The theory and practice of coupling formal concept analysis to relational databases

Jens Kötters and Peter W. Eklund†

†School of Information Technology
Deakin University, Geelong, Australia

Abstract. In Formal Concept Analysis, a many-valued context is a collection of objects described by attributes that take on more than binary values, such as age (as integers or ranges of integer values) or color (a list or even a hierarchy of color combinations). Conceptual scaling is the process by which such a many-valued context is transformed into a formal context, by associating a concept lattice with the many-valued context. A many-valued context can be compared to a single table in a relational database populated with multiple rows and non-binary values. A generalization of conceptual scaling as a relational database as a whole should take into account the relations between objects, as expressed by means of foreign keys. Previous approaches to scaling a relational database (e.g. relational scaling) take such relations into account, but either do not maintain a separation between objects and values, which is characteristic for the unary case, or result in unary contexts only. In the approach presented in this paper, the use of n-ary scales is suggested, whereby a relational database is transformed into a family of n-ary contexts (a so called power context family). This paper describes the fundamentals of a Web application that allows connection to a relational database, its scaling interactively into a power context family, and navigation within that context family.

Keywords: Formal concept analysis, relational databases, conjunctive queries, data navigation, power context families, conceptual scaling.

1 Background and Motivation

In a previous paper [12], a variant of Formal Concept Analysis was introduced that uses conjunctive queries as concept intents, and resulting tables as concept extents. The resulting complete lattice of these conjunctive-query/table pairs[1] is a mathematical model of the information space over a relational database (accessible through conjunctive querying). Conjunctive queries correspond to a subset of logical formulas over a relational signature Σ (i.e. Σ is the query vocabulary), and thus have interpretations in a relational structure, which represents the database.

[1] The actual concept lattices have been defined as certain sub-lattices $\mathfrak{C}[\{x_1, \ldots, x_n\}]$.

When representing a database by a relational structure, we have to decide whether the carrier set should consist of *table entries* or *table rows*. The importance of this decision is that it determines what the query variables represent; queries will then either be formulated in the *domain calculus* or in the *tuple calculus*(cf. [1, p.74]). The domain calculus is a natural choice if all database tables represent relations (not just technically, but also conceptually) between objects. We might then employ a simple one-to-one correspondence between databases and relational structures: the n-column tables are precisely the n-ary relations in the relational structure, and the query vocabulary Σ is the set of table names, which are used as n-ary predicates.

In practice, databases are centered around object tables in the ORM-style, and the above modeling option does not reflect how users conceptualize database content. Because objects are represented by table rows, these should constitute the carrier set, so that the relational structure provides interpretations for the tuple calculus (which is also used by SQL [1, p.74]). But, unlike in the hypothetical case above, there is no immediate suitable choice of relations. While the signature Σ should express conditions in a WHERE-clause, meaningless comparisons (e.g. $t.age = t.shoe\_size$) should be eliminated in the formation of concepts.

In this paper, we propose a method to build meaningful query vocabulary around a relational database with reasonable effort. To this end, we utilize an alternative formalization [13] of the conjunctive-query lattice model [12] consistent with Wille's concept graphs [18], work which relates FCA to Sowa's Conceptual Graphs [17]. In particular, the relational database is represented by a power context family [18], i.e. by a sequence of formal contexts.

Power context families and relational structures correspond to each other in an almost trivial way: the columns of the n-th context correspond to n-ary relations, i.e. its attributes are the n-ary symbols of Σ. But the change of formalism encourages to think about databases in terms of conceptual scaling [5, Section 1.3]. Conceptual scaling is a method of deriving a formal context from a *many-valued context*, which can be seen as a database table with value columns only (no foreign keys). Huchard et. al. [10] have presented a way to obtain a context family from a database (a set of many-valued contexts, together with binary inter-object relations), but only unary contexts (where attributes represent object properties) are obtained. The idea of scaling a database into a power context family has been formulated under the title of *relational scaling* [14, 8], but the domain calculus has been used, which may have been the reason for a conflation of objects and values, in so doing destroying the analogy to conceptual scaling. Our scaling approach adheres to the analogy by maintaining a clear separation between objects and values, which in turn leads to generic and reusable scales so that (after the initial creation of scales) scaling a database can be done on a point-and-click basis.

Section 2 describes the graph representation of conjunctive queries that is used throughout this paper. The SQL translation of graphs is detailed in Sect. 3. The scaling approach is described in Section 4. The scales are not only used for the creation of the power context family; they define facets, which control the

user options in a navigation application. Navigation in the power context family is discussed in Section 5. In this context, we revisit previous work on concept graphs and propose a new definition.

2 Conjunctive Queries

Conjunctive queries are a natural subset of database queries with nice theoretical properties [6]. Different representations for conjunctive queries are in popular use, including tableaux, formulas and Datalog rules [1]. In this paper, we represent conjunctive queries with *windowed intension graphs* [13] (similar to conceptual graphs [17]). An example windowed intension graph is shown in Fig. 2. This represents a query for 20th-century-born British authors who published in the 21st century. The rectangles are called *object nodes* and the rounded rectangles *relation nodes*. All object nodes of a query take on the role of variables. Colored nodes represent the subject(s) of the query; they are called *subject nodes*. Only object nodes can be subject nodes. The *window* represents the choice of subject nodes; so it can be thought of as a window into the data. Every object node carries a *sort* label, and a subject node carries in addition a *marker*, so that the combined label is of the form *sort/marker*. Every subject node is associated with a column in the result table, and the marker specifies the name of that column. The available sorts are precisely the table names in the database.

Each relation node is connected to object nodes by $n \geq 1$ outgoing arrows, labeled from 1 to n. A node with n outgoing arrows is said to have *arity* n. Two or more arrows may point to the same object node. A relation node carries one or more labels of the form *facet:attribute*. An *attribute* is a name for an n-ary relation, and a *facet* acts as a namespace for attributes. The label *facet:attribute* states that *attribute* applies to the objects at the end of the arrow tips (the i-th arrow points to the i-th argument). If a node label comprises two or more attributes, they must belong to the same facet.

A *relation sort* is an n-tuple $(s_1, \ldots, s_n)$ of table names. A facet only provides attributes of a single relation sort $(s_1, \ldots, s_n)$, which means that its attributes may only occur on n-ary relation nodes whose i-th arrows point to objects of sort s_i $(i = 1, \ldots, n)$. The nationality and DOB facets of Fig. 2 provide unary relations on Authors, pubdate provides unary relations on Books and wrote binary relations from Authors to Books.

Formally, an intension graph (cf. Fig. 1) is a 4-tuple (V, E, ν, κ) where V is the set of object nodes, E is the set of relation nodes, $\nu(e) := (v_1, \ldots, v_n)$ is the n-tuple of object nodes connected to $e \in E$, and $\kappa(u)$ is the label on the node $u \in V \cup E$. In the original definition of intension graphs [13], ν was required to be injective, so it was omitted. This is convenient for theory, but multiple edges make sense when working with facets.

A windowed intension graph is formalized as a pair $(\lambda, \mathcal{G})$, where $\mathcal{G}$ is an intension graph and $\lambda : X \nrightarrow V_{\mathcal{G}}$ is a partial map from a set X of *markers* to the object nodes of $\mathcal{G}$.

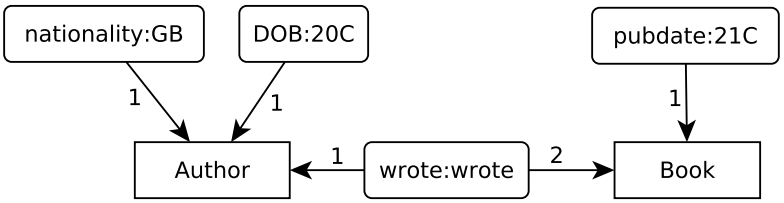

Fig. 1. Intension Graph of the statement "20th-century-born British authors who published in the 21st century"

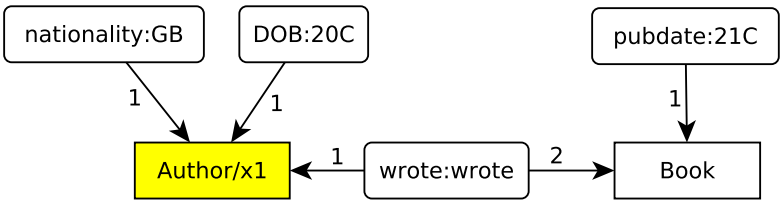

Fig. 2. Windowed Intension Graph of the statement "20th-century-born British authors who published in the 21st century"

Book

title	author	publication_date
Alice in Wonderland	1	1865-11-26
To the Lighthouse	2	1927-05-05
The Hitchhiker's Guide to the Galaxy	3	1979-10-12
Trigger Warning	4	2015-02-03
Harry Potter and the Deathly Hallows	5	2007-07-21
The Casual Vacancy	5	2012-09-27
The Shining	6	1977-01-28
Doctor Sleep	6	2013-09-24
The Da Vinci Code	7	2003-03-18
Inferno	7	2013-03-14

Author

id	first_name	last_name	nationality	date_of_birth
1	Lewis	Carroll	British	1832-01-27
2	Virginia	Woolf	British	1882-01-25
3	Douglas	Adams	British	1952-03-11
4	Neil	Gaiman	British	1960-11-10
5	J. K.	Rowling	British	1965-07-31
6	Stephen	King	American	1947-09-21
7	Dan	Brown	American	1964-06-22

Result Table

x1
Neil Gaiman
J. K. Rowling

Fig. 3. Database for the running example, consisting of tables `Book` and `Author`, and result table for the query in Fig. 2.

3 SQL Translation

A facet c provides an *SQL-interpretation* of its attributes by means of a function Φ_c, which maps an attribute a to a WHERE-condition $\Phi_c(a)$ in the syntax of the target database. If the attribute a has sort $(s_1, \ldots, s_n)$, then $\Phi_c(a)$ contains the placeholders $t_1, \ldots, t_n$, which have to be replaced by the respective arguments in every concrete case.

The expressions below interpret the attributes of the graph in Fig. 2 in terms of the schema of the database in Fig. 3.

$$\Phi_{\mathsf{wrote}}(\mathsf{wrote}) \equiv t_1.\mathsf{id} = t_2.\mathsf{author} \tag{1}$$

$$\Phi_{\mathsf{nationality}}(\mathsf{GB}) \equiv t_1.\mathsf{nationality}=\text{"GB"} \tag{2}$$

$$\Phi_{\mathsf{DOB}}(20\mathrm{C}) \equiv t_1.\mathsf{date\_of\_birth}\ \text{BETWEEN "1999-01-01" AND "1999-12-31"} \tag{3}$$

$$\Phi_{\mathsf{pubdate}}(21\mathrm{C}) \equiv t_1.\mathsf{publication\_date}\ \text{BETWEEN "2000-01-01" AND "2099-12-31"} \tag{4}$$

To obtain a result table as in Fig. 3, we have to specify in addition how the objects of each sort (i.e. the rows of each table) are printed. This is achieved by fixing an *output expression* Ω_s for each sort s. For the sorts of the database in Fig. 3 we specify

$$\Omega_{\mathsf{Author}} \equiv CONCAT(t1.first\_name, \text{" "}, t1.last\_name) \quad , \tag{5}$$

$$\Omega_{\mathsf{Book}} \equiv t1.title \quad . \tag{6}$$

The SQL translation of a windowed intension graph is thus given by a statement of the following form:

$$\begin{aligned} \text{SELECT DISTINCT } & \Omega_{\mathrm{sort}(u_1)}(u_1) \text{ AS } x_1 \text{ ,...,} \\ & \Omega_{\mathrm{sort}(u_m)}(u_m) \text{ AS } x_m \\ \text{FROM } & \mathrm{sort}(v_1) \text{ AS } v_1 \text{ ,...,} \\ & \mathrm{sort}(v_n) \text{ AS } v_n \\ \text{WHERE } & \Phi_{c_1}(a_1)(v_{11}, ..., v_{1n_1}) \text{ AND ...} \\ \text{AND } & \Phi_{c_k}(a_k)(v_{k1}, ..., v_{kn_k}) \end{aligned} \tag{7}$$

In eqn. (7), the object nodes are represented by variables $v_1, \ldots, v_n$, and the FROM-clause can be seen as a variable declaration, which declares v_i to be of sort $\mathrm{sort}(v_i)$. In SQL terminology, v_i is called a *table alias*. The WHERE-clause contains for each attribute $c_j : a_j$ a WHERE-condition $\Phi_{c_j}(a_j)(v_{j1}, \ldots, v_{jn_j})$, where $\Phi_{c_j}(a_j)(v_{j1}, \ldots, v_{jn_j})$ denotes substitution of t_i by the applicable table alias v_{ji}. The FROM-WHERE-part realizes the query's underlying intension graph (cf. Fig. 1). The SELECT-clause defines an output column for each marker x_i on a subject node u_i. In SQL terminology, x_i is called a *column alias*.

So far, result tables display objects, but no values are shown beyond those that occur in the output expression (cf. Fig. 3). In a navigation application user interface (cf. Sect. 5), we envision relation nodes as controls to show in addition (or hide) the column values associated with a facet, by modifying the SELECT-clause in eqn. (7). These additional columns are not part of the concept extent, but they are of course informative. In the next section, we make precise how facets are associated with column values.

4 Database Scaling

A *syntactic interpretation* defines the symbols of a given signature by expressions over another signature (cf. [4]). The attributes provided by a facet c can be understood as symbols of a relational signature. The SQL-interpretation Φ_c is in this sense a syntactic interpretation: namely it interprets the attributes of c in a given database schema S (which is not exactly a signature, but the database-theoretic analogue).

Each facet c defines a context $\mathbb{K}_c$. This is how the extension of the attribute a of $\mathbb{K}_c$ is defined:

$$\begin{aligned} &\text{SELECT } \Omega_{s_1}(t_1),\ \ldots,\ \Omega_{s_n}(t_n) \\ &\quad \text{FROM } s_1 \text{ AS } t_1,\ \ldots,\ s_n \text{ AS } t_n \\ &\text{WHERE } \Phi_c(a)(t_1,\ldots,t_n) \end{aligned} \tag{8}$$

Using eqns. (5) and (3) in eqn. (8), the 20C column in Fig. 4 is obtained. In this manner, the contexts for the DOB and pubdate facets (Figs. 4 and 5) can be derived from the database. The SQL definitions of the attributes *19C*, *20C* and

DOB	19C	20C	21C
Lewis Carroll	×		
Virginia Woolf	×		
Douglas Adams		×	
Neil Gaiman		×	
J. K. Rowling		×	
Stephen King		×	
Dan Brown		×	

Fig. 4. Context for the DOB facet

pubdate	19C	20C	21C
Alice in Wonderland	×		
To the Lighthouse		×	
Hitchhiker's Guide		×	
Harry Potter 7			×
The Casual Vacancy			×
Trigger Warning			×
The Shining		×	
Doctor Sleep			×
The Da Vinci Code			×
Inferno			×

Fig. 5. Context for the pubdate facet

21, which occur in both contexts, are not defined in the facets. Instead, every facet imports its attributes from exactly one underlying scale. A scale in FCA is a formal context which describes *values*; examples are ordinal scales (Fig. 7) and nominal scales (Fig. 8) [5]. The scale that underlies the DOB and pubdate facets is the Centuries scale in Fig. 11. The scales that we use to scale databases (i.e. generate context families from databases) should describe values that can occur in a database column; for the Centuries scale, these are ISO 8601 dates. Of course, it is generally not efficient or even possible to represent scales in the computer as cross-tables; we would expect the Centuries scale to describe all possible dates that can occur in a column, and not just the 17 dates of Fig. 11. A scale for a

database must be able to produce an SQL definition for an attribute. The SQL definition for the 20C attribute is

$$z_1 \text{ BETWEEN "1999-01-01" AND "1999-12-31"} \quad , \tag{9}$$

where $z_1, z_2, z_3, \ldots$ are variables reserved for values. A facet *binds* a scale to one or more columns (each variable z_i is bound to a column). The Centuries scale is a unary scale, so its attributes are described by a single variable z_1. The DOB facet binds z_1 to the Author.date_of_birth column (which yields Φ_{DOB}, cf. (3)), whereas pubdate binds z_1 to Book.publication_date. [2]. Scales encode the actual logic, whereas a facet merely translates a relation between values into a relation between objects, by means of a syntactic substitution that is specified by the binding. The scale interface and facet class are shown in Fig. 12.

Examples for binary scales are equality scales (Fig. 9), which have been used in a prototype to generate binary single-column contexts for foreign keys, or distance scales (Fig. 10), which can be used to measure spatial distance between objects, or time spans between events. A comparison to the classic unary scales (Figs. 7 and 8) shows that these binary scales are in the same spirit.

A syntactic interpretation provides, in addition to symbol definitions, a formula that defines the carrier of the derived structure (cf. [4]). In our scaling approach, this is the object set of the derived context. We call this formula a *domain expression* and denote it by $\Phi_c(*)$, where $*$ is a special symbol. The domain expression for the contexts in Figs. 4 and 5 is a sort restriction. Ideally, the domain expression would also be a WHERE-condition, but SQL requires special treatment in this case. However, a WHERE-condition is allowed in addition to the sort restriction. The domain expression for the wrote facet is "$t1$.id=$t2$.author", where $t1$ is an Author and $t2$ is a Book. On top of this, the *wrote* facet uses a distance scale, bound to Author.date_of_birth and Book.publication_date, to measure at what age an author wrote a particular book. A facet supports renaming of attributes to allow for more expressive attribute names than the generic names provided by the scales. The context derived from the wrote facet is the bottom context in Fig. 6.

The contexts that are derived from the facets can be assembled into a power context family (Fig. 6). Working with power context families is more convenient for mathematical investigations, whereas working with the contexts derived from the facets (as in Figs. 4 and 5) is more convenient for practical work. As with the scales, it is not necessary that the power context family, or individual facets, are explicitly constructed. The power context family has been realized in a prototype as a virtual layer around the database, although the computation of refinement options (cf. Section 5) required an additional query, and SQL does not adequately support all types of scales (such as taxonomies), which may require post-processing of result tables.

[2] Direct specification of $\Phi_c(a)$ in the facet (thus by-passing scales) is not supported.

0	sort: Author	sort: Book
Lewis Carroll	×	
Virginia Woolf	×	
Douglas Adams	×	
Neil Gaiman	×	
J. K. Rowling	×	
Stephen King	×	
Dan Brown	×	
Alice in Wonderland		×
To the Lighthouse		×
Hitchhiker's Guide		×
Harry Potter 7		×
The Casual Vacancy		×
Trigger Warning		×
The Shining		×
Doctor Sleep		×
The Da Vinci Code		×
Inferno		×

1	nationality: GB	nationality: USA	DOB: 19C	DOB: 20C	DOB: 21C	pubdate: 19C	pubdate: 20C	pubdate: 21C
Lewis Carroll	×		×					
Virginia Woolf	×		×					
Douglas Adams	×			×				
Neil Gaiman	×			×				
J. K. Rowling	×			×				
Stephen King		×		×				
Dan Brown		×		×				
Alice in Wonderland						×		
To the Lighthouse							×	
Hitchhiker's Guide							×	
Harry Potter 7								×
The Casual Vacancy								×
Trigger Warning								×
The Shining							×	
Doctor Sleep								×
The Da Vinci Code								×
Inferno								×

2	wrote: wrote	wrote: age≤30	wrote: age≤40	wrote: age≤50
(Lewis Carroll, Alice in Wonderland)	×		×	×
(Virginia Woolf, To the Lighthouse)	×			×
(Douglas Adams, Hitchhiker's Guide)	×	×	×	×
(Neil Gaiman, Trigger Warning)	×			
(J. K. Rowling, Harry Potter 7)	×			×
(J. K. Rowling, The Casual Vacancy)	×			×
(Stephen King, The Shining)	×	×	×	×
(Stephen King, Doctor Sleep)	×			
(Dan Brown, The Da Vinci Code)	×		×	×
(Dan Brown, Inferno)	×			×

Fig. 6. Power Context Family

Ordinal	≤1	≤2	≤3	≤4	≤5
1	×	×	×	×	×
2		×	×	×	×
3			×	×	×
4				×	×
5					×

Fig. 7. Ordinal Scale

Nominal	=1	=2	=3	=4	=5
1	×				
2		×			
3			×		
4				×	
5					×

Fig. 8. Nominal Scale

Equality	=
(1,1)	×
(1,2)	
(1,3)	
(2,1)	
(2,2)	×
(2,3)	
(3,1)	
(3,2)	
(3,3)	×

Fig. 9. Equality Scale

Distance	=0	≤1	≤2
(1,1)	×	×	×
(1,2)		×	×
(1,3)			×
(2,1)		×	×
(2,2)	×	×	×
(2,3)		×	×
(3,1)			×
(3,2)		×	×
(3,3)	×	×	×

Fig. 10. Distance Scale

Centuries	19C	20C	21C
1832-01-27	×		
1865-11-26	×		
1882-01-25	×		
1927-05-05		×	
1947-09-21		×	
1952-03-11		×	
1960-11-10		×	
1964-06-22		×	
1965-07-31		×	
1977-01-28		×	
1979-10-12		×	
2003-03-18			×
2007-07-21			×
2012-09-27			×
2013-03-14			×
2013-09-24			×
2015-02-03			×

Fig. 11. Centuries Scale

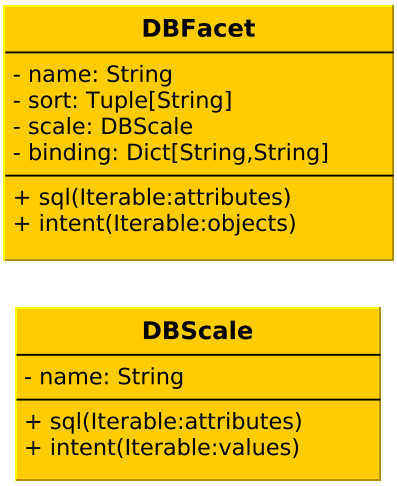

Fig. 12. The main API functions of the DBFacet class and DBScale interface. The internal representation of scales is up to the implementation.

5 Navigation using Projectional Concept Graphs

The ideas of the previous sections can be turned to account in a navigation application. The viability of its core features has already been explored in a prototype; a full version will be presented in an upcoming paper.

The application provides for two roles, user and admin. In the admin role, one can connect to an existing database, view its schema and a list of available scales, bind scales to database columns (thus scaling the database) and store the binding, together with the database connection info, in a file (i.e. the database can be read-only). A binding, together with the database that it references, constitutes a virtual power context family.

In the user role, one can choose from a list of available power context families to navigate in. Note that the user does not need to know that the power context family originates from a relational database, and indeed, there could be different back ends for different sources of data, such as RDF or object-oriented databases (although we have only worked this out for relational databases with SQL access). Different user interfaces are possible, but it is instructive to assume that a conjunctive query looks to the user like the graph in Fig. 2. As mentioned in Sect. 2, it is formalized by a windowed intension graph $(\lambda, \mathcal{G})$.

A *solution* of an intension graph $\mathcal{G}$ in a power context family $\vec{\mathbb{K}}$ is formalized by a map $\varphi : \mathcal{G} \to \vec{\mathbb{K}}$ from object nodes to objects (of the context $\mathbb{K}_0$). The set of all solutions is denoted by $\mathcal{S}(\mathcal{G}, \vec{\mathbb{K}})$. For a windowed intension graph $(\lambda, \mathcal{G})$, the rows in the result table are the maps $\lambda \circ \varphi$ with $\varphi \in \mathcal{S}(\mathcal{G}, \vec{\mathbb{K}})$. In the following, we introduce projectional concept graphs as a basic structure for navigation.

Definition 1 (Projectional Concept Graph). *A* projectional concept graph *is a 5-tuple* $(V, E, \nu, \kappa, \mathrm{ext}_{\vec{\mathbb{K}}})$ *comprised of an intension graph* $\mathcal{G} := (V, E, \nu, \kappa)$ *and its* extension map

$$\mathrm{ext}_{\vec{\mathbb{K}}}(v) := \{\varphi(v) \mid \varphi \in \mathcal{S}(\mathcal{G}, \vec{\mathbb{K}})\} \tag{10}$$

for a given power context family $\vec{\mathbb{K}}$ *with* $\mathcal{S}(\mathcal{G}, \vec{\mathbb{K}}) \neq \emptyset$ *(i.e.* $\mathrm{ext}_{\vec{\mathbb{K}}}(v) \neq \emptyset$ *for all* $v \in V$*). We call* $\mathrm{ext}_{\vec{\mathbb{K}}}(v)$ *the* node extent *of* v*.*

The node extent of the `Author` node in Fig. 1 is the extent of the windowed intension graph in Fig. 2. It is thus an extent in the lattice $\underline{\mathcal{B}}_1(\vec{\mathbb{K}})$ of unary concepts over the power context family $\vec{\mathbb{K}}$ (cf. [13]). Therefore, projectional concept graphs should indeed be considered concept graphs.

Considering node extents rather than whole result tables spares the user going through large result tables; the navigation approach allows however to place windows of arbitrary size on the graph, if the specific combinations of objects in the solution are of interest.

Refinement options are given by a triple $(E^+, \kappa^+, \theta^+)$. For each $v \in V$, $E^+(v)$ is a set of facets for which a new relation node can be connected to v, extending the graph structure. For each $u \in V \cup E$, $\kappa^+(u)$ is a set of scale concepts which can replace $\kappa(u)$. And θ^+ is a set of pairs of object nodes in the graph which can be merged. All refinement options lead to a refined projectional concept graph that has at least one solution in $\vec{\mathbb{K}}$.

6 Related Work

Concept graphs and power context families were defined by Wille [18]. Relational context families, used in Relational Concept Analysis (RCA) developed by Huchard et. al, [9] are similar to power context families but define different contexts for objects of different sorts. The contexts derived from facets in this paper represent facets of such sort contexts. Faceted navigation on the basis of FCA was suggested by Priss [15] and later developed by Eklund and Ducrou [3]. In RCA, conceptual scaling is generalized to relations but produces unary contexts only. The idea of scaling databases into power context families was formulated by Prediger and Wille [14] and expanded on by Hereth [8]. The scales presented there correspond to facets in our work; a central idea to conceptual scaling, the translation of properties of values into properties of objects, is not reflected in this work, but is addressed by the scales in our work.

From the beginning, conceptual graphs have been considered as a database interface [16]. Their translation into logical formulas, stated by Sowa [17], seems to imply that an interpretation as conjunctive queries is intended for conceptual graphs with variables. An SQL translation of Wille's concept Graphs, which treats concept graphs as conjunctive queries, is described by Groh and Eklund [7]. Interpretations were provided by a power context family, but it was encoded in a database (which imposes a particular format), not derived from the database, so scaling (as we define it here) was not involved. Both object and relation nodes were considered variables, whereas in the present work, only object nodes are considered variables.

An intension graph can be thought of as Wille's abstract concept graph (see [18, p. 300]). A concept graph (in the standard definition) defines in addition a realization ϱ which, like the extension map in Def. 1, assigns to each object node a nonempty set of objects. In a concept graph, the elements of the sets $\varrho(v)$ can be freely combined to obtain solutions, whereas for projectional concept graphs, each element of $\varrho(v)$ is part of *some* solution.

The navigation approach of Sect. 5 has been described in [11]. FCA-based navigation in relational data is also the subject of [2].

7 Conclusion

This paper describes the fundamental theory of a Web application that allows connection to a relational database, its scaling interactively into a power context family, and navigation within that context family. The conceptual scaling approach used is based on a syntactic interpretation of attributes, which results in an FCA-based method to build suitable query vocabulary around a relational database. Generic and reusable scales allow easy database scaling on a point-and-click basis. Scales constitute facets in a faceted-navigation approach based on projectional concept graphs, which are a new class of concept graphs.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Azmeh, Z., Huchard, M., Napoli, A., Hacene, M.R., Valtchev, P.: Querying relational concept lattices. In: Proc. of the 8th Intl. Conf. on Concept Lattices and their Applications (CLA'11). pp. 377–392 (2011)
3. Ducrou, J., Eklund, P.: Faceted document navigation using conceptual structures. In: Hitzler, P., Schärfe, H. (eds.) Conceptual Structures in Practice, pp. 245–271. Chapman & Hall/CRC (2009)
4. Ebbinghaus, H.D., Flum, J., Thomas, W.: Mathematical Logic. Springer, New York, second edn. (1994)
5. Ganter, B., Wille, R.: Formal concept analysis: mathematical foundations. Springer, Berlin (1999)
6. Gottlob, G., Leone, N., Scarcello, F.: On tractable queries and constraints. In: Bench-Capon, T.J., Soda, G., Tjoa, A.M. (eds.) Database and Expert Systems Applications. pp. 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
7. Groh, B., Eklund, P.W.: Algorithms for creating relational power context families from conceptual graphs. In: Proceedings of ICCS 1999. pp. 389–400 (1999), `https://doi.org/10.1007/3-540-48659-3_24`
8. Hereth, J.: Relational scaling and databases. In: Priss, U., Corbett, D., Angelova, G. (eds.) Proceedings of ICCS 2002. LNAI, vol. 2393, pp. 62–76. Springer, Heidelberg (2002)
9. Huchard, M., Hacene, M.R., Roume, C., Valtchev, P.: Relational concept discovery in structured datasets. Annals of Mathematics and Artificial Intelligence 49(1-4), 39–76 (2007)
10. Huchard, M., Roume, C., Valtchev, P.: When concepts point at other concepts: the case of UML diagram reconstruction. In: Proceedings of FCAKDD 2002. pp. 32–43 (2002)
11. Kötters, J.: Object configuration browsing in relational databases. In: Valtchev, P., Jäschke, R. (eds.) Proceedings of ICFCA 2011. LNCS, vol. 6628, pp. 151–166. Springer (2011)
12. Kötters, J.: Concept lattices of a relational structure. In: Pfeiffer, H.D., Ignatov, D.I., Poelmans, J., Gadiraju, N. (eds.) Proceedings of ICCS 2013. LNCS, vol. 7735, pp. 301–310. Springer (2013)
13. Kötters, J.: Intension graphs as patterns over power context families. In: Huchard, M., Kuznetsov, S. (eds.) Proceedings of CLA 2016. CEUR Workshop Proceedings, vol. 1624. CEUR-WS.org (2016)
14. Prediger, S., Wille, R.: The lattice of concept graphs of a relationally scaled context. In: William M. Tepfenhart, W.R.C. (ed.) Proceedings of ICCS 1999. LNAI, vol. 1640, pp. 401–414. Springer (1999)
15. Priss, U.: Lattice-based information retrieval. Knowledge Organization 27(3), 132–142 (2000)
16. Sowa, J.F.: Conceptual graphs for a data base interface. IBM Journal of Research and Development 20(4), 336–357 (1976)
17. Sowa, J.F.: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, Reading, MA (1984)
18. Wille, R.: Conceptual graphs and formal concept analysis. In: Lukose, D., Delugach, H.S., Keeler, M., Searle, L., Sowa, J.F. (eds.) Proceedings of ICCS 1997, 5th International Conference on Conceptual Structures. LNCS, vol. 1257, pp. 290–303. Springer, Heidelberg (1997)