Nested Concept Graphs: Applications for Databases

Frithjof Dau and Joachim Hereth Correia

Darmstadt University of Technology, Department of Mathematics,
Schlossgartenstr. 7, D–64289 Darmstadt, Germany,
{dau,hereth}@mathematik.tu-darmstadt.de

Abstract. The processing of information has increased to an astonishing level in the recent decades, database systems have grown more and more powerful. However, the power to use the stored information is not easily accessible. Often the interface is very restrictive, using predefined forms, or the database logic has to be accessed by entering queries in the database query language, most often SQL.
Given the proximity of the logic of relational databases and formal logic, a graphical logic system as they are being developed in the context of Conceptual Graphs, should be both easy and powerful enough to support users in their interaction with databases. The application of the results of [Dau03] to this task lead to the Nested Concept Graphs who are presented in an informal fashion in this paper. We show that they have indeed the expressivity to represent database queries, and argue that they might be suited as a practical query interface to relational databases.

1 Introduction

The need to support human users understand nowadays information systems is apparent: interaction with computers becomes more and more important in our daily lifes. The key to this empowerment of the user is an easy to understand interface to the underlying logic. The very goal of Conceptual Graphs has always been to provide a graphic representation for logic which is able to support human reasoning. Many of the possible applications of such an logical representation systems have been described in [Sow84, Sow92, Sow00], but one of the maybe most exciting ones has not been completely worked out until today: a consistent, graphical interface for database interaction.

In [Dau03] one of the authors studied in depth a calculus for the mathematization of Conceptual Graphs and their equivalence to first order logic. Starting from this equivalence and the known equivalence of the relational calculus and the relational algebra from database theory (cf. [Her02]), we compared the expressiveness of Concept Graph with negations with the standard database query language SQL (the *Structured Query Language*). This approach resulted in an extension to Concept Graphs in form of the Nested Concept Graphs with Cuts (cf. [DH03]).

The basic idea of the application presented in this paper is to provide a universal database query language which is more accessible to users than SQL. In this sense, it may be considered similar to the visual query languages available in many of the commercial database management systems. However, looking more closely, those interfaces provide only a graphical representation of a subset of SQL, while letting the more complicated features still to be entered in a textual way. We believe that the consistent presentation of queries in form of the Nested Concept Graphs leads to a more intuitive interface.

The general idea of connecting Conceptual Graphs to databases has already been used in [EGSW00, GE01]. In our approach the user has more liberties in defining the query graph, we imagine him using a graph editor to define his queries. The query graphs are supposed to be universally applicable, while some of the approaches in the cited articles recur on specialized graphical representations. Additionally, we consider negation and nesting, which are important components in a database query language.

To show the expressive power of Nested Conceptual Graphs as query graphs, we will translate the crucial parts of database queries as defined by the SQL standard into query graphs. A short introduction to SQL, which is the most common query language for relational databases, will be given in the following section. Then, in Sections 3 to 5, we will give a mapping from SQL to query graphs by example. Section 6 concludes this paper.

2 The SQL Standard

The idea of relational databases has been introduced into the scientific community by E. F. Codd from the IBM San Jose Research Center in [Cod70]. Major industrial research to implement was underway and lead to the advent of the Ingres, Oracle and System/R (later DB2) databases in the late 1970s. In 1986 and 1987 the American National Standards Institute and the International Standards Organization formed committees for the creation of a common standard for a relational query language. This effort resulted in the norm ANSI/ISO/IEC 9075 "Information Technology – Database Language SQL" in 1989. An improvement and extensive expansion (SQL2) was then published in 1992, providing different levels of conformance. The SQL validation service offered by the National American Standards Institute ended in 1996, such that today there is no official certificate of conformance. While a new standard (SQL3) is available since 1999, the older one may be considered more important, as most commercial products sill struggle to implement the more advanced features of SQL2. Most relational databases today provide at least the Entry Level conformance of SQL2 plus some proprietary extensions, but to our knowledge, no product even claims SQL3 conformance. For this reason, SQL2 can be considered to be the current de-facto standard.

The irony in the history of SQL lies in the fact, that SQL has originally been developed to support humans to interact with databases. The language should resemble normal human language, but be more structured (hence the

name "Structured Query Language"), such that a formal interpretation of it would be possible. However, there were many restrictions to be considered when writing SQL statements. Consequently, the language got a reputation of being rather complex and difficult to learn. Some of the reasons for the difficulties encountered by people learning SQL will be highlighted in the following.

SQL covers the whole range of database administration, from database schema management over data updates to the area of data queries. We will not elaborate here an equivalence between the standard of about 700 pages with Nested Concept Graphs. We believe that this is mostly feasible if we allow graphs to be interpreted dependent on the administration task at hand. The same graph may then be used to add, delete or query the existence of an element. Due to space restrictions we will focus on the query specification.

For our examples, we will use a small database called ConKids[1], as shown in Figure 1 (as this is only an example the database is by no means complete!).

Table Person:

Lastname	Firstname	Age
Cole	Richard	32
Dau	Frithjof	35
Hereth	Joachim	27
Klinger	Julia	26
Malik	Grit	30
Rudolph	Sebastian	26

Table ResearchArea:

Lastname	Topic
Cole	CG
Dau	CG
Dau	EG
Hereth	CG
Hereth	DBMS
Klinger	CG
Malik	CG
Rudolph	CG

Fig. 1. The database ConKids

3 Translating Simple Queries

Being a formal standard, all syntactical and conceptual elements of SQL are precisely defined and explained. Rules and conditions are given for the correctness of statements and the results of the formulated statement described. As we wrote in the introduction, we think that we could follow directly the SQL syntax and map it piece for piece into graphical representations. However, to make this article more pleasant to read than the SQL standard and due to space restrictions, we will skip those parts that are conceptually similar to those we already explained. Also, we will consider queries only for now. But let us start at the beginning: The query specification in the SQL standard is given as:

```
<query specification> ::=
   SELECT [ <set quantifier> ] <select list> <table expression>
```

[1] The *ConKids* are a small group of young researchers in Conceptual Knowledge Processing, who joined to help each other to work out new ideas and concepts.

A very simple realization of this pattern might be the following statement:

```
SELECT Lastname FROM Person WHERE Firstname='Grit'
```

which means that the user wants to get the Lastnames from the table Person, where the Firstname is 'Grit'. So far, the SQL query is readable for non-experts too. Our translation into Concept Graphs looks like this:

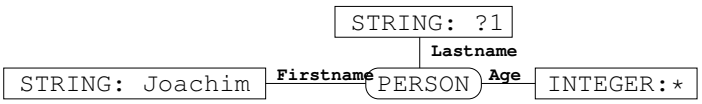

Fig. 2. A simple query as Conceptual Graph

This graph is as easy to understand to everyone used to reading Conceptual Graphs, and is in fact only a slightly modified version of the query graph example from [Sow92].

At first sight, it might be understood as a simple Concept Graph as defined in [Pre98] or [Dau03]. According to the formal definition there, we have to make explicit the set of signs we use for our concept graphs. The concept names are the data types, and the relation names the relations that are available in the given database. The object names are all possible values, i. e. all strings, integers, and so on that might appear in the database. Besides those, also the generic marker $*$ may be used as an object name or the query markers $?i$. Those special markers "$*$" and "$?i$" are not considered to be strings, but special entities. Object names and the generic marker have the usual meaning in the query graphs, either to fix the concept instance to one special object, or to represent an existing, but not precisely defined object respectively. For the task of evaluating a query graph (in the sense of [Dau03]), the query marks have to be valuated with arbitrary objects, but two nodes with the same query marker in the same context have to be mapped to the same object. For a more formal description of the query graphs and their evaluation, we refer the reader to [DH03]. The evaluation uses the data from the underlying in the natural way, the formal definition of the corresponding power context family works along the lines of the canonical transformation in [Her02].[2] To enhance the readability of the graphs, we make a little extension in their display. Usually, the arcs of a graph are numbered, but here the arcs are named. This directly relates to the question of named vs. unnamed perspective in database theory. For the theory, the version without (column) names is easier to handle, but for real applications, names

[2] In [Her02] the construction of $\mathbb{K}_0$ is not specified. For our purpose, the attributes of $\mathbb{K}_0$ are the available data types, the incidence relation is naturally defined. As only a common domain and no types were assumed in the former work, this definition was not possible. Additionally, to be able to make evaluations for graphs as those in Section 5, we model besides the given also the possible relations.

greatly support the user in identifying the meaning of the presented data. This is exactly the approach we take: For formal definitions as in [DH03] we use the unnamed perspective, but for our examples we use the named version. The equivalence for these perspectives has been shown for various models of database theory in [AHV95] and can be adopted.

The following examples make the `WHERE` clause a bit more difficult, instead of equality, we use a comparison with a constant value:

```
SELECT Lastname FROM Person WHERE Age <= 30
```

At first, this is very similar to the first example. However, as natural as the condition looks in SQL, it can not be expressed in the relational calculus as introduced by Codd. A basic version of a query graph for this example can be seen in Figure 3.

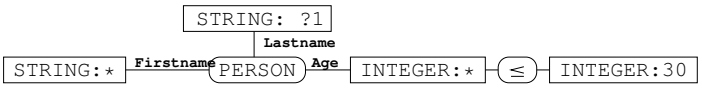

Fig. 3. A query graph with comparison.

To make the graphs more readable, we use for all comparison relations ($<$, $\leq$, $>$, $\geq$, $=$) a shortcut notation as can be seen in Figure 4. Instead of connecting two concept boxes of the same type, one of them with an asterisk, the other with a specific value in the referent field, we use one concept box of the same type and write the comparison symbol followed by the specific value in the referent field. Comparing these two graphs with the first one, you can see that the translation of the first query is not unique. As well we could have written it using the $=$-Relation. However, it is obvious that the query graphs in Figure 2 on the preceding page and those built analogously to Figure 3 and Figure 4 are all equivalent.

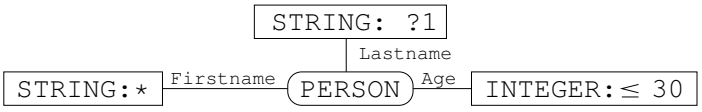

Fig. 4. A query graph with the graphical shortcut for the comparison operator

Now, we want to analyze the complications that more complex SQL queries bring and how to represent them with query graphs. One of these complications are what is called in the SQL standard a `<derived column>`. This means, that in the `<select list>` not simply a column is chosen from the data table, but an operation on one or multiple columns is performed. This operation may involve

addition and multiplication of values, concatenation of strings, and many more. For our example we represent a query with the concatenation operation. While not being in the Entry Level specification of the SQL2 standard, it is in various forms available in most databases. We use the syntax from the standard for Intermediate SQL. The query selects Firstname and Lastname for each Person, but instead of keeping them in two columns, this query joins the names to one string. The resulting relation has only one column as becomes obvious by the single query mark in the graph translation.

```
SELECT Firstname||Lastname FROM Person
```

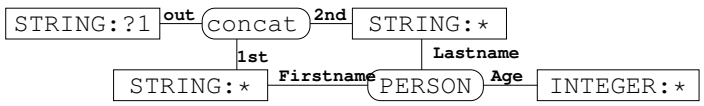

Fig. 5. The query graph with a "derived column"

So far, we have only seen queries on one table. However, the `<table expression>` part of the query specification allows much more complicated constructions. In a database are usually multiple data tables and often the information the user is looking for is distributed over more than one table. Therefore, queries refering to multiple tables are very common, e. g. the following query gives us the research topics of the sub-30 ConKids:

```
SELECT Topic FROM Person, ResearchArea
  WHERE Person.Lastname=ResearchArea.Lastname  AND Age <= 30
```

This query builds the cross product of the two tables `Person` and `ResearchArea` and selects the entries in the column `Topic`, provided that the Lastnames in both tables are the same and the Age is below 30. The condition of equal values in certain columns is a standard condition for joining columns. For this reason, the SQL standard provides the JOIN operation, with which we may reformulate the SQL query as follows:

```
SELECT Topic FROM Person JOIN ResearchArea
  USING (Lastname) WHERE Age <= 30
```

These two versions represent different approaches you could take if you want to get the answer by constructing intermediate tables by hand. However, most modern database systems will optimize the queries anyway, so the actual process in the system will probably be the same for both versions. Interestingly, the corresponding query graph looks also almost the same for both versions, as shown in Figure 6 on the facing page.

Fig. 6. A query graph joining two relations

The syntactical difference between the two formulations of the SQL query is not reflected in the graph. Of course, we can interpret the construction of the two SQL queries in graphical terms. The cross product of two relations means simply setting the two relations side by side. The equality condition in the `<where clause>` then additionally joins the two concept boxes on the Lastname arcs with the equality relation. The graph shown in Figure 6 then is a simple equivalent transformation, joining these two nodes together. Interpreting the join of two relations is even simpler; when joining relations you either have naturally to join the two boxes that belong to arcs with the same name (which has to be unique according to the SQL standard), or you have to denote explicitly over which arcs the relations should be joined (as it is done in our example). Then, you have the resulting joined relation. Of course, the result is equivalent, and which graphical representation is preferable depends on the taste of the user.

4 Negation in Queries

The examples we presented in the previous section were simple in the sense, that to evaluate them we need only positive knowledge about the data. But often arise questions not about what is, but about what is not. Visualizing queries with negation is rarely seen in commercial tools, probably because the visualization of the negation seems to be difficult. However, Charles Sanders Peirce found a way to do exactly this (cf. [Pei92, PS00]). He used ovals to mark the part of the graph which should be negated and developed a calculus for this graphical logic system. In [Dau03], the notion of *cuts* (as Peirce called these ovals) has been adopted for the system of Concept Graphs. Continuing this idea for our query graphs, we have thus a mechanism to make negation available as a graphical primitive.

Our next example selects those people, that are not focused on researching CGs, but also have other research areas (Of course, our example database is not complete in this respect and suffers from the usual defect of database systems. The queries have to use the closed world assumption – and this assumption is very often false):

```
SELECT Person.Lastname,Firstname FROM Person,ResearchArea
  WHERE Person.Lastname=ResearchArea.Lastname AND Topic <> 'CG'
```

A related question is, what ConKids are not interested in CGs? Semantically, there is only a small difference, in one case the question is if there are other research areas, in the second if there are only other research areas. Hence, it

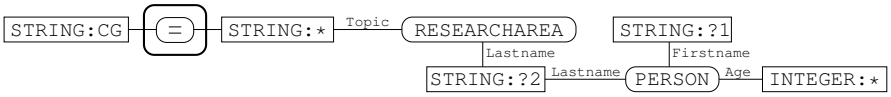

Fig. 7. A query graph with cut (negation)

is intuitive to expect a similar graph. In the graphical version, this means the same structure, but a different part of the graph is negated, as we can see in Figures. 7 and 8. However, in SQL the second query is rather different:

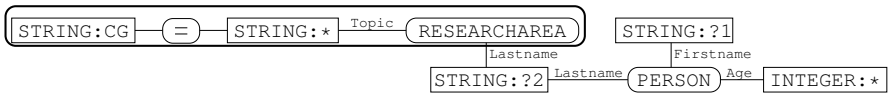

Fig. 8. Similar graph with different meaning

```
SELECT Lastname,Firstname FROM Person WHERE Lastname NOT IN
  ( SELECT Lastname FROM ResearchArea
      WHERE Topic='CG')
```

We can easily see that this query has the same result as the query graph in 8. However, the graph is *not* the canonical translation of this query (this will be discussed at the end of the next section). Graphs are parallel in nature, therefore we can write both queries ("Who works on CGs?" and "What are the peoples names?") side by side and make the necessary (negated) connection. In SQL we could write a simple query if we had not the negation in the question. Because of the negation, we get a completely different structure and have to declare a subquery. This is part of the reason why SQL looks so unapproachable for beginners; to solve related problems you have to write very different SQL statements, and sometimes even learn new syntactical elements.

As elaborated in [Dau03], the tools we have presented so far, provide the expressiveness of first order logic. More specifically this means, that we have the usual set operations like intersection, union, and complement available. The meaning of the latter naturally depends on the method used for the evaluation. This topic is treated in the discussion on safety of queries in [AHV95], and the approaches discussed there translate directly to approaches we can take for our database application. An important point is to assume that the universe contains only the objects known to the database (the closed-world-assumption), the so-called *domain* of the database. In [Dau03], this happens naturally by using the object set of the underlying power context family as domain. Of course, as the basic graphical primitives are conjunction and negation, disjunction is more complex in its visualization. Consider the query

```
SELECT Lastname FROM ResearchArea
  WHERE (Topic='CG') OR (Topic='EG')
```

The `OR` can not be directly translated but has to be transformed into a construction using three cuts, as can be seen in Figure 9. While it is difficult to explain the precise transformation, the graphical pattern is so strong that the introduction of a syntactical shortcut – as proposed in [Sow84, Sow92, Sow00] for similar problems – may be avoided.

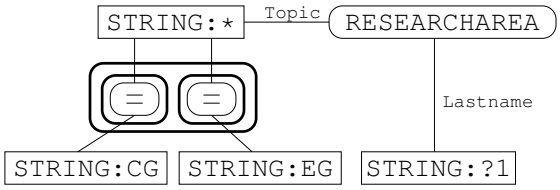

Fig. 9. A query graph with disjunction

5 Subqueries and Nested Graphs

While the power of the graphical elements presented so far helps to formulate many queries, we need more expressivity in practice. Not reasoning with, but reasoning about relations is a topic, that has not been addressed by Codd in [Cod70]. However, queries aiming at summarizing or *aggregating* sets are very important in daily (database) life. Therefore, the relational calculus had to be adapted on the theoretical side (cf.[Klu88]). More important for our considerations is the SQL standard, where we find these under the term *set function* as part of the `<select list>`. The standard provides five such set functions. They are `AVG` (average), `MAX`, `MIN`, `SUM`, and `COUNT`. All of them basically do one thing: They take a set as input and return a simple element as output. Of course, this breaks the idea we followed so far, that a function is a (special) relation between elements, because relations are not elements like strings or integers, but something connecting those elements.

In his book 'A Peircean Reduction Thesis' [Bur91], Robert Burch provides an algebraization of Peirces logic system and analyzes Peirces thesis, that any relation may be constructed out of ternary relations. For his solutions, he has to abstract from the relations and use them as objects to other relations, he calls those abstractions *hypostatic abstractions*. While a theory involving this concept may look confusing at first, the concept itself is very natural. In reasoning, humans sometimes consider the elements of a set and their properties, and sometimes the set itself as an element having certain other properties. This shift in perspective transforms a relation into an object which then may be attached

to relations again. In visualization, we use nested concept boxes as known from [Sow84, Sow92, Sow00].

To exemplify hypostatic abstractions, we consider the question about the average age of the ConKids. In SQL, we need for this the set function `AVG` and write the following:

```
SELECT AVG(Age) FROM Person
```

The difference to the previous queries is obvious: Before, we were interested in what is in the relation `Person`, now we are talking about a summarizing property of the relation itself. While the information we were looking for was contained in the tuples of the relation before, no single tuple can provide the answer now, only all of them together. Using the idea of hypostatic abstraction, we get the query graph depicted in Figure 10. Intuitively, by drawing the box around the description of a relation (in the example, of the plain relation `Person`), we transform it into an object, where we may attach relations like `AVG_2`. Of course, the two query marks ?1 that appear in the picture are not related, i. e. the query marks are only valid in their respective contexts (see [DH03] for details). In our example, we use the set function (`AVG_2`) with its canonical meaning, which is that a real number and a relation are in this (set) relation, if the average of the second column equals the relation. Modeling the set functions this way, we follow the presentation of the relational calculus with aggregate functions in [AHV95, Chap. 20].

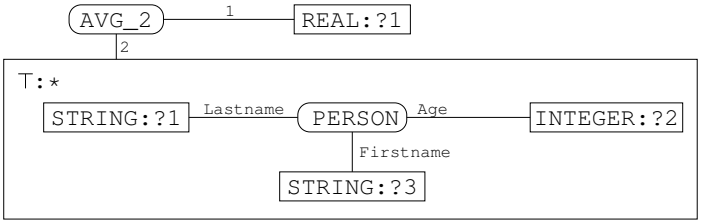

Fig. 10. A query graph with *hypostatic abstraction*

From the user point of view, an extension from this query to the question of average age per research topic seems rather simple. Using the graphical metaphor, it is: simply add the `ResearchArea` relation and add a query mark for the Topic string (cf. Figure 11 on the facing page). While this may look simple at first glance, it has also an effect on the relation defined by the hypostatic abstraction – after all, we draw something in the box defining the relation! Connecting a part of it to a node outside results either in a selection (if the node refers to a constant) or in a grouping, where the tuples that have the connecting property in common are grouped together.

In SQL we find the same idea, but we have to use a new syntactical construction, the `GROUP BY`. Reading the standard, you find them writing about the

result of a GROUP BY being a set of groups. But because relations as elements of the resulting relation are not possible, it has to be assured, that the result becomes flat again by adding set functions. As we know from our own experience, this too confuses beginners. The full SQL statement for the last question looks like this:

```
SELECT AVG(Age),Topic FROM Person, ResearchArea
  WHERE Person.Lastname=ResearchArea.Lastname
    GROUP BY Topic
```

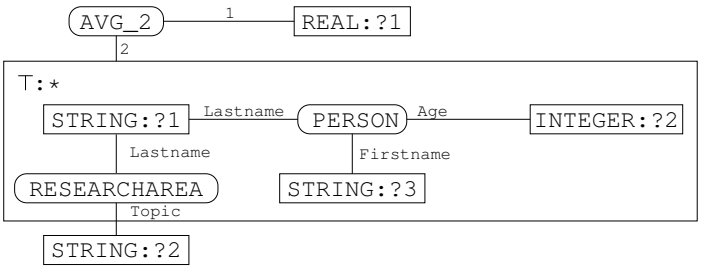

Fig. 11. A query graph extending the previous query

The last entry in the specification of the <table expression> in the SQL standard is the <having clause>. Again, this has to do with the fact, that SQL has to take different syntactical elements depending if they want to talk about elements or groups. Roughly said, the <having clause> is the <where clause> for groups. As our groups appear in the form of objects in the graphical representation, the translation of the <having clause> does not introduce new graphical elements. Let us consider the GROUP BY query from the last example. Now, we want to put an additional restriction on the groups, the average age should be below 30, but we are not interested in the exact average age. So we ask, for which research topics is the average age below 30? In the graphical notation, the graph looks almost exactly as the former one, with a small change in the obvious position (Figure 12 on the next page). In SQL, we use the keyword HAVING and get:

```
SELECT Topic FROM Person,ResearchArea
  WHERE Topic.Lastname=ResearchArea.Lastname
  GROUP BY Topic
    HAVING AVG(Age) < 30
```

After having discussed most of the crucial points of the query specification of the SQL standard, there is one point left which we have not yet mentioned: the <set quantifier> part of the specification. The two possible values are ALL (the default) and DISTINCT. Basically they switch between bag and set

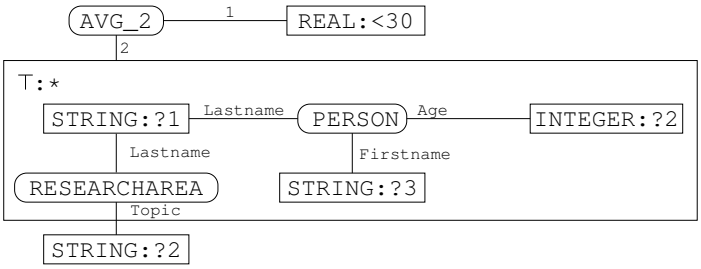

Fig. 12. A query graph corresponding to a SQL query with `HAVING`

semantics. For instance a, `SELECT COUNT(`x `Topic) FROM ResearchArea` gives 8 if x =`ALL` (duplicates are counted multiple times), and only 3 for x =`DISTINCT` (duplicates are eliminated). As Concept Graphs are based on a mathematical model, the evaluation usually is based on the set semantics. To accommodate the case of bag semantics (being the default for practical reasons), we can choose the bag semantics instead. The ramifications of such a choice have not yet been studied, but we do not expect any problems.

At the end of the discussion how to represent SQL queries in a graphical form, we return to the query of the previous section, where we presented the graph in Figure 8 on page 8 which was equivalent but not the canonical translation of the given SQL query. After having introduced the hypostatic abstraction, we can now also represent the more direct representation as graph, as is shown in Figure 13.

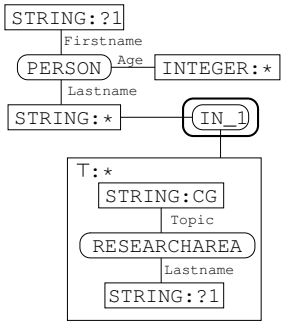

Fig. 13. A query graph for a SQL subquery

Subqueries are indeed a delicate part of SQL. Some common databases (e. g. MySQL, on which we tested all example queries except the subquery one) do not provide this functionality. The documentation proposes to use temporary tables, which is conceptually the same while being more effort to the user. For our concerns, we have to investigate the `IN_`i relation shown in the picture. It

means, that the given element is an element of the ith column of the connected relation. It is easily comprehensible that those relations are not essential for the diagrammatic representation of queries. To be or not to be in the set of elements having some property is of course equivalent to having this property or not. This is the reason why we did not need the concept of hypostatical abstraction to represent the query. However, for didactical reasons it might be preferable to have these relations available in a real system.

6 Conclusion

In the previous sections, we have shown how the relevant parts of the query specification of the SQL standard may be translated into graphical representations using Nested Concept Graphs. Of course, we could not describe in the available space how the whole specification maps to equivalent graphs, but we believe that we have covered the essential parts and that the reader can easily construct the mapping for the remaining parts analogously.

A prototypical system using query graphs as interface would be the next logical step. Such a system is also necessary to test if user interaction is verifiably improved by using a complete graphical interface. However, the graphical system presented here for database queries is based on the graphical logic systems developed by Peirce and Sowa. Based on the accounts, that these are easier to understand for non-logicians than the more common linear logic systems, the query graphs are supposedly easier to handle than SQL. The fact, that we do need less primitives to get the same expressiveness supports this idea. On the other hand, when compared to existing graphical database interfaces, the query graphs are more expressive due to the graphical representation of negation by cuts and the hypostatic abstractions. The learning effort of two additional graphical primitives is thus rewarded by a consistent representation of all queries, as has been shown in the previous sections. Of course, the opposite direction – translating graphs into database queries – is fundamental for such a prototypical implementation. The problems we encountered with subqueries, or more general with the varying capabilities of existing databases, indicates that solutions might be platform dependent.

The application of Nested Concept Graphs for databases is therefore a promising application of the idea of Conceptual Graphs. Additionally, the mathematization has made much progress in course of this research, as can be seen in [DH03].

References

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, Reading, MA, 1995.

[Bur91] Robert W. Burch. *A Peircean Reduction Thesis: The Foundation of Topological Logic*. Texas Tech. University Press, Lubbock, 1991.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[Dau03] Frithjof Dau. *The Logic System of Concept Graphs with Negations and its Relationship to Predicate Logic*. PhD thesis, Darmstadt University of Technology, 2003. (to appear in Lecture Notes in Computer Science, Springer).

[DH03] Frithjof Dau and Joachim Hereth Correia. Nested concept graphs: Mathematical foundations. Available at: http://www.mathematik.tu-darmstadt.de/~dau/DauHereth03b.pdf, 2003.

[EGSW00] Peter Eklund, Bernd Groh, Gerd Stumme, and Rudolf Wille. A contextual-logic extension of toscana. In Bernhard Ganter and Guy W. Mineau, editors, *Conceptual Structures: Logical, Linguistic and Computational Issues. 8th International Conference on Conceptual Structures, ICCS 2000*, number 1867 in LNAI, pages 453–467, Darmstadt, Germany, 2000. Springer, Berlin – Heidelberg – New York.

[GE01] Bernd Groh and Peter Eklund. A cg-query engine based on relational power context families. In Harry S. Delugach and Gerd Stumme, editors, *Conceptual Structures: Broadening the Base*, number 2120 in LNAI, Stanford, USA, July 2001. Springer, Berlin – Heidelberg – New York.

[Her02] Joachim Hereth. Relational scaling and databases. In Uta Priss, Dan Corbett, and Galia Angelova, editors, *Conceptual Structures: Integration and Interfaces. 10th International Conference on Conceptual Structures, ICCS 2002*, number 2393 in LNAI, pages 62–76, Borovets, Bulgaria, July, 15–19, 2002. Springer, Berlin – Heidelberg – New York.

[Klu88] Anthony Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the ACM*, 29(3):699–717, 1988.

[Pei92] Charles Sanders Peirce. Reasoning and the logic of things. In K. L. Kremer, editor, *The Cambridge Conferences Lectures of 1898*. Harvard Univ. Press, Cambridge, 1992.

[Pre98] Susanne Prediger. *Kontextuelle Urteilslogik mit Begriffsgraphen – Ein Beitrag zur Restrukturierung der Mathematischen Logik*. Shaker Verlag, Aachen, 1998. Dissertation, Technische Universität Darmstadt.

[PS00] Charles Sanders Peirce and John F. Sowa. Existential Graphs: MS 514 by Charles Sanders Peirce with commentary by John Sowa, 1908, 2000. http://users.bestweb.net/~sowa/peirce/ms514.htm.

[Sow84] John F. Sowa. *Conceptual structures: information processing in mind and machine*. Addison-Wesley, 1984.

[Sow92] John F. Sowa. Conceptual graphs summary. In Timothy E. Nagle, Janice A. Nagle, Lauries L. Gerholz, and Peter W. Eklund, editors, *Conceptual Structures: Current Research and Practice*, pages 3–51. Ellis Horwood, 1992.

[Sow00] John F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole, Pacific Grove, CA, 2000.