Journal of Data Acquisition and Processing

ISSN: 1004-9037
https://sjcjycl.cn/
DOI: 10.5281/zenodo.98549509

A NOVEL MODAL DRIVEN ENGINEERING APPROACH FOR OPTIMIZATION IN DATABASE OF LARGE SCALE IN NATURE USING CATEGORY THEORY

Leeladhar Chourasiya[1]; Dr Sunita Dwivedi[2]
[1]: Assistant Professor, Acropolis Institute of Technology and Research, Indore;
[2]: Associate Professor MCRPV Bhopal

Abstract

Data is the foundation of any contemporary software programme, and databases are the most frequent means for applications to store and handle data. Databases have progressed from classic relational databases to more sophisticated forms of databases such as NoSQL, columnar, key-value, hierarchical, and distributed databases as online and cloud technologies have proliferated. Each kind may work with organised, semi-structured, and even unstructured data.

Furthermore, databases are always dealing with mission-critical and sensitive data. When combined with regulatory constraints and the dispersed nature of most data sets, database management has become extremely difficult. As a result, enterprises need powerful, secure, and user-friendly technologies to keep these databases up to date.

In this paper we have demonstrate the how access time to data can be reduced using state-of-the-art Mathematical concepts and compare it with relative technologies.

Introduction

Businesses all around the world are growing more reliant on data to run their day-to-day operations and make informed business choices. With so much data being generated, it has become difficult to manage data throughout the corporation, which may be scattered over many geo-locations and utilising tens of business line apps. We're all aware that data is the new oil for every business. Unlocking its value may lead to enormous potential for businesses, which is why it is critical to have well-defined data management strategies in place in order to meet the most challenging difficulties that data management entails. The need to properly manage, change, and preserve these information assets has never been more critical, considering that more data has been created in the last two years than in the whole history of humanity. This condition has generally been satisfied by the major database providers. To put a stop to the confusion generated by the constant data explosion, a slew of competitors have entered the fray during the last 10 years.

What does "poor construction" mean exactly? When the database (DB) designer creates the database, he should identify the entities that rely on one another for existence and then limit the likelihood that one may ever exist independently of the other. Compared with traditional datasets, big data typically includes masses of unstructured data that need more real-time analysis. In addition, big data also brings about new opportunities for discovering new values, helps us to gain an in-depth understanding of the hidden values, and also incurs new challenges, e.g., how to effectively organize and manage such datasets. Graphs are extremely useful in understanding a wide diversity of datasets in fields. A graph database management system (henceforth, a graph database) is an online database management system with Create, Read, Update, and Delete (CRUD) methods that expose a graph data model. Graph databases are generally built for use with transactional (OLTP) systems. Accordingly, they are normally optimized for transactional performance, and engineered with transactional integrity and operational availability in mind. Graph databases are focused on efficient storing and querying highly connected data. They are a powerful tool for graph like queries, e.g., computing the shortest path between two nodes in the graph. They reach an excellent performance for local reads by traversing the graph and can use various data models for graphs and their data extensions.

For millions of years, the capacity to transmit knowledge on to future generations has propelled evolution, and in recent decades, data has risen immensely, giving rise to Large Scale Data (LSD). Businesses have understood that information is power, and that obtaining and keeping it should be a top focus. LSD is demonstrating its worth to businesses of all sizes and in a variety of industries. Enterprises that use LSD effectively reap concrete business benefits ranging from enhanced operational efficiency and increased visibility into quickly changing environments to the improvement of products and services for consumers.

The main sources of generation of LSD are primarily: from Machine, Social media, and Transactions. Furthermore, firms produce data internally through direct consumer involvement. This information is often retained within the company's firewall. It is then externally loaded into the management and analytics system.

LSD is focused on large amounts of data, which has an immediate impact on how well programmes perform (for example, when attempting to query a specific piece of information) and necessitates particular architectures to enhance them, such as the use of graph databases or distributed concurrent computations. Though there are many technologies available today to use Big Data, it is more difficult to find theories that can be used to fully comprehend the advantages and limitations of each architecture, as well as ways to combine them. Here, in order to overcome this restriction, we make use of the tools provided by Category Theory as well as a functional programming language (to put the ideas into practise and make experimenting easier).

The idea of natural isomorphism then establishes to demonstrate that two programmes or data structures convey the same information. In order to demonstrate that a natural transformation is just an optimization, the equations used to express the programmes can then be used to calculate computation steps (time complexity) and compare the results of two equivalent algorithms. The ability of Category Theory to be quickly and safely translated into the majority of functional programming languages makes it attractive for experimentation and the proposal of new tools or architectures to the big data community.

Graph databases are essential for applications that handle graph-like data. These applications, such as social networks, have recently proliferated, particularly on the web. Graph database concepts and algorithms represent a convergence of traditional domains such as database and graph theory. This background is useful to the database community in developing systems that manage graph databases. Graph Databases offer a strong option in applications like Facebook and Twitter where storing and searching for data is a crucial action in relationships. Due to issues such as high level performance, flexible schema, and scalability, RDBMS gave rise to a new storage technique known as Graph Databases. Today's world is densely interwoven, and graph databases aim to imitate those chaotic, sometimes-consistent links in a natural way. What distinguishes the graph paradigm from other database models is as follows: It more accurately represents how the human brain maps and analyses the environment around it.

Graph database querying has been studied for almost 25 years. Today, the question of how effectively searching graph databases is being forced to resurface by the rise of applications that manage large connected data. This led to the recent development of numerous query languages and algorithms for query optimization in graph databases. The apparent distinction between graph databases systems and the conventional DBMS is made by the statement that "Every graph node includes a direct pointer to its adjacent items and no index lookups are necessary." Because of this, the algorithms that have been researched for decades in graph theory serve as the foundation for all query languages for graph databases

Graph databases are required for applications that work with graph-like data. Social networking applications, for example, have recently proliferated, particularly on the web. Concepts and algorithms for graph databases represent a synthesis of traditional domains such as database and graph theory. This background will help the database community develop systems for managing graph databases. In this paper, we provided an overview of graph databases. We have concentrated specifically on the optimization techniques used to improve query response time. The majority of these techniques, such as query decomposition and pre-processing, have been used in traditional databases and distributed systems in the past. Graph databases are required for applications that work with graph-like data. Social networking applications, for example, have recently proliferated, particularly on the web. Concepts and algorithms for graph databases represent a synthesis of traditional domains such as database and graph theory. This background will help the database community develop systems for managing graph databases. In this paper, we provided an overview of graph databases. We have concentrated specifically on the optimization techniques used to improve query response time. The majority of these techniques, such as query decomposition and pre-processing, have been used in traditional databases and distributed systems in the past.

Functor - Data Structure Modeller

Some programming language deals with immutable values, if we want to transform a data structure, we'll need to create another one. Functors represent the idea of "mappable" data structures. Programming languages usually describe it as a type class since we want to link this idea to particular data structures but not others (e.g., lists, options, binary trees). The utilization of Functors to generalize APIs and avoid writing the same changes on diverse data structures. One can even add the map extension method to the data structures we want to support after generating Functor instances if they don't already have it.

For example

Ilist=List(1,2,3).map(x=>x+1)

This map transformation concept can be applied to other data structures as well.

A functor F from category C to category D (F: C → D) maps objects of C to objects of D, denoted F c for an object c of C, and morphisms f: a → b of C to morphisms of D denoted F fF a → F b, such that

- (Unit) F Idx = IDF x
- (Composition) Let f : a → b and g : b → c be morphisms of C, then F (g∘Cf) = (F g)∘D(F f).

One of the nice things about the definition of a functor is that they preserve commutative diagrams.

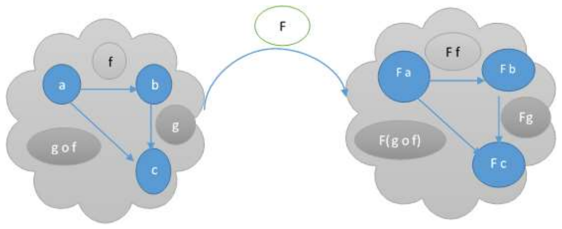


Fig 1 Commutative diagram using Functors

Data Structure Equivalency Categorically

Considered the following schema

Project (Title varchar, Start date, End date, deptid, int)

Department(deptidint, deptnamevarchar, parentdeptidint , leadpersonidint)

Person (personidint ,Firstnamevarchar, Lastnamevarchar, DOB date)

Categorical representation of above database schema is given in the diagram given below

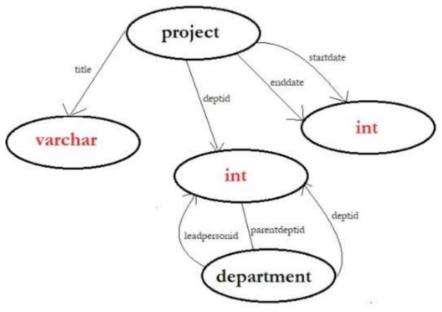


Fig 2 A simple schema as a category

Assume a functor F : C → Set consists of

• A set for each object of C and

• a function for each arrow of C, such that

• the declared equations hold

In other words, F fills the schema with data. The following SQL query on the database schema considered represented in diagram:

Select deptid AS deptid from project p, department d WHERE p.deptid =d.deptid AND p.name=”ERP”

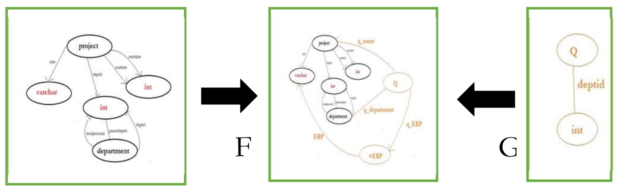


Fig 3 Categorical Representation of query and answer query

In above Figure 3, we have shown two functors F and G between the schema category C, an intermediate category Q which contains the basic schema plus objects representing the query and the query constants (e.g., Q and =ERP nodes) and arrows connecting these tables and types relevant to the query, and a simple category containing schema of the query answer A. The

composition properties defined in Q correspond to the condition in the where clause of the SQL statement. The two functors are defined very simply, where all objects and arrows with the same names are mapped. The only exception is the value of deptid in G, for which we explicitly state that the path deptid .q _ deptid in Q corresponds to the path deptid in A.
In Fig 3 we can see that category C, Q and A are representing the same data base instance. The equivalency was established with the help of the functor F and G and by the definition of Natural Transformation as well as functors are preserving the structure also.

Optimization and Natural Transformation

The idea of a "functor" is useful to model data structures. Products (x, y) ∈ X × Y relate to records that have two accessor functions: α: X × Y → X and β: X × Y → Y . Powerset represents the well-known datatype "set of Xs," i.e. {x, y, z}∈P(X). These functors can be combined to describe relations, such as (1, x),(2, y),(3, z). As an aside, the above example may be used to describe the datatype "list of Xs," i.e. [x, y, z]: L (X). Ls rather than sets are more intriguing since they are found natively in most programming languages.

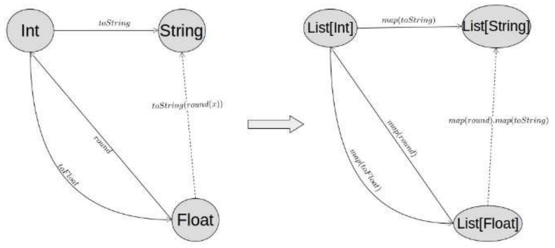
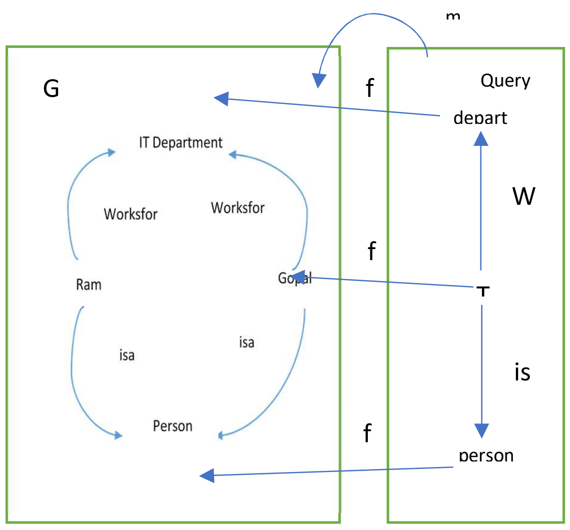


Fig 4 List Functor

This means functors can be composed to model more complex data structures. A directed unlabeled graph is represented by a set of edges and the functor G(N) = P(NXN), where N is a set of nodes and NXN edges. We can construct a graph morphism m(f) = P(fXf) with a function f: N-> N' that alters the nodes while keeping the structure of the graph (i.e. if (x, y) is an edge of g, then (f(x), f(y)) is an edge of m(g)). m(idN) is an identity morphism; morphisms can be combined . Graphs are essential in the LSD community and have numerous applications [1]. Querying a specific piece of information then entails locating a morphism from the graph representing a query to a graph database [2].
A query such as ”(T isa man) and (T worksfor IT department)” can be viewed as a labeled graph with two edges {isa : T → Person, worksfor : T → deparment}, where T denotes a variable as illustrated in Figure 36. By forgetting labels, the query is also represented by {(T, man),(T, IT deparment)} : G(N ∪ X). The result is then a set of sets of pairs {{(T, Ram)}, {(T, Gopal}} and the program finding the possible morphisms can be formalized by query : G(N’)×G(N) → P(P(T×N)) if N’ = N ∪X the union of constant nodes N plus variables X; P(T ×N) is here a shortcut for a mapping function f : T → N. Such a program has been already detailed in the literature with in particular [3] and [4].



Now, other representations of graphs are possible, e.g. G’ (N) = P(N × P(N)) that associates adjacent nodes to each node. The relation between the two functors can be represented by a natural transformation η : G’ (N) → G(N) with η(g’) = {(x, y)|(x, ys) ∈ g, y ∈ ys}. This transformation is invertible and the functors/datatypes are then said to be naturally isomorphic G’ (N) ∼=η G(N). If the two structures represent a ”same” information, the performance of a program depends on the structure selected.

As an example, a function/program to get the adjacent nodes, i.e. g(n) : G(N) → P(N), will have complexity O(n) where n is the number of edges when using G, and O(m) where m is the number of nodes when using G’ , and m ≤ n. So, g’ (n) : G’(N) → P(N) is ”faster” than g(n). The change from G to G’ can be viewed as an optimization technique called ”memorization” in the sense that G’ memorizes the result (i.e. adjacent node) for each input node and then eliminates extra computations [5]. The optimized version of the program will be obtained with g ‘ (n) = g(n) ◦ η −1 that can be simplified by using the definitions of g and η −1 (and is known as short-cut fusion optimization [6])

Conclusion

Its mathematical way of formulating equations is perfectly suited to our research. We apply Category Theory to numerous high-level data representations, such as tables, lists, maps, and graphs. Several Functors and the accompanying queries are proposed to move from one representation to the other. We illustrate three distinct representations as well as the various Functors. To target the native representation for specialized Databases, some particular isomorphisms are offered. The time it takes to load the initial data set using the given technique is commented on, and the degradation is measured if the initial dataset grows. The function finds extracts a large amount of data, and we measure the time again. We take some measures again because the amount appears to be significant. Finally, the dataset feeds the selected databases, and we measure the varying times for a query once more. A comparison of the different acquired times reveals that functional programming is often faster than traditional databases.

Reference

1. I. Robinson, J. Webber, and E. Eifrem, Graph Databases. O’Reilly Media Inc., 2013.

2. P. T. Wood, “Query languages for graph databases”, SIGMOD Rec., vol. 41, no. 1, pp. 50–60, 2012.
3. L. Thiry, M. Mahfoudh, and M. Hassenforder, “A functional inference system for the web”, International Journal of Web Applications, vol. 6, no. 1, pp. 1–13, 2014.
4. T. Segaran, C. Evans, J. Taylor, S. Toby, E. Colin, and T. Jamie, Programming the Semantic Web. O’Reilly Media, Inc., 2009.
5. C. Okasaki, Purely Functional Data Structures. New York, NY, USA: Cambridge University Press, 1998.
6. R. Bird and O. de Moor, Algebra of Programming, ser. Prentice-Hall international series in computer science. Prentice Hall, 1997.
7. David I. Spivak, Ryan Wisnesky. Relational Foundations for Functorial Data Migration. Proceedings of the 15th International Symposium on Database Programming Languages (DBPL 2015).
8. Patrick Schultz, David I. Spivak, Ryan Wisnesky. Functorial Data Migration: From Theory to Practice. NIST Interagency/Internal Report (NISTIR) (Publication ID: 919457, 2015)
9. Koutrika, G; Wisnesky, R; Hernandez, M; Krishnamurthy, R; Popa, L HIL: A High-Level Scripting Language for Entity Integration. Proceedings of the 16th International Conference on Extending Database Technology (EDBT 2013)
10. Bogdan Alexe, Douglas Burdick, Mauricio A. Hernandez, Georgia Koutrika, Rajasekar Krishnamurthy, Lucian Popa, Ioana R. Stanoi, Ryan Wisnesky. High-Level Rules for Integration and Analysis of Data: New Challenges. Festschrift celebrating Peter Buneman (PBF 2013).
11. Dessloch, S; Hernandez, M; Wisnesky, R; Radwan, A; Zhou, J Orchid: Integrating Schema Mapping and ETL. Proceedings of the 24th IEEE International Conference on Data Engineering (ICDE 2008).