecnologias computacionais sofisticadas para apoiar tanto o ciclo de vida dos dados científicos como a colaboração entre cientistas de diferentes áreas. Uma demanda recorrente em equipes multidisciplinares é a construção de múltiplas perspectivas sobre um mesmo conjunto de dados. Soluções atuais cobrem vários aspectos, desde o projeto de padrões de interoperabilidade ao uso de sistemas de gerenciamento de bancos de dados não-relacionais. Entretanto, nenhum desses esforços atende de forma adequada a necessidade de múltiplas perspectivas, denominadas *focos* nesta tese. Em termos gerais, um foco é projetado e construído para atender um determinado grupo de pesquisa (mesmo no escopo de um único projeto) que necessita manipular um subconjunto de dados de interesse em múltiplos níveis de agregação/generalização. A definição e criação de um foco são tarefas complexas que demandam mecanismos capazes de manipular múltiplas representações de um mesmo fenômeno do mundo real.

O objetivo desta tese é prover múltiplos focos sobre dados heterogêneos. Para atingir esse objetivo, esta pesquisa se concentrou em quatro principais problemas. Os problemas inicialmente abordados foram: (1) escolher um paradigma de gerenciamento de dados adequado e (2) elencar os principais requisitos de pesquisas multifoco. Nossos resultados nos direcionaram para a adoção de bancos de dados de grafos como solução para o problema (1) e a utilização do conceito de visões, de bancos de dados relacionais, para o problema (2). Entretanto, não há consenso sobre um modelo de dados para bancos de dados de grafos e o conceito de visões é pouco explorado nesse contexto. Com isso, os demais problemas tratados por esta pesquisa são: (3) a especificação de um modelo de dados de grafos e (4) a definição de um *framework* para manipular visões em bancos de dados de grafos. Nossa pesquisa nesses quatro problemas resultaram nas contribuições principais desta tese: (i) apontar o uso de bancos de dados de grafos como camada de persistência em pesquisas multifoco – um tipo de banco de dados de esquema flexível e orientado a relacionamentos que provê uma ampla compreensão sobre as relações entre os dados; (ii) definir visões para bancos de dados de grafos como mecanismo para manipular múltiplos focos, considerando operações de manipulação de dados em grafos, travessias e algoritmos de grafos; (iii) propor um modelo de dados para grafos – baseado em grafos de propriedade – para lidar com a ausência de um modelo de dados pleno para grafos; (iv) especificar e implementar um framework, denominado Graph-Kaleidoscope, para prover o uso de visões em bancos de dados de grafos e (v) validar nosso framework com dados reais em aplicações distintas – em biodiversidade e em recursos naturais – dois típicos exemplos de pesquisas multidisciplinares que envolvem a análise de interações de fenômenos a partir de dados heterogêneos.

Abstract

Scientific research has become data-intensive and data-dependent. This new research paradigm requires sophisticated computer science techniques and technologies to support the life cycle of scientific data and collaboration among scientists from distinct areas. A major requirement is that researchers working in data-intensive interdisciplinary teams demand construction of multiple perspectives of the world, built over the same datasets. Present solutions cover a wide range of aspects, from the design of interoperability standards to the use of non-relational database management systems. None of these efforts, however, adequately meet the needs of multiple perspectives, which are called *foci* in the thesis. Basically, a focus is designed/built to cater to a research group (even within a single project) that needs to deal with a subset of data of interest, under multiple aggregation/generalization levels. The definition and creation of a focus are complex tasks that require mechanisms and engines to manipulate multiple representations of the same real world phenomenon.

This PhD research aims to provide multiple foci over heterogeneous data. To meet this challenge, we deal with four research problems. The first two were (1) choosing an appropriate data management paradigm; and (2) eliciting multifocus requirements. Our work towards solving these problems made us choose graph databases to answer (1) and the concept of views in relational databases for (2). However, there is no consensual data model for graph databases and views are seldom discussed in this context. Thus, research problems (3) and (4) are: (3) specifying an adequate graph data model and (4) defining a framework to handle views on graph databases. Our research in these problems results in the main contributions of this thesis: (i) to present the case for the use of graph databases in multifocus research as persistence layer – a schemaless and relationship driven type of database that provides a full understanding of data connections; (ii) to define views for graph databases to support the need for multiple foci, considering graph data manipulation, graph algorithms and traversal tasks; (iii) to propose a property graph data model (PGDM) to fill the gap of absence of a full-fledged data model for graphs; (iv) to specify and implement a framework, named Graph-Kaleidoscope, that supports views over graph databases and (v) to validate our framework for real world applications in two domains – biodiversity and environmental resources – typical examples of multidisciplinary research that involve the analysis of interactions of phenomena using heterogeneous data.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Motivation

Increasingly, the world of science is being changed, induced by the advances of information technology. Scientific data is being produced and collected at an unprecedented scale and outpaces the speed with which it can be analyzed and understood [41]. Computer science has become a key element in scientific research in many areas [24], such as bioinformatics [28], social computing [82] and health [61]. Data-intensive science is a new paradigm for scientific exploration [51] and involves the capture, curation and analysis of large amounts of data. It requires sophisticated computer science techniques and technologies to support all steps in these activities.

Data-intensive science is usually multidisciplinary and demands collaboration among scientists from distinct areas. It is also characterized by the use of large amounts of data, captured by instruments or generated by simulations that are produced at all scales with different quality levels. The main computer science challenges are targeted by management and analysis mechanisms. The volume of these datasets brings a high complexity to interpretation, compounded by the large quantities of variables available. Standard database systems have limitations to deal with such datasets, in which data are unstructured and often come from networks with complex relationships between their entities [36]. Most data management tools are designed with retrieval efficiency in mind, highly dependent on the data model used, leaving data exploration as a secondary role [40].

A particular issue involves letting researchers work with the data subset of interest, under a specific aggregation/generalization level, a given perspective and a specific vocabulary. This problem is addressed by [74], which defines a *focus* as a perspective of the study of a given problem, where data can be restricted to one specific scale/representation, zooming in and out, e.g., hiding or revealing details. Additionally, a focus can put together objects from distinct perspectives.

Given the same set of data, distinct foci will also arise when the data are analyzed under different models, processed using focus-specific algorithms, or even visualized with particular means. The definition and creation of a focus are complex tasks that require mechanisms and engines to manipulate multiple representations of the same real world phenomenon.

1.2 Problem Statement and Research Problems

This thesis aims at answering the following research question: "How can we provide multiple foci over heterogeneous data collections ?" There are two issues involved: (i) how to deal with heterogeneous data and (ii) how to provide multiple foci. Due to the complexity involving both subjects, we start our research exploring two research problems: (1) eliciting multifocus requirements; (2) choosing an appropriate data management paradigm for handle heterogeneous data.

Problem 1: Eliciting Multifocus Requirements

The basic idea of multifocus work is to support construction of arbitrary perspectives of a given dataset. In some cases, a simple operation is enough – just to restrict a subset of the variables available or to restrict to data having some property. The challenge arises when there is need for data transformation – e.g., in terms of aggregating or disaggregating parts of the data or combining data from different perspectives. Data transformation may also require rearrangement of data according to the semantic relations among data. Whether using simple or complex operations, the result is a representation of the same real world phenomena under different perspectives.

In geographic data research, scale studies are often considered as (geographic) scale transformations, but the ideas can be extrapolated. They usually apply abstraction levels to describe the original data [76], generalization algorithms [85] and identify, from the underlying dataset, which elements are relevant to a given perspective [67]. Related work about semantic transformations usually applies ontologies (i.e., abstract model of terms [45]) as a central role to provide a perspective of the data [46]. The use of ontologies helps to solve interoperability issues in knowledge representation [52], allowing to effectively share data in research communities [7, 9, 21]. A different approach appears in [60], which treated each perspective as a version of the dataset.

In this thesis, two different real world datasets used in interdisciplinary research were analyzed to gather the essential data transformations of multifocus research. The first one concerns biodiversity data – a dataset of recordings of animal sounds [29]. The dataset includes all observation metadata (54 attributes), such as information about the species (taxonomy, gender), the place where the sound was recorded (geographic coordinates, biome), the recording and digitalization devices, data and time of the observation, and so on. In this dataset, a focus might concern, for instance, geographical location, a set of natural conditions (biome), a group of species or a period of time [35].

The second dataset involved environmental data – a water resources database covering the Brazilian waterways. The dataset includes all elements of the drainage network – a set of drainage points and stretches – and attributes about the rivers, hydrographic catchment areas, watersheds and main watercourses. A focus might concern, for instance, the connectivity of rivers, the influence of drainage stretches or watersheds [33].

The results of this problem led us to choose views as an appropriate means to construct a focus.

Problem 2: Choosing an Appropriate Data Management Paradigm

The continued growth of data and its heterogeneity led to the emergence of new database models and technologies – the *NoSQL* databases. These databases do not have all properties of traditional relational databases and are generally not queried with SQL [50]. Related work classifies NoSQL databases in four categories, according to their data model: (i) key value stores, in which a piece of data is addressed by a unique key and it is isolated from remaining data [78]; (ii) document stores, in which data is stored in an interoperable document (JSON, for instance) addressed by a unique key [4]; (iii) column oriented stores, which create a sparse sorted map in which rows store an arbitrary number of key-value pairs [23, 37]; and (iv) graph stores, in which data are stored as nodes, edges and properties [83].

Each NoSQL data model attempts to solve a particular data management issue and the diversity of these models shows that there is no data model or database able to deal with all challenges [72]. However, with few exceptions, graph stores are the only capable of handling relations among data [50]. Indeed, the interpretation of complex datasets usually requires understanding data connections, interactions with other data and topological properties about data organization. Besides, graph stores are schemaless, in the sense that it is not necessary to first create a schema and then insert data - schema and data are inserted together. Nevertheless, it is possible to add properties to each individual vertex or edge at any time, an important requirement to deal with heterogeneous data.

Given these considerations, we chose graphs as our data management paradigm.

Research Overview

Figure 1.1 provides an overview of our research, presenting how the research problems are correlated under the graph data management paradigm. Each research problem is highlighted in the figure (indicated by its number).

Raw data are represented in the bottom. The left side of the figure shows the user perception of the graph data management paradigm: data in the database are understood to be organized in a graph, in the simplest form $G = (V, E)$. Each application has to build its own "stack of standards", shown in the right side of the figure, to deal with graph database system issues. Since there is no consensus on a formal graph data model or even for a graph data structure, there are many possible "stacks of standards" available in the literature to meet this architecture. Applications and users needs of multifocus research are represented on the top of both logical and physical perceptions.

Research problem **(1)** concerns applications and user needs in multifocus research. Problem **(2)** involves the data management paradigm and the datasets explored. To achieve our goal, two additional research problems needed to be explored: (3) adopting an adequate graph data model and (4) defining a framework to handle views on graph databases. Research problem **(3)** concerns about physical issues of graph data management, in terms of graph data model. Research problem **(4)** concerns about how to provide the user perception of views over graphs.

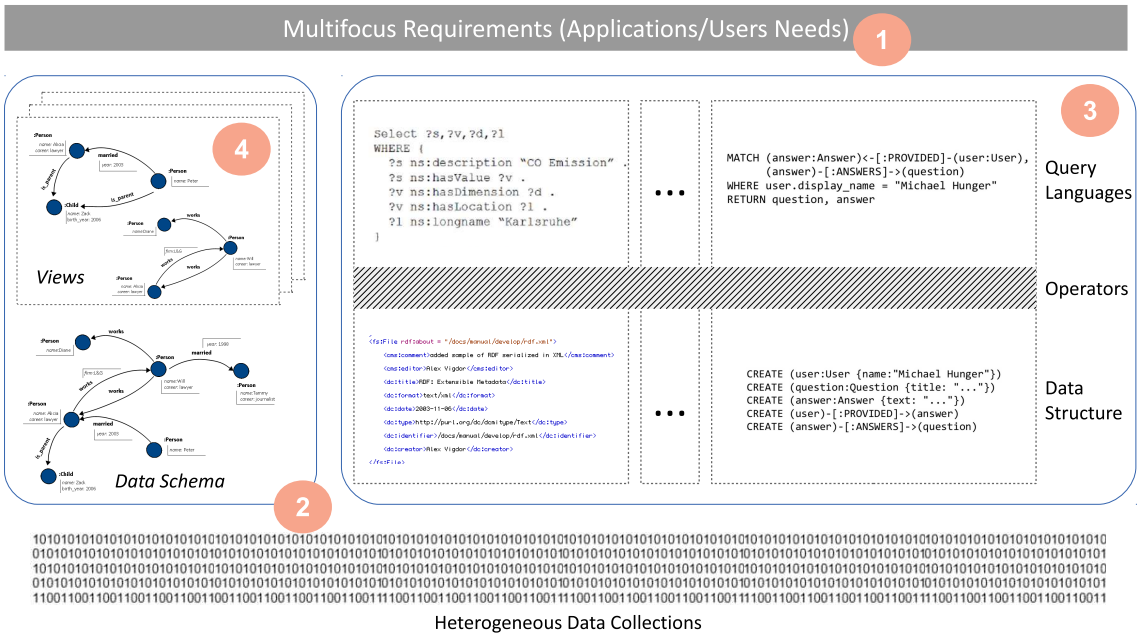


Figure 1.1: Overview of the Research Problems

Problem 3: Adopting an Adequate Graph Data Model

While graph stores are being increasingly adopted in cases where relationships among data elements are first-class citizens, graph database systems are at the same overall level of maturity as object-oriented database systems were in the early to mid-90's. The term "graph database systems" refers to systems that are both graph stores and DBMS in terms of being ACID compliance systems.

To explain our point of view, we explicitly paraphrased two sentences of the classical "Object-Oriented Manifesto" [8], concerning the state of the art of OO databases in 1989, when that paper first appeared, and which we repeat here.*"Three points characterize the field at this stage: (i) the lack of a common data model, (ii) the lack of formal foundations and (iii) strong experimental activity. Whereas Codd's original paper [25] gave a clear specification of a relational database system (data model and query language), no such specification exists for object-oriented database systems"* (Manifesto page 2). If we now replace the term "object-oriented" by "graph", the entire sentence holds.

Related work about (i), lack of data model, involves at least three different data structures: RDF[62, 30], property graphs [1, 72, 71] and hypergraphs [58, 70, 57]. A discussion about graph query languages can be found in [27, 39, 19, 20, 10, 62, 84, 59]. As far as we know, no related work deeply discusses integrity rules. The middle physical layer, represented in Figure 1.1 **(3)** as **Operators**, is almost unexplored by related work.

In this scenario, the lack of formal foundations (item (ii)) can be observed. There are plenty of research about graphs as data structure. In fact, graphs are widely studied in computer science, with a very strong formal foundation and algorithms. The gap that we refer to here concerns the formal foundations of the use of graphs as data model for a database, as opposed to formal foundations of relational databases. The strong experimental activity mentioned in (iii) can be easily observed in the site *DB-Engines*

Ranking [1], that presents monthly rankings of over 300 DBMS. There, graph DBMS is an independent category of database management systems and currently ranks 26 software solutions.

Due to this lack of consensus in formalization, one of the challenges addressed by our research was to formalize the graph data model that underpins our research in terms of Codd's principles [25]. That means that we had to determine a graph data structure, integrity rules and the elementary operations [75]. This resulted in our Property Graph Data Model (PGDM), presented in the thesis. As will be seen in Chapter 4, these operations are used to generate ours views on graph databases.

Problem 4: Defining a Framework to Handle Views

The first proposal for views over graphs appeared in the 80's, together with the first graph data models [56]. With the arrival of graph stores, some view mechanisms have started to appear since 2014.

The major weakness of all related work about views on graphs is the limited kind of data transformation allowed. Few approaches adopt the classical idea, borrowed from relational databases, of performing queries in graphs to extract views [61]. This idea, though intuitive, is itself a challenge, since a graph query answer may not be a graph. The more broad approach is to use graph patterns to extract views, which are nevertheless only capable of performing simple data restrictions [38, 11].

Given this scenario, we propose the formalization of views in graph databases. We specify a framework to handle multiple views over graph databases. This resulted in Graph-Kaleidoscope, presented in the thesis. Though the notion of views as perspectives (foci) does not depend on the underlying system, view mechanisms are strongly dependent on the underlying model. For this reason, our framework is based on our PGDM Data Model.

1.3 Contributions

This PhD research resulted in five main contributions, summarized as follows:

- To present the case for the use of graph databases in multifocus research. From a survey of data management requirements, we justify the use of graph databases as a suitable persistence layer to meet these requirements and to store/analyze datasets of highly connected data. This contribution is a result of the research problems 1 and 2 and it is presented in Chapter 2;

- To propose a property graph data model (PGDM) with a set of operators to manipulate and retrieve graph data. Ours is a flexible approach, to be incorporated in any graph data structure and query language. This is the main contribution of our thesis, motivated by research problem 3 and proposed to fill the gap of the absence

[1] db-engines.com/en/ranking/graph+dbms

of a full-fledged data model for graph databases. Results are presented in Chapter 4;

- To define views for graph databases to support the need for multiple perspectives, motivated by research problems 1 and 4. Views are specified through view generating functions, considering graph data manipulation, classical graph algorithms and traversal tasks. This is introduced in Chapter 2 and formalized in Chapter 4;

- To analyze real life examples of interdisciplinary research, showing how they can benefit from our proposal. We present how biodiversity and environmental resource datasets can be modeled and explored by experts using graph databases and multiple perspectives, pointing out the advantages of this approach. This contribution converged from all the research problems explored, and is presented in Chapters 2, 3 and 4;

- The specification and implementation of a prototype of the Graph-Kaleidoscope framework to support views over graph databases in a graph software engine. This contribution is a result of research problem 4 and is presented in Chapter 4.

1.4 Thesis Organization

This chapter presented the motivation, goal, research problems and main contributions of this PhD research. The remainder of this text is organized as a collection of papers, as follows.

Chapter 2 corresponds to the paper "Handling Multiple Foci in Graph Databases", published in the Proceedings of the 10th International Conference on Data Integration in Life Science [35]. This chapter discusses the requirements of the multifocus research (research problem 1) targeted to the biodiversity domain (research problem 2). It also presents the first version of our framework (research problem 4) considering two main approaches to building views on graphs: general queries and exploration around a central concept. These two approaches were subsequently aggregated in our graph operators. This chapter also presents how biodiversity studies of animal observations can be benefit from our research.

Chapter 3 corresponds to the paper "Hydrograph: Exploring Geographic Data In Graph Databases", published in the Brazilian Journal of Cartography [33]. This chapter presents our progress in the requirements of multifocus research (research problem 1) by exploring another interdisciplinary domain – environmental research on hydrography. This progress confirmed our solution to the research problem 2 and the choice of graphs as data management paradigm. This chapter presents our software technology choice for our framework, Neo4j [2], and presents all steps for data migration from relational to graph database while maintaining application semantics. We also present a set of important analysis operations recurrently performed in this dataset and how they can benefit from the graph paradigm and the idea of views.

[2] www.neo4j.com

Chapter 4 corresponds to the paper "Graph-Kaleidoscope: A Framework to Handle Multiple Perspectives in Graph Databases", submitted to the International Journal of Data Science and Analytics [34]. This paper, currently under review, describes our definition of a graph data model (research problem 3), PGDM, based on the property graph data structure. The most important part of our definition are operators, a middle layer between graph data structures and query languages. Our operators are defined in a generic way and they can be implemented in different stack of standards of physical graph representations. This chapter also presents the specification and the prototype of Graph-Kaleidoscope (research problem 4). This is an improved version of the framework presented in Chapter 2 and adopts our operators to define the view generating function. Some challenges faced and lessons learned are presented at the end of this chapter.

Chapter 5 contains conclusions and some directions for future work.

Besides the papers in Chapters 2, 3 and 4, others were also published in the course of this thesis, directly related to this research. There follows a list of publications, including the ones that compose the thesis.

- J. Daltio and C. B. Medeiros. Handling multiple foci in graph databases. In *Proceedings of 10th International Conference on Data Integration in the Life Sciences*, volume 8574, pages 58-65, Lisboa, Portugal, 2014.

- J. Daltio and C. B. Medeiros. HydroGraph: Exploring Geographic Data in Graph Databases. In *Proceedings XVI Brazilian Symposium on Geoinformatics - GeoInfo*, Brazil, 2015.

- J. Daltio and C. B. Medeiros. HydroGraph: Exploring Geographic data in Graph Databases (extended version of the 2015 GeoInfo). In *Brazilian Journal of Cartography*, volume 68, number 6, 2016.

- J. Daltio and C. B. Medeiros. A view handler for semantic graphs. In *Proceedings of the IEEE 10th International Conference on Semantic Computing*, pages 1-5, Los Angeles, 2016.

- J. Daltio and C. B. Medeiros. Graph-Kaleidoscope: A Framework to Handle Multiple Perspectives in Graph Databases. In *International Journal of Data Science and Analytics*, 2017 (*under review*).

Chapter 2

Handling Multiple Foci in Graph Databases

2.1 Introduction and Motivation

eScience, sometimes used as a synonym for data-intensive science [51], is characterized by joint research in computer science and other fields to support the whole research cycle – from data collection, mining, and visualization to data sharing. Biodiversity research – our target domain – is a good example of eScience. It is a multidisciplinary field that requires associating data about living beings and their habitats, constructing models to describe species' interactions and correlating different information sources. Such data includes information on environmental and ecological factors, as well as on species, and includes images, text, video and sound recordings [29], in multiple spatial and temporal scales.

Sharing and reuse of data are hampered by the heterogeneity of data and user requirements inherent to such domains. Each community applies different data extraction and processing methodologies and has distinct research perspectives and vocabularies. Several researchers have adopted graph representations (and graph database systems) as a computational means to deal with such integration challenges [68], especially in situations where relations among data and the data itself are at the same importance level [6].

However, graph database systems present limitations when it comes to creating and processing multiple perspectives of the underlying data. This paper presents our approach to these issues, which consists of a conceptual framework that allows experts to specify and construct arbitrary perspectives on top of graph databases. This framework, under construction, takes advantage of some of our previous implementation work, in particular concerning ontology management [31]. Informally, the idea is to support a notion similar to that of database views, constructed on top of graph databases. However, our constructs go beyond standard database views.

Here, we follow the terminology we introduced in [74], and use the term *focus* for such views. Intuitively, a *focus* is a perspective of study of a given problem, where data can be restricted to one specific scale/representation, or put together objects from distinct scales. Moreover, given the same set of data, distinct foci will arise when the data is analyzed

under different models, processed using focus-specific algorithms, or even visualized with particular means.

This paper has two main contributions. The first is to explore the notion of views on graph database systems, which is not yet supported in such systems. This requires extending the traditional specification of views, while at the same time maintaining the same principles. The second contribution is to show, via the running example, how to model and create multiple foci, for biodiversity research, thereby allowing experts to manage and analyze the same underlying datasets under arbitrary perspectives.

2.2 Theoretical Foundations and Related Work

2.2.1 Graph Databases

Graph databases allow to represent information about the connectivity of unstructured data – a recurrent scenario in scientific research. The interpretation of scientific data usually requires the understanding about linked data, interactions with other data and topological properties about data organization.

The formal foundation of all graph data structures is based on the mathematical definition of graphs and, on top of this basic layer, several graph data structures were proposed [6, 71], including features such as directed or undirected edges, labeled or unlabeled edges and hypernodes. One of the most popular structures supported by many graph database systems is the *property graph*. It tries to arrange all the features that these graph types express in a single and flexible structure through key-value pairs to describe vertex and edge characteristics, such as type, label or direction.

To manipulate these data, graph query languages can be used to [84]: (i) find vertices that satisfy a pattern; (ii) find pairs (x, y) of vertices such that there is a path from x to y whose sequence of edge labels matches some pattern; (iii) express relations among paths; (iv) compute aggregate functions based on graph properties; and (v) create new elements. Each query language has its own syntax and considers its own data structure to represent a graph.

2.2.2 Views

In the context of relational databases, a *view* can be regarded as a temporary relation against which database requests may be issued [42]. Views are widely used to restrict, protect or reorganize relational data. Views are built by a combination of operations applied on the underlying relations, creating alternative or composite representations of existing database objects. The sequence of operations that creates a particular view is called *view generating function*.

The concept of view is used in many data management contexts. A *view of an ontology* is a subset of the original ontology, built by the extraction of some relevant parts thereof. Tools and languages for ontologies usually take advantage of their graph structure; vertices represent classes and instances and edges represent properties, relations and class hierarchies. There are different approaches to create ontology views [65]. Some are

based on query languages and others are based on guidelines to navigate through ontology concepts, using the notion of *central concept* – a class around which the view is built and that defines which elements must be part of a view. Different from databases in which a query always results in an instance set, a query on an ontology can result in a partial schema (classes, relations), an instance set or a combination of both [31].

2.2.3 Multifocus Research

The notion of *focus* (a perspective of study of a given problem) appears naturally in eScience. The idea behind a focus is similar to the idea of an application – each application has its own perception of the world, goal, complexity and specific requirements. For the same underlying datasets, each focus represents a perception of the data, how it can be analyzed, visualized and interpreted.

A focus allows to restrict data, manage spatial and temporal scales thereof (multiple representations) and create distinct scenarios, including the vocabulary, constraints, process and rules that should be applied to the dataset [74, 85]. The same data item can be interpreted in distinct ways – a species observation, for example, could represent an organism to be analyzed in a small level of detail or, in a macro perspective, a feature of a biome.

One important problem in focus-related research is how to improve data semantics, increasing its understanding and removing ambiguity. The use of ontologies has been pointed out as a means to deal with some of these issues and used to drive data management. This notion, known as *"ontology-driven information systems"* [46], uses ontologies as a central role with impact on the main components of the system and providing multiple perspectives of the data.

2.3 A Framework to Generate Foci

The goal of our research is to specify and implement a framework to build and explore arbitrary foci. To achieve this purpose, we extend the traditional definition of views to represent a focus, providing a reorganization of the original data or part thereof. The framework uses graph databases as the basis of data management, taking advantage of their ability to deal with highly connected datasets, a common scenario in eScience. Since graph databases do not implement the view concept, the framework introduces extensions to existing systems.

Figure 2.1 gives a general overview of the framework. The interface receives a focus specification as input and provides the focus as output. Both focus and underlying databases are represented as graphs (a focus may be built combining one or more graphs). The focus specification is a text file whose content and format are still under definition, using existing graph query languages (e.g. Cypher, SPARQL [71]) and the parameters of graph algorithms. Following the figure, step (1) decomposes the focus specification to define the focus generation strategies, operators and parameters. Next, the focus is created using either a query view mechanism (2); a central concept view mechanism (3); or a combination of both.

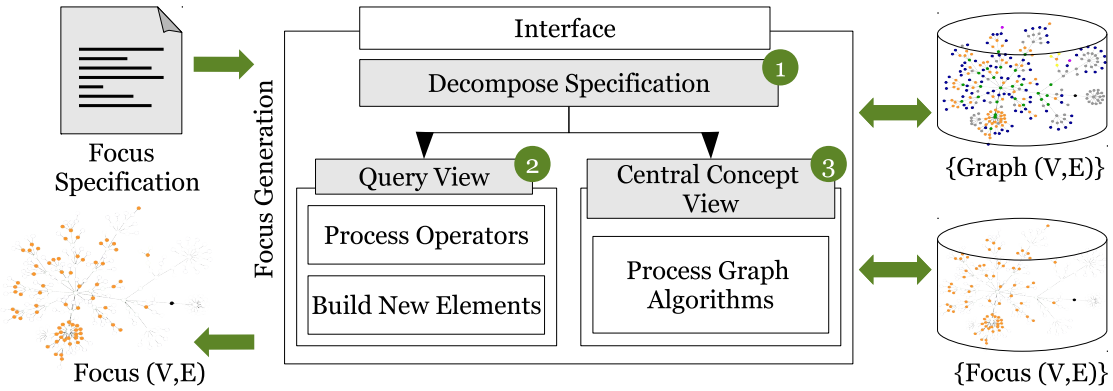


Figure 2.1: Overview of the Focus Generation Process

The query view approach (2) adopts concepts from relational databases. Here we have two tasks: processing the operators that compose the query and creating new elements that do not belong to the original graph. Part of the focus specification is used to create the *"view generator function"*, the sequence of operators to be applied to the database. The traditional operators are adapted by the framework: (i) selection: to filter parts of the graph applying predicates; (ii) projection: to restrict parts of the original graph; (iii) join: to combine two or more graph databases via join conditions; (iv) aggregate functions: to provide graph summarizations, extracting vertex and edge properties.

The central concept view approach (3) is inspired by approaches to construct views on ontologies. Here, just one task is executed: processing of graph algorithms, starting from a central concept, namely a vertex defined in the focus specification. This graph algorithm can provide, for instance, the neighborhood, the shortest path to another vertex, the maximum clique, and so on [16]. The combination of these approaches allows expressiveness higher than graph query languages alone, usually untyped [26], based on triple patterns [71] and without native graph algorithms. Besides that, graph languages have limitations to create temporary elements without altering the original database and the result of a query is not necessarily a graph.

Graph databases and the foci created on the top of them are stored in a persistence layer, so that a focus can be reused. Moreover, since a focus is represented as a sub-graph, it can be used to construct other foci. We also keep the specification that originates a focus for provenance information – e.g., to describe the perspective materialized in the focus and to allow to update a focus when the graph databases used to generate it are updated.

2.4 Running Example

Our running example concerns biodiversity studies of animal species, concentrating on observation metadata. In particular, we deal with observations of animal vocalizations, motivated by the challenges faced by the Fonoteca Neotropical Jacques Vielliard (FNJV) at the University of Campinas (UNICAMP) [1]. FNJV has a large collection of animal sound

[1] http://proj.lis.ic.unicamp.br/fnjv

recordings (about 30 thousand observations), whose metadata is stored in a relational database [29]. Observation metadata include information about the species, the place where the sound was recorded, the recording devices, date and time of the observation, and so on.

Although the metadata is, currently, structured as a relational database, it can be directly converted to a property graph database [71], applying straight formal approaches, e.g. [12, 68]. Each row of each table can be modeled as a vertex, using the column names as attributes, and each foreign key can be modeled as an edge. Altogether, an observation has 54 metadata attributes, which can be combined in different ways to determine the edges of the graph database. Figure 2.2 shows one possible graph database denoted by G_{obs}. In the figure, vertices 1 through 6 represent the taxonomic hierarchy of the observed species, and vertices 8 through 11 characterize an observation, represented by vertex 7.

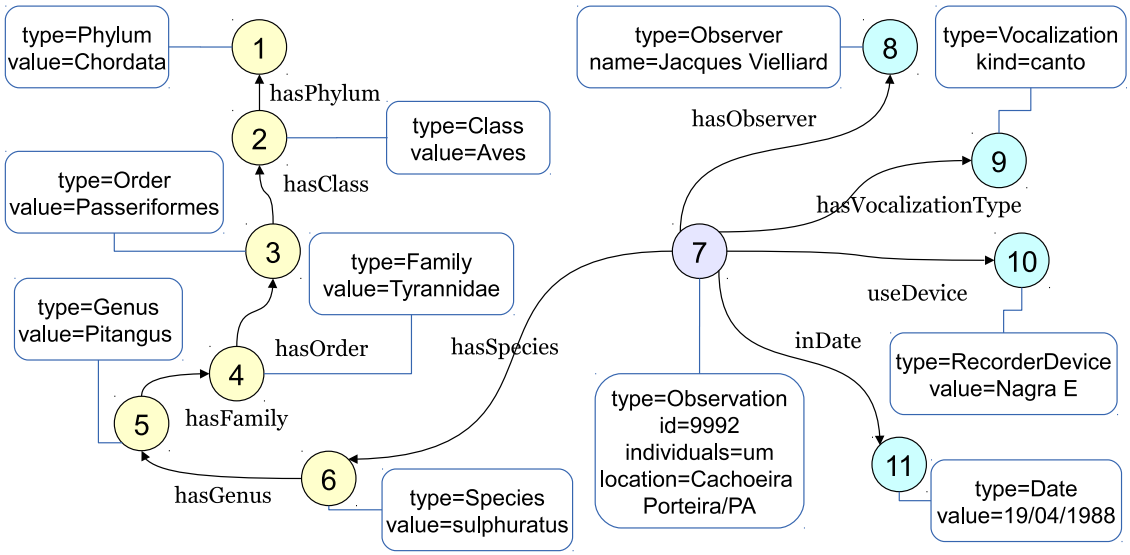


Figure 2.2: Partial Metadata Graph Database of FNJV Observations - G_{obs}

G_{obs} can be integrated with many additional information sources, such as biological and environmental variables to describe the context in which vocalizations were recorded. Distinct pieces of information can be used to produce specific analyses and to build foci. A focus may concern, for example, a geographical scale or a group of species of interest. The following examples describe some use scenarios of foci for this graph database.

2.4.1 Example Focus 1: Location and Biomes

An example of focus which changes the perspective of analysis is defined as:

"Set of all locations in which observations were made, summarizing the number of distinct species observed at each location, and connecting the locations that belong to the same biome".

This kind of focus can be helpful to analyze the biological and environmental characteristics of locations that were targets of study. To process this focus, it is necessary

to aggregate the observation data to generate new information (here, the number of distinct species) and to link the original data with biome information (graph external to our database).

Let us first consider just the first part of the focus: *"Set of all locations in which observations were made, summarizing the number of distinct species observed at each location"*. This kind of focus can also be processed by the query view approach (2) of the framework, combining: (i) "build new element" operator, to create the set of vertices with type **Location** from the attribute *location* of vertices of type **Observation** in G_{bio}; (ii) "aggregate function" operator, to count the number of distinct species observed in each **Location** and store the value in *numberOfSpecies* attribute; (iii) "projection" operator, to filter the vertex and edge types that should be part of the focus (in this case, **Location**).

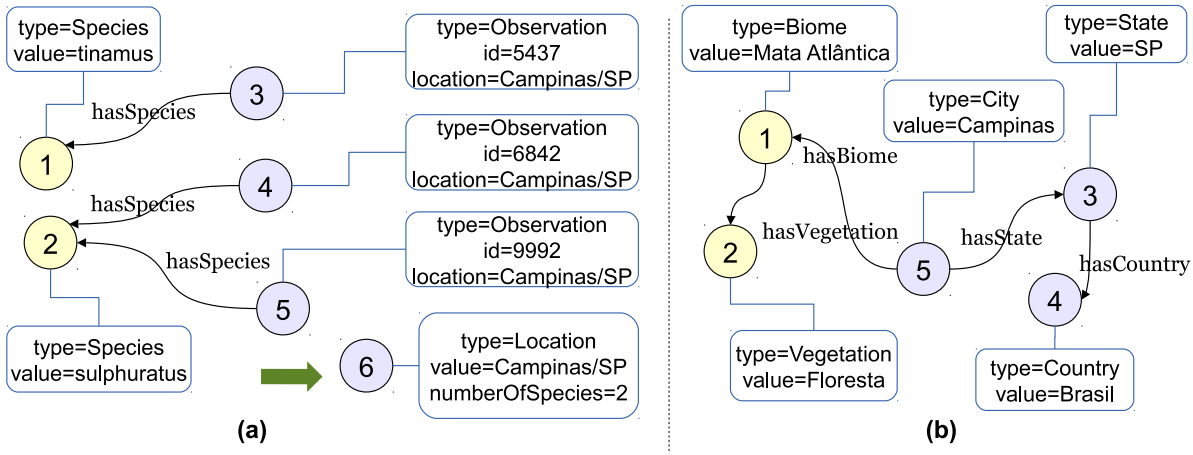


Figure 2.3: Focus: (a) location and number of distinct species and (b) Partial Biome Graph Database - G_{bio}

Figure 2.3 (a) presents a portion of G_{obs} and explains these steps, with the creation of vertex 6 **Campinas SP** of type **Location** and *numberOfSpecies* (here, set to value 2). To connect the locations of the same biome, it is necessary to add biome information not available in G_{obs}. Figure 2.3 (b) shows a partial biome graph database (here shortened to G_{bio}), which is used to integrate this information, using the join operator. In this case, the focus specification combines: (i) "join" operator, to link each vertex with type **Location** in G_{obs} with the corresponding vertex of type **Biome** in G_{bio}, creating an edge (**hasBiome**) between **Location** and **Biome**; (ii) "build new element" operator, to create the set of edges with type **sameBiome** beetween the **Locations** connected to the same **Biome**; (iii) "projection" operator, to filter the vertex and edge types that should be part of the focus (vertices of types **Location** and **Biome**). A partial view of the result focus is shown in Figure 2.4 (a).

2.4.2 Example Focus 2: Species "Closely Related" to Tinamus tao

Another possible scenario builds the focus from a central concept. Here, an example would be:

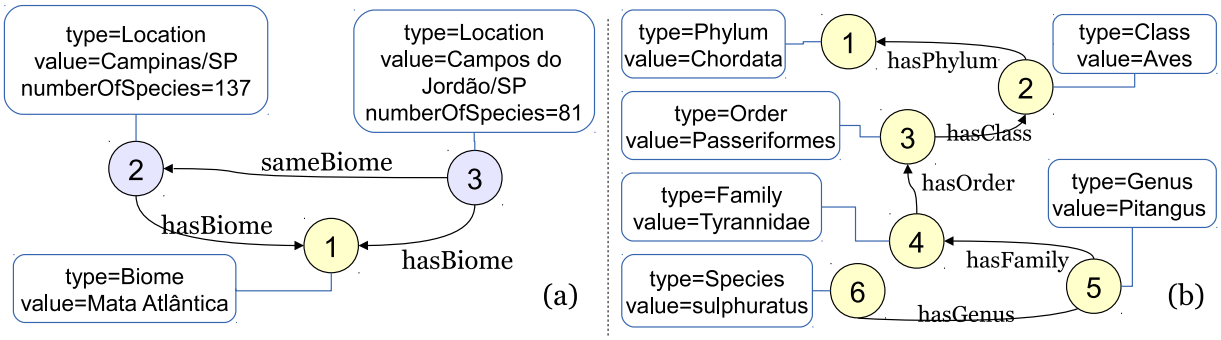


Figure 2.4: (a) Query View Focus: Observation Locations and Biomes (b) Central Concept Focus: species closest to **Tinamus tao**

*"Which are the species closest in the taxonomy to the species **Tinamus tao**".*

This kind of focus can be helpful to analyze the diversity of the species observed according to the "closeness" to other species within a taxonomic level (e.g. genus, family or order). This focus can be processed by the central concept view approach (3) of the framework, starting from species *Tinamus tao* in G_{obs}. The graph for this focus is built considering only edges related with taxonomic classification levels. The notion of closeness here is defined considering the distance between the vertices in G_{obs}: closest mean shortest paths.

The generating function combines: (i) "projection" operator, to filter from G_{obs} the set of vertex and edge types that should be part of the focus (in this case, vertex types related to taxonomic level); (ii) "central concept", in this case, the vertex of type **Species** that represents the species *Tinamus tao*; (iii) the graph algorithm to be applied, in this case, shortest path. The focus result contains all species vertices in the graph for which the paths to species *Tinamus tao* are minimal. A partial result focus is shown in Figure 2.4 (b).

This focus can be further restricted to *"Species closest in taxonomy to **Tinamus tao**, observed in the same locations"*. This can be helpful to understand the similarity among environments where "closely related" species are observed. In this case, specification of focus 2 should be extended, including a "selection" operator to filter only species observed in the same locations. This focus demands a combination of all functionalities available in the focus generation module.

2.5 Conclusions and Ongoing Work

This paper presented the specification of a framework to build and explore arbitrary foci in scientific databases, using graph databases as the basis of data management. The approach extends the traditional definition of views in relational databases to represent a focus, combining graph query languages with graph algorithms to build customized foci. The internals of the framework were explained via examples in biodiversity data management, pointing out some of challenges to be faced. The implementation of the framework will take advantage of previous work of ours in ontology management [31].

The first challenge involves extending the concept of view of relational databases to graph databases. Another challenge is related to the specification of a focus. At the moment, we assume that a focus is specified by indicating a suite of operations to be applied to the underling graph databases. This, however, will need to be improved once we formalize focus construction operators.

Chapter 3

Hydrograph: Exploring Geographic Data In Graph Databases

3.1 Introduction and Motivation

During the last decade, the volumes of data that are being stored have increased massively. This has been called the "industrial revolution of data", and directly affected the world of science. Nowadays, the available data volume easily outpaces the speed with which it can be analyzed and understood [41]. Computer science has thus become a key element in scientific research.

This phenomenon, known as eScience, is characterized by conducting joint research in computer science and other fields to support the whole research cycle, from collection and mining of data to visual representation and data sharing. It encompasses techniques and technologies for data-intensive science, the new paradigm for scientific exploration [51].

Besides the huge volume, the so-called "big data" carries many heterogeneity levels – including provenance, quality, structure and semantics. To try to deal with these requirements, new database models and technologies emerge aiming at scalability, availability and flexibility. The term *NoSQL* was coined to describe a broad class of databases characterized by non-adherence to properties of traditional relational databases [50]. It encompasses different attempts to propose data models to solve a particular data management issue.

Geospatial big data (i.e., big data with a geographic location component) faces even more challenges – it requires specific storage, retrieval, processing and analysis mechanisms [2]. In addition, it demands improved tools to handle knowledge discovery tasks.

The more widely accepted kinds of NoSQL databases include key-value, document, column-family and graph models. Of these, graph databases are the most suitable choice to handle geospatial big data [3]. Indeed, graphs are the only data structure that natively deals with highly connected data, without extra index structures or joins. No index lookups are needed for traversing data, since every node has links to its neighbors. Besides, in GIS, topological relationships play an important role. These relationships can be naturally modeled with graphs, providing flexibility in traversing geospatial data based on diverse aspects.

Geospatial data about water resources fits these graph connectivity criteria – e.g., watersheds or drainage networks. Owing to the shortage of drinking water, reliable information about volume and quality in each watershed is important for management and proper planning of their use. A watershed is usually represented as drainage network, with confluences, start and end points connected by *drainage stretches* (the network edges).

This paper presents an ongoing work that explores geospatial watershed data taking advantage of graph databases. The goal is to show that this scenario provides additional opportunities for knowledge discovery tasks through classical graph algorithms. The Brazilian Watershed database is used as a case study. The mapping between geospatial and graph models is based on the natural network that emerges from the topological relationships among geographic entities.

The rest of this paper is organized as follows. Section 3.2 contains a brief description of the main concepts involved and gives an overview of the Brazilian Watershed relational database. Section 3.3 presents the process of loading watersheds to a graph database and presents results of important and recurrent queries over watersheds. Some research challenges involved are presented in section 3.4. Finally, section 3.5 presents conclusions and ongoing work.

3.2 Research Scenario and Theoretical Foundations

3.2.1 Brazilian Water Resources Database

Brazil is a privileged country in the water-shortage scenario: it holds 12% of the world total and the largest reserve of fresh water on Earth [18]. Its distribution, however, is uneven across the country. Amazonas, for instance, is the state with the largest watershed and one of the less populous in Brazil. Furthermore, some rivers are being contaminated by waste of illegal mining activities (such as mercury), agricultural pesticides, domestic and industrial sewage leak and garbage.

Reliable information about volume and quality in water resources is extremely important to management and proper planning of their use. To this end, the Brazilian Federal Government approved in 1997 the National Water Law [17] aiming to adopt modern principles of management of water resources and created in 2000 the National Water Agency (ANA), legally responsible for accomplishing this goal and ensuring the sustainable use of fresh water.

To organize the required data and support management tasks, ANA adopts the watershed classification proposed by Otto Pfafstetter [69], constructing a database that covers the entire country, named *Brazilian Ottocoded Watershed*. This database represents the hydrography as a drainage network: a set of drainage points and stretches. This network is represented as a binary tree-graph, connected and acyclic, whose edges – the drainage stretches – go from the leaves to the root, i.e., upstream to downstream.

The Brazilian drainage network is composed by 620.280 drainage points (vertices, in graph terms) and 620.279 drainage stretches (edges). Drainage points represent diverse geographic entities:

(i) a watercourse start point, usually a spring or water source;

(ii) a watercourse end point, usually a river mouth;

(iii) a stream mouth point, which flows into the sea; and

(iv) the shoreline start or end point, two reference points in the coast (one of each) that delimit the shoreline line, being the integrating elements of the entire drainage system.

The first three kinds of drainage points can be seen in Figure 3.1. The degree of a drainage point represents its valence, value 1 represents start or end points and value 3 represents confluences.

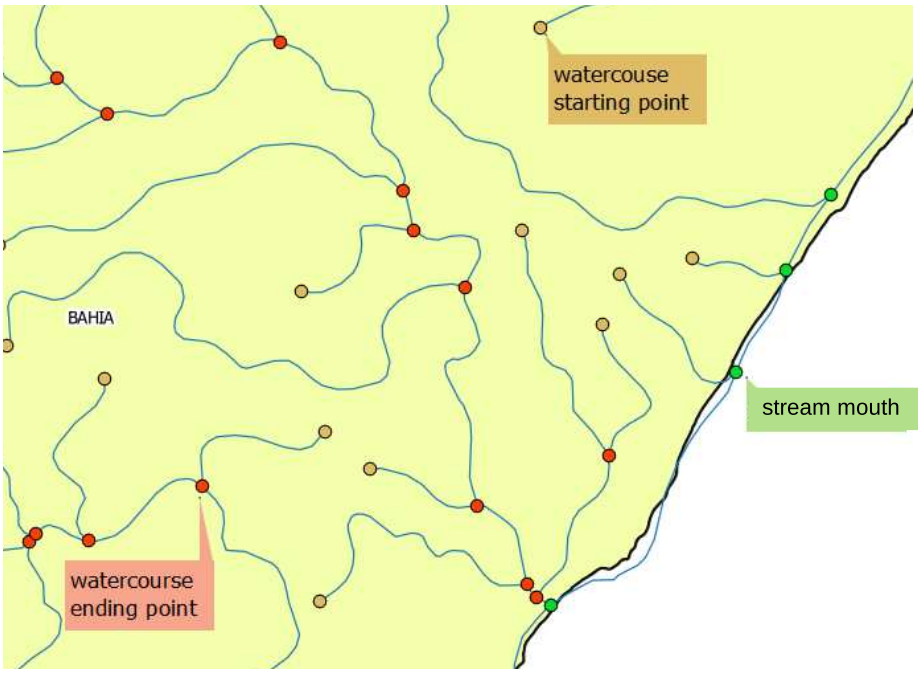


Figure 3.1: Kinds of Points in Drainage Network

The drainage stretches, on the other hand, represent only one geographic entity: the connection between two drainage points. Each stretch has two important attributes: (i) the hydronym, i.e., the name of the water body to which it belongs; and (ii) the hydrographic catchment area, which represents its importance in the drainage network – higher values indicate critical stretches with large areas of water catchment.

Cartographic Aspects

The scale of the Brazilian drainage network varies according to the cartographic mapping used as base in each geographic region, as shown in Figure 3.2. The Brazilian official cartography, projected in the WGS84 Spatial Reference [1] is the start point of the mapping process. The steps of the hydrographic vectorization comprise the representation of each

[1] spatialreference.org/ref/epsg/4326

watercourse as a one-line entity, and identification of their crossing areas as start, end or confluence points. Digital elevation models (such as SRTM - Shuttle Radar Topography Mission [2]) are usually applied in the process of layout refine.

Research on specific watersheds is funded according to their strategic or economic importance, thus generating more detailed data in some regions. Figure 3.2 shows part of the drainage stretches in three scales: 1:1.000.000 (the majority of Brazilian watersheds), 1:250.000 (river Paraiba do Sul) and 1:50.000 (basin of rivers Piracicaba, Capivari and Jundiai) [3]. The latter, for instance, supplies one of Brazil's most populated regions and is the target of several studies, headed by the "PCJ Consortium". This consortium is composed by a group of cities and companies concerned about planning and financial support actions towards the recovery of water sources and raising societal awareness about the importance of watersource issues.

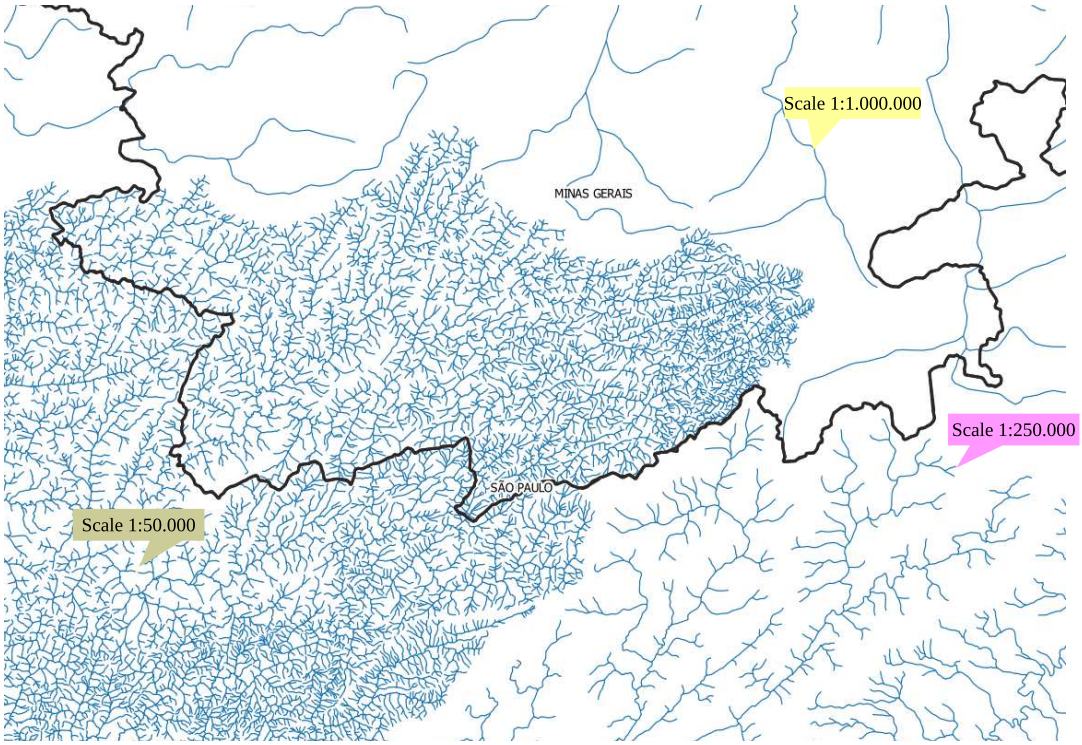


Figure 3.2: Different Drainage Stretch Scales in Drainage Network

The cartographic representation of the drainage network provides an important input to territorial analyses, i.e., when it is necessary to overlay the hydrographic data with other layers (using the geospatial information as the integrating component), in an attempt to understand some spatial phenomenon.

Logical Elements

There are at least four important logical elements in the Brazilian water resources database: hydronyms, hydrographic catchment areas, watersheds and main watercourses. The hy-

[2] www2.jpl.nasa.gov/srtm

[3] Metadata available in: http://metadados.ana.gov.br/geonetwork/srv/pt/main.home?uuid=7bb15389-1016-4d5b-9480-5f1acdadd0f5

dronym is an immutable attribute associated with each drainage stretch that indicates the logical element commonly known as "river". A river is composed by all drainage stretches that are connected and have the same hydronym. Figure 3.3 (a) partially shows the drainage network under this perspective.

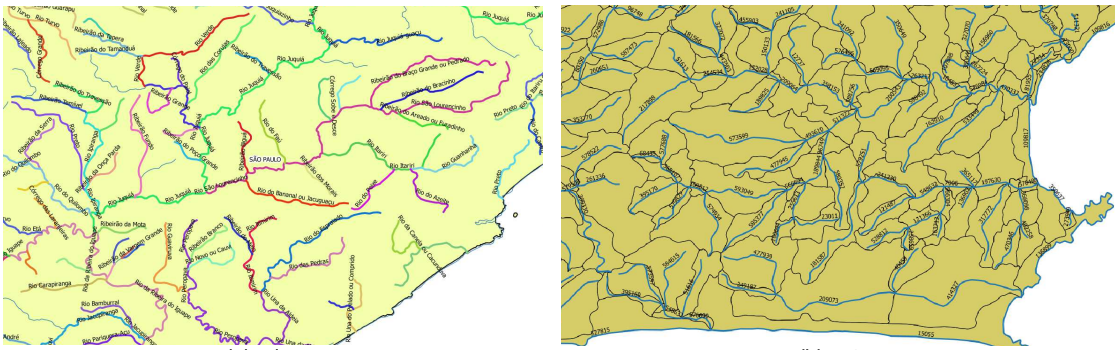


Figure 3.3: (a) Rivers: continuous drainage stretches with the same hydronym and (b) HCA: drainage stretches and their hydrographic catchment area

The other three elements are computed. Every time that the drainage network is updated these elements have to be recalculated. Updates occur for instance during some cartographic refinement process (more accurate scales) or to reflect human actions (e.g., by river transposition or construction of artificial channels). Updates do not occur very often. Thus, if the algorithms that construct the network are well defined, it is possible to materialize network elements, and update them whenever necessary.

The hydrographic catchment area (HCA) is a drainage stretch attribute, represented as a polygon, that delimits the water catchment area of the stretch. This delimitation is highly influenced by relief, given its influence in the water flow. Although HCA is a geospatial attribute, as shown in Figure 3.3 (b), only its area is relevant in most analyzes.

Watersheds and watercourses are two correlated elements – one is used to determine the other in a recursive way. A watershed is the logical element that delimits a drainage system channel. It is the official territorial unit for the management of water resources adopted by ANA. Unlike a basin – that refers only to where the water passes through – a watershed comprises the entire area that separates different water flowing. Every watershed has a main watercourse.

ANA adopts the Otto Pfafstetter Coding System [69](ottocode) to define the watershed division process and watercourse identification. Each digit in the ottocode embeds a context about the stream (the main river or inter-basin, for instance). The main watercourse of a watershed is a set of connected drainage stretches selected by a traversal in the sub drainage network. It is constructed by selecting, in every confluence, the stretch with the largest hydrographic catchment accumulated area upstream (from the mouth to the spring). Following the watercourse layout, the watershed can be split in a set of sub-watersheds and the ottocode allows retrieving their hierarchical relations. A $n-level$ watershed has a code with n digits. Figure 3.4 illustrates one step of this methodology: 3.4 (a) shows the drainage network of the watershed *Rio Trombetas* and its main watercourse, which has the ottocode 454 (level 3). Figure 3.4 (a) shows the 9 new watersheds

created (level 4) by applying recursively the same methodology. The original code 454 is held as prefix to new watershed codes. More details about this methodology can be found in [69].

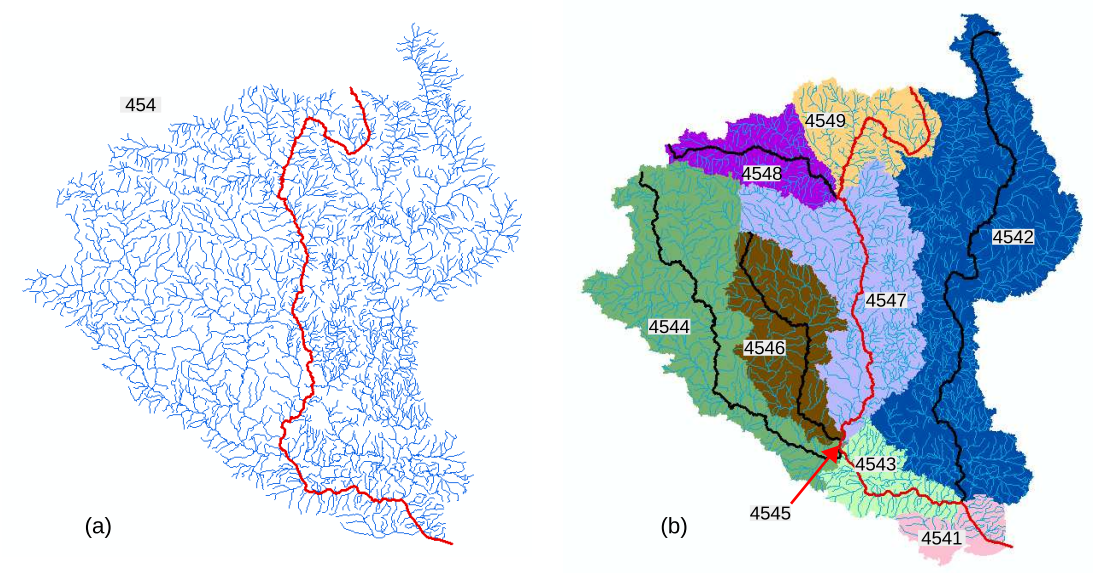


Figure 3.4: Otto Pfafstetter methodology

As can be seen, there are many studies that can take advantage of the network structure of this database and its logical preprocessed elements, even without considering geospatial aspects. Graph algorithms can be used, for instance, to ensure the network consistency or even to determine the main watercourse in a watershed; the latter can be found through a traversal algorithm in a subset of the drainage network, using higher HCA values as the navigation criterion.

3.2.2 Graph Data Management Paradigm

The graph data management paradigm is characterized by using graphs (or their generalizations) as data models and graph-based operations to express data manipulation. It is relationship driven, as opposed to the relational data model which requires the use of foreign keys and joins to infer connections between data items. Graph databases are usually adopted to represent data sets where relations among data and the data itself are at the same importance level [6]. Graph data models appeared in the 90's; nevertheless, only in the past few years they have been applied to information management systems, propelled by the rise of social networks such as Facebook and Twitter.

The formal foundation of all graph data models is based on variations on the mathematical definition of a graph. In its simplest form, a graph G is a data structure composed by a pair (V, E), where V is a finite non empty set of vertices and E is a finite set of edges connecting pairs of vertices. On top of this basic layer, several graph data structures were proposed by the database community, attempting to improve expressiveness, representing data in a better (and less ambiguous) way, such as property graph (or attributed graph) [72, 71], hypernode [57] and RDF graph [13].

Considering the edges, a graph can be directed (i.e., there is a tail and head to each edge); single relational or multi-relational (i.e., multiple relationships can exist between two vertices). The connection structure affects the traversal. An edge can have different meanings, such as attributes, hierarchies or neighborhood relations. Despite their flexibility and efficient management of heavily linked data, there is no consensual data structure and query language for graph databases.

One of the most popular graph structures is the property graph (or attributed graph) [72, 71]. It tries to arrange vertex and edge features in a flexible structure through key-value pairs (e.g., type, label or direction).

3.3 Implementation

3.3.1 Original Relational Database: pgHydro

The pgHydro project [4] – developed by ANA and started in 2012 – aims to implement a spatial relational database to manage the hydrographic objects that compose the Brazilian Water Resources database [79]. It encompasses tables, constraints and views, and a set of stored procedures to ensure data consistency and to process routine calculations. The conceptual model of pgHydro is illustrated in Figure 3.5.

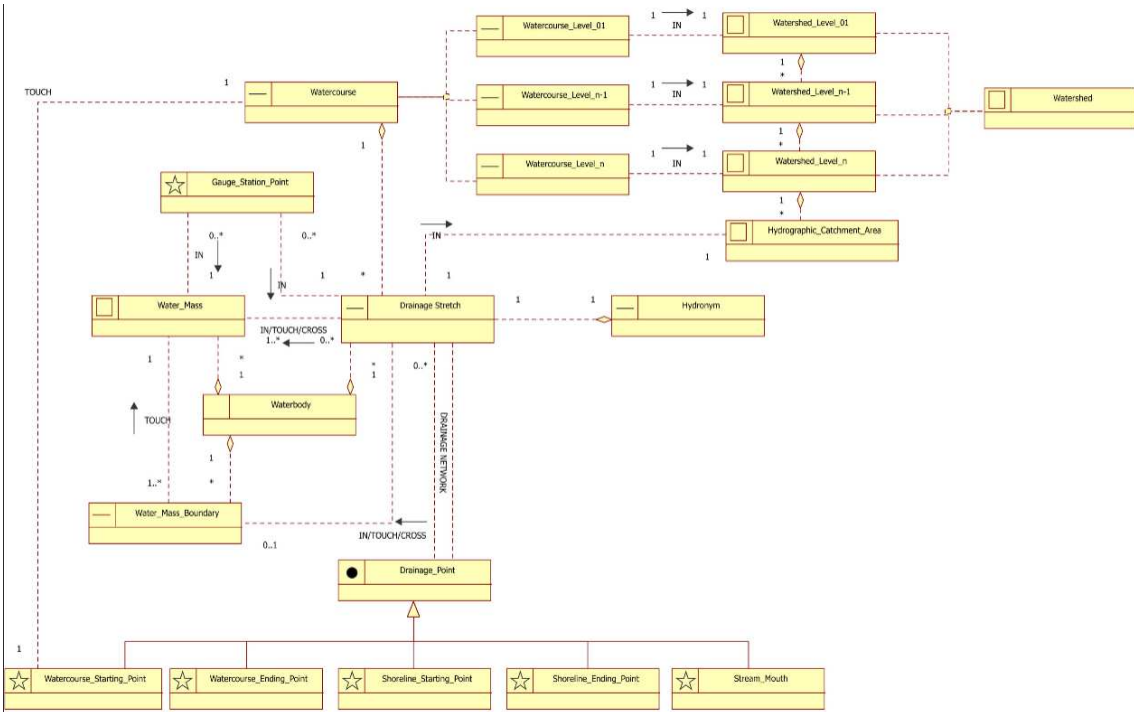


Figure 3.5: PgHydro Database Conceptual Model

PgHydro was implemented in PostGIS/PostgreSQL and a Python interface. PgHydro is a free and open source project and is available for companies and organizations with

[4] pghydro.org

an interest in management and decision making in water resources. More spatial analysis can be done using GIS, such as ArcGIS [5] or QuantumGIS [6].

3.3.2 Proposal Graph Database: HydroGraph

We have transformed ANA relational database (the drainage network) into a graph database, here denoted by G_{Hydro} (partially illustrated in Figure 3.6), keeping the same basic structure of vertices (the drainage points) and edges (the drainage stretches). This data model makes easier to understand the drainage network as it really is: a binary tree-graph, connected and acyclic, whose edges go from the leaves to the root.

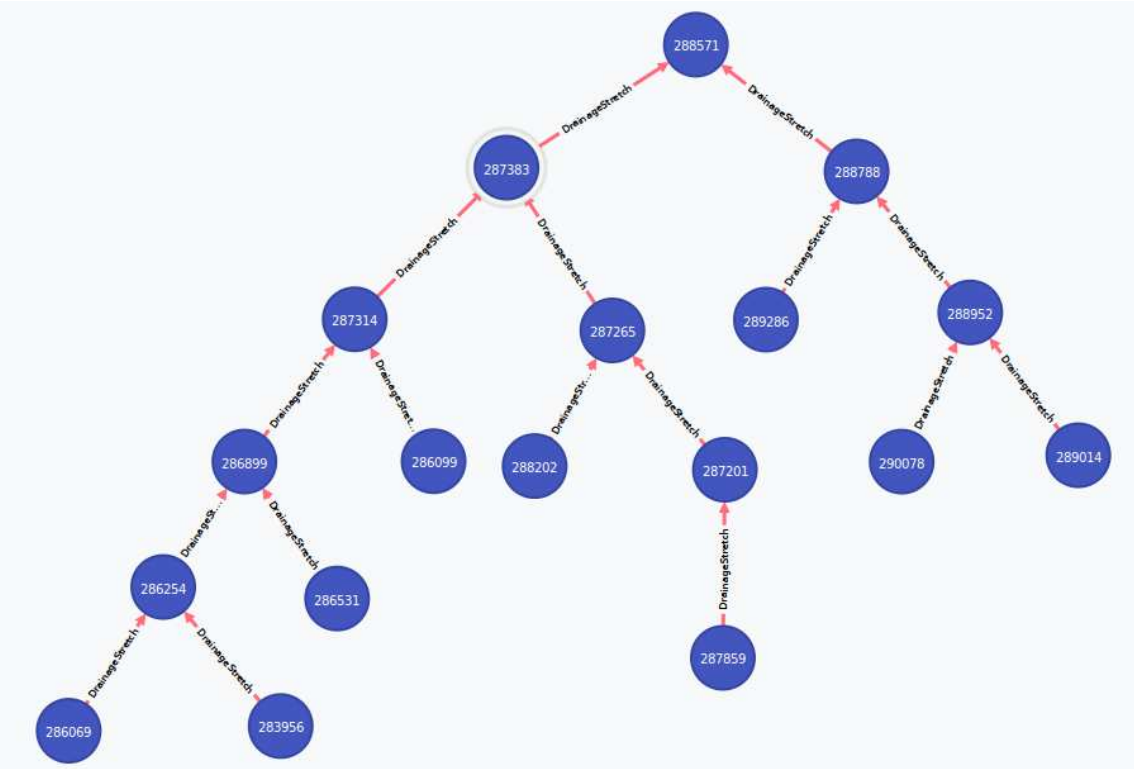


Figure 3.6: G_{Hydro}: Brazilian Drainage Network as a Graph Database

The graph database chosen was Neo4j [7] – a labeled property multigraph [71]. Every edge must have a relationship type, and there is no restrictions about the number of edges between two nodes. Both vertices and edges can have properties (key-value pairs) and index mechanism. Neo4j implements a native disk-based storage manager for graphs, a framework for graph traversals and an object-oriented API for Java. It is an open source project and it is nowadays the most popular graph database [8].

The creation and population of G_{Hydro} were done through **LOAD CSV** command – a load engine provided by Neo4j. The input could be a local or a remote classical CSV

[5] www.arcgis.com
[6] www.qgis.org
[7] neo4j.com
[8] According to DB-Engines Ranking of Graph DBMS (accessed on September, 2015) [db-engines.com/en/ranking/graph+dbms]

file – containing a header and a set of lines in which each line represents a record, and the line is a set of fields separated by comma. The CSV files were extracted from the PostgreSQL database using the **COPY** command [9]. Figure 3.7 shows some of the LOAD CSV commands that giving rise to G_{Hydro} (commands (i) to drainage points and (iii) to drainage stretches). Commands (ii) and (iv) ensure the integrity constraint of unique values for all the identifiers.

(i)
```
LOAD CSV WITH HEADERS FROM "file:<path>/drainage_point.csv" AS line
CREATE (p:DrainagePoint {id:toInt(line.id), valence:toInt(line.valence), geom:line.geom})
```

(ii)
```
$ CREATE CONSTRAINT ON (point:DrainagePoint) ASSERT point.id IS UNIQUE
```

(iii)
```
$ LOAD CSV WITH HEADERS FROM "file:<path>/drainage_stretch.csv" AS line match
(source:DrainagePoint { id: toInt(line.drs_drp_pk_sourcenode)}),
(target:DrainagePoint { id: toInt(line.drs_drp_pk_targetnode)}) CREATE (source)-
[:DrainageStretch { id: toInt(line.drs_pk), upstreamstretch: toInt
(line.drs_drs_pk_upstreamstretch), downstreamstretch: toInt
(line.drs_drs_pk_downstreamstretch), distancetosea: line.drs_nu_distancetosea,
distancetowatercourse: line.drs_nu_distancetowatercourse, waterbodyoriginal:
line.drs_nm_waterbodyoriginal, length: line.drs_gm_length, hca:
line.drs_hca_pk,  upstreamarea: line.drs_nu_upstreamarea, wtc: line.drs_wtc_pk,
hdr: line.drs_hdr_pk, geom: line.st_astext, domain:line.drs_wtc_ds_domain
}]->(target)
```

(iv)
```
$ CREATE CONSTRAINT ON (stretch:DrainageStretch) ASSERT stretch.id IS UNIQUE
```

Figure 3.7: LOAD CSV commands

The **LOAD CSV** command is based on Cypher syntax, the graph query language available on Neo4j [71]. Cypher is a pattern oriented, declarative query language. It has two kinds of query structures: a read and a write query structure. The pattern representation is inspired by traditional graph representation of circles and arrows. Vertex patterns are represented in parenthesis; and edge patterns in brackets between hyphens, one of which with a right angle bracket to indicate the edge direction. For example, the expression **(a)-[r:RELATED]->(b)** is interpreted as two vertex patterns **a** and **b** and one edge pattern **r**, type **RELATED**, that starts on vertex **a** and ends in vertex **b**.

3.3.3 PgHydro Functions

The most important functions of pgHydro are:

1. To validate drainage network consistent;
2. To define the direction of water flow;
3. To apply Otto Pfafstetter's watershed coding system;
4. To select the set of upstream/downstream stretches;
5. To calculate the upstream hydrographic/downstream catchment area.

[9] www.postgresql.org/docs/9.2/static/sql-copy.html

As can be readily seen, most of these functions can be solved applying to graph algorithms on G_{Hydro}. Execute these tasks over relational databases would require many join operations – one of the most computationally expensive processes in SQL databases. Another possibility would be to build an in-memory network representation on top of the relational storage model and to use APIs and programming languages. Graph databases exempt the need of intermediate models from storage to application logic layer.

Consistency tests over the drainage network concern mainly two aspects: connectivity of all stretches and the binary tree structure. In graph terms – considering G_{Hydro} implementation – we can apply the connected component analysis solution. A connected component in a graph G is a subgraph H of G in which, for each pair of vertices u and v, there is a path connecting u and v. If more than one connected component is found in G_{Hydro}, the database is inconsistent. The binary tree structure, on the other hand, is checked selecting all vertices whose degree value are different from 1 (start or end points) or 3 (confluences).

The selection of the upstream stretches can be done applying to Depth-First Search, starting on the stretch of interest and ending on the watershed root. To calculate the upstream hydrographic catchment area, we sum the HCA from each drainage stretch returned in the previous selection. The same approach can be applied to downstream stretches, using the opposite navigation direction and aggregating all subtrees.

The calculation of the Otto Pfafstetter watershed coding is a more complex task, but it is still a graph traversal. The base task is to define the main watercourse. Here, unlike the previous computations we need to establish graph traversal criteria on each node: selecting, at every confluence, the stretch with the largest HCA accumulated upstream.

Among all these functions, only the definition of water flow direction is actually a GIS task and depends on the geospacial information. This calculation involves solving equations that examine the relationship among several variables such as stream length, water depth, resistance of the surface and relief.

3.4 Research Challenges

There are at least three important challenges involved in our approach. The first is related to the incompleteness of graph data models. According to the classical definition, a complete data model should be composed by three main elements: (i) data structure types, (ii) operators to retrieve or derive data and (iii) integrity rules to define consistent the database states [25]. Related work on graph data models shows that they are incomplete concerning on least one of these aspects. Most of them concern only data structures – hypergraphs, RDF or property graphs. Others describe only query languages or APIs to manipulate or retrieve data. There are few attempts to discuss consistency or ACID properties over graph data models. This scenario hampers the formalizing of a complete graph data model. Besides, most implementations of graph databases do not adhere to the theoretical models.

Second, traditional Relational Database Management Systems (RDBMS) are the most mature solution to data persistence and usually the best option when strong consistency

is required. Besides, there are many spatial extensions over RDBMS current used as foundation to geospatial systems and services. Therefore, in some cases there is need for the coexistence of both models – relational and graph – dividing tasks of management and analysis according to their specialties. This requires the development of hybrid architecture to enable the integration of relational and graph databases, as proposed by [22].

Finally, the task of network-driven analysis is not completely solved once the graph database is available. The graph data design (i.e., which data is represented as vertices, which is represented as edges and what kind of properties they have) can streamline or even render non-viably the extraction of topological or graph properties. There is no simple way to crossing through different designs in graph databases. This challenge is also goal of our research, as described in [35]. The idea is to specify and implement an extension of the concept of view (from relational databases) to graph database, thereby allowing managing and analyze a graph database under arbitrary perspectives. Consider this specific database, it would be possible to explore not only the drainage network, but also the network among the logical elements – rivers, watersheds and watercourses.

3.5 Conclusions

This paper presented our ongoing work to construct a graph database infrastructure to support analysis operations on the Brazilian Water Resources database. Our research shows the importance of graph driven analysis over the drainage network, rather than the computationally expensive process of relational databases for such analysis. It presented G_{Hydro} – a version of the original relational database implemented on Neo4j, composed by 620.280 drainage points (vertices) and 620.279 drainage stretches (edges).

Our research takes advantage of graph structures to model and navigate through relationships across the network and its logical elements – watersheds and watercourses. This helps analysts' work in analysis and forecast. However, given the complexity of geospatial data – mainly on big data proportions – there is still no single solution to solve all persistence, management and analysis issues. Hybrid architecture approaches seem to be the most flexible and complete choice.

Chapter 4

Graph-Kaleidoscope: A Framework to Handle Multiple Perspectives in Graph Databases

4.1 Introduction

Increasingly, the world of science is being changed. Data is being produced and collected at an unprecedented scale and outpaces the speed with which it can be analyzed and understood [41]. *Data - intensive science* emerged as a new paradigm for scientific exploration [51, 53]. Computer science has become a key element in scientific research in many areas, such as bioinformatics [28], social network sciences [15] and health [61].

One important requirement of data-intensive science is to support multiple perspectives on large and complex datasets. Many research scenarios require dealing with a subset of data of interest, under multiple aggregation / generalization levels, for given perspectives and a specific vocabulary. This problem is addressed by [73], who define a *focus* as a perspective of study of a given problem, in which data can be restricted to one specific scale or representation, or where objects from distinct scales can be put together.

Environmental research, which relies heavily on geographic data, is a prime example of multi-focus work in which experts from multiple domains must collaborate at distinct spatial and temporal scales. Besides demanding tools to handle knowledge discovery tasks [2], geographic data deal with wide coverage area datasets, in multiple geographic scales and composed by many geographic logical entities that interpret the same data in different ways. Typically, experts need to build "what-if" scenarios, in which a wide variety of factors interact through many kinds of relationships. Though many data management mechanisms have been proposed to deal with such requirements, appropriate handling of perspectives and relationships is still an open problem.

Our research is motivated by this scenario, for environmental research concerned with water resources. This provides a data-rich, problem-rich scenario, in which distinct groups need to investigate problems such as pollution, health or erosion, to name but a few examples of projects that are based on the same set of (water) data, which is analyzed and combined with other specific data sources. These applications have a few requirements in

common. First, they need to provide "multiple perspectives" of the water resource network, for any given project, or even across projects, so that experts can perform distinct analyses. Second, despite the additional semantics provided by geospatial information, a large number of such analyses rely on the underlying network structure, which is subsequently used to create new (logical) geographic entities. Third, water resources are naturally structured and represented through a drainage network, which can be naturally modelled using graphs, thereby supporting many kinds of analyses via graph algorithms – e.g, to find out connectivity between rivers (and thus identify a focus of pollutant flow).

Given these characteristics, we have designed and partially implemented a computing framework that allows environmental applications that involve water resource management to build multiple perspectives, and correlate these perspectives and resources using graphs as the main underlying representation. This framework, called Graph - Kaleidoscope [1], allows users to build multiple perspectives over graph databases. Perspectives are defined through an adaptation of the concept of views in relational databases – i.e., each perspective is handled as a view in the graph database. As such, Graph-Kaleidoscope extends the concept of view from relational databases to graph databases – for which, so far, little has been done in terms of views (either as a needed feature, or definition).

While graph systems are being increasingly adopted in cases where relationships among data elements are first-class citizens, graph database systems are at the same overall level of maturity as object-oriented database systems were in the early to mid-90's. To that effect, we explicitly paraphrased two sentences of the classical "Object-Oriented Manifesto" [8], concerning the state of the art of OO databases in 1989, when that paper first appeared, and which we repeat here.*"Three points characterize the field at this stage: (i) the lack of a common data model, (ii) the lack of formal foundations and (iii) strong experimental activity. Whereas Codd's original paper [25] gave a clear specification of a relational database system (data model and query language), no such specification exists for object-oriented database systems"* (Manifesto page 2). If we now replace the term "object-oriented" by "graph", the entire sentence holds.

The first challenge addressed by our research was to formalize the operators for graph data that underpins our framework. As a result, we define PGDM – a property graph data model – together with its operators. Though geared towards environmental applications, Graph-Kaleidoscope has been specified in a generic way, so that it can be extended and adopted by other kinds of application domains with similar analysis requirements. Our second contribution lies in the definition of the framework itself, based on the use of PGDM, and for which we have developed a first prototype, also described in this paper.

The rest of this paper is organized as follows. Section 4.2 contains a brief description of our motivation scenario. Theoretical foundations are described in section 4.3. Sections 4.4 and 4.5 respectively present the specification of Graph-Kaleidoscope – our property graph data model (PGDM) and the framework architecture. A real case study for analysis of environmental water resource data is presented in section 4.6. Related work is described in section 4.7 and section 4.8 commenting on research challenges and lessons learned. Finally, section 4.9 concludes the paper, presenting ongoing work.

[1] Thus named because a Kaleidoscope allows to merge parts of an underlying structure to provide a distinct perspective of the parts.

4.2 Motivation Scenario - Brazilian Water Resources Database

Reliable information about volume and quality in water resources is extremely important for management and proper planning of their use. To this end, the Brazilian Federal Government created in 2000 the National Water Agency (ANA), legally responsible for accomplishing this goal and ensuring the sustainable use of fresh water. With droughts brought about by global warming, this has become even more critical – even though Brazil is considered to be the country with the largest supply of fresh water resources.

To organize the required data and support management tasks, ANA constructed a relational database to support the effective management of Brazilian water resources, representing the hydrography as a drainage network, i.e., a set of drainage points and stretches. The Brazilian drainage network is currently composed by 620.280 drainage points and 620.279 drainage stretches. Drainage points represent diverse geographic entities, such as a watercourse start/end point or a stream mouth point. The drainage stretches represent the connection between two drainage points.

This relational database, called *pgHydro*[2], encompasses tables, constraints, views and a set of stored procedures to ensure data consistency and to process routine calculations. An excerpt of the conceptual model of *pgHydro* is illustrated in Figure 4.1, extracted from [3]. Attributes were omitted for legibility. The database is public domain and available via ANA official website [4]. Figure 4.2 shows part of the current drainage stretches. It comprises three different cartographic scales: 1:1.000.000 (available for the majority of Brazilian watersheds), 1:250.000 (river Paraiba do Sul) and 1:50.000 (basin of rivers Piracicaba, Capivari and Jundiai).

While the core storage element is the stretch, the official territorial unit for the management of water resources adopted by ANA is a *watershed*. A watershed delimits a drainage system channel and comprises a set of drainage stretches and points. The drainage network can be repeatedly split, giving rise to watershed and sub-watersheds. Another element in this database is *rivers* – connected drainage stretches that have the same waterbody name.

Rivers and watersheds are only examples of the many useful elements that need to be derived from *pgHydro* and which are not supported by the database. These elements are calculated based on drainage network attributes. Network data is constantly updated, e.g., reflecting natural or anthropogenic interventions. This further complicates data analysis, given the mismatch between analysis requirements and the underling data model.

Many routine calculations adopted by ANA to retrieve information (e.g. watersheds, drainage network) depend on "reachability". Such calculations repeatedly check *if* two elements are connected and *how*. However, since the data is stored in a relational database, such calculations are computationally costly, performed using several recursive join operators. These queries are extremely complex, unreadable, and with a high maintenance

[2] pghydro.org

[3] *http://pghydro.org/downloads*

[4] Available in: metadados.ana.gov.br/geonetwork/srv/pt/ main.home?uuid=7bb15389-1016-4d5b-9480-5f1acdadd0f5

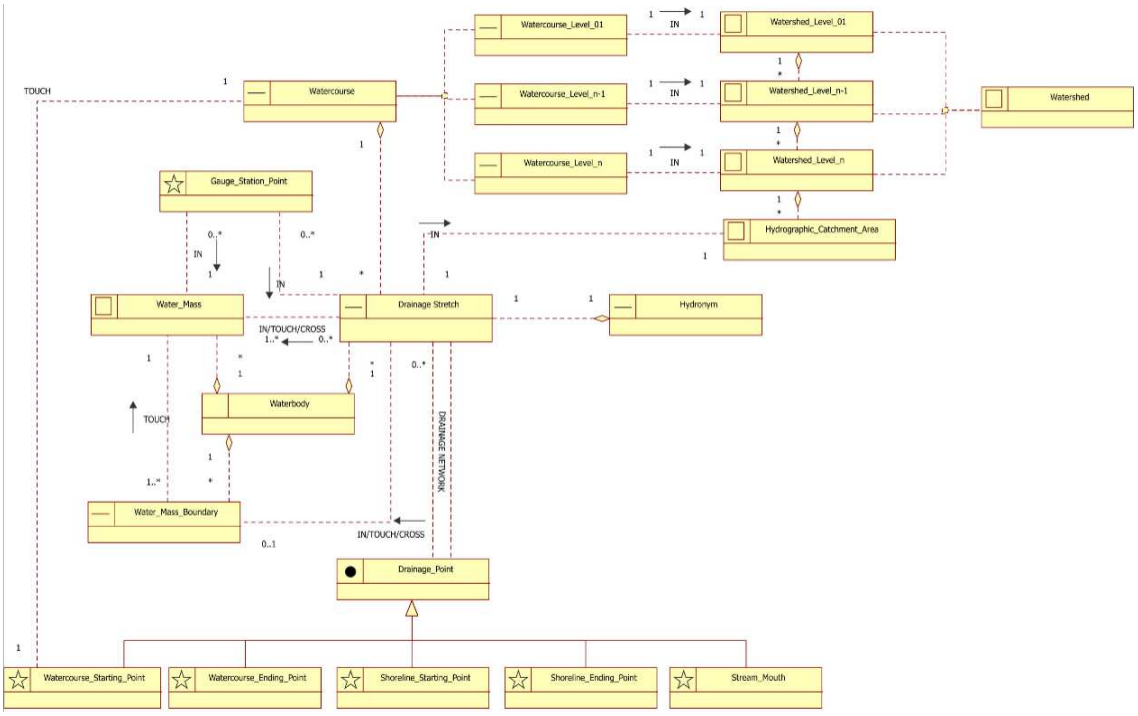


Figure 4.1: *PgHydro* Database Conceptual Model

cost. More complex queries that require graph traversal demand the construction of in-memory graphs on top of the relational database, sometimes reaching hardware limits. In this data scenario, the use of graph databases can bring huge advantages. As will be seen, applying views over graph databases can help end-users construct their multiple perspectives.

4.3 Theoretical Foundations

The goal of Graph-Kaleidoscope is to support domain experts in their needs to build and explore arbitrary perspectives. A perspective is constructed using the notion of database view and data are stored in graph databases. This section concentrates on associated issues, namely graph data management and views.

4.3.1 The Graph Data Management Paradigm

For decades, relational databases and Codd's relational algebra have been the primary storage mechanism and query formalism for datasets. The continuous growth of data and associated heterogeneity led to the emergence of new database models and technologies, aiming at scalability and flexibility. The term *NoSQL* was coined to describe the class of databases that do not have all properties of relational databases and that are generally not queried with SQL [50]. NoSQL data models vary widely. A data model is composed of a data structure, operators to retrieve or derive data and integrity rules to define consistent database states [25].

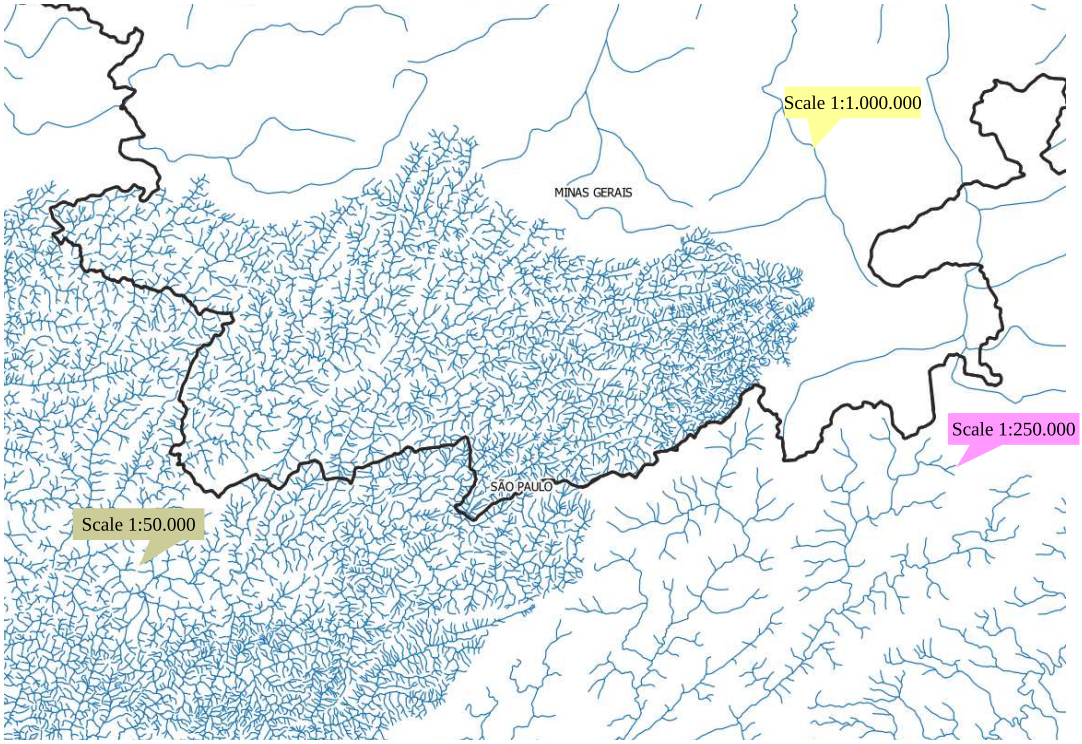


Figure 4.2: Coexisting stretch scales in the drainage network, extracted from [32]

Comparing Relational and Graph Databases

The graph data management paradigm is defined by the use of graphs as data models and the use of graph-based operators to express data manipulation [6]. The graph data model is relationship driven, as opposed to the relational data model that requires the use of foreign keys and joins to infer connections between data items. Graph databases are usually adopted to represent data sets where relations among data and the data itself are at the same importance level [44]. In this model, queries are performed through graph traversals, graph pattern matching or graph algorithms.

Relational database systems rely on one data model, one query language, a single set of basic operators, and one data structure. Though slight variations exist (e.g., object-relational systems), they still preserve the [model, language, operators, structure] principles. There is no such consensus for graph database systems.

The formal foundation of graph data models is based on variations on the mathematical definition of a graph. In its simplest form, a graph G is a data structure composed of a pair (V, E), where V is a finite nonempty set of vertices and E is a finite set of edges connecting pairs of vertices. On top of this basic layer, several graph data structures were proposed by the database community, including features such as directed or undirected edges, labeled or unlabeled edges and vertices and hypernodes. These features attempt to improve expressiveness, representing data in a better (and less ambiguous) way. Each graph data structure has its own data manipulation commands, which can be only invoked at the application level, such as via an API, or at a user-friendly level, such as via a query language. In other words, unlike the relational model, there does not exist a single graph data model.

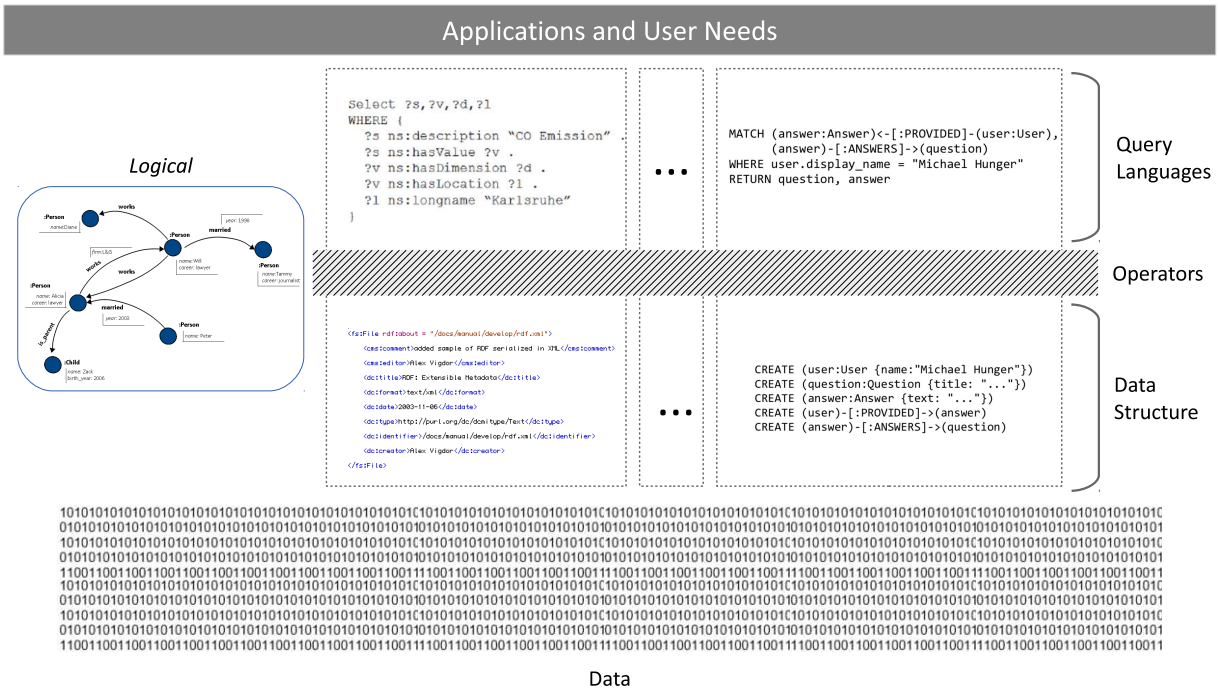


Figure 4.3: Overview of Graph Data Management - a multitude of models, operators and query languages

Figure 4.3 presents an overview of this idea. The left side of the picture shows the user perception of the graph data management paradigm: data in the database are understood to be organized in a graph, in the simplest form $G = (V, E)$. Each application has to build its own "stack of standards" to deal with graph database system issues. Since there is no consensus on a formal graph data model or even for a graph data structure, there are many possible "stacks of standards" available in the literature to meet this architecture.

Each "stack", repeated as a dotted rectangle at the right of the picture, is based on at least one physical data structure and one query language. For instance, the leftmost "stack" shows an example of the SPARQL query language, which is applicable to data structured as RDF.

The gray layer in the middle of the stack of standards remains relatively unexplored in related work: the *operators* layer. The idea behind this layer is to formally define how to manipulate data regardless of implementation details. Graph query languages should be based upon a set of data manipulation operators, just as SQL is based on relational algebra. Once these operators are defined, it becomes possible not only to compare different "stack of standards", but also to swap between "stacks" writing equivalent queries.

The first contribution of this research is to propose this set of operators. To better understand the requirements of the graph data management paradigm, two stacks of standards are presented in the next two sections.

Stack: RDF - SPARQL

RDF (*Resource Description Framework*) is a framework to describe resources based on unique web identifiers URIs (Uniform Resource Identifier [30]). It is the standard model

for data interchange on the web of W3C (World Wide Web Consortium). RDF specification is based on statements about resources following an atomic pattern of triples: subject – predicate – object. The subject denotes the resource, the predicate denotes a property of the resource and the object denotes the value of the property. RDF has several serialization formats, such as N3, XML and JSON.

A collection of RDF statements intrinsically represents a labeled, directed multi-graph. Under this perspective, resources and objects are modeled as vertices and the properties are modeled as edges, used to associate these elements. An example of this approach is shown in Figure 4.4. As can be seen, the attributes of a node are themselves represented as nodes, such as the *name* **Mary** and *city* **Central City** attributes of the node *a1* **person**.

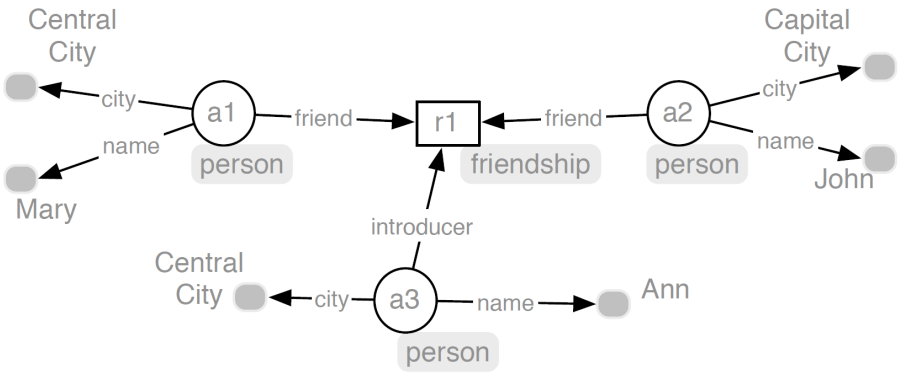


Figure 4.4: An example of the RDF graph extracted from [62]

The standard query language for RDF is SPARQL [48]. It is an SQL-like language whose predicates are defined as triple patterns. A query is processed by selecting which triples satisfy the predicates of the where clause defined in terms of **SELECT** and **WHERE** commands. In addition, SPARQL provides property path patterns, allows to route through a graph between two graph nodes. It is a generalization of a triple pattern to include a property path expression in the property position.

Despite the appealing graph nature of the triple syntax, RDF specification falls short of a definition of a graph in a mathematical sense[5]. There is no clear definition about the set of vertices and the set of edges, as pointing out by [49]. This can be ever more complex since RDF allows swapping between roles in different statements: a given property can be a subject in another statement. For this reason, although RDF/SPARQL is a possible "stack of standards", it is a more complex one. Besides, RDF stores and graph databases are two different categories of DBMS. RDF data are usually stored in document-store database or XML database. There are no RDF store systems that are also native graph databases.

[5]www.w3.org/TR/2014/REC-rdf11-concepts-20140225

Stack: Property Graph - Cypher

Unlike the RDF standard, there are different definitions of a *property graph* [1, 72]. The definition adopted by us characterizes a property graph as a heterogeneous network in which the vertex and edge features are arranged through key-value pairs in a flexible structure [71]. Both vertices and edges can have any number of properties associated with them. This data structure represents a multi-relation graph, since two given vertices can be connected via many edges, each of which represents a distinct relationship between the vertices. This flexibility is one of the advantages of property graphs as opposed to other graph structures (e.g., RDF).

Figure 4.5 shows an example of a property graph. In the figure, the nodes have an attribute to identify the node type (**article**, **person** or **university**) and the edge **attends** has a *start* and an *end* year. This graph data structure is supported by the Neo4j DBMS [83], which, as will be seen, was our implementation choice, due to its widespread adoption.

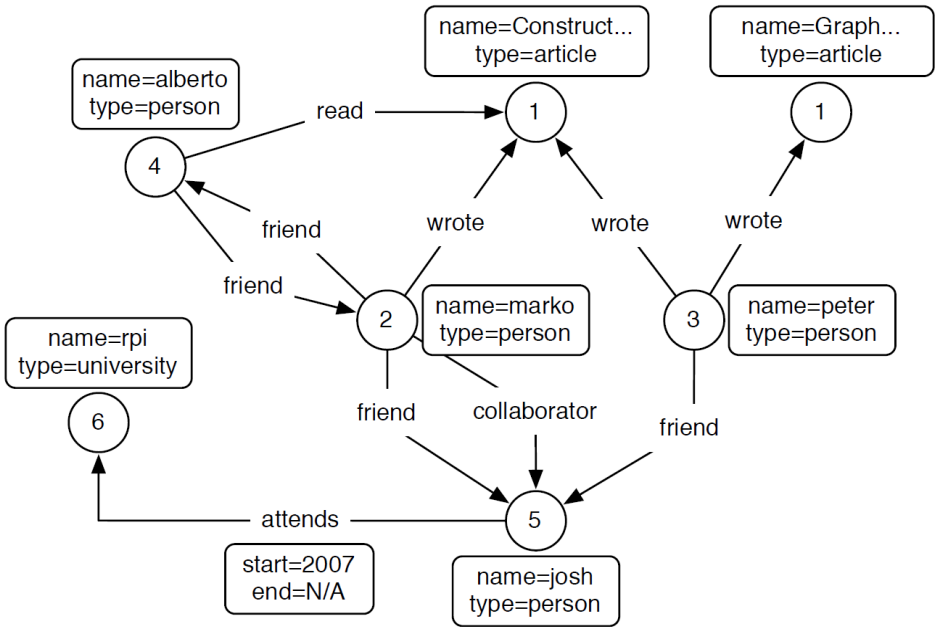


Figure 4.5: An example of a property graph, extracted from [83]

Cypher is the declarative query language defined by Neo4j DBMS for property graphs. Cypher contains a variety of clauses that allow to express traversal queries in a relatively simple way. The **MATCH** clause is used for describing the structure of the pattern searched for, primarily based on relationships. The **WHERE** clause is used to add additional constraints to patterns. The result of the query is defined by the **RETURN** clause.

The Cypher syntax is inspired by the traditional graph representation of circles and arrows. Vertex patterns are represented between parentheses; and edge patterns as brackets between hyphens, where a right angle bracket indicates the edge direction. For example, the expression:

(a)-[r:is_related]→(b)

is interpreted as a pattern that is composed of two vertices **a** and **b**, and an edge **r** of type **is_related**, that starts on vertex **a** and ends in vertex **b**.

4.3.2 Extending Database Views

As emphasized in [73], though a database view was originally defined to be the result of a query, its definition has evolved with time to designate a portion of the data that is of interest to a specific group of users. Under this premise, views are no longer "mere" queries, but an adequate means to support multiple, interdisciplinary perspectives of a given database. Thus, the concept of views adopted in our research extends views in relational databases, where [42] defines a *view* as a temporary relation against which database requests may be issued. Each view may expose the same database in a different way.

For us, a view can be used to achieve different purposes: (i) restriction: to simplify the use of data – to hide unnecessary details or to provide data protection, blocking the access to sensitive data; (ii) scale: to create virtual units derived from aggregations of database objects; or (iii) restructure: to create virtual units derived from rearrangements of database objects. Figure 4.6 presents this idea.

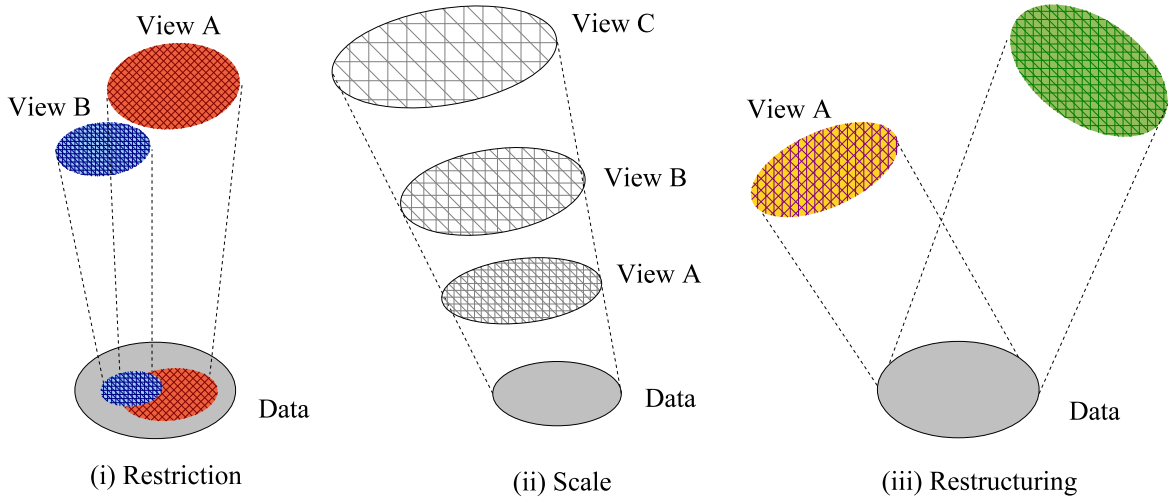


Figure 4.6: Different purposes of Views – adapted from [14] (dimensions of heterogeneity)

Research on relational views ranges from view specification and construction - materialization to mapping updates; views are also involved in query optimization strategies, and in security concerns – e.g., [42, 64, 47, 66]. Relational views are built from queries. Formally, they are specified using language operators (e.g., those of relational algebra). In relational theory, these operators are those used to construct queries, e.g., projection, cartesian product. The sequence of operators that creates a particular view is called *view generating function*, which is a composition of operators applied on the underlying relations.

The concept of view also appears outside the relational database context, using data-specific operators to specify the *view generating function*. In the field of knowledge representation, for instance, an ontology [45] can be too complex for a given application need; hence, an ontology view can be built by the extraction of relevant parts of the original one [31]. In this case, the view generating function can be based on guidelines to navigate through ontology concepts starting from a *central concept* [65] or based on a query language [81]. Although the latter approach is closer to the original definition of views, it faces a huge challenge to ensure that an ontology view is also a valid ontology.

Views and their implementation are consolidated aspects of relational systems. The same does not occur in graph databases, mostly due to the absence of a consensual model and multitude of "stacks" – see section 4.3.1.

4.4 PGDM: The Data Model of the Graph-Kaleidoscope Framework

There is no consensus on a formal definition of operators for graph data or even a graph data structure. Thus, the first challenge addressed by our research is to formalize the graph data model that underpins our framework in terms of Codd's principles [25]. That means that we have to determine a graph data structure, integrity constraints and the elementary operations that can be combined/composed in the view generating function. This resulted in our PGDM Data Model.

In proposing these operators, we do not intend to provide a mathematical formalism for graph data models. The idea here is only to provide the basic constructors, in a non-ambiguous way, capable of explaining operator behavior using a high-level definition. These operators can be implemented in any stack for graph data models, thereby helping interoperability across models.

The second challenge addressed by our research is the definition of the framework itself – even after the graph operators are defined, there is still much to done to deal with views on graphs.

This section deals with the first challenge - the definition of the data model - whereas the next section will describe the architecture of the framework. Our model – PGDM – explicitly defines a data model's elements: data structure, constraints and operators to retrieve data. Our definition of a property graph data model is inspired by the preliminary early definition found in [56]'s thesis, in which the notion of views is briefly explored.

4.4.1 PGDM - Data Structure

PGDM is based on the graph data structure presented in section 4.3.1 – property graphs. For us, property graphs are a flexible data structure approach to accommodate different features of vertices and edges in a generic way, such as label or weight. Besides, they preserve semantics, allowing many attributes in a given entity, which is not possible in RDF model, for example.

Although graph databases have no explicit schema – actually, being schemaless is one

of the strongest points of the graph paradigm – we stress that there is, indeed, an *implicit schema* which receives different names in the literature such as *reference graph* [11] or metamodel [54]. It is important to distinguish between the claims that graph databases have no structure and that they are schemaless. Graph databases are schemaless in the sense that it is not necessary to first create a schema and then insert data – schema and data are inserted together and it is possible to add properties to each individual vertex or edge at any time (whereas in relational databases, which have schemas, one must first create a table, and then insert data).

For us, the *"schema"* of a graph data model is defined by the data design process, which describes the semantic organization of all modeled information – i.e., specifying entities, which relations are valid for each entity, or what kinds of attributes are relevant. Based on this idea, we start the definition of PGDM defining a *graph schema* denoted by G_S. G_S is a logical description of a graph database and consists of a non empty set of vertex types, V_S, and a set of edge types, E_S:

$$G_S = (V_S, E_S)$$

Let V_i be a *vertex type* ($V_i \in V_S$). A vertex type represents an abstraction of some aspect of the real world – a thing with independent existence that can be uniquely identified. A vertex type has *label*, denoted by l, and a set of attributes $\{A_i, \ldots, A_n\}$. Each attribute A_k has a name and a domain for its values. Let a_k denote a value of A_k. The vertex type V_i and its occurrence v_i are expressed as:

$$V_i = (l,\{A_1, \ldots, A_n\}) \text{ and } v_i = (l, \{a_1, \ldots, a_n\})$$

An example of a vertex type *DrainagePoint* and an occurrence is:

V_{DP} = (**DrainagePoint**, {id: integer, type: string})
v_{dp} (**DrainagePoint**, {id: 287383, type: confluence})

Let E_j be an *edge type* ($E_j \in E_S$). An edge type represents how entities, represented as vertices, are related to one another. The initial vertex type is denoted by V_i and the terminal vertex type V_t. An edge type has also a *label*, denoted by l, and a set of attributes $\{B_i, \ldots, B_n\}$. Each attribute B_k has a name and a domain for its values. Let b_k denote a value of B_k. The edge type E_j and it occurrence e_j are expressed as:

$$E_j = (l, V_i, V_t,\{B_1, \ldots, B_n\}) \text{ and}$$
$$e_j = (l, v_i, v_t, \{b_1, \ldots, b_n\})$$

An example of a edge type *is_ connected* and an occurrence is:

E_{IC} = (**is_ connected**, **DrainagePoint**, **DrainagePoint**,
{stretch: integer, length: float, ottocode: integer, waterbody: string})

e_{ic} (**is_ connected**, **DrainagePoint**{id: 287383}, **DrainagePoint**{id: 288571},
{stretch:78991, length:1.05, ottocode:78666279, waterbody: Corrego Vargem Grande})

From now on, we will omit the attribute domain for legibility. A state of a given graph schema G_S, denoted by $\mathbb{G}_S$, is determined by occurrences of vertex types and edge types at a given time, expressed as:

$$\mathbb{G}_S = (\mathbb{V}, \mathbb{E})$$

where $\mathbb{V} = \{v | v \text{ is an occurrence of a vertex type } V_i \in V_S\}$ and $\mathbb{E} = \{e | e \text{ is an occurrence of an edge type } E_j \in E_S\}$. A graph G is expressed as:

$$G = (G_S, \mathbb{G}_S)$$

where G_S is G schema and $\mathbb{G}_S$ is G state. Figure 4.7 shows an example of a graph database we have named G_{Hydro}. It corresponds to the migration of the *pgHydro* relational data (see section 4.2) into a graph database. Details about this transformation process can be found in [32]. G_{Hydro} was created using in the same geographic scale of the original database: the drainage points were modeled as vertices and the drainage stretches as edges. All the original attributes were preserved in the migration process.

Figure 4.7 (a) presents partially the schema G_S of G_{Hydro} composed by one vertex type (**DrainagePoint**, {id, type}) and one edge type (**is_connected**, DrainagePoint, DrainagePoint, {stretch, length, waterbody, ottocode}). Figure 4.7 (b) presents part of the state $\mathbb{G}_S$ with, here, eight instances of the vertex type **DrainagePoint** and seven instances of the edge type **is_connected**.

4.4.2 PGDM - Integrity Constraints

A graph *integrity constraint* defines a restriction on its possible states. The goal of a graph integrity constraint is to ensure data integrity and consistency over a graph's life-cycle. A constraint is checked only when an operation that changes the state $\mathbb{G}_S$ is performed, i.e. creation/update/delete of vertices or edges. PGDM has three types of integrity constraints:

- **Entity Integrity**: adapts the concept of a "primary key", where entities are vertices and edges. Every entities in PGDM – vertex or edge – must have an unambiguous access key. The primary key of a vertex type V_i is denoted by $\underline{V_i}$ and defined as $\underline{V_i} = (l, K_a)$, where l is a vertex label and K_a is a subset of attributes $\{A_1, \ldots, A_n\}$ of the vertex type. The primary key of an edge E_j is denoted by $\underline{E_j}$ and defined as $\underline{E_j} = (l, \underline{V_i}, \underline{V_t}, K_b)$, where l is the edge label, $\underline{V_i}$ is the primary key of the initial vertex, $\underline{V_t}$ is the primary key of the terminal vertex, and K_b is a subset of attributes $\{B_1, \ldots, B_n\}$ of the edge type. An entity key should be unique and not null.

- **Referential Integrity**: adapts the concept of a "foreign key" of relational theory. Every edge in PGDM must have an initial vertex and a terminal vertex. Both vertices are referenced by their primary keys. Once one of the vertices that is part of an edge is deleted, the edge must be deleted too.

- **Domain Integrity**: specifies that all attributes must be declared on a defined domain. A domain is a set of values of the same type.

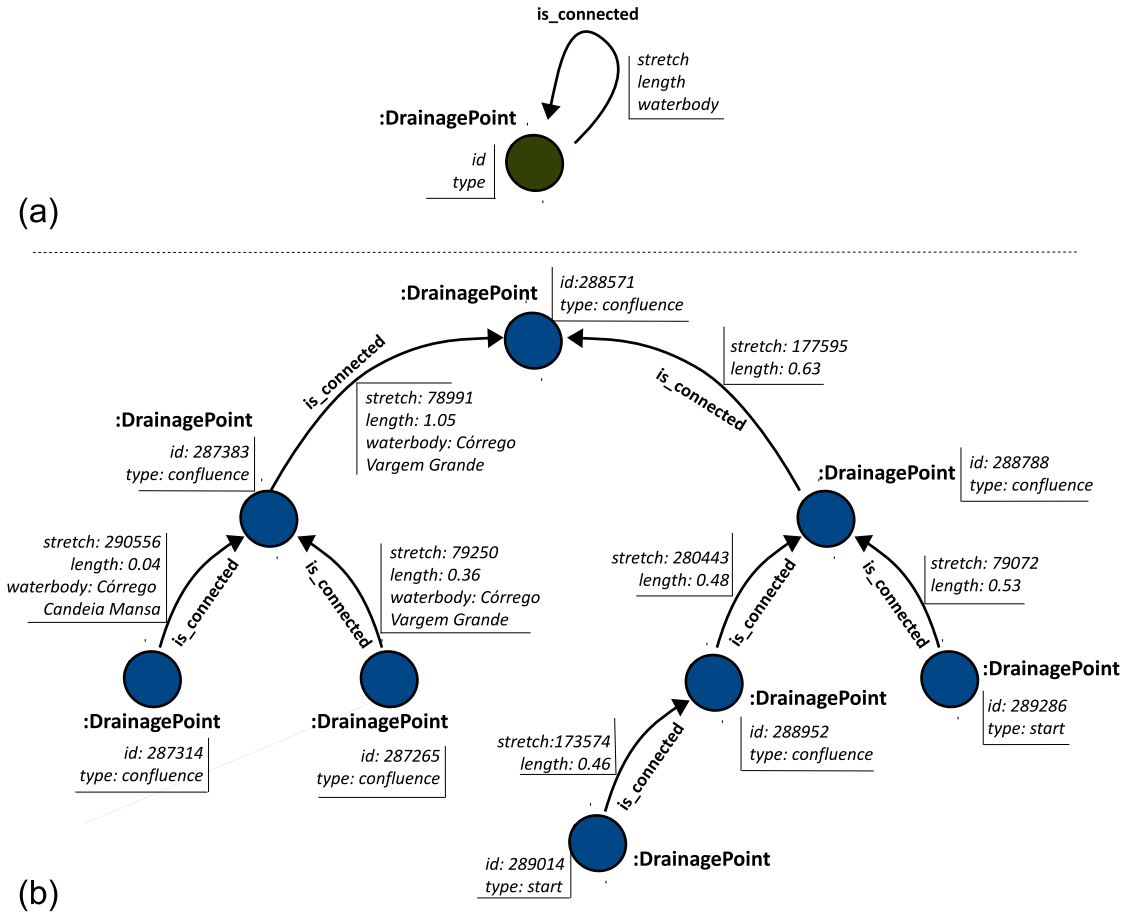


Figure 4.7: G_{Hydro} graph database schema G_S (a) and state $\mathbb{G}_S$ (b)

In G_{Hydro}, shown in figure 4.7, there are two entity integrity constraints:

$$\underline{V_{DP}} = (\textbf{DrainagePoint}, \{\text{id}\})$$
$$\underline{E_{IC}} = (\textbf{is\_connected}, (\text{DrainagePoint}, \{\text{id}\}), (\text{DrainagePoint}, \{\text{id}\}), \{\text{stretch}\}))$$

where $\underline{V_{DP}}$ denotes the access key of vertex type **DrainagePoint** and $K_a = \{\text{id}\}$ and $\underline{E_{IC}}$ denotes the access key of the edge type **is_connected** and $K_b = \{\text{stretch}\}$.

4.4.3 PGDM - Elementary Operators

The manipulation of property graphs is based on elementary operators. To simplify the formalization of operators, we define an intermediate *property graph path*. Property graph paths are an intermediate construct on top of which operators are applied. Next, we define a *predicate* before describing the operators in details.

Property Graph Path

A property graph path GP is a temporary structure created over the graph database to evaluate an operator or to specify the output graph schema of an operator. A GP is

the representation of a linear path connecting vertices and the edges involved in these connections. The GP schema is composed by vertex and edge types of G_S. Analogously, the graph schema of a GP is also composed by vertex and edge types of G_S. In other words, a GP schema is indicating that a graph path cuts across a certain set of vertices and edges, defining the types of interest. A state of a GP, denoted by gp is determined by occurrences of these elements in the graph data.

Two vertices can only be neighbours in a GP in one of two cases:

1. There is a connection between them (an edge). In this case, GP schema is a path in G_S, starting and ending in vertices, defined as: $GP = (V_i, E_{ij}, V_j, E_{jk}, \ldots, V_n)$ where V_i, V_j, V_n are vertex types in V_S and E_{ij}, E_{jk} are edge types in E_S. GP state is the occurrences of this path in the graph $\mathbb{G}_S$. An example of a simple GP in Figure 4.7 is: {**DrainagePoint, is_connected, DrainagePoint**}.

2. They have the same vertex type. In this case, there is no path to consider and GP schema is defined as: $GP = (V_i, V_i)$ where V_i is a vertex type in V_S. GP state is the cartesian product of all occurrences of this vertex type in $\mathbb{G}_S$. An example in Figure 4.7 is: {**DrainagePoint, DrainagePoint**}.

Predicate

Many of our elementary operators apply predicates to retrieve subsets of the original graph data. A predicate is a statement that may be *TRUE* or *FALSE* depending on the values of its variables. In PGDM, a predicate is used to evaluate attributes of the occurrences of vertex/edge types.

Predicate $\varphi(x)$ is written as $\{x|\varphi(x)\}$, defined as collection of all the instances for which $\varphi(x)$ is *TRUE*. A predicate can be atomic or composite. An atomic predicate is defined as:

$$x \ \langle op \rangle \ literal \text{ or } x \ \langle op \rangle \ y$$

where $\langle op \rangle$ is a standard binary operator $=, \neq, <, \leq, >$ and $\geq$. Composite predicates combine atoms by logical operators $\wedge$ (and), $\vee$ (or) and $\neg$ (negation).

Table 4.1 presents an overview of the elementary operators of PGDM. As can be seen, PGDM has seven unary operators and three binary operators. These operators can be progressively composed to create complex operators. Except for rename and conditional traversal operators, all operators require the creation of a property graph path GP to support predicate evaluation and define the graph schema of the result. The table shows the output of each operator in two ways: the output graph schema and the output graph occurrences. The notation of the operators are:

$$\langle \ \textbf{symbol} \ \rangle \ (\langle \ parameters \ \rangle) \ \langle \ \text{input} \ \rangle$$

Notation $G.GP$ indicates a GP created on top of original graph G. To help understanding and clarify the details of the operators, we illustrate them using our mapping G_{Hydro} of the Brazilian water network database.

	Name	Goal	Input	Parameters	GP	Output		Notation
						Schema	Occurrences	
Unary	Restriction (σ)	Select a subset of data that satisfy a given predicate	G	φ: predicate	Yes	GP graph schema	Subset of $\mathbb{G}_S$ in GP whose vertex/edge attributes satisfy the predicate	$\sigma\ (\varphi)$ $G.GP$
	Projection (Π)	Create a new vertex type (l,{A})	G	l: label {A}: set of attributes	Yes	The new vertex type	Subset of vertex/edge attributes of $\mathbb{G}_S$ in GP	$\Pi\ (l,\ \{A\})$ $G.GP$
	Rename (ρ)	Alter an attribute name in a vertex/edge type	G	V_i: vertex type or E_j: edge type $A_i \mid B_i$: attribute name C: new attribute name	No	G_S changed	$\mathbb{G}_S$	ρ (V_i, A_i, C) G ρ (E_j, B_i, C) G
	Edge Creation (ε)	Connect vertices that satisfy a predicate	G	E_j: edge type φ: predicate to connect vertices	Yes	GP graph schema and the new edge type	$\mathbb{G}_S$ plus new edge instances connecting the vertices whose attributes satisfy the predicate	$\varepsilon\ (E_j, \varphi))$ $G.GP$
	Group (γ)	Create a new vertex type by exposing a common attribute value	G	l: label A_i: attribute for grouping	Yes	The new vertex type	Subset of distinct values of a_i in $\mathbb{G}_S$	$\gamma\ (l, A_i)$ $G.GP$
	Attribute Creation (α)	Alter a vertex/edge type by adding a new attribute	G	V_i: vertex type or E_j: edge type C: attribute name f: function or literal	Yes	GP graph schema and new attribute C	$\mathbb{G}_S$ and the new attribute values	α (V_i, C, f) $G.GP$ α (E_j, C, f) $G.GP$
	Conditional Traversal (τ)	Select a subset of data visited in a traversal	G	v_i: initial vertex E_j: edge type φ: predicate sc: stop condition	No	Subset of G_S	Subset of $\mathbb{G}_S$ visited	τ (v_i, E_j, φ, sc)

Table 4.1: Elementary Unary Operators

	Name	Goal	Input	Parameters	GP	Output		Notation
						Schema	Occurrences	
Binary	Union ($\cup$)	Select all data of both inputs	G_1, G_2	None	Yes	G_{S1} plus G_{S2}	$\mathbb{G}_{S1}$ plus $\mathbb{G}_{S2}$	$\bigcup$ $G_1.GP, G_2.GP$
	Difference ($\setminus$)	Select a disjoint subset of data	G_1, G_2	$\{I\}$: set of compatible attributes	Yes	G_{S1}	Subset of $\mathbb{G}_{S1}$ not present in $\mathbb{G}_{S2}$	$\setminus$ ($\{I\}$) $G_1.GP, G_2.GP$
	Intersection ($\cap$)	Select a common subset of data	G_1, G_2	$\{I\}$: set of compatible attributes	Yes	G_{S1}	Subset of $\mathbb{G}_{S1}$ common to $\mathbb{G}_{S2}$	$\bigcap$ ($\{I\}$) $G_1.GP, G_2.GP$

Table 4.2: Elementary Binary Operators

Restriction Operator

Restriction creates a new graph with the schema GP, selecting a subset of data that satisfy a given predicate. The restriction operator is defined as:

$$\sigma\ (\varphi)\ G.GP : \{gp_i : gp_i \in gp \wedge \varphi(gp_i)\}$$

where φ is a predicate for the restriction, gp is the state of GP and gp_i is the i occurrence in gp. To illustrate the operator, consider the example: *Create a view where drainage stretches have length larger than 1 kilometer*. This example is defined as:

$$\sigma\ (is\_connected.length \geqslant 1)\ G.(DrainagePoint, is\_connected, DrainagePoint)$$

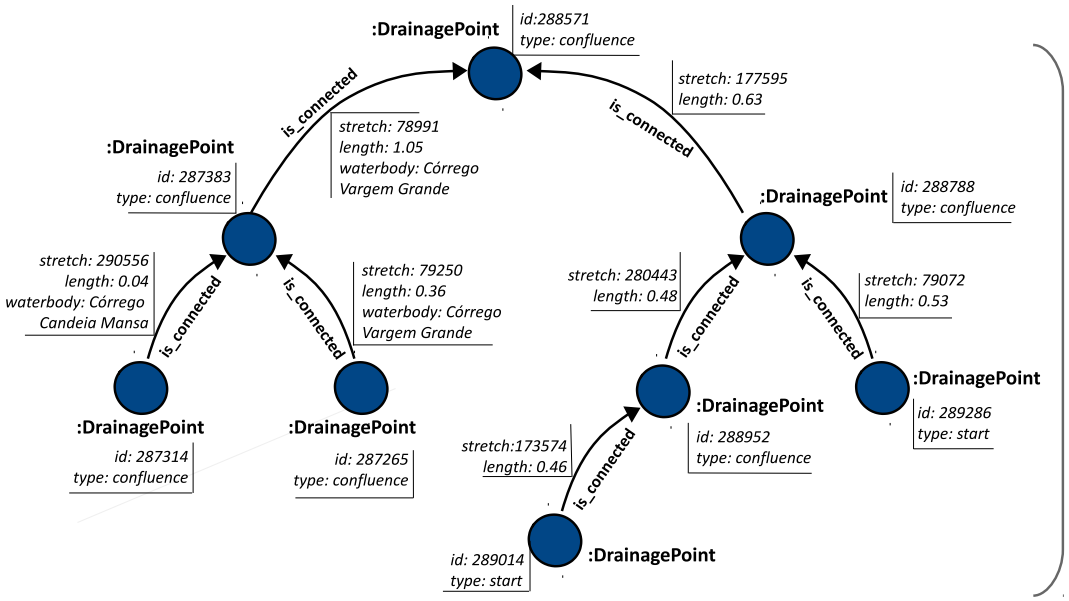
The example is partially shown in Figure 4.8 and has:

- **Input:** G shown in Figure 4.7.
- **GP:** {**DrainagePoint, is_connected, DrainagePoint**}. The predicate $\varphi = is\_connected.length \geqslant 1$ selects only the first line, highlighted in Figure 4.8.
- **Output:** Graph containing schema GP and the subset of occurrences of $\mathbb{G}_S$ that satisfy φ in GP, here composed by (**DrainagePoint**, {id: 287383}), (**DrainagePoint**, {id: 288571}) and (**is_connected**, (**DrainagePoint**, {id: 287383}), (**DrainagePoint**, {id: 288571}), {stretch:78991}) (that has length 1.05 kilometer).

Projection Operator

Projection creates a new graph whose schema contains only a single vertex type with a set of attributes. The projection operator is defined as:

$$\prod\ (l,\ \{A\})\ G.GP\text{: } \{gp_i.A_j : gp_i \in gp \wedge gp_i.A_j \in \{A\}\}$$



:DrainagePoint		is_connected			:DrainagePoint	
id	type	stretch	length	waterbody	id	type
287383	confluence	789191	1.05	Córrego Vargem Grande	288571	confluence
288788	confluence	177595	0.63		288571	confluence
287314	confluence	290556	0.04	Córrego Candeia Mansa	287383	confluence
287265	confluence	79250	0.36	Córrego Vargem Grande	287383	confluence
288952	confluence	280443	0.48		288788	confluence
289286	start	79072	0.53		288788	confluence
289014	start	173574	0.46		288952	confluence

GP

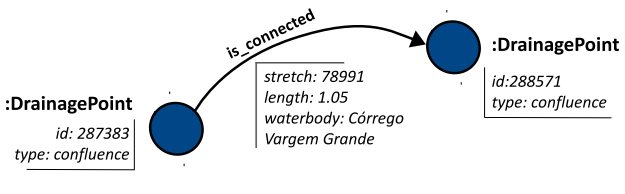


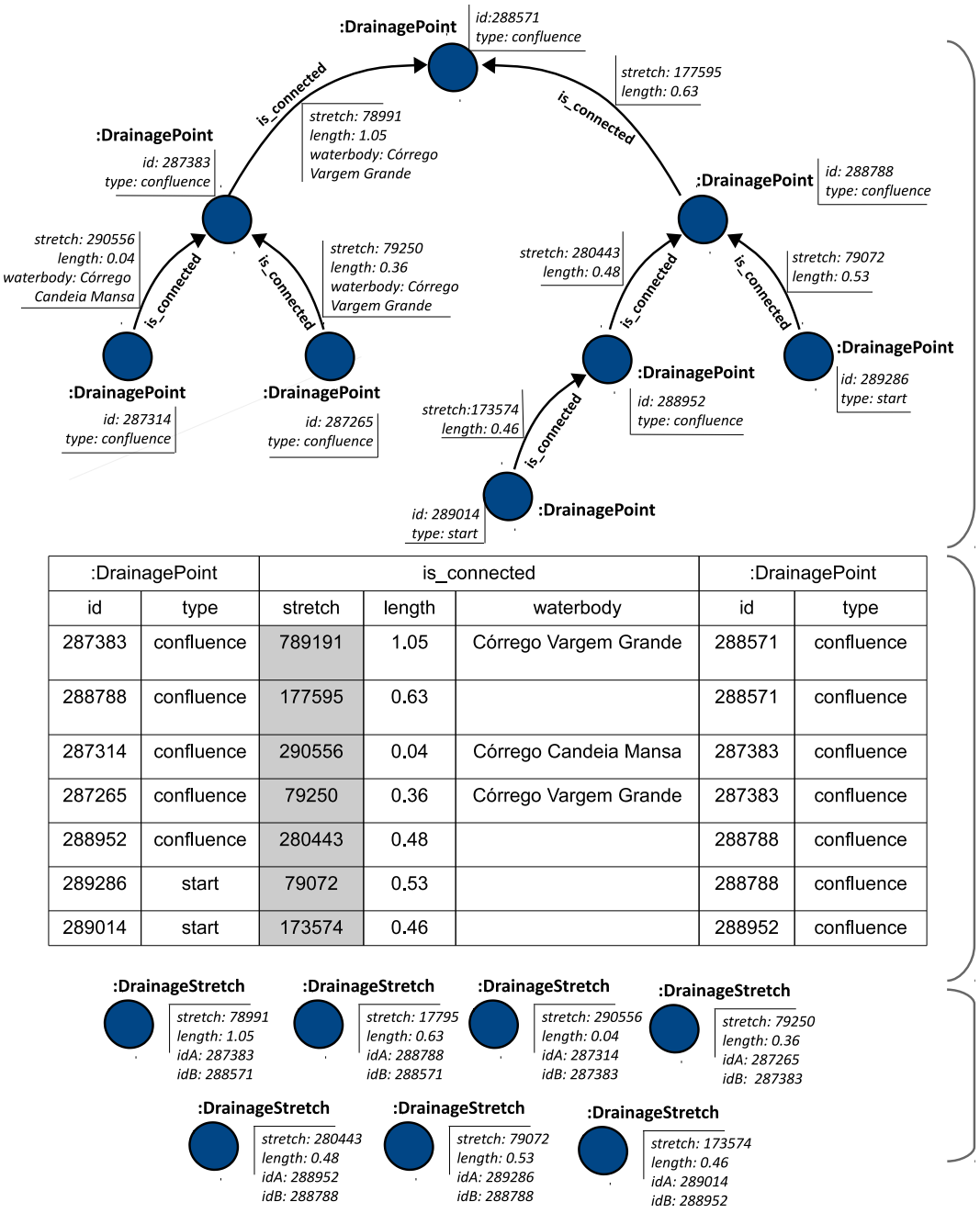
Figure 4.8: Example of the Restriction Operator

where l denotes the label of the new vertex type and $\{A\}$ a set of attributes in GP that compose the new vertex type. To illustrate the operator, consider the example: *Create a view in which vertices represent the drainage stretches.* This example is defined as:

$$\prod (DrainageStretch, \{ic.stretch, ic.length, A.id, B.id\})$$
$$G.(DrainagePoint \text{ as } A, is\_connected \text{ as } ic, DrainagePoint \text{ as } B)$$

The example is partially shown in Figure 4.9 and has:

- **Input:** G shown in Figure 4.7.
- **GP:** {**DrainagePoint, is_connected, DrainagePoint**}. All occurrences of the attribute (**is_connected**.stretch) are selected to compose the output, as highlighted in Figure 4.9.



:DrainagePoint		is_connected			:DrainagePoint	
id	type	stretch	length	waterbody	id	type
287383	confluence	789191	1.05	Córrego Vargem Grande	288571	confluence
288788	confluence	177595	0.63		288571	confluence
287314	confluence	290556	0.04	Córrego Candeia Mansa	287383	confluence
287265	confluence	79250	0.36	Córrego Vargem Grande	287383	confluence
288952	confluence	280443	0.48		288788	confluence
289286	start	79072	0.53		288788	confluence
289014	start	173574	0.46		288952	confluence

GP

Figure 4.9: Example of Projection Operator

- **Output:** Graph whose schema is new vertex type (**DrainageStretch**, {stretch, length, idA, idB}), where *idA* is the initial **DrainagePoint**.id and *idB* is the terminal **DrainagePoint**.id. The output has 7 occurrences as shown in Figure 4.9, e.g., (**DrainageStretch**, {stretch: 79072, length: 0.53, idA: 289286, idB: 288788}).

Rename Operator

Rename creates a new graph where the schema of G is altered by changing the attribute name of a vertex/edge type. The rename operator is defined as:

$$\rho\ (V_i, A_i, C)\ G\text{: } \{V_i.C \mid (V_i.A_i \in V_i)\} \text{ or}$$

$$\rho\ (E_j, B_i, C)\ G\colon \{E_j.C \mid (E_j.B_i \in E_j)\}$$

where V_i denotes the vertex type and A_i is the old attribute name or E_j denotes the edge type and B_i is the old attribute name. C is the new attribute name. To illustrate the operator, consider the example: *Create a view in which attribute* ***type*** *of the* ***DrainagePoint*** *is renamed to* ***status***. This example is defined as:

$$\rho\ (DrainagePoint, type, status)\ G$$

The example id partially shown in Figure 4.10 and has:

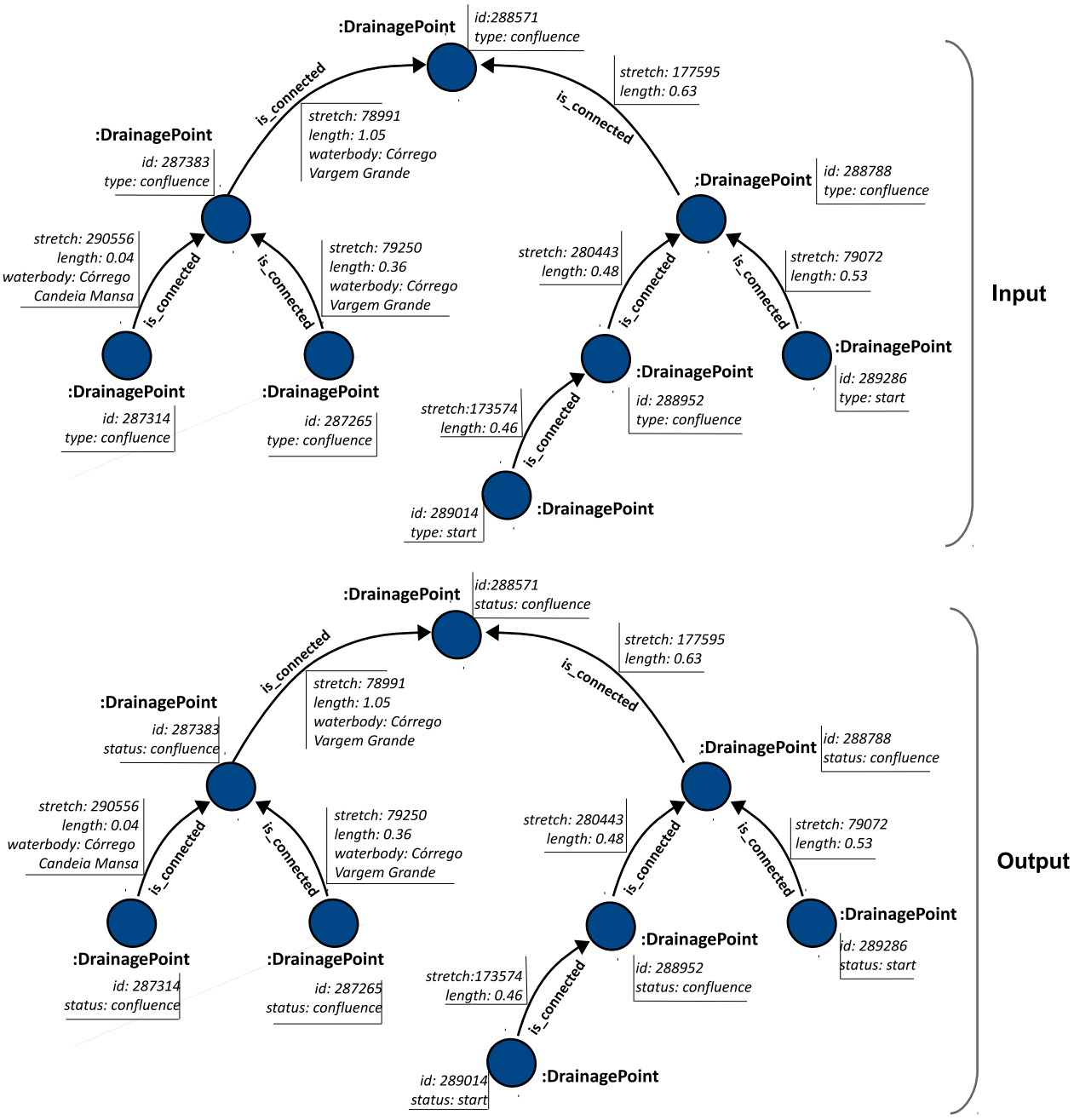


Figure 4.10: Example of Rename Operator

- **Input:** G shown in Figure 4.7.
- **Output:** Graph in which attribute *type* of **DrainagePoint** has been renamed to *status*, e.g., (**DrainagePoint**, {id: 287383, status: confluence}).

Edge Creation Operator

Edge creation creates a new graph whose schema contains the same schema of GP plus a new edge type, by connecting two vertex types that satisfy a predicate. It is defined as:

$$\mathcal{E}\ (E_{it},\ \varphi)\ G.GP\text{: } \{e_{it} : (v_i, v_t \in gp_i) \wedge (gp_i \in gp) \wedge \varphi(gp_i)\}$$

where E_{it} is a edge type and φ is a predicate to connect vertices defined over GP. If condition φ is omitted, each occurrence of the initial vertex type of E_{it} is connected to all occurrences of terminal vertex type of E_{it}.

To illustrate the operator, consider the example: *Create a view connecting the neighbour drainage points that have the same type.* This example is defined as:

$$\mathcal{E}\ (\textbf{has\_sameType}\ (DrainagePoint, DrainagePoint), (DPI.type = DPT.type))$$
$$G.(DrainagePoint\ \text{as}\ DPI, is\_connected, DrainagePoint\ \text{as}\ DPT)$$

The example is partially shown in Figure 4.11 and has:

- **Input:** G shown in the view of Figure 4.7.
- **GP:** {**DrainagePoint, is_connected, DrainagePoint**}. The predicate $(DPI.type = DPT.type)$ selects the lines highlighted in Figure 4.11.
- **Output:** Graph containing the same schema of GP plus new edge type (**has_sameType**, DrainagePoint, DrainagePoint) created and the occurrences that satisfy selected φ in GP, e.g.,(**has_sameType**, (**DrainagePoint**,{id:287314}), (**DrainagePoint**,{id:287383})).

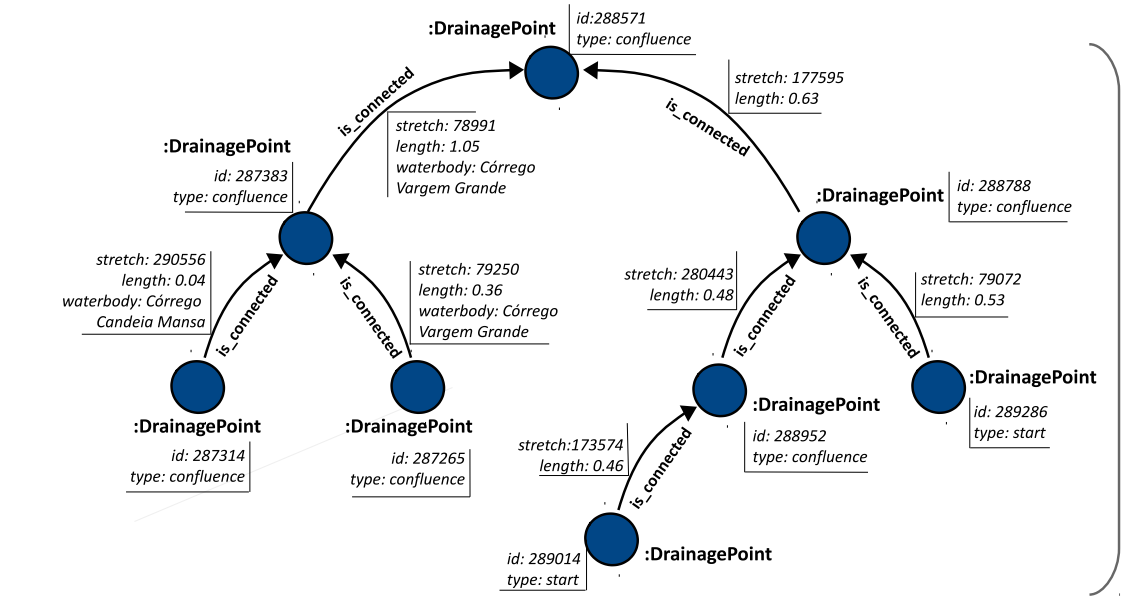
Group Operator

Group creates a new graph whose schema contains only a single vertex type. Unlike the projection operator, group exposes an attribute value common to a subset of occurrences. The new vertex type will have only one attribute. The operator is defined as:

$$\gamma\ (l, A_i)\ G.GP\text{: } \{a_i : a_i \in gp \wedge a_i \cap D = \emptyset\}$$

where l denotes the label of the new vertex type, A_i denotes the attribute of a vertex/edge type in GP to be grouped, a_i is the value of the attribute A_i and D is the set of distinct values of A_i. The attribute name in the new vertex type is also A_i. To illustrate the operator, consider the example: *Create a view in which all waterbodies of stretches appears as vertices.* This example is defined as:

$$\gamma\ (WaterBody, is\_connected.waterbody)$$
$$G.(DrainagePoint, is\_connected, DrainagePoint)$$



:DrainagePoint		is_connected			:DrainagePoint	
id	type	stretch	length	waterbody	id	type
287383	confluence	789191	1.05	Córrego Vargem Grande	288571	confluence
288788	confluence	177595	0.63		288571	confluence
287314	confluence	290556	0.04	Córrego Candeia Mansa	287383	confluence
287265	confluence	79250	0.36	Córrego Vargem Grande	287383	confluence
288952	confluence	280443	0.48		288788	confluence
289286	start	79072	0.53		288788	confluence
289014	start	173574	0.46		288952	confluence

GP

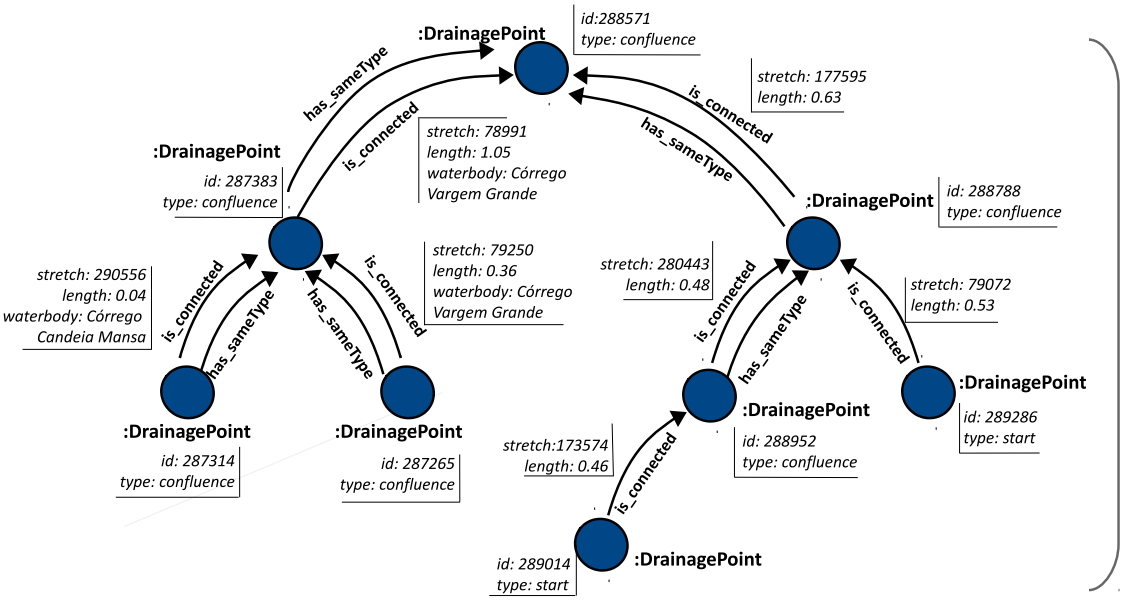
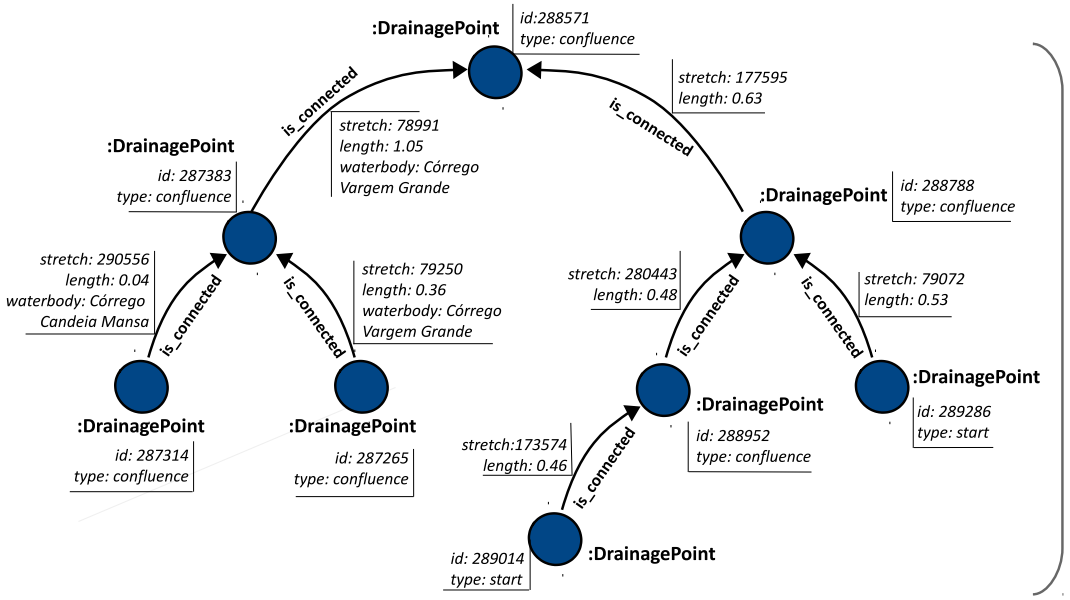


Figure 4.11: Example of Edge Creation Operator

The example is partially shown in Figure 4.12 and has:



:DrainagePoint		is_connected			:DrainagePoint	
id	type	stretch	length	waterbody	id	type
287383	confluence	789191	1.05	Córrego Vargem Grande	288571	confluence
288788	confluence	177595	0.63		288571	confluence
287314	confluence	290556	0.04	Córrego Candeia Mansa	287383	confluence
287265	confluence	79250	0.36	Córrego Vargem Grande	287383	confluence
288952	confluence	280443	0.48		288788	confluence
289286	start	79072	0.53		288788	confluence
289014	start	173574	0.46		288952	confluence

GP

Figure 4.12: Example of Group Operator

- **Input:** G shown in Figure 4.7.
- **GP:** {**DrainagePoint, is_connected, DrainagePoint**}.
- **Output:** Graph containing as schema the newly created vertex type (**WaterBody**, {waterbody}) and composed by 2 occurrences (**WaterBody**, {waterbody: Corrego Vargem Grande}) and (**WaterBody**, {waterbody: Corrego Candeia Mansa}). Notice that even though $\mathbb{G}_S$ has three instances of waterbodies (*Corrego Vargem Grande*, *Corrego Candeia Mansa* and *Corrego Vargem Grande* highlighted), they will be grouped and give rise to only two vertices, as shown in Figure 4.12.

Attribute Creation Operator

Attribute creation creates a new graph whose schema is the schema of GP, altered by changing a vertex/edge type by adding a new attribute. The values of this new attribute are defined via a *literal* (e.g, new static data) or the result of a function on GP (e.g., computed data). Attribute creation operator is defined as:

$$\alpha\ (V_i, C, f)\ G.GP\text{: } \{Vi.C|(gp_i \in gp)\ \wedge(Vi.C = f\ (gp_i) \vee Vi.C = \langle f \rangle)\}$$
$$\alpha\ (E_j, C, f)\ G.GP\text{: } \{E_j.C|(gp_i \in gp)\ \wedge(E_j.C = f\ (gp_i) \vee E_j.C = \langle f \rangle)\}$$

where V_i denotes the altered vertex type or E_j the altered edge type, C the new attribute and f is the value or the function that will assign values to C. To illustrate the operator, consider the example: *Create a view in which each stretch has an attribute containing how many stretches have the same waterbody name.* This example is defined as:

$$\alpha\ (is\_connected, number\_stretches, count(\ equals(is\_connected.waterbody))$$
$$G.(DrainagePoint, is\_connected, DrainagePoint)$$

The function *count* counts the number of occurrences and the function *equal* builds a equality predicate with the parameter. The example is partially shown in Figure 4.13 and has:

- **Input:** G shown in Figure 4.7.
- **GP:** {**DrainagePoint, is_connected, DrainagePoint**}, highlighting the waterbody values to be considered *Corrego Candeia Mansa, Corrego Vargem Grande*
- **Output:** Graph containing the same schema of GP altered by changing the edge type (**is_connected**, **DrainagePoint**, **DrainagePoint**, {stretch, length, waterbody, number_stretches}) and the calculated values.

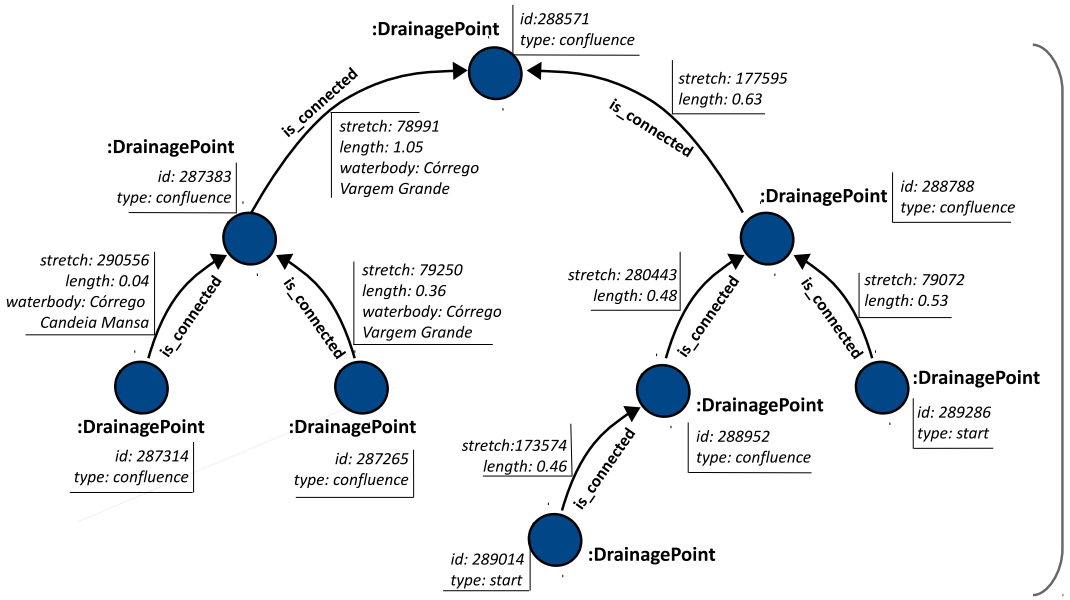
Conditional Traversal Operator

Conditional traversal creates a new graph that contains the selection of a subset of graph data that satisfy given traversal predicates. It differs from the restriction operator in the way in which the predicates and parameters are defined and evaluated. Traversal conditions are defined in terms of reachability – the predicates do not consider which vertex types are involved in the path or in which order. Its output is a new graph containing the subset of the input whose vertices and edges were visited in the traversal. The operator is defined as:

$$\mathcal{T}\ (v_i, E_j, \varphi, sc)$$

where v_i is an initial vertex – an occurrence of a vertex type V_i in $\mathbb{G}_S$, E_j is an edge type, φ is the traversal predicate and sc is the stop condition. A stop condition can be a target vertex v_j or the number of visited vertices n.

Only v_i is mandatory in the conditional traversal operator. The predicate φ is evaluated whenever more than one option is available to continue path traversal. When E_j is



:DrainagePoint		is_connected			:DrainagePoint	
id	type	stretch	length	waterbody	id	type
287383	confluence	789191	1.05	Córrego Vargem Grande	288571	confluence
288788	confluence	177595	0.63		288571	confluence
287314	confluence	290556	0.04	Córrego Candeia Mansa	287383	confluence
287265	confluence	79250	0.36	Córrego Vargem Grande	287383	confluence
288952	confluence	280443	0.48		288788	confluence
289286	start	79072	0.53		288788	confluence
289014	start	173574	0.46		288952	confluence

GP

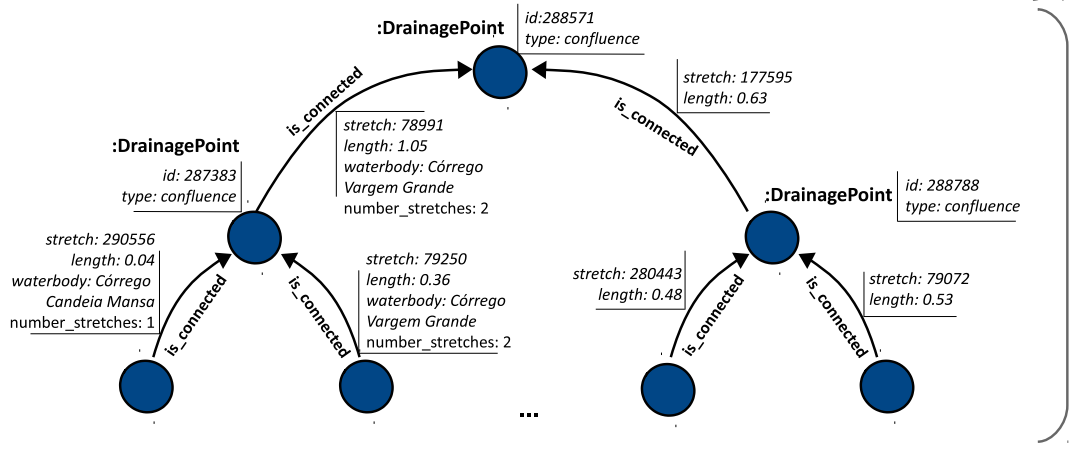


Figure 4.13: Example of Attribute Creation Operator

omitted, all edge types will be considered in the traversal and, in the case of more than one possible path, a random one is chosen. When the stop condition *sc* is omitted, the operator will traverse the graph until the entire input is explored and there are no more possible vertices to visit.

To illustrate the operator, consider the example: *Create a view with all the drainage*

points which are reachable from the drainage point 289014. In case of water contamination, for instance, it is important to know what areas will be affected. This example is defined as:

$$\mathcal{T}\ ((DrainagePoint : \{id = 289014\}, is\_connected)$$

The traversal has no predicate or stop condition. The example is partially shown in Figure 4.14 and has:

- **Input:** G shown in Figure 4.7.
- **Output:** New graph with the same schema of G.

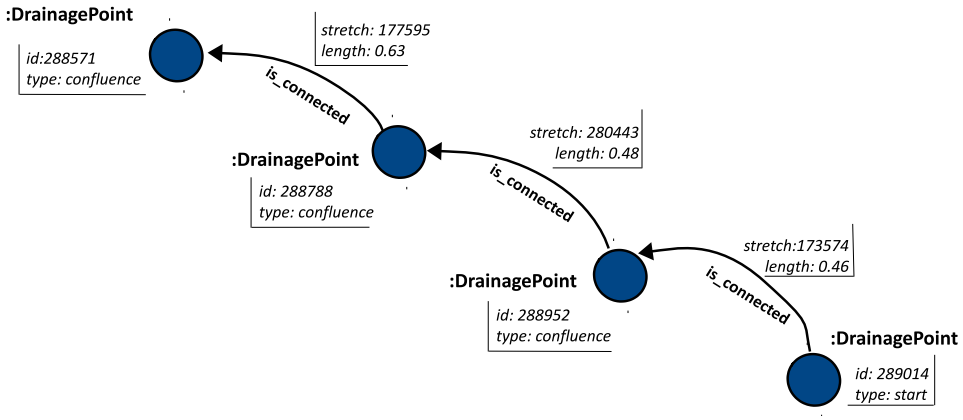


Figure 4.14: Example of Conditional Traversal Operator

Set Operators

Set operators correspond to a category of operators from set theory: union ($\cup$), difference ($\setminus$), intersection ($\cap$). In our model, these operators follow the conventional definition; therefore, we do not discuss them in detail.

An important issue is the idea of *compatibility*, i.e., these operators can only be applied to compatible graph schemas G_{S1} and G_{S2}. Our set operators are defined in terms of property graph paths $G_{S1}.GP$ and $G_{S2}.GP$. A set of attributes $\{I\}$ is defined as a parameter; these attributes are used to verify the compatibility between the graph schemas. For obvious reasons, the number of attributes and their domains should be also compatible.

Intersection

Intersection selects a subset of data common to both inputs. It can only be performed in compatible graphs, and defined as:

$$\bigcap\ (G_1, G_2, \{I\})\ G_1.GP, G_2.GP\text{: } \{gp_i : gp_i.\{I\} \subset G_1.GP \wedge\ gp_i.\{I\} \subset G_2.GP\}$$

where $\{I\}$ is set of attributes used to assess compatibility. The output is a graph with the schema G_{S1} and all occurrences of $\mathbb{G}_{S1}$ whose values of the attributes in the set $\{I\}$

was also present in $\mathbb{G}_{S2}$.

Difference

Difference selects a subset of data from the first input not present in the second. It can only be performed in compatible graphs, and is defined as:

$$\setminus (G_1, G_2, \{I\})\ G_1.GP, G_2.GP\colon \{gp_i : gp_i.\{I\} \subset G_1.GP \wedge\ gp_i.\{I\} \not\subset G_2.GP\}$$

where $\{I\}$ is set of attributes used to assess compatibility. The output is a graph with schema G_{S1} and all occurrences of $\mathbb{G}_{S1}$ whose values of the attributes in the set $\{I\}$ was not found in $\mathbb{G}_{S2}$.

Union

Union selects all data in G_{S1} and G_{S2}. Unlike the other set operators, no compatibility is required in this case – the operator does not compare attributes between source graphs. The union operator is defined as:

$$\cup\ (G_1, G_2)\ G_1.GP, G_2.GP\colon \{x : x \in G_1 \vee x \in G_2\}$$

and the output is a disconnected graph whose schema is the union of both schemas of G_{S1} and G_{S2} and all occurrences of $\mathbb{G}_{S1}$ and $\mathbb{G}_{S2}$.

The standard (database-oriented) union operator requires that inputs have to be compatible. This can be achieved by combining our intersection and difference operators, namely:

$$\cup\ (G_1, G_2)\colon \{\cap\ (G_1, G_2, \{I\})\ \} \cup \{\ \setminus\ (G_2, G_1, \{I\})\ \} \cup \{\ \setminus\ (G_1, G_2, \{I\})\ \}$$

where the result of the intersection of G_1 and G_2, the result of the difference of G_1 and G_2 and the result of the difference of G_2 and G_3 are combined using our union operator.

Revisiting our view construction requirements, Figure 4.6, we point out that related work (e.g., [38, 11, 61]) only proposes "Restriction" operators. On the other hand, PGDM contemplates the three kinds of requirements, as show in Figure 4.15. The Figure shows which operators can be used for each demand.

4.5 Graph-Kaleidoscope Framework - Architecture and Prototype

This section presents the architecture of Graph-Kaleidoscope and its first prototype. As mentioned before, the idea is to adapt and extend the concept of views in relational databases to graph databases. To this effect, we define a *graph view* to be a temporary (non-materialized) perspective defined and extracted from a graph database. Our graph view definition embeds a recursive idea: graph views can be built on the top of other graph views.

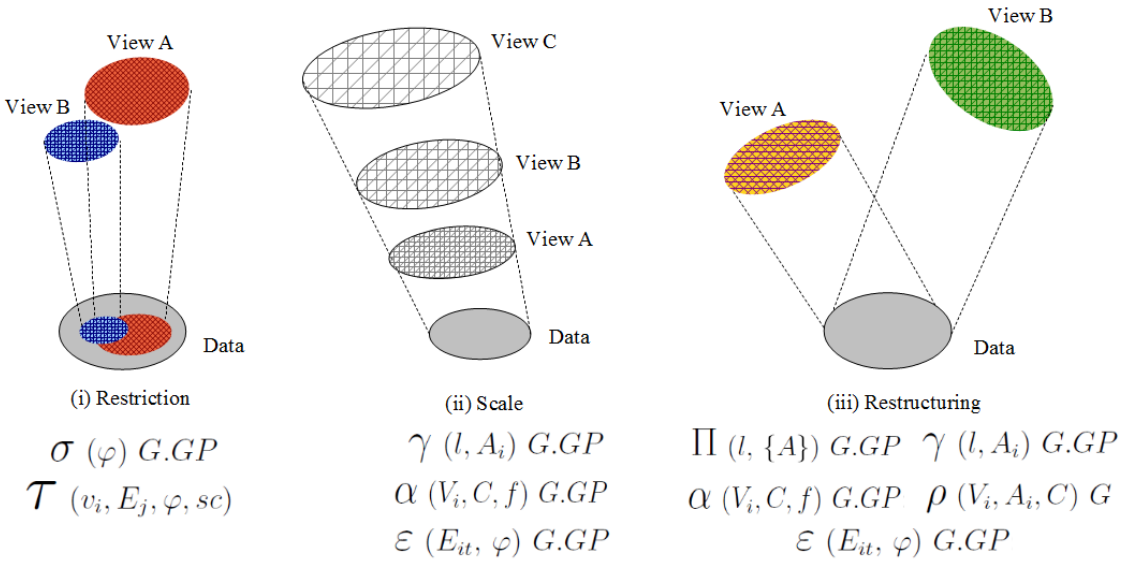


Figure 4.15: View Requirements and Operators

A core idea is to separate view definition (namely, the specification of the view generating function) from actual view creation (namely, the execution of the function). Figure 4.16 gives an outline of this concept, in which these two main functionalities are respectively called "View Builder" and "View Factory". The interface, at the top of this figure, receives requests from users. We point out that views are not materialized, only their definition.

Users can either *define a view* (i.e., specifying the view generating function) or *construct a view* (computing the view generating function). The persistence layer of the framework is composed of a graph database management system; it provides management mechanisms to store graph data and view generating functions. Views can eventually be materialized, thereby becoming part of the graph database.

The **View Builder** receives a view generating function defined as a combination of the graph operators described in Section 4.4.3, and forwards this function to be stored in the view catalog of the graph DBMS. Storing this definition ensures that it can be invoked in the construction of other views. The View Builder delegates to the **View Factory** all necessary steps to compute the graph view. This is presented in the architecture by the black engine connected to the **View Factory**.

The heart of the architecture is the **View Factory**. It is responsible for computing a graph view, i.e, it transforms a view generating function into a graph. It is composed by four modules, executed in the order (i) to (iv):

(i) **Parse**: parses the view generating function and builds an intermediary data structure composed by the required operators, their parameters and the property graph path necessary to process each operator.

(ii) **Process**: accesses the underlying graph database to process the operators. Each operator request is translated to a graph query in the underlying DBMS query language. The results are used to build the elements of the view defined by the operator.

(iii) **Check**: verifies if the resulting view is a valid graph, checking the integrity rules

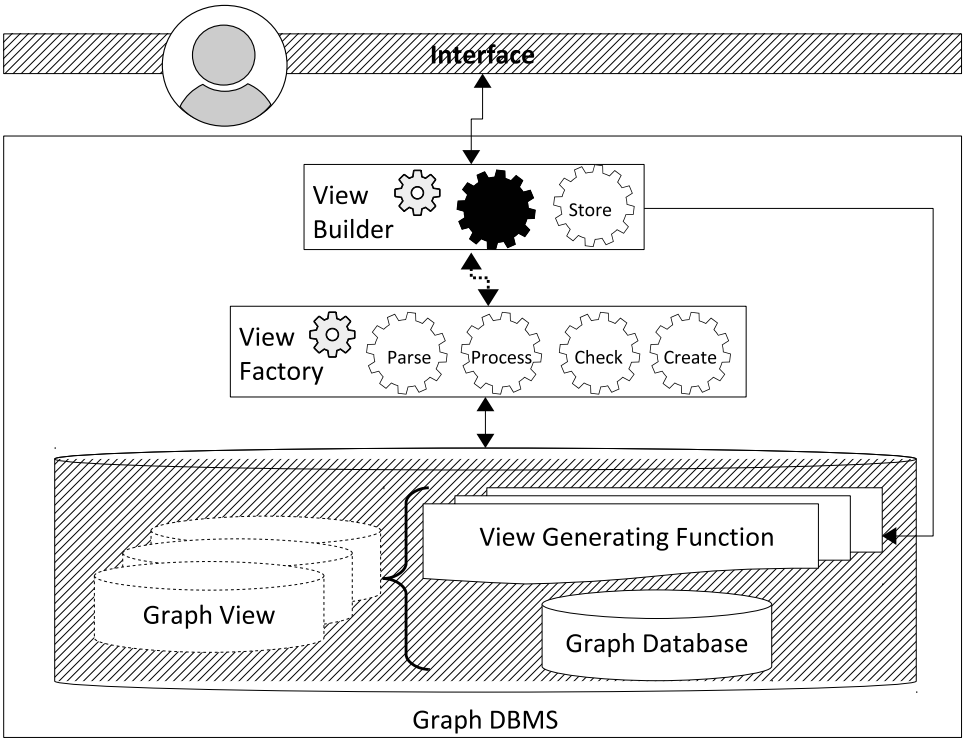


Figure 4.16: Graph-Kaleidoscope Architecture

of the graph resulting from executing the function.

(iv) **Create**: stores this computed graph as a temporary graph.

Our framework was designed to deal with data that, besides being heterogeneous and massive, requires ACID-compliant transactional properties. Its architecture is generic to accommodate several "stack of standards" of graph data management, as long as the operators are implemented on the DBMS structure [5]. The only specific part of our architecture is the module **Process** of **View Factory**. The implementation of this module requires a middleware to map each operator and its parameters into a valid command to manipulate the graph data.

Site DB-Engine tabulates over 300 DBMS, 9 of which are open source and use graphs as data model and thus natural candidates for implementing our framework [6], e.g., Neo4j, TITAN [7] and JanusGraph [8]. We chose the Neo4j open source graph database [9] – a labeled property multigraph [71]. Neo4j implements a native disk-based storage manager for graphs and it is nowadays the most popular graph database system (according to DB-Engines Ranking) [55]. In Neo4j, every edge must have a relationship type and there are no restrictions on the number of edges between two nodes. Both vertices and edges can have properties (key-value pairs). Neo4j offers programming tools, drivers and libraries, including an object-oriented API for Java, currently used in our implementation. Our operators are being mapped to Cypher commands, according to read and write query

[6]DB-Engines Ranking of Graph DBMS (accessed on may, 2017) [db-engines.com/en/ranking/graph+dbms]

[7]titan.thinkaurelius.com

[8]janusgraph.org

[9]neo4j.com

structures.

To allow the temporary creation of user-specified perspectives (i.e., views), we introduce the notion of *session*. This allows the construction of several what-if scenarios, without the need to permanently store them, so that experts can explore alternatives. A session is an interactive information interchange between users and our framework. It is initiated upon request at a certain point in time, and then closed at some later point. Our framework identifies each session by a *session id* created by the framework at session begin. Sessions allow users to interact independently with their own views.

To illustrate view generation in the prototype, the next section presents a real environmental management case study based on analysis of water resources, in which user demands are translated into different views.

4.6 Case Study: Providing Perspectives of the Water Resources Database for Environmental Resource Applications

We return to our motivating scenario, of environmental analysis centered on management of water resources. This section discusses some of the computational analyses required by ANA experts, many of which have proved to be hard to develop on top of the existing (relational) database *pgHydro* (see Figure 4.1), and explains how such analyses can be easily represented in G_{Hydro} (see Figure 4.7) or through views over G_{Hydro}.

Once our prototype was implemented in Neo4j, the queries presented in this section are expressed in Cypher – the native graph query language of Neo4j. The goal is not to compare performance, but to emphasize the "reachability" issue intrinsic to these routines, explicitness and readability of queries and low maintenance cost. There are many repeated efforts in literature that compare the performance between graph and relational databases [80, 63].

First of all, it is important to deal with data consistency. The water resources database is organized according to Otto Pfafstetter methodology [69]. That organization implies that the drainage network must be represented as a binary tree-graph, connected and acyclic, whose edges go from the leaves to the root. Hence, to execute data consistency tests over G_{Hydro}, there are at least two important features to check: connectivity of all stretches and the binary tree-graph structure. The first feature can be ensured verifying if there are unconnected vertices. To check that, we perform the Cypher query, that returns the unconnected vertices:

```
START n = node(*)
MATCH n-[r?]-()
WHERE r IS NULL
RETURN n
```

If at least one vertex is found, the database is inconsistent. The second constraint, the binary tree structure, is checked selecting all vertices whose degree value is different

from 1 (start or end points) or 3 (river confluences), provided there are no cycles (since the real world problem assumes mapping to trees). If at least one such vertex is found, the database is inconsistent. To check that, we perform the Cypher query:

```
START n = node(*)
MATCH n-[r]-()
WITH count(r) as c
WHERE c ⟨ ⟩ 1 AND c ⟨ ⟩ 3
RETURN n, c
```

Next, we present some computational analyses that require to build views over G_{Hydro} and how to solve them using Graph-Kaleidoscope.

All examples that follow are based on requirements by ANA experts. In the relational database implementation of ANA, they either require queries with many recursive self joins or long stored procedure code. Since the DBMS query system did not support the computation of our examples, instead, are based on two steps. Step (1) consists in constructing a view that rearranges the underlying graph into a new graph. Step (2) poses a Cypher query on this new graph.

Moreover, we selected situations in our case study in which we can illustrate views as depicted in Cypher. Example 4.6.1 shows graph views in a restriction role. Example 4.6.2 illustrate how our views can be used to restructure data. Finally, 4.6.3 portrays the role of views in support multiscale analysis.

4.6.1 View $GV_{Hydro454}$: Determining the Longer Drainage Stretch of Watershed 454

The official territorial unit for the management of water resources adopted by ANA is a watershed. A watershed is composed by a set of drainage points and stretches. It delimits a drainage system channel and comprises the entire area that separates different water flows. Every watershed has one main watercourse; following its layout, the watershed can be split into a set of sub-watersheds and the process is applied recursively, as shown in Figure 4.17.

Each watershed receives a numeric ottocode[10] and the sub-watersheds have the same ottocode as prefix (e.g., as shown in Figure 4.17, watershed 454 has 9 sub-watersheds ottocoded as 4542, 4543,..., 4549). More details about this methodology can be found in [69].

Two steps are needed to determine the longer drainage stretch of the watershed with ottocode 454 in G_{Hydro} by constructing a view that restrict data. The view corresponding to step (1), here called $GV_{Hydro454}$, creates a graph which is a subset of G_{Hydro}, containing vertex type **DrainagePoint** and edge type **is_connected** and only the occurrences of **is_connected** whose attribute *ottocode* starts with the value *454*. This can be constructed using the operator restriction, defined as:

[10]Thus called because of the Otto Pfafstetter methodology adopted

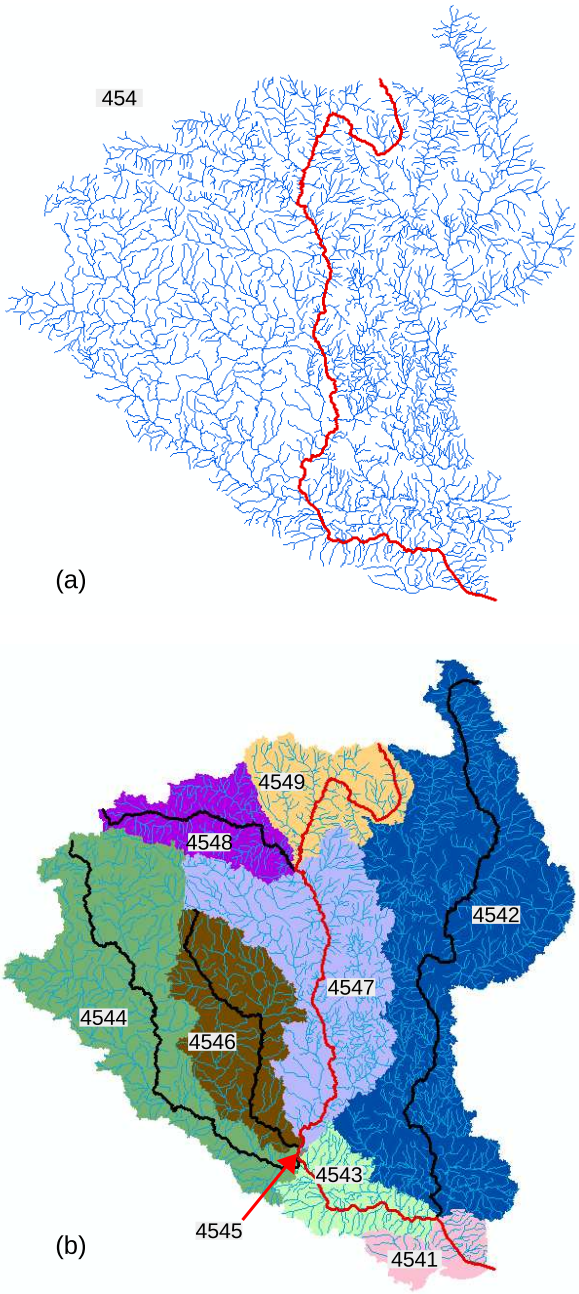


Figure 4.17: Ottocoded Watersheds - The code itself follows Pfafstetter methodology

$$\sigma\ (substring(\textbf{is\_connected}.\text{ottocode}, 1, 3) = 454)\ G_{Hydro}.(\textbf{DrainagePoint, is\_connected, DrainagePoint})$$

where the restrict predicate checks for stretches with prefix 454 (substring is an auxiliary function defined as *substring (original string, start position, length)*. The resulting view $GV_{Hydro454}$ has the same schema that G_{Hydro} and has 3.233 drainage stretches.

Step (2) performs a query over $GV_{Hydro454}$, to retrieve the drainage stretch having the longest length. This query can be expressed in Cypher as:

```
MATCH ()-[r:is_connected]-()
WITH max(r.length) as max
```

RETURN r

The result of this query is the occurrence of the **is_connected** edge type: (**is_connected**, (**DrainagePoint**, {id:101961}), (**DrainagePoint**, {id:102102}), {stretch: 436742, length: 47.94, ottocode: 4544665}) whose length is 47.94 km.

4.6.2 View GV_{River}: Determining the Most Connected River

A river is a logical element of the water resources database. It is composed by all drainage stretches that are connected and have the same hydronym (i.e., river name) - an immutable attribute, which we associate with each stretch (named waterbody in our graph database). Figure 4.18 partially shows the drainage network under this perspective, that must be constructed using a view. Here, besides length and average discharge, an important river will be highly connected with other rivers and this is the feature that this query is looking for.

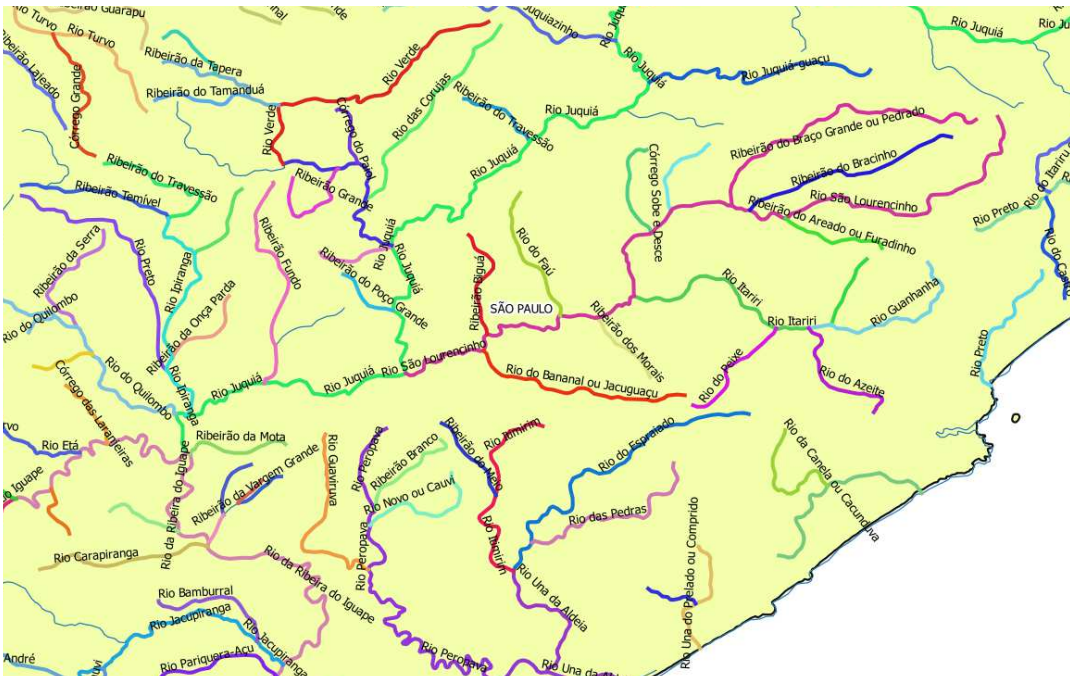


Figure 4.18: Rivers View of Drainage Network

Two steps are necessary to determine the most connected river in G_{Hydro} by constructing a view that restructures data. The view corresponding to step (1) here called GV_{River}, creates a graph in which each vertex, labelled as **River**, represents an entire river and edges, labelled as **is_connected**, represent connections between rivers. This can be constructed using operators group, attribute creation, rename and edge creation, defined as:

$GV_{P1} = \gamma$ (**River**, ic_1.waterbody) α (**River**, neighbours, {ic_2.waterbody})
G.(**DrainagePoint**, **is_connected as** ic_1, **DrainagePoint**,**is_connected as** ic_2, **DrainagePoint**)

$$GV_{P2} = \rho\ (\textbf{River}, \text{waterbody, name})\ GV_{P1}$$

$$GV_{River} = \varepsilon\ ((\textbf{is\_connected, River, River}), (R_a.name \in R_b.neighbours))$$
$$GV_{P2}.(\textbf{River}\ \text{as}\ R_a, \textbf{River}\ \text{as}\ R_b)$$

View GV_{P1} was created by composing operators group and attribute creation. This view is a graph having the vertex type **River** with two attributes: *waterbody*, the name of the river, and *neighbours*, a set of the river names that share a drainage point with that waterbody. GV_{P1} is the input for the rename operator, which in turn creates the view GV_{P2}. GV_{P2} is next used by the edge creation operator to create GV_{River}, which connects two rivers R_a and R_b if the name of R_a is on the list of neighbours of R_b. The resulting view GV_{River} is partially shown in Figure 4.19. G_{Hydro} has 54.267 rivers.

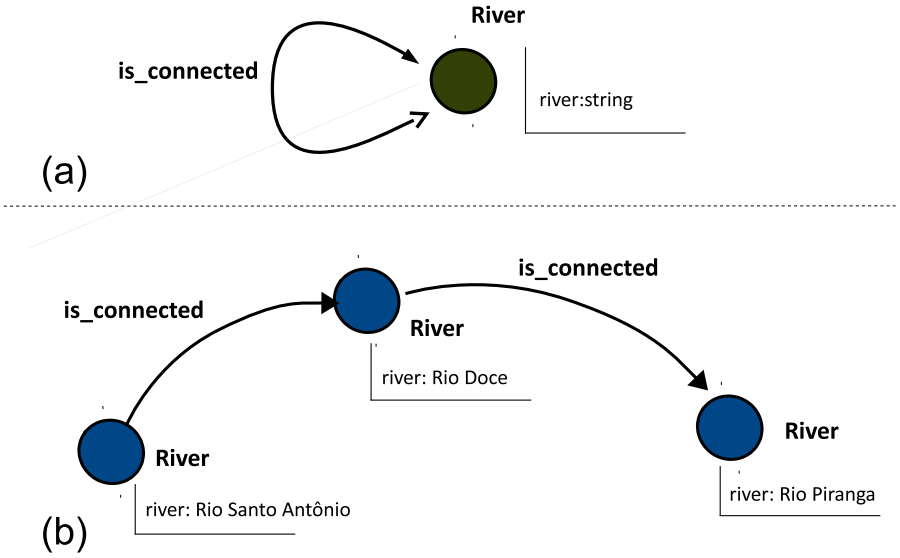


Figure 4.19: GV_{River} graph view schema (a) and state (b)

Step (2) performs a query over GV_{River} to retrieve the vertex with most connections (higher degree). This query can be expressed in Cypher as:

```
START n = node(*)
MATCH n-[r]-()
RETURN n, count(r) as c
ORDER BY c desc
LIMIT 1
```

The result of this query is *"Rio Sao Francisco"*, the fourth longest river in Brazil that crosses 5 states. Our query shows that *"Rio Sao Francisco"* is connected with 303 other rivers.

4.6.3 View $GV_{Watershed}$: Determining the Most Influential Sub-watershed of a Given Watershed

As shown in Figure 4.17, every watershed has one main watercourse that divides it into a set of sub-watersheds, building a watershed hierarchy. Each drainage stretch, part of the

watershed, has a hydrographic catchment area (HCA). An HCA, partially exemplified in Figure 4.20, is an extent or an area of land where all surface water from rain converges to a single point at a lower elevation, usually the exit of the basin, where the waters join another body of water. An HCA is represented by a polygon and its area.

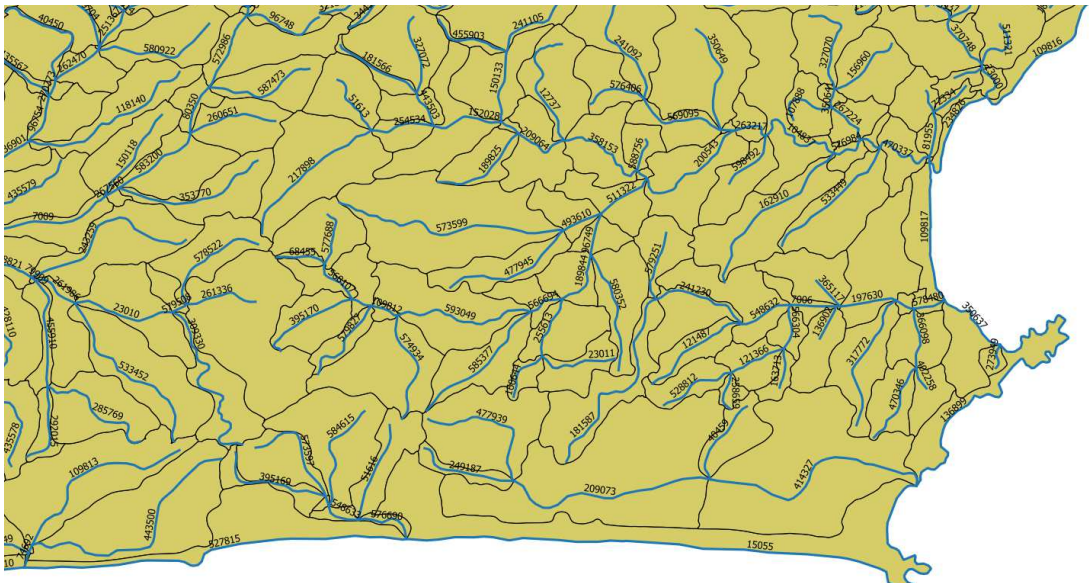


Figure 4.20: HCA: drainage stretches and their hydrographic catchment area

The influence of a watershed can be measured by the accumulated HCA of all drainage stretches that are part of the watershed. This influence is the feature that this query is looking for.

Let us take the watershed 454, illustrated in Figure 4.17, as an example. Two steps are necessary to identify the most influential sub-watershed of watershed 454 in G_{Hydro}. This is an example of multiple scales view – each ottocode level can be interpreted as a watershed scale.

The view corresponding to step (1) here called $GV_{Watershed}$, creates a graph in which each vertex, labelled **Watershed**, represent an watershed and edges, labelled **part_of**, represent the connections between watersheds. To reduce the data to be processed, this view will be created on the top of $GV_{Hydro454}$ instead of GV_{Hydro}. This can be constructed using operators group, attribute creation, union, and edge creation, in the following syntax:

$$GV_{P1} = \gamma\ (\textbf{Watershed}, \mathit{substring}(\textbf{is\_connected}.\text{ottocode},1,3))$$
$$\alpha\ (\textbf{Watershed}, \text{hca}, \text{sum}(\textbf{is\_connected}.\text{hca}))\ GV_{Hydro454}.(\textbf{DrainagePoint, is\_connected, DrainagePoint})$$

$$GV_{P2} = \gamma\ (\textbf{Watershed}, \mathit{substring}(\textbf{is\_connected}.\text{ottocode},1,4))$$
$$\alpha\ (\textbf{Watershed}, \text{hca}, \text{sum}(\textbf{is\_connected}.\text{hca}))\ GV_{Hydro454}.(\textbf{DrainagePoint, is\_connected, DrainagePoint})$$

$$GV_{P3} = \bigcup\ (GV_{P1}, GV_{P2})$$

$GV_{Watershed} = \mathcal{E}((\textbf{part\_of, Watershed, Watershed}), (\text{substring}(W_a.\text{ottocode}, 1, 3) = W_b.\text{ottocode})\ GV_{P3}.(\textbf{Watershed}\ \text{as}\ W_a, \textbf{Watershed}\ \text{as}\ W_b)$

The first group operator creates GV_{P1} with the watersheds level 3 (ottocodes with length 3) and the second creates GV_{P2} with the watersheds level 4 (ottocodes with length 4). In both cases, the attribute creator operator to give rise to HCA sum needs to be performed in the same GP than the group operator – the reason why both are performed together.

Both outputs are inputs to the union operator, that creates GV_{P3}, finally used by the edge creation operator to create $GV_{Watershed}$, that connects two watersheds W_a and W_b if the prefix of the ottocode of W_a is equal to the ottocode of W_b. The resulting view $GV_{Watershed}$ is partially shown is Figure 4.21, resulting in 10 vertices.

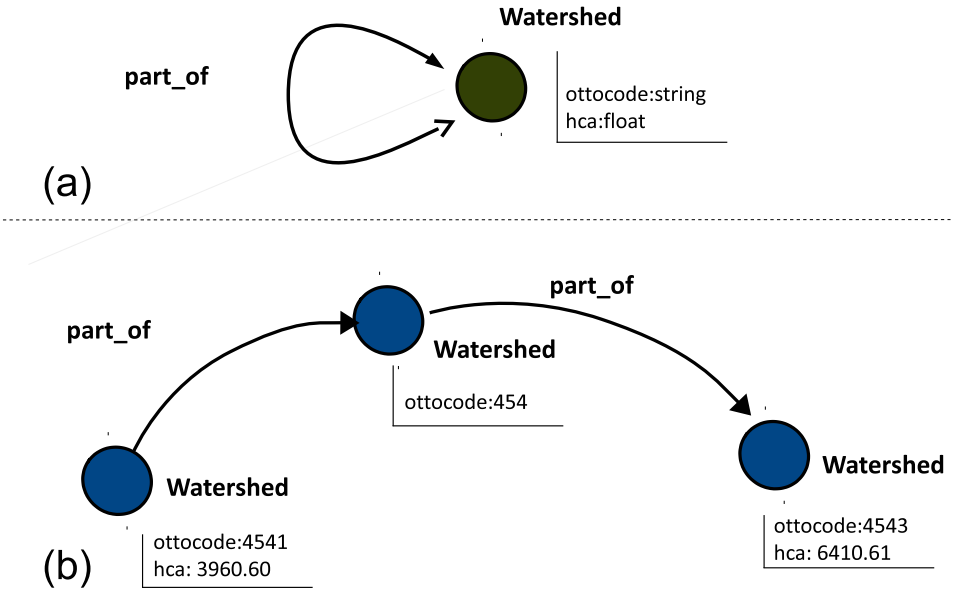


Figure 4.21: $GV_{Watershed}$ graph view schema (a) and state (b)

Step (2) performs a query over $GV_{Watershed}$ to retrieve the sub-watershed with highest HCA of $GV_{Watershed}$. This query can be expressed in Cypher as:

```
START n = node:nodes(ottocode = '454')
MATCH n-[]-(m)
WITH max(m.hca) as max
RETURN m
```

The result of this query is sub-watershed (**Watershed**, {ottocode: 4542, hca: 41.360,68}), with HCA 41.360,68 km^2.

4.7 Related Work

Some parts of the related work has already appeared throughout this text, in particular in Section 4.3, and will not be repeated here. The first proposal for views in graphs appeared

in the 80's, together with the first graph data models. Kunii's research [56] very briefly approaches graph views in her thesis, proposing a view definition composed by a list of vertex type definitions and list of link type definitions. Each definition consists of two elements: a specification of its structure and a specification of mappings that describe how they are derived from a schema. The main limitation of this proposal was to assume that only select expressions (with filters) are enough to create graph views. This work was not followed by others, and as far as we know views on graphs were not further exploited as such in related work.

With the arrival of graph database management systems, some view mechanisms have started to appear in the last 3 years. An example of construction of views in graph databases appears in [38]. The research concerns the development of efficient algorithms to answer graph queries using a set of graph views. Here, a graph view is a graph pattern query, i.e., a subgraph with a set of pattern nodes and a set of pattern edges. This was proposed to support handling data in large and distributed databases.

Another example is found in [11], for optimization of view maintenance over graph databases; this concentrates on efficiency issues, for deductive graph databases. A view definition specifies the graph patterns used to derive views. For each match of the graph pattern, an annotation is created to mark the match. A view graph consists of a set of annotations that references all nodes that participate in a certain match. This approach allows to deal with multiple graphs in a single view.

Similar to us, [43] proposes a graph data model. The authors define a graph data structure, integrity rules and a set operators to deal with graph data based on property graphs and hypernodes. The main difference from our approach is in the set of operators and the kind of graph restructuring allowed. The operators of [43] are based on GraphQL[11] and UML builders, separating the vertex on classes and instances and classifying the edges as generalization, aggregation or composition. There is the idea of bind graph patterns with graph templates, however it does not properly define or formalize the operators.

Last but not least, the notion of graph views is also proposed for domains in which relationships are analyzed under the so-called *complex networks*, which are sometimes materialized in graph databases (e.g., [61]). The analysis of complex networks, however, is not geared towards database-related issues; rather, the focus is on extracting and mining relationships and patterns of connections across data elements. In this context, view-like mechanisms are proposed to extract a portion of a network for analysis, but the construction of such views is based on invoking queries using the database language.

Figure 4.22 presents a comparative table of these proposals and our framework in terms of the different purposes of views presented in Figure 4.6. As can be seen, the general idea of views in graphs is present in all approaches – they all deal with graph data and adopt some kind of view definition mechanism/paradigm to describe how to derive graph data from other data graphs. The main difference is the approach adopted: many employ graph patterns to formulate graph queries. We, instead, define generic operators that can be used to construct views over generic graph databases. Our operators allow to achieve all purposes of a view presented in Figure 4.6, mainly aggregation and restructuring not

[11] graphql.org

cover by other works. Therefore, a view definition is also considered as graph query over a set of base graphs and/or view graphs for deriving new view graphs.

	Graph Data Structure	Goal			Approach
		Restrict	Aggregate	Restructuring	
Kunii, 1983	Records and Links	Yes	No	No	Conditional Select and Mapping
Fan, 2014	Directed Labeled Graph	Yes	No	No	Graph Pattern
Beyhl, 2015	Directed Graph with Attributes on Nodes	Yes	No	No	Graph Pattern
Ghrab, 2015	Property Graph + Hypernode	Yes	No	No	Graph Operators
Lysenko, 2016	Property Graph	Yes	No	No	Graph Query
Graph-Kaleidoscope	Property Graph	Yes	Yes	Yes	Graph Operators

Figure 4.22: Related Work – Comparative Table

4.8 Research Challenges and Lessons Learned

The notion of view as a particular perspective built on the underlying data is independent from the database model. View mechanisms, however, are strongly dependent on the underlying model. Graphs were chosen by us as the database model, since they are appropriate to represent highly connected data, in which the connectivity information is as important as the connected entities themselves (again, a requirement in environmental applications). Graph models support explicit representation of relationships, as opposed to the more traditional database (relational) models, which require costly computations to derive such relationships – e.g. via foreign keys and joins.

This choice, though appropriate to solve expert requirements, posed many theoretical and implementation challenges. First, there is no consensual graph data model (namely, neither consensual data structure nor query language), and thus we had to propose PGDM, discussed in this paper, to support all required operations. Second, once a graph database is created according to certain user needs, it is almost impossible to "turn the data over" to support alternative perspectives. Our views provide this restructuring possibility, without the need to create a new database. Third, as also stressed by [11], graph databases do not yet support views (in the sense that the view concept is not fully supported by such databases, as opposed to relational or object-oriented views). Therefore, we had to define the appropriate theoretical infrastructure.

Though seemingly simple, supporting views in graph databases involves many challenges that we had to face. The first is how to specify and compute a view. In rela-

tional databases, views are defined through database queries, using SQL. Unlike these databases, there is no consensual query language for graph databases. Thus, to specify Graph-Kaleidoscope, we had to analyze and compare several graph representations. As a result, we concluded that we could not choose "the most adequate" model, given that each proposal is geared towards distinct goals.

Another important issue, still connected with view generation, is the type of the result returned. Queries in relational databases return tables (or values, in specific cases) - and thus a view is always a table. Most graph query languages, however, can return graphs, tables, or values. Since for us views have to be graphs, this posed the challenge of defining appropriate operators, so that the result would always be a graph. All these problems related to view generating functions were solved by us through the definition of our elementary operators, all of which receive a graph as input, and produce a graph as output. Their implementation in graph query languages is still an open problem, since this will depend on each language.

We also had to choose between materializing views, or maintaining them as temporary structures. Each of these options has pros and cons, discussed in relational database view literature. We solved this through the definition of "sessions", thereby allowing temporary views, and at the same time supporting materialization when needed. A discussion of the advantages and disadvantages of these options is outside the scope of our work, since it involves, among others, performance issues, which we are not concerned with.

Last but not least, the choice of the framework persistence layers was a research challenge. NOSQL databases are a field in which there is much debate – e.g., are they really database solutions or just products that are being developed to meet performance needs that cannot be satisfied by relational databases ? To this end, which kind of features are being disregard [77]? Related work shows that graph databases bring flexibility to many applications, being moreover inspired by a robust mathematical model, and centuries of algorithms that have been thoroughly studied. Many real world problems can be directly mapped to graphs. However, from an implementation point of view, there is still much to be done before graph databases can be directly compared with relational databases.

4.9 Conclusions and Ongoing Work

This paper presented Graph-Kaleidoscope, a computational framework that supports management of views in graph databases. Views, here, are provided to let experts exploit multiple perspectives from the underlying data. This research was motivated by the needs from researchers that deal with environmental, geographic data, characterized by a wide heterogeneity of data that are highly related across multiple spatial and temporal scales. This paper contributes therefore towards solving problems of multi-perspective research in applications that are characterized by inter-disciplinarity (and thus multiple ways of analyzing a problem). Graph-Kaleidoscope is characterized by: (1) use of graph databases to store and analyze datasets of highly connected data; (2) adapting the concept of views from relational databases to represent the idea of focus; and (3) specification and implementation of a graph view framework to support views over graph databases – views that

are themselves graphs.

Though we take advantage of work published on relational databases, our work distinguishes itself in at least two points. First, though view mechanisms are recognized as useful, no such mechanisms exist in graph databases, for lack of a consensual model. Second, unlike other proposals, our graph view operators allow creation of graphs with a topology that is completely different from that of the underlying data, to accommodate distinct semantic needs on top of stored data.

This notwithstanding, our specification is general enough so that Graph-Kaleidoscope can be used in other research contexts, in which data present the same kind of characteristics (e.g., for domains that deal with complex multi-scale networks, such as health or biodiversity). Our paper also presents a preliminary prototype of our framework, used to solve needs of researchers in water resource management, for a real life case study, covering Brazil's entire water network resources. The database used is public domain and it is available in the official website of the Brazil National Water Agency.

Ongoing work covers both theoretical and implementation issues. Implementation efforts require improvement in the framework's code, in particular considering the underlying data catalogue, and generating function storage and indexing. Another direction involves designing and developing a user-friendly interactive interface, to help users define and explore views. Still another issue is to try to (re)implement the framework using another "stack" of graph representation and technology, e.g., RDF-SPARQL. From a theoretical point of view, we might think of considering other operators, e.g., defined as a combination of our elementary operators, and to explore geographic features of data.

Chapter 5

Conclusions and Extensions

5.1 Overview

The research presented in this thesis concerns challenges in data-intensive science, in particular to overcome the problem of supporting multiple perspectives on heterogeneous datasets. Our motivation came from interdisciplinary and multifocus research, where each stakeholder has a particular perspective on a given problem, e.g. dealing with a subset of data, adopting specific vocabularies or considering objects from distinct domains together. This scenario usually requires complex data transformation processes to create multiple representations of the same real work phenomena.

The main tenet verified of this PhD research is that graph databases are a suitable approach for handling a wide range of demands for managing heterogeneous data. Moreover, the concept of views, from relational databases, can be applied to provide multiple perspectives, as proposed by [73].

The unstructured nature of data and the high level of importance of data connections led the research to adopt the graph data management paradigm, a schemaless relationship-driven NoSQL solution. Our definition of views is based on view generating functions, for us a composition of graph operators. These operators are part of our PGDM model, the property graph data model defined in this research. Based on this approach, we specify and implement a prototype of a framework to handle views over graph databases, named Graph-Kaleidoscope. The specification of our operators and framework are as generic as possible and they can be implemented in different graph database engines.

Two different real world datasets – in biodiversity and in environmental resources – were analyzed to gather the requirements of view mechanisms and to validate our research. Both case studies can clearly benefit from our framework to perform complex queries. We also believe that our solution can be extended and adopted by other kinds of application domains with similar management and analysis requirements.

5.2 Main Contributions

Our first contribution, presented in Chapter 2, was to present the case for the use of graph databases in multifocus research. From a study of data management requirements, we

justify the use of graph databases as a suitable persistence layer to meet these requirements and to store/analyze datasets of highly connected data.

The second contribution of this thesis, presented in Chapter 4, is to propose a property graph data model (PGDM) with a set of operators to manipulate and retrieve graph data. Ours is a flexible approach, to be incorporated in any graph data structure and query language. This is the main contribution of our thesis, proposed to fill the gap of the absence of a full-fledged data model for graph databases.

The third contribution, introduced in Chapter 2 and formalized in Chapter 4, is to define views for graph databases to support the need for multiple perspectives in interdisciplinary research. Views are specified through view generating functions, considering graph data manipulation, classical algorithms and traversal tasks.

Our fourth contribution, presented in Chapters 2, 3 and 4, is to analyze real life examples of interdisciplinary research and how they can benefit from our proposal. We present how biodiversity and environmental resource datasets can be modeled and explored by experts using graph databases and multiple perspectives, pointing out the advantages of this approach.

The last contribution is presented in Chapter 4: the specification and implementation of a prototype of Graph-Kaleidoscope to support views over graph databases using a graph database engine.

5.3 Extensions

This research can be extended to different practical/implementation and theoretical aspects. Some possibilities are listed below.

- Document the prototype source code and make it available in a software repository. This action will help disseminate our ideas and results outside the academic field to the graph database community and start a practical discussion about how they can bring new benefits to users and applications.

- Investigate performance issues to perform adaptations in our framework to improve performance, i.e., to use less computational resources and to reduce execution time. Index structures are usually adopted in conventional relational databases to improve performance in data manipulation.

- Develop a graphical user interface to our prototype. The graph data structure is often better understood in a visual way. Analogously, to provide a graphical interface for view definition and exploration would improve our prototype, make it more interesting to stakeholders.

- Include graph database configurations in our prototype to allow other graph database management systems besides Neo4j. DB-Engine Ranking[1] indicates the existence of 26 graph databases engines, of which 9 are open source native graph database systems and strong candidates to be used as persistence layer of our prototype.

[1] db-engines.com/en/ranking/graph+dbms

- Implement a new prototype using another stack of standards of graphs. As presented in the thesis, RDF is a possible option, but a more complex one. The graph nature of RDF triple syntax (subject-predicate-object) is indeed appealing, but the RDF specification[2] falls short of a definition of a graph in a mathematical sense – e.g., the definition of the set of vertices and edges, as pointed out by [49]. Consider a RDF structure in which some resource p occurs as a predicate of some statements and as the subject of others - should p be considered a vertex or an edge? Work with RDF as a graph model will demand additional formalisms to mapping RDF statements in graphs to base our framework. On the other hand, our framework would benefit from the well established SPARQL query language to implement our operators.

- Gather new requirements of multifocus research. Due to the complexity involving heterogeneous datasets, our thesis delimits a scope and a list of research problems to deal with. Indeed, many other requirements can arise outside this initial scope, for example, the need to dis aggregate a data record in smaller pieces of data. In Graph-Kaleidoscope, this requirement may result in a new operator capable of splitting a vertex or to create multiple edges between two vertices with a different meaning, for instance.

- Design adaptations in our framework to explore semantic aspects of data. Connectivity of data can refer to some semantic relation and it is possible to formalize the context of a dataset through ontologies [45]. An initial approach is the edge creation operator, which could consider as a possible predicate the existence of a semantic relation between two attribute values, for instance.

- Design adaptations in our framework to explore geographic aspects of data. As shown by our environmental resources case study, connectivity can also express some kind of spatial correlation about data records. Thus, to consider geographic coordinates in vertices, for instance, could be helpful in aggregation tasks or even to restrict or project data based on topological relations. Another related extension would be to include temporal predicates, e.g., temporal algebra.

- Design adaptations in our framework to accommodate classical graph algorithms. Many analyses can be done finding a minimum spanning tree or a clique. Some cases are mappable to our traversal operator, while other would be better performed with specific algorithms. The proposal is centered on the adapting relational operators to graph databases. Other graph analytical operators could also be examined.

- Analyze other multidisciplinary domains and datasets to validate the appropriateness and flexibility of our solution. The health care domain, for instance, is composed of many unstructured and highly connected data and requires interpretations in many complexity levels, such as patients, drug, diseases, medical treatment protocols.

[2] www.w3.org/TR/2014/REC-rdf11-concepts-20140225

- Improve the integrity rules of PGDM. By definition schemaless, graph databases and data models usually do not support constraints and integrity rules. The fact that is not necessary to first create a graph schema and then use it leads to integrity being frequently forgotten. Investigate this topic would bring more reliability to our data model and therefore to our framework, e.g., adding business rules.

- Investigate issues motivated with view materialization, e.g., for performance issues. This would also require conducting research on update propagation, view updatabiliting and refresh mechanisms.

Bibliography

[1] S. Amer-Yahia, L. V. S. Lakshmanan, and C. Yu. Socialscope: Enabling information discovery on social content sites. *CoRR*, abs/0909.2058, 2009.

[2] P. Amirian, A. Basiri, and A. Winstanley. Efficient online sharing of geospatial big data using nosql xml databases. In *Proceedings of the 2013 Fourth International Conference on Computing for Geospatial Research and Application*, COMGEO '13, pages 152–, Washington, DC, USA, 2013. IEEE Computer Society.

[3] P. Amirian, A. Basiri, and A. Winstanley. Evaluation of data management systems for geospatial big data. In *Computational Science and Its Applications - ICCSA 2014*, volume 8583 of *Lecture Notes in Computer Science*, pages 678–690. Springer International Publishing, 2014.

[4] J. C. Anderson, J. Lehnardt, and N. Slater. *CouchDB: The Definitive Guide Time to Relax*. O'Reilly Media, Inc., 1st edition, 2010.

[5] R. Angles. A comparison of current graph database models. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops*, ICDEW '12, pages 171–177, Washington, DC, USA, 2012. IEEE Computer Society.

[6] R. Angles and C. Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, February 2008.

[7] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. the gene ontology consortium. *Nature Genetics*, 25(1):25–29, May 2000.

[8] M. Atkinson, D. DeWitt, D. Maier, F. Bancilhon, K. Dittrich, and S. Zdonik. The object-oriented database system manifesto. In François Bancilhon, Claude Delobel, and Paris Kanellakis, editors, *Building an Object-oriented Database System*, pages 1–20. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[9] P. G. Baker, A. Brass, S. Bechhofer, C. Goble, N. Paton, and R. Stevens. TAMBIS–Transparent Access to Multiple Bioinformatics Information Sources. In *Int Conf Intelligent Systems for Molecular Biology*, volume 6, pages 25–34, Montreal, Canada, June 1998.

[10] P. Barcelo, C. Hurtado, L. Libkin, and P. Wood. Expressive languages for path queries over graph-structured data. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '10, pages 3–14, New York, NY, USA, 2010. ACM.

[11] T. Beyhl and H. Giese. Efficient and Scalable Graph View Maintenance for Deductive Graph Databases based on Generalized Discrimination Networks. Technical Report 99, Hasso Plattner Institute at the University of Potsdam, 2015. Technical Report No. 99.

[12] C. Bizer. D2rq - treating non-rdf databases as virtual rdf graphs. In *In Proceedings of the 3rd International Semantic Web Conference*, 2004.

[13] V. Bonstrom, A. Hinze, and H. Schweppe. Storing rdf as a graph. In *Web Congress, 2003. Proceedings. First Latin American*, pages 27–36, Nov 2003.

[14] P. Bouquet, M. Ehrig, J. Euzenat, E. Franconi, P. Hitzler, M. Krötzsch, L. Serafini, G. Stamou, Y. Sure, and S. Tessaris. Specification of a common framework for characterizing alignment. Knowledge Web Deliverable 2.2.1v2, University of Karlsruhe, DEC 2004.

[15] D. M. Boyd and N. B. Ellison. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1):210–230, 2007.

[16] U. Brandes and T. Erlebach. *Network Analysis: Methodological Foundations (LNCS)*. Springer-Verlag New York, Inc., Secaucus, USA, 2005.

[17] Brazil. Lei federal de recursos hídricos (9.433). Diário Oficial [da] República Federativa do Brasil, Poder Executivo, Brasília, 9 jan. 1997. Seção 1, p. 470, january 1997.

[18] C.A. Brebbia and V. Popov. *Water Resources Management VI*. WIT transactions on ecology and the environment. WIT Press, 2011.

[19] D. Calvanese, G. Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, *KR*, pages 176–185. Morgan Kaufmann, 2000.

[20] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting of regular expressions and regular path queries. *Journal of Computer and System Sciences*, 64(3):443 – 465, 2002.

[21] C. Caracciolo, A. Stellato, A. Morshed, G. Johannsen, S. Rajbhandari, Y. Jaques, and J. Keizer. The agrovoc linked dataset. *Semantic Web*, 4(3):341–348, 2013.

[22] P. Cavoto and A. Santanche. Fishgraph: A network-driven data analysis. In *Proceedings of the 11th IEEE International Conference on eScience*, pages 1–10, Munich, Germany, 2015.

[23] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, June 2008.

[24] C.L. Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275:314 – 347, 2014.

[25] E. F. Codd. Data models in database management. *SIGMOD Rec.*, 11(2):112–114, June 1980.

[26] D. Colazzo and C. Sartiani. Typing query languages for data graphs. *I. W. on Graph Data Management: Techniques and Applications*, 2014.

[27] M. P. Consens and A. O. Mendelzon. Low complexity aggregation in graphlog and datalog. In *Proceedings of the third international conference on database theory on Database theory*, ICDT '90, pages 379–394, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[28] The International Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, February 2001.

[29] D. C. Cugler, C. B. Medeiros, and L. F. Toledo. An architecture for retrieval of animal sound recordings based on context variables. *Concurrency and Computation: Practice and Experience*, pages 1–17, 2011.

[30] R. Cyganiak, D. Wood, and M. Lanthaler. RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, February 2014. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[31] J. Daltio and C. B. Medeiros. Aondê: An ontology web service for interoperability across biodiversity applications. *Information Systems*, 33(7-8):724–753, 2008.

[32] J. Daltio and C. B. Medeiros. HydroGraph: Exploring Geographic Data in Graph Databases. In *Proc XVI Brazilian Symposium on Geoinformatics*, pages 44–55, 2015.

[33] J. Daltio and C. B. Medeiros. Hydrograph: Exploring geographic data in graph databases (extended version). *Brazilian Journal of Cartography*, 68(6):1181–1189, 2016.

[34] J. Daltio and C. B. Medeiros. Graph-kaleidoscope: A framework to handle multiple perspectives in graph databases). *International Journal of data Science and Analytics*, 2017.

[35] J. Daltio and C. Ba. Medeiros. Handling multiple foci in graph databases. In Springer International Publishing Switzerland, editor, *Lecture Notes in Bioinformatics (LNBI) - Proceedings of 10th International Conference on Data Integration in the Life Sciences*, volume 8574, pages 58–65, Lisboa, Portugal, 2014.

[36] V. Dhar. Data science and prediction. *Commun. ACM*, 56(12):64–73, December 2013.

[37] N. Dimiduk and A. Khurana. *HBase in action.* Manning Publications, 1st edition, 2012.

[38] W. Fan, W. Wang, and Y. Wu. Answering Graph Pattern Queries Using Views. In *Proc. 30th International Conference Data Engineering - ICDE*, pages 167–176, 2014.

[39] D. Florescu, A. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '98, pages 139–148, New York, NY, USA, 1998. ACM.

[40] P. Fox and J. Hendler. Changing the equation on scientific data visualization. *Science (New York, N.Y.)*, 331(6018):705–708, February 2011.

[41] B. J. Fry. *Computational Information Design.* PhD thesis, Massachusetts Institute of Technology, 2004. AAI0806331.

[42] A. Furtado, K. Sevcik, and C. Santos. Permitting updates through views of databases. *Informations Systems*, 4:269–283, 1979.

[43] A. Ghrab, O. Romero, S. Skhiri, A. A. Vaisman, and E. Zimányi. Grad: On graph database modeling. *CoRR*, abs/1602.00503, 2015.

[44] T. Goodwin and S. M. Harabagiu. Automatic generation of a qualified medical knowledge graph and its usage for retrieving patient cohorts from electronic medical records. In *Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on*, pages 363–370, 2013.

[45] T. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal of Human-Computer Studies*, 43(5-6):907–928, 1995.

[46] N. Guarino. Formal ontology and information systems. In *Proceedings of Formal Ontology in Information System*, pages 3–15. IOS Press, 1998.

[47] A. Halevy. Answering Queries using Views: a Survey. *The VLDB Journal*, 10:270–294, 2001.

[48] S. Harris and A. Seaborne. SPARQL 1.1 query language. W3C recommendation, W3C, March 2013. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[49] J. Hayes and C. Gutierrez. *Bipartite Graphs as Intermediate Model for RDF*, pages 47–61. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[50] R. Hecht and S. Jablonski. Nosql evaluation: A use case oriented survey. In *Cloud and Service Computing (CSC), 2011 International Conference on*, pages 336–341, 2011.

[51] T. Hey, S. Tansley, and K. Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, Redmond, Washington, 2009.

[52] I. Horrocks. Ontologies and the semantic web. *Commun. ACM*, 51(12):58–67, 2008.

[53] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi. Big data and its technical challenges. *Commun. ACM*, 57(7):86–94, July 2014.

[54] F. Jouault and J. Bézivin. *KM3: A DSL for Metamodel Specification*, pages 171–185. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[55] P. Kivikangas and M. Ishizuka. Improving semantic queries by utilizing unl ontology and a graph database. In *Semantic Computing (ICSC), 2012 IEEE Sixth International Conference on*, pages 83–86, Sept 2012.

[56] H. Kunii. *Graph data language : a high level access-path oriented language*. PhD thesis, University of Texas at Austin, 1983.

[57] M. Levene and G. Loizou. A graph-based data model and its ramifications. *IEEE Trans. Knowl. Data Eng.*, 7(5):809–823, 1995.

[58] M. Levene and A. Poulovassilis. The hypernode model and its associated query language. In *Proceedings of the fifth Jerusalem conference on Information technology*, JCIT, pages 520–530, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.

[59] L. Libkin and D. Vrgoč. Regular path queries on graphs with data. In *Proceedings of the 15th International Conference on Database Theory*, ICDT '12, pages 74–85, New York, NY, USA, 2012. ACM.

[60] J. S. C. Longo and C. B. Medeiros. Providing multi-scale consistency for multi-scale geospatial data. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, SSDBM, pages 8:1–8:12, New York, NY, USA, 2013. ACM.

[61] A. Lysenko, I. A. RoznovăŢ, M. Saqi, A. Mazein, C. J. Rawlings, and C. Auffray. Representing and querying disease networks using graph databases. *BioData Mining*, 9(1):23, 2016.

[62] M. S. Martin, C. Gutierrez, and P. T. Wood. Snql: A social networks query and transformation language. In Pablo Barceló and Val Tannen, editors, *AMW*, volume 749 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.

[63] R. C. McColl, D. Ediger, J. Poovey, D. Campbell, and D. A. Bader. A performance evaluation of open source graph databases. In *Proceedings of the First Workshop on Parallel Programming for Analytics Applications*, PPAA '14, pages 11–18, New York, NY, USA, 2014. ACM.

[64] C. B. Medeiros, M. J. Bellosta, and G. Jomier. Multiversion Views: Constructing Views in a Multiversion Database. *Data & Knowledge Engineering*, 33:277–306, 2000.

[65] N. F. Noy and M. A. Musen. Specifying ontology views by traversal. In *International Semantic Web Conference*, volume 3298 of *LNCS*, pages 713–725, 2004.

[66] B. Olivier, S. Cohen-Boulakia, S. Davidson, and C. Hara. Querying and Managing Provenance through User Views in Scientific Workflows. In *Proc. International Conference Data Engineering - ICDE*. IEEE, 2008.

[67] C. Parent, S. Spaccapietra, and E. Zimányi. *Conceptual Modeling for Traditional and Spatio-Temporal Applications: The MADS Approach*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[68] Y. Park, M. Shankar, B. Park, and J. Ghosh. Graph databases for large-scale healthcare systems: A framework for efficient data management and data services. *I. W. on Graph Data Management: Techniques and Applications*, 2014.

[69] O. Pfafstetter. Classificação de bacias hidrográficas: metodologia de codificação. Departamento Nacional de Obras de Saneamento (DNOS). Rio de Janeiro, RJ, 1989.

[70] A. Poulovassilis and M. Levene. A nested-graph model for the representation and manipulation of complex objects. *ACM Trans. Inf. Syst.*, 12(1):35–68, January 1994.

[71] I. Robinson, J. Webber, and E. Eifrem. *Graph Databases*. O'Reilly Media, Incorporated, 2013.

[72] M. A. Rodriguez and P. Neubauer. Constructions from dots and lines. *Bulletin of the American Society for Information Science and Technology*, 36(6):35–41, 8 2010.

[73] A. Santanche, J. Longo, G. Jomier, M. Zam, and C. B. Medeiros. Multi-focus research and geospatial data - Anthropocentric concerns. *JIDM*, 5(2):146–160, 2014.

[74] A. Santanche, C. B. Medeiros, J. Jomier, and M. Zam. Challenges of the Anthropocene epoch - Supporting multi-focus research. In *Proc XIII Brazilian Symposium on Geoinformatics*, 2012.

[75] A. Silberschatz, H. F. Korth, and S. Sudarshan. Data models. *ACM Computing Surveys*, 28(1):105–108, March 1996.

[76] S. Spaccapietra, C. Parent, and C. Vangenot. Gis databases: From multiscale to multirepresentation. In *Proceedings of the 4th International Symposium on Abstraction, Reformulation, and Approximation*, SARA '02, pages 57–70, London, UK, UK, 2000. Springer-Verlag.

[77] M. Stonebraker. Sql databases v. nosql databases. *Commun. ACM*, 53(4):10–11, April 2010.

[78] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah. Serving large-scale batch computed data with project voldemort. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, FAST'12, pages 18–18, Berkeley, CA, USA, 2012. USENIX Association.

[79] A. A. Teixeira, A. M. Silva, G. S. F. Molleri, F. V. Ferreira, and A. J. Borelli. Pghydro - hydrographic objects in geographical database (in portugueses). In *Proceedings of the 2013 Brazilian Symposium on Water Resources*, pages 1–8, 2013.

[80] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins. A comparison of a graph database and a relational database: A data provenance perspective. In *Proceedings of the 48th Annual Southeast Regional Conference*, ACM SE '10, pages 42:1–42:6, New York, NY, USA, 2010. ACM.

[81] R. Volz, D. Oberle, and R. Studer. Implementing Views for Light-Weight Web Ontologies. In *Proc. of Int. Database Engineering and Application Symposium - IDEAS*, Hong Kong, China, 07 2003.

[82] F. Y. Wang, K. M. Carley, D. Zeng, and W. Mao. Social computing: From social informatics to social intelligence. *IEEE Intelligent Systems*, 22(2):79–83, March 2007.

[83] J. Webber. A programmatic introduction to neo4j. In *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*, SPLASH '12, pages 217–218, New York, NY, USA, 2012. ACM.

[84] P. T. Wood. Query languages for graph databases. *SIGMOD Rec.*, 41(1):50–60, April 2012.

[85] S. Zhou and C. B. Jones. A multi-representation spatial data model. In *I. S. on Advances in Spatial and Temporal DBs*, volume 2750 of *LNCS*, pages 394–411, 2003.