International Journal on Digital Libraries (2022) 23:113–132
https://doi.org/10.1007/s00799-021-00317-8

# SchenQL: in-depth analysis of a query language for bibliographic metadata

Christin Katharina Kreutz[1] · Michael Wolz[1] · Jascha Knack[1] · Benjamin Weyers[1] · Ralf Schenkel[1]

Received: 30 April 2021 / Revised: 14 October 2021 / Accepted: 14 October 2021 / Published online: 23 November 2021
© The Author(s) 2021

**Abstract**
Information access to bibliographic metadata needs to be uncomplicated, as users may not benefit from complex and potentially richer data that may be difficult to obtain. Sophisticated research questions including complex aggregations could be answered with complex SQL queries. However, this comes with the cost of high complexity, which requires for a high level of expertise even for trained programmers. A domain-specific query language could provide a straightforward solution to this problem. Although less generic, it can support users not familiar with query construction in the formulation of complex information needs. In this paper, we present and evaluate SchenQL, a simple and applicable query language that is accompanied by a prototypical GUI. SchenQL focuses on querying bibliographic metadata using the vocabulary of domain experts. The easy-to-learn domain-specific query language is suitable for domain experts as well as casual users while still providing the possibility to answer complex information demands. Query construction and information exploration are supported by a prototypical GUI. We present an evaluation of the complete system: different variants for executing SchenQL queries are benchmarked; interviews with domain-experts and a bipartite quantitative user study demonstrate SchenQL's suitability and high level of users' acceptance.

**Keywords** Domain-specific query language · Bibliographic metadata · Digital libraries · Graphical user interface

## 1 Introduction

Scientific writing almost always starts with a thorough bibliographic research on relevant papers, authors, conferences, journals and institutions. While web search is excellent for question answering and intuitively performed, not all retrieved information is correct, unbiased and categorised [3]. The arising problem is people's tendency of rather using poor information sources that are easy to query than more reliable sources which might be harder to access [4]. This introduces the need for more formal and also structured information sources such as digital libraries specialised in the underlying data that at the same time need to be easy to query.

Currently existing interfaces of digital libraries often provide keyword search on metadata or offer to query attributes [15,25]. However, in many cases, these interfaces do not allow to directly express more advanced queries such as *"Which are the five most cited articles written by person P about topic T after year Y?"*, but require complex interaction. Popular examples of such limited systems are dblp[1] [25] or Semantic Scholar.[2] More complex tools, e.g. GrapAL[3] [7], are capable of answering said complex queries, but come with complex and often not very intuitive query languages. Another option would be to use structured query languages such as SQL, a widespread language for querying databases, which unfortunately tends to be difficult to master [37]. This is critical as in most cases domain-experts are familiar with the schema of the data but are not experienced in using all-purpose query languages such as SQL [1,26]. This is even worse for casual users of digital libraries who

✉ Christin Katharina Kreutz
kreutzch@uni-trier.de

Benjamin Weyers
weyers@uni-trier.de

Ralf Schenkel
schenkel@uni-trier.de

1 Trier University, 54286 Trier, DE, Germany

1 https://dblp.uni-trier.de/.
2 https://www.semanticscholar.org/.
3 https://grapal.allenai.org/.

neither have knowledge of the structure of the data nor of SQL.

To close this gap, we present the SchenQL Query Language, in short SchenQL[4], for the domain of bibliographic metadata [21,22]. SchenQL is designed to be easily utilised by experts as well as casual users from the domain as it uses the vocabulary of digital libraries in its syntax. While domain-specific query languages (DSLs) provide a multitude of advantages [9], the most important aspect in the conception of SchenQL was that no programming skills or database schema knowledge is required to use it. For SchenQL to be widely applicable, we introduce a prototypical graphical user interface (the SchenQL GUI) which supports the construction of queries and offers visualisations of query results and an additional dimension of retrieving information by exploring data and its relations through clicking. As an example of SchenQL, the aforementioned question, can be formulated as follows: `MOST CITED (ARTICLES WRITTEN BY "P" ABOUT "T" AFTER Y) LIMIT 5.`

In addition to the SchenQL query language, another major contribution of this paper is the empirical evaluation of SchenQL as domain-specific query language on bibliographic metadata including the investigation of a prototypical GUI that is designed to assist users in creating queries. SchenQL is evaluated threefold: (1) query execution times were benchmarked to underline the suitability for interactive retrieval tasks, (2) interviews with domain-experts were conducted to identify applications as well as options for further development and (3) a quantitative user study consisting of two parts measured effectiveness, efficiency and users' satisfaction with our whole system: we first evaluated the usage of command line SchenQL against SQL, followed by a study which compared the usage of the SchenQL GUI to the previous results. Here, the User Experience Questionnaire [34] was conducted for assessing of users' experience.

The remainder of this paper is structured as follows: Sect. 2 discusses related work. Section 3 introduces the structure and syntax of SchenQL with a special focus on the implementation including the presentation of the SchenQL Parser, Compiler and Front End. The system is evaluated in three parts in Sect. 4. Section 5 describes possible future research.

This paper is an extended version of the work presented at ICADL'20 [22]. The main extensions are contained in Sects. 3.4 and 4.1.

## 2 Related work

Areas adjacent to the one we are tackling are *search on digital libraries*, *search interfaces on bibliographic metadata*, *formalised query languages* and *domain-specific query languages*.

For *search on digital libraries*, the MARC format is a standard for information exchange [3]. While it is useful for known-item search, topical search might be problematic as contents of the corresponding fields can only be interpreted by domain-experts [3]. Most interfaces on digital libraries provide a field-based Boolean search [33] which can lead to difficulties in formulating queries that require the definition and concatenation of multiple attributes. This might cause a substantial cognitive workload on the user [6]. In contrast, withholding or restriction of faceted search on these engines fails to answer complex search tasks [5]. Thus, we focus on a search of topical information that even casual users can utilise while also offering the possibility to clearly define search terms for numerous attributes in a single query.

Several *search interfaces on bibliographic metadata* exist, the most well-known ones might be dblp [19,25], Bibsonomy [15], Google Scholar,[5] ResearchGate[6] or Semantic Scholar. All of those systems allow for a systematic refinement of result sets by the application of filter options via facets to varying extends. Only dblp and Semantic Scholar (on a small scale) support search on venues. The formulation of complex queries with aggregations is not targeted by any of them. In contrast, SchenQL supported by a GUI specialises on these functionalities. GrapAL[7] [7] actually provides all functions of SchenQL but is a complex tool utilising the Cypher [13] query language (QL).

*Domain-specific query languages* can come in various shapes. They can be SQL-like [24], visual QLs [1,11] or use a domain-specific vocabulary [36] but are typically specialised on a certain area. They also come in different complexities: for example, MathQL [14] is a query language in markup style on RDF repositories, but a user needs to be mathematician to be able to operate it. The DSL proposed by Madaan [26] stems from the medical domain and is designed to be used by inexperienced patients as well as medical staff. Some DSLs are domain-unspecific such as the aforementioned Cypher [13], BiQL [12] or SnQL [27] and depend on complicated SQL-like syntax. Naturally, there are hybrid forms: some natural language to machine-readable query options are domain-specific [32] and some DSLs might be transferable to other domains [9]. With our SchenQL system, we provide a QL which uses vocabulary from the domain of bibliographic metadata while being useful for experts as well as casual users and avoiding complicated syntax.

[4] The name SchenQL is a pun on the name Ralf Schenkel and gives kudos to him as he proposed the first version of the language's grammar.

[5] https://scholar.google.com/.

[6] https://www.researchgate.net.

[7] https://grapal.allenai.org/.

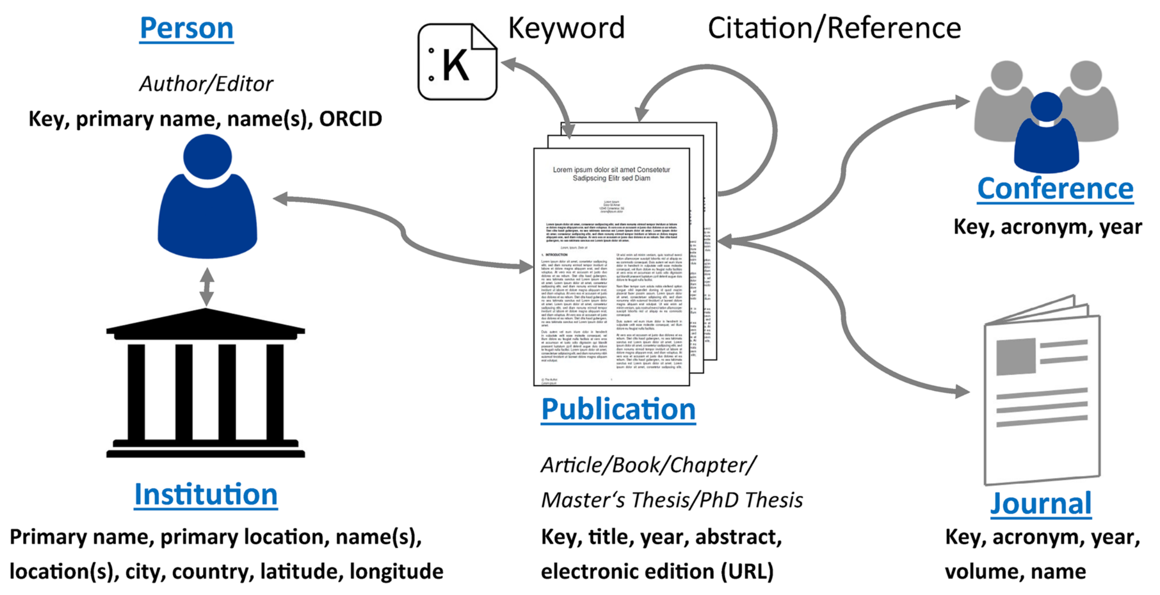


**Fig. 1** SchenQL database model with relations, base concepts (underlined), specialisations (italic) and selected attributes (bold)

## 3 SchenQL: QL and GUI

For simplicity, we refer to SchenQL including its GUI as the *SchenQL system*. SchenQL was developed to access bibliographic metadata textually, which resembles natural language for casual as well as expert users of digital libraries [21,22]. The fundamental idea is to hide complex syntax behind plain domain-specific vocabulary. This enables usage from anyone versed in the vocabulary of the domain without experience in sophisticated query languages such as SQL. The prototypical GUI supports SchenQL: it helps in query formulation with the auto-completion and keyword suggestion. Additionally, it provides visual exploration of query results supporting two standard visualisations: Ego Graph [31] and BowTie [18].

### 3.1 Data model

For our data model (see Fig. 1), we assume bibliographic metadata consists of persons and the publications they authored or edited. These persons can be affiliated with certain institutions. Publications can be of multiple types and may be published in conferences or journals. Publications can reference previously published papers and might be cited themselves by more recent work building upon them.

*Persons* can both be authors and editors of publications and might be working for institutions. For persons, we assume a unique key, their primary name, possible other names and their ORCID are given. For *institutions*, we model their primary name, primary location, further names and locations as well as the location of the institution in form of city, country, latitude and longitude. *Publications* can be either of type article, book, chapter, Master's thesis or PhD thesis. For publications, we assume a unique key, the title and publication year, abstract and electronic editions can be available. Publications can be associated with keywords. We additionally model links to referenced and citing papers as well as authors/editors of the publications and the publication venue. As venues for the publications, we consider *conferences* and *journals*. For both of them, we model a unique key, acronym and a specific year. For journals, we also store volume and name information.

Figure 2 shows the SchenQL relational data model.

### 3.2 Building blocks

Base concepts are the basic return objects of SchenQL. A base concept is connected to an entity of the data set and has multiple attributes. Those base concepts are `publications`, `persons`, `conferences`, `journals` and `institutions`. Upon these concepts, queries can be constructed. Base concepts can be specialised. For example, `publications` can be refined by the specialisations `books`, `chapters`, `articles`, `master` or `PhD theses`. A specialisation can be used instead of a base concept in a query.

Filters can restrict base concepts by extracting a subset of the data. Literals can be used as identifiers for objects from base concepts, and they can be utilised to query for specific data. Attributes of base concepts can be queried; for an overview of attributes, see Fig. 1. Table 1 gives an overview of literals, specialisations, filters and the standard return value for every base concept. Queries with strings as filter parameters, e.g. titles or names, utilise exact matching

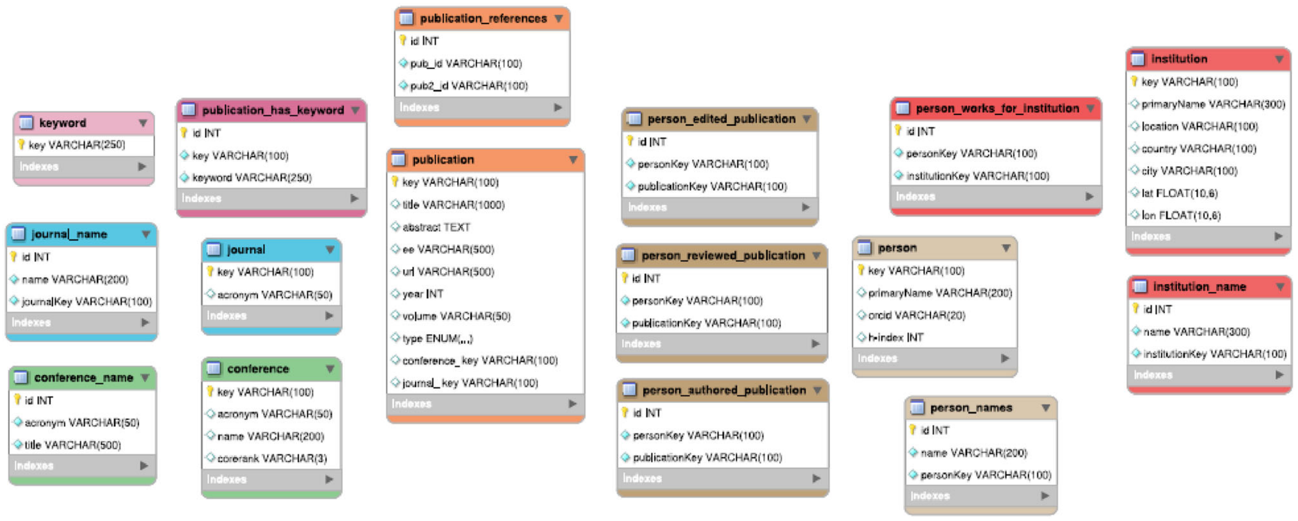


**Fig. 2** SchenQL relational data model (online in colour)

in general. Prepending a ∼ to such a query enables full-text search in case of titles: `PUBLICATIONS TITLED` ∼ `"DAFFODIL"`. If ∼ is used with a following person name, it provides the functionality of the original Soundex algorithm. Keywords as well as strings are case- and accent-insensitive.

Functions can be used to aggregate data or offer domain-specific operations. Right now, SchenQL provides four functions: `MOST CITED`, `COUNT`, `KEYWORD OF` and `COAUTHOR OF`. `MOST CITED (PUBLICATION)` can be applied on publications. This function returns titles as well as numbers of citations of papers in the following set. By default, the top five results are returned. `COUNT` returns the number of objects contained in the following sub-query. `KEYWORD(S) OF (PUBLICATION | CONFERENCE | JOURNAL)` returns the keywords associated with the following base concept. The next function `COAUTHOR(S) OF (PERSON)` returns the co-authors of an author. The `LIMIT` $x$ operator with $x \in \mathbb{N}$ can be appended at the end of any query to change the number of displayed results to at most $x$.

## 3.3 Syntax

The syntax of SchenQL follows simple rules resulting in queries similar to natural language which are aiming at simple construction. Sub-queries have to be surrounded by parentheses. It is possible to write singular or plural when using base concepts or specialisations (e.g. `JOURNAL` or `JOURNALS`). Filters following base concepts or their specialisations can be in arbitrary order and get connected via conjunction if not specified otherwise. (`OR` and `NOT` are also possible.) Most filters expect a base concept as their parameter (e.g. `WRITTEN BY (PERSONS)`); however, some filters anticipate a string as their parameter (e.g. `COUNTRY "de"`). Specialisations can be used in place of base concepts. Instead of a query `PERSON NAMED "Ralf Schenkel"`, a specialisation like `AUTHOR NAMED "Ralf Schenkel"` would be possible. If a filter requires a base concept, parentheses are needed except for the case of using literals for identifying objects of the base concept. For example, `PUBLICATIONS WRITTEN BY "Ralf Schenkel"` is semantically equivalent to `PUBLICATIONS WRITTEN BY (PERSONS NAMED "Ralf Schenkel")`. Attributes of base concepts can be accessed by putting the queried for attribute(s) in front of a base concept and connecting both parts with an `OF` (e.g. `"name", "acronym" OF CONFERENCES ABOUT KEYWORDS ["DL", "QLs"]`).

## 3.4 Implementation

The SchenQL system contains four main components (see Fig. 3). The *SchenQL DB Parser* parses all the different data sources and combines them in a MySQL database, the *SchenQL CLI* is the command line interface that also contains the *SchenQL Compiler* for the query language, the *SchenQL Front End* represents the web interface (introduced in Sect. 3.4.4), and the *SchenQL API* connects the SchenQL CLI with the SchenQL Front End. The SchenQL API also runs some direct queries on the database to execute high-level functions that SchenQL itself is not capable of. Our QL can be used in a terminal client similar to the MySQL shell or via the graphical front end.

### 3.4.1 SchenQL DB Parser

Our database model (see Fig. 1) was specifically designed for the syntax of SchenQL so that every base concept

**Table 1** SchenQL base concepts `Publications` (PU), `persons` (PE), `conferences` (C), `journals` (J) and `institutions` (I) with their respective literals (L), specialisations (S), filters (F) and standard return values (V, relevant for the CLI)

| | PUBLICATION | PERSON | CONFERENCE | JOURNAL | INSTITUTION |
|---|---|---|---|---|---|
| L | Key, title | Key, primary name, ORCID | Key, acronym | Key, acronym | |
| S | `MASTERTHESIS, BOOK, CHAPTER, PHDTHESIS, ARTICLE` | `AUTHOR, EDITOR` | | | |
| F | `PUBLISHED BY (I), ABOUT (keywords), WRITTEN BY (PE), EDITED BY (PE), APPEARED IN (C\|J), BEFORE year, IN YEAR year, AFTER year, TITLED title, REFERENCES (PU), CITED BY (PU)` | `PUBLISHED IN (C\|J), PUBLISHED WITH (I), WORKS FOR (I), NAMED name, ORCID orcid, AUTHORED (PU), REFERENCES (PU), CITED BY (PU)` | `ACRONYM acronym, ABOUT (keywords), BEFORE year, IN YEAR year, AFTER year` | `NAMED name, ACRONYM acronym, ABOUT (keywords), BEFORE year, IN YEAR year, AFTER year, VOLUME volume` | `NAMED name, CITY city, COUNTRY country, MEMBERS (PE)` |
| V | Title | Primary name | Acronym | Acronym | Primary name + location |

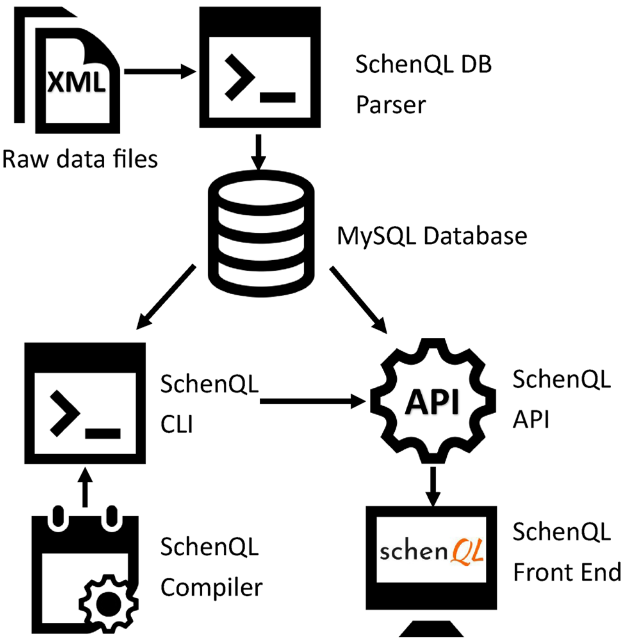


**Fig. 3** Overview of the SchenQL components

represents an entity in the database. Data on references and citations are contained in a single table. The chosen database uses the MyISAM storage-engine instead of the MySQL 8 default InnoDB. In comparison with InnoDB, MyISAM does not support transactions, so there is no need to commit after inserting data into the database. In case of SchenQL, transactions are not required, since no data are changed after the creation of the database. On the one hand, this strongly influences the performance of the database parser, and on the other hand, MyISAM has a higher support for full text search, which is necessary for queries like `PUBLICATIONS ABOUT "DL"` or `PUBLICATIONS TITLED ~ "Daffodil"`.

### 3.4.2 SchenQL CLI and Compiler

The SchenQL CLI defines the core of the QL. It does not only provide an interface to use the language, but it also includes the compiler. The compiler translates SchenQL queries to the target language SQL and uses Java Database Connectivity to run them against a MySQL 8.0.16 database holding the data.[8] We built the lexer and parser of our compiler using ANTLR with Java as the target language.[9] In the parser, we use the *visitor* approach to iterate through the nodes in the constructed parse tree.

[8] See Sect. 4.1 for an evaluation of the target language and the implementation of the SchenQL to SQL compiler.

[9] We utilise language specific functions in the lexer so that the grammar is no longer generally usable for other programming languages but would have to be adapted.

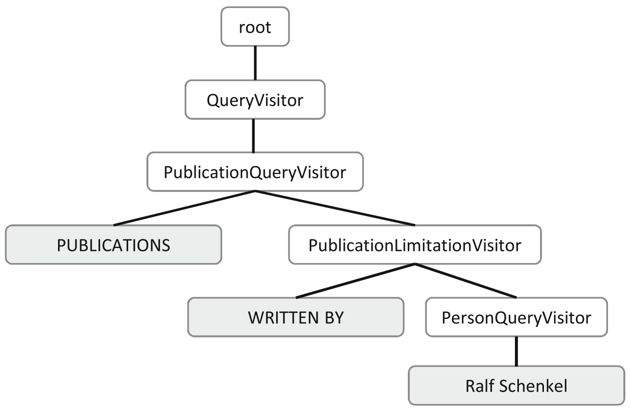


**Fig. 4** Syntax tree of the query `PUBLICATIONS WRITTEN BY "Ralf Schenkel"`

SQL queries are generated from SchenQL input in multiple steps: first, a SchenQL expression (for example, `PUBLICATIONS WRITTEN BY "Ralf Schenkel"`) is parsed and a parse tree is constructed (see Fig. 4). It represents the abstract syntax tree of the parsed input expression. Afterwards, this syntax tree is traversed from the root onwards. The parser calls the root node where it checks whether the input query is a request for one of the basic concepts or whether it is a function call (alias) to a sub-query, e.g. `COUNT`. Next, the child node of the root node is visited. In the case of the example from Fig. 4, the QueryVisitor is called, the type of the query is checked and the next child node is visited (in the example: PublicationQueryVisitor). The PublicationQueryVisitor processes the child nodes by depth-first search and collects all filters used in the query in an array using the (Publication)LimitationVisitor. Subsequently, the PublicationQueryVisitor generates a SQL select statement and adds the filters to it. It also checks whether a specialisation was used to call the query, i.e. if the user queried `ARTICLES WRITTEN BY "`*A*`"` instead of `PUBLICATIONS WRITTEN BY "`*A*`"`. This process is performed recursively until the input has been completely processed.

#### 3.4.3 SchenQL API

The SchenQL API handles all communication between the compiler and the front end. It has two tasks: first, it implements an endpoint for handling all types of valid SchenQL queries, and second, it serves additional information based on base concepts, such as retrieving citations and references for publications or authors for performance reasons. We use the REST architecture pattern for the API. The API is implemented using Spring and Spring Boot for handling HTTP-requests.

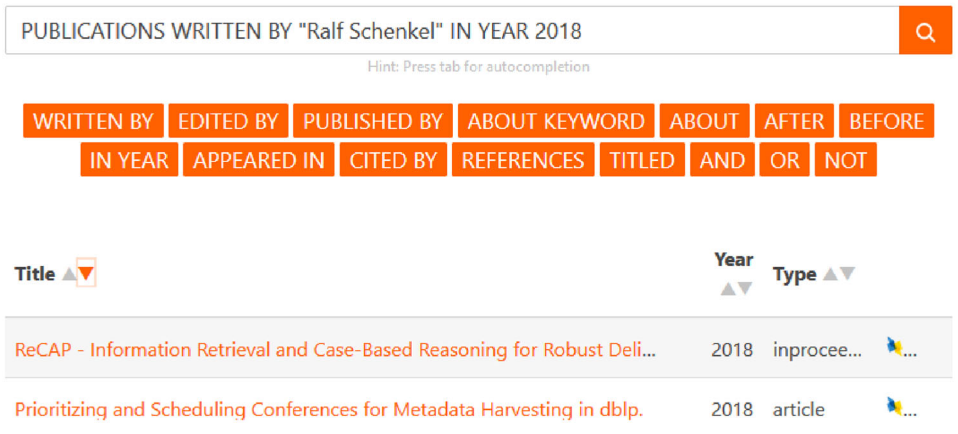
#### 3.4.4 SchenQL Frond End

The SchenQL Front End (also called GUI) is inspired by results from the qualitative study described in Sect. 4.3. It provides access to information by supporting the construction of queries including the interactive navigation with the GUI. It also offers auto-completion of SchenQL query keywords and suggestions for the formulation of queries. Results of queries can be sorted for every column of the result table. In Fig. 5, query formulation with suggested keywords and result representation in the SchenQL GUI is depicted. If a search result is selected by clicking on it, detail views open (see Fig. 6 for the detail view of a person) which offer all information available for the respective element of a base concept. Furthermore, we incorporated two already established visualisations: *Ego Graph* [31] and *BowTie* [18]. The *Ego Graph* for persons (see Fig. 6 top right part) supports the analysis of persons' most important co-authorships. At one glance, the most common cooperators are visualised and compared to each other such that the overall productivity and interdependence of a person can be estimated. If a person has many equally close collaborators, they might either be active in multiple fields or they could produce papers with many co-authors at once. If a person has only few very close co-authors and multiple further dependencies, this pattern could, for example, hint at a PhD student–supervisor relationship. The *BowTie* visualisation can be used for the easy estimation of a person's, publication's or venue's influence in terms of gained citations and its actuality (see Fig. 7). If for example a lot of recent papers are referenced (estimated by the detailed view) by a paper in focus, one could assume that this paper is well positioned in that time's publication landscape. The distribution of incoming citations could be very telling on whether, for example, a venue is still relevant to this day.

## 4 Evaluation

Before the actual evaluation of the SchenQL system, we conduct benchmarks for two possible database engine and target languages for the compilation of SchenQL queries: SQL (with data stored in a MySQL relational database) and Cypher (with data stored in a Neo4J graph database). Afterwards, we evaluate the performance of the current implementation of the compiler that translates SchenQL into the target query language.

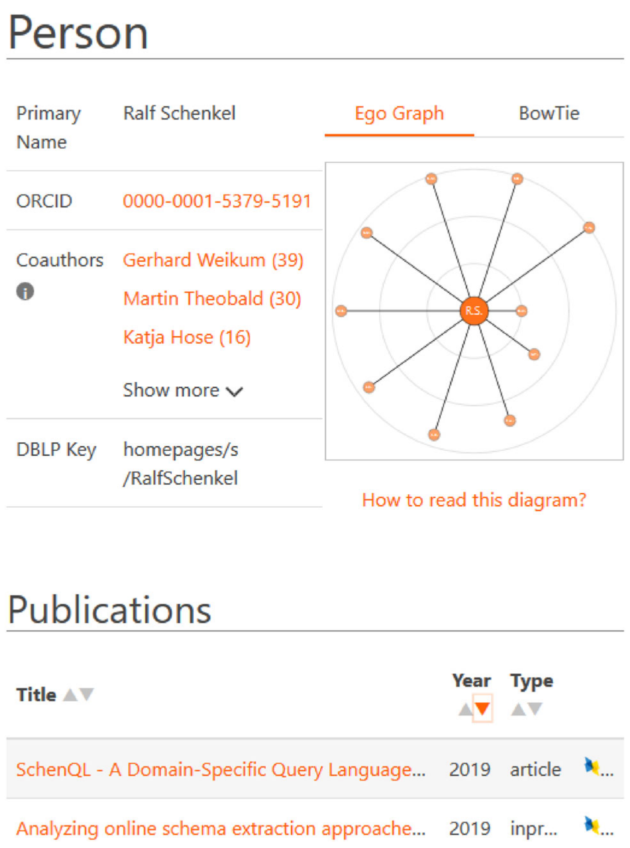
Our evaluation of the SchenQL system consists of a qualitative and a quantitative investigation which are followed by a performance evaluation. In a first qualitative study, we examine domain experts' use-cases and desired functionality of a DSL such as SchenQL as well as an accompanied GUI. The major goal of this first investigation was to check SchenQL for completeness and suitability for the addressed


PUBLICATIONS WRITTEN BY "Ralf Schenkel" IN YEAR 2018

Hint: Press tab for autocompletion

| Title | Year | Type | |
|---|---|---|---|
| ReCAP - Information Retrieval and Case-Based Reasoning for Robust Deli... | 2018 | inprocee... | ... |
| Prioritizing and Scheduling Conferences for Metadata Harvesting in dblp. | 2018 | article | ... |

**Fig. 5** SchenQL Front End for a search with suggested language components and search result

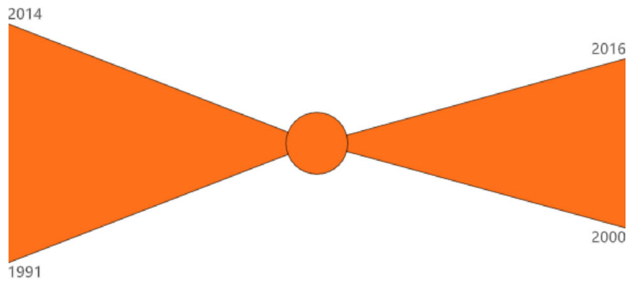

Person

| | |
|---|---|
| Primary Name | Ralf Schenkel |
| ORCID | 0000-0001-5379-5191 |
| Coauthors | Gerhard Weikum (39)<br>Martin Theobald (30)<br>Katja Hose (16)<br>Show more |
| DBLP Key | homepages/s/RalfSchenkel |

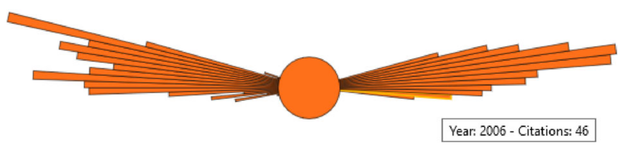
Publications

| Title | Year | Type | |
|---|---|---|---|
| SchenQL - A Domain-Specific Query Language... | 2019 | article | ... |
| Analyzing online schema extraction approache... | 2019 | inpr... | ... |

**Fig. 6** Person detail view with Ego Graph depicting up to the ten most common co-authors. Nodes symbolise persons; the further an author is from the middle (person in focus), the less publications they share with the person in focus



**(a)** Regular BowTie view.



**(b)** Detailed BowTie view.

**Fig. 7** Regular (top) and detailed (bottom) BowTie view with referenced and citing papers of a person with numbers of referenced (bows left of knot) and citing (bows right of knot) papers. For the regular view year numbers limit the period of time from which a paper referenced (left) and is cited itself (right). In the detailed view, the references and citations are separated in single slices per year. Hovering over single slices depicts the year and the associated number of references from or citations acquired in the specific year. The higher the number of citations or references, the longer the bow; the longer the spanned time, the higher the bow

use cases. In a subsequent step, we conducted a quantitative study in which we first compared SchenQL with SQL, both used through a command line interface (CLI) to ensure comparability. The goal was to measure the effectiveness, efficiency and users' satisfaction with SchenQL as query language. As a follow-up, we evaluated the web-based GUI of the SchenQL system using the same queries and compared the results with those received from usage of the SchenQL CLI. We additionally investigated the SchenQL system's user experience using the User Experience Questionnaire (UEQ) [34].

Considering the overall goals for SchenQL, we derived the following five hypotheses to be investigated:

$H_1$ MySQL as database engine with SQL as a target language for the SchenQL compiler is more suitable than Neo4j as database engine with Cypher as target engine.

$H_2$ The SchenQL-to-SQL compiler's performance in translating and executing queries is comparable to that of manually formulated queries.

$H_3$ Utilisation of the SchenQL CLI achieves better results in terms of higher correctness, lower perceived difficulty of query construction as well as lower required time for query formulation than usage of SQL.

$H_4$ SchenQL is as suitable for domain-experts as it is for non-experts.

$H_5$ The SchenQL system provides high suitability and user experience (indicated by values $> .8$ for all six quality dimensions assessed with the UEQ[10]) for users not familiar with structured queries.

For all studies, we used a data set from the area of computer science: our structures were filled with data from dblp [25] integrated with fitting data from Semantic Scholar (for citations and abstracts) and enriched with information about institutions from Wikidata.[11] As keys of persons, publications and venues, we utilised dblp keys of the respective entities. Utilising the dblp data set as of June 2020[12] leads to 2,518,198 entries for persons, 5,095,451 entries for publications, 1849 entries for journals, 91,694 entries for conferences and 10,059 entries for institutions.

## 4.1 Benchmarks: database engines and target language as well as SchenQL CLI Compiler evaluation

The technical evaluation of our system consists of two parts: a comparison of execution times of queries for two different target engine candidates with their respective query languages and a comparison of the target queries generated by the SchenQL compiler to manually optimised queries for typical query types found in digital libraries.

We first compare two specific implementations of viable target engine options: a relational database and a graph database. Both database types are reasonable options for the representation of bibliographic metadata. The actual data are clearly structured (e.g. in a publication record with clearly defined attributes) which supports usage of relational databases. The multiplicity of direct relations between bibliographic entities (e.g. persons citing papers instead of only persons writing papers and those papers citing other papers) and the graph-like structure (see Fig. 1) hints at the utilisation of graph databases.

We then evaluate the implementation of our SchenQL compiler with regard to its suitability.

### 4.1.1 Selection of database engines and query languages

For the selection of a relational as well as a graph-based database management system (DBMS), we defined important factors which a database needed to satisfy in order to narrow down the numerous options for our application:

- Open source license. We did not want to introduce legal restrictions or license fees.
- Availability of the source code. The technical implementation should be accessible to allow research and adjustments.
- Active further development of DBMS. Guaranteed operation of the DBMS in the future was important, this was accessed by the date of latest release.
- Possibility of querying the DBMS from Java and Python programs without a further surrounding system but through a query language directly. This property ensured a low structural adaptation effort for the SchenQL ecosystem in case the underlying database is changed.

For the selection of a relational database engine, we considered the wide-spread options MySQL and PostgreSQL. As we did not come across clear arguments for or against one or the other,[13] MySQL was chosen. It has the advantage of providing the MyISAM storage engine which has a higher support for full text search which we consider highly relevant. It does not support transactions, but transactions are not required in our use case. The target language for the relational database MySQL is SQL.

For the selection of a graph-based database engine[14] and query language, we additionally deemed the structural and syntactical similarities to SQL important. This ensured the best possible comparability between the query languages. Consideration of the four general properties and the last one specific to the graph-based variants produced the query language Cypher as the best option. Cypher is supported by the DBMS Neo4j, Redis and AgensGraph. As a related work [7] also utilises Cypher as target language and Neo4j as DBMS, we followed their example in our decision.

Note that execution times of queries are highly dependent on the utilised execution environment. We tried to select the best possible options for the relational as well as graph-based databases and respective target languages for our specific use case. We cannot exclude that other target languages for the database types may achieve better or different results.

[10] User Experience Questionnaire Handbook: https://www.ueq-online.org/Material/Handbook.pdf.

[11] https://www.wikidata.org/wiki/Wikidata:Main_Page.

[12] https://dblp.org/xml/release/dblp-2020-06-01.xml.gz.

[13] https://db-engines.com/en/system/MySQL%3BPostgreSQL.

[14] https://db-engines.com/en/ranking/graph+dbms.

**Table 2** Overview of queries evaluated in the benchmark

| ID | Query in plain text |
|---|---|
| $B_1$ | Titles of all publications |
| $B_2$ | Titles of publications written by $A$ |
| $B_3$ | Titles of publications written by $A$ which appeared in journal J |
| $B_4$ | Primary names of persons who authored a publication with title $P$ |
| $B_5$ | Titles of publications about $K$ |
| $B_6$ | Keywords of publications with title $P$ |
| $B_7$ | Primary names of co-authors of $A$ |
| $B_8$ | Primary names of co-authors of co-authors of $A$ |
| $B_9$ | Titles of publications which reference publications which were published by institutions where $A$ is member |
| $B_{10}$ | Titles of $A$'s most cited publications |
| $B_{11}$ | Number of $A$'s publications |
| $B_{12}$ | Primary names of persons with a name that sounds like $F$ |
| $B_{13}$ | Titles of publications written by $A$ or $B$ |
| $B_{14}$ | Titles of publications written by $A$ and not by $B$ |
| $B_{15}$ | Titles of publications containing $T$ |

$A$, $B$ are unique names of different authors, $J$ is a journal acronym, $P$ is a publication title, $K$ is a keyword, $F$ is a forename, and $T$ is a term

### 4.1.2 Queries

Table 2 shows the different benchmarking queries we observed. They utilise a representative amount of all SchenQL language elements. The queries were inspired by typical search scenarios [8] and categorisations [29] in digital libraries. Queries $B_1$ to $B_6$ incorporate only one or two concepts and simple conditions and combinations. $B_7$ observes the co-authorship relation, and $B_8$ introduces a second indirection layer to this query type. $B_9$ generates an especially large result set. Queries $B_{10}$ as well as $B_{11}$ evaluate more domain specific functions. Query $B_{12}$ utilises Soundex. In queries $B_{13}$ and $B_{14}$, logical operators are combined with single concepts. $B_{15}$ evaluates full text search. We filled the main variables of the queries randomly, and the dependent variable was set with respect to the main variable. For $B_3$, we randomly chose a person $A$ who has published at least one paper in a journal $J$; for $B_{14}$, we randomly chose a pair of co-authors $A$ and $B$ where $A$ has also published several papers without $B$.

### 4.1.3 Setting

We run the performance benchmarks on a Ubuntu 20.04 machine with 32 GB RAM and a 2 TB SSD. A MySQL 8.0.21 database handles the SQL queries, and an equivalent Neo4j 4.0.8 database handles the Cypher queries. We only set the variables for each benchmarking query once, and the same variables were used throughout all experiments. Each query was run 100 times, and we report average execution times on both databases. To minimise the impact of caching and prediction effects of modern hardware on the measurements, the DBMS and the surrounding docker containers were restarted after each query execution.

Figure 8 shows the simplified SchenQL graph-based data model without attributes. The colours of entities from this depiction correspond to the respective tables from the SchenQL relational data model in Fig. 2. Keywords, journals and conferences are directly linked with publications. Persons are linked with publications and institutions. Person names, for example, are no longer stored in a separate table contrasting the relational data model, but they now appear as attributes of persons.

## Benchmark part I: database engine and target language performance

In this first part of the technical evaluation, we assess the performance of two different database engines with specified languages for our SchenQL compiler for domain specific query types: the relation database engine MySQL with SQL and the graph-based database engine Neo4j with Cypher [13]. Here, we strive to investigate the suitability of SQL as a target language for the SchenQL compiler with MySQL as database engine compared to Cypher with Neo4j as database engine and thus verify or falsify hypothesis $H_1$. A target language for the SchenQL compiler has to support the formulation of typically required query types, and the execution time of queries in general should not interrupt a user's flow of thought (see Table 4). These two properties thus define our perception of suitability of a target engine and language. We deliberately do not include cost of learning or conciseness in our percep-

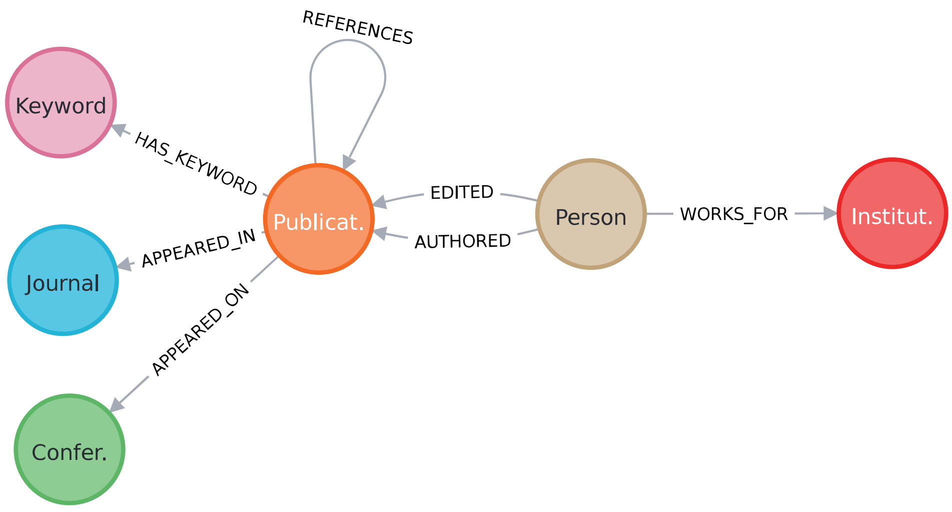


**Fig. 8** SchenQL graph-based data model (online in colour)

**Table 3** Evaluation query categories, queries and figures

| Exec. time | Queries | Figures |
|---|---|---|
| Low ($s * 10^{-3}$) | $B_1$, $B_2$, $B_4$, $B_5$, $B_6$, $B_9$, $B_{11}$, $B_{14}$ | 9 |
| Medium ($s * 10^{-2}$) | $B_7$, $B_{10}$, $B_{15}$ | 10 |
| High ($s$) | $B_3$, $B_{12}$, $B_{13}$ | 11, 12 |
| Very high ($s * 10^2$) | $B_8$ | 13, 14 |

**Table 4** Categories (C) of system response times in ms and associated user experience

| C | Time | User experience |
|---|---|---|
| 1 | $\leq 100$ | Users feel system reacts instantaneously |
| 2 | $\leq 1000$ | User's flow of thought is uninterrupted |
| 3 | $\leq 10000$ | Limit for user's attention span |

tion of suitability of a target language as users of SchenQL will not come in contact with the target language itself.

### 4.1.4 Analysis of $H_1$

All queries from Table 2 could be formulated both with SQL (run against a MySQL database) and Cypher (executed on a Neo4j database). Suitability in terms of both target languages being appropriate to express the information needs is therefore given, and no database engine and target language surpass the other in this aspect. So in the following, we focus on the assessment of execution times for queries.

The following shows an exemplary formulation of $B_{14}$ with Cypher:

```
MATCH (per:Person{primaryName:''A''})-[:
    ↪ AUTHORED]->(pub:Publication)
WHERE NOT
EXISTS((:Person{primaryName:''B''})-[:AUTHORED
    ↪ ]->(pub:Publication))
RETURN pub
```

The queries from Table 2 were put in four different categories depending on their execution time (see Table 3). The labels of the boxplots are defined as follows: *Cypher* (white boxes) describes manually written and optimised Cypher queries executed on a Neo4j database, *oSQL* (light grey boxes) marks manually constructed and optimised SQL queries, and *sSQL* (dark grey boxes) is SQL queries generated by the SchenQL compiler. SQL queries were run against the MySQL database. This part of the plots is utilised in the following evaluation in Sect. 4.1.5. Missing oSQL data points indicate that their value is identical to the respective sSQL data point. The whiskers extend to the $5^{th}$ and $95^{th}$ percentile [23]. The notches are defined as +/-1.58*IQR/sqrt(n) and represent the 95% confidence interval for each median [10]. To enable conclusions about the probability that two medians differ, we can compare the notch area of two queries in the visualisation, which is only possible with linear scale. If data points of two different queries have a large separation on the time scale, we provide a separate plot with corresponding scale factor (Figs. 13 and 14).

There seems to be a general execution difference of 0ms to 10ms in favour of SQL. The evaluation of the speed in relation to user experience is based on the absolute measured values according to the criteria of Nielsen [28] (see Table 4) constructed for response times of systems. All executions of Cypher formulations of queries except $B_{12}$ fall into category 1. $B_{12}$ takes 3517ms in the Cypher version and 584ms in the SQL version. Checking for a substring inside a property or a field seems to cause a complete search in all relevant nodes (Cypher) and a full table scan (SQL). The relational database

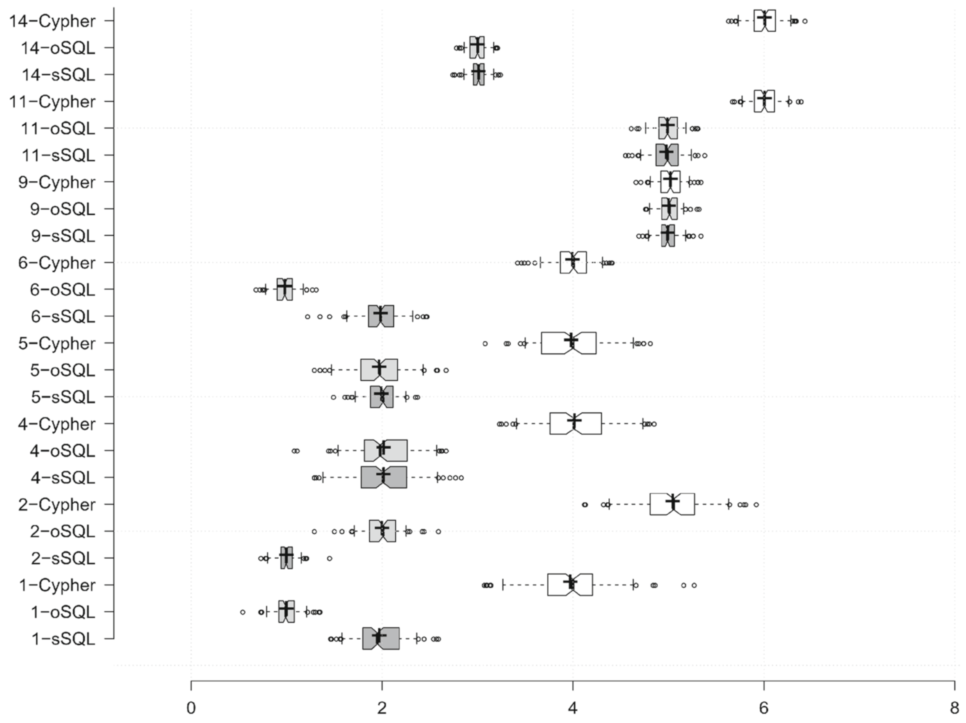


**Fig. 9** Queries with low execution time in ms

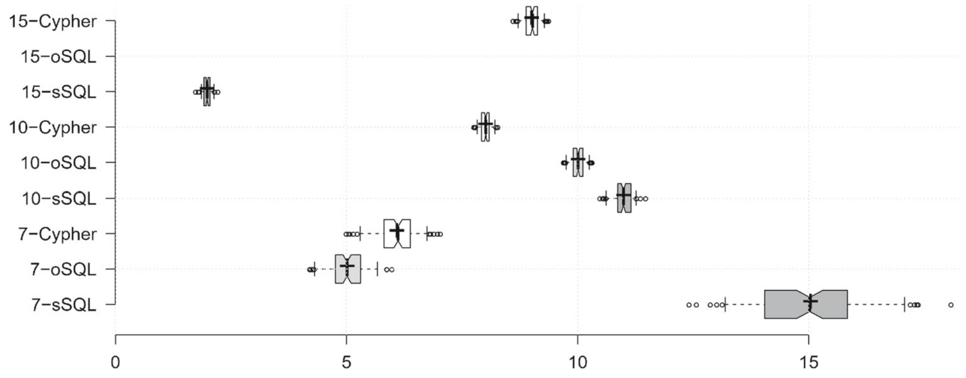


**Fig. 10** Queries with medium execution time in ms

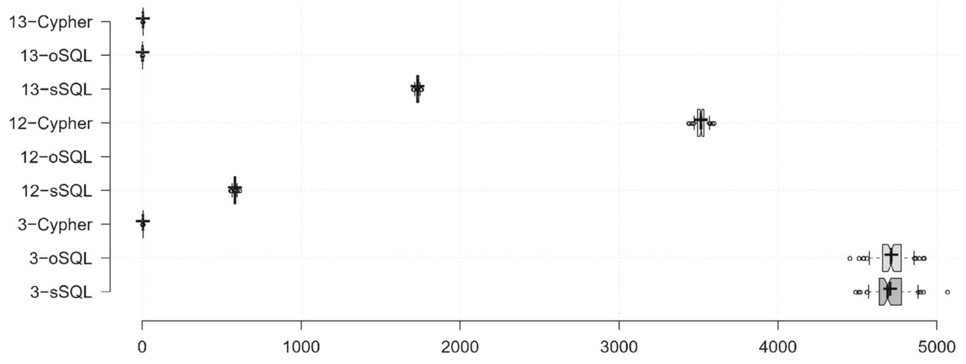


**Fig. 11** Queries with high execution time in ms, overview of all formulation times

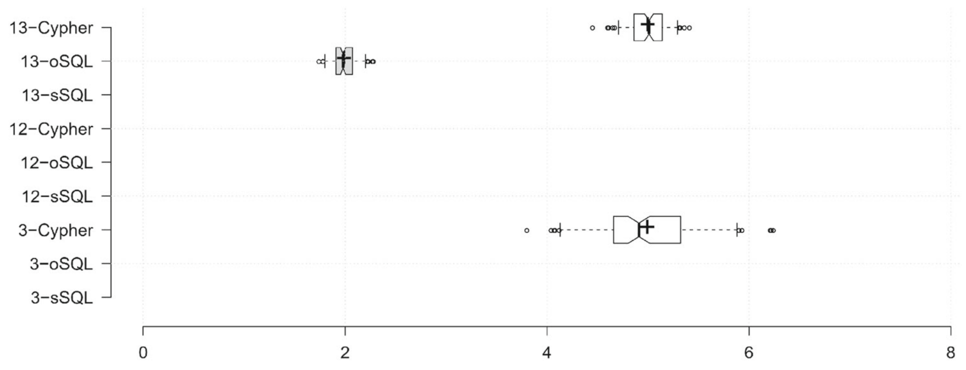


**Fig. 12** Queries with high execution time in ms, zoom on formulations with low execution time

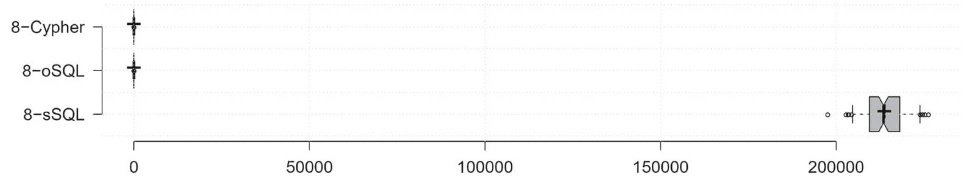


**Fig. 13** Queries with very high execution time in ms, overview of all formulation times

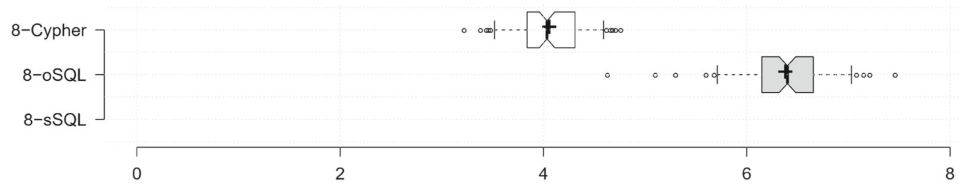


**Fig. 14** Queries with very high execution time in ms, zoom on formulations with low execution time

seems to have a more efficient way to execute this functionality compared to the graph database. The Cypher query falls in category 3, the SQL one in category 2. Execution of all SQL versions of queries except $B_3$ and $B_{12}$ falls into category 1. The SQL version of $B_3$, concatenation in combination with a second concept, lies in category 3, while the Cypher version falls in category 1.

### 4.1.5 Discussion

Both database engines and target languages are perfectly suitable to formulate prototypical queries with. With SQL executed on a MySQL database as well as Cypher run against a Neo4j database, we found one query type which does perform badly (category 3): Soundex for Cypher ($B_{12}$) and combination of different full-text searches for SQL ($B_3$). We argue that the Soundex functionality might be more important for users of digital libraries to be computed in a fast fashion as this is an integral part of all queries containing author names which are oftentimes hard to spell correctly. In general, the SQL queries on the relational database have lower execution time by a small margin than the Cypher ones on the graph-based DBMS. So regarding all problems we conclude that SQL run on a MySQL database is a more suitable target language for the SchenQL compiler compared to Cypher on a Neo4j database engine, thus validating hypothesis $H_1$.

## Benchmark Part II: SchenQL performance

In the second part of the benchmark, we want to assess the performance of our SchenQL to SQL compiler compared to queries directly formulated in SQL. Here, we hope to assess the optimisation degree of compiled SchenQL queries contrasting directly formulated SQL ones. We intend to assess hypothesis $H_2$.

### 4.1.6 Setting

In this part of the benchmark, we compare execution time of queries generated by our SchenQL to SQL compiler to ones in SQL we constructed already in the previous evaluation in Sect. 4.1.4. We again utilise the same queries (see Table 2) and evaluation environment described in Sect. 4.1.3. Table 5 contains SchenQL formulations of the queries.

### 4.1.7 Analysis of $H_2$

Here, we again refer to the figures mentioned in Table 3. We compare execution times for queries generated by the SchenQL compiler (labelled *sSQL*) to those of manually constructed and optimised SQL queries (labelled *oSQL*).

The query creation strategy of the SchenQL compiler relies on the DBMS optimiser to flatten subqueries and reorder joins. This approach works for most of the evalu-

**Table 5** Overview of SchenQL queries derived from Table 2 evaluated in the benchmark

| ID | Query in SchenQL |
|---|---|
| $B_1$ | PUBLICATIONS |
| $B_2$ | PUBLICATIONS WRITTEN BY "$A$" |
| $B_3$ | PUBLICATIONS WRITTEN BY "$A$" AND APPEARED IN (JOURNAL NAMED "$J$") |
| $B_4$ | PERSON AUTHORED (PUBLICATION TITLE "$P$") |
| $B_5$ | PUBLICATIONS ABOUT KEYWORD "$K$" |
| $B_6$ | KEYWORDS OF (PUBLICATION TITLED "$P$") |
| $B_7$ | COAUTHORS OF "$A$" |
| $B_8$ | COAUTHORS OF (COAUTHORS OF "$A$") |
| $B_9$ | PUBLICATION REFERENCES (PUBLICATION PUBLISHED BY (INSTITUTION MEMBERS "$A$")) |
| $B_{10}$ | MOST CITED (PUBLICATIONS WRITTEN BY "$A$") |
| $B_{11}$ | COUNT (PUBLICATIONS WRITTEN BY "$A$") |
| $B_{12}$ | PERSON NAMED ~ "$F$" |
| $B_{13}$ | PUBLICATIONS WRITTEN BY "$A$" OR WRITTEN BY "$B$" |
| $B_{14}$ | PUBLICATIONS WRITTEN BY "$A$" AND NOT WRITTEN BY "$B$" |
| $B_{15}$ | PUBLICATIONS ABOUT "$T$" |

$A$, $B$ are unique names of different authors, $J$ is a journal acronym, $P$ is a publication title, $K$ is a keyword, $F$ is a forename, and $T$ is a term

```
SELECT DISTINCT title, year FROM
    ↪ publication p join
person_authored_publication ON
    ↪ publicationKey = p.key  JOIN
    ↪ person
pe ON pe.key = personKey WHERE pe.key
    ↪ IN ( SELECT DISTINCT key FROM
person JOIN person_names ON personKey =
    ↪ key WHERE name = ''A'') OR
key IN ( SELECT DISTINCT key FROM
    ↪ person JOIN person_names ON
personKey = key WHERE name = ''B'')
    ↪ LIMIT 100;
```

**Listing 1** SchenQL compiler-generated SQL for $B_{13}$.

ated queries ($B_1$, $B_2$, $B_4$, $B_5$, $B_6$, $B_7$, $B_8$, $B_{10}$, $B_{11}$, $B_{14}$, $B_{15}$) as their execution times fall in category 1 (see Table 3). The query execution times for $B_3$ and $B_{12}$ lie in the same categories (category 3 and category 2) for compiler-generated and manually optimised SQL formulations. $B_8$ and $B_{13}$ are exceptions: the manually formulated queries both belong to category 1, whereas the compiled versions lie in category 3, thus outperforming the queries generated by the SchenQL compiler by orders of magnitude. $B_{13}$ utilises logical disjunction and joins the publication table, which contains about $5 \times 10^6$ records, with the person table ($2.5 \times 10^6$ records). There could be a problem with the MySQL optimiser, and the compiler-generated query seems simple to optimise (see Listing 1). In $B_8$, the two layers of indirection are causing the SchenQL-to-SQL compiler to generate a cascade of subqueries. It seems that the optimiser is unable to increase performance here so that the execution time difference between the optimised query and the generated version is about factor $10^4$.

### 4.1.8 Discussion

In general, it was evident that in most cases the optimiser of the MySQL RDBMS enhances the compiler-generated SQL queries to the level of the manually optimised version. Only in the case of two conditions in a query and the search for co-authors of co-authors of a person, the SchenQL SQL code requires orders of magnitude more execution time than the manually written and optimised queries. From these observations, we derive a general high performance of the SchenQL-to-SQL compiler which is comparable to manually formulated queries and thus validate hypothesis $H_2$.

## 4.2 Qualitative study: interviews

To get a comprehensive picture of SchenQL's completeness and suitability, we conducted semi-structured one-on-one interviews with four employees of the dblp team to discover realistic use-cases as well as desirable functionalities and potential extensions. Leading questions were which queries they would like to answer with the data and which functions or visualisations they envisioned in a GUI. The participants do work daily on digital libraries and are thus considered highly experienced in the area. They were only aware of the domain of interest and the underlying data set but did not know anything about SchenQL.

The interviews showed that the dblp staff wished to formulate queries to compute keywords of other publications that were published in the same journal as a given publication, the determination of the most productive or cited authors, as well as the most cited authors with few co-authors. Furthermore, a GUI should support numerous visualisations: colour-coded topics of publications or co-author-groups were explicitly asked for. Another participant requested intermateable components for the visualisation of graphs to display co-publications, co-institutions or connections between different venues. Other desired functionalities were a fault-tolerant person name search and sophisticated ranking methods.

As expected, the experts' suggestions were quite specific and strongly shaped by their daily work with dblp, which may not fit classic non-expert use of digital libraries. SchenQL is able to formulate several of the desired questions; however, it needs to be evaluated by non-power-users as we have done in the quantitative evaluation described below to ensure usability for casual users as well. The experts' comments on visualisation drove the design of the GUI's visual analysis components.

### 4.3 Quantitative study: SchenQL CLI vs. SQL, GUI and UEQ

Our quantitative study consists of two parts: first, the SchenQL CLI is compared to SQL, then the usability of the GUI and thus the SchenQL system as a whole is assessed. For the first part, it is not feasible to compare a specialised system such as the SchenQL CLI to a commercial search engine, as differences between the compared systems should be minor [17]. Additionally, as stated above, search interfaces in this domain [15,25] do not provide as many functionalities as SchenQL. We also refrained from evaluating the CLI against other DSLs such as Cypher [13] as test users would have been required to learn two new query languages. Comparing our CLI against SPARQL would have required the definition of classes, properties and labels for the data set and was therefore also disregarded in favour of the comparison against SQL.

Users participated voluntarily in the study, they were aware of being able to quit any time without negative consequences. They actively agreed on their data being collected anonymously and their screens being captured. We assume gender does not influence the measured values, so it is not seen as additional factor in the evaluation. Kelly [17] advises to examine quasi-independent variables such as sex of test users if researchers believed they influenced the outcome variable. We assume domain-experts are versed in the vocabulary and connections between bibliographic objects, and non-experts might have their first encounter with bibliographic metadata.

**Table 6** Templates of all queries used in the qualitative evaluations

| | |
|---|---|
| $Q_1$ | What are the titles of publications written by author $A$? |
| $Q_2$ | What are the names of authors which published at conference $C$? |
| $Q_3$ | What are the titles of the publications referenced by author $A$ in year $Y$? |
| $Q_4$ | What are the titles of the five most cited publications written by author $A$? |

$A$ are unique names of different authors, $C$ is the acronym of a conference and $Y$ is a year

For our significance tests, we used an independent two-sample t test in case data are normally distributed (checked with Shapiro–Wilk test) and if variances are homogeneous (checked with Levene's test). Otherwise and if we do not specify differently we applied Mann–Whitney $U$ tests. We consider a $p$-value of .05 as significance level. We use Fisher's exact tests to check whether the frequency distributions of categorical (nominal or ordinal) variables differ from the expected distributions in cases where the expected value is less than five. If we encounter nominal and scaled variables, we utilise the ETA coefficient to calculate correlations. We measure correlation between ordinal values or ordinal and scaled values with Kendall's $\tau_B$. Correlation between ordinal and nominal values is estimated with likelihood ratio (LR), and effect size is given with Cramér's $V$.

#### 4.3.1 Queries

In both parts of the study, we asked the participants to find answers to the queries given in Table 6 using either SchenQL CLI/SQL (part I) or the GUI (part II). The used queries are inspired by everyday search tasks of users of digital libraries. Common information needs are, for example, lookup of titles of specific publications or identification of persons working in a specific area [8]. Such information needs can be classified as simple information search as well as exploratory search tasks [30]. We formulated four different types of queries targeting core concepts found in the domain. Variables were switched between query languages to prevent learning effects based on query results. $Q_1$, $Q_3$ and $Q_4$ are publication searches, while $Q_2$ targets person search. $Q_1$ and $Q_2$ can be answered by using dblp [25] alone. Except for $Q_3$, Semantic Scholar could technically be used to find answers for the queries. The following formulation of $Q_3$ in SQL intends to show the complexity of those queries:

**Table 7** SchenQL and SQL formulations of queries utilised in our evaluations

| Query | SchenQL | SQL |
|---|---|---|
| $Q_1$ | `PUBLICATIONS WRITTEN BY "A"` | `SELECT title FROM publication p JOIN person_authored_publication pap ON p.key=pap.publicationKey NATURAL JOIN person_names WHERE person_names.name = "A"` |
| $Q_2$ | `AUTHORS PUBLISHED IN "C"` | `SELECT primaryName FROM person p JOIN person_authored_publication pap ON p.key=pap.personKey NATURAL JOIN publication WHERE conference_key = "C"` |
| $Q_4$ | `MOST CITED (PUBLICATIONS WRITTEN BY "A")` | `SELECT title, COUNT (*) FROM publication p JOIN person_authored_publication pap ON p.key = pap.publicationKey NATURAL JOIN person_names JOIN publication_references pr ON p.key = pr.pub2_id WHERE name = "A" GROUP BY title ORDER BY COUNT (*) DESC LIMIT 5` |

```
SELECT DISTINCT title FROM publication
    ↪ p, publication_references r
WHERE p.key = r.pub2_id AND r.pub_id IN
    ↪ ( SELECT publicationKey FROM
person_authored_publication pap NATURAL
    ↪ JOIN person_names JOIN
publication p2 ON p2.key = pap.
    ↪ publicationKey WHERE
person_names.name = ''A'' AND year = Y)
    ↪ ;
```

In SchenQL, the query could be formulated as follows (for all queries, see Table 7):

```
PUBLICATIONS CITED BY (PUBLICATIONS
    ↪ WRITTEN BY ''A'' IN YEAR Y);
```

We refrained from evaluating more complex queries to keep the construction time for SQL queries feasible.

## User study Part I: SchenQL CLI vs. SQL

With this first part of the quantitative study, we assess the usability, suitability as well as user satisfaction of usage of the SchenQL CLI compared to SQL for queries typically answered with an information retrieval system operating on bibliographic metadata. Additionally, the need for a DSL in the domain of bibliographic metadata is analysed as we try to verify or falsify hypotheses $H_3$ and $H_4$. Participants of this evaluation needed to be familiar with SQL.

### 4.3.2 Setting

We defined the evaluation process of our archetypical interactive information retrieval study [17] as follows: every user performed the evaluation alone in the presence of a passive investigator on a computer with two monitors. The screens were captured in order to measure times used to formulate the queries. All participants formulated all queries in SQL and SchenQL. A query language was assigned with which a user was going to start the evaluation, and it was switched between users to compensate for learning effects. Users were permitted to use the Internet at any stage of the evaluation. A SchenQL cheat sheet, the ER diagram and examples for the database schema provided to test subjects can be found in Kreutz et al. [20].

At first, a video tutorial[15] for the introduction and usage of SQL and the SchenQL CLI was shown; afterwards, subjects were permitted to formulate queries using the system they were starting to work with. Following this optional step, users were asked to answer a first online questionnaire to assess their current and highest level of SQL knowledge (both on a scale from *1 (no knowledge)* to *6 (very good knowledge)*), the number of times they used SQL in the past three months (*0 times*, *1-5 times*, *more than 5 times* and *daily*) and their familiarity with the domain of bibliographic metadata.

[15] SchenQL Evaluation CLI vs. SQL - Tutorial: https://youtu.be/g7J64wzbE5I.

**Table 8** Correctness (CORR) in percent, assessed average difficulty (DIFF) and average time in minutes for the four queries for SQL and the SchenQL CLI

| | SQL | | | SchenQL CLI | | |
|---|---|---|---|---|---|---|
| | CORR | DIFF | TIME | CORR | DIFF | TIME |
| $Q_1$ | 90.48 | 2.86 | 4:57 | 90.48 | 1.57 | 2:57 |
| $Q_2$ | 90.48 | 3. | 4:35 | 100. | 2.1 | 3:11 |
| $Q_3$ | 23.81 | 4.86 | 8:55 | 47.62 | 2.71 | 3:33 |
| $Q_4$ | 23.81 | 5.91 | 10:36 | 95.24 | 1.71 | 1:53 |

Participants were asked to submit the queries in SQL and SchenQL, respectively. The queries were always formulated in the following order: $Q_1$, $Q_2$, $Q_3$, $Q_4$. We consciously ordered the queries such that more complex SQL queries followed the easier/shorter ones to help users in query formulation. This part of the first quantitative evaluation was concluded with a second online questionnaire regarding the overall impression of SchenQL, the rating of SchenQL and SQL for the formulation of queries as well as several open questions targeting possible advantages and improvements of SchenQL. We evaluated 21 participants from the area of computer science with SQL knowledge. In total, ten subjects started by using SQL, and eleven participants began the evaluation using SchenQL.

### 4.3.3 Analysis of $H_3$

To assess the validity of hypothesis $H_3$ of SchenQL leading to better results than using SQL, we observe the *number of correctly formulated queries*, the *rated difficulty* and the *required time* for the formulation of queries with the SchenQL CLI and SQL. For each of these values, we first conducted significance tests on all four queries together, and here, the two languages SchenQL and SQL were regarded as groups; afterwards, we performed significance tests on each of the four queries. Table 8 gives an overview of correctness, average rated difficulty and average time for formulating all four queries for both languages. Difficulty was rated on a scale from *1 (very easy)* to *7 (very difficult)* to allow neutral ratings.

*Correctness* 57.14% of queries were correctly formulated using SQL, whereas 83.33% of queries were correctly formulated using the SchenQL CLI. This result clearly shows the significantly (U=2604, p=0) superior effectiveness of SchenQL compared to SQL in terms of overall correctness. While $Q_1$ and $Q_2$ were answered correctly by most participants, the number of correctly formulated queries for $Q_3$ and $Q_4$ highly depends on the system. $Q_4$ was correctly answered by a quarter of the subjects using SQL, while more than 95% of users were able to formulate the query in SchenQL, and this difference is significant (U=63, p=0). These observations support the partial verification of $H_3$ in terms of higher number of correctly formulated queries with the SchenQL CLI compared to SQL.

*Rated difficulty* The mean rating of difficulty of the formulation of queries with SQL was 4.16 ($\sigma$ = 1.94); with SchenQL, the mean rating was significantly lower (2.02, $\sigma$ = 1.11; U=1341, p=0). On average, the query construction using SQL is rated more difficult for every query. The averaged highest rated difficulty for a query in SchenQL is still lower than the averaged lowest rated difficulty of a query in SQL. We found significantly lower ratings of difficulties of queries for all four queries ($Q_1$: U=114, p=.005; $Q_2$: U=143.5, p=.044; $Q_3$ (t test): t=-5.539, p=0; $Q_4$: U=0, p=0) when using SchenQL compared to utilisation of SQL. These observations support the partial verification of $H_3$ in terms lower perceived difficulty in query formulation with the SchenQL CLI compared to SQL.

*Time* Average construction of queries in SQL took 7:15 minutes ($\sigma$ = 4:47 min.); with the SchenQL CLI, the construction was significantly quicker and took 2:52 minutes ($\sigma$ = 1:51 min.; U=1165.5, p=0) on average. This documents the efficiency of SchenQL. We found significantly lower required times for query formulation for all four queries ($Q_1$ (t test): t=-3.433, p=.001; $Q_2$: U=141.5, p=.047; $Q_3$: U=62, p=0; $Q_4$: U=7, p=0) when using SchenQL compared to the utilisation of SQL. These observations support the partial verification of $H_3$ in terms of lower required time for query formulation with the SchenQL CLI compared to SQL.

*General results* The queries $Q_3$ and $Q_4$ in SQL are assumed to be complex which is supported by the low percentage of correct formulations using SQL. They are also much longer than the respective SchenQL ones. That means the time required to write them down is higher and there is more opportunity to make mistakes which causes a query reformulation [33]. The overall rating of suitability of SchenQL for constructing the queries resulted in an average of 6.43 ($\sigma$ = .6), while the rating was significantly (U=7, p=0) lower (3.14, $\sigma$ = 1.2) for SQL on a scale from *1 (very bad)* to *7 (very good)*. While SQL was rated below mediocre, SchenQL was evaluated as excellent which shows users' satisfaction with it. These results lead to the conclusion of SchenQL being highly suitable for solving the given tasks which represent everyday queries of users of digital libraries and a high user acceptance of SchenQL.

In summary, utilisation of SchenQL achieves higher correctness of queries, lower perceived difficulty and requires less time than using SQL, which together verifies hypothesis $H_3$.

### 4.3.4 Analysis of $H_4$

To assess validity of hypothesis $H_4$ of SchenQL being as suitable for experts as it is for non-experts, we conduct tests of independence for correctness and rated difficulty and correlation tests for required time for query formulation. We run tests on all queries separately and on the SchenQL system as a whole. Our dependent variable is knowledge in the area of bibliographic metadata. The 21 participants from before form the two user groups: nine participants are non-experts, and twelve participants are familiar with bibliographic metadata.

*Correctness* In general, 75% of queries were correctly formulated by domain-experts, whereas the non-experts achieved only 63.89% in both QLs. Participants which were (non-)experts were able to solve 65.58% (47.22%) of queries in SQL and 85.42% (80.56%) in SchenQL. Tian et al. [36] stated that for a domain-expert, it would be easier to write queries in a DSL than in SQL. We found that the observed frequencies for correct and incorrect formulation of queries per group do not significantly deviate from the expected frequencies (separated by query and in general for all SchenQL queries; Fisher's exact tests if there were both correct and incorrect results for queries). We did not find enough evidence to suggest that domain knowledge and correctness of formulated queries are associated.

*Rated difficulty* We did not find enough evidence to suggest that domain knowledge and rated difficulty of query formulation are associated (separated by query and in general for all SchenQL queries; Fisher's exact tests).

*Time* We found no strong correlation between the two groups of domain knowledge and required time for query formulation (separated by query and in general for all SchenQL queries; ETA coefficient).

*Result* No user group is consistently better than the other, and we found no deviations from expected frequencies for correctness and rated difficulty. We also did not find strong correlations between required time for query formulation and domain knowledge. The SchenQL CLI seems to be as suitable for domain-experts as it is for non-experts; thus, $H_4$ is verified.

#### 4.3.5 Open questions and discussion

In the open questions, the short, easy and intuitive SchenQL queries were complimented by many participants. Users noted the comprehensible syntax was suitable for non-computer scientists as it resembles natural language. Some noted their initial confusion due to the syntax and their incomprehension of usage of literals or limitations. Others asked for auto-completion, syntax highlighting, a documentation and more functions such as a most cited with variable return values. No participant wished for visualisations which could be caused by design fixation [16] or generally lower requirements for such a system compared to the experts from the qualitative study.

The average overall impression of the SchenQL QL was rated by the subjects as 5.05 ($\sigma$ = .74) on a scale from *1 (very bad)* to *6 (very good)*, enforcing a non-neutral rating. Assessed difficulty and required times to formulate the four queries were significantly lower when utilising SchenQL compared to SQL, and the overall correctness of all queries was significantly higher for SchenQL as well. This verified hypothesis $H_3$ of the CLI leading to generally better results than SQL. Our hypothesis $H_4$ of the SchenQL system being as suitable for domain-experts as it is for casual users is also verified. No user group was found to be consistently better than the other one, and we did not find significant deviations from expected frequencies. We also did not find strong correlations between required times for query formulation and knowledge of bibliographic metadata.

We performed correlation tests on the collected data of participants regarding their current and highest level of SQL knowledge as well as the number of times they used SQL in the three months preceding the evaluation. The participants' current skill in SQL highly correlates with their overall rating of our QL ($\tau_B$ = .53). Being versed in advanced SQL could lead to a higher appreciation of complexity hidden from users in SchenQL. The quantity in which the participants were using SQL in the last three months correlates with their rating of difficulty of $Q_1$ ($\tau_B$ = .54) and $Q_2$ ($\tau_B$ = .42) in SQL and $Q_4$ with the CLI ($\tau_B$ = .46). Having recently used SQL could lead to higher familiarity with it and therefore perceived easier construction of queries if they are not too complex. The number of times SQL was used in the last three months correlates with the correctness of $Q_3$ (LR, $V$ = .67) and $Q_4$ (LR, $V$ = .87) in SQL. Having used SQL recently seems to help persons formulate difficult queries more successfully.

This evaluation leads to the construction of the prototypical GUI with its syntax suggestion as well as auto-completion features. Additionally, although they were not mentioned by participants in this evaluation, some visualisations were included following suggestions from the qualitative evaluation.

## User study Part II: SchenQL GUI vs. CLI and User Experience Questionnaire

This second part of the quantitative study focused on evaluating the GUI and, thus, the SchenQL system as a whole. We assessed how usage of the web interface compared to users' impressions and performance when utilising the SchenQL CLI. Besides a part where test users answered queries with the GUI, we conducted the User Experience Questionnaire [34] to measure user experience with the SchenQL system. To resemble our target audience, we did not pose the precondition of users being familiar with SQL or the formulation of structured queries. Here, we intend to assess the hypothesis $H_5$.

**Table 9** Correctness (CORR) in percent, assessed average difficulty (DIFF) and average time in minutes for the four queries for the GUI

| | SchenQL GUI | | |
|---|---|---|---|
| | CORR | DIFF | TIME |
| $Q_1$ | 90 | 1.3 | 1:05 |
| $Q_2$ | 90 | 2.2 | 1:41 |
| $Q_3$ | 40 | 3.6 | 2:56 |
| $Q_4$ | 90 | 2.4 | 2:18 |

#### 4.3.6 Setting

This evaluation is performed analogous to the previous part: every user performed the evaluation alone but in the presence of a passive investigator on a computer with two monitors. We measured times used to find answers by capturing screens. The same SchenQL cheat sheet as in the first part was provided to the test subjects. At first, a video tutorial[16] introduced the usage of the SchenQL GUI. The next part was the formulation or the navigation towards solutions of the four queries introduced in Table 6 using the GUI. Afterwards, the subjects completed the User Experience Questionnaire [34] followed by questions regarding the overall impression of the GUI as well as possible improvements.

We evaluated ten participants from the area of computer science and adjacent fields, which did not yet take part in a previous evaluation of the SchenQL system.

#### 4.3.7 Partial analysis of $H_5$: users unfamiliar with query formulation

To assess partial validity of hypothesis $H_5$ in terms of the GUI's suitability for users unfamiliar with query formulation, we conduct significance tests on all queries together and each separate query for correctness, rated difficulty and required time. We observe the results from usage of the SchenQL CLI from the previous evaluation and participants' results from utilisation of the GUI as the two groups. Table 9 gives an overview of the correctness, average rated difficulty and average required time for all four queries when using the SchenQL system.

*Correctness* Except for $Q_3$, participants mostly solved the queries correctly, resulting in an overall correctness of 77.5% (-10.83% compared to CLI, difference not significant). We found no significant differences between the two groups for correctness in any of the four queries.

*Rated difficulty* Users rated the perceived difficulty of queries as 2.38 (+.35 compared to CLI, difference not significant) on average. We found no significant differences between the two groups for rated difficulty in any of the four queries.

[16] SchenQL Evaluation GUI - Tutorial: https://youtu.be/56-23zyUDPQ.

**Table 10** Description of the six dimensions measured by the UEQ to capture users' impressions of interactive products [34]

| Dimension | Description |
|---|---|
| Attractiveness | Overall impression, users' approval or disapproval |
| Perspicuity | Easiness to familiarise with product and to learn usage |
| Efficiency | Effort required to solve tasks, reaction times of product |
| Dependability | Security and predictability of product, level of control of users' interaction |
| Stimulation | Excitement, motivation, fun |
| Novelty | Creativeness and innovation of product, sparks user's interest |

*Time* Users took about 2:15 minutes for the retrieval of a solution (-0:37 minutes compared to CLI, difference is significant; U=1207, p=.011) on average. We found significant differences in times required to solve queries $Q_1$ and $Q_2$. Times required for formulating the queries with the GUI were significantly ($Q_1$: U=33.5, p=.002; $Q_2$: U=41, p=.006) lower than those resulting from using the CLI. As these queries were relatively simple, we assume the auto-completion and suggestion-feature of the GUI is especially helpful in the fast construction of straightforward queries or the GUI offering other suitable ways of quickly obtaining simple bibliographic information. Usage of the GUI might be more intuitive compared to writing simple queries in the SchenQL CLI.

*General results* We want to point out that participants from the first part of the quantitative study who were familiar with query formulation, but were not offered help in the construction, did not significantly differ in rating of difficulty and correctness from users of this user study. In case of the GUI, the subjects were supported in the formulation of queries but were not necessarily familiar with this kind of task. Hence, we assume the system's suggestion and auto-completion feature is useful for redemption of unequal prior knowledge in this case.

Correctness and rating of difficulty did not differ significantly between usage of CLI and GUI, but users were significantly faster in finding answers for simple queries with the GUI which underlines the suitability of the interface for everyday usage. Participants from this study resemble SchenQL's target audience, which additionally emphasises its usefulness and partly verifies hypothesis $H_5$ in terms of the GUI being suitable for users not versed with structured query formulation.

#### 4.3.8 Partial analysis of $H_5$: UEQ

The *attractiveness*, *perspicuity*, *efficiency*, *dependability*, *stimulation* and *novelty* of interactive products can be measured with the User Experience Questionnaire (UEQ) [34] even at small sample sizes. Table 10 describes the aspects the UEQ measures. Here, we want to conclude the assessment of the validity of hypothesis $H_5$ in terms of rating of user experience.

Participants of this study answered the 26 questions of the UEQ regarding usage of the SchenQL system. Ratings on pairs of contrasting stances (-3 to 3) such as *complicated-easy* or *boring-exciting* were then grouped to the six dimensions mentioned before. Values above .8 are generally considered as positively evaluated equalling high user experience, and values above 2 are rarely encountered.

In general, users seem to enjoy using the SchenQL system (attractiveness = 2.07, $\sigma$ = .25). The handling of our system is extremely easy learned (perspicuity = 2.3, $\sigma$ = .19). Tasks can be solved without unnecessary effort (efficiency = 2.03, $\sigma$ = .49) and users feel in control of the system (dependability = 1.83, $\sigma$ = .63). They seem exited to use the SchenQL system (stimulation = 1.73, $\sigma$ = .33) and rate the system as innovative and interesting (novelty = 1.58, $\sigma$ = .68).

As all six quality dimensions achieved ratings well over .8, the system is positively evaluated which equals high user experience and partially verifies $H_5$.

### 4.3.9 Open questions and discussion

In the open questions, participants praised the intuitive usability, the auto-completion and the suggestion feature. For future development, suggestions for literals were requested and two participants wished for a voice input. Remarkably, not a single user mentioned the need for more or other visualisations, and this is possibly attributed to design fixation [16] but might also stem from the advanced needs of power users from the expert interviews.

The users were significantly faster in solving simple queries when using the GUI compared to the CLI. As we found no significant impairments from utilisation of the GUI, we assume its usefulness and usability for query formulation. Participants from this study were far less familiar with the construction of structured queries compared to those of the previous study but seemed to be adequately supported by the GUI in the retrieval of information. Together with the UEQ which showed users' high ratings ($>$ .8) for all six quality dimensions (which proves high user experience), hypothesis $H_5$ could be partially verified.

## 5 Conclusion and future work

We evaluated the SchenQL system, a domain-specific query language operating on bibliographic metadata from the area of computer science with accompanying GUI supporting query formulation. Our thorough evaluation against SQL showed the need for such a DSL. Test subjects' satisfaction with the SchenQL system was assessed with application of the UEQ. The introduction of a GUI and its evaluation with users resembling our target audience did not significantly change the correctness of answers or the users' rating of difficulty of the queries compared to the CLI, but instead the time needed to formulate simple queries was reduced significantly. Missing prior knowledge with structured query formulation seems to be compensated by using a GUI with a suggestions and auto-completion feature. As the CLI and the GUI proved to be viable tools for information retrieval on bibliographic metadata, users' preferences should decide which one to use.

The target language SQL run on a MySQL database engine for the SchenQL compiler was a more suitable choice than Cypher run on a Neo4j database engine ($H_1$), and the performance of the generated queries is as high as manually formulated ones ($H_2$). Using SchenQL leads to generally better results compared to the utilisation of SQL ($H_3$). The system seems to be as suitable for domain-experts as it is for non-experts ($H_4$). Our GUI has high usability for users not familiar with structured query formulation ($H_5$).

Future efforts could focus on the identification of query types which would better be run on a graph database and then decide which query will be translated in SQL and which one will be translated to Cypher. Enhancements of functionalities could include more visualisations such as colour-coded topics or graph visualisation as the experts from the qualitative study requested. Furthermore, more specific query options such as a filter for papers with few co-authors or most cited with variable return values could be included. As visualisations were not relevant for users in our quantitative evaluation, future efforts could focus on supporting more advanced query options: algorithms for social network analysis as PageRank, computation of mutual neighbours, hubs and authorities or connected components [35] would fit. Centrality of authors, the length of a shortest path between two authors and the introduction of aliases for finding co-citations [12] would also be useful query building blocks. As user-defined functions [35] were well received in other work [33], they are a further prospect. Incorporation of social relevance in the search and result representation process as shown in [2] could also be an extension. User profiles could store papers and keywords, which in terms influence results of search and exploration.

**Funding** Open Access funding enabled and organized by Projekt DEAL.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

## References

1. Amaral, V., Helmer, S., Moerkotte, G.: A visual query language for HEP analysis. In: IEEE NSS 2003. vol. 2, pp. 829–833. IEEE Computer Society (2003)
2. Amer-Yahia, S., Lakshmanan, L.V.S., Yu, C.: SocialScope: enabling information discovery on social content sites. In: CIDR 2009. www.cidrdb.org (2009)
3. Baeza-Yates, R., Ribeiro-Neto, B.A.: Modern Information Retrieval - The Concepts and Technology Behind Search, 2nd edn. Pearson Education Ltd., Harlow (2011)
4. Bates, M.: Task Force Recommendation 2.3 Research and Design Review: Improving User Access to Library Catalog and Portal Information: Final Report (version 3) (2003)
5. Beall, J.: The weaknesses of full-text searching. J. Acad. Librariansh. **34**(5), 438–444 (2008)
6. Berget, G., Sandnes, F.E.: Why textual search interfaces fail: a study of cognitive skills needed to construct successful queries. Inf. Res. **24**(1) (2019)
7. Betts, C., Power, J., Ammar, W.: GrapAL: Connecting the dots in scientific literature. In: ACL 2019. pp. 147–152. ACL (2019)
8. Bloehdorn, S., Cimiano, P., Duke, A., Haase, P., Heizmann, J., Thurlow, I., Völker, J.: Ontology-based question answering for digital libraries. In: ECDL 2007. vol. 4675, pp. 14–25. Springer (2007)
9. Borodin, A., Kiselev, Y., Mirvoda, S., Porshnev, S.: On design of domain-specific query language for the metallurgical industry. In: BDAS 2015. vol. 521, pp. 505–515. Springer (2015)
10. Chambers, J., Cleveland, W., Kleiner, B., Tukey, P.: Graphical methods for data analysis (vol 17, pg 180, 1983). J. Sleep Res. **21**, 484–484 (08 2012)
11. Collberg, C.S.: A fuzzy visual query language for a domain-specific web search engine. In: Diagrams 2002. vol. 2317, pp. 176–190. Springer (2002)
12. Dries, A., Nijssen, S., Raedt, L.D.: BiQL: A query language for analyzing information networks. In: Bisociative Knowledge Discovery 2012, vol. 7250, pp. 147–165. Springer (2012)
13. Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P., Taylor, A.: Cypher: An evolving query language for property graphs. In: SIGMOD 2018. pp. 1433–1445. ACM (2018)
14. Guidi, F., Schena, I.: A query language for a metadata framework about mathematical resources. In: MKM 2003. vol. 2594, pp. 105–118. Springer (2003)
15. Hotho, A., Jäschke, R., Benz, D., Grahl, M., Krause, B., Schmitz, C., Stumme, G.: Social bookmarking am beispiel bibsonomy. In: Social Semantic Web 2009, pp. 363–391. Springer (2009)
16. Jansson, D.G., Smith, S.M.: Design fixation. Des. Stud. **12**(1), 3–11 (1991)
17. Kelly, D.: Methods for evaluating interactive information retrieval systems with users. Found. Trends Inf. Ret. **3**(1–2), 1–224 (2009)
18. Khazaei, T., Hoeber, O.: Supporting academic search tasks through citation visualization and exploration. Int. J. Digital Librar. **18**(1), 59–72 (2017)
19. Klink, S., Ley, M., Rabbidge, E., Reuther, P., Walter, B., Weber, A.: Browsing and visualizing digital bibliographic data. In: VisSym 2004. pp. 237–242. Eurographics Association (2004)
20. Kreutz, C.K., Wolz, M., Schenkel, R.: SchenQL - A domain-specific query language on bibliographic metadata. CoRR **abs/1906.06132** (2019)
21. Kreutz, C.K., Wolz, M., Schenkel, R.: SchenQL: A concept of a domain-specific query language on bibliographic metadata. In: ICADL 2019. vol. 11853, pp. 239–246. Springer (2019)
22. Kreutz, C.K., Wolz, M., Weyers, B., Schenkel, R.: Schenql: Evaluation of a query language for bibliographic metadata. In: ICADL 2020, vol. 12504, pp. 323–339. Springer (2020)
23. Krzywinski, M., Altman, N.: Visualizing samples with box plots. Nat Methods **11**, 119–20 (02 2014)
24. Leser, U.: A query language for biological networks. In: ECCB/JBI 2005. p. 39 (2005)
25. Ley, M.: DBLP - some lessons learned. PVLDB **2**(2), 1493–1500 (2009)
26. Madaan, A.: Domain specific multi-stage query language for medical document repositories. PVLDB **6**(12), 1410–1415 (2013)
27. Martín, M.S., Gutiérrez, C., Wood, P.T.: SNQL: A social networks query and transformation language. In: AMW 2011. vol. 749. CEUR-WS.org (2011)
28. Nielsen, J.: Usability Engineering. Academic Press, Cambridge (1993)
29. Numminen, P., Vakkari, P.: Question types in public libraries' digital reference service in finland: Comparing 1999 and 2006. J. Assoc. Inf. Sci. Technol. **60**(6), 1249–1257 (2009)
30. Pirolli, P.: Powers of 10: modeling complex information-seeking systems at multiple scales. IEEE Comput. **42**(3), 33–40 (2009)
31. Reitz, F.: A framework for an ego-centered and time-aware visualization of relations in arbitrary data repositories. CoRR **abs/1009.5183** (2010)
32. Rohil, M.K., Rohil, R.K., Rohil, D., Runthala, A.: Natural language interfaces to domain specific knowledge bases: an illustration for querying elements of the periodic table. In: ICCI*CC 2018. pp. 517–523. IEEE Computer Society (2018)
33. Schaefer, A., Jordan, M., Klas, C., Fuhr, N.: Active support for query formulation in virtual digital libraries: a case study with DAFFODIL. In: ECDL 2005 (2005)
34. Schrepp, M., Hinderks, A., Thomaschewski, J.: Applying the user experience questionnaire (UEQ) in different evaluation scenarios. In: HCI 2014. vol. 8517, pp. 383–392. Springer (2014)
35. Seo, J., Guo, S., Lam, M.S.: SociaLite: an efficient graph query language based on datalog. IEEE Trans. Knowl. Data Eng. **27**(7), 1824–1837 (2015)
36. Tian, H., Sunderraman, R., Calin-Jageman, R.J., Yang, H., Zhu, Y., Katz, P.S.: NeuroQL: A domain-specific query language for neuroscience data. In: EDBT Workshops 2006. vol. 4254, pp. 613–624. Springer (2006)
37. Xu, B., Cai, R., Zhang, Z., Yang, X., Hao, Z., Li, Z., Liang, Z.: NADAQ: natural language database querying based on deep learning. IEEE Access **7**, 35012–35017 (2019)

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.