Estimation, Impact and Visualization of Schema Evolution in Graph Databases

Dominique Hausler
dominique.hausler@ur.de
University of Regensburg
Regensburg, Bavaria, Germany

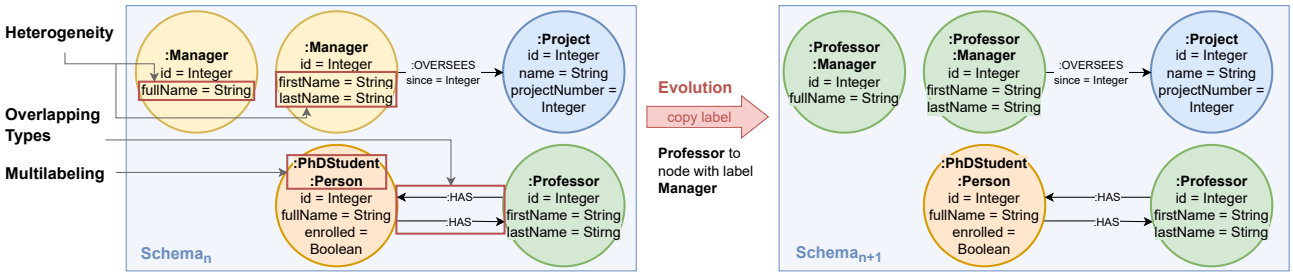

Figure 1: Impact of Schema Evolution in Graph Databases

ABSTRACT

Graph databases offer a flexible storage of interconnected data. Due to NoSQL databases being schema-less, heterogeneous data can occur when performing data changes. Evolution is conducted through so-called evolution operations like add, rename, delete, merge, copy, move or split. As a user cannot foresee the results of the evolution operation, neither the amount of data changes nor the possible schema violations or a relaxed schema, a system to show the impact of evolution is essential. To ensure schema conformity, we present an approach to close the gap of a schema management tool for graph databases in order to estimate and illustrate the impact of evolution on the schema level. To illustrate, explore, evolve and change the schema, all required information is handled through a schema management layer in the Nautilus program. Besides extracting the schema, so-called structure profiles are designed for an initial data exploration. The preview of the schema and structure profiles shows the impact of evolution on the data by comparing versions. Moreover, the system uses an intuitive syntax to also enable the usage of graph databases for non-experts.

CCS CONCEPTS

• **Information systems**; • **Data management systems**; • **Query languages**; • **Query languages for non-relational engines**;

KEYWORDS

Evolution language, Schema Evolution, Graph Database, Neo4j, Schema Management, Schema Extraction, Profiles

MODELS Companion '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3688196

ACM Reference Format:
Dominique Hausler. 2024. Estimation, Impact and Visualization of Schema Evolution in Graph Databases. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3652620.3688196

1 PROBLEM

To store highly interconnected data, graph databases are preferred. When operating a database over a long period, data undergoes change, manipulation or updates. This is accomplished through single-type (add, rename, delete), multi-type (copy, move, split, merge) and graph-specific operations (transform defined in [26]), describing the change of entity types (nodes or relationships) into one another. Considering context when looking at evolution is therefore, utmost important, especially for graph databases.

As graph databases can be schema-less, heterogeneity can occur through optional elements or structural error like in the *Manager* node in Figure 1. Detecting such phenomena is necessary to provide an overview of the current schema and statistics of the accessed data, which must be handled through a schema management layer. Especially for cooperative work on a database, this is a major point to ensure transparency. Also, heterogeneity can be either intended or causes structural errors. Consequently, our goal is to involve users by informing them about such occurrences through structure profiles and giving the option to make changes. Such changes can range from using evolution operations to fix structural errors to correcting a given input before executing an operation. A preview option aims to prevent unintended outcomes by illustrating version n+1 (= after the evolution) of the data beforehand. The necessity of a preview – illustrating the schema and structure profiles – is the consequence of not being able to predict the impact of evolution operations on the schema, especially for large datasets. Figure 1 demonstrates the impact of evolution by copying a label, resulting

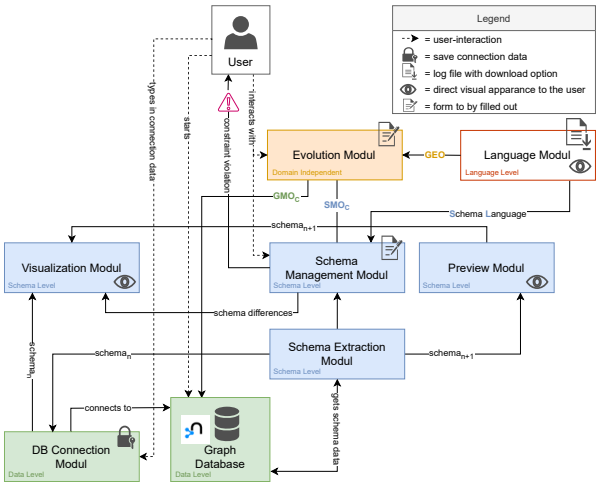

Figure 2: Architecture of Nautilus

in three different schema of nodes with the label `Professor`. Furthermore, the proposed program uses our evolution language with an intuitive syntax to conduct schema evolution.

2 EXPECTED CONTRIBUTIONS

The novelty of the proposed work lies in the following aspects which will be integrated in the Nautilus tool:

- A Schema Management Layer (SML), on base of which the schema and statistical information can be extracted and compared.
- The detection of graph schema constraints (or conditions).
- The extraction of structure profiles to easily detect multi-labeling, overlapping types (see Figure 1) and optional and mandatory elements to outline heterogeneous data.
- Compare the schema before and after the evolution.
- A graph-like schema visualization including the structure profiles to extend the use of graph databases for interdisciplinary research projects.

3 RELATED WORK

Meta-Model Approaches. [4] points out the importance of schema when working with NoSQL databases. [5] aims to manage heterogeneous schema of multiple data stores within one system. Amongst others HyDRa [23] works with cross-database integrity constraints to handle schema evolution in polystores. As schema evolution in single stores is already challenging, we focus on graph databases which were not taken into account in the literature above.

Schema of Graph Databases. There already was work done on the schema of graph databases [2] as well as for other NoSQL databases [33, 48]. In Neo4j possible while using the PG-Schema presented in [2]. The importance of schema and the relevance of (key) constraints are discussed in [3]. Neo4j offers several options to use constraints either for a schema-first or -second approach [42] also discussed in [44]. Currently, little work has been done on graph schema.

Schema Extraction. Schema extraction already was developed for other database systems e.g., in [32]. As a result of graph databases being schema-less, several schema extraction tools using a schema-second approach are available for property graphs [9, 18, 21]. Overlapping types in combination with multi-labeling are a challenge for Neo4j's command `apoc.meta.schema`. Neo4j itself offers the creation of constraints, either schema-first or -second. In both cases users have the option of relaxing the schema by adding optional elements [42]. We plan on building our extraction service on base of [21] with the novelty of accurately handling overlapping types.

Updates and Evolution. In [10–12] a schema language is described as well as the validation of graph databases against the schema is described. We built upon this work and furthermore, extended the evolution operations by `transform` to incorporate the complexity of graph databases. Approaches on eager vs. lazy migration are presented in [14, 28, 33]. [13] focuses on single-type evolution operations (add, delete, rename and retype) in the context of lazy migration and data evolution. VersionGraph on the other hand is an example for storing graph data at different versions [15].

Even though there now is a standardized **G**raph **Q**uery **L**anguage (GQL) [30], it does not contain an evolution language. Consequently, the use cases differ. A real-world use case for a relational database presented in [17], shows how the historical schema can be extracted and visualized to reconstruct the evolutionary process. This thesis focuses on the evolutionary process while aiming to ensure an intuitive syntax for a platform independent evolution language to perform, handle and visualize schema evolution in graph databases.

Data Exploration. The impact and number of entities affected by evolution, possibly violating the schema, cannot be estimated by a user. Consequently, informing the user is mandatory to prevent unintended outcomes. There are a number of tools to explore a variety of interconnected data as a graph like [31, 50]. A simple, native schema visualization excluding e.g., multi-labeling is possible in Neo4j. An exemplary UI based on expert interviews about how to visualize the schema of property graphs is presented in [7]. We plan to implement a similar schema illustration as graph and use it to demonstrate the impact of evolution on both the schema and data level. Additionally, the visualization will be used to highlight changes and compare the first and latest version.

In [51] a graph-like illustration for constraints and queries is recommended and traceability links for multi-stores are considered in [27]. Even though the illustration of queries in a graph-like way can be rather helpful, we believe that additionally the output schema together with the estimated output graph data are needed in the evolution process.

Graph Data Profiles. Profiling data provides rich insight into the data being used for a variety of tasks [19, 20]. In [47], **t**uple-**g**enerating **d**ependencies (tgd) and similarity constraints are represented by the concept of Graph Generating Dependencies, demonstrating the importance of constraint detection when working with data profiles. The issue of graph dependencies is broached in [46], too. Queries to extract graph data profiles directly from the databases are presented in [39]. Here the functionalities of them are compared between relational databases and property graphs. The three major categories specified are single property tasks, graph patterns and dependencies. Some of the data discussed in [39] such as uniqueness or the absolute number of occurrences for entity

types will be integrated in the proposed structure profiles. The combination of schema and statistical information to highlight heterogeneity and the evolutionary impact is new.

4 PROPOSED APPROACH

The visualization of the schema and data statistics, illustrating heterogeneous parts of the dataset and showing data changes upon their comparison, are necessary. This eases their usage to new users [45]. All **R**esearch **Q**uestions – short RQs – will be analyzed under the aspect of the overarching RQ_O: *How to make graph database evolution accessible to a wider range of society while ensuring an easy usability?* In previous work [26], we answered **RQ1:** *How can evolution operations be described domain independent for graph databases?* (see Section 6.1). The emerging RQs are:

- **RQ2:** Which schema and statistical information is needed to define and control evolution and how to align the extracted data with a schema language?
- **RQ3:** How to estimate the effort (**3a**) and illustrate the impact of evolution and the affected elements (**3b**)?
- **RQ4:** How to optimize GMO_C and improve the performance if multiple evolution operations are executed?

4.1 Schema Management and Extraction

The **S**chema **M**anagement **L**ayer (SML) for **RQ2** is needed to **illustrate, explore, evolve and change the schema** and consequently provides the base for the *Evolution Module* (**RQ3**) shown in Figure 2. To accomplish the implementation of an SML and later on handle the extracted data, a **S**chema **L**anguage (SL) for the *Language Module* will be developed. The SL based on [8] is used to describe the implicit schema at different versions while taking all variations, caused by the highly interconnected data in graph databases, adequately into account. Schema constraints of existing graph databases like Neo4j, TigerGraph or OrientDB will be considered, too, enabling the detection of violations.

Graph Schema - Formal Definition. A graph G can be defined as a tuple containing a set of vertices (= node) V, edges (= relationships) E and the databases name n_{db} analogous to [21].

$$G = (n_{db}, V, E)$$

From the graph G a graph schema S_G is extracted whereby f represents a schema extraction function for graph data which has to be developed. Similar to extraction functions for other data models [32], it derives from the implicit structures of each data an explicit schema of the whole graph database containing information about nodes and relationships. S_G is a tuple consisting of a schema for both nodes S_V and relationships S_E.

$$f : G \rightarrow S_G$$
$$S_G = (S_V, S_E)$$

In contrast to [21] both S_V and S_E include the element *id* with a uniqueness condition. *id* is defined as a special kind of property as it is automatically added by Neo4j upon the creation of any entity and cannot be changed manually by the user. Moreover, properties are outlined as P with the distinction of being only the property name n_p. This limitation is due to schema evolution affecting only the property names n_p. Nodes, in addition, inherit a set of labels L and associates edges S_E, while relationships enclose a type t which cannot be empty. L, D, L_{SN}, L_{EN} are all of type String. Their direction D is defined as boolean and thus can take two states: 1) ingoing or 2) outgoing. The start node is specified by a set of its labels L_{SN}. Same goes for the end node L_{EN}.

$$S_V = \{(id, L, P|_{n_p}, S_E) \big| \text{ unique id}\}$$
$$S_E = \{(id, t, D, L_{SN}, L_{EN}, P|_{n_p}) \mid t \neq \text{' '}, \text{ unique id}\}$$

Concept. The SML is the base for the extraction of the graph's schema and the structure profiles. This enables visualizing the initial data and comparing $schema_n$ and $schema_{n+1}$. The schema itself is described through the SL. Moreover, the SML will be used to handle constraints. To illustrate the impact of evolution on the schema via **RQ3b**, an interactive graph schema visualization (*Visualization Module*) will be displayed. This includes showing nodes S_V and their connections S_E, while the embedded entity type schema will be shown on-click. The illustration will be graph-like like in Figure 1.

Extension in Nautilus. An algorithm to extract the initial schema information directly from the database via Python will be designed as described in Section 3. A JSON file will be used to easily extract schema differences. The data will be parsed frontend using JavaScript, for an interactive integration in the UI for the *Visualization* and *Preview Module* (see Figure 2). Additionally, the extracted data will be used to detect and handle schema constraints, informing a user about violations.

4.2 Structure Profiles

Besides the schema, *structure profiles* will be used to visualize the data. Structure profiles are defined as relative amount of schema features, demonstrating whether a feature is optional or mandatory. Features are labels, types and property keys.

Concept. Based on the SML of **RQ2** a preview of the structure profiles at version n+1 as part of the schema visualization will be created. The profiles are utilized to show the impact together with changes caused by evolution, in order to answer **RQ3a+b**. Based on the structure profiles, schema constraints are detected by an algorithm that searches for mandatory and optional elements. Due to graph databases offering a schema-less environment, such information needs to be extracted and visualized schema-second on a given dataset. Outlining optional elements simplifies the detection of structural errors. GEO can be used to either perform schema evolution or to make updates via GEO's filter functions which could either cause or adjust heterogeneity. Figure 3 illustrates that the creation of constraints based on $schema_n$ is part of this work package. Mandatory elements play a major role because they define whether a constraint is necessary or not. This information is forwarded to the user upon a possible violation (e.g., actions relaxing the schema).

Extension in Nautilus. An algorithm to calculate the frequency for each structural component will be developed. This information then is used for the structure profiles. Those will be displayed frontend to inform the user of optional and required elements. For elements with an occurrence of 100%, a constraint will automatically be created. Constraints can either be saved in a separate file or directly created on the database. In the latter case they cannot

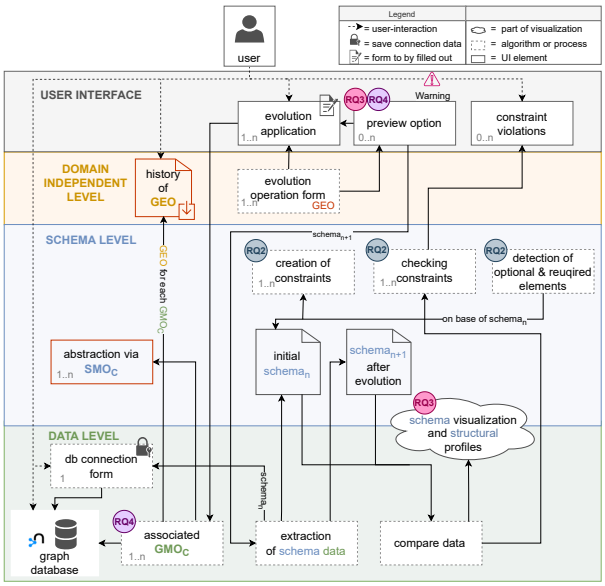

Figure 3: Schema Management Layer (RQ2) and Preview Option (RQ3) to Estimate the Impact of Evolution

be violated. This could be an issue as it might not be intended to force a schema-first approach upon the user, but rather to make the user aware that an action will result in a schema change. Therefore, warnings are created, to preserve the flexibility and capabilities provided by graph databases. This approach aims to prevent structural errors from happening while giving a better overview over the data already available, ensuring the self-determination of the human in the loop.

4.3 Evolution

A user cannot estimate the degree of change evolution might cause, which is why a quantitative (through the structure profiles) and a visual preview option (*Preview Module* of Figure 2) are essential.

Concept. The extracted schema and statistical data from **RQ2** will be utilized to display the schema emerging after an evolution operation ($schema_{n+1}$) in the *Preview Module*. The graph-like illustration is part of **RQ3b** to show the impact of evolution. Seeing the output before taking action is especially helpful for inexperienced users. The impact of evolution can range from

(1) none (defined pattern not available in the database),
(2) some (required elements become optional and vice versa, resulting in a relaxed schema)
(3) to all (e.g., split of all nodes with label x).

There will also be a display of the estimated execution time. The structure profiles, presented in Section 4.2, are used to estimate and display the affected elements through the *Preview Module* (**RQ3a**). After submitting the evolution form an interactive schema graph at version n+1 is shown – identical to the preview of the graph schema visualization.

Extension in Nautilus. Figure 3 shows a detailed architecture of how the preview will be integrated. As an extension, the general schema will be displayed as a graph S_G, due to the findings of [7]. Detected schema differences will be directly highlighted, to ensure a quick overview over schema changes. A focus algorithm is planed, to directly visualize schema changes. The data extracted by the algorithm of Section 4.1 will be used to integrate a reusable visualization of the schema before and after the evolution. The profiles are also integrated to show required and optional elements. Moreover, the schema visualization is planned to be interactive, allowing the user to execute evolution operations directly by manipulating the schema graph and returning the associated GMO_C.

4.4 Optimization

RQ4 focuses on addressing optimization. To optimize GMO_C, a query optimization needs to be done. This includes relational as well as graph specific rules [24, 38]. Work in the context of Knowledge Graphs like [1, 6, 29] will be considered, too. Accordingly, Nautilus will be extended by another graph databases system such as OrientDB or TigerGraph. Especially, when complex workarounds are used to perform evolution operations such as *split*, the execution time is higher as for single-type operations using single-line commands. Consequently, optimizing these queries will increase the performance. The same goes for workarounds using a paired approach in combination with Python for collection handling. In such cases, the performance of the Python code has to be measured and considered additionally.

Another important point is composition, if several evolution operations are executed in one submit [33, 40]. This helps catch operations that are nullified like renaming a label from `Student` to `Person` and later on back to `Student`, resulting in the same output as if no evolution would have taken place.

5 PLAN FOR EVALUATION

As part of RQ_O user studies are planed to evaluate the evolution language and the UI as well as the functionality of Nautilus. Nautilus strives to not only provide an easy to use UI, but also a simplified understanding of the evolution operations through GEO. Since GEO plays a major part, it is crucial that the description of what each evolution operation does, is easy to comprehend. Task-based studies are planned to evaluate the usability of the UI while a thinking aloud approach ensures the identification of the component (GEO and UI) causing issues [35, 37, 43]. To measure the workload the NASA-TLX will be used [41]. Besides that, a learning and testing phase will be integrated to identify precision and recall [36]. Testing is performed after the implementation of each component presented in Section 4 to identify and resolve issues in a timely manner. Moreover, we will conduct an eye-tracking study to identify any potential issues related to GEO.

To quantitatively analyse the limitations, the time take vs. estimated as well as the number of elements affected vs. estimated by each operation will be compared. This ensures the quality of the estimations made. To test the reliability of Nautilus's functionalities native graph datasets of different sizes will be used, like the Neo4j example datasets[1].

[1]Git repository: neo4j-graph-examples

6 CURRENT STATUS

We completed answering **RQ1** *How can evolution operations be described domain independent for graph databases?* by developing the evolution language GEO. Moreover, we implemented GEO in a first version of our prototype called Nautilus.

6.1 Domain Independent Evolution Language

To answer **RQ1** we developed a formal language with an intuitive syntax called GEO [26] – short for **G**raph **E**volution **O**peration. This evolution language is integrated in our tool named Nautilus and is part of the domain independent level in Figure 2. The operations defined by GEO are similar to those of other database systems. While the `alter table` statements for relational databases are applicable to specify single-type evolution operations, for document databases, more complex single and multi-type evolution operations are already available showing their importance e.g., [16, 34, 49]. In contrast, the implementation of our evolution language GEO in Nautilus focuses on graph databases, adapting the operations in a graph-specific context. In addition to these works, we implemented a first collection of filter functions. From the domain independent GEO derives the platform-dependent **S**chema **M**odification **O**peration – SMO_C, through which a precise implementation in Neo4j's query language Cypher – GMO_C – is possible. The domain independent GEO intuitively describes what each evolution operation does, demonstrated in Figure 4 for copying labels. Figure 1 showcases a concrete graph schema for the same operation.

GEO - **G**raph **E**volution **O**peration

```
copyLabels ::= 'copy' label Professor 'to' node
    'with' label Manager
```

SMO_C - **S**chema **M**odification **O**peration

```
selectPattern (*to copy from*)
WITH LABELS(n) AS labels
selectPattern (*to copy to*)
addLabels YIELD node RETURN node
```

GMO_C - **G**raph **M**anipulation **O**peration

```
MATCH (n:Professor) WITH labels(n) AS labels
MATCH (n2:Manager)
CALL apoc.create.addLabels(n2, labels)
YIELD node RETURN node
```

Figure 4: The Evolution Operation "copy all labels" and its Realization in Neo4j

6.2 Nautilus – Implementation of GEO

The implementation of all parts discussed above will be integrated into our momentary program Nautilus[2] [25].

Schema Management and Extraction. An APOC function is momentarily utilized to extract the schema from a given database. Based on the schema extraction a visualization was implemented. In the context of RQ_O this ensures that users can directly see the data within the accessed database. The schema at version n+1 is visualized shown as table – partitioned by entity types – and an interactive diagram comparing the schema before and after the evolution is at hand.

Structural Database Statistics. The **precursor of the structure profiles** are called **S**tructural **D**atabase **S**tatistics (SDS). Theses are a hybrid approach, using both data statistics like frequency combined with schema information. The illustration of the SDS aims to help users to built GEO as all labels, types and property keys are displayed in the diagram, together with associated relationships for nodes. The visualization as scatter plot shows the data before and after the evolution, enabling users to see changes in quantity and schema information of each entity type. On-click the SDS are shown for each data point.

Evolution. Nautilus is based on our evolution language GEO, being visually present at all steps. Via the formal evolution language, an easy comprehension of each evolution operation is ensured. A log file, containing all performed GEOs, offers the convenience of traceability of the performed operations. Drop-down menus are used to minimize the occurrences of syntax errors and to reduce the necessary knowledge of GEO. The usage of evolution operations over time is displayed accordingly to [22].

Next Step: User Study. A task-based study using a **thinking aloud** approach was designed to gain first insights into the understandability of GEO and our UI. To evaluate our evolution language adequately we included a questionnaire making use of true-false statements to see how well GEO describes an operation. We will then re-engineer each component according to the findings.

7 CONCLUSION

In order to use a system successfully over a long period changes need to be made. Schema evolution is conducted through the evolution operations add, rename, delete, copy, move, split and merge. The benefit of graph databases enabling the storage of highly interconnected data, also is challenging as context is utmost important making a transform operation necessary. Especially challenging is the estimation of the impact of evolution on the database. To address this challenge, we will design a schema management tool, including a schema management layer, being the base of a schema visualization. Moreover, structure profiles – a hybrid approach making use of statistical and schema data – give insight into heterogeneous parts of the data. As query languages are complex, making professional knowledge necessary, our approach uses an easy-to-understand formal language called GEO. GEO is an evolution language capable of performing updates to parts of the graph data and consequently, covering different use cases than a query language lacking an evolution language. For the next step we will analyse the data gained through our first evaluation and take necessary steps of improvement for the UI as well as the evolution language's comprehensibility.

ACKNOWLEDGMENTS

This work has been funded by Deutsche Forschungsgemeinschaft (German Research Foundation) – 385808805.

[2]Project Website of Nautilus

REFERENCES

[1] Christian Aebeloe, Gabriela Montoya, and Katja Hose. 2022. The Lothbrok approach for SPARQL Query Optimization over Decentralized Knowledge Graphs. *CoRR* abs/2208.14692 (2022).

[2] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Alastair Green, Jan Hidders, Bei Li, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savkovic, Michael Schmidt, Juan Sequeda, Slawek Staworko, Dominik Tomaszuk, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Dusan Zivkovic. 2023. PG-Schema: Schemas for Property Graphs. *Proceedings of the ACM on Management of Data* 1, 2 (June 2023), 1–25. https://doi.org/10.1145/3589778

[3] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Keith W. Hare, Jan Hidders, Victor E. Lee, Bei Li, Leonid Libkin, Wim Martens, Filip Murlak, Josh Perryman, Ognjen Savkovic, Michael Schmidt, Juan F. Sequeda, Slawek Staworko, and Dominik Tomaszuk. 2021. PG-Keys: Keys for Property Graphs. In *SIGMOD Conference*. ACM, 2423–2436.

[4] Paolo Atzeni, Francesca Bugiotti, Luca Cabibbo, and Riccardo Torlone. 2020. Data modeling in the NoSQL world. *Comput. Stand. Interfaces* 67 (2020).

[5] Paolo Atzeni, Francesca Bugiotti, and Luca Rossi. 2014. Uniform access to NoSQL systems. *Inf. Syst.* 43 (2014), 117–133.

[6] Yushi Bai, Xin Lv, Juanzi Li, and Lei Hou. 2023. Answering Complex Logical Queries on Knowledge Graphs via Query Computation Tree Optimization. In *ICML (Proceedings of Machine Learning Research, Vol. 202)*. PMLR, 1472–1491.

[7] Nimo Beeren. 2022. Designing a Visual Tool for Property Graph Schema Extraction and Refinement: An Expert Study. *CoRR* abs/2201.03643 (2022). arXiv:2201.03643 https://arxiv.org/abs/2201.03643

[8] Angela Bonifati. 2023. The Quest for Schemas in Graph Databases (keynote). In *DOLAP (CEUR Workshop Proceedings, Vol. 3369)*. CEUR-WS.org, 1–2.

[9] Angela Bonifati, Stefania-Gabriela Dumbrava, Emile Martinez, Fatemeh Ghasemi, Malo Jaffré, Pacome Luton, and Thomas Pickles. 2022. DiscoPG: Property Graph Schema Discovery and Exploration. *Proc. VLDB Endow.* 15, 12 (2022), 3654–3657.

[10] Angela Bonifati, Stefania Dumbrava, and Nicolas Mir. 2022. Hierarchical Clustering for Property Graph Schema Discovery. In *Proc. EDBT*. OpenProceedings.org, 2:449–2:453.

[11] Angela Bonifati, Peter Furniss, Alastair Green, Russ Harmer, Eugenia Oshurko, and Hannes Voigt. 2019. Schema Validation and Evolution for Graph Databases. In *ER (Lecture Notes in Computer Science, Vol. 11788)*. Springer, 448–456.

[12] Angela Bonifati, Peter Furniss, Alastair Green, Russ Harmer, Eugenia Oshurko, and Hannes Voigt. 2019. Schema Validation and Evolution for Graph Databases. https://arxiv.org/abs/1902.06427

[13] Soumaya Boukettaya, Ahlem Nabli, and Faïez Gargouri. 2018. Towards the Evolution of Graph Oriented Databases. In *ISDA (2) (Advances in Intelligent Systems and Computing, Vol. 941)*. Springer, 392–399.

[14] Soumaya Boukettaya, Ahlem Nabli, and Faïez Gargouri. 2020. Data Migration in Graph-oriented Databases. *Res. Comput. Sci.* 149, 10 (2020), 317–336.

[15] Arnaud Castelltort and Anne Laurent. 2013. Representing history in graph-oriented NoSQL databases: A versioning system. In *ICDIM*. IEEE, 228–234.

[16] Alberto Hernández Chillón, Meike Klettke, Diego Sevilla Ruiz, and Jesús García Molina. 2022. A Taxonomy of Schema Changes for NoSQL Databases. *CoRR* abs/2205.11660 (2022).

[17] Anthony Cleve, Maxime Gobert, Loup Meurice, Jerome Maes, and Jens H. Weber. 2015. Understanding database schema evolution: A case study. *Sci. Comput. Program.* 97 (2015), 113–121.

[18] Isabelle Comyn-Wattiau and Jacky Akoka. 2017. Model driven reverse engineering of NoSQL property graph databases: The case of Neo4j. In *IEEE BigData*. IEEE Computer Society, 453–458.

[19] Will Epperson, Vaishnavi Gorantla, Dominik Moritz, and Adam Perer. 2024. Dead or Alive: Continuous Data Profiling for Interactive Data Science. *IEEE Trans. Vis. Comput. Graph.* 30, 1 (2024), 197–207. https://doi.org/10.1109/TVCG.2023.3327367

[20] Besnik Fetahu, Stefan Dietze, Bernardo Pereira Nunes, Davide Taibi, and Marco Antonio Casanova. 2013. Generating structured Profiles of Linked Data Graphs. In *Proceedings of the ISWC 2013 Posters & Demonstrations Track, Sydney, Australia, October 23, 2013 (CEUR Workshop Proceedings, Vol. 1035)*, Eva Blomqvist and Tudor Groza (Eds.). CEUR-WS.org, 113–116. https://ceur-ws.org/Vol-1035/iswc2013_demo_29.pdf

[21] Angelo Augusto Frozza, Salomão Rodrigues Jacinto, and Ronaldo dos Santos Mello. 2020. An Approach for Schema Extraction of NoSQL Graph Databases. In *IRI*. IEEE, 271–278.

[22] Fanis Giachos, Nikos Pantelidis, Christos Batsilas, Apostolos V. Zarras, and Panos Vassiliadis. 2023. Parallel lives diagrams for co-evolving communities and their application to schema evolution. In *ER (Companion) (CEUR Workshop Proceedings, Vol. 3618)*. CEUR-WS.org.

[23] Maxime Gobert, Loup Meurice, and Anthony Cleve. 2023. Modeling, manipulating and evolving hybrid polystores with HyDRa. *Sci. Comput. Program.* 230 (2023), 102972.

[24] Maxime Gobert, Csaba Nagy, Henrique Rocha, Serge Demeyer, and Anthony Cleve. 2023. Best practices of testing database manipulation code. *Inf. Syst.* 111 (2023), 102105.

[25] Dominique Hausler and Meike Klettke. 2024. Nautilus: Implementation of an Evolution Approach for Graph Databases. In *MoDELS (Companion)*. IEEE.

[26] Dominique Hausler, Meike Klettke, and Uta Störl. 2023. A language for graph database evolution and its implementation in Neo4j. In *ER (Companion) (CEUR Workshop Proceedings, Vol. 3618)*. CEUR-WS.org.

[27] Ábel Hegedüs, Ákos Horváth, István Ráth, Rodrigo Rizzi Starr, and Dániel Varró. 2016. Query-driven soft traceability links for models. *Softw. Syst. Model.* 15, 3 (2016), 733–756.

[28] Andrea Hillenbrand, Maksym Levchenko, Uta Störl, Stefanie Scherzinger, and Meike Klettke. 2019. MigCast: Putting a Price Tag on Data Model Evolution in NoSQL Data Stores. In *SIGMOD Conference*. ACM, 1925–1928.

[29] Sonia Horchidan. 2023. Query Optimization for Inference-Based Graph Databases. In *PhD@VLDB (CEUR Workshop Proceedings, Vol. 3452)*. CEUR-WS.org, 33–36.

[30] ISO/IEC 39075:2024 2024. *Information technology— Database languages — GQL*. Standard. International Organization for Standardization, Geneva, CH.

[31] Weihao Jiang, Li Yan, Yaofeng Tu, Xiangsheng Zhou, and Zongmin Ma. 2022. PG-explorer: Resource Description Framework data exploration with property graphs. *Expert Syst. Appl.* 198 (2022), 116789. https://doi.org/10.1016/J.ESWA.2022.116789

[32] Meike Klettke, Uta Störl, and Stefanie Scherzinger. 2015. Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores. In *BTW (LNI, Vol. P-241)*. GI, 425–444.

[33] Meike Klettke, Uta Störl, Manuel Shenavai, and Stefanie Scherzinger. 2016. NoSQL schema evolution and big data migration at scale. In *IEEE BigData*. IEEE Computer Society, 2764–2774.

[34] Pavel Koupil, Jáchym Bártík, and Irena Holubová. 2022. *MM-evocat:* A Tool for Modelling and Evolution Management of Multi-Model Data. In *CIKM*. ACM, 4892–4896.

[35] Rebecca Krosnick, Fraser Anderson, Justin Matejka, Steve Oney, Walter S. Lasecki, Tovi Grossman, and George Fitzmaurice. 2021. Think-Aloud Computing: Supporting Rich and Low-Effort Knowledge Capture. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 199, 13 pages. https://doi.org/10.1145/3411764.3445066

[36] Tobias Kuhn. 2013. The understandability of OWL statements in controlled English. *Semantic Web* 4, 1 (2013), 101–115.

[37] Clayton Lewis. 1982. *Using the "Thinking-aloud" Method in Cognitive Interface Design*. IBM Thomas J. Watson Research Division.

[38] Bingqing Lyu, Xiaoli Zhou, Longbin Lai, Yufan Yang, Yunkai Lou, Wenyuan Yu, and Jingren Zhou. 2024. A Graph-Native Query Optimization Framework. arXiv:2401.17786 [cs.DB] https://arxiv.org/abs/2401.17786

[39] Sofía Maiolo, Lorena Etcheverry, and Adriana Marotta. 2020. Data Profiling in Property Graph Databases. *ACM J. Data Inf. Qual.* 12, 4 (2020), 20:1–20:27.

[40] Mark Lukas Möller, Dominique Hausler, Sebastian Strasser, Tanja Auge, and Meike Klettke. 2023. Heterogeneity in NoSQL Databases - Challenges of Handling schema-less Data. In *LWDA (CEUR Workshop Proceedings, Vol. 3630)*. CEUR-WS.org, 134–145.

[41] NASA. 2012. NASA-TLX - Task Load Index. https://humansystems.arc.nasa.gov/groups/TLX/index.php Accessed: 2024-07-03.

[42] Neo4j, Inc. 2024. Constraints. https://neo4j.com/docs/cypher-manual/current/constraints/ Accessed: 2024-05-07.

[43] Nielsen, Jakob. 2012. Thinking Aloud: The #1 Usability Tool. https://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/ Accessed: 2024-07-03.

[44] Jaroslav Pokorný, Michal Valenta, and Jirí Kovacic. 2017. Integrity constraints in graph databases. In *ANT/SEIT (Procedia Computer Science, Vol. 109)*. Elsevier, 975–981.

[45] Chandan Sharma and Roopak Sinha. 2019. A Schema-First Formalism for Labeled Property Graph Databases: Enabling Structured Data Loading and Analytics. In *BDCAT*. ACM, 71–80.

[46] Larissa Capobianco Shimomura, George H. L. Fletcher, and Nikolay Yakovets. 2023. ProGGD - Data Profiling on Knowledge Graphs using Graph Generating Dependencies. In *Proceedings of the ISWC 2023 Posters, Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 22nd International Semantic Web Conference (ISWC 2023), Athens, Greece, November 6-10, 2023 (CEUR Workshop Proceedings, Vol. 3632)*, Irini Fundulaki, Kouji Kozaki, Daniel Garijo, and José Manuél Gómez-Pérez (Eds.). CEUR-WS.org. https://ceur-ws.org/Vol-3632/ISWC2023_paper_434.pdf

[47] Larissa Capobianco Shimomura, Nikolay Yakovets, and George Fletcher. 2024. GGDMiner - Discovery of Graph Generating Dependencies for Graph Data Profiling. *CoRR* abs/2403.17082 (2024). https://doi.org/10.48550/ARXIV.2403.17082 arXiv:2403.17082

[48] Uta Störl and Meike Klettke. 2022. Darwin: A Data Platform for Schema Evolution Management and Data Migration. In *EDBT/ICDT Workshops (CEUR Workshop Proceedings, Vol. 3135)*. CEUR-WS.org.

[49] Pablo Suárez-Otero, Michael J. Mior, María José Suárez Cabal, and Javier Tuya. 2023. CoDEvo: Column family database evolution using model transformations.

J. Syst. Softw. 203 (2023), 111743.

[50] Michael Thane, Kai M. Blum, and Dirk J. Lehmann. 2023. CatNetVis: Semantic Visual Exploration of Categorical High-Dimensional Data with Force-Directed Graph Layouts. In *25th Eurographics Conference on Visualization, EuroVis 2023 - Short Papers, Leipzig, Germany, June 12-16, 2023*, Thomas Höllt, Wolfgang Aigner, and Bei Wang (Eds.). Eurographics Association, 91–95. https://doi.org/10.2312/EVS.20231049

[51] Zoltán Ujhelyi, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, Benedek Izsó, István Ráth, Zoltán Szatmári, and Dániel Varró. 2015. EMF-IncQuery: An integrated development environment for live model queries. *Sci. Comput. Program.* 98 (2015), 80–99.