DBKDA 2020

The Twelfth International Conference on Advances in Databases, Knowledge, and Data Applications

ISBN: 978-1-61208-790-0

September 27th – October 1st, 2020

DBKDA 2020 Editors

Malcolm Crowe, University of the West of Scotland, UK
Lisa Ehrlinger, Software Competence Center Hagenberg GmbH, Austria
Fritz Laux, Reutlingen University, Germany
Andreas Schmidt, Karlsruhe Institute of Technology, Germany

DBKDA 2020

Forward

The Twelfth International Conference on Advances in Databases, Knowledge, and Data Applications (DBKDA 2020) continued a series of events covering a large spectrum of topics related to advances in fundamentals on databases, evolution of relation between databases and other domains, data base technologies and content processing, as well as specifics in applications domains databases.

Advances in different technologies and domains related to databases triggered substantial improvements for content processing, information indexing, and data, process and knowledge mining. The push came from Web services, artificial intelligence, and agent technologies, as well as from the generalization of the XML adoption.

High-speed communications and computations, large storage capacities, and load-balancing for distributed databases access allow new approaches for content processing with incomplete patterns, advanced ranking algorithms and advanced indexing methods.

Evolution on e-business, ehealth and telemedicine, bioinformatics, finance and marketing, geographical positioning systems put pressure on database communities to push the 'de facto' methods to support new requirements in terms of scalability, privacy, performance, indexing, and heterogeneity of both content and technology.

We take here the opportunity to warmly thank all the members of the DBKDA 2020 technical program committee, as well as all the reviewers. The creation of such a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors who dedicated much of their time and effort to contribute to DBKDA 2020. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions. We also thank the members of the DBKDA 2020 organizing committee for their help in handling the logistics of this event.

DBKDA 2020 Chairs

DBKDA 2020 Steering Committee
Friedrich Laux, Reutlingen University, Germany
Lisa Ehrlinger, Johannes Kepler University Linz, Austria / Software Competence Center Hagenberg GmbH, Austria
Andreas Schmidt, Karlsruhe Institute of Technology / University of Applied Sciences, Germany
Peter Kieseberg, St. Pölten University of Applied Sciences, Austria
Erik Hoel, Esri, USA

DBKDA 2020 Publicity Chair
Joseyda Jaqueline More, Universitat Politecnica de Valencia, Spain
Marta Botella-Campos, Universitat Politecnica de Valencia, Spain

DBKDA 2020 Industry/Research Advisory Committee

Jerzy Grzymala-Busse, University of Kansas, USA
Filip Zavoral, Charles University Prague, Czech Republic
Konstantinos Kalpakis, University of Maryland Baltimore County, USA
Shin-ichi Ohnishi, Hokkai-Gakuen University, Japan
Thomas Triplet, Ciena inc. / Polytechnique Montreal, Canada
Stephanie Teufel, iimt - international institute of management in technology | University of Fribourg, Switzerland
Rajasekar Karthik, Oak Ridge National Laboratory, USA

DBKDA 2020
Committee

DBKDA 2020 Steering Committee
Friedrich Laux, Reutlingen University, Germany
Lisa Ehrlinger, Johannes Kepler University Linz, Austria / Software Competence Center Hagenberg GmbH, Austria
Andreas Schmidt, Karlsruhe Institute of Technology / University of Applied Sciences, Germany
Peter Kieseberg, St. Pölten University of Applied Sciences, Austria
Erik Hoel, Esri, USA

DBKDA 2020 Publicity Chair
Joseyda Jaqueline More, Universitat Politecnica de Valencia, Spain
Marta Botella-Campos, Universitat Politecnica de Valencia, Spain

DBKDA 2020 Industry/Research Advisory Committee
Jerzy Grzymala-Busse, University of Kansas, USA
Filip Zavoral, Charles University Prague, Czech Republic
Konstantinos Kalpakis, University of Maryland Baltimore County, USA
Shin-ichi Ohnishi, Hokkai-Gakuen University, Japan
Thomas Triplet, Ciena inc. / Polytechnique Montreal, Canada
Stephanie Teufel, iimt - international institute of management in technology | University of Fribourg, Switzerland
Rajasekar Karthik, Oak Ridge National Laboratory, USA

DBKDA 2020 Technical Program Committee
Zeyar Aung, Masdar Institute of Science and Technology, UAE
Gilbert Babin, HEC Montréal, Canada
Flavio Bertini, University of Bologna, Italy
Zouhaier Brahmia, University of Sfax, Tunisia
Martine Cadot, LORIA, Nancy, France
Ricardo Campos, Polytechnic Institute of Tomar, Portugal
Sanjay Chaudhary, AhmedabadUniversity, India
Yung Chang Chi, National Cheng Kung University, Taiwan
Monica De Martino, Istituto per la Matematica Applicata e Tecnologie Informatiche "Enrico Magenes" | Consiglio Nazionale delle Ricerche, Italy
Marianna Di Gregorio, University of Salerno, Italy
Anton Dignös, Free University of Bozen-Bolzano, Italy
Ivanna Dronyuk, Lviv Polytechnic National University, Ukraine
Cedric du Mouza, CNAM (Conservatoire National des Arts et Métiers), Paris, France
Lisa Ehrlinger, Johannes Kepler University Linz, Austria / Software Competence Center Hagenberg GmbH, Austria
Gledson Elias, Federal University of Paraíba (UFPB), Brazil
Barbara Gallina, Mälardalen University, Sweden
Ana González-Marcos, Universidad de La Rioja, Spain
Luca Grilli, University of Foggia, Italy
Robert Gwadera, Cardiff University, UK

Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission or reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article is does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

Table of Contents

A Graph Database Storage Engine for Provenance Graphs

Changhong Liu

School of Computer Science and Engineering,
University of Electronic Science and Technology,
Chengdu, China 610000
Email: 314979677@qq.com

Hancong Duan

School of Computer Science and Engineering,
University of Electronic Science and Technology,
Chengdu, China 610000
Email: duanhancong@uestc.edu.cn

***Abstract*—The rapid development of high-speed networks has created a massive amount of data. Storing and mining such data is of great research value. Knowledge graphs and graph databases have widely been studied and applied as an effective means to mine the associated data in the past few years. Provenance graphs provide powerful ways to observe the changes in a graph, especially in graph analysis. The update operation will produce massive provenance graphs from a given graph as time goes on. It is a challenge to store and query these massive provenance graphs efficiently. Meanwhile, the query performance itself must be guaranteed. To address this challenge, this paper presents a graph database storage engine called T-GDB (Temporal dimension - Graph Database). This system binds the topology of the graph to each vertex in the graph and rebuilds the graph in real-time when analyzing the graph. T-GDB can analyze the changes in a graph over time and can also access the provenance of the specified graph through the index tree. T-GDB can support these application scenarios such as the knowledge reasoning of knowledge graphs and the information mining for specified graphs. This paper describes the format of data storage, the index, and the implementation of this system. Finally, this paper compares the proposed graph database storage engine to several existing mainstream graph databases to verify the feasibility and efficiency of this design. Our experimental results demonstrate that the proposed graph database storage engine has better performance and more efficient graph analysis than existing methods.**

Keywords–Graph Database; Graph Analytics and Storage; Provenance Graphs.

I. INTRODUCTION

In the age of big data, big graph analysis has widely been studied in recent years because of its many applications in a wide variety of practical fields. Many algorithms of graph computing are NP-hard (non-deterministic polynomial-time hard) problems such as Graph Partition [1]. It is challenging to study how to store graph-structured data and reduce the computing latency for graph computing. As a research field of artificial intelligence, knowledge graphs [2] play an important role in intelligent data analysis. Knowledge graphs can be stored in Resource Description Framework (RDF) [3], XML (Extensible Markup Language) [4], or other formats. Property graphs [5] (see Figure 2) are also effective data models for applying knowledge mining in graph databases. Graph databases can efficiently query the properties attached to vertices and edges of the graph, while RDF is less effective at doing that. NoSQL (Not Only Structured Query Language) databases have several storage types: Key-Value like Redis Graph [6], Document like CouchDB [7], Column-oriented like Bigtable [8] and graph database like Neo4j [9]. Therefore, graph database is one type of NoSQL databases. However, existing graph databases still employ several storage formats. In this paper, the storage format of the storage engine is similar to Key-Value. The simple statement queries of graph databases do not care much about the memory usage. However, memory usage is vital for graph analysis because it always traverses the whole graph. To address this, Trinity [10] presented an optimized memory management for graph-structured models. Although graph databases have great advantages in dealing with relationships between data, graph indexing [11] is also necessary to speed up graph computing. This paper's contributions are as follows:

- Propose a unique tree-structured index for provenance graphs and an efficient graph storage model for graph traversal.
- Provide a storage engine architectural design. This system can read, write and analyze the graph-structured data conveniently and quickly.

The rest of this paper is structured as follows. This paper describes the background of the system and the related work in Section II. In Section III, this paper details the storage format of the system, both in memory and on the disk. In Section IV, this paper provides the architecture and implementation of the system. This paper discusses the performance of T-GDB and compares the results with other graph databases in Section V. This paper discusses the future work related to the research in Section VI.

II. BACKGROUND AND RELATED WORK

In many existing graph databases, the changes in a graph over time can not be queried. Graph databases usually deal with the relationships between data. However, many existing graph databases can not do anything about the relevant causality of data. For example, the process of knowledge reasoning will produce the relevant causality of data. Data provenance has widely been studied in the field of databases. Provenance graphs provide powerful ways to analyze the graph-structured data like Ariadne [12]. However, developers did not specially design effective storage for provenance graphs in graph databases. Knowledge mining must be considered in our system. Knowledge mining can mine the potential information of the graph-structured data and can also deduce new knowledge through knowledge mining algorithms. Cook et al. [13] present the details of many kinds of graph mining algorithms. In the research field of knowledge graphs, DBpedia [14] is the leader in knowledge storage. Freebase [15] is

 ISBN: 978-1-61208-790-0

a graph database for building human knowledge. However, these previous studies can not query provenance graphs. Many existing graph databases can also not query knowledge graphs or lack support for knowledge mining. There are some major graph databases, such as Neo4j, TigerGraph [16] and JanusGraph [17]. These major graph databases are not suitable for storing knowledge graphs or provenance graphs. Neo4j uses an orthogonal list to represent the graph-structured data. JanusGraph uses an adjacency matrix to represent the graph-structured data. However, T-GDB uses an array to represent the graph-structured data. Neo4j uses the ID of vertices or edges as the graph indexing. Neo4j reads the graph-structured data from disk through the graph indexing. JanusGraph uses the Key-Value to read the graph-structured data from disk. These existing graph databases can not compactly store the graph-structured data according to the characteristics of graph. The compact storage can help graph databases read the graph-structured data from disk sequentially. T-GDB provides a special design for compact storage. The final goal of T-GDB is to meet both OLAP (On-Line Analytical Processing) and OLTP (On-Line Transaction Processing) requirements.

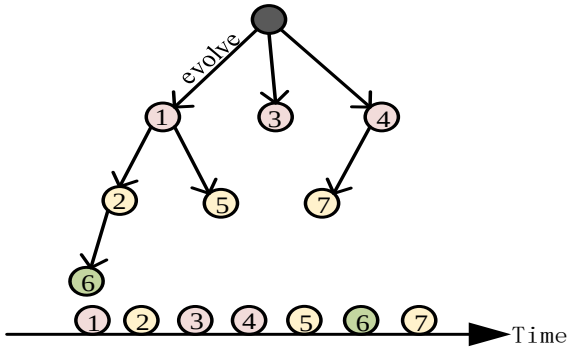


Figure 1. The logical relationship of provenance graphs.

Knowledge reasoning [18] is a key technology in knowledge graphs. Knowledge graphs deduce new knowledge over a given graph according to different rules. Knowledge reasoning may need to access the provenance of the specified graph and reason repeatedly. That (see Figure 1) is a good explanation for provenance graphs. Pugliese et al. [18] proposed a temporal RDF model. Lu et al. [19] proposed a temporal data storage based on TDSQL. Time is a key metadata to query the changes in data according to [18] and [19]. Leskovec et al. [20] describe the changes in graph over time. Knowledge reasoning can form the logical relationship of provenance graphs (shown in Figure 1) according to the above studies. Each node of the tree-structure represents an index file for the special graph. Time properties are also important for the graph databases to observe the subtle changes in a graph.

III. Data Model

The storage engine uses property graphs as the data model in this paper. Property graphs are directed graphs consisting of vertices, edges, and properties (see Figure 2). Labels and relationship types are particular properties of vertices and edges, respectively. Graph queries can filter out a lot of useless data according to labels and relationship types. The time is also a unique property in our storage engine. Because these particular properties always play an essential role in graph query, our storage engine stores them in different formats. This storage engine has mainly two parts: memory storage format and disk storage format. This paper will describe them in detail below.

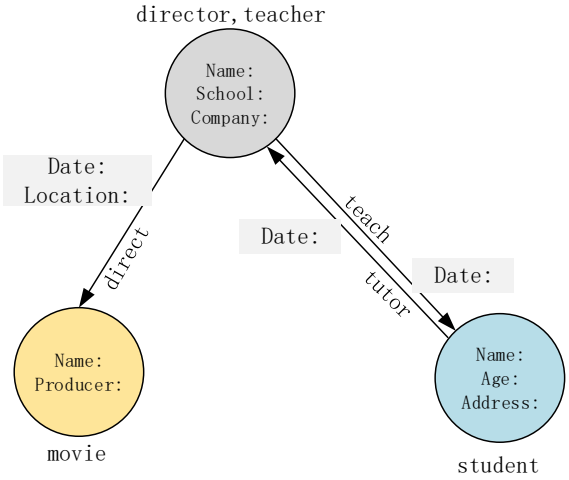


Figure 2. An example of property graphs.

A. Memory Storage Format

Many existing graph databases usually use adjacency lists or cross lists to store the graph-structured data. However, we use arrays to store the graph-structured data in this paper. Every vertex and edge of property graphs has a fixed-length byte in arrays. Each vertex accesses its neighbors through the array address. The time complexity is O(1) when this system traverses property graphs. The storage format is an excellent benefit for graph queries. Because all delete, update, and insert operations of our graph database are done in an append manner, this system has no restrictions on storing graph-structured data in arrays. The graph has three parts: vertices, edges, and topologies. The graph-structured data will be serialized from disk to memory when graph queries need to access the specified graph. However, simple graph queries can get data directly through graph indexing without rebuilding the graph.

TABLE I. The structure of the vertex.

type:	uint32	uint32	uint32	uint32
vertex array:	Pid+VertexId	TopoOffset	Flag+OEOffset	Time

The structure of the vertex is shown in Table I. All vertices of a graph are stored in a vertex array. Each vertex has a fixed-length byte in the vertex array. The Pid is short for partition id. Because a big graph will be divided into many subgraphs, the partition id is the id of one subgraph. The VertexId is a unique vertex number in one subgraph. The TopoOffset (Topology Offset) is the topology index of topology array. The Flag field is reserved for particular purposes. The OEOffset (OutEdge Offset) is an offset relative to TopoOffset. This system needs to distinguish incoming edges and outcoming edges by the OEOffset. The Time is a property of the vertex. This system can observe the subtle changes in a graph through the Time.

 ISBN: 978-1-61208-790-0

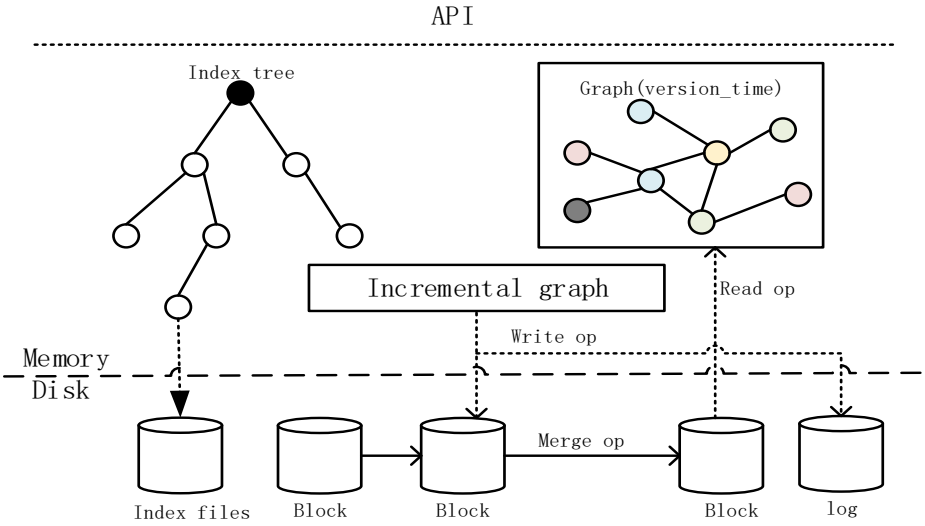


Figure 3. The framework of T-GDB.

TABLE II. THE STRUCTURE OF THE EDGE.

type:	uint32	uint32	uint32	uint32	uint32
edge array:	Pid+SrcId	Flag+DistId	EdgeId	RelType	Time

The structure of the edge is shown in Table II. All edges of a graph are stored in an edge array. Each edge has a fixed-length byte in the edge array. The SrcId and DistId are the id of source and destination of the edge, respectively. They are the vertex index of vertex array. This system will assign a unique EdgeId number to each edge. The EdgeId is the edge index of edge array. Relationship types and labels will be stored in a dictionary mode. The RelType is the dictionary number in edge array.

TABLE III. THE STRUCTURE OF THE TOPOLOGY.

type:	uint32	uint32	uint32	uint32	uint32
topo array:	VertexId	Flag	Label	OutEdgeId	InEdgeId

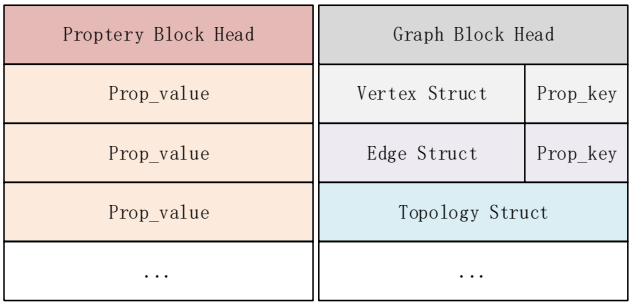


Figure 4. Left is the storage of the graph, and Right is the properties of the graph.

The structure of the topology is shown in Table III. The topo array is an array containing all topologies of a graph. The Label is the dictionary number of labels. The OutEdgeId is the id of outcoming edges. The InEdgeId is the id of incoming edges. The topology is attached to the vertex. The state of topologies will be changed depending on the state of vertices and edges. Meanwhile, this system can finish graph queries within a limited time through the topology structure.

B. Disk Storage Format

The graph-structured data is stored in 4G-sized file blocks (see Figure 4). The size of file blocks is 4G because of the uint32 type. Meanwhile, our system will merge data on disk periodically to speed up graph queries. It is also beneficial to implement multiple replicas with 4G-sized file blocks. The latest file block holds the newest data because new data is appended to the existing one. The period information of file

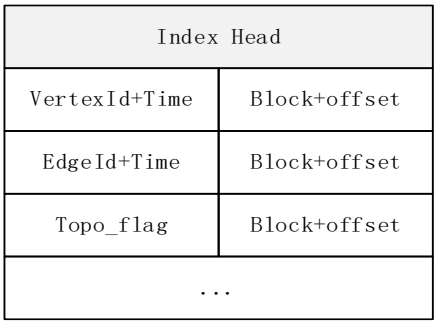


Figure 5. The index files.

blocks can help this system speed up searching provenance graphs. This system can directly access the properties of vertices and edges through the prop_key.

C. Index File

In this system, there is a unique index for provenance graphs. There are parallel meanings and chronological order between provenance graphs, according to the Figure 1. Our storage engine stores the relationship between provenance graphs in a multi-fork tree. Each node of the multi-fork tree is an index file. An index file may be full or incremental

index for a graph. The head of index file has some important basic informations. This system can directly access the graph-structured data through index files (see Figure 5). Index files play a crucial role in building the graph or accessing partial graph. Index files are compressed to reduce storage overhead according to the contents of index files.

IV. IMPLEMENTATION

In this section, this paper details our storage engine implementation written in C++ (see Figure 3). The core design of our system reflects the features of provenance graphs. So far, we have only implemented a stand-alone system. In future work, we will implement a distributed graph database with the ability to handle large-scale storage and graph computing. The architecture of our storage engine is straightforward. The architecture has three major parts: reading, writing and merging the graph-structured data. The components of this system are described in detail below.

1) Index Tree: The index tree is a critical component in our storage engine. Each node of the index tree includes an index file and the basic informations of provenance graphs. The path from the root node to the leaf node in the index tree represents the changes of provenance graphs. There are two storage forms for index files. One is the incremental index based on the parent index file. The other is the full index for a graph. The form of index files depends on the changes of provenance graphs. It takes a little time to read index files because of the serialization of index files. The index tree is beneficial for this system to observe the changes of provenance graphs base on the timeline.

2) Updating and Building Graph: Although the graph-structured data is updated by appending data, there is still a memory buffer for a graph named Incremental graph. The Incremental graph sorts the graph-structured data according to the time in memory and puts the graph-structured data on disk. The Incremental graph can store the same provenance graphs together. It can reduce the reading time by reading data in micro blocks. Meanwhile, the update operation must be logged to ensure that the data can be recovered in the event of a system crash.

Simple statement queries typically access a part of the graph. This system can finish simple statement queries by the index of vertices or edges. This system reads the particular provenance graphs in micro blocks according to index files. The size of micro blocks depends on the distribution information of graph-structured data on disk. This system can batch load the graph when it needs the whole graph. Our storage engine is also very efficient for graph computing.

3) Merge Block: There will be hot and cold data because of the graph changes based on the timeline. The fragmented data of a graph is distributed across many file blocks. Therefore, this system will regularly merge the data of a particular graph on disk. The merge operation does not affect the previous provenance graphs. At the same time, this system removes the unused graph-structured data to increase disk utilization. The merge operation is very effective for reading data.

V. PERFORMANCE EVALUATION

In the section, this paper presents experiments to demonstrate the performance of the proposed system. These experiments were based on a machine with Intel(R) Xeon(R) CPU e5-2603@1.80GHz, ubuntu 16.04.10 server, 96GB RAM, and 300G SSD (DELL PERC H310 2.12). Because this system is only a stand-alone version now in this paper, all of the following experiments were tested on a single-core CPU to achieve fairness.

The datasets having Graph500 [21] and com-Orkut [22] for experiments are from the website of public datasets (see Table IV). This paper performed the experiments according to the benchmark of TigerGraph [23]. Neo4j, TigerGraph, and JanusGraph were compared in the following experiments. The version of Neo4j is community-3.4.17. The version of TigerGraph is 2.5.0-developer. The version of JanusGraph is 0.2.1-hadoop2. This paper evaluates the performance of each system according to three query types:

TABLE IV. THE INFORMATION OF DATASETS

data:	*vertices*	*edges*	*Description*
Graph500	2396019	67108864	Synthetic Kronecker Graph
com-Orkut	3072441	117185083	Orkut online social network

A. The common query in graph database

The most common queries are the one-hop traversal of the graph in graph databases. It means that the one-hop traversal operation is executed from a source vertex to destination vertex through the edge. Then queries can access the properties of vertices or edges during one-hop traversal. The other common query is the three-hop traversal of the graph. However, it puts more pressure on the system.

This paper made ten thousand initial vertices to Graph500 and five hundred thousand initial vertices to com-Orkut in one-hop traversal, respectively. Figure 6(a) and Figure 6(b) are the result of a one-hop traversal query for Graph500 and com-Orkut, respectively. Because the three-hop traversal can almost traverse the whole graph, this experiment only made ten initial vertices to Graph500 and com-Orkut in three-hop traversal. Figure 7(a) and Figure 7(b) are the result of a three-hop traversal query for Graph500 and com-Orkut, respectively. It can seen that our system has an absolute advantage in the one-hop query of Graph500 and com-Orkut from Figure 6. Because our system does not have a cache yet, a vertex or edge will be reread from disk each time. TigerGraph has a built-in memory component that benefits from its data compression technology to reduce the overhead of disk. Therefore, our system is a little bit slower than TigerGraph in the three-hop traversal query. However, our system still has more advantages than the comparative databases.

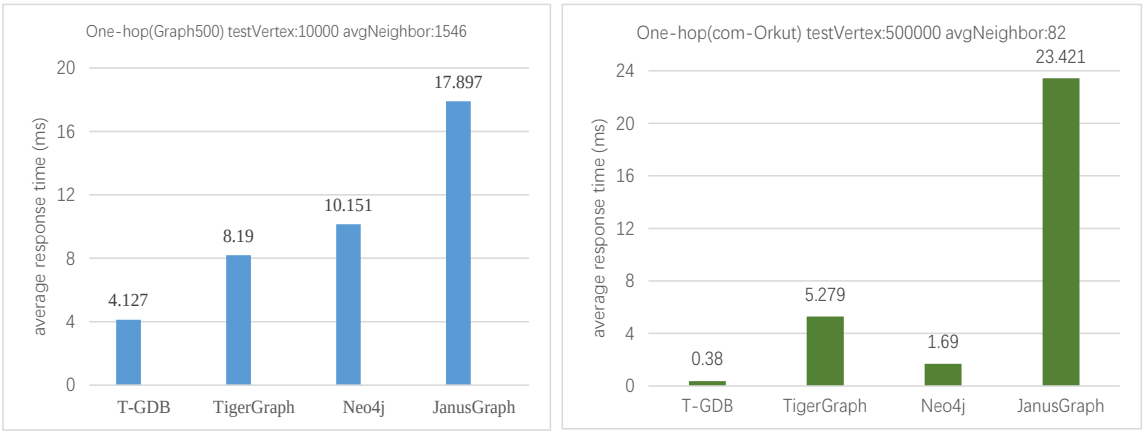
B. The graph analysis

There are many complex queries, such as PageRank, SSSP (Single Source Shortest Path), WCC (Weighted Community Cluster). This experiment chose the classic PageRank [24] algorithm. Figure 8 and Figure 9 are the results of the PageRank query for Graph500 and com-Orkut, respectively. It can see from Figure 8 and Figure 9 that our system still has a great advantage in graph computing.

C. The query of provenance graphs

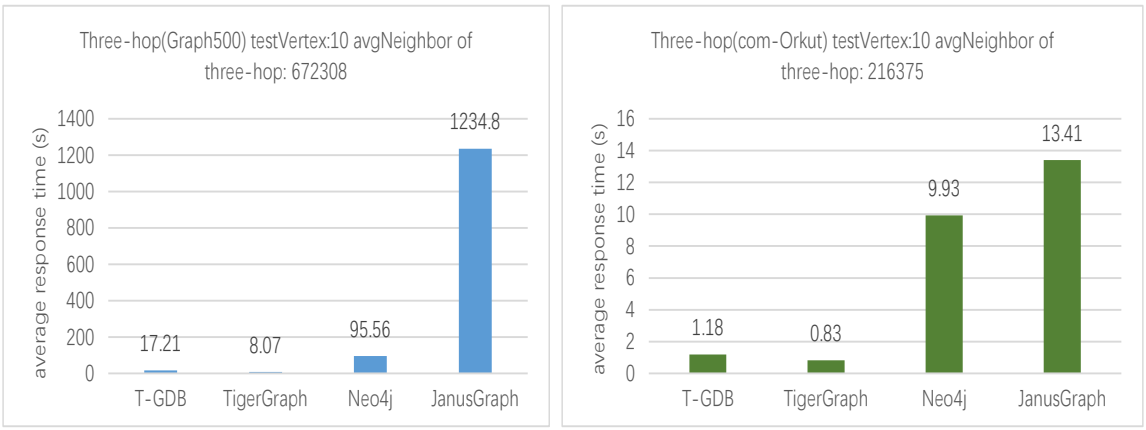
Our system has better performance in graph queries and graph analysis from the above experimental results. Different knowledge graphs can be deduced according to different rules

 ISBN: 978-1-61208-790-0



(a) (b)

Figure 6. (a): The average response time of one-hop query for Graph500. (b): The average response time of one-hop query for com-Orkut.



(a) (b)

Figure 7. (a): The average response time of three-hop query for Graph500. (b): The average response time of three-hop query for com-Orkut.

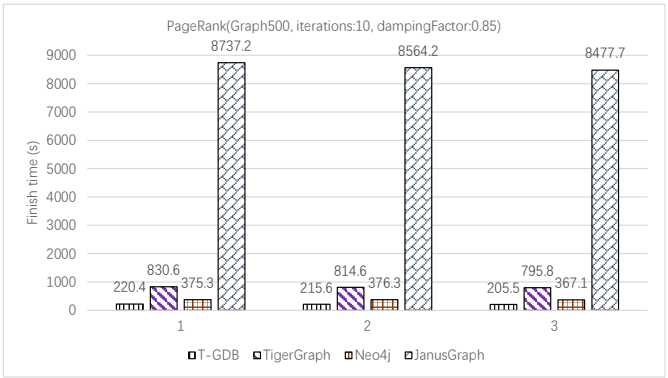


Figure 8. The finish time of PageRank having 10 iterations in graph500, Test three times.

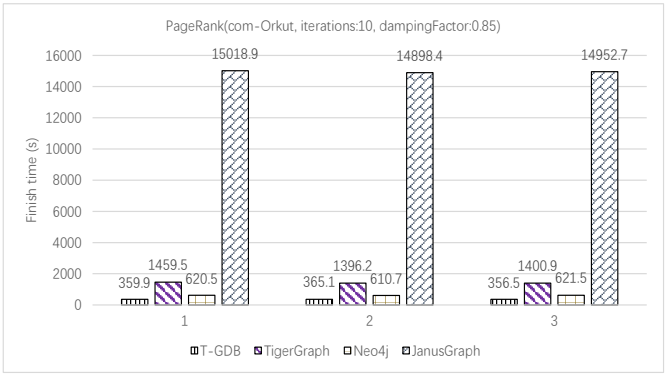


Figure 9. The finish time of PageRank having 10 iterations in com-orkut, Test three times.

in knowledge reasoning. The subtle changes of the graph can be observed through time properties. Because other graph databases do not support this kind of queries, the paper only does this queries experiment on our system. The experiment reads different provenance graphs from massive provenance graphs. For example, the Gaph500 produces many provenance graphs over a period of time (see Figure 1). This system randomly updates the time properties of vertices or edges

 ISBN: 978-1-61208-790-0

without changing the size of the Graph500.

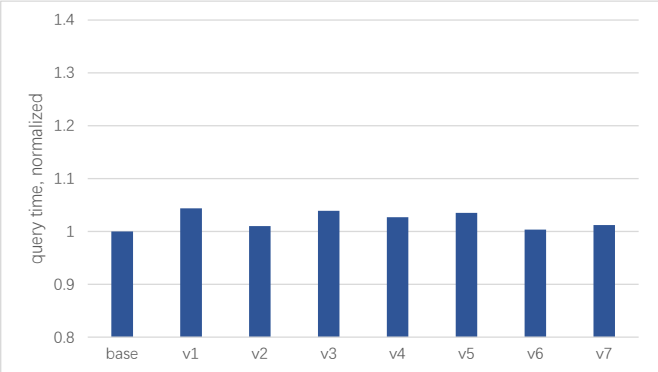


Figure 10. The cost of getting the different provenance graph.

The result of the query is shown in Figure 10. The costs are almost the same when this system reads different provenance graphs. The performance of our system does not change with the amount of data and the length of time, only with the size of the graph. Meanwhile, this system can detect partial changes of the graph through the time properties. This experiment demonstrates that the storage engine is useful for storing provenance graphs in this paper.

VI. Conclusion and Future Work

In this work, this paper proposed T-GDB, a high-performance graph database storage engine for provenance graphs. This system has a unique design to store provenance graphs efficiently without affecting the performance of graph queries and graph computing. We presented the index tree to apply the function of the provenance graphs. In this system, both index and data are stored in an append mode. The append mode is effective to observe the changes in a graph over time. Meanwhile, time plays a critical role to observe subtle changes in a graph. Although our system does not fully support the applying function of the time-series databases, it is a key to support the graph query that having time properties. Another critical point is that our system can support writing effectively because of updating the data in an appended mode.

In future work, we will focus on implementing a distributed graph database. We will ensure the data fault tolerance and the consistency of the distributed graph database. Meanwhile, we will support the application needs of artificial intelligence as much as possible.

Acknowledgment

Firstly, the authors would like to express their gratitude for the support from colleagues. Finally, they would also like to thank the benchmark of the TigerGraph.

References

[1] T. Ayall, H. Duan, and C. Liu, “Edge property based stream order reduce the performance of stream edge graph partition,” Journal of Physics: Conference Series, vol. 1395, 2019, p. 012010.

[2] L. Ehrlinger and W. Wolfram, “Towards a definition of knowledge graphs,” in Joint Proceedings of the Posters and Demos Track of 12th International Conference on Semantic Systems, 2016.

[3] F. Manola, E. Miller, and B. McBride, “Rdf primer,” W3C recommendation, vol. 10, no. 1-107, 2004, p. 6.

[4] C. C. Aggarwal, N. Ta, J. Wang, J. Feng, and M. Zaki, “Xproj: a framework for projected structural clustering of xml documents,” in Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2007, pp. 46–55.

[5] M. A. Rodriguez and P. Neubauer, “Constructions from dots and lines,” Bulletin of the American Society for Information Science and Technology, vol. 36, no. 6, 2010, pp. 35–41.

[6] “Redis Graph.” URL: https://github.com/tblobaum/redis-graph/ [accessed: 2020-02-08].

[7] J. Mondal and A. Deshpande, “Managing large dynamic graphs efficiently,” in Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. ACM, 2012, pp. 145–156.

[8] F. Chang et al., “Bigtable: A distributed storage system for structured data,” Acm Transactions on Computer Systems, vol. 26, no. 2, pp. p.1–26.

[9] I. Robinson, J. Webber, and E. Eifrem, Graph databases. ” O’Reilly Media, Inc.”, 2013.

[10] B. Shao, H. Wang, and Y. Li, “Trinity: A distributed graph engine on a memory cloud,” in Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM, 2013, pp. 505–516.

[11] X. Yan, P. S. Yu, and J. Han, “Graph indexing: a frequent structure-based approach,” in Proceedings of the 2004 ACM SIGMOD international conference on Management of data. ACM, 2004, pp. 335–346.

[12] V. Papavasileiou, K. Yocum, and A. Deutsch, “Ariadne: Online provenance for big graph analytics,” in Proceedings of the 2019 International Conference on Management of Data, 2019, pp. 521–536.

[13] D. J. Cook and L. B. Holder, Mining graph data. John Wiley & Sons, 2006.

[14] S. Auer et al., “Dbpedia: A nucleus for a web of open data.” in Semantic Web, International Semantic Web Conference, Asian Semantic Web Conference, Iswc + Aswc, Busan, Korea, November, 2007.

[15] K. Bollacker, P. Tufts, T. Pierce, and R. Cook, “A platform for scalable, collaborative, structured information integration,” in Intl. Workshop on Information Integration on the Web (IIWeb07), 2007, pp. 22–27.

[16] A. Deutsch, Y. Xu, M. Wu, and V. Lee, “Tigergraph: A native mpp graph database,” arXiv preprint arXiv:1901.08248, 2019.

[17] “Compose for JanusGraph,” URL: https://www.ibm.com/cloud/compose/janusgraph/ [accessed: 2020-02-08].

[18] A. Pugliese, O. Udrea, and V. Subrahmanian, “Scaling rdf with time,” in Proceedings of the 17th international conference on World Wide Web. ACM, 2008, pp. 605–614.

[19] W. Lu et al., “A lightweight and efficient temporal database management system in tdsql,” Proceedings of the VLDB Endowment, vol. 12, no. 12, 2019, pp. 2035–2046.

[20] J. Leskovec, J. Kleinberg, and C. Faloutsos, “Graphs over time: densification laws, shrinking diameters and possible explanations,” in Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. ACM, 2005, pp. 177–187.

[21] “Graph500 Large Network Dataset Collection.” URL: https://graph500.org/ [accessed: 2020-02-08].

[22] “Stanford Large Network Dataset Collection.” URL: http://snap.stanford.edu/data/ [accessed: 2020-02-08].

[23] “Benchmark for TigerGraph.” 2018, URL: https://www.tigergraph.com/benchmark/ [accessed: 2020-02-08].

[24] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web.” Stanford InfoLab, Tech. Rep., 1999.

 ISBN: 978-1-61208-790-0

Solving a Combinatorics Challenge by Exploiting Computational Techniques Available on Relational Databases

Wei Hu
Software Engineering, Fairfield University
Fairfield, Connecticut USA
e-mail: wei.hu@student.fairfield.edu

Mirco Speretta
Gateway Community College
New Haven, Connecticut USA
e-mail: msperetta@gwcc.commnet.edu

***Abstract*—Experimental studies are based on data that, sometimes, needs to be manually created. Moreover, the data is handled in relational databases to exploit their capabilities of manipulating (i.e., sorting, combining, and inserting) data. In this study, we show how this approach was successful in solving a combinatorics challenge to create a data set used in a separate research study that involves all the possible card combinations of the SET game®. The data required for the study was very extensive. The exact number was unknown, as this is an open combinatorics question, but the estimate was in the order of hundreds of millions. We solved this challenge by using a relational database (i.e., MySQL) as a computational tool to generate the data set. Advanced SQL scripts, based on cross joins, were applied to generate all the data. Table partitioning was also applied to improve the database performance of tables whose number of records exceeded the size capability of the database table. The data set created from this project was then used to support a Web based user interface that collects data to be used in a separate research study based on the SET® game.**

Keywords-MySQL; partitioning; computation; cross join.

I. INTRODUCTION

SET game® [1] is a popular card game created by Marsha Jean Falco in 1974. She is also the founder of Set Enterprises, Inc., the company that published the game in 1981. In this game, 12 cards (i.e., a hand), randomly selected from a deck of 81 cards, are placed in front of the players. The winner of a hand is the first player that identifies a group of three cards that makes a *SET*. There are four types of *SETs* that a player can identify.

Two professors from the department of mathematics, at Fairfield University, were responsible for a Math club in a middle school of the town. They incorporated the SET game into the sessions with the students. Noticing the selection of specific SETs by the students, the professors wanted to investigate further this behavior with a research study whose main goal was to explore whether the personal information of the player, such as gender, age, or academic interests, can influence the types of *SET* identified. The study is based on a statistical analysis of data collected from anonymous users that are playing the game using a Web based interface. To avoid any statistical bias in the study, each hand must include one (and only one) instance of each of the four types of *SET*. To give a rough estimate of the amount of data to be processed, first we needed to look at the total of possible combinations: given 81 cards (i.e., the deck) there are 7.07×10^{13} possible ways to choose a hand (i.e., 12 cards). Out of this number of combinations, we had to identify and remove all the hands that did not satisfy the requirement of the statistical design. This requirement presented the following two main challenges. The first challenge is the combinatorics challenge, which can be described as follows: it is not known how to count the total number of combinations of hands that comply with the requirements mentioned above. This is still an open question in the mathematical community. The second challenge is the technical challenge to guarantee efficiency, which can be described as follows: the number of combinations is too high and it would take too much time to select the cards that form a hand in real time; lots of time would be wasted generating combinations that do not comply with the requirement. Because of the two challenges explained above, the users of the Web interface would not be able to play the game properly.

The solution to both challenges was to pre-generate four types of *SET* to compose all the possible hand combinations and store them into a database table. This approach would allow to show a randomly picked hand of cards in real time.

In this paper, we describe the process of generating and storing the data set using the MySQL® [2] relational database. More specifically, in Section II, we list the software used in the study. In Section III, we provide the context to this study by describing specific features of the SET game. Section IV outlines the details of the methodology, along with the information about the final data generated. We conclude our work in Section V.

II. BACKGROUND

MySQL [2] is a popular database that supports SQL along with transactions. The idea to use a relational database to manipulate data in our study comes from various research ideas, especially in the context of testing data [7].

We implemented our database on a MySQL 5.7.22 server, to store pre-generated hand data set. The server we used was equipped with 16 CPU (2.4 GHz) and 64 GB of RAM, running Linux Ubuntu (version 18.04.4 LTS). In our study, the largest amount of computational effort is needed to fulfill the following tasks: 1) joining tables to get the maximum number of possible card combinations; 2) validating that no more than one occurrence of a card appears in one hand; 3) validating that each hand card combination included only one instance of the four types of *SET*. Cross joins, comparisons between data tuples and row

 ISBN: 978-1-61208-790-0

by row computations are all very time and resource consuming due to the big size of data. In this study, we show that all these tasks can be carried out using a relational database in a simple and efficient way.

III. THE SET GAME

The SET game is a popular card game that has been widely disseminated by online media such as the New York Times. It has been used in mathematics learning by several educational institutions at different school levels [3]-[6].

A. Cards of the SET game

The game SET has a rich mathematical structure based on combinatorics principles. An example of cards is shown in Figure 1.

Figure 1. Typical cards of the SET game.

Cards have four attributes: number, shading, color, and shape. Each attribute has three features. The complete list is given in Table I.

TABLE I. SET CARD ATTRIBUTES AND THEIR VARIATIONS.

Attribute	Feature
Number	{One, Two, Three}
Shading	{Solid, Striped, Open}
Color	{Red, Green, Purple}
Shape	{Oval, Squiggle, Diamond}

The deck of the SET game has eighty-one cards, one for each possible combination of attributes.

B. Rules of the SET game

Three cards are called a *SET* if, with respect to each of the four attributes, the cards are either all the same or all different. The goal of the game is to find collections of three cards satisfying this rule. For example, the three cards in Figure 2 compose a *SET* because all cards have different shapes, different colors, and different shading, and each card has the same number of shapes (three).

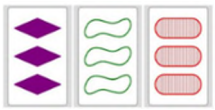


Figure 2. Typical *SET*.

C. Four types of SETs

Case 1: One attribute has different features; three attributes have the same features

a) different: shape; same: shade, color, number
b) different: shade; same: shape, color, number
c) different: color; same: shape, shade, number
d) different: number; same: shape, shade, color

Case 2: Two attributes have different features; two attributes have the same features

a) different: shape, shade; same: color, number
b) different: shape, color; same: shade, number
c) different: shape, number; same: shade, color
d) different: shade, color; same: shape, number
e) different: shade, number; same: shape, color
f) different: color, number; same: shape, shade

Case 3: Three attributes have different features; one attribute has the same feature

a) different: shape, shade, color; same: number
b) different: shape, shade, number; same: color
c) different: shape, color, number; same: shade
e) different: shade, color, number; same: shape

Case 4: All four attributes have different features

a) different: shape, shade, color, number

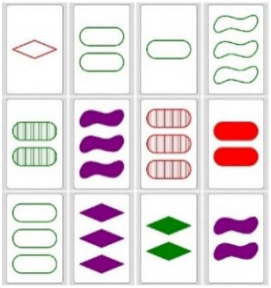


Figure 3. Typical hand of the SET game.

D. Hands of the SET game

To play the game, twelve cards, called a hand as shown in Figure 3, are dealt face up in front of players. Players search for *SETs*. After all *SETs* in the hand are found, the hand is refreshed and another twelve random cards are dealt out of the deck.

IV. METHODOLOGY

As the question on how to count the number of hand combinations containing exactly four *SET*s, one for each type, remains open, we could not know the exact number of records we were supposed to generate. The only solution to this problem was to work on an efficient algorithm to generate all the possible combinations.

Our goal is to find all possible hand card combinations that satisfy the requirements of the experimental study: every hand includes exactly four *SET*s, one for each type. In a typical play, 12 cards are dealt randomly out of the 81 cards deck. The total number of hand combinations to consider is about 7.07×10^{13}. This number of combinations is too big to be handled practically. For this reason, we had to apply some heuristic to reduce it to a number that was computationally feasible. If we start from the four types of *SETs* and we consider them as the basic components that form a hand, the biggest number of hand combinations to

screen is no more than number of Type 1 *SET* × number of Type 2 *SET* × number of Type 3 *SET* × number of Type 4 *SET* = *432 × 324 × 108 × 216 = 3,265,173,504 = 3.265*×10^9. The number of each type of *SET* is shown in Table III. All sets for each type can be stored in a table and all four tables can be merged using cross joins. In this way, the four types of *SETs* are automatically combined to form all the possible hands. From this number, we need to remove the combinations that do not satisfy the requirements:

1. each card occurrence is unique in one specific hand.
2. no extra new *SET*s are formed other than the four built-in *SET*s.

We used the database server MySQL (version 5.7.22) to store and manipulate all the card combinations. The data flow from the generation of four types of *SETs* through the generation of the final hand combinations is illustrated in the following three steps.

Step 1: We implemented a Java program to generate the four basic tables storing the four types of *SET* (Figure 4).

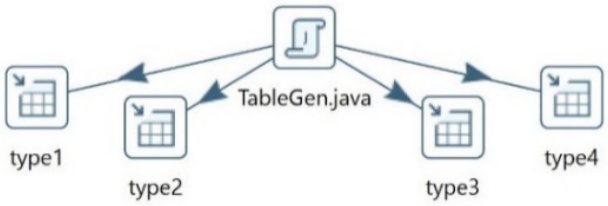


Figure 4. Java program generating the four basic tables.

Step 2: We implemented an SQL script to combine the four attributes of each card into a 4-digit number. This step also includes the generation of a new group of the four basic tables storing all the 4-digit numbers (Figure 5).

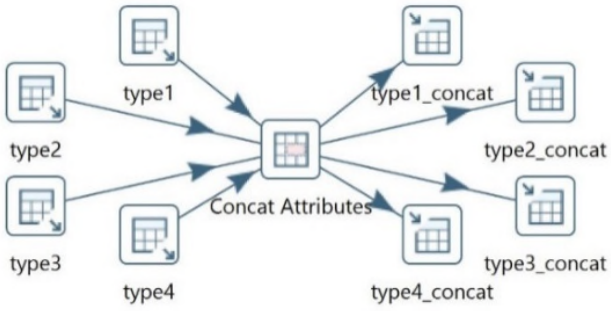


Figure 5. SQL script combining basic tables.

Step 3: We implemented an SQL script to cross join the four types of *SETs*. Then, we removed the records where any card occurred more than once, along with the records where the new *SET*s are formed (Figure 6).

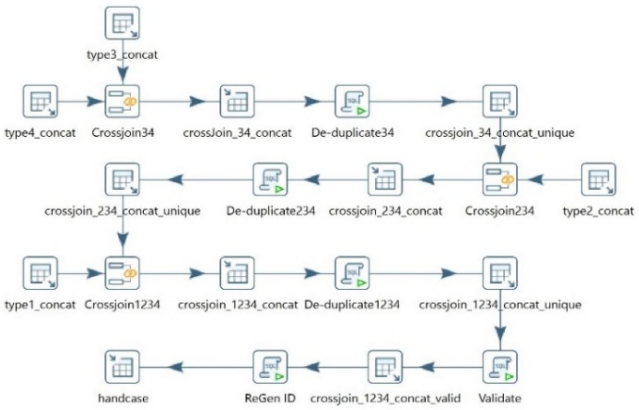


Figure 6. SQL script generating more SET hands and removing duplications.

In the remaining part of this section, we will provide more details about the work involved in carrying out the above steps.

A. Card Value Definition

We numbered all the cards using the following representation. A four-digit number is assigned to each card based on its specific attributes. From left to right, each digit refers to one attribute. Each digit can be either 0, 1, or 2. Each value represents one variable of an attribute (shown in Table II.) Using this representation, each card can assume a value in the range [0000] - [2222].

TABLE II. CARD DEFINITION.

Attribute / Variable	Number	Shading	Color	Shape
	Position (from left to right)			
0	One	Open	Red	Diamond
1	Two	Striped	Green	Oval
2	Three	Solid	Purple	Squiggle

One example is shown in Figure 7.

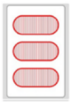


Figure 7. Visual representation of the card [2101].

This card value is [2101]: three, striped, red, and ovals. Another example is shown in Figure 8.

Figure 8. Visual representation of the card [0010].

This card value is [0010]: one, open, green, and diamond.

B. SET Type Definition

We define four types of *SET* as Type 1, Type 2, Type 3 and Type 4. The details of the definition are described below.

1) Type 1 *SET*: Only one attribute is the same, the other three attributes are different. The number of this type of *SET* is 432.

2) Type 2 *SET*: Two attributes are the same and the remaining two attributes are different. The number of this type of *SET* is 324.

3) Type 3 *SET*: Three attributes are the same, only one attribute is different. The number of this type of *SET* is 108.

4) Type 4 *SET*: All attributes are different. The number of this type of *SET* is 216.

C. Four Basic Tables Creation

We created four basic tables, one table for each type of *SET*. Potential card combinations of hands are created using these four basic tables (Table III).

TABLE III. FOUR BASIC TABLES DEFINITION.

Table No.	Table Name	*SET* Type	Number of *SETs*
1	type1	Type 1	432
2	type2	Type 2	324
3	type3	Type 3	108
4	type4	Type 4	216

Every *SET* has three cards, namely Card1, Card2 and Card3. Each card has four attributes namely A1, A2, A3 and A4. We run a Java program to generate data for the four basic tables type1, type2, type3 and type4. We create one attribute of each card at one time. The structure of table is shown in Figure 9 (cNaM – card N attribute M, N = 1, 2, 3 M = 1, 2, 3, 4).

SetId	c1a1	c1a2	c1a3	c1a4	c2a1	c2a2	c2a3	c2a4	c3a1	c3a2	c3a3	c3a4
1	0	0	0	0	0	1	1	1	0	2	2	2
2	0	0	0	0	0	1	1	2	0	2	2	1
3	0	0	0	1	0	1	1	2	0	2	2	0
4	0	0	0	1	0	1	1	0	0	2	2	2
5	0	0	0	2	0	1	1	0	0	2	2	1
6	0	0	0	2	0	1	1	1	0	2	2	0
7	0	0	0	0	0	1	2	1	0	2	1	2
8	0	0	0	0	0	1	2	2	0	2	1	1

Figure 9. Basic SET table (type1-type4).

We then create and run an SQL script to validate the correctness of the data stored in these four tables.

In order to facilitate the calculation and improve the efficiency of the database, we combined the four attributes of each card into a one 4-digit number. The output of this process was to create the four new tables type1_concat, type2_concat, type3_concat, and tyep4_concat. They store the four types of *SETs* represented by the four-digit attribute value of cards. See a sample of these data in Figure 10.

Every *SET* is composed by three 4-digit numbers, each 4-digit number representing one card. For example: [0000,0111,0222] is a type 1 *SET*. The value of the first card is [0000] and refers to “one open red diamond".

Eight tables were created after validating and concatenating the data. For the remainder of the process, only the four tables with concatenated attributes were used.

SetId	card1	card2	card3
1	0	111	222
2	0	112	221
3	1	112	220
4	1	110	222
5	2	110	221
6	2	111	220
7	0	121	212
8	0	122	211

Figure 10. Basic SET table with concatenated attributes (type1_concat-type4_concat).

D. Cross join of the Four basic Tables and Deduplication

In order to work with the smallest possible amount of data, at any given time, we started the merging process using the two tables with the fewest number of records. They are represented by the tables type3_concat and type4_concat. Their joined table was then cross joined with type2_concat. As the last step, we cross joined this newly created table with the biggest table, type1_concat.

Due to the exponential increase of data size, after each cross join, we used SQL queries to validate records and filter out those records where the same cards were used more than once. In this study, this deduplicate validation is different from the typical deduplicate operation, which is responsible to remove redundant records from the table. We then performed another cross-join, the goal of which was to minimize the number of records of each cross join as much as possible. Table IV describes the number of records, the table size and the time spent to create each specific table. The tables whose names end by ‘_unique’ refer to the tables created after the deduplication operation. In table IV we can notice the reduction in terms of both number of records and table size. The last table, ‘crossjoin_1234_concat_valid’, stored the valid hand combinations that were used in the research study.

TABLE IV. CROSS JOIN TABLES.

Table name	Number of records	Table size (MB)	Time elapsed
crossjoin_34_concat	23328	2	NS
crossjoin_34_concat_unique	20736	2	NS
crossjoin_234_concat	6718464	277	NS
crossjoin_234_concat_unique	5351040	216	NS
crossjoin_1234_concat	2311649280	118410	7h, 5m
crossjoin_1234_concat_valid	269635392	13121	27h, 40m

E. Hand Combination Validation

Once the four tables type1_concat, type2_concat, type3_concat, and type4_concat were cross joined into one table, the number of records grew fast. Although, upon the completion of each cross join, we removed the records where the instance of specific cards appeared more than once, still, a large amount of calculation had to be carried out to validate the *SET*s that were added. This was necessary because each group of three cards, one from each type of *SET*, could have made up a new *SET*. We used SQL queries again to validate which hand combinations contain exactly four *SET*s, one for

each type excluding the new *SET*s added after the cross join operations. To improve the performance of the database and accelerate the speed of the calculation, we used the partitioning technique, built-in in MySQL, to organize each cross joined table into (~20) partitions.

TABLE V. POTENTIAL NEWLY FORMED *SET* COMBINATIONS.

Group	Potential *SETs*		
Type 3 + Type 4 + Type 2	card1+card4+card7	card1+card4+card8	card1+card4+card9
	card1+card5+card7	card1+card5+card8	card1+card5+card9
	card1+card6+card7	card1+card6+card8	card1+card6+card9
	card2+card4+card7	card2+card4+card8	card2+card4+card9
	card2+card5+card7	card2+card5+card8	card2+card5+card9
	card2+card6+card7	card2+card6+card8	card2+card6+card9
	card3+card4+card7	card3+card4+card8	card3+card4+card9
	card3+card5+card7	card3+card5+card8	card3+card5+card9
	card3+card6+card7	card3+card6+card8	card3+card6+card9
Type 3 + Type 4 + Type 1	card1+card4+card10	card1+card4+card11	card1+card4+card12
	card1+card5+card10	card1+card5+card11	card1+card5+card12
	card1+card6+card10	card1+card6+card11	card1+card6+card12
	card2+card4+card10	card2+card4+card11	card2+card4+card12
	card2+card5+card10	card2+card5+card11	card2+card5+card12
	card2+card6+card10	card2+card6+card11	card2+card6+card12
	card3+card4+card10	card3+card4+card11	card3+card4+card12
	card3+card5+card10	card3+card5+card11	card3+card5+card12
	card3+card6+card10	card3+card6+card11	card3+card6+card12
Type 3 + Type 2 + Type 1	card1+card7+card10	card1+card7+card11	card1+card7+card12
	card1+card8+card10	card1+card8+card11	card1+card8+card12
	card1+card9+card10	card1+card9+card11	card1+card9+card12
	card2+card7+card10	card2+card7+card11	card2+card7+card12
	card2+card8+card10	card2+card8+card11	card2+card8+card12
	card2+card9+card10	card2+card9+card11	card2+card9+card12
	card3+card7+card10	card3+card7+card11	card3+card7+card12
	card3+card8+card10	card3+card8+card11	card3+card8+card12
	card3+card9+card10	card3+card9+card11	card3+card9+card12
Type 4 + Type 2 + Type 1	card4+card7+card10	card4+card7+card11	card4+card7+card12
	card4+card8+card10	card4+card8+card11	card4+card8+card12
	card4+card9+card10	card4+card9+card11	card4+card9+card12
	card5+card7+card10	card5+card7+card11	card5+card7+card12
	card5+card8+card10	card5+card8+card11	card5+card8+card12
	card5+card9+card10	card5+card9+card11	card5+card9+card12
	card6+card7+card10	card6+card7+card11	card6+card7+card12
	card6+card8+card10	card6+card8+card11	card6+card8+card12
	card6+card9+card10	card6+card9+card11	card6+card9+card12

We represented a hand, including the four types of *SETs,* with the labels card1, card2, card3 through card12. In these 12 cards, there are 4 groups (i.e., 27 3-card combinations per group) that could make up new *SETs* (Table V). We need to check all the possible 3-card combinations (27 × 4 = 108) to filter out those hands that include extra *SETs* (i.e., more combinations than the four built-in *SETs* that are required.)

TABLE VI. ATTRIBUTE CHECK OF 3-CARD COMBINATION.

Color attribute of 3 cards	All different	All the same	Neither (Non-SET)
Attribute value of 3 cards	0+1+2	All 0, 1 or 2	0+0+1, 0+0+2, 1+1+2, 1+1+0, 2+2+0, 2+2+1
Sum of attribute value	3	0, 3 or 6	1, 2, 4 or 5
Remainder (Sum%3)	0	0	1 or 2

Each card has four attributes, for example [0000]. To validate a 3-card combination, we need to check each of their attributes. Only when three cards are either all-the-same or all-different with respect to their four attributes, they form a *SET*. If any attribute of 3 cards is neither all-the-same nor all-different, the 3-card combination is not a *SET*. Let us consider the color attribute of three cards to explain how to check each attribute. TABLE VI illustrates this process.

We summed up the value of the color attributes of three cards, then we divided the sum by three and we looked at the remainders. When the attributes are either all different or all the same, then the remainders are zero. Otherwise, the remainder is either one or two.

card1	card2	card3	card4	card5	card6	card7	card8	card9	card10	card11	card12	id
1	1001	2001	222	1111	2000	110	1010	2210	2012	2101	2220	1
1	1001	2001	0	1111	2222	100	112	121	20	1210	2100	2

Figure 11. Typical hand records before validation

Let us consider two hand records, shown in Figure 11 as an example to illustrate how to check four attributes of 3-card combinations. From the two hand records shown in Figure 11, we took a 3-card combination namely card1, card4 and card7 of each hand (shown in Table VII.). For each hand, we summed up the corresponding four attributes of three cards to get four sums, then divided the four sums by three. Only when the four remainders are all zeros, the 3-card combination forms a SET. Otherwise, they are not a SET. Table VII shows the details of the validation, where A1 refers to attribute 1 of the card, and so on.

TABLE VII. 3-CARD COMBINATION VALIDATION

Hand (id=1)	**A 1**	**A 2**	**A 3**	**A 4**	**Hand (id=2)**	**A 1**	**A 2**	**A 3**	**A 4**
Card 1	0	0	0	1	Card 1	0	0	0	1
Card 4	0	2	2	2	Card 4	0	0	0	0
Card 7	0	1	1	0	Card 7	0	1	0	0
Sum	0	3	3	3	**Sum**	0	1	0	1
Remainder (Sum%3)	0	0	0	0	**Remainder (Sum%3)**	0	1	0	1
	SET					***Non-SET***			

Next, we converted this algorithm into an SQL script that can be run on the database to validate every hand by checking each 3-card combination that could form a *SET*. As an example, let us consider the validation of the 3-card combination of card 1, card4 and card7. We used the condition sentence of *Not (((card1+card4+card7) div 1000) mod 3) + (((card1+card4+card7) div 100) mod 10 mod 3) +(((card1+card4+card7) div 10) mod 10 mod 3) + ((card1+card4+card7) mod 10 mod 3) = 0* to validate that the 3-card combination is not a *SET*; and added the condition in a WHERE clause of a SELECT statement to retrieve the records that do not include new *SETs*.

After validation, those hand combinations having extra *SETs* other than the four built-in ones were all filtered out. Finally, we achieved 269,635,392 hand combinations that were meeting our requirements and stored them into a table.

V. CONCLUSIONS

Experimental studies, which are based on the users' feedback, face various challenges. The main goal of this study was about answering the question whether the identification of *SET*s in the card game SET is to be linked to personal traits. In this paper, we tackled the specific problem of generating the data used to support a user facing Web application that was required in the process of collecting the experimental data.

The amount of data to be generated was very considerable and presented a challenge since it was not possible to count mathematically the number of card combinations (i.e., hands) to consider. By implementing our computational design into a relational database server, we were able to generate all the card combinations required in the Web based user interface.

Our process solved the following two main problems. The first one was about defining the appropriate representation of the cards in the SET game. This task required to consider minimal memory usage and quick validation of SET card combinations. The second challenge was about using a database server that could handle the required amount of data and could easily generate SET cards combinations by applying advanced SQL scripts.

Not only our methodology was successful in generating the required data, but also provided a computational answer to the mathematical challenge of providing the counts of hands combinations. We believe that this approach can be used in many other scenarios in which the creation of data generation is required. Experimental studies based on users' feedback should particularly benefit from this approach.

ACKNOWLEDGMENT

We would like to thank Dr. Janet Striuli and Dr. Laura McSweeney for sharing their research idea and providing support throughout the project. We would also like to express our gratitude to the faculty and students of Fairfield University and Gateway Community College for their commitment to our data collection. A special mention to the Physics department of Fairfield University to let us use their server where we were able to run our code. Thanks also to Set Enterprises, Inc. for allowing us to implement our study based on their SET game. Finally, we would like to thank Dr. Rankin and the Graduate Student Research Committee for providing financial support to our study.

REFERENCES

[1] PlayMonster. LLC, "SET - PlayMonster," [Online]. Available: https://www.playmonster.com/product/set/. [Accessed 07 2020].

[2] Oracle Corporation, "MySQL," [Online]. Available: https://www.mysql.com/. [Accessed 07 2020].

[3] B. L. Davis and D Maclagan, "The Card SET game," *The Mathematical Intelligencer,* vol. 25, no. 3, pp. 33-40, 2003.

[4] J. Vinci, "The maximun number of SETs for N cards and the total number of interal SETs for all partitions of the deck," June 2009. [Online]. Available: https://www.setgame.com/sites/default/files/teacherscorner/SETPROOF.pdf. [Accessed 07 2020].

[5] P. J. Fogle, "SET® AND MATRIX ALGEBRA," 15 03 2019. [Online]. Available: https://www.setgame.com/set-and-matrix-algebra. [Accessed 07 2020].

[6] N. Taatgen, M. van Oploo, J. Braaksma, and J. Niemantsverdriet, "How to construct a believable opponent using cognitive modeling in the game of Set," in *The fifth international conference on cognitive modeling,* pp. 201-206, Bamberg, 2003.

[7] C. De La Riva, M. J. Suárez-Cabal, J. Tuya, "Constraint-based test database generation for SQL queries," in *Proceedings of the 5th Workshop on Automation of Software Test (i.e., AST)* pp. 67–74, 2010.

 ISBN: 978-1-61208-790-0

The Typed Graph Model

Fritz Laux
Fakultät Informatik
Reutlingen University
D-72762 Reutlingen, Germany
email: fritz.laux@fh-reutlingen.de

***Abstract*—In recent years, the Graph Model has become increasingly popular, especially in the application domain of social networks. The model has been semantically augmented with properties and labels attached to the graph elements. It is difficult to ensure data quality for the properties and the data structure because the model does not need a schema. In this paper, we propose a schema bound Typed Graph Model with properties and labels. These enhancements improve not only data quality but also the quality of graph analysis. The power of this model is provided by using hyper-nodes and hyper-edges, which allows to present a data structure on different abstraction levels. We demonstrate by example the superiority of this model over the property graph data model of Hidders and other prevalent data models, namely the relational, object-oriented, and XML model.**

Keywords–typed hyper-graph model; semantic enhancement; data quality.

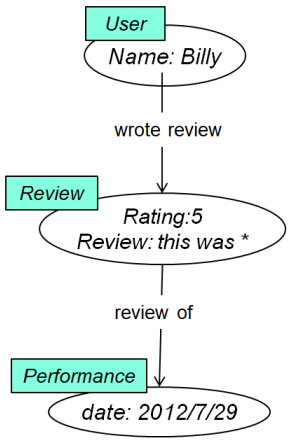


Figure 1. Example graph taken partially from [5], p. 42

I. INTRODUCTION

The popularity of the Graph Model (GM) stems primarily from its application to social networks. Commercial graph database products like Neo4J [1], ArangoDB [2], JanusGraph [3], Amazon Neptune [4], and others have been successfully applied to many domains. There are applications to medicine, drug analysis, scientific literature analysis, power and telephone networks.

The flexibility of the GM and its schema-less implementations are prone to data quality problems. Advocates of the GM like Robinson et al. of Neo4J recommend in their book [5] to use specification by example, which builds on example objects. But this reaches not far enough as the following example taken from Robinson's book shows. It is depicted in Figure 1 and shows a *User* named Billy with its 5-star *Review* on a *Performance* dated 2012/7/29. From this example we cannot know if Billy is allowed to have multiple reviews (on the same performance). For good data quality, a review should depend on the existence of a user and a performance. But this cannot be derived from one example. This means that we have to deal with class things (like a generic Person) and not only with real objects (like Billy) and specify if a relationship is mandatory or optional.

In order to express this information, it is necessary to abstract from a particular situation and specify integrity constraints. The use of a schema would help to ensure data integrity and would clarify the intended situation of the example. Daniel et al. [6] also point out the importance of a schema for data consistency and efficient implementation of a graph database.

Another weakness of the GM is that it has no notation to support different levels of detail and abstraction, which is apparently important for modeling large and complex data structures.

A. Contribution

To overcome these limitations we introduce in this paper a new typed graph model allowing hyper-nodes with complex structured properties (even sub-graphs) and hyper-edges connecting (recursively) one, two or more hyper-nodes. The graph schema provides data types, which allow type checking for instance elements. This ensures a formal data quality. Our model has a higher semantic expressiveness and precision than the prevalent data models, namely the relational, object oriented, and XML data model. This will be demonstrated with typical modeling patterns.

B. Structure of the Paper

With the following overview of Related Work the context for our new typed graph model will be settled. Section III introduces and defines formally the Typed Graph Model (TGM) consisting of a typed schema and a hyper-graph instance connected to the schema. We present a compact and easy to read visualization of the model. The definitions are illustrated by some examples. In the next Section IV our TGM is compared to the Graph Data Model (GDM) of J. Hidders [7]. Then, the semantic expressiveness of the TGM is demonstrated with typical data structures and compared with the prevalent data models, namely the relational, object oriented, and XML

 ISBN: 978-1-61208-790-0

data model. The paper ends with a summary of our findings and gives an outlook on ideas for future work.

II. Related Work

Since the beginning of 1980 many papers on the GM have been published. DBLP [8] alone retrieves 757 matches for the key words "graph data model". If we ignore the papers that present specific applications for the GM incl. XML or Hypertext applications a few dozen of relevant papers remain. In the following, we discuss only papers that present the GM and its extensions (e. g., the Property Graph Model (PGM)) with a formal foundation or papers that use a graph schema:

The notion of PGM was informally introduced by Rodriguez and Neubauer [9]. Spyratos and Sugibuchi [10] use property graphs with hyper-nodes and hyper-edges for their graph data model. The main difference to our approach is that no schema is used and properties have no predefined data type. Another approach with hyper-edges is presented by Bu et al. [11] who treats a label like a node connecting a set of nodes, which he calls hyper-edge. The nodes itself can be of different types. In this case Bu calls the graph a unified hyper-graph. The unified hyper-graph model is then applied to problem of ranking music content and combining it with social media information. Compared to our TGM the unified hyper-graph of Bu is only defined for graph instances. It is not not clear if the nodes have any type checking and if the whole graph is ruled by a schema.

Ghrab et al. [12] present GRAB, a schemaless graph database based on the PGM. It supports integrity constraints but cannot ensure data quality because of missing data types for properties and labels. Neo4J [5] has similar foundations and features. It has optional support for integrity constraints and comes with a powerful and easy to use graph query language, called *Cypher*.

All these PGM originate as instance graphs and no special attention is given to the graph schema. No attempt is made to specify the different types of edges and the multiplicity of connections (edges) between different node types. Nodes are not typed and labels are not a proper substitute.

Amann and Scholl [13] seem to be the first authors who connect a graph schema with its graph database instance. Nodes and edges do not have properties but both must conform to the schema. Their model is used for an algebra (hyperwalk algebra) for traversing the graph.

Marc Gyssens et al. [14] and Jan Hidders [7] use a labeled GM to represent a database schema where each property of an object is modeled as a node in the graph. Labels are used to name node classes and edges. The models become confusing because a node represents either an object, a property or a data type. Still, it is not possible to restrict the cardinality of schema edges (relationships). Hidders' model is explained in more detail and compared to our TGM in Section IV.

Similar to Amann and Scholl the paper of Pabón et al. [15] uses a graph schema to query the graph database. They distinguish different node types, which they call "sort". The supported types are: *object class nodes* (complex objects), *composite-value class nodes* (for aggregate values), and *basic-value class nodes* (primitive data types). This model seems to be equivalent to (complex) nodes with properties governed by a schema. A mechanism to abstract and group sub-graphs would help to make the model easier to communicate.

Pokorný [16] uses a binary ER-Model as graph conceptual schema. For the graphical rendering he uses a compact entity representation for the nodes with attribute names inside the entity box. This solves the problem using the same node symbol for entities and attributes (properties) as it is the case with Gyssens [14] and Hidders [7] models. The edge cardinality is represented in a form of crow-foot notation.

In order to make the GM usable for real life scenarios with hundreds of schema elements, it is necessary to group or combine graph elements to higher abstracted objects. This would make the model easier to handle.

The need for grouping graph elements is addressed by Junghanns et al. [17]. Their model allows to form logical sub-graphs (graph collections) with heterogeneous nodes and edges. With this it is possible to aggregate sub-graphs, e. g., user communities. The authors use UML-like graphical rendering of nodes to make the model better readable but their model fails to specify the cardinality of schema edges.

A step toward to complex composite nodes as an alternative approach to aggregation presents Levene [18] by allowing the graph vertices to be recursively defined as a finite set of graphs. These hyper-nodes do not form a well-founded set as a node may contain itself, which violates the foundation axiom for the Zermelo-Fraenkel set theory.

A relatively new formal definition including integrity constraints was given by Angles [19]. However, his model does not allow structured objects and grouping or aggregation. In the following section, we simplify his definitions and use it as basis for our TGM.

A. Comparison with Ontology Languages

Ontology languages like RDFS [20] and OWL [21] are designed to specify ontologies and have their strength in allowing reasoning over instances of it. They are often used to semantically describe Linked Open Data (LOD) and the statement triples are usually visualized as graph structures. RDFS and OWL provide a general type system that could be used to form user defined types. This would allow to use it as basis for a graph schema language. But if we look at the W3C OWL 2 Structural Specification [22] it seems difficult to define user specific classes and W3C itself uses UML class diagrams to illustrate OWL structures.

The specification of data structures is not their core intention. In RDFS for instance it is not possible to define the cardinality of relationships. Likewise, OWL Lite has strong limitations on allowing only 0 or 1 as multiplicity of properties. Simple unique requirements and relations like one-to-one, one-to-many and many-to-one are cumbersome to define even in OWL Full. Complex data structures need a modeling language that allows to define different levels of abstraction, which is not the strength of these ontology languages. Most examples of RDFS or OWL do not care about the multiplicity of relationships (cardinalities may be guessed via property names) and grouping of attributes seems to be on the same level as objects or subjects.

 ISBN: 978-1-61208-790-0

III. The Typed Graph Model

Our TGM informally constitutes a directed property hypergraph that conforms to a schema. In the following definitions our notation uses small letters for elements (nodes, edges, data types, etc.) and capital letters for sets of elements. Sets of sets are printed as bold capital letters. A typical example would be $n \in N \in \mathbf{N} \subseteq \wp(N)$, where $\wp(N)$ is the power-set of N.

A. Graph Schema

Let T denote a set of simple or structured (complex) data types. A data type $t := (l, d) \in T$ has a name l and a definition d. Examples of simple (predefined) types are $(int, \mathbb{Z})$, $(char, ASCII)$, etc. It is also possible to define complex data types like an order line $(OrderLine, (posNo, partNo, partDescription, quantity))$. The components need to be defined in T as well, e. g., $(posNo, int > 0)$. Recursion is allowed as long as the defined structure has a finite number of components.

Definition 1 (Typed Graph Schema)**.** *A typed graph schema is a tuple $TGS = (N_S, E_S, \rho, T, \tau, C)$ where:*

- *N_S is the set of named (labeled) objects (nodes) n with data type $t := (l, d) \in T$, where l is the label and d the data type definition.*
- *E_S is the set of named (labeled) edges e with a structured property $p := (l, d) \in T$, where l is the label and d the data type definition.*
- *ρ is a function that associates each edge e to a pair of object sets (O, A), i. e., $\rho(e) := (O_e, A_e)$ with $O_e, A_e \in \wp(N_S)$. O_e is called the* tail *and A_e is called the* head *of an edge e.*
- *τ is a function that assigns for each node n of an edge e a pair of positive integers (i_n, k_n), i. e., $\tau_e(n) := (i_n, k_n)$ with $i_n \in \mathbb{N}_0$ and $k_n \in \mathbb{N}$. The function τ defines the min-max multiplicity of an edge connection. If the min-value i_n is 0 then the connection is optional.*
- *C is a set of integrity constraints, which the graph database must obey.*

The notation for defining data types T, which are used for node types N_S and edge types E_S, can be freely chosen. This makes the expressiveness of the TGS at least as strong as the models to which it is compared in Section IV.

B. Typed Graph Model

Definition 2 (Typed Graph Model)**.** *A typed graph Model is a tuple $TGM = (N, E, TGS, \phi)$ where:*

- *N is the set of named (labeled) nodes n with data types from N_S of schema TGS.*
- *E is the set of named (labeled) edges e with properties of types from E_S of schema TGS.*
- *TGS is a typed graph schema as defined in Subsection III-A.*
- *ϕ is a homomorphism that maps each node n and edge e of TGM to the corresponding type element of TGS, formally:*

$$\begin{aligned} \phi: \quad TGM &\rightarrow TGS \\ n &\mapsto \phi(n) := n_S(\in N_S) \\ e &\mapsto \phi(e) := e_S(\in E_S) \end{aligned}$$

The fact that ϕ maps each element (node or edge) to exactly one data type implies that each element of the graph model has a well defined data type. The homomorphism is structure preserving. This means that the cardinality of the edge types are enforced, too. Data type and constraint checking is applied for all nodes and edges before any insert, update, or delete action can be committed. If no single type can be defined, union type or *anyType* (sometimes called *variant*) may be applied. Usually this is an indication for a weak data model and it should be clear that this could affect data quality and processing.

As graphical representation for the TGS we adopt the UML-notation for nodes and include the properties as attributes including their data type. Labels are written in the top compartment of the UML-class. Edges of the TGS are represented by UML associations. For the label and properties of an edge we use the UML-association class, which has the same rendering as an ordinary class but its existence depends on an association (edge), which is indicated by a dotted line from the association class to the edge. This not only allows to label an edge but to define user defined edge types. The correspondence between the UML notation and the TGS definition is the following:

TABLE I. TGS correspondence with UML notation

TGS	UML
$n \in N_S$	class
$e \in E_S$	association
$t = (l, d) \in T$	l = name of n resp. e; d = type of n resp. e
$\rho(e)$	all ends of e
$\tau_e(n)$	(min,max)-cardinality of e at n
C	constraints in [ ] or { }

The use of hyper-nodes $n \in N_S$ and hyper-edges $e \in E_S$ instead of simple nodes resp. edges allow to group nodes and edges to higher abstracted complex model aggregates. This is particularly useful to keep large models clearly represented and manageable. Large graph models may then be grouped into sub-graphs like in Junghanns et al.[17]. Each sub-graph can be rendered as a hyper-node. If the division is disjoint these hyper-nodes are connected via hyper-edges forming a higher abstraction level schema (see Figure 3 (b)).

C. Examples

Lets recall the example graph from Figure 1 and model its corresponding schema. We want to make clear that a user may write as many reviews as he likes, but only one for a particular performance. A rating needs to refer exactly to one performance and one user. This is reflected in Figure 2 by the "1:many" and "0 or many:1" relationships. We use the UML-notation for the schema and keep the notation from Figure 1 for the instance graph for clarity.

 ISBN: 978-1-61208-790-0

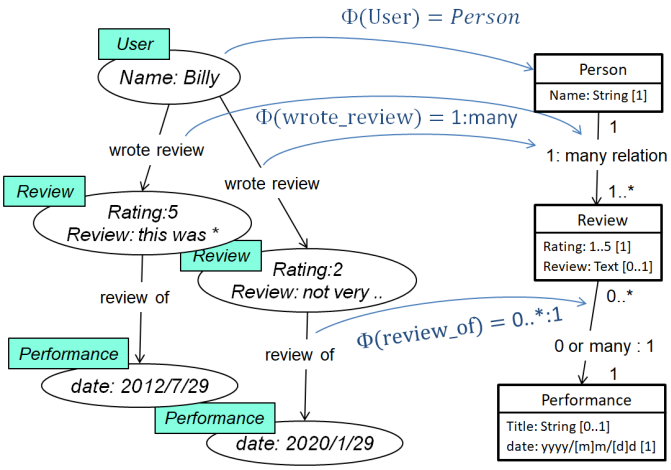


Figure 2. Example graph with schema in UML notation

The homomorphic mapping ϕ guaranties that the instance graph obeys the schema, i. e. type, cardinality, and constraint checking. Now, it is clear from the schema that a user must have at least one review. The review is existence dependent on the user and a performance. The "wrote review" edge is a 1:many relation and "review of" is an optional many:1 relation. This has the consequence that a review needs a person and a performance. But, a performance may exist without any review.

In the next example we present a commercial enterprise that sells products and parts to customers. The enterprise assembles products from parts and if the stock level is not sufficient it purchases parts from different suppliers. Figure 3 models this situation using UML rendering. It demonstrates the abstraction power of the TGM showing two schema abstraction levels. The upper part (a) shows the TGM on a detailed level. The properties are suppressed in the diagram for simplicity except for *Customer* and *CustOrder*. The schema is grouped into 3 disjoint sub-graphs depicted with dashed shapes.

In the lower part (b) these sub-graphs are shown as hyper-nodes of the graph schema. This allows a simplified and more abstracted view of the model. Also, some aggregate properties (e. g. #orders) are shown to illustrate the modeling capabilities. The hyper-edges connecting these abstracted nodes must use the most general multiplicity of the multiple edges it combines. In the example the edge *orders/from* combines two edges, i. e., *orders* with 0..1 - 1 multiplicity and *from* with 0..* - 0..* multiplicity, which leads to the most general multiplicity.

IV. Comparison with other Data Models

In the following, we compare our TGM to other models with respect to structural differences and schema support. We point out modeling restrictions of these models and show how such situations are modeled with TGM. Query and manipulation languages are beyond the scope of this paper.

A. Comparison with GDM of Jan Hidders

Jan Hidders' [7] model added labels and properties together with their data types to nodes and edges (relationships). Property names are modeled as edges in the schema. This allows to model labeled relationships with complex properties. Structured and base data types share the same graphical representation, which makes it difficult to distinguish both. The ISA-relationship is rendered as a double line arrow. Hidders' model does not allow to restrict the cardinality of relationships. This restriction limits its modeling power compared to the TGM, which provides a min-max notation for the cardinality.

The example in Figure 4 is from the publication of Hidders [7]. The schema shows *Employee* and *Department* classes linked by a *Contract*. The relationship *Contract* is existence dependent on the connected nodes. The properties of *Contract* are salary of type *int*, begin-date and end-date of structure-type $date = (day, month, year)$. In Hidders' model these dates are modeled on the element level using data type *int*. Hidders' schema elements, i. e., nodes (objects), edges (properties) and data types appear on the same visual level, which makes it difficult to read and obscures semantics. The modeling power of complex data types provide a clear advantage for the TGM.

B. Comparison with the Relational Model (RM)

There is a 1:1 correspondence between attributes and properties and any relation can be modeled as a node with properties. The min-max notation for relationship multiplicity can model any link cardinality. The TGM can therefore easily represent tabular structures, foreign key constraints (many-to-one relationships), and join-tables as the building blocks of the RM. Beyond this, the TGM is able to directly model many-to-many relationships of any min-max multiplicity. This makes the TGM strictly stronger than the relational model. Another difference to the RM is that foreign keys (FK) are not necessary because their function is taken over by an edge linking the FK-node (Table 1 without FK) with the referenced node (Table 2). This can be seen in Figure 5 (a).

A join-table in the RM is existence dependent on the tables it refers to by FKs. The FKs forming the primary key (PK) of the join table are not necessary in the TGM because of the same reason as mentioned above.

In Figure 5 (b) the join-table RST maps directly to an hyper-edge labeled RST with property col_3 and without FKs. To make the ternary relationship example less abstract the RST could be an offer of products from Table 1 from a supplier of Table 3 to the client of Table 2. With this in mind it is clear that an offer depends on the product, the supplier, and the client.

The TGM can also represent non-normalized tables because the model supports complex structured data types having multivalued or array data. It is only necessary to define the necessary data types in the set of available data types T.

C. Comparison with XML Schema

XML documents represent hierarchical hypertext documents. The document structure is defined by an XML schema. The hierarchy of XML-documents is directly supported by the TGM using directed edges. XLink provides references (arcs) between elements of internal or external XML-documents. Extended XLinks can connect to more than one element, but the references are always instance based, i. e. the target elements must be listed by URI. The TGM is more abstract and expressive allowing the definition of non-hierarchical references on the schema level.

 ISBN: 978-1-61208-790-0

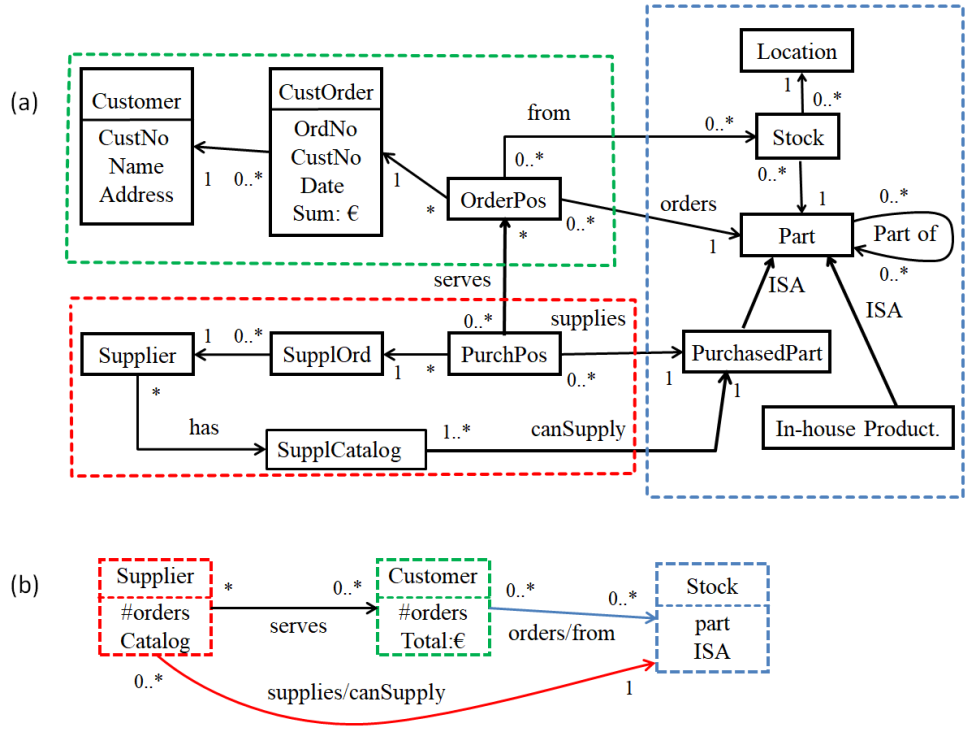
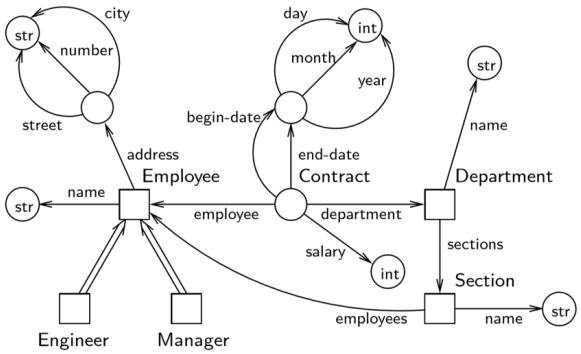
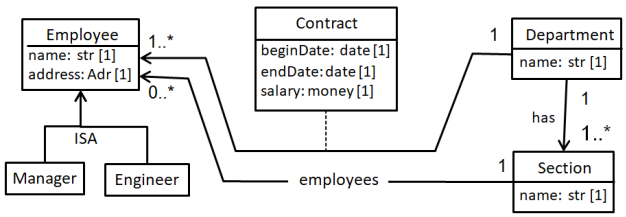


Figure 3. Example TGM of a commercial enterprise showing two levels of detail

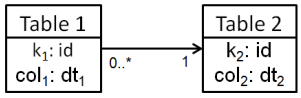


Example schema from Hidders' GDM

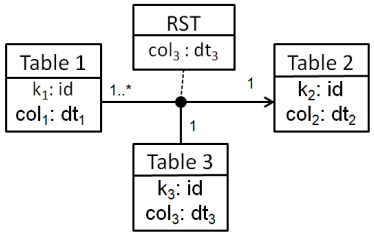


The same example modeled with TGM

Figure 4. Comparison by example with Hidders' GDM



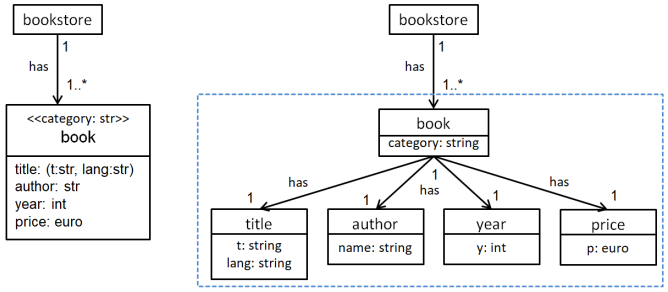
(a) 2 tables with FK relationship



(b) 3 tables connected by a ternary join-table

Figure 5. Modeling a many-to-one relationship (FK) and a ternary join-table with TGM

As example serves a bookstore offering an unlimited number of books. A simple XML-schema for the bookstore is given by w3schools.com. The schema defines books with elements like "title", "author", etc. and its corresponding data types. Some data types are not as precise as they could, e. g. the data type xs:double for the price element. We will replace cs:double in our TGM by the money-type *euro* to be more precise. Some elements have attributes attached like the language ("lang") of a book title. The attribute minOccurs="1" of xs:sequence requires the bookstore to have a least one book.



(a) Compact representation (b) Detailed showing each XML-element

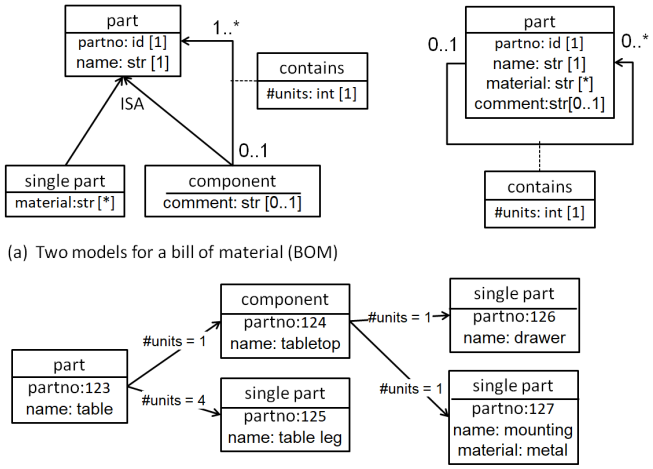
Figure 6. Comparison by example with the XML Schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema ... >
  <xs:element name="bookstore" >
    <xs:complexType >
      <xs:sequence minOccurs="1"
          maxOccurs ="unbounded" >
        <xs:element name="book" >
          <xs:complexType>
            <xs:sequence>
              <xs:element name="title" >
              <xs:complexType>
                <xs:simpleContent>
                  <xs:extension base="xs:string">
                    <xs:attribute name="lang"
                        type="xs:string" />
                  </xs:extension>
                </xs:simpleContent>
              </xs:complexType>
                </xs:element>
              <xs:element name="author"
                  type ="xs:string"/>
              <xs:element name="year"
                  type ="xs:integer"/>
              <xs:element name="price"
                  type ="xs:double"/>
            </xs:sequence>
            <xs:attribute name="category"
                type="xs:string"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

If we model the XML-elements as nodes in TGM then XML-attributes and the element values should be represented as properties. The name of an XML-element is mapped to a node label. The order of the XML-elements cannot be represented with this approach and XML-element values can be distinguished from XML-attributes by convention only.

An alternative TGM model represents the complete book structure as one node. In this case the XML-elements and their attributes are modeled as structured properties of the book. The order of the elements and their associated attributes can be preserved. In fact, if XML Schema is used for specifying the data types N_S and E_S (see Subsection III-A) all the flexibility and semantics provided be XML Schema can be represented with the TGS. This argument shows that the TGM is at least as powerful as the XML model.

The example bookstore is depicted in Figure 6 where the left part (a) shows the compact version with the whole book



(a) Two models for a bill of material (BOM)

(b) An instance of a product structure conforming to the above schemas

Figure 7. Comparison by example with the OOM

modeled as one node and the right part (b) shows the version where each XML-element is modeled as node. We see from this example another possibility to use sub-graphs for higher abstracted models.

D. Comparison with the Object-Oriented Model

Because we already use the UML for rendering the TGM, it is easy to see that classes correspond one-to-one with typed hyper-nodes. Any methods are simply ignored as we only deal with the network structure of OOM. Any complex internal class structure can be directly modeled by appropriate data types $t \in T$. The type set T is defined beforehand but can contain any user defined structures. In contrast to the OOM the TGM allows different levels of abstraction in the modeling depending whether a structure is modeled by a detailed graph with simple types or a more compact graph using complex data types. This shows the same semantic expressiveness for structures, but a higher flexibility of the TGM. Considering the operations on data the OOM has the advantage to specify the allowed operations by methods.

The UML provides a rich set of association types, which need to be mapped to the label of the edges. Our TGM provides types not only for nodes but also for edges (called associations in UML). With this information it is possible to model different association types like aggregation, generalization, etc. Even user defined associations are possible, e. g., an aggregate could be further qualified as un-detachable or detachable composition or a loose containment. The arrow of the edge only indicates the reading direction of the association but does not limit the navigation of the TGM.

It is also possible to model recursive structures as the examples from Figure 7 illustrates. The bill of material (BOM) is an important example for a recursive structure used in production planning and control. It defines recursively a (compound) part with its components until a single part is reached. As example a table is given in Figure 7 (b) consisting of 4 table legs and a tabletop consisting of a drawer and a mounting.

If the edge of *contains* in Figure 7 (a) is followed against the arrow direction it is possible to find the component where

an individual part is built-in. A complete *where-used list* for a generic (not an individual) part may be obtained with a small schema modification. The from-end of the *contains*-edge needs to change its multiplicity from 0..1 to 0..*. With this modification all components can be identified where a generic part is used.

V. Conclusion and Future Work

This paper presents a structure definition of the TGM and an UML-like notation to visualize a graph database and its graph schema. Due to the TGS with predefined and user-defined data types the TGM improves the formal data quality compared to other graph models. We have demonstrated the superior modeling power in comparison to other graph data models and prevalent data models, namely relational, object oriented and XML model. The model supports built in and user defined complex data types, which allow different abstraction levels. Another possibility for abstraction is to compress a sub-graph into a hyper-node reducing the visible complexity.

Because of its semantic modeling power the TGM could serve as a model that supports data integration from various data sources with different data models. The main challenge for an automated data integration are incompatible data sources where the TGM could help to solve quality issues and resolve inconsistent data. Details still need to be investigated. The development of a manipulation and query language for the TGM is future work. The idea is to combine elements of other graph languages with the dot-notation known from object-oriented languages.

References

[1] Neo4J - Homepage, [Online] URL: https://neo4j.com [retrieved: 2020-04-14]

[2] ArangoDB - Graph and Beyond, [Online] URL: https://www.arangodb.com [retrieved: 2020-04-14]

[3] JanusGraph - Homepage, [Online] URL: https://janusgraph.org [retrieved: 2020-04-14]

[4] Amazon Neptune, [Online] URL: https://aws.amazon.com/de/neptune [retrieved: 2020-04-14]

[5] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*, 2^{nd} ed.,O'Reilly Media, 2015.

[6] G. Daniel, G. Sunyé, and J. Cabot, "UMLtoGraphDB: Mapping Conceptual Schemas to Graph Databases", in Proceedings of Conceptual Modeling - 35^{th} International Conference ER, Gifu, Japan, pp. 430 - 444, 2016. [Online] URL: https://hal.archives-ouvertes.fr/hal-01344015/document [retrieved: 2020-02-04]

[7] J. Hidders, "Typing Graph-Manipulation Operations", In Proceedings of the 9^{th} International Conference on Database Theory" (ICDT), Siena, Italy, pp. 391 - 406, 2003.

[8] DBLP computer science bibliography, [Online] URL: https://dblp.uni-trier.de [retrieved: 2020-04-14]

[9] M. A. Rodriguez and P. Neubauer, "Construction from dots and lines", Bulletin of the American Society for Information Science and Technology, Vol. 36(No 6), pp. 35-41, ISSN:1550-8366, 2010. [Online] URL: https://arxiv.org/pdf/1006.2361.pdf [retrieved: 2020-04-14]

[10] N. Spyratos and T. Sugibuchi, "PROPER - A Graph Data Model Based on Property Graphs", ISIP 10th International Workshop, Communications in Computer and Information Science, vol.622,pp. 23 - 35, Springer, 2015.

[11] J. Bu, S. Tan, C. Chen, C. Wang, H. Wu, L. Zhang, and X. He, "Music Recommendation by Unied Hypergraph: Combining Social Media Information and Music Content", In Proceedings of the 18th International Conference on Multimedia (ACM Multimedia 2010), Firenze, Italy, pp. 391-400, 2010

[12] A. Ghrab,O. Romero, S. Skhiri, A. Vaisman, and E. Zimányi, "GRAD: On Graph Database Modeling", Cornell University Library,arXiv:1602.00503, 2016. [Online] URL: https://arxiv.org/ftp/arxiv/papers/1602/1602.00503.pdf [retrieved: 2020-04-14]

[13] B. Amann and M. Scholl, "Gram: A Graph Data Model and Query Language", In Proceedings of the ACM Conference on Hypertext (ECHT '92), pp. 201 - 211, Milan, Italy, 1992.

[14] M. Gyssens, J. Paredaens, J. Van den Bussche, and D. Van Gucht, "A Graph-Oriented Object Database Model", IEEE Transactions on Knowledge and Data Engineering, Vol 6 No 4., pp. 572 - 586, 1994.

[15] M. C. Pabón, C. Roncancio, and M. Millán, "Graph Data Transformations and Querying", In Proceedings of the 2014 International C* Conference on Computer Science & Software Engineering (C3S2E '14), Montreal Canada, Article No 20, pp. 1 - 6, 2014. [Online] URI: https://doi.org/10.1145/2641483.2641521 [retrieved: 2020-04-14]

[16] J. Pokorný, "Conceptual and Database Modelling of Graph Databases", In Proceedings of the 20^{th} International Database Engineering & Applications Symposium (IDEAS 2016), Montreal, Canada, pp. 370 - 377, 2016.

[17] M. Junghanns, A. Petermann, N. Teichmann, K. Gómez, E. Rahm, "Analyzing extended property graphs with Apache Flink", Proceedings of the 1^{st} ACM SIGMOD Workshop on Network Data Analytics (NDA@SIGMOD 2016), San Francisco, USA, pp. 3:1 - 3:8 2016.

[18] M. Levene and A. Poulovassilis, "The hypernode model and its associated query language", In Proceedings of the 5^{th} Jerusalem Conference on Information Technology, Jerusalem, pp. 520 - 530, 1990.

[19] R. Angles, "The Property Graph Database Model", Proceedings of the 12^{th} Alberto Mendelzon International Workshop on Foundations of Data Management, Cali, Colombia, CEUR WS Proc., 2018, [Online] URL: http://ceur-ws.org/Vol-2100/paper26.pdf [retrieved: 2020-04-14]

[20] D, Brickley and R.V. Guha (eds.), RDF Schema 1.1 W3C Recommendation, published 25 February 2014, [Online] URL: https://www.w3.org/TR/rdf-schema/ [retrieved: 2020-04-14]

[21] W3C OWL Working Group, OWL 2 Web Ontology Language Document Overview (Second Edition) W3C Recommendation, published 11 December 2012, [Online] URL: https://www.w3.org/TR/owl2-overview/ [retrieved: 2020-04-14]

[22] B. Motik, P. F. Patel-Schneider, and B. Parsia (eds.), OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition) W3C Recommendation, published 11 December 2012, [Online] URL: https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/ [retrieved: 2020-04-14]

 ISBN: 978-1-61208-790-0

Automated Generation of Graphs from Relational Sources to Optimise Queries for Collaborative Filtering

Ahmad Shahzad
School of Electrical Engineering and Computer Science
University of Liverpool, Liverpool, L69 3BX, U.K
Email: ahmads@liverpool.ac.uk

Frans Coenen
School of Electrical Engineering and Computer Science
University of Liverpool, Liverpool, L69 3BX, U.K
Email: coenen@liverpool.ac.uk

***Abstract*—Graph abstraction is an intuitive and effective approach for collaborative filtering as used in, for example, recommender engines. However, for many collaborative filtering applications, the transactional data is kept in a relational database and, through bespoke processes, is Exported, Transformed and Loaded (ETL) into a graph database where collaborative filtering algorithms can be applied. However, the ETL process requires knowledge of the source relational database, the target graph database and the application domain. The ETL process, therefore, tends to be expensive, non-optimised for graph queries and relies heavily on application domain knowledge and understanding of the property graph engine for the graph database. In this paper, a mechanism is presented whereby data in a relational format, which is normalised to 5^{th} normal form, can be automatically converted to a graph database format, through an automated process. The presented evaluation demonstrates, using the recommendation engine application domain as an example, that the proposed mechanism is more efficient than comparable approaches to reduce the execution time required for collaborative filtering.**

***Keywords*—Graph Construction; Collaborative Filtering; Query Optimization; Normalization; Cold Start.**

I. INTRODUCTION

Graphs are featured in a range of different application domains. They play a pivotal role for the solution of a variety of problems, from simple path finding problems to much more complex problems, such as collaborative filtering for *Recommendation Engines*. The principal advantage of representing data as graphs is that, at the physical level, a graph database satisfies the so called index-free adjacency property in which each node stores information about its neighbours only; there is no requirement for a global index of the connections between nodes. As a result, the traversal of an edge is independent of the size of the data. This makes it very efficient to conduct local analysis of the graph and means it is well suited to the processing of large data collections, or tasks, like collaborative filtering, where for any given vertex v it is required to find the most similar vertices from a set of vertices V. Although relational databases can provide a basis for collaborative filtering, they feature slow runtime, especially when there are many joins between entities. Graph databases are inherently faster in modelling and identifying associations between entities because they do not require expensive join operations and can be instantiated on distributed data frameworks. Hence, it is desirable to transform a relational database into a graph database for the purpose of performing graph analytical queries, for example collaborative filtering for recommender engines. However, the transformation process, which involves Export, Transform and Loading (ETL), is usually a bespoke process. Thus, the ETL processes tend to be expensive and require knowledge of the relational and graph databases used, and the mapping functions to associate relation and graph database entities. In this paper, a mechanism is proposed to automate the process of migration from a relational database format to a graph database format. The generated graph database shows better performance compared to other approaches in terms of the efficient execution of collaborative filtering algorithms.

A good example of a graph analytical query application is collaborative filtering in the context of *Recommendation Engines*. This is the exemplar graph analytical application domain used throughout this paper to illustrate the proposed approach. *Recommendation Engines* have a significant impact on

 ISBN: 978-1-61208-790-0

the success of business and diversity of sales [1]. The intuition underpinning *Recommendation Engines* is to take advantage of past history to make predictions. However, this means that users with little or no history cannot be provided with recommendations to any degree of accuracy. This is known as the *cold start recommendation engine* problem. There have been various techniques which have been effectively used to make recommendations given the *cold start* problem [2][3][4]. However, all these techniques rely on auxiliary information concerning the target user for whom recommendations need to be generated. This extra information is compared with other users to identify a "user group". Recommendations are then made based on the identified user group and candidate items for recommendation are ranked according to some criteria.

The problem of migrating from a relational database to a graph database, to support the faster execution of collaborative filtering algorithms, can be stated as follows. Let $S(R_1, R_2, ...R_n)$ be a relational schema which consists of a set of Relations. Each Relation R_i from schema S consists of a set of attributes, $A_i(A_{i1}, A_{i2}, \ldots, A_{im})$, which can be uniquely identified by a set of *Primary Keys*, $PK_i(PK_{i1}, PK_{i2}, \ldots, PK_{in})$. A primary key attribute $Pk_{ij} \in Pk_i$ can be used as a reference to another relation R_j, also known as a *Foreign Key*. The set of foreign keys for a relation R_j is defined as $FK_j(FK_{j1}, FK_{j2}, .., FK_{jn})$. Note that $FK_j \subset PK_j$, and that r is an instance of R and comprises a tuple of the form $\langle r_1, r_2, .., r_k \rangle$.

A graph G is defined as $G = (V, E, T_V, T_E)$, where V is a finite set of nodes, $E \subseteq V \times V$ is a finite multi-set of edges, T_V is a finite set of node types, and T_E is a finite set of edge types. Each node is mapped to a node type by a mapping function $\phi V : V \rightarrow T_V$, and each edge is mapped to an edge type by another mapping function $\phi E : E \rightarrow T_E$. A node $v_i \in V$, or an edge $ek(v_i, v_j) \in E$, has a set of $\langle attribute, value \rangle$ pairs which constitute the properties of the vertex or edge. The schema of a property graph G is defined as a directed graph $GS = (T_V, T_E)$, where T_V is a finite set of node types, and $T_E \subseteq T_V \times T_V$ is a finite set of edge types. The task is to transform S into GS.

In this paper, a comprehensive approach to the automated migration of relational to graph database storage is proposed. The proposed approach converts a relational database schema *S* into a graph database schema *GS*. As already noted, the advantage offered is that the execution of queries, which require filtering over values of low cardinality, will be more efficient than alternative relational to graph databases processes. The translation takes advantage of integrity constraints assuming 5^{th} normal form. This paper targets property graph databases to model graph data and the associated graph engine specific query language. This makes the approach independent of the specific graph engine implementation. In order to test the feasibility of the proposed approach, a complete software solution was developed for converting relational to graph databases. The solution was evaluated in the context of the *cold start* recommendation engine *collaborative filtering problem* using the *Movielens* data set. The evaluation demonstrated that there was no loss of data in translation and that the execution of queries was more efficient than in the case of other compatible approaches to achieve the same result.

The rest of this paper is organized as follows. An overview of relevant previous work is presented in Section II. This is followed by a description of the proposed approach in Section III. The evaluation of the approach is given in Section V. The paper is then completed with some conclusions presented in Section VI.

II. Related Work

There has been some previous work directed at automating the ETL process. In [5], an approach was presented to convert a relational database into a graph database founded on the property graph model. The significance of the property graph model was that it was expressive enough to cover many real world applications and it was applicable to a range of graph database realisations, such as Neo4J[6], Titan[7] and OrientDb[8]. The authors focused on speeding up the processing of queries over the constructed graph by building a graph structure based on joinable tuple aggregations. However, the approach had two main limitations: (i) a simplified version of a property graph was considered where only nodes have properties, while edges have labels that represent relationships between the data at nodes, and (ii) only relational database style queries were

considered. The work presented in [9] improved on the ideas presented in [5] by considering graph edge properties and n-way relationships when more than two foreign keys were involved. However, this led to a design which added redundant information to the edges, information that semantically did not exist. In [10], a scalable "map reduce" approach was proposed for converting relational databases into graph databases, however, the design and mapping of the relational to graph database remained a choice for the user; in other words, it was a semi-automated approach. In the context of collaborative filtering, Filho et al. [11] proposed topological analysis for the generation of heuristics in terms of betweenness, closeness and degree centrality, so as to identify nodes which could be considered to be *hubs* and/or *authorities*, so as to improve collaborative filtering results. However, the approach did not improve the run time performance. The approach presented in this paper addresses the disadvantages associated with this earlier work.

The *cold start* problem in context of *collaborative filtering* has been well studied. Suggested solutions can be categorised according to how the missing information is collected [12]: (i) explicit information collection and (ii) implicit information collection. However, in the "real world", it is not always possible to explicitly gather information about a user, hence, implicit information is typically the most feasible approach. Implicit information collection relies on using information about the user which is already available in the system or freely available in public space, such as social media. An implementation of Quantitative Association Rules (QAR) [13] for implicit information collection of *Recommendation Engines* was used with respect to the *cold start Recommendation Engine* problem presented in this paper.

III. Converting a Relational Data Model to a Graph Model

The proposed relational to graph transformation is founded on the application of a set of rules. Let K be the total number of foreign keys in a relation R_i. For any given relation R_i with a set of primary keys defined by PK_i, and any given attribute of this relation A_{ij}, let $|CA_{ij}|$ be the cardinality of that attribute. Thus, $|PK_i|$ is the total number of rows in a relation. The ratio of values to rows is:

$$\lambda = |CA_{ij}|/|PK_i| \quad (1)$$

Let T be the selected threshold for a given domain. Then the following proposed rule set can be applied to transform a relational schema S to a graph schema GS:

- Rule1: If $K == 1$ in R_i, which references R_j. For each $r_i \in R_i$ referencing $r_j \in R_j$ create two vertices for r_i and r_j linked by an edge with the property FK_{i1}.
- Rule2: If $K == 2$ in R_i, which references R_j and R_k. For each $r_i \in R_i$, referencing $r_j \in R_j$ and $r_k \in R_k$, create two vertices for r_j and r_k linked by an edge with properties equivalent to all the attributes of r_i.
- Rule3: If $K \geq 3$ in R_i, which references $R_j ... R_n$. For each $r_i \in R_i$, referencing $r_j \in R_j ... r_n \in R_n$, create $n+1$ vertices $r_i, r_j ... r_n$ with edge properties $FK_{i1} ... Fk_{in}$, respectively.
- Rule4: For any A_i at any node, if λ is less than threshold T, create a new node with a relationship to the node A_i where the property belongs. An automated Id will be generated for such new node and it will be connected to the parent node where it was originally located.
- Rule5: If a vertex r_i already exists, then it will not be created again; instead, it will be used with other vertices created using the first three rules.

The above can be applied in parallel as only one of the above rules will be applicable to each relational table contained in the relational database to be transformed.

Database normalisation ensures integrity of the data and prevents the chance of the duplication of data. A good database design should conform to at least 3^{rd} normal form. If a schema is in 3^{rd} normal form, then, in most cases, although not all, it can qualify to be in 5^{th} normal form. In practice, this is generally true for all transactional data. However, some applications do not enforce any constraints on the database side and, hence, do not have any foreign keys in the relations. In these special cases,

 ISBN: 978-1-61208-790-0

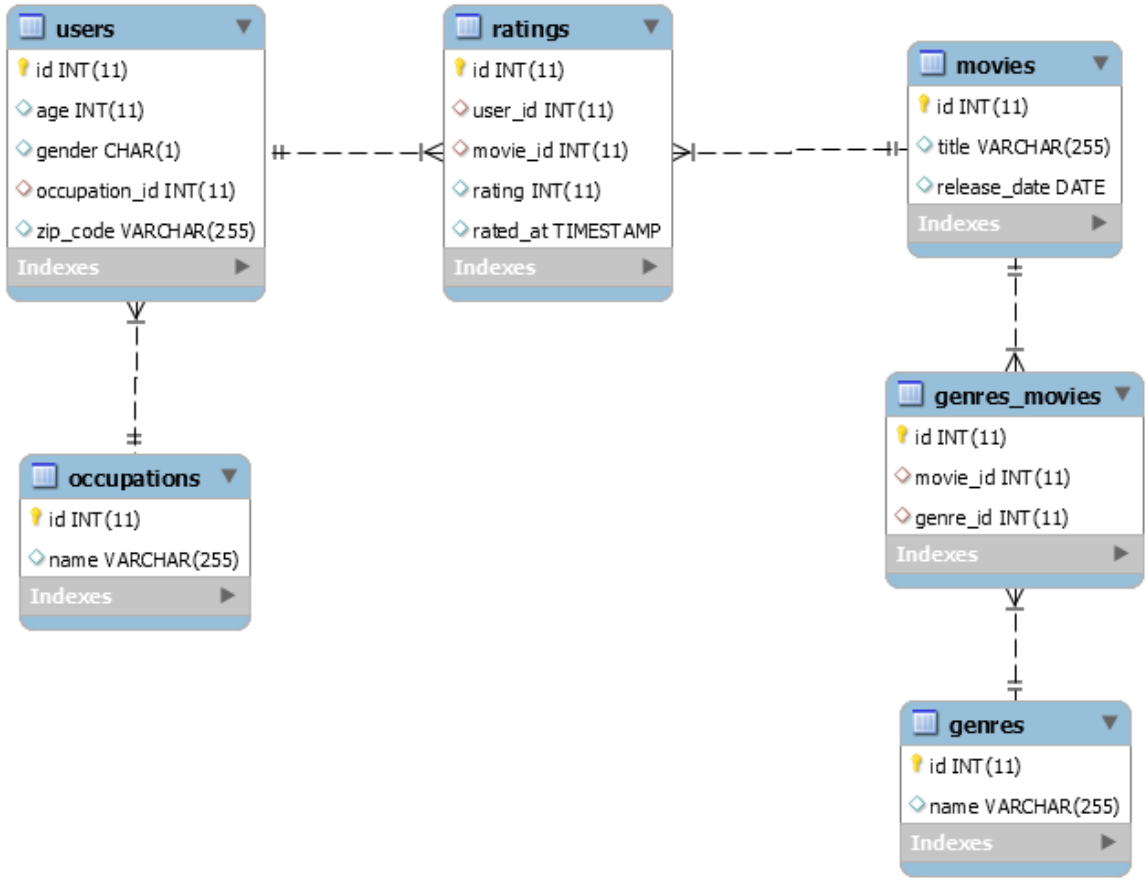


Figure 1. Entity Relationship Diagram of Movilens Database in 5^{th} normal form.

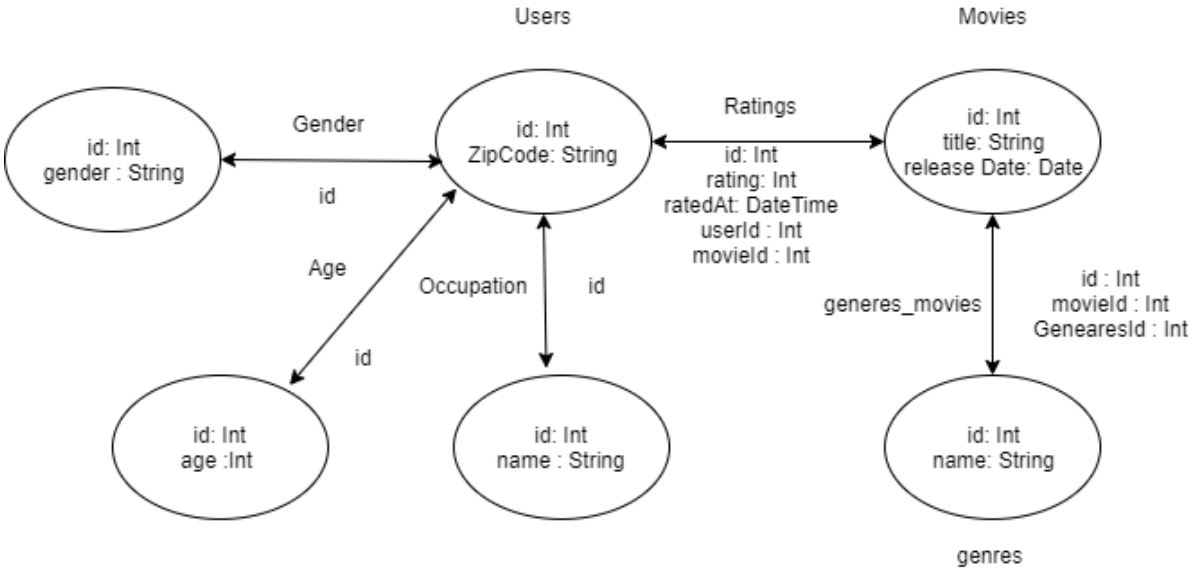


Figure 2. Generated Graph Schema for Movielens Dataset.

 ISBN: 978-1-61208-790-0

constraints are handled in the software application layers instead of the database. Also, in some cases, for the sake of faster insertion into the database, some transactional tables do not enforce normalisation principles. However, if data is not in 5^{th} normal form, then there are well-understood techniques through which the database design can be adjusted so that it is in 5^{th} normal form[14][15]. In this paper, in the context of the proposed mechanism for translating a relational database into a graph database, it is assumed that the input relational database is already in 5^{th} normal form. The reason that the relational database schema *S* is required to be in 5^{th} normal form is because, for any lower normalisation, there may exist a relation in which all columns could form a composite primary key; in other words, there will be no non-key columns. When migrating such a relational database to a graph database format, each relational row will be recorded as a new vertex, thus unnecessarily increasing the number of vertices and hence slowing down the resolution of any query that may be directed at the graph databases.

IV. MOVIELENS EXAMPLE

The Movielens database [16] is a popular choice for the study of collaborative filtering algorithms. MovieLens is a Web-based recommender system that can be used to recommend movies to its members according to their film preferences. It operates by applying collaborative filtering to its members' movie ratings and reviews. It contains some 11 million reviews for some 8500 movies. *Ratings* are expressed using the numeric range 1 to 5. For the evaluation presented in the following section, $100{,}000$ ratings, provided by 943 members with respect to 1682 movies, were used. Figure 1 shows the entity relationship diagram for the Movielens database. It shows the relational database in 5^{th} normal form. From figure1, it can be seen that, for the *User* and *Occupation* tables, Rule 1 will be applicable, because the *Users* table has only one foreign key. As a result, all rows within the *User* table become nodes in the graph that have edges linking to occupations with `occupation_id` as the edge property. Rule 2 is applicable to the *Ratings* table which features two foreign keys, therefore all user nodes have edges to movies nodes, and all the attributes of *Ratings* are set as properties of the edge between users and movies nodes. If $T = 0.1$ is assumed, then *age* and *gender* of *users* falls within the threshold T and, thus, are allocated generated Ids, taken out of the users node and placed in their own new nodes with a link to the *users* node. Figure 2 shows the resulting graph scheme, GS, after the proposed automated translation has been applied.

V. EXPERIMENTAL RESULTS

The performance evaluation of the proposed approach was conducted by comparing it against an approach founded on *Neo4j* (Neo4j ETL) which adopted the first three steps defined in Section III. An implementation of Quantitative Association Rules (QAR) [13] for recommendation engines was used with respect to cold start users. The QAR-based implementation only had limited information like *age*, *gender*, *zipcode* and *occupation* for cold start users. Based on this information, the graph database could be queried. A recommendation engine, written in Scala, was used to make recommendations according to the similarity between users based on the information provided. Experiments were conducted using five fold cross validation with 80% of users as the training data set and the remaining 20% as the test set. Both approaches produce a success rate of 67% for the recommended movies. The proposed approach, referred to as the *Optimised Collaborative Filtering (OCF)* approach, was faster by a substantial margin, as shown in Figure 3. The same results in terms of recommendation quality were produced by both approaches. However, the advantage offered by the proposed *OCF* approach was that it was more efficient in that it only selected a handful of nodes within the graph database to find similar nodes according to the input information, as opposed to scanning all user nodes and then filtering according to the input information. All property graphs can find a node in a constant time given its Id, resulting in a significant speed-up advantage for the proposed method with respect to recommender query resolution.

To further strengthen the idea, a number of example queries were executed to find the execution time taken to search the graph databases according to the given information for a simulated new MovieLens member. The results with respect to five of these queries are shown in Figure 4. With reference to the

 ISBN: 978-1-61208-790-0

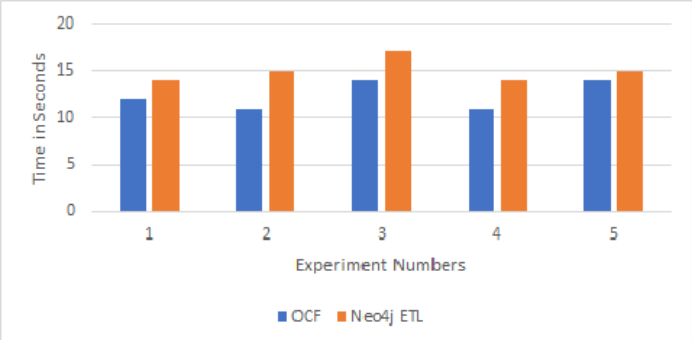


Figure 3. Five fold cross validation results comparing OCF operation with Neo4j ETL.

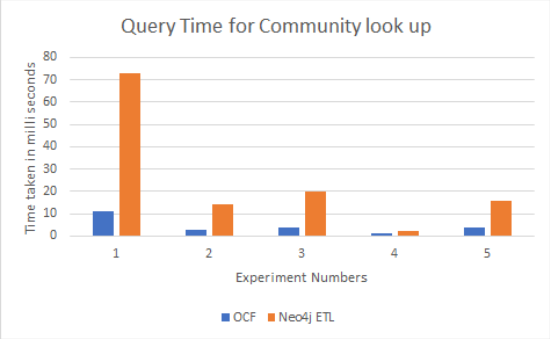


Figure 4. Runtime for five example community detection queries comparing OCF operation with Neo4j ETL.

figure, it should be noted that each "bin" represents a separate graph query. The aim was to find all users which matched the criteria for the specified member. Of course, for comparison purposes, the queries had to be expressed differently because the underlying graph schemas were different, however, the end result was the same. From the figure, it can be seen that the proposed *OCF* approach provided significant efficiency gains over the *Neo4j ETL* approach. It should also be noted that the efficiency gains were more marked when larger datasets were returned as a consequence of the query resolution.

VI. Conclusions and Further Suggestions

In this paper, an approach to the automated generation of a graph database from a given relational database has been described. The proposed approach operates in a more sophisticated manner than earlier approaches. Compared to an alternative current approach, the *Neo4j ETL* approach, the proposed approach operates much more efficiently while producing the same outcomes. This approach can be extended and tried with further splitting up the properties of edges into separate vertices if the edge property conforms to a particular threshold set in the same manner as for the proposed approach to vertex properties. The results can also be tried on distributed graph databases to reinforce the results presented in this paper.

References

[1] D. Fleder and K. Hosanagar, "Blockbuster culture's next rise or fall: The impact of recommender systems on sales diversity." *Management Science*, vol. 55, no. 5, pp. 697–712, 2009. [Online]. Available: https://pubsonline.informs.org/doi/10.1287/mnsc.1080.0974

[2] X. N. Lam, T. Vu, T. D. Le, and A. D. Duong, "Addressing cold-start problem in recommendation systems," in *Proceedings of the 2nd International Conference on Ubiquitous Information Management and Communication*, ser. ICUIMC '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 208âĂŞ211. [Online]. Available: https://doi.org/10.1145/1352793.1352837

[3] N. Mirbakhsh and C. X. Ling, "Improving top-n recommendation for cold-start users via cross-domain information," *ACM Trans.*

 ISBN: 978-1-61208-790-0

Knowl. Discov. Data, vol. 9, no. 4, Jun. 2015. [Online]. Available: https://doi.org/10.1145/2724720

[4] J. Zhu, J. Zhang, C. Zhang, Q. Wu, Y. Jia, B. Zhou, and P. S. Yu, "Chrs: Cold start recommendation across multiple heterogeneous information networks," *IEEE Access*, vol. 5, pp. 15 283–15 299, 2017.

[5] R. D. Virgilio, A. Maccioni, and R. Torlone, "Converting relational to graph databases." *Graph Data Management Experiences and Systems*, pp. 1–6, 2013. [Online]. Available: https://dl.acm.org/doi/10.1145/2484425.2484426

[6] J. J. Miller, "Graph database applications and concepts with neo4j," in *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, vol. 2324, no. 36, 2013.

[7] R. Angles, "The property graph database model." in *AMW*, 2018.

[8] "Orient db, property graph model." [Online]. Available: https://orientdb.org/docs/3.0.x/datamodeling/Tutorial-Document-and-graph-model.html

[9] D. W. Wardani and J. Kiing, "Semantic mapping relational to graph model." in *Proceeding - 2014 International Conference on Computer, Control, Informatics and Its Applications, "New Challenges and Opportunities in Big Data", IC3INA 2014*, Sebelas Maret University, 2014, pp. 160–165. [Online]. Available: https://ieeexplore.ieee.org/document/7042620

[10] S. Lee, B. H. Park, S. Lim, and M. Shankar, "Table2graph: A scalable graph construction from relational tables using map-reduce," in *IEEE First International Conference on Big Data Computing Service and Applications*, 2015, pp. 294–301.

[11] S. P. L. Filho, M. C. Cavalcanti, and C. M. Justel, "Graph modeling for topological data analysis." *Enterprise Information Systems*, pp. 193–214, 2019. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-26169-6_10

[12] J. Gop and S. Jain, "A survey on solving cold start problem in recommender systems." in *Proceeding - IEEE International Conference on Computing, Communication and Automation, ICCCA 2017*, vol. 2017-January, no. Proceeding - IEEE International Conference on Computing, Communication and Automation, ICCCA 2017, Department of Computer Engineering, National Institute of Technology, 2017, pp. 133–138. [Online]. Available: https://ieeexplore.ieee.org/document/8229786

[13] S. Tyagi and K. K. Bharadwaj, "Enhanced new user recommendations based on quantitative association rule mining," *Procedia Computer Science*, vol. 10, pp. 102 – 109, 2012, aNT 2012 and MobiWIS 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050912003742

[14] M. L. Wilson, "A requirements and design aid for relational data bases," in *Proceedings of the 5th International Conference on Software Engineering*, ser. ICSE '81. IEEE Press, 1981, p. 283âĂŞ293.

[15] B. Lira, A. Cavalcanti, and A. Sampaio, "Automation of a normal form reduction strategy for object-oriented programming," in *Proceedings of the 5th Brazilian workshop on formal methods*, 2002, pp. 193–208.

[16] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context." *ACM Transactions on Interactive Intelligent Systems*, no. 4, 2015. [Online]. Available: https://dl.acm.org/doi/10.1145/2827872

 ISBN: 978-1-61208-790-0

Reconsidering Optimistic Algorithms for Relational DBMS

Malcolm Crowe
School of Computing, Engineering and Physical Sciences
University of the West of Scotland
Paisley, UK
e-mail: malcolm.crowe@uws.ac.uk

Fritz Laux
Department of Informatics
Reutlingen University
Reutlingen, Germany
e-mail: fritz.laux@fh-reutlingen.de

***Abstract*—At DBKDA 2019, we demonstrated that StrongDBMS with simple but rigorous optimistic algorithms, provides better performance in situations of high concurrency than major commercial database management systems (DBMS). The demonstration was convincing but the reasons for its success were not fully analysed. There is a brief account of the results below. In this short contribution, we wish to discuss the reasons for the results. The analysis leads to a strong criticism of all DBMS algorithms based on locking, and based on these results, it is not fanciful to suggest that it is time to re-engineer existing DBMS.**

Keywords - transactions; concurrency; optimistic.

I. INTRODUCTION

While the Standard Query Language (SQL) standard [9] famously describes the well-known four transaction levels of read uncommitted, read committed, repeatable read, and serializable, it wisely does not mandate any particular strategy for ensuring correct transaction behaviour, as explained in Note 47 [9]. However, all commercial database management systems (DBMS) use locking to ensure correct transactional behaviour in the face of concurrent accesses to a database.

This approach, with the attendant use of pessimistic concurrency algorithms, may have seemed attractive in 1974, and is still the easiest to explain. If the client has acquired locks on all the data it needs, it appears that a successful commit can be guaranteed. However, if the client and server are communicating over a network, the Consistency-Availability-Partition tolerance (CAP) theorem and the two-army thought experiment both demonstrate that the success of the commit may be indefinitely delayed unless the client's locks are overridden. To these theoretical objections two practical considerations can be added, first, that locking systems are complex, so that deadlocks are almost unavoidable, and, second, that client-side locks are subject to timeout. As a result, the apparent guarantee of success does not work well over the internet where interactive clients expect to have a comparatively long time to complete a transaction.

In practice, many software developers instead use application-level protocols to provide optimistic concurrency for distributed applications communicating with web-servers that handle all access to the database. The resulting mismatch of concurrency strategies between application and database has led to middleware trying to provide a concurrency mechanism that is more application affine and abstract from the database provided concurrency control (e.g., see [2, 3, 14]). But, far from solving the problem of transaction coordination, this only compounds the problem by adding another competing source of persistence, and the difference in approach to concurrency does not help. It becomes natural to ask whether the database server itself should also use optimistic algorithms for concurrency control

The significance of the StrongDBMS [7] demonstration [1][15] was that its optimistic algorithms were extremely simple and startlingly effective in providing fully serializable transactions under conditions of high data conflict. The experiment was set up so that correct operation would necessitate most transactions failing to commit, but much greater overall throughput resulted from StrongDBMS' optimistic operation. StrongDBMS's transaction log demonstrated that all committed transactions had been serialised, despite the large number of overlapping long transactions.

The implementation of StrongDBMS was also interesting in featuring the use of immutable data structures, and it seems plausible that all the usual DBMS features could be implemented using this approach. Work has been progressing since DBKDA 2019 to achieve this by modifying the existing PyrrhoDB to use a similar architecture to StrongDBMS.

The structure of this paper is as follows. Section 2 contains an analysis of the reasons for the demonstrated differences in performance between optimistic DBMS (such as StrongDBMS and PyrrhoDB) and other systems. Section 3 explains some minor departures from standard SQL semantics in the test. Section 4 discusses the details of the modified benchmark test used in the demonstration. Section 5 presents a synopsis of the test results. Finally, Section 6 summarises the conclusions of this study.

II. CONFLICT DETECTION AND ROLLBACK

The essential point of optimistic transactions is that conflicts are detected only at the end of the transaction when commit is attempted. At this point, if it is found that conflicts have occurred, the commit will fail, and none of the transaction's work will be written to the database.

This approach is sometimes called First Committer Wins (FCW). It has the advantage that short transactions are more likely to succeed. In the literature [4, page 170], it has been

 ISBN: 978-1-61208-790-0

assumed that FCW systems would have high validation costs or reduced throughput because of unnecessary rollbacks that would occur if the check includes only 'dangerous structures' [5]. But the demonstration showed that, when combined with optimistic execution, throughput was enhanced through use of FCW. Some database textbooks suggest that optimistic execution is inherently less effective than the usual locking-based approach when load is high, but this is now seen to be another myth. In the rare situation where transactions access the same data (hot spot), it might be possible that a transaction is repeatedly aborted (starving problem).

III. SOME DEROGATIONS

For simplicity, we focus exclusively on SERIALIZABLE transactions. It seems worthwhile here to explain other technical respects in which the implementations depart from the standard description. The standard stipulates that all changes made on commit are accessible to concurrent transactions. We interpret this as excluding concurrent *serializable* transactions, as it is more natural that a serialisable transaction continues to see the database as it stood at the time the transaction started ("snapshot isolation"), apart from the changes it is making. In the case that the transaction does not intend to commit changes, it is intrusive to advise on changes that other users have made.

It is well known that snapshot isolation is insufficient to ensure consistency [6]. Even optimistic algorithms need to lock the database *during commit* while the transaction is checked for conflicts. This however is quite different from acquiring locks at an earlier stage in the transaction.

One further simplification in our work is always to enforce constraints and integrity checks . For example, the "no action" options are disallowed for referential constraints. This ensures that the database is kept in a consistent state even after each step in a schedule. For constraints that cannot be satisfied with one SQL-statement, our chosen solution is to allow deferral of triggers to the end of a transaction.

IV. THE CASE STUDY

The demonstration of StrongDBMS used the Transaction Processing Council Benchmark C (TPC-C) [13] with a modification to create high levels of data conflict between clerks who enter new orders for a warehouse.

To begin with, the TPC-C benchmark normally has 1 clerk per warehouse, so that the conflict rate is around 4%. In the reported tests, we deliberately increased the concurrency challenge by using multiple clerks for a single warehouse. When the number of clerks goes above 10, most New Order tasks will fail with a write-write conflict on the next order number for the district (NEXT_O_ID) as there are only 10 districts. Worse, the single row in the WAREHOUSE table contains a running total for the year (W_YTD), which is updated by the payment task, and fields from this row are read by all the NewOrder tasks and others so that a great many more tasks are aborted because of read/write conflicts. In all the products tested, apart from PyrrhoDB and StrongDBMS, read/write conflicts are detected at the row level or wider.

Both PyrrhoDB and StrongDBMS see no conflict between the Payment and NewOrder task because Payment is the only task that accesses W_YTD, and one of the available tests in the ReadConstraint for detecting read/write conflicts is a set of fields in a specific single row of a table.

There are actually three levels of read/write conflict detection in these DBMS. The following comment in the source code for Read Set dates from about 2005 [8] (tb refers to the base table affected):

"ReadConstraints record all of the objects that have been accessed in the current transaction so that this transaction will conflict with a transaction that changes any of them. However, for records in a table, we allow specific non-conflicting updates, as follows:

"(a) (CheckUpdate) If unique selection of specific records cannot be guaranteed, then we should report conflict if any column read is updated by another transaction.

"(b) (CheckSpecific) If we are sure the transaction has seen a small number of records of tb, selected by specific values of the primary or other unique key, then we can limit the conflict check to updates of the selected records (if any), or to updates of the key TableColumns.

"(c) (BlockUpdate) as (a) but it is known that case (b) cannot apply."

If the isolation level is reduced to repeatable-read or read-committed, most of the competing products achieve performance comparable with Pyrrho and StrongDBMS. However, there is a risk that the database may show wrong results or an inconsistent state. This is what we found for a commercial product.

The use of escrow methods [11][12] could avoid hot spot conflicts like in NEXT_O_ID (resp. W_YTD) for many DBMS if the semantics is known, e.g., an increment semantic (resp. commutative semantics). Laiho and Laux [10] also developed a method of using row-versioning to ensure correct non-blocking operation of distributed applications. Both these approaches require changes to the application protocols, but they can be used with existing commercial DBMS products.

V. THE BENCHMARK RESULTS

The TPC-C benchmark simulates a telephone-based order entry system for 100000 products where each warehouse has 30000 customers assigned to 10 districts. There is one clerk per warehouse, and the simulation includes a randomised set of tasks with time-delays so that a realistic work rate for the clerk is simulated, allowing the clerk to process 16 new orders in 10 minutes: each order has between 5 and 15 lines. There is some scope for concurrency verification for the DBMS, as items can be supplied from other warehouses, and the specification results in about 4% of conflicting transactions.

We adapted this test by providing multiple clerks for a single warehouse, and then the database design results in much higher levels of conflict as described above. In the 10-minute experiments, the maximum number of new orders per clerk remains 16, but the actual throughput will be much less owing to transaction conflict. DBMS generally allow a range of transaction isolation levels. From the viewpoint of this

 ISBN: 978-1-61208-790-0

paper, the interesting results are for SERIALIZABLE transactions only.

The initial state of the database, and the details of what the tasks involve, are specified in great detail on the TPC-C website. In simple terms, each task requires committing some changes to the database. Many of the tasks perform a single insert or update on a single table. The commit for the new order task inserts new rows in HISTORY, ORDER and ORDER_LINE (5 to 15 order lines per order) and updates WAREHOUSE, DISTRICT, CUSTOMER and 5 to 15 rows in STOCK. All the updates involved in a new order have a good chance of conflict since there is only 1 warehouse and 10 districts. There is a smaller chance of conflict on STOCK and CUSTOMER since there are more of these. The distinction between ORDER and NEW_ORDER is that customers are expected to pay for completed ORDERS, and NEW_ORDERS require delivery. In the 10 minute test, the delivery for a NEW_ORDER might be scheduled but won't complete.

For StrongDBMS, we found the behaviour shown in Table I. This shows 241 (= 30241 - 30000) new orders for 30 clerks, and also indicates the reported number of failed transactions (="Exceptions").

TABLE I. RESULTS FOR STRONGDBMS

Name	Initial	1 clerk	10 clerks	20 clerks	30 clerks
Commits	0	39	302	512	565
Exceptions	0	0	104	387	1071
ORDER	30000	30016	30138	30199	30241
NEW_ORDER	9000	9016	9138	9199	9241
ORDER_LINE	285007	285158	286207	286638	286965
DELIVERY	0	1	13	22	32

A major commercial DBMS, using serializable transaction isolation, completed only 132 NEW_ORDERS for 30 clerks, as shown in Table II.

TABLE II. RESULTS FOR COMMERCIAL DEBMS (SERIALIZABLE,USING 2PL)

Name	Initial	1 clerk	10 clerks	20 clerks	30 clerks
Commits	0	41	211	276	290
Exceptions	0	0	43	132	213
ORDER	30000	30016	30111	30127	30132
NEW_ORDER	9000	9016	9111	9127	9132
ORDER_LINE	285007	285158	286114	286223	286295
DELIVERY	0	1	12	18	18

The commercial DBMS frequently aborted the transaction with a report of deadlock, without attempting to commit.

Some investigation took place on using other isolation levels and other DBMS. These tests are reproducible, and versions of the software for several major commercial DBMS are available on the GitHub website [16]. However, this software is implemented with a thread for each clerk with its own database connection, and in some cases this seemed to result in the DBMS erroneously reporting that transactions were being nested, or already completed.

Callum Fyffe continued the tests for StrongDBMS to over 100 clerks [14], and while the numbers continued to rise, eventually the results became less reproducible as the operating system intervened to deal with memory saturation. Our collected results for SERIALIZABLE isolation are shown in Table III, where the asterisks indicate that further tests were not carried out owing to reducing throughput.

TABLE III. FURTHER RESULTS

Name	1 clerk	2	5	10	20	30	40	50	60
StrongDBMS laptop	16			138	199	241	*		
StrongDBMS 16GB RAM	16			129	220	254	409	331	328
Commercial 1	16			111	127	132	16	*	
Commercial2	16			107	114	119	124	117	*
Commercial3	16	33	69	6	*				

Figure 1 shows the comparable results from Table III as a chart.

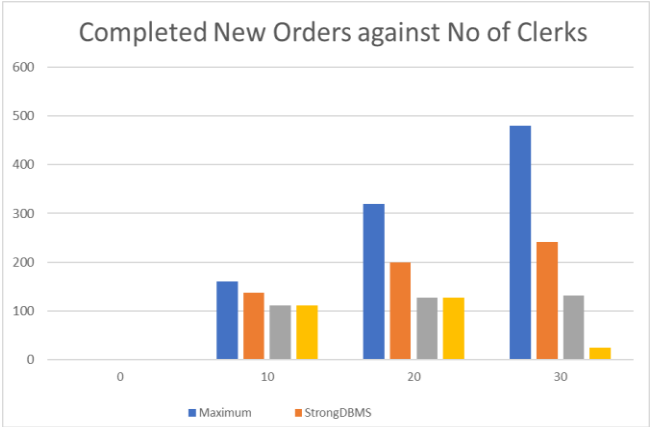


Figure 1. Comparable test results. The first bar in each group shows the maximum possible (16x number of clerks), and the second is StrongDBMS.

VI. CONCLUSIONS

The study reported here makes a case for extending optimistic algorithms to other database products. This would provide a radical and welcome way of removing the "impedance mismatch" between application and DBMS protocols. Myths about such algorithms are deeply entrenched in the database community, but it is time for better and more considered analysis.

 ISBN: 978-1-61208-790-0

REFERENCES

[1] M. Crowe and C. Fyffe, “Benchmarking StrongDBMS”, Keynote speech, DBKDA 2019, https://www.iaria.org/conferences2019/filesDBKDA19/MalcolmCrowe_CallumFyffee_Keynote_BenchmarkingStrongDBMS.pdf [retrieved: April, 2020]

[2] Java Platform, Enterprise Edition: The Java EE Tutorial, “Concurrency Utilities for Java EE” https://docs.oracle.com/javaee/7/tutorial/concurrency-utilities.htm [retrieved: April, 2020]

[3] A. Harrer, T. Irgang, N. Sattes, and K. Pfahler, “SoCCR – Optimistic Concurrency Control for the Web-Based Collaborative Framework Metafora”, Proceedings of the 18th International Conference, CRIWG 2012, Raesfeld, Germany, pp. 153-160, 2012

[4] G. Weikum and G. Vossen, Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery, Morgan Kaufmann, 2002

[5] M. Cahill, U. Röhm, and A. Fekete, “Serializable isolation for snapshot databases”, ACM Trans. Database Syst., 34(4), pp. 20:1-20:42, 2009

[6] H. Berenson, P. Berenson, J. Gray, J. Melton, W. O’Neil, and P. O‘Neil, “A Critique of ANSI SQL Isolation Levels”, Microsoft Research, 1995 Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA, pp. 1 - 10, 1995

[7] M. Crowe, S. Matalonga, and M. Laiho, “StrongDBMS: Built from Immutable Components.” *The Eleventh International Conference on Advances in Databases, Knowledge, and Data Applications*. ThinkMind, pp. 11-16, 2019

[8] M. Crowe, *An introduction to the source code of the Pyrrho DBMS*. University of Paisley, 2007.

[9] International Standards Organisation: latest edition ISO/IEC 9075-2:2016/Cor 1:2019

[10] M. Laiho and F. Laux, “Implementing Optimistic Concurrency control for persistence Middleware using row version verification.” *2010 Second International Conference on Advances in Databases, Knowledge, and Data Applications*. IEEE, pp. 45-50, 2010

[11] F. Laux and T. Lessner, “Escrow serializability and reconciliation in mobile computing using semantic properties.” *International Journal on Advances in Telecommunications* 2.2, pp. 72-87, 2009

[12] F. Laux and T. Lessner, “Transaction processing in mobile computing using semantic properties.” *2009 First International Conference on Advances in Databases, Knowledge, and Data Applications*. IEEE, pp. 87-94, 2009

[13] F. Raab, W. Kohler, and A. Shah, “Overview of the TPC-C benchmark: The Order-Entry Benchmark”, Transaction Processing Performance Council, Tech. Rep., 2013, http://www.tpc.org/tpcc/detail.asp [retrieved: April, 2020]

[14] MS Azure Cosmos DB, Transactions and optimistic concurrency control (12/04/2019), https://docs.microsoft.com/en-us/azure/cosmos-db/database-transactions-optimistic-concurrency [retrieved: April, 2020]

[15] C. Fyffe, “Benchmarking StrongDBMS”, MSc Thesis, University of the West of Scotland, 2019/ Available in [16]

[16] GitHub, https://github.com/MalcolmCrowe/ShareableDataStructures [retrieved: September 2020]

 ISBN: 978-1-61208-790-0

Comparative Analysis of RDBMS and NoSQL Databases

Jam Jahanzeb Khan Behan
Free University of Brussels
Bruxelles, Belgium
Email: jbehan@ulb.ac.be

Ali Inam
Institute of Business Administration
Karachi, Pakistan
Email: ali.inam03@gmail.com

Meesum Ali
Institute of Business Administration
Karachi, Pakistan
Email: meesumdex@gmail.com

Muhammad Talha Khan
Institute of Business Administration
Karachi, Pakistan
Email: talhakhan298@gmail.com

***Abstract*—Big Data has been the subject of increased research since data has been termed the new oil for the 21st century. Recently, smart grids have been used by energy providers to store the massive amount of data that is generated at regular time intervals. K-Electric is one such company in Pakistan that provides the residents of Karachi City with electrical energy. The company stores their data in a Not only Structured Query Language (NoSQL) database, since the smart grid data has a high volume, accelerated velocity, and tremendous variety. Hence, we feel that we can provide an important comparison of NoSQL tools using this data. NoSQL tools have been actively used for storage purposes in the industry. Companies like eBay, GitHub, and Amazon have been using these tools for storage and analytical purposes alike. In this paper, we compare and analyze four different technologies: MySQL, MongoDB, MonetDB, and InfluxDB using the data generated by the smart grids of K-Electric.**

Keywords–NoSQL; Big Data; RDBMS; Performance Comparison; Smart Grid.

I. INTRODUCTION

A smart grid is an electrical grid that provides a variety of operations and energy measures. These measures can include smart meters, smart appliances, renewable energy resources, and energy-efficient resources. The most important aspects of the smart grid are electronic power conditioning, control of the production, and distribution of electricity.

The Big Data phenomenon is defined using 3 Vs, where we have too much data (volume) that is being collected at an extremely high rate (velocity) and contains mostly unstructured data (variety) [1] [2]. Traditionally, data has been managed and stored in Relational Database Management Systems (RDBMSs) with the focus to optimize the storage space. However, querying is a time consuming task in these traditional RDBMS technologies. In RDBMS, the data is distributed in different tables, and then these tables are virtually joined for performing advanced querying, hence the slow response time. However, with the sudden explosion of data, due to the Web and data accessibility, the old technologies could not handle the increasing demand for data storage and querying. Unfortunately, since this amount of data is not manageable by traditional RDBMS technologies, we witnessed the rise of NoSQL databases. These new technologies have been used to analyze Big Data to reveal new insights and optimize the decision making strategy for executives. As of present, there are more than 225 NoSQL databases [3]–[7].

A database system that is distributed does not require a fixed table schema. The schema is mostly built at runtime based on the query. As there is no schema, (i) the join operations are usually avoided, (ii) the technology can be scaled horizontally, (iii) the system does not expose a Structured Query Language (SQL) interface and (iv) the tool can be open source [8]. However, even though the NoSQL databases are the by-products of the Web 2.0 era, these tools were solely used when the Web service providers had a large number of users. These providers discovered that the RDBMS can be used either when the database is small but requires frequent read & write transactions, or when the database is large but requires batch transactions while rarely needing write transactions. They concluded that RDBMS cannot be used for large databases with heavy read & write workloads [5].

In this paper, we aim to use the data that is stored in the RDBMS and see how well it can be analyzed using a NoSQL system. The data is collected from K-Electric, a vertically integrated investor-owned utility company managing the generation, transmission, and distribution of energy to consumers. The purpose of this paper is to analyze the performance of K-Electric's relational data in a non-relational environment.

The rest of the paper is structured as follows: Section II highlights some of the related work done on comparing RDBMS technologies with NoSQL technologies. In Section III, we provide a detailed account of the technologies we have selected for our experiments. Section IV briefly outlines the structure of the data obtained from K-Electric. Section V explains the technical setup for the technologies, the experiments that we have performed, and the results of these experiments. Finally, we give our concluding remarks in Section VI.

II. RELATED WORK

In this section, we highlight other works that have aimed at comparing NoSQL databases.

Hadjigeorgiou et al. [9] have compared the performance of MongoDB and MySQL when they are scaled and sharded. The metrics they have used are (i) total queries per second and (ii) total queries per second per thread. The authors tested the systems on a dataset related to the music industry. Firstly, they make different schemas for the RDBMS and for the NoSQL systems. Metrics are recorded for three experiments that are done using (i) a single node, (ii) multiple nodes, and (iii) sharding. The authors conclude that the most important factor was the query type used since MongoDB was able to handle more complex queries faster, due mainly to its simpler schema while having to duplicate the data. MongoDB also performed better during insertions. They also state that MySQL

 ISBN: 978-1-61208-790-0

performs better when deleting data since it performs better in simple search queries. This might be the case because deletion requires finding the record to be deleted first, which is easier in MySQL since there is only one instance. Finally, the authors highlight that both databases have had a linear trend in the benchmarks.

Ansari et al. [10] have selected Hbase, MongoDB, Cassandra, and Elasticsearch NoSQL technologies and compared them using data from smart grids. The smart grid meter data that they were using was structural column-based data. For experimentation purposes, they used the default configuration of the respective NoSQL technology. They compared the databases on effectiveness (using the WRITE and READ parameters) and scalability (by measuring the execution performance of the full mechanism). The results showed that Cassandra had the smallest average latency in both read and write processes. This is possible because Cassandra is one of the best column-based databases and the data to be evaluated was column-based data.

Venkatraman et al. [11] discussed the four main data models of non-relational databases and compared them to SQL databases. They first presented the context of Big Data analytics and NoSQL databases and then compared them based on high availability, partition tolerance, high scalability, consistency, auto-sharding, write frequently & read less (priority is given to write operations as compared to read operations), fault tolerance (no single point of failure), multiversion concurrency control (MVCC), and, finally, concurrency control (locks). The authors performed benchmark tests, however, they did not provide the results. The authors only discussed and explained the results. They state that Couchbase processes more operations per second with lower average latency in reading and writing data than both MongoDB and Cassandra. Also, Cassandra is faster in writing than MongoDB, however, both have almost equal reading speed. The authors conclude that the flexible data modeling of NoSQL is well suited to support dynamic scalability and improved performance for Big Data analytics.

Santos et al. [12] have used Geographic Information Systems (GIS) data to compare PostGIS (a spatial database extender for PostgreSQL object-relational database), MongoDB, and Neo4j with Neo4j-Spatial. For comparison purposes, the authors have performed different types of operations (read, write, etc.), where each operation contains a group of queries. Even though all groups include 20 parameterized queries, the parameter values vary within predefined ranges for each group. The data comparison metrics used are (i) Nearby Points of Interest Radius and K-Nearest Neighbors (KNNs), (ii) Urban Routing, (iii) Map View, and (iv) Position Tracking. In the conclusion, the authors have highlighted that, since the spatial attributes are much more complex to handle as compared to strings, numbers, and other relational data types, evaluating and benchmarking spatial DBMS performances is not as simple as doing so in RDBMS. The authors also state that there was a need for data heterogeneity within the same RDBMS, as each type of query runs faster in a different data structure.

III. Selected Technologies

In this paper, we have selected three NoSQL technologies to compare against MySQL [13], the RDBMS technology in place at K-Electric. We have selected MongoDB [14], MonetDB [15], and InfluxDB [16] as the NoSQL datastores.

A. MongoDB

In MongoDB, the data is stored in flexible, JavaScript Object Notation (JSON) like documents, where fields can vary from document to document and data structure can be changed over time. The document model maps to the objects in the application code, making data easy to work with. Ad hoc queries, indexing, and real-time aggregation provide powerful ways to access and analyze the data. It is a distributed database, so high availability, horizontal scaling, and geographic distribution are built-in and easy to use while providing querying and indexing functionalities. Furthermore, MongoDB is an open-source project, hence, aiding in its popularity of use. We have selected MongoDB because it contains the best mixture obtained from RDBMS and NoSQL technologies, which in turn enables users to build new applications. It provides the data model flexibility, elastic scalability, and high performance of NoSQL databases, hence aiding in a continuous enhancement of applications, while scaling on commodity hardware [17].

B. InfluxDB

We have selected InfluxDB because it is an open-source time-series database that is optimized for fast, high-availability storage, and retrieval of time series data. It has no external dependencies and provides an SQL-like language with built-in time-centric functions for querying. Each point consists of several key-value pairs called the fieldset and a timestamp. A series is defined when a set of key-value pairs are grouped together. Finally, series are grouped together by a string identifier to form a measurement. Points are indexed by their time and tagset. Retention policies are defined on measurement and control of how data is downsampled and deleted. Continuous queries run periodically, storing results in a target measurement.

C. MonetDB

MonetDB is an open-source column-oriented database management system designed to provide high performance on complex queries against large databases, such as combining tables with hundreds of columns and millions of rows. Its architecture is represented in three layers, each with its own set of optimizers. The front-end provides a query interface for SQL, where queries are parsed into domain-specific representations, like relational algebra for SQL, and optimized. The generated logical execution plans are then translated into instructions, which are passed to the next layer. The middle or back-end layer provides a number of cost-based optimizers. The bottom layer is the database kernel, which provides access to the data stored in Binary Association Tables, where each table consists of an Object-identifier and value columns, representing a single column in the database. Internal data representation also relies on the memory addressing ranges of contemporary CPUs using demand paging of memory-mapped files and, thus, departing from traditional DBMS designs involving complex management of large data stores in limited memory. We have selected MonetDB because it has been designed to provide high performance on complex queries against large databases and also because it has been applied in high-performance applications for better analytics.

 ISBN: 978-1-61208-790-0

IV. DATA

The data comprises of meter readings from over 9000 smart meters spread throughout Karachi, Pakistan. Based on the type of the meter installed, the data is generated at different intervals. These smart meters are installed at consumer sites, on Pole Mounted Transformers (PMTs), and on distribution feeders. The data is initially stored in an internal buffer, of each respective meter. The device then communicates with the server, based on configurable intervals (using the push/pull protocol). In case of any communication lapse, the infrastructure is designed to record the lost data over a period of seven days. The data is recorded in Head End, the objective of which is to acquire meter data and monitor device parameters automatically, thus avoiding any human intervention.

The data being utilized for this project is collected over a period of three months, with an uncompressed size of approximately 45 GB. The devices installed at the consumers end generate data over a 30 minutes interval, while the devices placed on distribution assets generate data after every 15 minutes. As a first step, we aimed to understand the data on hand by (i) manually looking at a smaller chunk of manageable data and by (ii) asking the domain experts. We also had regular meetings with the employees of K-Electric, who provided an extensive explanation regarding the data: what each field corresponded to, how a particular field is important for further processing, the type of values that each field contains, and which fields were of high importance.

Once the data was analyzed, we gained the understanding of the fields provided in the data. The details of the fields are stated in Table I.

TABLE I. DATA FIELD DESCRIPTIONS

Field name	Field definition
DeviceID	The unique meter identification ID
Time	The time at which the reading was recorded at
Date	The date on which the reading was recorded at in MM/DD/YYYY format
Value	The profile value we obtain against the corresponding Result-TypeID
MeasuredUnit	The unit of measurement we obtain after multiplying Value with the number (10$\hat{}$caler)
Scaler	Represents the number (10$\hat{}$caler) to be multiplied with the Value to get the Value measured according to the units in MeasuredUnit
ResultTypeID	The profile for which the value has been generated.
Status	This is a 32 bit number to represent the status of the meter itself
Description	The description of the DeviceID. Not properly maintained

We imported the original data from the databases to use the data for querying purposes and then evaluate the query execution times. Some fields have been highlighted as an essential part of the analysis. However, we omitted the fields: **Description**, **Status**, **MeasuredUnit**, and **Scalar** since these columns did not provide any information relevant to our analysis. Moreover, a new field by the name of **Timestamp** was created by concatenation of the Time and Date fields.

V. EXPERIMENTATION

In this section, we provide the technical details for each of the selected technology and how they were set up. Also, in this section, we provide the queries that have been devised for comparison purposes, the benchmark we obtained while using MySQL (since it is the main technology at K-Electric), and how the other tools performed as compared to the results of MySQL.

A. Technical Details

Since we wanted to work independently, that is without the restriction of having to carry the data, or the setup, we decided to use Amazon Web Services (AWS). To setup the environment, we created instances (not a VM environment) of each of the four technologies. We also had to make customized adjustments to some of the databases instances, and the details are as follows:

MySQL: As stated in the previous section, the database deployed at K-Electric is MySQL and, for our purpose, we created an AWS instance for MySQL and accessed that instance by means of MySQL Workbench on our personal computers. To enable working on the data in RDBMS, we intially required a schema of the data and store data in form of tables. Fortunately, K-Electric stored the data into one huge table–and we, therefore, kept our own schema in accordance to that. For our instance, we stored the data in a table named "dataset".

MongoDB: In MongoDB every dataset is a collection, and we can query each collection using their keys. For the experimentation, we created a collection called "SM_RECS". Furthermore, we created custom indexes on two attributes: DateTime and ID. We would like to mention, due to its nature of complexity, we decided to opt out of the UNION queries in MongoDB.

MonetDB: We followed the same procedure as that of MySQL and created a single table called "dataset".

InfluxDB: For InfluxDB, we stored our data in a table named "dataset" and, after much searching, we found out that InfluxDB does not, in fact, have native support for UNION. Hence, we were unable to perform the UNION queries [18].

B. Queries

We have written queries of different categories for each database (see Table III) and ran them on the AWS instances. The queries belong to one of the following categories:

1) Simple Query
2) Range Query
3) Aggregated Query
4) Nested Query
5) UNION Query

As stated previously, we were unable to perform UNION queries for MongoDB and InfluxDB.

C. Results

To provide an unbiased experimental runtime, the repeated the experiments 10 times. The average time required for experimentation to complete and the results are outlined in Table II. As it can be seen from the results, all the technologies were able to obtain the same results (in terms of the number of records) for identical queries. Hence, we compare the results based on the Runtime(s) columns in Table II that correspond to the total time taken to obtain the results while computing on the given technology instance. As stated previously, the results obtained from MySQL serve as a baseline for the NoSQL technologies, and it is safe to say that all the NoSQL technologies were able to obtain better results than the baseline.

 ISBN: 978-1-61208-790-0

It can be observed that MonetDB was able to outperform MySQL, and was still able to provide results for all categories of queries stated in Section V-B. It is also worth mentioning that InfluxDB outperformed all the systems in terms of computational time. However, since it does not provide UNION query facilities, we cannot rely on this system for being a replacement of the traditional RDBMS.

TABLE II. QUERY RUNTIME AND RESULTS

Query Number	Result	Runtime (s)
MongoDB 1	84965 row(s) returned	0.65
MongoDB 2	781 row(s) returned	57.26
MongoDB 3	1 row(s) returned	0.04
MongoDB 4	697409 row(s) returned	0.88
MongoDB 5	33856 row(s) returned	854.75
MongoDB 6	33856 row(s) returned	874.18
MongoDB 7	0 row(s) returned	1054.73
InfluxDB 1	84965 row(s) returned	1.38
InfluxDB 2	781 row(s) returned	13.55
InfluxDB 3	1 row(s) returned	10.85
InfluxDB 4	697409 row(s) returned	0.08
InfluxDB 5	33856 row(s) returned	15.37
InfluxDB 6	33856 row(s) returned	10.37
InfluxDB 7	0 row(s) returned	58.88
MonetDB 1	84965 row(s) returned	8.54
MonetDB 2	781 row(s) returned	60.46
MonetDB 3	1 row(s) returned	57.19
MonetDB 4	697409 row(s) returned	1.88
MonetDB 5	33856 row(s) returned	78.64
MonetDB 6	33856 row(s) returned	76.63
MonetDB 7	0 row(s) returned	256.15
MonetDB 8	101444 row(s) returned	265.65
MonetDB 9	3 row(s) returned	9054.64
MySQL 1	84965 row(s) returned	134.70
MySQL 2	781 row(s) returned	121.69
MySQL 3	1 row(s) returned	117.76
MySQL 4	697409 row(s) returned	121.81
MySQL 5	33856 row(s) returned	148.98
MySQL 6	33856 row(s) returned	158.33
MySQL 7	0 row(s) returned	451.45
MySQL 8	101444 row(s) returned	473.36
MySQL 9	3 row(s) returned	18184.06

VI. CONCLUSION

In this paper, we have analyzed the data stored in an RDBMS, using NoSQL technologies. However, due to their respective limitations, we were unable to use InfluxDB and MongoDB to their full potential. We have provided a baseline for analyzing smart grid data on NoSQL technologies. In the future, we aim to perform eperimentations on NewSQL [19] technologies to compare the results of RDBMS, NoSQL, and NewSQL on smart grid data.

REFERENCES

[1] Gartner, "Gartner Glossary," https://www.gartner.com/en/information-technology/glossary/big-data, [Online; accessed April 23rd, 2020].

[2] P. Zikopoulos and C. Eaton, Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data. McGraw-Hill Osborne Media, 2011.

[3] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland, "The End of an Architectural Era: It's Time for a Complete Rewrite," in Proceedings of the 33rd International Conference on Very Large Data Bases. VLDB Endowment, 2007, pp. 1150–1160.

[4] A. Reeve, "Big Data and NoSQL: The Problem with Relational Databases," http://infocus. emc. com/april_reeve/big-data-and-nosql-the-problem-with-relationaldatabases/, [Online; accessed April 23rd, 2020].

[5] S. Edlich, "Your Ultimate Guide to the Non-Relational Universe!" http://nosql-database.org/, [Online; accessed April 23rd, 2020].

[6] S. Sagiroglu and D. Sinanc, "Big data: A review," in 2013 International Conference on Collaboration Technologies and Systems (CTS). IEEE, 2013, pp. 42–47.

[7] G. Stevens, "List of Nosql Database Management Systems," http://nosql-database.org/, [Online; accessed April 23rd, 2020].

[8] R. Agrawal et al., "The Claremont Report on Database Research," ACM Sigmod Record, vol. 37, no. 3, 2008, pp. 9–19.

[9] C. Hadjigeorgiou et al., "RDBMS vs NoSQL: Performance and Scaling Comparison," MSc in High, 2013.

[10] M. H. Ansari, V. T. Vakili, and B. Bahrak, "Evaluation of big data frameworks for analysis of smart grids," J. Big Data, vol. 6, 2019, p. 109.

[11] S. Venkatraman, K. Fahd, S. Kaspi, and R. Venkatraman, "SQL versus NoSQL movement with Big Data Analytics," International Journal of Information Technology and Computer Science, vol. 8, no. 12, 2016, pp. 59–66.

[12] P. O. Santos, M. M. Moro, and C. A. D. Jr., "Comparative Performance Evaluation of Relational and NoSQL Databases for Spatial and Mobile Applications," in Database and Expert Systems Applications - 26th International Conference, DEXA 2015, Valencia, Spain, September 1-4, 2015, Proceedings, Part I, ser. Lecture Notes in Computer Science, Q. Chen, A. Hameurlain, F. Toumani, R. R. Wagner, and H. Decker, Eds., vol. 9261. Springer, 2015, pp. 186–200.

[13] Oracle Corporation, "MySQL," https://www.mysql.com/, [Online; accessed April 23rd, 2020].

[14] MongoDB, Inc., "MongoDB," https://www.mongodb.com/, [Online; accessed April 23rd, 2020].

[15] MonetDB B.V., "MonetDB," https://www.monetdb.org/Home, [Online; accessed April 23rd, 2020].

[16] InfluxData Inc, "InfluxDB," https://www.influxdata.com/, [Online; accessed April 23rd, 2020].

[17] C. Kristina and D. Michael, "MongoDB: The Definitive Guide," 2010.

[18] Joson Morn, "Issues with InfluxDB," https://groups.google.com/forum/msg/influxdb/jGVE3uDStNg/9KYxjY46AQAJ, [Online; accessed 23-April-2020].

[19] A. Pavlo and M. Aslett, "What's Really New with NewSQL?" ACM Sigmod Record, vol. 45, no. 2, 2016, pp. 45–55.

 ISBN: 978-1-61208-790-0

TABLE III. QUERIES WRITTEN FOR EACH DATABASE

Name	Query
MySQL 1	SELECT * FROM dataset WHERE ID = ‘644’
MySQL 2	SELECT DISTINCT(ID) FROM dataset
MySQL 3	SELECT COUNT(*) FROM dataset
MySQL 4	SELECT * FROM dataset WHERE Date >= ‘09/01/2016’ AND Date < ‘09/02/2016’
MySQL 5	SELECT ID,Date,SUM(Value) AS SUM FROM dataset WHERE ResultTypeID = ‘Voltage_L1_015min’ GROUP BY ID,Date
MySQL 6	SELECT ID,Date,avg(Value) AS average FROM dataset WHERE ResultTypeID = ‘Voltage_L1_015min’ GROUP BY ID,Date
MySQL 7	SELECT ID,ResultTypeID,avg(Value) AS average FROM dataset WHERE ResultTypeID = ‘Current_L1_015min’ or ResultTypeID = ‘Current_L2_015min’ or ResultTypeID = ‘Current_L3_015min’ GROUP BY ID,ResultTypeID Having avg(Value) > 0 AND avg(Value) < 5
MySQL 8	SELECT ID,Date,avg(Value) AS SUM FROM dataset WHERE ResultTypeID = ‘Voltage_L1_015min’ GROUP BY ID,Date UNION SELECT ID,Date,avg(Value) AS SUM FROM dataset WHERE ResultTypeID = ‘Voltage_L2_015min’ GROUP BY ID,Date UNION SELECT ID,Date,avg(Value) AS SUM FROM dataset WHERE ResultTypeID = ‘Voltage_L3_015min’ GROUP BY ID,Date
MySQL 9	SELECT ID, Date, Value FROM dataset WHERE Value = (SELECT MAX(Value) FROM dataset WHERE ResultTypeID = ‘Current_L1_015min’) UNION SELECT ID, Date, Value FROM dataset WHERE Value = (SELECT MAX(Value) FROM dataset WHERE ResultTypeID = ‘Current_L2_015min’) UNION SELECT ID, Date, Value FROM dataset WHERE Value = (SELECT MAX(Value) FROM dataset WHERE ResultTypeID = ‘Current_L3_015min’)
InfluxDB 1	SELECT * FROM dataset WHERE ID = ’644’
InfluxDB 2	SELECT DISTINCT(ID) FROM dataset
InfluxDB 3	SELECT COUNT(Value) FROM dataset
InfluxDB 4	SELECT * FROM dataset WHERE TimeStamp >= ‘2016-09-01T00:00:00Z’ AND TimeStamp <= ‘2016-09-02T23:59:59Z’
InfluxDB 5	SELECT SUM(Value) FROM dataset WHERE ResultTypeId = ‘Voltage_L1_015min’ GROUP BY time(1d)
InfluxDB 6	SELECT avg(Value) FROM dataset WHERE ResultTypeId = ‘Voltage_L1_015min’ GROUP BY time(1d)
InfluxDB 7	CREATE CONTINUOUS QUERY “meter_cq” ON “KE_SM_1” BEGIN SELECT avg(Value) AS “mean_meters” INTO “aggregate_meter” FROM dataset WHERE ResultTypeID = ‘Current_L1_015min’ AND ResultTypeID = ‘Current_L2_015min’ AND ResultTypeID = ‘Current_L1_015min’ GROUP BY time(1d); SELECT “mean_meters” FROM “aggregate_meter” WHERE “mean_meter” < 5
MongoDB 1	db.SM_RECS.find(‘ID’:644)
MongoDB 2	db.SM_RECS.DISTINCT(‘ID’)
MongoDB 3	db.SM_RECS.find().COUNT()
MongoDB 4	db.SM_RECS.find(’Date’ :$gt:‘09/01/2016’, ‘Date’:$lt:‘09/02/2016’)
MongoDB 5	db.SM_RECS.aggregate([$match: “ResultTypeID”: “ Voltage_L1_015min” , $group: _id: “ID”, date:“Date”,SUM: $SUM: “Value”])
MongoDB 6	db.SM_RECS.aggregate([$match: “ResultTypeID”: “ Voltage_L1_015min” , $group: _id: “ID”, date:“Date”,average: $avg: “Value”])
MongoDB 7	db.SM_RECS.aggregate([$or: [$match: “ResultTypeID”: “Current_L1_015min”,“ResultTypeID”: “Current_L2_015min”,“ResultTypeID”: “Current_L2_015min”], $AND: [$avg:“Value” > 0,$avg:“Value” < 5], $group: _id: “$ID”, typeID:“ResultTypeID”,average: $avg: “$Value”], allowDiskUse: true)
MonetDB 1	SELECT * FROM dataset WHERE ID = ‘644’
MonetDB 2	SELECT DISTINCT(ID) FROM dataset
MonetDB 3	SELECT COUNT(*) FROM dataset
MonetDB 4	SELECT * FROM dataset WHERE Date >= ‘09/01/2016’ AND Date < ‘09/02/2016’
MonetDB 5	SELECT ID, Date,SUM(Value) AS SUM FROM dataset WHERE ResultTypeID = ‘Voltage_L1_015min’ GROUP BY ID,Date
MonetDB 6	SELECT ID,Date,avg(Value) AS average FROM dataset WHERE ResultTypeID = ‘Voltage_L1_015min’ GROUP BY ID,Date
MonetDB 7	SELECT ID,ResultTypeID,avg(Value) AS average FROM dataset WHERE ResultTypeID = ‘Current_L1_015min’ or ResultTypeID = ‘Current_L2_015min’ or ResultTypeID = ‘Current_L3_015min’ GROUP BY ID,ResultTypeID Having avg(Value) > 0 AND avg(Value) < 5
MonetDB 8	SELECT ID,Date,avg(Value) AS SUM FROM dataset WHERE ResultTypeID = ‘Voltage_L1_015min’ GROUP BY ID,Date UNION SELECT ID,Date,avg(Value) AS SUM FROM dataset WHERE ResultTypeID = ‘Voltage_L2_015min’ GROUP BY ID,Date UNION SELECT ID,Date,avg(Value) AS SUM FROM dataset WHERE ResultTypeID = ‘Voltage_L3_015min’ GROUP BY ID,Date
MonetDB 9	SELECT ID, Date, Value FROM dataset WHERE Value = (SELECT MAX(Value) FROM dataset WHERE ResultTypeID = ‘Current_L1_015min’) UNION SELECT ID, Date, Value FROM dataset WHERE Value = (SELECT MAX(Value) FROM dataset WHERE ResultTypeID = ‘Current_L2_015min’) UNION SELECT ID, Date, Value FROM dataset WHERE Value = (SELECT MAX(Value) FROM dataset WHERE ResultTypeID = ‘Current_L3_015min’)

 ISBN: 978-1-61208-790-0

Tackling Semantic Shift in Industrial Streaming Data Over Time

Lisa Ehrlinger*†, Christian Lettner*, Johannes Himmelbauer*
*Software Competence Center Hagenberg, Softwarepark 21, 4232 Hagenberg, Austria
†Johannes Kepler University Linz, Altenberger Straße 69, 4040 Linz, Austria
email: lisa.ehrlinger@jku.at, christian.lettner@scch.at, johannes.himmelbauer@scch.at

***Abstract*—Industrial production processes generate huge amounts of streaming data, usually collected by the deployed machines. To allow the analysis of this data (e.g., for process stability monitoring or predictive maintenance), it is necessary that the data streams are of high quality and comparable between machines. A common problem in such scenarios is *semantic shift*. For example, a sensor's weight unit might shift from tons to kilograms after a firmware update and still store the collected values to the same variable. In this paper, we discuss *semantic shift* theoretically and by means of an industrial case study from a production plant in Austria, where several hundred injection molding machines are employed. The data collected by these machines is used to monitor the stability of the production process with machine learning algorithms. In the following, we present and discuss the data preprocessing system we developed for the production plant to handle semantic shift for huge amounts of streaming data.**

Keywords–Semantic Shift; Streaming Data; Data Quality; Process Stability Monitoring; Data Preprocessing.

I. INTRODUCTION

Semantic shift originally describes the evolution of word meaning over time [1]. In this paper, we observe the semantic shift of industrial data streams, where the meaning of variables (also: attributes, features, column names) changes over time. Semantic shift has a negative effect on data analysis and thus, needs to be tackled strategically. We claim that semantic shift can be seen as a Data Quality (DQ) problem, which however, has been little discussed in this context so far.

The awareness for this problem has been raised by different research projects with company partners from industry. In this paper, we specifically describe the use case from one production plant in Austria where injection molding machines are employed to produce plastic products. Such industrial production processes have natural fluctuations due to the physical conditions of the machines, as well as the variability of the used materials [2]. To guarantee the production of high-quality products, it is essential to continuously monitor process signals and to ensure it moves within the specified limits [2]. To support stability monitoring of the production process with statistical measures, we developed L* (pronounce: L-star) a data preprocessing infrastructure that overcomes semantic shift in the process variables.

Therefore, we make the following twofold contribution: (1) a discussion of *semantic shift* as a data quality problem and outlook for future research direction, and (2) a case study from an industrial plant where we deployed the data preprocessing system L*, which tackles the problem of semantic shift in industrial data streams. The system has been deployed at our company partner and is currently under ongoing evaluation.

In Section II, we describe semantic shift as it appears in literature and related work. The case study at the production plant, which highlights the practical relevance of the concept is discussed in Section III. In Section IV, we present L*: a process data preprocessing system where we specifically describe the components that tackle semantic shift. We conclude with an outlook on future work in Section V.

II. SEMANTIC SHIFT – A DATA QUALITY PROBLEM

Historically, *semantic shift* (also: *semantic change*, *semantic drift*) is a term stemming from linguistics and describes the evolution of word meaning over time [1]. According to Bloomfield [1], it can have different triggers and different development. Although used interchangeably in linguistics, we explicitly want to highlight the focus of our research on *shift* (i.e., changes that can be attributed to a specific point in time [3]) in contrast to *drift* (i.e., continuous transformation [3]). The reason is that semantic changes in process data can usually be traced back to specific triggers, e.g., firmware update of a machine, or a change in the production process.

A similar and intensively studied term from Machine Learning (ML) research is *concept drift*, which refers to a drift in the target variable predicted by a ML model [4][5]. Such drifts are usually caused by changes in the hidden context and can be handled with regular updates of the ML model to ensure that the properties of the variable remain stable over time [5]. Klenner and Hahn [6] discuss the problem of semantic shift under the term *concept versioning* for technical standards.

Although there is a lot of research into DQ dimensions (cf. [7]–[10]), there is little discussion on the specific topic "semantic shift". In terms of DQ assessment in ontologies, Guarino and Welty [11] introduce the properties "identity" and "rigity", which are related to the stability of a variable. A similar DQ dimension is *timeliness*, which can be described as "how current data is for the task at hand" [12]. Semantic shift generalizes this dimension since the validity of data depends on the context within it appears (e.g., on the respective machine and the point in time). Thus, we define *semantic shift* in the context of DQ as the circumstance when "the meaning of data evolves depending on contextual factors". Consequently, when these factors are modeled accordingly (e.g., described with rules), it is possible to handle semantic shift even in very complex environments as outlined in the following case study.

III. SEMANTIC SHIFT IN INDUSTRIAL DATA STREAMS

In this section, we motivate the problem of semantic shift with the description of an industrial case study. Due to confidentiality, we are not allowed to publish details of the production process.

Our Austrian manufacturing company partner works in the field of plastics industry with injection molding machines. These machines are tools being able to produce plastic products and multi material-parts by the injection molding process

 ISBN: 978-1-61208-790-0

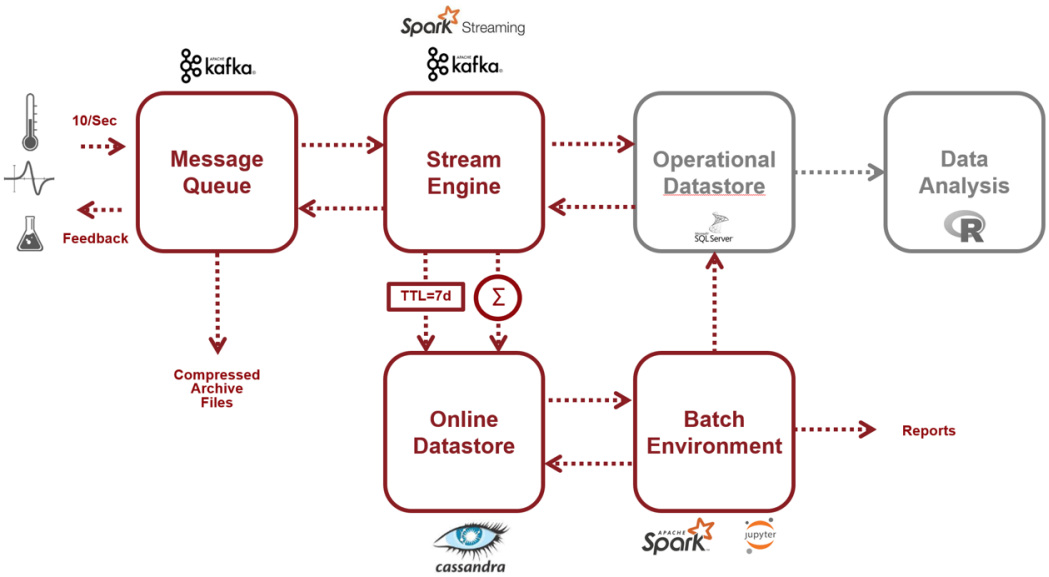


Figure 1. Architecture of the L* Data Infrastructure to Handle Semantic Shift

with a clear focus on mass-production. Injection molding represents a very complex physical-chemical process and there exists a wide variety of situations that can lead to a bad, or at least unstable, condition of a machine, as well as its production process, often ending up in (increased) production rejects. Considering that there are usually more than one hundred machines simultaneously in operation explains the company's aim for a monitoring system that can automatically send alerts where process instabilities that are potentially relevant for production quality show up. The benefit of such a system is manifold. At first, detecting unstable process situations is a prerequisite to actuate countermeasures in order to decrease scrap rates or to avoid machine damage. Moreover, the company operates in the field of massive production of very small pieces and thus complete quality inspection is unfeasible. Time-related knowledge about the process stability for each machine enables us to focus the quality inspection on produced pieces from critical production time periods.

In the collaboration with our company partner, we have worked towards the design of a data-driven solution being able to automatically recognize such critical situations. Thus, it is necessary to note that the machines cyclically supply status values to a machine data acquisition system. These machine statuses are recorded shot by shot and currently stored for several months. By analyzing this data, machine states should be determined by our data-driven solution. The process conditions depend on the following factors, which partly influence each other: machine condition, tool condition, material condition, environmental influences, and processor operating point setting. Our data-driven solution can find diverse known error patterns (ranging from occasionally occurring, isolated critical shots to slowly (in terms of weeks) deteriorating machine conditions, e.g., due to wearing of machine parts) in all machine data. For this purpose, firstly, we analyze in which data sources it is possible to find relevant information from which we can benefit. Based on that information, our solution tries to learn recurrent error patterns. For these tasks, we use different methods including stream data processing, classical machine learning algorithms, outlier detection, robust learning algorithms, and causal discovery.

For use in real production, these developed applications should be easy to integrate in the existing operation system and they should be applicable to as many machines as possible without specific adaptions. Applying certain stability checks to only a few out of many machines is unsatisfactory. Here, we want to point out that all these applications are based on making use of the data that a machine provides and each algorithm expects to be fed with data in a predefined standardized format. Fortunately, the injection molding machines of our customer are almost exclusively from the same vendor; shipped with a standardized data Application Programming Interface (API), which logs data about the injection molding process (in the following indicated with `MD`, short for measurement data) in a system called "MES system". However, there exist different machine types and machine versions. Moreover, machines with identical machine type and version can still provide differences with respect to provided data as different firmware might include also changes in the data schema. Due to the fact that all the machines come from the same vendor, all in all, we found a high level of data consistency; in the sense that variable names remain the same and major changes in newer versions mainly consist in extensions with additional variables. However, there exist cases when certain variables undergo a semantic shift. For example, a variable that represents the measurements of some pressure sensor for one machine might be stored in bar while for another machine or even for the same in a later version the same variable is recorded in millibar. Ignoring such semantic shifts would result in situations when algorithms produce wrong results.

IV. L* System Architecture

The entire data preprocessing system has been implemented on four nodes with Linux Ubuntu 16.04 installed, respectively. Figure 1 illustrates the system architecture of

 ISBN: 978-1-61208-790-0

the implementation, where the red components (which are are part of L*) are described in the following subsections. Each component refers to exactly one node in the system infrastructure. The system is designed for linear scalability and therefore employs tools from the Big Data ecosystems. The gray components illustrate the original data analysis infrastructure at the production plant used for process stability monitoring.

A. Data Loading

Initially, process data collected at the injection molding machines is loaded every 10 seconds with a message queue to the stream engine. The message queue has been implemented with Apache Kafka [13] and aims at a robust transmission of huge amount of messages. One advantage of using an asynchronous solution here is that the message queue represents a buffer, which is why messages are not lost even if L* is offline temporarily. We installed Apache Kafka through the Confluent platform [14], which is an event streaming platform that allows to manage and organize data streams (from different sources) for industry applications with high-performance requirements.

B. Online Datastore

The online datastore has been implemented with Apache Cassandra [15], a column-based NoSQL DB that is optimized to manage large amounts of measurement data. Since we deployed the system for our company partner, we selected Cassandra also due to its popularity [16] in comparison to other NoSQL DBs that have similar features. Figure 2 shows the creation statements of the two tables used for storing the process data.

```
create table MDavro (
  jahr int,
  seriennummer int,
  interval int,
  zeitpunkt timestamp,
  value blob,
  primary key((jahr, seriennummer, interval),
      zeitpunkt)
);

create table MD (
  jahr int,
  seriennummer int,
  metric text,
  zeitpunkt timestamp,
  value text,
  primary key((jahr, seriennummer, metric),
      zeitpunkt)
);
```

Figure 2. Cassandra Tables to Store Process Data

C. Data Preprocessing

We used Spark [17] to implement the data preprocessing system, which specifically tackles the problem of semantic shift for our use case. Three different Spark jobs have been implemented: (1) `LoadMD- Avro`, (2) `PreProMDStream`, and (3) `PreProMDBatch`, where the first two are implemented as Spark streaming jobs, and the last one as batch job. Figure 3 displays the three data streams between Cassandra and the streaming platform Confluent.

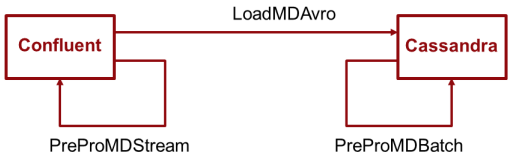


Figure 3. Spark Data Streams

1) Stream Engine: `PreProMDStream` receives data, which is encoded with the Apache Avro [18] data serialization from the machines. The data is decoded and preprocessed according the the defined rules (cf. Table I) to handle semantic shifts. Eventually, the task returns the encoded data back to Confluent.

2) Batch Environment: `PreProMDBatch` basically has the same functionality as `PreProMDStream`, only that it is conducted as Spark batch job. Thus, it requires a defined time interval (start and end point) to load and process the data.

L* supports linear transformations and the application of a time offset (lag). The current version does not allow to represent calculated values, which needs to be done with an external program.

Table I shows an excerpt of preprocessing rules defined to handle semantic shift. Depending on the machine type (`machinetype`), the table maps a machine internal parameter name to a consolidated, meaningful parameter name. In addition, a simple linear transformation (`scale` and `offset`), as well as a time delay (`lag`) may be applied. In the example provided in Table I, a semantic shift has happened on `process_value_3` for machine type `T3`. Starting with machine type `T3`, the production mode is divided into two phases. Further, in machine type `M3`, the temperature values are measured in degrees Fahrenheit. To consolidate these values to `M1` and `M2` values, which are measured in degree Fahrenheit, the values must be multiplied by 1.8 and shifted by 32. The process parameter `Process Temperature1 Previous` makes use of a time delay functionality to provide the previously measured temperature.

D. Performance Metrics

Since L* should be capable for deployment in productive environments, we calculated a few performance metrics to verify its suitability to handle Big Data.

In a test, 28.8 million records have been processed from the MES system, which contained a total of 1,216 million measurement values. This yielded an average of 42.2 measurement values per record. Table II summarizes the processed records or measurement values (short “values”) per Spark job. In total, `LoadMDAvro` generated disk storage of 5.01 GB for the Cassandra table `MDavro` and `PreProMDBatch` disk space of 6.49 GB for the table `MD` in the 2.5 weeks time period.

V. Conclusion and Outlook

In this paper, we presented the data preprocessing system L*, which tackles semantic shift in data streams used for process stability monitoring. The rule-based solution is a first attempt to systematically overcome shift in process variables and aligns with the predominant idea how to solve DQ issues in practice (cf. [8]). In the future, we would like to extent

 ISBN: 978-1-61208-790-0

TABLE I. DATA PREPROCESSING DEFINITION

MD_paramname	process_paramname	machinetype	scale	offset	lag	datatype
process_value_1	Mode Stopped	T1, T2, T3	1	0	0	bool
process_value_2	Mode Starting	T1, T2, T3	1	0	0	bool
process_value_3	Mode Production	T1, T2	1	0	0	bool
process_value_4	Product Counter	T1, T2, T3	1	0	0	long
process_value_5	Process Temperature1	T1, T2	1	0	0	float
process_value_6	Process Preasure	T1, T2	1	0	0	float
process_value_3	Mode Production Phase 1	T3	1	0	0	bool
process_value_7	Mode Production Phase 2	T3	1	0	0	bool
process_value_5	Process Temperature1	T3	1.8	32	0	float
process_value_6	Process Temperature2	T3	1.8	32	0	float
process_value_5	Process Temperature1 Previous	T3	1.8	32	1	float
process_value_8	Process Preasure	T3	1	0	0	float

TABLE II. PERFORMANCE METRICS

Spark Data Stream	Unit	Throughput (unit/sec)	Storage (byte/unit)
`LoadMDAvro`	Records	358	182
`PreProMDBatch`	Values	174,343	5.6
`PreProMDStream`	Values	4,816	-

this rule-based system with a semantic solution that takes into account the context (e.g., of the respective machine) since it allows to reach a higher degree of automation.

In our ongoing work, we are going to generalize the problem of semantic shift by investigating DQ assessment for streaming data more broadly. Although there exist many context-independent DQ metrics for batch data sets (cf. [7]), so far, there is little research specifically on data streams. Thus, we would like to extract domain-independent properties that can be applied to measure the DQ of any data stream.

ACKNOWLEDGMENT

The research reported in this paper has been supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Digital and Economic Affairs, and the Province of Upper Austria in the frame of the COMET center SCCH.

REFERENCES

[1] L. Bloomfield, *Language*. Allen & Unwin, 1933.

[2] R. D. Snee, "Crucial Considerations in Monitoring Process Performance and Product Quality," *Pharmaceutical Technology*, vol. 34, no. 10, 2010, pp. 38–40.

[3] Oxford University Press, "Oxford Dictionaries," https://www.lexico.com/?search_filter=dictionary [retrieved: April, 2020].

[4] A. Tsymbal, "The Problem of Concept Drift: Definitions and Related Work," *Computer Science Department, Trinity College Dublin*, vol. 106, no. 2, 2004, p. 58.

[5] G. Widmer and M. Kubat, "Learning in the Presence of Concept Drift and Hidden Contexts," *Machine Learning*, vol. 23, no. 1, 1996, pp. 69–101.

[6] M. Klenner and U. Hahn, "Concept Versioning: A Methodology for Tracking Evolutionary Concept Drift in Dynamic Concept Systems," in *ECAI*, vol. 94. PITMAN, 1994, pp. 473–477.

[7] B. Heinrich, D. Hristova, M. Klier, A. Schiller, and M. Szubartowicz, "Requirements for Data Quality Metrics," *Journal of Data and Information Quality*, vol. 9, no. 2, January 2018, pp. 12:1–12:32.

[8] L. Ehrlinger, E. Rusz, and W. Wöß, "A Survey of Data Quality Measurement and Monitoring Tools," 2019, https://arxiv.org/abs/1907.08138 [retrieved: April, 2020].

[9] R. Y. Wang and D. M. Strong, "Beyond accuracy: What data quality means to data consumers," *Journal of Management Information Systems*, vol. 12, no. 4, 03 1996, pp. 5–33.

[10] M. Scannapieco and T. Catarci, "Data Quality Under a Computer Science Perspective," *Archivi & Computer*, vol. 2, 2002, pp. 1–15.

[11] N. Guarino and C. Welty, "Evaluating Ontological Decisions with OntoClean," *Communications of the ACM*, vol. 45, no. 2, 2002, pp. 61–65.

[12] B. Heinrich and M. Klier, "A Novel Data Quality Metric for Timeliness Considering Supplemental Data," in *Proceedings of the 17th European Conference on Information Systems*. Verona, Italy: Università di Verona, Facoltà di Economia, Departimento de Economia Aziendale, 2009, pp. 2701–2713.

[13] Apache Software Foundation, "Apache Kafka – A Distributed Streaming Platform," Online, 2020, https://kafka.apache.org [retrieved: April, 2020].

[14] Confluent Inc., "Confluent," Online, 2020, https://docs.confluent.io [retrieved: April, 2020].

[15] Apache Software Foundation, "Apache Cassandra," Online, 2020, http://cassandra.apache.org [retrieved: April, 2020].

[16] solid IT gmbh, "DB-Engines Ranking of Wide Column Stores," Online, 2020, https://db-engines.com/en/ranking/wide+column+store [retrieved: April, 2020].

[17] Apache Software Foundation, "Apache Spark," Online, 2020, https://spark.apache.org [retrieved: April, 2020].

[18] Apache Software Foundation, "Apache Avro," Online, 2020, https://avro.apache.org [retrieved: April, 2020].

 ISBN: 978-1-61208-790-0

Principle Structure and Architecture of a Code Generator

Andreas Schmidt*†
* Faculty of Computer Science and Business Information Systems,
Karlsruhe University of Applied Sciences
Karlsruhe, Germany
Email: andreas.schmidt@hs-karlsruhe.de
† Institute for Automation and Applied Informatics
Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: andreas.schmidt@kit.edu

***Abstract*—Code generators often have something mystical about them. Especially undergraduate students, who can still remember their first steps in programming, become in awe when they hear the term "Software Generator". The paper is an attempt to take this awe away from the students and to show them, by means of a very simple example implementation with well-known tools and technologies, that software generators are not witches' work, but a powerful, but easily understandable tool to support the software development process.**

Keywords–Code generation; template system; (meta) model; model transformation.

I. INTRODUCTION

In this paper, the general structure of a code generator is presented. It is intended as additional material to the tutorial with the title "Code generation for Database Developers" which is also given by the author at the DBKDA-2020 conference in Lisbon [1]. The principle structure and architecture of a general purpose code generator will be explained with the help of a simple example implementation, using well known tools and techniques. The procedure is from the backend of the generator, over the kernel to the frontend. The advantage of this approach is that one can see the final result (the generated code) right at the beginning and then deal with the details to achieve this result. The Template Engine of the generator, the internal metamodel, the import module, the external metamodel and the transformation of XMI (**X**ML **M**etadata **I**nterchange) - the standard exchange format for models - into the previously developed metamodel are then presented.

A. Principle Function of a Generator

The principle mode of operation of a generator is shown in Figure 1. The generator obtains as input an abstract model description and a set of transformation rules, which describe the transformation of the abstract model into the source code. It is crucial that the model is formal and the model description is available in a form that abstracts from implementation specific details. Through one or more model transformations, the implementation details are added to the target platform. This achieves a separation between the business logic and the technical aspects of the target platform.

B. Advantages of Generative Software Development

Herrington [2] names four main advantages of generative software development, which are to be presented in the following briefly.

1) Quality: The quality of the software is determined by the transformation rules. Over time, these rules gain more and more quality, so that the quality of the generated source code increases. The automatic transformations avoid careless mistakes. If individual transformation rules are faulty, these errors occur at all places that use the faulty transformation rules and are therefore easy to find and correct. Furthermore, when developing the transformation rules, more thought is given to the architecture of the application in advance than when starting directly with the coding. The previously considered architecture is then consistently implemented in the complete source code by the transformation rules.

2) Consistency: Source code generated by transformation rules is very consistent regarding naming, calling conventions and parameter passing, so that it is quite easy to understand and use. This offers a starting point for further possible automations. Cross-sectional functionalities such as logging or error handling can be defined centrally and thus be adapted to changing requirements at any time (analogous to aspect-oriented programming).

3) Productivity: Productivity in application development increases. Even if only so-called infrastructure code is generated, which is often considered to be the boring part of programming, more time remains to take care of the actual (exciting) application logic. Furthermore, it is possible to react faster to design changes or change requirements, because only the corresponding transformation rules have to be adapted and the application has to be regenerated.

4) Abstraction: The model represents an abstract description of the application to be realized. The strict separation of domain-oriented logic (model) and technical aspects (transformation rules) reduces complexity. This, in turn, allows for a

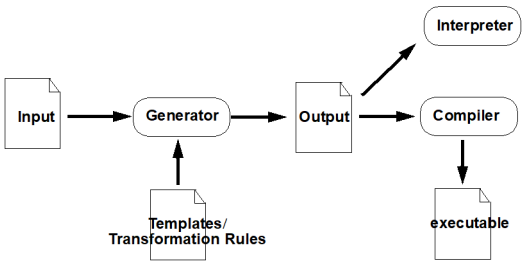


Figure 1. General Architecture

 ISBN: 978-1-61208-790-0

better integration of domain experts within the development project, as they can be involved in the development of the model. Another advantage is the easier transition to a new technology, since only the transformation rules have to be adapted, since the domain-oriented logic of the model remains valid. On the other hand, transformation rules once developed can be reused in other applications.

The remaining paper is structured as follows. In Section 2, we will discuss which artifacts can typically be generated. In Section 3, the development of the generator is presented in detail, divided into backend, kernel and frontend functionality. In Section 4, the automation of the single steps using the Unix tool *make* is discussed. Section 5 concludes with a discussion of possible extensions for the generator prototype.

II. What can be generated?

The goal is the partial or complete generation of the source code for an application to be realized. The degree of automation usually ranges from 20% to 80% of a application. Higher levels of automation are possible but often not useful, because this would make the generator much more complex than implementing the missing 20% of the software by hand [3]. For web-based applications, a degree of automation of about 60-70% can often be achieved. Typical parts of an application that can be generated include the following areas:

- Database schemas
- Access layers for databases
- User interfaces
- Parts of the application logic
- Documentation
- Configurations (e.g., in combination with frameworks like Struts, Spring, Hibernate, etc.)
- Tests (unit tests, constraint tests, generation of mock objects, load tests, etc.)
- wrapper
- Import/Export Modules
- etc.

III. Development of the Generator

In the following, a multipurpose generator is to be built up by the simplest means. This is done exemplarily with the programming language PHP [4]. The reasons for using PHP are the following: PHP is a macro language and can therefore also be used as a template system, which can be used for the definition of the mapping rules. In addition, there are also special template languages for PHP, which can be used for this purpose. Due to its primary field of application as a language for creating dynamic websites, PHP is characterized by its powerful string handling. This is also useful for generating source code. Furthermore, there are many free libraries available for PHP (PHP Extension & Application Repository - PEAR). Other languages suitable for this task are Perl, Python and Ruby.

Besides PHP the following tools/technologies are used:

- An XSLT [5] or XQuery [6] processor to transform XMI into a simpler meta-format
- The Unix tool make [7] for automation of the entire workflow

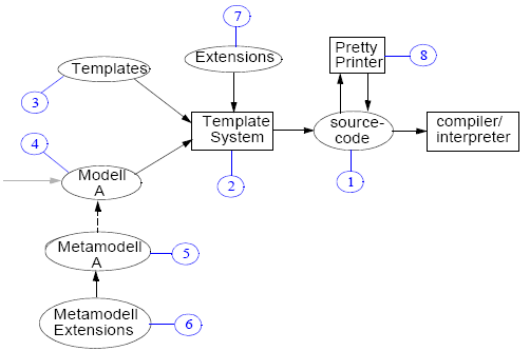


Figure 2. Generator Backend

- The Smarty template engine [8].
- An UML modeling tool for graphical modeling (i.e., [9]).

A. Scope of Functions and Expansion Options

The functionality of the generator to be developed is limited to the generation of artifacts based on the information of a simple class model. This does not represent a limitation for the basic architecture. In Section V it is shown how the generator can be extended to process further model elements (e.g. state transition diagrams).

B. Generator Backend

The basic design of the generator jaw is shown in Figure 2. The backend is responsible for the actual generation of the source code (1). For this purpose, a template system (2) is used, whose task it is to create clear mapping rules from the model to the target language by separating the dynamic and static parts. For this purpose, the template system uses as input on the one hand the so-called templates (3), in which the transformation rules for the generation of the source code in the form of static text and simple control flow elements, such as loops and conditional statements as well as placeholders for the information originating from the model are stored, and on the other hand the model (4) on which the application to be generated is based, which contains the dynamic parts of the source code to be generated.

In the concrete case, the model is available in the form of an arbitrarily complex object network, which describes the artifacts to be modeled such as classes, attributes with types, as well as relationships. The model is also based on the so-called meta model (5), which defines the modeling possibilities in the form of classes and the associated methods. This metamodel is realized by means of PHP classes. Figure 3 shows a code snippet for defining a model using the Metamodel API. In the code section, the two classes ”person” and ”film” are defined with their attributes and furthermore the relationship ”film_director”, which models a 1:n relationship between film and person.

An example of a Smarty template is shown in Figure 4. The example shows a template for generating the database schema. Language elements of the template language are indicated by `[@ ... @]` brackets. Available language elements include loops, conditional statements, variable assignments, calling

```
$model = MetaModel::createModel('Film DB');

$p = $model->addClass('Person');
$f = $model->addClass('Film');

$p->addAttribute('id','Integer',10, true);
$p->addAttribute('name','String',30);
$p->addAttribute('prename','String',30);
$p->addAttribute('birthday','date');
$p->addRelation('film', 'Film',0, -1, "film_regisseur");


$f->addAttribute('id', 'Integer',10,  true);
$f->addAttribute('title', 'String',50);
$f->addAttribute('year', 'date');
$f->addRelation('regisseur', 'Person',0 ,1, "film_regisseur");
```

Figure 3. Programmatic Model Definition

```
drop database if exists mdsd;

create database mdsd;

use mdsd;
[@ $model->setTargetSystem('mysql') @]

[@ foreach from=$model->classes item=class @]

create table [@ $class->name @] (
  [@ foreach from=$class->attributes item=att @]
    [@ $att->name @] [@ $att->type @]
    [@ if ($att->length > 0) @] ([@ $att->length @]) [@ /if @],
  [@ /foreach @]
  primary key([@ $class->primary_key->name @])
) Type=InnoDB;
[@ /foreach @]

[@ include file="ddl_1_n_relations.tpl model=$model @]
```

Figure 4. Template for generating the Database Schema

other templates and calling properties and methods of the meta model.

In order to make the creation of the templates as simple and clear as possible, it is often helpful to extend the meta model (6) or the generator (7) for certain specific language constructs of the target language instead of formulating these language constructs within the templates (3).

C. Generator Kernel

The actual heart of the generator is represented by the generator kernel. Figure 5 shows the transformation and validation components of the generator. The methods of the metamodel already monitor a number of constraints in the model, for example that the classes have different names and that the attributes must be of certain predefined types. However, there are also constraints that cannot be enforced in this way, for example the constraint that each class must have a primary key or that certain attributes/relationships must exist for each class. For this purpose, the generator provides an interface for the formulation of validation rules (11). These are implemented in the form of PHP methods (12). An example of such a method is shown in Figure 6. This method monitors that each class must have a primary key. The methods also work like the templates on the properties and methods of the Metamodel API.

Furthermore, model transformations (13, 14) can be formulated. In the simple case these are transformations within the same metamodel (13). For example, additional administrative information (created_at, created_from, etc.) can be added to a model for each class (see Figure 6. For the formulation of

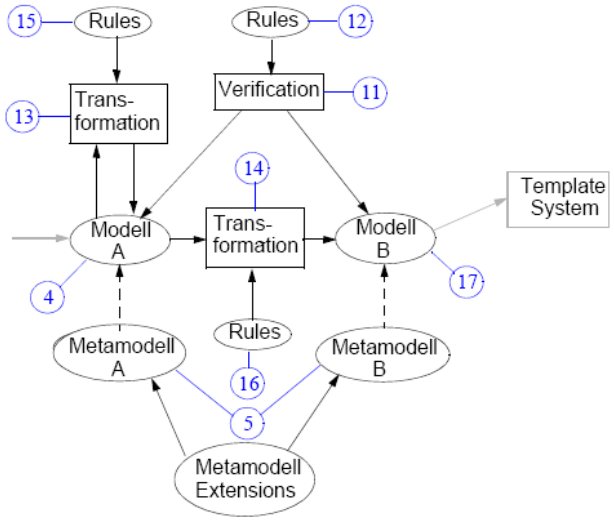


Figure 5. Generator kernel

```
include_once 'MetaInterfaces.php';

class add_administrative_fields implements MetaTransformer  {

  static function transform(MetaModel &$model) {

    foreach ($model->classes as $class) {
      $class->addAttribute('created_at', 'date');
      $class->addAttribute('modified_at', 'date');
    }
  }
}
```

Figure 6. Example Model Transformation

transformation rules, the generator also provides an interface, which allows the formulation of transformations in the form of methods (15). Furthermore, transformations (14) to another metamodel (16) are also possible (e.g., to a metamodel with the concepts table, attribute, foreign key, constraints, etc.).

D. Generator Frontend

1) Model Import: Up to now, models can only be built using the methods available in the metamodel, i.e., programmatically through a series of API calls. However, this is not desirable and so an XML format (Figure 7, point 21) is defined in an extension of the generator, which allows the formulation of the model as an XML file (22). In this case, the meta model is represented by the DTD (21) and thus defines what can be formulated in the model file. In an import process (24) the DOM tree of the XML file is then created and a transformation (24) to the internal model (4) is carried out by the methods available in PHP for processing XML, i.e., the corresponding methods of the internal meta model are called during navigation through the DOM tree and thus the internal model representation (4) is built up. Optionally, the XML file (22) can be modified by means of an XSLT transformation (25) before import. Meaningful transformations on this level are, for example, the addition of further attributes or primary keys, if these have not already been specified in the UML model.

 ISBN: 978-1-61208-790-0

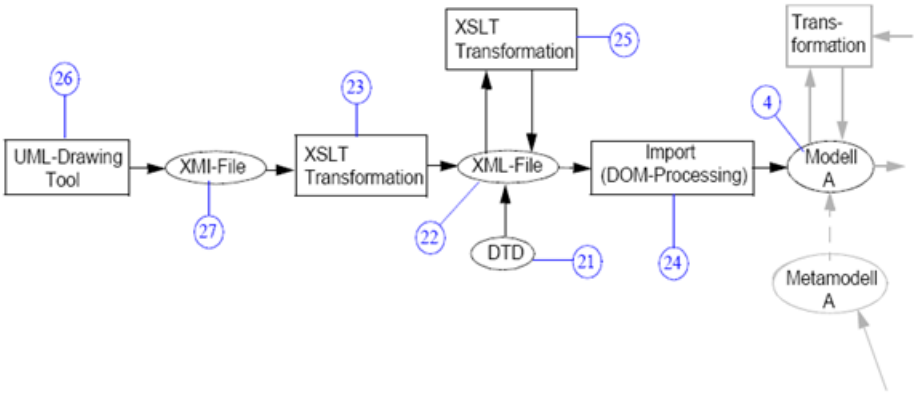


Figure 7. Generator Frontend

2) Connecting the Frontend: The connection of a UML modeling tool (26) is realized by the XMI export interface provided by most tools. XMI is an XML-based, standardized exchange format for UML models. To connect to the generator, all that is required is the development of an XSLT stylesheet (27), which extracts the relevant information from the XMI file and transforms it into the previously developed XML format (22). It is also possible to do without the own XML format and import the XMI file directly from the generator. The disadvantage of this variant, however, is that XMI is an extremely ”chatty” format and the import and transformation into the internal meta model is much more complicated than via the detour of the intermediate format.

IV. Automation

From the creation of the UML model to the export as XMI file, the XSLT transformation into the generator's own XML format, the model validation/transformation, the actual code generation based on the created templates, and any subsequent source code formatting (Figure 2, point 8), a complete generator run represents a complex workflow consisting of many individual steps and dependencies. To automate this, the development tool ”make” is used here. It allows the formulation of sequences of work steps as well as dependencies, which then cause a conditional execution of parts of the workflow.

V. Extension of the Generator

The generator introduced so far supports the generation of artifacts, which can be derived from a simple class model. In the context of the lectures carried out at the University of Applied Sciences Karlsruhe - Technology and Economics as a compulsory elective subject in the field of Business Informatics as well as further tutorials [10], [11] it was shown that a very high learning effect can be achieved by letting the participants extend the generator by additional diagram types. In contrast to the initial introduction of the generator, a forward-looking approach is suitable for the extension, i.e., starting from an XMI file generated by a modeling tool, the own XML format is extended and the corresponding XSLT transformation is adapted. Subsequently, the internal meta model must also be extended by the corresponding concepts and the import filter must be adapted accordingly. The last step is to create additional templates or to extend the existing templates. As extension for example the addition of state transition diagrams or the addition of the inheritance concept for class diagrams is suitable. A further instructive extension is the mapping of the present meta model to another meta model, which represents the concepts of relational databases and the subsequent adaptation of the templates.

VI. Conclusion

The presented framework shows in a simple way how a software generator works. Due to its easy extensibility, it can be adapted to own needs very easily. However, it is not intended to compete with existing tools but is mainly used in teaching. Nevertheless, it can be used to create own generators for applications where it is not worthwhile to learn a commercial or freely available tool.

References

[1] A. Schmidt, “Code Generation for Database Developers. Twelfth International Conference on Advances in Databases, Knowledge, and Data Applications - DBKDA ,” Lisbon, Portugal, 2020, URL: https://www.iaria.org/conferences2020/ProgramDBKDA20.html, last accessed: September 2020.

[2] J. Herrington, Code Generation in Action. USA: Manning Publications Co., 2003.

[3] T. Stahl, M. Voelter, and K. Czarnecki, Model-Driven Software Development: Technology, Engineering, Management. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2006.

[4] K. Tatroe and P. Macintyre, Programming PHP, O'Reilly. Sebastopol: O'Reilly, 2006.

[5] T. Doug, XSLT. Sebastopol: O'Reilly, 2008.

[6] P. Walmsley, XQuery: Search Across a Variety of XML Data. Sebastopol: O'Reilly, 2007.

[7] R. Mecklenburg, Managing Projects with GNU Make. Sebastopol: O'Reilly, 2004.

[8] L. Gheorghe, H. Hayder, and J. P. Maia, Smarty PHP Template Programming and Applications. Packt Publishing, 2006.

[9] “argo UML,” URL: https://github.com/argouml-tigris-org/argouml, last accessed: September 2020.

[10] A. Schmidt, “Supporting the development of web-based applications with lightweight software generators. Third International Conference on Internet technologies and Applications - ITA09,” Wrexham, Wales, 2009, Tutorial session.

[11] A. Schmidt, “The power of regular expressions in the software development process. International Conferences on Informatics 2010 Software Engineering and Applications SEA 2010,” Marina del Rey, USA, 2007, Tutorial session.

 ISBN: 978-1-61208-790-0