Applications of Ordinal Ranks to Flexible Query Answering*

Lucie Urbanova[1], Vilem Vychodil[1], and Lena Wiese[2]

[1] DAMOL (Data Analysis and Modeling Laboratory)
Dept. Computer Science, Palacky University, Olomouc
17. listopadu 12, CZ77146 Olomouc, Czech Republic
`lucie.urbanova01@upol.cz`, `vychodil@acm.org`
[2] Institute of Computer Science, University of Hildesheim
Samelsonplatz 1, 31141 Hildesheim, Germany
`lena.wiese@uni-hildesheim.de`

Abstract. Exact querying and retrieving relevant data from a database is a difficult task. We present an approach for flexibly answering algebraic queries using an extension of Codd's relational model with ordinal ranks based on residuated lattices and similarities on attribute domains.

Keywords: Flexible query answering, ranked data tables, complete residuated lattices, similarity, relational algebra

1 Introduction

As nowadays massive amounts of data are stored in database systems, it becomes more and more difficult for a database user to exactly retrieve data that are relevant to him: it is not easy to formulate a database query such that:

1. on the one hand, the user retrieves all the answers that interest him; that is, false negatives and empty answers are avoided by also giving the user data that are closely related to his original query,
2. on the other hand, the user does not retrieve too much irrelevant data (that is, false positives in the form of "overabundant" and "unsatisfactory answers") – hence avoiding data not related to the user's original intention.

In this article, we assume the following setting: A user sends a query (expressed in relational algebra) to a relational database – that is, a set of data tables, where again each data table consists of a set of attributes and each attribute has been assigned a fixed domain of values (like strings or integers). However, based on the previous work in [3], we extend the relational database by ranks: each tuple in a data table has a rank value to denote how much a tuple matches the user's query. Ranks come from an ordinal scale bounded by 0 (no

* Supported by grant no. `P103/11/1456` of the Czech Science Foundation and internal grant of Palacky University no. `PrF_2012_029`. DAMOL is supported by project reg. no. `CZ.1.07/2.3.00/20.0059` of the European Social Fund in the Czech Republic.

match) and 1 (full match) and we allow to have imperfect matches represented by intermediate ranks. These ordinal ranks have a *comparative* meaning: the higher the rank, the better it matches the user's query. Before the user starts querying the database, all ranks for tuples explicitly contained in the data tables are assumed to be 1, whereas implicitly all other tuples (not occurring in the data tuples but consisting of valid combinations of domain values) are ranked 0. By using ordinal ranks, query answering becomes more *flexible* in the following sense: The user will not only retrieve answer tuples with rank 1 (which might not even exist); instead the user will also retrieve answer tuples with ranks lower than 1 which still may contain relevant information for him. To obtain the ranks, we will employ a notion of similarity on each attribute domain: for any two values from a domain we assume a predefined value that denotes how similar the two domain values are. In particular, we show how to apply the notion of ranked data tables (RDTs) and complete residuated lattices

- to *rank* answer tuples according to their *relevance* for the user,
- to let a user specify *preferences* on equality conditions in his queries,
- and to suppress *irrelevant* answers by a global threshold.

2 Background on Ranked Data Tables

The flexible query answering approach we discuss in this paper is based on a similarity-based generalization of Codd's relational model of data [10]. As in the traditional Codd's model, we assume a set Y of attributes where each attribute $y \in Y$ has a specific fixed domain denoted D_y. A relation scheme consists of a finite subset of the attributes $R \subseteq Y$. Each data table (or relation instance) for a relation scheme R is a finite set of tuples where a tuple is a map $r\colon R \to \bigcup_{y \in R} D_y$ such that the value of the tuple for an attribute complies with the domain of the attribute, that is: $r(y) \in D_y$. The set of all tuples $r\colon R \to \bigcup_{y \in R} D_y$ will also be denoted by $\mathrm{Tupl}(R)$.

An important aspect of Codd's model is that data tables (relations) represent both the stored data and results of queries. In fact, in theory there is no distinction between the two roles as stored data can be seen as results of queries (show all stored data) and results of queries can again be stored. In both the cases, a data table $\mathcal{D}$ is a finite subset of $\mathrm{Tupl}(R)$, i.e., we may think of the tuples in $\mathcal{D}$ as tuples assigned a rank 1 indicating a match. Analogously, the tuples not present in $\mathcal{D}$ can be seen as tuples assigned a rank 0 indicating no match. Clearly, there is a one-to-one correspondence between data tables $\mathcal{D} \subseteq \mathrm{Tupl}(R)$ and such assignments where at most finitely many tuples from $\mathrm{Tupl}(R)$ are assigned rank 1. It is natural to interpret the ranks 0 and 1 as truth degrees which come from a two-element Boolean algebra with the usual interpretation (0 for falsity and 1 for truth) and ordering $0 < 1$. From this point of view, we argue that ordinal ranks which we use here in a more general setting are already present in the original Codd's model, only they do not appear explicitly. The primary role of ranks stems directly from the model – they indicate whether a tuple matches or does not match a given query (formulated in a particular query language, e.g. a relational algebra, and evaluated in a database instance).

Example 1. As a running example we assume a database of book stores and their stock. That is, we have a table of bookstores storing the name of a bookstore, the ZIP of the warehouse where the book is on stock, and the ISBN and price of each book sold in each store; and we have a table of books storing ISBN, title, and level of presentation. Hence a database instance may look as follows.

Stores	*rank*	*name*	*zip*	*isbn*	*price*
	1.0	Bookworm	38457	037-4-592-24599-7	59.95
	1.0	Bookworm	38457	834-3-945-25365-9	19.95
	1.0	Bookmarket	32784	834-3-945-25365-9	23.95
	1.0	BooksBooks	98765	945-7-392-66845-4	89.95

Books	*rank*	*bookid*	*title*	*level*
	1.0	037-4-592-24599-7	SQL	beginner
	1.0	834-3-945-25365-9	Databases	advanced
	1.0	945-7-392-66845-4	DB systems	professional

In this paper, to allow for flexible query answering, we utilize an extension of Codd's model which allows us to consider general ranks, not only 0 (false or no match) and 1 (true or match), coming from a general partially ordered set which is bounded by 0 and 1. Technically, the extension we use here results from Codd's model if we replace the two-valued Boolean algebra which serves as the structure of truth degrees by a more general structure. Hence, the approach we use in this paper builds upon Codd's model which is developed using a weaker metamathematics. There are several important practical consequences:

Clarity: The model stays purely relational. There is no "ad hoc" ranking module attached on top of the classic model. The classic model results from the model used here by a particular choice of the structure of ranks. Namely, if the structure is bivalent (the two-valued Boolean algebra), our model becomes the ordinary model with yes/no matches.

Ranked data tables are used instead of the classic data tables as the basic structures and represent both the results of queries and stored data. Each ranked data table $\mathcal{D}$ (an RDT) is a map assigning to each tuple $r \in \mathrm{Tupl}(R)$ a rank denoted $\mathcal{D}(r)$. The rank is interpreted as a degree to which r matches a query. The ranks have ordinal interpretation, $\mathcal{D}(r_1) > \mathcal{D}(r_2)$ means that r_1 is a better match than r_2, $\mathcal{D}(r) = 1$ means that r matches fully (a given query), $\mathcal{D}(s) = 0$ means that s does not match the query at all.

Support for imperfect matches: As in Codd's model, queries are represented by expressions which are evaluated in database instances (there are several equivalent query systems like relational algebra and domain relational calculus with range declarations). The fact that a general structure of ranks is used influences the rules how the queries are evaluated – there is a need to aggregate values of ranks that can be other than 0 and 1 and thus the operations of Boolean algebra can no longer do the job. In order to evaluate general queries, the structure of ranks shall be equipped by additional operations for aggregation of ranks. Nevertheless, the evaluation stays truth functional as in Codd's model. As a consequence, the results of queries are given only by the database instance and the structure of ranks.

Equalities replaced by similarities: Equalities on domains are an integral part of Codd's model and appear in restrictions (selections), natural joins, the semantics of functional dependencies, etc. In our model, equalities on domains (tacitly used in Codd's model) become explicit similarity relations, assigning to each two elements from a domain D_y of attribute y a degree to which they are similar. Similarity degrees come from the same scale as ranks and have the same ordinal interpretation: higher degrees mean higher similarity and thus higher preference if one is asked to choose between alternatives.

Other important aspects: The model is general and not limited just to data querying. There are results on various types of similarity-based dependencies in data [2] that can be exploited in the process of flexible query answering.

We now outline a fragment of the model which is sufficient to discuss flexible query answering with ordinal ranks. We use the unit interval on the reals $L = [0,1]$ in all our examples which will both constitute the ranks of tuples in a data table as well as similarity degrees between any two values from a domain. In general, the scale can be an arbitrary set L bounded by 0 and 1. In order to be able to compare ranks and similarities, we equip L with a partial order $\leq$ so that $\langle L, \leq \rangle$ is a complete lattice. That means, for each subset of L there exists a supremum and an infimum with respect to $\leq$ and $\langle L, \leq \rangle$ can be alternatively denoted by $\langle L, \wedge, \vee, 0, 1 \rangle$ where $\wedge$ and $\vee$ denote the operations of infimum and supremum, respectively. In case of the real unit interval $L = [0,1]$ and its natural ordering, the suprema and infima of finite nonempty subsets of $[0,1]$ coincide with their maxima and minima. Moreover, we accompany the complete lattice with two binary operations that operate on any two elements of L, result in an element of L and play roles of truth functions of logical connectives "conjunction" and "implication" which are used in the process of evaluating queries as we shall see later. These operations are a multiplication $\otimes$ and a residuum $\rightarrow$. We postulate that $\langle L, \otimes, 1 \rangle$ is a commutative monoid and $\otimes$ and $\rightarrow$ satisfy a so-called adjointness property: $a \otimes b \leq c$ iff $a \leq b \rightarrow c$ (for all $a, b, c \in L$). Altogether, $\mathbf{L} = \langle L, \wedge, \vee, \otimes, \rightarrow, 0, 1 \rangle$ is called a *complete residuated lattice.*

Remark 1. The conditions for $\otimes$ and $\rightarrow$ were derived by Goguen [13] from a graded counterpart of *modus ponens* and were later employed in various multiple-valued logics with truth-functional semantics, most notably in BL [16], MTL [11] and their schematic extensions. Nowadays, adjointness is considered as a property which ensures that $\otimes$ and $\rightarrow$ are truth functions for multiple-valued conjunction and implication with reasonable properties. The properties are weaker than the properties of two-valued conjunction and implication but sufficient enough to have syntactico-semantically complete logics, see [16] and [9] for an overview of recent results. A particular case of a complete residuated lattice is a two-valued Boolean algebra if $L = [0,1]$, $\wedge = \otimes$ is the truth function of ordinary conjunction, $\vee$ and $\rightarrow$ are truth functions of disjunction and implication, respectively.

Examples of complete residuated lattices include finite as well as infinite structures. For the real unit interval $L = [0, 1]$ and its natural ordering, all complete residuated lattices are given by a left-continuous t-norm $\otimes$, see [1, 16]. Moreover, all complete residuated lattices with continuous $\otimes$ can be constructed by means of ordinal sums [8] from the following three pairs of adjoint operations:

Łukasiewicz	$a \otimes b = \max(a + b - 1, 0)$	$a \to b = \min(1 - a + b, 1)$
Gödel	$a \otimes b = \min(a, b)$	$a \to b = b$ if $a > b$; 1 otherwise
Goguen	$a \otimes b = a \cdot b$	$a \to b = \frac{b}{a}$ if $a > b$; 1 otherwise

Recall that an $\mathbf{L}$-set (a fuzzy set) A in universe U is a map $A\colon U \to L$, $A(u)$ being interpreted as "the degree to which u belongs to A". A binary $\mathbf{L}$-relation (a binary fuzzy relation) B on U is a map $B\colon U \times U \to L$, $B(u_1, u_2)$ interpreted as "the degree to which u_1 and u_2 are related according to B". See [1] for details.

Definition 1 (ranked data table). Let $R \subseteq Y$ be a relation scheme. A *ranked data table* on R (shortly, a RDT) is any map $\mathcal{D}\colon \mathrm{Tupl}(R) \to L$ such that there are at most finitely many tuples $r \in \mathrm{Tupl}(R)$ such that $\mathcal{D}(r) > 0$. The degree $\mathcal{D}(r)$ assigned to tuple r by $\mathcal{D}$ shall be called a *rank* of tuple r in $\mathcal{D}$.

Remark 2. The number of tuples which are assigned nonzero ranks in $\mathcal{D}$ is denoted by $|\mathcal{D}|$, i.e. $|\mathcal{D}|$ is the cardinality of $\{r \mid \mathcal{D}(r) > 0\}$. We call $\mathcal{D}$ *non-ranked* if $\mathcal{D}(r) \in \{0, 1\}$ for all tuples r. The non-ranked RDTs can be seen as initial data, i.e., relations in the usual sense representing stored data as in the classic model.

In this paper, we consider queries formulated using combinations of relational operations on RDTs. The operations extend the classic operations by considering general ranks. For our purposes, it suffices to introduce the following operations:

Intersection: An intersection (a $\otimes$-*intersection*) of RDTs $\mathcal{D}_1$ and $\mathcal{D}_2$ on relation scheme T is defined componentwise using $\otimes$: $(\mathcal{D}_1 \otimes \mathcal{D}_2)(t) = \mathcal{D}_1(t) \otimes \mathcal{D}_2(t)$ for all tuples t. If $\mathcal{D}_1$ and $\mathcal{D}_2$ are answers to queries Q_1 and Q_2, respectively, then $\mathcal{D}_1 \otimes \mathcal{D}_2$ is an answer to the conjunctive query "Q_1 *and* Q_2".

Projection: If $\mathcal{D}$ is an RDT on relation scheme T, the *projection* $\pi_R(\mathcal{D})$ *of* $\mathcal{D}$ *onto* $R \subseteq T$ is defined by $(\pi_R(\mathcal{D}))(r) = \bigvee_{s \in \mathrm{Tupl}(T \setminus R)} \mathcal{D}(rs)$ for each $r \in \mathrm{Tupl}(R)$. Note that rs denotes a usual concatenation of tuples r and s which is a set-theoretic union of r and s. Projection has the same meaning as in the Codd's model and the supremum $\bigvee$ aggregating the ranks $\mathcal{D}(rs)$ is used because of the existential interpretation of the projection.

Cross join (Cartesian product): For RDTs $\mathcal{D}_1$ and $\mathcal{D}_2$ on disjoint relation schemes S and T we define an RDT $\mathcal{D}_1 \bowtie \mathcal{D}_2$ on $S \cup T$, called a *cross join of* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ (or, a Cartesian product of $\mathcal{D}_1$ and $\mathcal{D}_2$), by $(\mathcal{D}_1 \bowtie \mathcal{D}_2)(st) = \mathcal{D}_1(s) \otimes \mathcal{D}_2(t)$. The cross join $\mathcal{D}_1 \bowtie \mathcal{D}_2$ contains tuples which consist of concatenations of tuples from $\mathcal{D}_1$ and $\mathcal{D}_2$. Note that in our model, we can have $|\mathcal{D}_1 \bowtie \mathcal{D}_2| < |\mathcal{D}_1| \cdot |\mathcal{D}_2|$ (e.g., if $\otimes$ is the Łukasiewicz conjunction).

Renaming attributes: The same operation as in the Codd's model [19].

If we apply these operations on non-ranked RDTs, we always obtain a non-ranked RDT. RDTs with ranks other than 0 and 1 result from non-ranked RDTs by using similarity-based restrictions (selections): Given a nonranked RDT $\mathcal{D}$ and a formula like $y \approx d$ saying "(the value of the attribute) y is *similar to* d", we select from $\mathcal{D}$ only the tuples which match this similarity-based condition. Naturally, the condition shall be matched to degrees: each tuple from $\mathcal{D}$ is assigned a rank representing the degree to which the tuples matches the condition.

In order to formalize similarity-based restrictions, we equip each domain D_y with a binary $\mathbf{L}$-relation $\approx_y$ on D_y which satisfies the following conditions: (i) for each $u \in D_y$: $u \approx_y u = 1$ (reflexivity), and (ii) for each $u, v \in D_y$: $u \approx_y v = v \approx_y u$ (symmetry). Such a relation shall be called a *similarity.* Taking into account similarities on domains, we introduce the following operation:

Restriction (selection): Let $\mathcal{D}$ be an RDT on T and let $y \in T$ and $d \in D_y$. A *similarity-based restriction* $\sigma_{y\approx d}(\mathcal{D})$ *of tuples in* $\mathcal{D}$ *matching* $y \approx d$ is defined by $\big(\sigma_{y\approx d}(\mathcal{D})\big)(t) = \mathcal{D}(t) \otimes t(y) \approx_y d$. Considering $\mathcal{D}$ as a result of query Q, the rank of t in $\sigma_{y\approx d}(\mathcal{D})$ is interpreted as a degree to which "t matches the query Q and the y-value of t is similar to d". We can consider more general restrictions $\sigma_{\varphi}(\mathcal{D})$, where φ is a more complex formula than $y \approx d$.

Using cross joins and similarity-based restrictions, we can introduce *similarity-based equijoins* such as $\mathcal{D}_1 \bowtie_{p\approx q} \mathcal{D}_2 = \sigma_{p\approx q}(\mathcal{D}_1 \bowtie \mathcal{D}_2)$. Various other types of similarity-based joins can be introduced in our model. Details can be found in [3].

3 Similarity-Based Ranking of Database Answers

If a user specifies equality conditions in his query to select tuples from the database, these equality conditions may be too strict to give the user a satisfactory answer. The simplest setting to retrieve more relevant answers for the user is to replace equality $=$ in each condition with similarity $\approx$ and then use the similarity-based algebraic operators to obtain ranks for the answer tuples. The ranks will be based on the predetermined similarities between domain values. For simplicity, we concentrate in this paper on selection-projection-join (SPJ) queries. Other operators like division, union and intersection can be incorporated like in [3]. We may also introduce in our model a top_k operation in much the same sense as, e.g., in RankSQL [18] but in a truth-functional way. We do not discuss these issues here because of the limited scope of this paper.

Example 2. For example, we can ask for titles and prices of books sold by a bookstore with ZIP code 35455 by doing an equi-join over the ISBN:

$$\pi_{\texttt{title,price}}[\sigma_{\texttt{zip=35455}}(\mathit{Books} \bowtie_{\texttt{bookid=isbn}} \mathit{Stores})]$$

Or ask for the name of a bookstore that sells "professional books" entitled "Databases":

$$\pi_{\texttt{name}}[\sigma_{\texttt{level=professional,title=Databases}}(\mathit{Books} \bowtie_{\texttt{bookid=isbn}} \mathit{Stores})]$$

In general, we consider SPJ queries of the form:

$$\pi_{\mathcal{A}}[\sigma_{\mathcal{C}}(\rho_{\mathcal{R}}(\mathcal{D}_1) \bowtie_{\mathcal{E}_1} \cdots \bowtie_{\mathcal{E}_m} \rho_{\mathcal{R}}(\mathcal{D}_n))] \quad \text{where}$$

– $\mathcal{D}_i$ is a ranked data table
– $\mathcal{C}$ is a conjunction of selection conditions consisting of
  - equalities $y = d$ (where y is an attribute in the relation scheme and d is a constant from its domain D_y)
  - equalities $y_1 = y_2$ (where y_1 and y_2 are attributes in the relation scheme with a common domain $\mathcal{D}_i$)
– $\mathcal{E}_j$ is a set of join conditions as equalities between attributes $y_1 = y_2$ (where y_1 and y_2 are attributes in the relation scheme of some $\mathcal{D}_i$)
– $\mathcal{R}$ is a list of renaming conditions $y' \leftarrow y$ giving attribute y a new name y' (where y is an attribute in the relation scheme of the appropriate $\mathcal{D}_i$ but y' does not occur in any relation scheme)
– $\mathcal{A}$ is the set of projection attributes (after renaming according to $\mathcal{R}$)

Example 3. For the example SPJ queries there are no tuples in the example tables that satisfy the equality conditions. Now assume that similarity is defined on attribute `zip` as (`35455`≈`38457`)=0.9 as well (`35455`≈`32784`)=0.8 and (`35455`≈`98765`)=0.2; whereas for `bookid` and `isbn` we assume that similarity is defined by strict equality, that is it is 1 only for exactly the same ISBN and 0 otherwise: $(d \approx d) = 1$, but $(d \approx d') = 0$ for $d \neq d'$. To obtain a ranked data table as a flexible answer for the first query, we replace equality with similarity:

$$\pi_{\texttt{title,price}}[\sigma_{\texttt{zip}\approx\texttt{35455}}(\mathit{Books} \bowtie_{\texttt{bookid}\approx\texttt{isbn}} \mathit{Stores})]$$

results in the following ranked answer table:

rank	*title*	*price*
$0.9(= 0.9 \otimes 1)$	SQL	59.95
$0.9(= 0.9 \otimes 1)$	Databases	19.95
$0.8(= 0.8 \otimes 1)$	Databases	23.95
$0.2(= 0.2 \otimes 1)$	DB systems	89.95

By furthermore assuming similarity on the attributes `level` and `zip` we can also relax the second query. For example, let (`professional`≈`advanced`)=0.6 and (`professional`≈`beginner`)=0.1, as well as (`Databases`≈`DB systems`)=0.9 and (`Databases`≈`SQL`)=0.7. Then the query

$$\pi_{\texttt{name}}[\sigma_{\texttt{level}\approx\texttt{professional},\texttt{title}\approx\texttt{Databases}}(\mathit{Books} \bowtie_{\texttt{bookid}\approx\texttt{isbn}} \mathit{Stores})]$$

returns the table

rank	*name*
$0.9(= 1 \otimes 0.9 \otimes 1)$	BooksBooks
$0.6(= 0.1 \otimes 0.7 \otimes 1 \vee 0.6 \otimes 1 \otimes 1)$	Bookworm
$0.6(= 0.6 \otimes 1 \otimes 1)$	Bookmarket

4 Emphasizing some Equality Conditions

Extending this simple setting, a user might want to express importance of some equality conditions in his queries. For one, he might want to express that he requires equality (that is full similarity to degree 1) in a selection condition. More generally, a user must be able to express that some conditions are more important for him – and hence to require a higher degree of satisfaction for these conditions; whereas for other conditions he is more willing to relax the equality requirement – and he will be content with a lower degree of satisfaction for these conditions. In our model, this kind of emphasis mechanism can be implemented using so-called residuated shifts as follows. Similarity-based restrictions $\sigma_{y\approx d}(\mathcal{D})$ that appear in our queries can be generalized so that we consider a more general formula of the form $a \Rightarrow y \approx d$ instead of $y \approx d$. In this setting, we introduce a *similarity-based restriction* $\sigma_{a\Rightarrow y\approx d}(\mathcal{D})$ *of tuples in* $\mathcal{D}$ *matching* $y \approx d$ *at least to degree* $a \in L$ defined with the help of the residuum operator $(\rightarrow)$ by

$$\big(\sigma_{a\Rightarrow y\approx d}(\mathcal{D})\big)(t) = \mathcal{D}(t) \otimes \big(a \rightarrow t(y) \approx_y d\big). \tag{1}$$

Using adjointness, we get that $1 \rightarrow a = a$ for all $a \in L$. Hence, $\sigma_{1\Rightarrow y\approx d}(\mathcal{D}) = \sigma_{y\approx d}(\mathcal{D})$. On the other hand, $0 \rightarrow a = 1$ for all $a \in L$, meaning $\sigma_{0\Rightarrow y\approx d}(\mathcal{D}) = \mathcal{D}$. Due to the monotony of $\otimes$ and antitony of $\rightarrow$ in the first argument, we get

$$\big(\sigma_{b\Rightarrow y\approx d}(\mathcal{D})\big)(t) \leq \big(\sigma_{a\Rightarrow y\approx d}(\mathcal{D})\big)(t)$$

whenever $a \leq b$. By a slight abuse of notation, the latter fact can be written as $\sigma_{b\Rightarrow y\approx d}(\mathcal{D}) \subseteq \sigma_{a\Rightarrow y\approx d}(\mathcal{D})$. Therefore, $a \in L$ in $\sigma_{a\Rightarrow y\approx d}(\mathcal{D})$ acts as a *threshold degree*, the lower the degree, the lower the emphasis on the condition $y \approx d$ and, in consequence, the larger the answer set. In the borderline cases, $\sigma_{0\Rightarrow y\approx d}(\mathcal{D})$ means no emphasis on the condition, $\sigma_{1\Rightarrow y\approx d}(\mathcal{D})$ means full emphasis, i.e., the original similarity-based selection.

Remark 3. Let us comment on the role of degrees as thresholds. Using adjointness, for all $a, b \in L$, we have $a \leq b$ iff $a \rightarrow b = 1$. Applied to (1), $a \rightarrow t(y) \approx_y d = 1$ iff the y-value of t is similar to d at least to degree a. Therefore, if $\mathcal{D}$ is a result of query Q, then the rank $\sigma_{a\Rightarrow y\approx d}(\mathcal{D})(t)$ shall be interpreted as a degree to which "t matches Q and the y-value of t is similar to d at least to degree a". This justifies the interpretation of $a \in L$ as a threshold degree. The benefit of using $a \rightarrow t(y) \approx_y d$ in (1) which is in fuzzy relational systems [1] called a residuated shift, is that the threshold is exceeded gradually and not just "exceeded in terms yes/no". For instance, if $a > t(y) \approx_y d$ and the similarity degree $t(y) \approx_y d$ is sufficiently close to a, the result of $a \rightarrow t(y) \approx_y d$ is not 1 but it shall be sufficiently close to 1, expressing the fact that the threshold has almost been exceeded, i.e., that the y-value of t is similar to d almost to degree a. This is illustrated by the following example.

Example 4. Assume we want to express that a similarity on the ZIP code of above 0.8 is perfectly fine for us, then the first query looks like this

$$\pi_{\texttt{title,price}}[\sigma_{0.8\Rightarrow(\texttt{zip}\approx 35455)}(\mathit{Books} \bowtie_{\texttt{bookid}\approx\texttt{isbn}} \mathit{Stores})]$$

and results in the following ranked answer table by evaluating $0.8 \to 0.9$ to 1.0, $0.8 \to 0.8$ to 1.0, and $0.8 \to 0.2$ to 0.2 in the Gödel algebra (this value would be 0.4 in Łukasiewicz and 0.25 in Goguen algebra):

rank	*title*	*price*
$1.0(= (0.8 \to 0.9) \otimes 1)$	`SQL`	`59.95`
$1.0(= (0.8 \to 0.9) \otimes 1)$	`Databases`	`19.95`
$1.0(= (0.8 \to 0.8) \otimes 1)$	`Databases`	`23.95`
$0.2(= (0.8 \to 0.2) \otimes 1)$	`DB systems`	`89.95`

In the second query, if we insist on books with professional level, but we are indeed interested in books with related titles up to a similarity of 0.5:

$$\pi_{\texttt{name}}[\sigma_{1 \Rightarrow (\texttt{level} \approx \texttt{professional}), 0.5 \Rightarrow (\texttt{title} \approx \texttt{Databases})}(\mathit{Books} \bowtie_{\texttt{bookid} \approx \texttt{isbn}} \mathit{Stores})]$$

then we get the answer table

rank	*name*
$1.0(= (1 \to 1) \otimes (0.5 \to 0.9) \otimes 1)$	`BooksBooks`
$0.6(= (1 \to 0.1) \otimes (0.5 \to 0.7) \otimes 1 \vee (1 \to 0.6) \otimes (0.5 \to 1) \otimes 1)$	`Bookworm`
$0.6(= (1 \to 0.6) \otimes (0.5 \to 1) \otimes 1)$	`Bookmarket`

5 Global Relevance Threshold for Subqueries

The results of queries in our model are influenced by the underlying structure of truth degrees **L** because the operations of **L** are used to aggregate ranks. Hence, different choices of **L** in general lead to different answer sets. By a careful choice of the structure of truth degrees, we can allow users to influence the size of the answer set so that the user can tune the structure to obtain an answer set of the most desirable size. In this section, we consider a situation where we want to emphasize answer tuples that satisfy at least some subqueries (similarity conditions) with a high degree. Opposed to this, answer tuples that satisfy all subqueries (similarity conditions) with lower degrees than desirable should be ranked considerably lower. The user might then specify a global threshold (as opposed to the local thresholds in Section 4) to denote that values above the threshold are relevant to him whereas values below the threshold are irrelevant. Interestingly, this idea can be implemented in our model by a choice of the structure **L** *without* altering the relational operations.

Remark 4. Note that the Goguen $\otimes$ (usual multiplication of real numbers) has the property that $a \otimes b > 0$ for all $a, b > 0$ ($\otimes$ is called strict), see Fig. 1 (right). The Łukasiewicz $\otimes$ does not have this property: for each $0 < a < 1$ there is $b > 0$ such that $a \otimes b = 0$ ($\otimes$ is called nilponent), see Fig. 1 (left). Thus, if one uses the Goguen operations on $[0, 1]$, each query considered in this paper has a nonempty answer set provided that all similarities have the property that $d_1 \approx_y d_2 > 0$ which can be technically ensured. In practice, the benefit of always having a nonempty answer set this way is foiled by having (typically) a large number of answers with very low ranks which match the initial query only to a

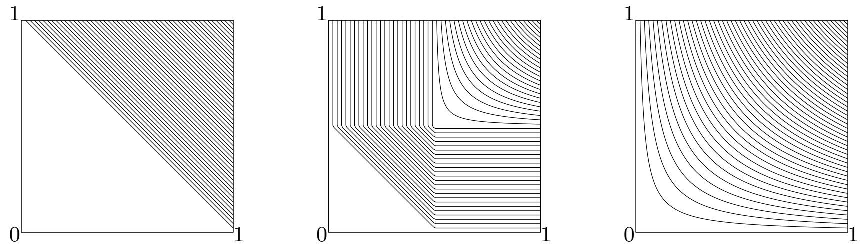

Fig. 1. Contour diagrams: Łukasiewicz $\otimes$ (left), $\otimes_{0.5}$ (middle), Goguen $\otimes$ (right).

very low degree and are (typically) not interesting for a user. On the contrary, using the Łukasiewicz operations instead of the Goguen ones, one can have a situation where there are subqueries satisfied to high degrees, say $a_1, \ldots, a_k$ but $a_1 \otimes \cdots \otimes a_k = 0$, i.e., the answer tuples will be lost since their rank will be zero.

Considering the behavior of the Goguen and Łukasiewicz operations mentioned in Remark 4 and our motivation to separate answers with sufficiently high degrees from the rest, we may consider complete residuated lattices $\mathbf{L}$ defined on the real unit interval that act as the Goguen structure on a subinterval $(\alpha, 1]$ and as the Łukasiewicz structure on the subinterval $[0, \alpha)$. The degree $\alpha \in [0, 1]$ is then a threshold that can be set by a user saying that "*if ranks of answer tuples exceed the threshold $\alpha \in L$, they shall not vanish from the answer set*". The combination of Goguen and Łukasiewicz structure we need for this particular purpose can be described as a result of algebraic operation called an ordinal sum [8]. For $\alpha \in [0, 1]$, we let

$$a \otimes_\alpha b = \begin{cases} \alpha + \dfrac{(a-\alpha)(b-\alpha)}{1-\alpha}, & \text{if } a, b \in (\alpha, 1), \\ \max(0, a + b - \alpha), & \text{if } a, b \in (0, \alpha), \\ \min(a, b), & \text{otherwise,} \end{cases}$$

$$a \rightarrow_\alpha b = \begin{cases} 1, & \text{if } a \leq b, \\ \alpha + \dfrac{(1-\alpha)(b-\alpha)}{a-\alpha}, & \text{if } 1 > a > b > \alpha, \\ \alpha - a + b, & \text{if } \alpha > a > b, \\ b, & \text{otherwise,} \end{cases}$$

for all $a, b \in [0, 1]$. Then, $\mathbf{L}_\alpha = \langle L, \wedge, \vee, \otimes_\alpha, \rightarrow_\alpha, 0, 1 \rangle$ is a complete residuated lattice (an ordinal sum of an isomorphic copy of a Łukasiewicz structure and Goguen structure with the idempotent $\alpha \in L$, see [1, 8, 16] for details. Fig. 1 (middle) shows $\otimes_{0.5}$, i.e., the multiplication of $\mathbf{L}_{0.5}$ which behaves as the Łukasiewicz conjunction on $[0, 0.5)$ and the Goguen conjunction on $(0.5, 1]$.

Remark 5. Taking $\mathbf{L}_\alpha$ for the structure of degrees has the advantage that the threshold $\alpha \in L$ is *not* acting as a clear "cut" where answer tuples with a degree below the threshold are completely disregarded; instead, it leads to a stepwise degradation (but potentially with a degree still above 0) depending on how many subqueries are satisfied by the answer tuple only to a degree below the threshold: the more subqueries are below the threshold the closer the overall degree will be to 0. In effect, with this threshold we can flexibly increase or decrease the size (that is, number of tuples) in the result as it is shown by the following assertions.

The following theorem states that you get more results with a lower threshold and that results above a certain threshold will be ranked lower by a lower threshold.

Theorem 1. *Let $\mathcal{D}^{Q}_{\mathbf{L}_\alpha}$ be a result of an SPJ query Q when $\mathbf{L}_\alpha$ is used as the structure of truth degrees. If $\alpha < \beta$, then $|\mathcal{D}^{Q}_{\mathbf{L}_\beta}| \leq |\mathcal{D}^{Q}_{\mathbf{L}_\alpha}|$. If $\alpha < \beta$ and a tuple t satisfies all subqueries of Q to a degree greater than β, then $\big(\mathcal{D}^{Q}_{\mathbf{L}_\alpha}\big)(t) < \big(\mathcal{D}^{Q}_{\mathbf{L}_\beta}\big)(t)$.*

Proof (a sketch). The claim follows from fact that $a \otimes_\alpha b = 0$ implies $a \otimes_\beta b = 0$ whenever $\alpha \leq \beta$, i.e. α yields a greater (or equally sized) answer set than β. The second claim is a consequence of $a \otimes_\alpha b < a \otimes_\beta b$ for $a, b \in (\beta, 1)$. □

Example 5. Consider the following query with four similarity conditions

$$\sigma_{\texttt{title}\approx\texttt{SQL},\texttt{level}\approx\texttt{beginner},\texttt{zip}\approx\texttt{56571},\texttt{price}\approx\texttt{20.00}}(\mathit{Books} \bowtie_{\texttt{bookid}\approx\texttt{isbn}} \mathit{Stores})$$

In the table $\mathit{Books} \bowtie_{\texttt{bookid}\approx\texttt{isbn}} \mathit{Stores}$ we have the tuple ⟨`834-3-945-25365-9`, `Databases`, `advanced`, `Bookworm`, `32784`, `834-3-945-25365-9`, `19.95`⟩. With the following given similarities (`SQL`≈`Databases`)=0.7, (`beginner`≈`advanced`)=0.4, (`56571`≈`32784`)=0.1, and (`20.00`≈`19.95`)=0.9, we see that two similarity conditions are satisfied at high degrees (0.7 and 0.9) while the other two are satisfied at low degrees (0.4 and 0.1). Comparing the different $\otimes$ operators we get:

Łukasiewicz	Gödel	Goguen	$\otimes_{0.5}$	$\otimes_{0.4}$	$\otimes_{0.3}$
0	0.1	0.0252	0	0.1	0.1

We see that for threshold $\alpha = 0.5$ the tuple is not included in the answer table (ranked 0); when decreasing the threshold to $\alpha = 0.4$ it is included (ranked 0.1) and stays in the answer for the lower threshold $\alpha = 0.3$.

Another potential tuple is ⟨`945-7-392-66845-4`, `DB systems`, `professional`, `BooksBooks`, `98765`, `945-7-392-66845-4`, `89.95`⟩. With similarities (`SQL`≈`DB systems`)=0.3, (`beginner`≈`professional`)=0.1, (`56571`≈`98765`)=0.1, as well as (`20.00`≈`89.95`)=0.1, we see that all similarity conditions are satisfied at low degrees (0.3 and 0.1). Comparing the different $\otimes$ operators we get:

Łukasiewicz	Gödel	Goguen	$\otimes_{0.5}$	$\otimes_{0.4}$	$\otimes_{0.3}$
0	0.1	0.0003	0	0	0

Hence, even for low thresholds the tuple does not occur in the answer table.

6 Related Work

The notion of answering queries in a flexible and user-oriented manner has been investigated for a long time; see [12] for some recent examples and [21] for an extensive study on fuzzy querying (and a query language called SQLf). Other related work of Bosc et al considers extending the support of a fuzzy membership function in order to weaken fuzzy conditions in a query [4] where a tolerance relation can be used to flexibly model closeness in a domain. In [14], Hadjali and Pivert analyze the use of fuzzy views and study the influence of different fuzzy implication operators (like Gödel, Goguen and Łukasiewicz implication) on which views in a distributed database system are chosen to answer a fuzzy query. Pivert et al ([22]) handle the empty answer set (EAS) problem and the unsatisfactory answer set (UAS) problem, where there are either no answers at all or only answers with a low degree of satisfaction with respect to a user-defined threshold α. They efficiently determine minimal failing subqueries (MFS) of conjunctive queries defined by α-cuts; these MFS can give the user some explanation on why his query failed (that is, the answer set was empty or unsatisfactory). Although the notion of a threshold is common with our work, in [22] (sub-)queries below the threshold are totally disregarded, whereas our relevance threshold leads to more emphasis for answer tuples with some subqueries rated above the threshold (as opposed to lower ranks for answer tuples with all subqueries rated below it).

Vaneková and Vojtáš [23] provide an implementation of another fuzzy-set-based modeling of user preferences. They consider searching for data based on several attributes. A user expresses his search preferences as a membership function on the attribute domains.

In sum, most of the related works discussed above relies on gradual predicates or trapezoidal membership functions. In contrast, our approach relies on similarity defined on attribute domains. We argue that it might be equally difficult to define the membership degrees (see also the FITA/FATI problem discussed in [15]) for the fuzzy-set-based approaches as it is difficult to define the similarity values on attribute domains which our similarity-based approach relies on; hence no preference should be given to one or the other approach: while there are cases where user preferences can be easily modeled by fuzzy sets, in other cases it might be more natural to use similarities on domains. The paper [5] also provides a survey of related techniques (including similarity-based approaches) and argues how they are included in the fuzzy-set-based system. The similarity-based approaches surveyed there however do not take advantage of lattice-based fuzzy logic but rely more or less on metric calculations of distances. In this paper we showed how to put similarity-based flexible query answering on a sound logical foundation.

Preference queries – extensively studied by Chomicki (et al) [6, 7, 20] – are a further field of related work: a preference order $\prec$ must be predefined on attribute domains by a user; these attribute preference relations can then be combined into a tuple preference relation and this tuple preference relation can be used to return the most-preferred tuples upon a user query for example with the algebraic "winnow" operator. In particular, a special case of preference relations can be expressed by a scoring function f that is used to compare two constant values. The work in [24] extends preferences to work between sets of tuples. The advantage of preference orders is that they have some desirable properties (like transitivity: if A is preferred to B and B is preferred to C then A is also preferred to C). By iterating the winnow operator [6], a ranking of answer tuples (into best, second-best, ...) can be obtained; this is similar to our approach where the similarity degree of each answer tuple gives the user a fine-grained means of deciding if the tuple is relevant to him. A formal comparison between the properties of preference queries and the similarity-based approach we propose in this paper might be an interesting topic of future work.

Moreover, the user can specify dependencies between data – in particular, dependencies between different relations in the database. These dependencies can be given by logical rules and can be applied to the query to retrieve other answers that are relevant for the user under the background knowledge specified by the set of rules. This operation is known as "Goal Replacement" (e.g., [17]); it might be worthwhile to study its behavior in similarity-based query answering.

7 Conclusion

We applied similarities on attribute domains to rank tuples in answers to SPJ queries on relational databases. These ranks allow for flexible query answering as they have a comparative meaning and help users identify the answer tuples that match their intention best. Ranks are computed by i) replacing equality with similarities (Section 3), ii) emphasizing individual equality conditions with the residuated shift (Section 4), and iii) setting a global threshold to alter the size of the answer tables using ordinal sums of residuated lattices (Section 5). Our model has a strong theoretical background in the theory of ranked data tables where the classical bivalent structure is replaced by a structure with more than two truth values (that is, a lattice of ranks). Several hints towards future work have been given in Section 6. An efficient prototypical implementation of query answering in ranked data tables is under development at the Data Analysis and Modeling Laboratory of Palacky University; its performance will be closely analyzed in the future.

References

1. Belohlavek, R.: Fuzzy Relational Systems: Foundations and Principles. Kluwer Academic Publishers, Norwell, MA, USA (2002)
2. Belohlavek, R., Vychodil, V.: Data tables with similarity relations: Functional dependencies, complete rules and non-redundant bases. In: Database Systems for Advanced Applications, Lecture Notes in Computer Science, vol. 3882, pp. 644–658. Springer (2006)
3. Belohlavek, R., Vychodil, V.: Query systems in similarity-based databases: logical foundations, expressive power, and completeness. In: ACM Symposium on Applied Computing (SAC). pp. 1648–1655. ACM (2010)
4. Bosc, P., HadjAli, A., Pivert, O.: Incremental controlled relaxation of failing flexible queries. Journal of Intelligent Information Systems (JIIS) 33(3), 261–283 (2009)
5. Bosc, P., Pivert, O.: Fuzzy queries and relational databases. In: 9th ACM Symposium on Applied Computing (SAC). pp. 170–174 (1994)

6. Chomicki, J.: Preference formulas in relational queries. ACM Transactions on Database Systems 28(4), 427–466 (2003)
7. Chomicki, J.: Logical foundations of preference queries. IEEE Data Engineering Bulletin 34(2), 3–10 (2011)
8. Cignoli, R., Esteva, F., Godo, L., Torrens, A.: Basic fuzzy logic is the logic of continuous t-norms and their residua. Soft Computing - A Fusion of Foundations, Methodologies and Applications 4, 106–112 (2000)
9. Cintula, P., Hájek, P.: Triangular norm based predicate fuzzy logics. Fuzzy Sets and Systems 161, 311–346 (2010)
10. Codd, E.F.: A relational model of data for large shared data banks. Communications of the ACM 26, 64–69 (1983)
11. Esteva, F., Godo, L.: Monoidal t-norm based logic: towards a logic for left-continuous t-norms. Fuzzy Sets and Systems 124(3), 271–288 (2001)
12. Galindo, J. (ed.): Handbook of Research on Fuzzy Information Processing in Databases. IGI Global (2008)
13. Goguen, J.A.: The logic of inexact concepts. Synthese 19, 325–373 (1979)
14. HadjAli, A., Pivert, O.: Towards fuzzy query answering using fuzzy views - a graded-subsumption-based approach. In: 17th International Symposium on Foundations of Intelligent Systems (ISMIS). Lecture Notes in Computer Science, vol. 4994, pp. 268–277 (2008)
15. HadjAli, A., Pivert, O.: A fuzzy-rule-based approach to the handling of inferred fuzzy predicates in database queries. In: 9th International Conference on Flexible Query Answering Systems (FQAS). Lecture Notes in Computer Science, vol. 7022, pp. 448–459. Springer (2011)
16. Hájek, P.: Metamathematics of Fuzzy Logic. Kluwer Academic Publishers, Dordrecht, The Netherlands (1998)
17. Inoue, K., Wiese, L.: Generalizing conjunctive queries for informative answers. In: Flexible Query Answering Systems (FQAS). Lecture Notes in Computer Science, vol. 7022, pp. 1–12. Springer (2011)
18. Li, C., Chang, K.C.C., Ilyas, I.F., Song, S.: RankSQL: query algebra and optimization for relational top-k queries. In: Proc. 2005 ACM SIGMOD. pp. 131–142 (2005)
19. Maier, D.: The Theory of Relational Databases. Computer Science Press (1983)
20. Mindolin, D., Chomicki, J.: Contracting preference relations for database applications. Artificial Intelligence 175(7–8), 1092–1121 (2011)
21. Pivert, O., Bosc, P.: Fuzzy Preference Queries to Relational Databases. Imperial College Press (2012)
22. Pivert, O., Smits, G., HadjAli, A., Jaudoin, H.: Efficient detection of minimal failing subqueries in a fuzzy querying context. In: 15th International Conference on Advances in Databases and Information Systems (ADBIS). Lecture Notes in Computer Science, vol. 6909, pp. 243–256. Springer (2011)
23. Vaneková, V., Vojtáš, P.: Fuzziness as a model of user preference in semantic web search. In: European Society of Fuzzy Logic and Technology Conference (EUSFLAT). pp. 998–1003 (2009)
24. Zhang, X., Chomicki, J.: Preference queries over sets. In: 27th International Conference on Data Engineering (ICDE). pp. 1019–1030. IEEE Computer Society (2011)