Miroslav Hudec; Miljan Vučetić

Some issues of fuzzy querying in relational databases

Kybernetika, Vol. 51 (2015), No. 6, 994–1022

Persistent URL: http://dml.cz/dmlcz/144821

KYBERNETIKA — VOLUME *51* (2015), NUMBER 6, PAGES 994–1022

SOME ISSUES OF FUZZY QUERYING IN RELATIONAL DATABASES

Miroslav Hudec and Miljan Vučetić

Fuzzy logic has been used for flexible database querying for more than 30 years. This paper examines some of the issues of flexible querying which seem to have potential for further research and development from theoretical and practical points of view. More precisely, defining appropriate fuzzy sets for queries, calculating matching degrees for commutative and non-commutative query conditions, preferences, merging constraints and wishes, empty and overabundant answers, and views on practical realizations are discussed in this paper. Suggestions how to solve them and integrate into one compact solution are also outlined in this paper.

Keywords: membership functions, aggregation functions, preferences, commutative queries, non-commutative queries, empty and overabundant answers, application

Classification: 03E72, 68U35

1. INTRODUCTION

Each database tuple (database record) is characterized by values of attributes (columns) in one or several related tables. The goal of queries over databases is to separate relevant tuples from non-relevant ones, in other words to reveal them from a database for further use. The common way to realize such a query is to formulate a logical condition. According to the condition a relational database management system returns a list of tuples. In general, a logical condition consists of several atomic (elementary) conditions that define some constraints over domains of the attributes. However, from the users' point of view, knowledge about the entities represented in a database as well as users' preferences (expressed by conditions) are often vague or imprecise.

Fuzzy set theory [63] offers an environment to work with the uncertainty described by linguistic terms when sharply defined data selection criteria could not be created. Fuzzy set and fuzzy logic approaches in flexible querying have been applied in two main ways: querying databases containing crisp data and querying databases containing imprecise information. This paper is focused on the former; the latter one is out of the scope of this paper.

DOI: 10.14736/kyb-2015-6-0994

The main reason for using fuzzy set theory to make querying more flexible is discussed in e. g. [17]. Firstly, fuzzy sets provide a better description of the data requested by user. For example, the meaning of a query such as *find municipalities with high unemployment and altitude about 500 meters above sea level* can be understood at first glance. Linguistic terms clearly suggest that there is a smooth transition between acceptable and unacceptable tuples. As a result, some database tuples are a definite match, some definitely do not match the user's request, and some match to a certain degree. Several fuzzy query implementations have been proposed such as [13, 28, 31, 40, 41, 46] and [55]. A consistent survey of the status and opportunities of fuzzy sets in all areas of databases and information systems including fuzzy queries can be found in [6]. A detailed description of fuzzy queries can be found in [44] where authors examined queries on crisp and fuzzy databases. In addition, authors gave a brief description of relational databases and calculus.

The goal of this paper is to offer an overview and discuss some issues of flexible querying related to crisp relational databases. Section 2 briefly presents basic concepts of fuzzy queries and gives a list of items discussed in this paper. Following sections are devoted to each of these items respectively. Finally, some conclusions are drawn in Section 8.

2. PRELIMINARIES TO FLEXIBLE DATABASE QUERIES

A relational database is a collection of relations (often called tables). The relation schema has the following form [44]:

$$R(A_1 : D_1, \ldots, A_n : D_n) \tag{1}$$

where R is the name of relation e. g. customer or municipality, A_i is the i–th attribute $(i = 1, \ldots, n)$, often called column (e. g. *number of days with snow coverage*) and D_i is the domain of attribute A_i defining a set of possible values (e. g. interval [0, 365] of integers for the above mentioned attribute). The crucial element in a relation is the primary key i. e. attribute(s) which unambiguously identify tuple. A relation instance of given relation schema is a set of tuples [44] stored in a table inside a database. Each tuple $t_j (j = 1, \ldots, m)$ consists of values of the attributes in the following way:

$$t_j = \{(d_{1j}, \ldots, d_{nj}) \mid (d_{1j} \in D_1, \ldots, d_{nj} \in D_n)\} \tag{2}$$

where d_{ij} is the value of the tuple t_j for attribute A_i. For the sake of simplicity, usually the term "relation instance" is abbreviated to relation.

The Structured Query Language (SQL) is a standard query language for relational databases. SQL has the following basic structure:

select [distinct] ⟨*attributes*⟩ from ⟨*relations*⟩ where ⟨*condition*⟩.

The query returns a relation (set of tuples that satisfy the condition). Although SQL is a powerful tool, it is unable to satisfy requirements for data selection based on imprecise conditions (linguistic terms) and provide degrees of matching query conditions. A brief overview of SQL, its limitations and ways for improving it using fuzzy forms of

the *where* and *having* clause can be found in [12, 15]. Other authors also advocate the advantages of using linguistic terms in queries e. g. [36].

Similarly to SQL, the basic structure of fuzzy query is the following [12]:

select [distinct] $\langle attributes \rangle$ from $\langle relations \rangle$ where $\langle fuzzy\ condition \rangle$.

The fuzzy query returns a fuzzy relation consisting of set of tuples that satisfy the fuzzy condition and matching degree to condition for each tuple t. The set of answers to fuzzy query Q_f could be written in the following way:

$$A_{Q_f} = \{(t, \mu(t)) \mid t \in R \wedge \mu(t) > 0\} \tag{3}$$

where $\mu(t)$ indicates how well the selected tuple t satisfies the query. If $\mu(t) = 1$ the tuple fully satisfies the query and value of $\mu(t)$ in the interval (0, 1) meaning that the tuple t partially satisfies the query.

Because fuzzy query returns tuples as well as their satisfaction degree, it is possible to apply an additional clause: a threshold to restrict the list to only tuples which significantly meet the query condition or first k tuples ($k \in \mathbb{N}$) according to the matching degrees.

Presumably, the first attempt to deal with flexible SQL-like queries is [49]. It fuzzifies the *where* clause and makes it possible to use vague terms similar to natural language. One of the first practical realizations of flexible queries is FQUERY, an add-in that extends the MS Access's querying capabilities with linguistic terms in the *where* clause [31]. The SQLf [12, 13] is a more comprehensive fuzzy extension of SQL queries. SQLf extends SQL by incorporating fuzzy predicates not only in the *where* clause but also wherever it makes sense [44]. For example, SQLf supports, among others, subqueries inside the fuzzified *where* clause and fuzzy *joins*. The Fuzzy Query Language (FQL) [55] extends SQL queries with fuzzy condition inside the *where* clause in the usual way and adds another two clauses which provide additional functionality: *weight* and *threshold*. The fuzzification of *group by* clause is examined in [9].

To clarify the understanding of queries, let us show one illustrative example. The relation municipality (#id, name, number_of_inhabitants, area, altitude) is represented as

#id	name	number_of_inhabitants	area	altitude
1	Mun 1	550	2500	123
2	Mun 2	790	7930	856
3	Mun 3	810	8030	354

where #id represents the primary key.

The query *find municipalities (attributes: name, area and altitude) which have area greater than 8000* m^2 is expressed in the following way:

select name, area, altitude from municipality where area > 8000.

The query delivers relation:

name	area	altitude
Mun 1	8030	354

It is possible to find municipalities which have high area through a fuzzy query expressed in the following way [12]:

select name, area, altitude from municipality where area is high.

The answer depends on the definition of the linguistic term *high*. The query could deliver the following fuzzy relation:

name	area	altitude	membership degree
Mun 2	7930	856	0.91
Mun 3	8030	354	1

In addition, flexible querying has been employed in many areas e. g. evaluation of the truth value of short sentences expressed by linguistic summaries. Linguistic summaries are of structure: *Q objects in database have S* [41, 60] where Q is the fuzzy quantifier in sense of Zadeh [62] and S is the summarizer expressed by linguistic terms.

In this paper the following aspects of flexible queries are examined:

- the construction of appropriate fuzzy sets for each query;
- the calculation of matching degrees considering commutative aggregation, non-commutative aggregation, preferences and quantified propositions;
- dealing with empty and overabundant answers;
- some of the effects of fuzzy logic in flexible querying;
- practical realizations, mainly creating a user-friendly interface and fuzzy query evaluation.

A list of main issues relevant for fuzzy query language realisation is examined in [34]:

- syntax;
- representation of linguistic terms;
- semantics, mainly the question of matching degree calculation;
- efficiency of application.

These issues have also been taken into account in this paper.

3. CONSTRUCTING APPROPRIATE FUZZY SETS FOR QUERIES

The matching degree of each database record to a query condition critically depends on the constructed membership functions. Therefore, these functions should be carefully constructed.

Let $D_{\min}$ and $D_{\max}$ be the lowest and the highest domain values of numeric attribute A i. e. $Dom(A) = [D_{\min}, D_{\max}]$ and L and H be the lowest and the highest values in the current content of a database respectively [29]. Usually the attribute's domain is defined in a way that all theoretically possible values can be stored. In practice, collected data

can be far from the values of $D_{\min}$ and $D_{\max}$; that is, $[L, H] \subseteq [D_{\min}, D_{\max}]$. This means that only part of the domain contains data. This fact should be considered in defining not only query conditions but also flexible rules and linguistic summaries among others. Let us consider the attribute describing the daily frequency of a measured phenomenon during the year (e. g. *number of days with snow coverage*). The domain is the [0, 365] interval of integers. However, in practice collected values could be far from the $D_{\min}$ and $D_{\max}$ values. If the value of H is 199, then each query with condition > 200 will returns an empty result.

Generally, there are two main aspects for constructing fuzzy sets for queries. In the first aspect, users define parameters of each fuzzy set in order to create flexible boundaries in query condition [28]. This way gives users freedom to choose attributes of fuzzy sets (L_d, L_p, L_q and L_g) depicted in Figure 1. Nevertheless, in this approach users are asked to set crisp values in flexible query condition (e. g. two crisp values to clarify meaning of the term *high distance* – Figure 1a, whereas in classical query user assign only one value e. g. *distance* > 500). It could be treated in some sense as not fully flexible querying. This problem could be mitigated by adjusting fuzzy sets parameters by moving sliders to define ideal and acceptable values [46].

Users usually consider their preferences on whole domains of attributes. If they are not familiar with the current database content, the query might easily end up as empty or overabundant. Empty and overabundant answers are examined in Section 5.

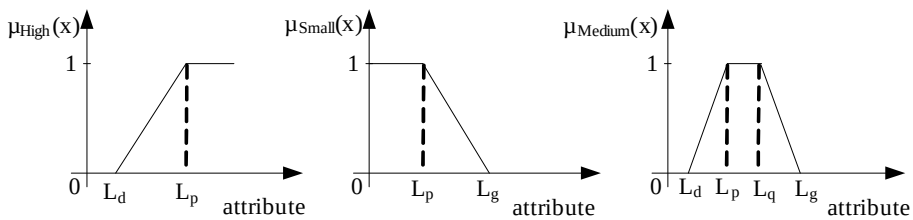

Fig. 1. Fuzzy sets for flexible querying.

The second aspect deals with dynamic modeling of fuzzy sets parameters over the domain of attribute. In the first step values of L and H are retrieved from a database. These parameters are used to create fuzzy sets e. g. *small*, *medium* and *high* over the attribute's domain directly from the current content of collected data in a database.

If collected data is more or less uniformly distributed in the domain then the uniform domain covering method [51] could be applied. In this case the parameters of fuzzy sets (Figure 2) are created in the following way:

$$\alpha = \frac{1}{8}(H - L) \tag{4}$$

$$\beta = \frac{1}{4}(H - L). \tag{5}$$

Consequently, it is easy to calculate required parameters A, B, C and D (Figure 2). If it is a requirement for more fuzzy sets (e. g. five sets: very small, small, medium,

high, very high) these sets can be straightforwardly constructed adjusting parameters of α and β.

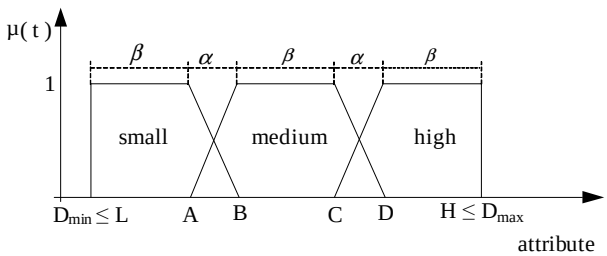

Fig. 2. Linguistic values uniformly distributed over an attribute domain.

In cases like municipal statistics databases [25], the majority values of several attributes are often close to the value of L, whereas only a few are close to the value of H and therefore, belong to the fuzzy set *high*. Illustrative examples are attributes such as population density, length of roads, area, etc. where only a few big entities have high values of these attributes. This kind of data distribution cannot always be properly evaluated by the uniform domain covering method. The logarithmic transformation [29] is a rational option which might provide a solution. The interval $[L, H]$ is transformed into the interval $[\log(L), \log(H)]$. Consequently, in this interval, logarithms of α, β and A, B, C and D are calculated using equations (4) and (5). Finally, obtained values are delogarithmised into the real values. Another possible solution is adding the statistical mean into the construction of fuzzy sets [51]. The middle of the medium fuzzy set core is the statistical mean of attribute. In this approach, cores of all three fuzzy sets (β) have equal size; lengths of fuzzy sets slopes (α) are different.

In order to choose an appropriate method users could rely on their knowledge, common sense, attainments about examined entities (e. g. municipalities or customers), and knowledge of physical laws. In the case of attributes like the number of inhabitants, in which a database usually contains a few big municipalities and many smaller ones the choice is the logarithmic transformation or statistical mean based algorithm. In the case of attributes like gas consumption per inhabitant, users could assume that the uniform domain covering method is an appropriate option.

Actually, an improved solution can be obtained by merging both aspects. In the first step parameters of fuzzy sets are calculated from the current content of a database. In the second step users can modify these parameters if they are not satisfied with the suggested ones. Evidently, the user can obtain information about stored data before running a query. The querying process could become more tedious but on the other hand, it might reduce the empty answer problem. In this case, user has two options (i) to quit the query process knowing that there is no data that satisfy the query condition (it saves computational resources and time) or (ii) to accept the suggested parameters. In any case, the assistance for constructing fuzzy sets parameters should be optional for users.

In situation where user wants to retrieve the best possible answer(s) of flexible conditions and is not always aware of the current content in a database, Bosc et al. [14] suggests the definition of predicates in a contextual and relative manner related to data. Moreover, authors discussed issues of efficiently retrieving the top-k answers.

In the construction of fuzzy sets we should consider complex criteria when elementary conditions are not independent or non-commutative. The qualification of linguistic term relative to another elementary gradual condition requires dynamically defining dependable linguistic terms [50]. This topic is discussed later in Section 4.4.

In [36] another inspiration for fuzzy set creation is found. Intervals of each database attribute are transformed into the interval [-10, 10]. Inside this interval users can assign parameters of fuzzy sets in order to support fuzzy relations or queries such as *unemployment rate is much lower than migration*. In the case of queries like *altitude above sea level is small and number of inhabitants is high*, attribute values can be also transformed into the [-10, 10] interval. In this case, fuzzy sets *small*, *medium*, *high* can be created using $\alpha = 3.3$.

Flexible querying systems could be broadly divided into two categories: those for general use and those devoted to a specific purpose. In the latter category predefined selection attributes and fuzzy sets should be available in advance and the user can specify only his/her preferences inside them.

The topic of fuzzification is covered by vast literature, because the fuzzufication is part of applications in many areas, mainly in fuzzy systems. The review of ways for constructing membership functions as well as various interpretations can be found in e. g. [2] where authors discussed various aspects such as taking into account objectivity, subjectivity, group and individual opinions about membership functions. Verkuilen [53] has identified three general ways for constructing fuzzy sets: direct assignment, indirect assignment and assignment by transformation. In this section we have shown direct assignment by users input, mining from data and transformation of the domain into unified intervals. Construction of fuzzy sets for fuzzy systems is further discussed in [38, 45]. The shapes of membership functions have adopted several conventions [19]. Generally, the membership functions are normalized, convex, distinct, piecewise linear functions. These types constitute only a small subset of the all possible shapes of membership functions. Discussion about other possible shapes can be found in [19].

However, we should be careful when consider constructing fuzzy sets in queries due to the following reasons:

- In fuzzy systems rules and fuzzy sets should cover whole domains of attributes in a proper way by the family of fuzzy sets in order to properly classify or control all possible occurrences of input attributes. On the other hand, queries only select a subset of data which might be relevant for users.
- If we want to develop an easy to use tool e. g. a website for broad audience and less demanding for the calculation, then fuzzification should be as simple as possible. Therefore, sophisticated approaches (even supported by neural networks or genetic algorithms) are not suitable.
- In non-commutative queries where answer to the first part of query influences adjusting condition of the second part of query we have to rapidly and efficiently fuzzify second attribute in the middle of the querying process.

The fuzzification in fuzzy queries can be improved by mining parameters from the recorded history in two ways. If a user is registered then the history of preferences can be stored. During the next attempt for similar query, the parameters of fuzzy sets from recorded history can be offered. The second option is recommending parameters according to the parameters used by the most similar users. The main drawback of this is keeping history and building recommendations. On the other hand, this option is more tailored to users. In any case, fuzzification is an interesting topic for further investigation and finding suitable approaches.

The attention dedicated should be also given to quantified queries where attributes influence each other e. g. *find regions where most of low polluted municipalities have high unemployment*. This kind of queries is discussed in Section 4.5.

4. CALCULATION OF MATCHING DEGREE

Query conditions usually consist of more than one elementary condition. Generally, we can divide this topic into several categories. The simplest category is aggregation of the classical *and* and *or* operators. Elementary query conditions have the same importance and are independent i. e. order of the execution is irrelevant (commutative queries). The next category is queries bearing different importance for elementary conditions. In this category commutative and non-commutative queries can be found. Commutative queries can be solved by fuzzy implications or Ordered Weighted Averaging Operator (OWA).

A non-commutative query could bear different structure. Bipolar queries i. e. merging positive and negative judgements in a query is nowadays a popular research topic. The bipolar query consists of constraints (have to be satisfied) and wishes (could be satisfied). An example of such a query is A_1 *is high and possibly* A_2 *is cheap*. Another example are non-commutative queries containing only constraints where answer of the first elementary condition influences answer to the second one e. g. *small values of* A_1 *among high values of* A_2. In the next several subsections these categories are examined. In order to keep the naming of variables consistent throughout the paper, letter A is related to the attribute and letter P to the predicate (e. g. *P: A is small*).

4.1. Fuzzification of *and* operator

This is the simplest form of aggregation for solving queries like *select municipalities where pollution is high and unemployment is low*. The t-norm functions are used as *and* logical operator. More about t-norms can be found in e. g. [37]. The property of the associativity [38] ensures that all t-norm functions can be extended to n attributes. Actually, it is not easy to use all t-norm functions when we have higher number of elementary conditions. An example of computational burden for one aggregation function in case of just three attributes can be found in [45]. Due to this reason, and the practical realizations (which is also mentioned in Section 7) the following t-norm functions can be easily aggregated [45] and therefore could be suggested for fuzzy queries:

- minimum

$$\mu(t) = \min(\mu_{Ai}(t)), i = 1, \ldots, n\ ; \tag{6}$$

- product

$$\mu(t) = \Pi_{i=1}^{n}(\mu_{Ai}(t)) \; ; \tag{7}$$

- Lukasiewicz

$$\mu(t) = \max\left(0, \sum_{i=1}^{n} \mu_{Ai}(t) - (n-1)\right) \; ; \tag{8}$$

where $\mu_{Ai}(t)$ denotes the membership degree of the value of attribute A_i to the i–th fuzzy set for tuple t.

From the above mentioned t-norms only the minimum t-norm (6) is an idempotent t-norm [38] which makes it the most acceptable for users accustomed to working with the crisp queries. This is one of the reasons for the wide use of this operator in many approaches e. g. [49] or [55]. However, min t-norm has a limitation which is examined in Table 1. The product t-norm (7) takes into account all membership degrees and balances the query satisfaction degree across each of the elementary conditions inside the overall query condition. The product t-norm distinguishes records which have the same value of the lowest membership degree of elementary conditions and different values of the satisfaction degree of other elementary conditions. The Lukasiewicz t-norm (8) is a nilpotent t-norm, that is, $\mu(t) > 0$ only for records which significantly satisfy the condition in a way:

$$\sum_{i=1}^{n} \mu_{Ai}(t) - (n-1) > 0.$$

It is not easy to offer an answer to the question which of these t-norms is the most appropriate. Let us have two records which satisfy the first elementary condition (P_1) and the second elementary condition (P_2) as is shown in Table 1.

tuple	P_1	P_2	min(6)	prod(7)	luk(8)	$\gamma = 0.5$ (9)
1	0.11	0.2	0.11	0.02	0	0.08
2	0.10	0.9	0.10	0.09	0	0.29

Tab. 1. Example of matching degrees using t-norms.

The min-t-norm (6) prefers the *tuple 1*. But, this contradicts the human reasoning when selecting the best tuple: although the *tuple 1* is only slightly better in the first elementary condition and notably worse according to the second elementary condition, it is the preferred solution. The product t-norm prefers the second tuple with the membership degree lower than the values for both elementary conditions. The Lukasiewicz t-norm calculates membership degrees of 0 for both tuples because they do not significantly satisfy both elementary conditions.

This issue can be solved using compensatory operators to model the fuzzy or linguistic *and* operator. The compensation of a bad value of one attribute by a good value of another attribute can be achieved e. g. by the γ–operator [69] adapted to the fuzzy queries in the following way:

$$\mu(t) = (\Pi_{i=1}^{n}\mu_{Ai}(t))^{1-\gamma}(1 - \Pi_{i=1}^{n}(1 - \mu_{Ai}(t)))^{\gamma} \tag{9}$$

where $\gamma \in [0,1]$, other elements have the same meaning as in (6) – (8). Applying the γ–operator with the value of 0.5 implies that all attributes have the same weight. Furthermore, neither largest nor smallest membership degree is the solution (like in min and max operators respectively). A short discussion of the applicability of γ–operator can be found in [56].

The γ-operator requires more computational time in comparison with the examined t-norm functions (6) – (8). When the most restrictive matching degree of elementary condition is relevant then the min t-norm is the appropriate solution. When users consider satisfaction degrees of each elementary condition then the product t-norm is the solution. In addition, if user would like to see in the result only tuples which significantly meet all elementary conditions then the Lukasiewicz t-norm is suitable. However, product t-norm and Lukasiewicz t-norm produces satisfaction degree significantly lower than the degrees of elementary conditions.

4.2. Fuzzy preferences between elementary query conditions

Membership functions expressing preferences inside elementary query conditions are discussed in Section 3. The aim of preferences between elementary conditions is to distinguish elementary conditions according to their importance. Not all conditions always have the same priority for users. An example of such a query condition is: *select municipality with high altitude above sea level and low pollution where the second condition is more important than the first one.* In order to achieve the mentioned distinction weights $w_i \in [0,1]$ can be associated with elementary conditions. Two types of weights can be applied [66]: static and dynamic. Static weights are fixed and known in advance. These weights are independent of the current values of attributes in a database and cannot be changed during query processing. In the case of dynamic weights neither weights nor their association to criteria are known in advance.

The idea for how to calculate the matching degree of elementary conditions P_i according to an importance weight w_i has the following form [66]:

$$\mu(P_i^*, t) = (w_i \Rightarrow \mu(P_i, t)) \tag{10}$$

where $\Rightarrow$ is a fuzzy implication. A broader discussion of existing fuzzy implications can be found in [22, 57, 61]. In order to be meaningful, weights should satisfy several requirements [66]. One of them: *if $w_i = 0$ then the result should be such as if P_i does not exist* is briefly examined below. By applying this requirement, it is easy to conclude that the regular implications e. g. Kleene–Dienes, Gödel and Goguen match this requirement.

Using the Kleene–Dienes implication the following query condition for the conjunction is achieved:

$$\mu(t) = \min_{i=1,\ldots,n} \Big(\max_{i=1,\ldots,n} (\mu(P_i, t), 1 - w_i) \Big) \tag{11}$$

if the minimum function is used as a t-norm. The equation (11) corresponds to the definition of the weighted conjunction operator introduced in [18]. Therefore, we could apply fuzzy implications or weighted minimum and maximum operators. The paper focuses on the former case. For a small importance of P_i (w_i is close or equal to 0), the satisfaction of elementary condition P_i does not have an influence on the query satisfaction: $w_i \to 0 \Rightarrow \mu(P_i^*, t) \to 1$, where arrow means approaching the value of. In

another case when w_i is closer to 1, the satisfaction of P_i is essential for satisfaction of the overall query ($w_i \to 1 \Rightarrow \mu(P_i^*, t) \to \mu(P_i, t)$). Moreover, this implication has continuous values of query satisfaction from 0 to 1.

Besides these implication functions, in some applications it is common to describe implication by t-norms [22]. This especially holds for the minimum t-norm (6) which is called the Mamdani implication. Let us see whether this assumption holds for so-called Mamdani implication in the query preferences. The query condition is achieved in the following way:

$$\mu(t) = \min_{i=1,\ldots,n} \Big(\min_{i=1,\ldots,n} (\mu(P_i, t), w_i) \Big). \tag{12}$$

We can easily prove that this implication is not suitable. For the small importance of w_i the overall query satisfaction will be close to 0. In the case when $w_i = 0$, the overall query satisfaction is 0 regardless of other elementary conditions. It implies that the requirement *if* $w_i = 0$ *then the result should be such as if* P_i *does not exist* is not satisfied for the "implication"(12).

Another way for realisation of preferences can be found in [55] where the authors suggest not only crisp values but also fuzzy sets to describe the importance value in the *weight* clause. Weights are realised as a separated query clause. The importance weight can be crisp numbers from the [0, 1] interval or fuzzy sets defined beforehand in a separate table. Users benefit through the possibility to select the importance defined by linguistic terms.

The second approach capable to take into account the importance of conditions is based on the OWA operator [66]. The OWA operator [59] is defined by a weight vector $W = [w_1, \ldots, w_n]$, $\sum_{i=1}^{n} w_i = 1$ and for a vector of satisfaction degrees of elementary conditions $P = [\mu(a_i), \ldots, \mu(a_n)]$ yields the following result

$$O_w(\mu(a_1), \ldots, \mu(a_n)) = \sum_{i=1}^{n} w_i b_i \tag{13}$$

where b_i is the i–th largest element among $\mu(a_i)$ elements. In this way variety of aggregation operators could be generalized. In particular for $W = [0, \ldots, 0, 1]$ the minimum t-norm is produced. If the value of $1/n$ is set for each weight then we get the arithmetic mean in which all attributes are equally important. This operator is very useful because of its versatility. It provides parameterized family of aggregation operators. However, we cannot say in advance which elementary condition is the most important because arguments are sorted according to their membership degrees.

The OWA operator is usually focused on aggregation with crisp weights. If one attribute is more important than another one then it is expressed by higher value of weight. On the other hand, people tend to describe weights as imprecise values (A_1 is a bit more important than A_2, A_3 is the most important, etc). In order to cope with this issue Zhou et al. [68] have suggested *type-1* OWA operator to aggregate uncertain weights by the OWA mechanism. Weights are expressed as *type-1* fuzzy sets. More about this approach can be found in [67].

Modeling preferences by implications (10) and by OWA (13) are not competitive but complementary. If we declare in advance which condition is the most important then implications are more suitable otherwise the OWA is suitable.

4.3. Bipolar queries

When people express their requirements they could have in mind negative (constraints) and positive (wishes) preferences. A suitable example is *find hotel which has low price and possibly short distance to point M*. The budget is the limitation; we just cannot afford something beyond our budget. The distance is a wish, we would prefer shorter walk if possible. We can formally write query as [65]: *find tuples satisfying N and if possible P* where N denotes negative preference and P denotes positive preference. Answer to a bipolar query is written in the following way:

$$A_{Q_{fbp}} = \{t \mid N(t) \text{ and possibly } P(t)\}. \tag{14}$$

In the classical approach the condition of bipolar query for tuple t is expressed as [39]

$$N(t) \text{ and possibly } P(t) = N(t) \wedge \exists s(N(s) \wedge P(s)) \Rightarrow P(t). \tag{15}$$

In the first step tuples satisfying N are selected from a database. This step ensures that tuples which do not satisfy N are not considered. If no tuple meet N then answer to bipolar query is empty. In the second step tuple t is preferred if no other tuples satisfies P or tuple t satisfies P. The approach: first select using N then order using P cannot be directly applied when satisfaction is a matter of degree [16, 65].

Formula (15) corresponds to the "global" interpretation of the term *possible* (checking whether the constraint is satisfied by at least one tuple from the dataset considered). On the other hand a "local" interpretation can also be considered. Then satisfying the constraint gives a benefit to the tuple, but there is no need to check the other tuples from the dataset to assess a particular tuple. Therefore, tuples are analyzed independently. An example of the "local" interpretation, which however cannot handle bipolarity, is (17).

The condition of bipolar query (15) is expressed in fuzzy terms in the following way [58, 65]:

$$A_{(N,P,T)} = \min\left(N(t),\ \max\left(1 - \max_{s \in T} \min\left(N(s) \wedge P(s)\right),\ P(t)\right)\right) \tag{16}$$

where (N, P, T) means answer of N and P against set of tuples T. Quantifier $\exists$ is modelled by the maximum operator. The implication is characterized by the Kleene–Dienes implication. Min t-norm, max t-conorm and standard negation form a triplet such that min t-norm is dual to max t-conorm when we apply standard negation. Other triplets like (product, probabilistic sum, standard negation) or (Lukasiewicz t-norm and t-conorm and standard negation) can also be applied if reinforcement effect is needed. Influences of different functions for quantifier, implication, t-norm and t-conorm to solution are discussed in [65].

It is evident that bipolar queries are not commutative. Query *short distance and if possible low price* has different semantic meaning (budget is not a problem but distance is).

Let us have four tuples satisfying N and P with different degrees. Table 2 depicts answers obtained by *and if possible* operator (16).

N	P	Eq. (16)
0.8	0.3	0.5
0.8	0.5	0.7
0.1	0.9	0.1
0.2	0.7	0.2

Tab. 2. Answer by *and if possible* operator.

The most suitable tuple is tuple satisfying N with high degree (0.8) and higher value of P in comparison with other tuples. Low satisfaction degree of N is directly reflected in matching degree.

Construction of fuzzy sets for P and N could be modelled as independent i. e. construction of fuzzy sets for N do not have influence on construction of fuzzy sets for P (in our example low price for N and short distance for P).

Bosc and Pivert [11] have analyzed four non-commutative operators. One of them is P_1 *and if possible* P_2. At first glance, when we consider P_1 as N and P_2 as P then it is bipolar query expressed above. Bosc and Pivert found appropriate function for this non-commutative operator. A function which meets six required axioms [11] is:

$$\alpha(\mu_{P1}, \mu_{P2}) = \min(\mu_{P1}, k\mu_{P1} + (1-k)\mu_{P2}), \quad k \in [0,1]. \tag{17}$$

When $k = 1$ only P_1 is relevant which is in line with bipolar queries. But when $k = 0$ it becomes ordinal minimum operator. For $k = 0.5$ *and if possible operator* has the following structure

$$\alpha(\mu_{P1}, \mu_{P2}) = \min\left(\mu_{P1}, \frac{\mu_{P1} + \mu_{P2}}{2}\right). \tag{18}$$

Bosc and Pivert [11] argued that this operator (although has similar name like operator suggested by [65]) cannot handle bipolarity, because α does not keep both P_1 and P_2 separate. Operator (17) is suitable for more and less relevant constraints in the query. Fuzzy sets for P_1 and P_2 do not necessary influence each other. Fuzzy set for P_1 is constructed on the domain of attribute included in P_1 and fuzzy set for P_2 is constructed on the domain of attribute included in P_2. In the Section 4.4 non-commutative query where construction of fuzzy sets for one part of a query is highly dependent on the answer from another part of the query is demonstrated.

The second approach for managing bipolarity is based on possibility theory [16]. The answer is measured on bipolar scales either on one scale where the middle point is neutral and ends bear extreme positive or extreme negative values. In case of two scales, one scale measure positive and another one negative preferences. This is supported by the possibility function and its dual necessity function. Asymmetric bipolarity deals with two unrelated information [16]. For tuple which is not forbidden (satisfies N) does not means that it is explicitly possible. More about this approach and related discussion can be found in [16].

The third approach for bipolar queries is based on the lexicographic ordering [10]. In this approach degrees for N and P are evaluated separately, i. e. no aggregation between the constraint and the wish is performed. Let us $(N(t_1), P(t_1))$ and $(N(t_2), P(t_2))$ denote

the satisfaction degrees of t_1 and t_2 with respect to the negative preference N and the positive preference P respectively, then t_1 is preferred against t_2 if:

$$t_1 \succ t_2 \Leftrightarrow (N(t_1) > N(t_2)) \text{ or } (N(t_1) = N(t_2) \text{ and } P(t_1) > P(t_2)). \quad (19)$$

In this way, the satisfactions of the constraint and the wish rank the database tuples with a priority given to the constraint. A tuple which is beaten on the constraint cannot win even if it is notably better on the wish [10]. A tuple which is equal on the constraint wins, if it is even slightly better on the wish.

4.4. Answer to first elementary condition influences answer to the second one

This is an occurrence of queries where the condition P_2 (elementary or consisting of several elementary ones) is defined a priori and the condition P_1 is defined in a relative manner of satisfying the condition P_2. One example for such a query is *find entities having small values of* A_1 *among entities having high values of* A_2. From the axiom that all t-norm functions are commutative [38] implies that the order of elementary conditions in overall query condition is irrelevant. In this class of problems elementary conditions are not independent, that is, the second elementary condition depends on answers obtained from the first one. We should realize first elementary condition and depending on the result continue with second elementary condition and so on. Number of elementary conditions is not limited. The *among* operator [51] meets this requirement having the following structure:

$$\mu_{P_1 among P_2} = \min(\mu_{P_1/P_2}(t), \mu_{P_2}(t)) \quad (20)$$

where μ_{P_2} is the membership function defining fulfillment of the independent elementary condition and μ_{P_1/P_2} is the fulfillment degree of the dependent elementary condition relative to the independent one.

The example of such a query is: *select municipalities having small migration* (P_1) *among municipalities having high unemployment rate* (P_2). The solution applying the *among* operator and solving task by one query condition is discussed in [51].

We would like to discuss another option. The query could be realised as a two step query. In the first step entities satisfying the condition P_2 are selected. The membership function for the linguistic term of the elementary condition P_2 could be calculated by one of the approaches examined in Section 3 by applying the interval $[L_{P2}, H_{P2}]$ from the current content in a database. Entities selected by P_2 create a fuzzy subrelation of all entities. Applying function for sorting entities according to values of attribute P_1 in our example we obtain a reduced interval $[L_{P1-red}, H_{P1-red}] \subseteq [L_{P1}, H_{P1}]$ of the dependent attribute P_1. In the next step the fuzzy set describing P_1 is created on the subdomain $[L_{P1-red}, H_{P1-red}]$ by the uniform domain covering method [51]. Even if the user can define parameters for the membership function μ_{P_2} without suggestion from current database content, defining the membership function for μ_{P_1/P_2} on the interval $[L_{P1-red}, H_{P1-red}]$ depends on the selected tuples in the first step. Finally, the overall query matching degree is calculated. If the order of elementary conditions is permuted to the following query *select municipalities having high unemployment rate* (P_2) *among*

municipalities having small migration (P_1) the solution is different. It is not a surprising result because, in the first step of permuted elementary conditions only tuples having small migration are selected. This condition deliminates a subrelation. In the second step the limited subdomain of the unemployment rate has been used as a basis for constructing fuzzy set *high unemployment*.

The suggested approach is an associative one, that is, it is not limited to queries containing two elementary conditions.

4.5. Quantified queries

This is a class of database queries which uses linguistic quantifiers to represent amount of tuples satisfying a condition. An example of such a condition is *most of regions have medium unemployment*. These sentences (called Linguistic Summaries (LSs)) have been initially introduced in [60] and developed in tool called SummarySQL [41]. LSs were topic of many papers e. g. [24, 30, 35] and [64] .

LSs could be used for mining abstract from data (linguistic summaries from predefined linguistic term sets which meet majority of tuples) or as nested subqueries, especially for hierarchical data structures like municipalities – regions or customers – orders.

An elementary LS is of the structure *Q entities in database are (have) S*, where Q is the quantifier (almost all, most of, about half, few, etc.) and S is a summarizer expressed by fuzzy condition (e. g. unemployment is small) or crisp conditions (e. g. unemployment < 5 %). The validity (truth value) is computed in the following way [64]:

$$v(LS) = \mu_Q\Big(\frac{1}{n}\sum_{i=1}^{n}\mu_S(t_i)\Big) \tag{21}$$

where n is the cardinality of a relation in a database (number of tuples), $\frac{1}{n}\sum_{i=1}^{n}\mu_S(t_i)$ is the proportion of tuples in a database that satisfy the S and μ_Q is the membership function of chosen quantifier. The validity gets value form the unit interval.

Summarizer can concerns more than one atomic condition joined by the *and* connectives [20]. If summarizer consists of several atomic conditions connected by conjunction $\mu_S(t_i)$ is calculated in the following way [24]:

$$\mu_S(t_i) = f(\mu_{Sj}(t_i)), \quad j = 1 \dots m, \tag{22}$$

where f is a t-norm and m is number of atomic conditions. Selecting appropriate t-norm function is explained in Section 4.1.

A more complex type of LS has the form *Q R entities in database are (have) S* where R is a restriction focusing on a part of database relevant for the summarization task. An example of such a LS is *most high polluted and small sized municipalities have high number of respiratory diseases*. The procedure for calculating validity has the following form [41]:

$$v(LS) = \mu_Q\Big(\frac{\sum_{i=1}^{n} f(\mu_S(t_i), \mu_R(t_i))\Big)}{\sum_{i=1}^{n}\mu_R(t_i)}) \tag{23}$$

where $\frac{\sum_{i=1}^{n} f(\mu_S(t_i), \mu_R(t_i))}{\sum_{i=1}^{n} \mu_R(t_i)}$ is the proportion of the tuples in a database that satisfy S and belong to R, f is a t-norm and μ_Q is the membership function of quantifier.

The validity of a summary is computed by the chosen quantifier [21, 30]. Relative quantifiers *few*, *about half*, *most of* and *almost all* and possible values of their respective parameters are shown in Figure 3. The universe of discourse X of these quantifiers is the unit interval. LSs can be also created by absolute quantifiers such as *about 100 tuples*, *more than 300 tuples*.

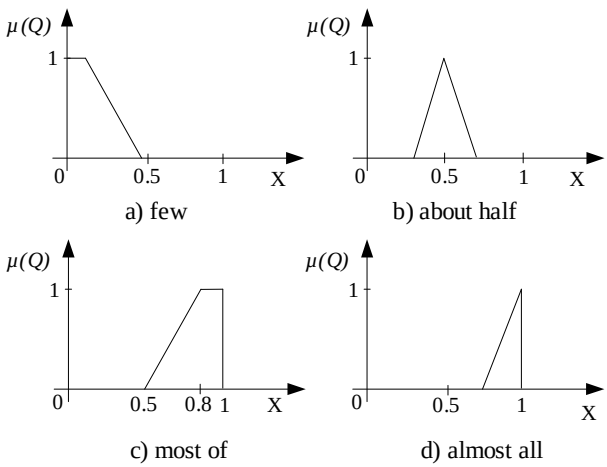

Fig. 3. Relative quantifiers.

The suitable option is offering the possibility for users to adjust parameters of quantifiers e. g. by sliders in the same way as parameters of fuzzy conditions in queries [46].

An example of quantified query is the following *select regions where most of municipalities have small water consumption per inhabitant*. In the first step, the validity of summaries is calculated for each region. In the second step regions are ranked downwards starting with region having the highest value of validity. The procedure for calculating validity is created as the extension of (21) [24]:

$$v_j(LS) = \mu_Q\Big(\frac{1}{N_j}\sum_{i=1}^{N_j}\mu_S(t_{ij})\Big), \quad j = 1 \ldots C, \quad \sum_{j=1}^{C} N_j = n, \tag{24}$$

where n is the number of entities in the whole database, N_j is the number of entities in cluster j (e. g. municipalities in region j), C is the number of clusters in a database (e. g. regions), v_j is validity of LS for j–th cluster, and $\frac{1}{N_j}\sum_{i=1}^{N_j}\mu_S(t_{ij})$ is the proportion of tuples in j–th cluster that satisfy summarizer S.

When LS contains the restriction part R then the equation is the extension of (23)

in the following way:

$$v_j(LS) = \mu_Q\Big(\frac{\sum_{i=1}^{N_j} f(\mu_S(t_{ij}), \mu_R(t_{ij}))}{\sum_{i=1}^{N_j} \mu_R(t_{ij})}\Big), \quad j = 1 \dots C, \quad \sum_{j=1}^{C} N_j = n, \tag{25}$$

where variables have the same meaning as in (23) and (24).

The main benefit is that data on a lower level remains undisclosed. When data on the lower lever is sensitive or in first step the focus is on searching relevant entities on higher level for further deeper analysis of relevant lower levels this approach is suitable.

Concerning issues related to construction of fuzzy sets, care should be taken when working with LSs with the restriction part. Collected data are usually situated only in a part of the respective attributes' domains. The value of 0 for validity of LS could convey two meanings (Figure 4) [23]: (i) parts of domains included in LS have insufficient tuples for calculation of relations between data (or are empty); (ii) parts of domains contain tuples but there is no significant relation between them. So, it is unclear which interpretation of the value of 0 is the correct one. In order to avoid this trap, membership functions should be constructed from the current content of a database considering only parts of domains which are not empty by the uniform domain covering method.

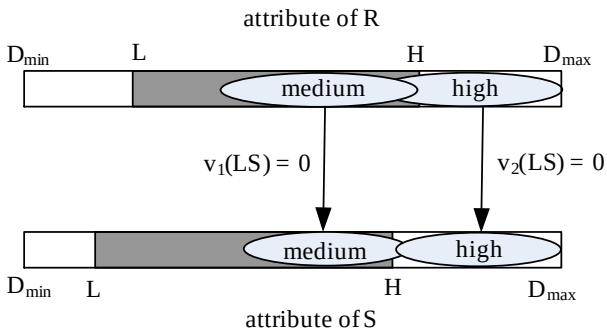

Fig. 4. Ill-defined membership functions.

The next possibility is usage of linguistic quantifiers in order to combine matching degrees of sub-conditions to yield the overall matching degree of a query. The aim is to find all the tuples such that most of attributes (out of a given subset) are as specified [33] using a linguistic quantifier instead of the *and* and *or* connectives. This method handles queries such as *select most of (about half, etc.) tuples out of* $\{A_1$ *is high,* A_2 *is small,* A_3 *is about 5*$\}$ match. Furthermore, quantifiers may be crisp such as *all*, *at least one* and *exactly five*. For instance, a user may be interested to know whether air pollution is severe in the selected destination considering several attributes (relevant pollutants). The pollution can be considered as serious if most of the attributes (measured pollutants) exceed their specified limits [32]. In order to solve this kind of queries, Kacprzyk et al. [32] suggested the FQUERY III+ system.

5. EMPTY AND OVERABUNDANT ANSWERS

Crisp and fuzzy queries contain a logical condition which describes entities we are looking for. A user may be confronted with two extreme situations: no data or very large amount of data satisfy the logical condition. In some cases, these answers are informative enough e. g. empty answer of respondents in delay means that all respondents meet the deadline and no reminder should be generated. In other cases, the main objective is to solve the problem; that is, to obtain a nonempty result to support usersal needs to know why an empty answer occurred and how close are the entities to meet the query condition. The same holds for a plethoric answer. A survey [7] has provided a detailed view into these two problems.

5.1. Empty answer problem

The empty answer problem simply means that there is no data matching the query condition. The query Q_f (3) results in an empty set if $A_{Q_f} = \emptyset$. When the empty answer occurs, it could be useful to provide some alternative data which might indicate why an empty answer problem appeared. This problem was initially recognized in [1] where linguistic modifiers were suggested as a solution. The goal of modifiers is to modify fuzzy sets in order to obtain a less restrictive variant.

Generally, the query Q is transformed into a less restrictive query Q_T by the transformation T. The querying process is then repeated until the answer is not empty or the modified query is semantically far from the original one.

In [8] the following ways for dealing with the empty answer problem are recognized: the linguistic modifier based approach, the fuzzy relative closeness based approach and the absolute proximity based approach. Another approach [5] is focused on replacing the query (which ended as empty one) by a semantically similar one which has been already processed and provided non empty answer.

In these approaches the query weakening should meet the following constraints for each predicate involved in the weakening process [7]:

$$C1: \quad \forall a \in Dom(A),\ \mu_{T(P)}(a) \geq \mu_P(a).$$

Transformation T does not decrease the membership degree for any element a in a domain of attribute A.

$$C2: \quad support\ (P) = \{a \mid \mu_P(a) > 0\} \subset\ support\ (T(P)) = \left\{a \mid \mu_{T(P)}(a) > 0\right\}.$$

Transformation extends the support of fuzzy set (in case of fuzzy set *medium* it is the $[L_d,\ L_g]$ interval depicted in Figure 1) constructed for the predicate P. It means that the condition is relaxed to retrieve more tuples.

$$C3: \quad core\ (P) = \{a \mid \mu_P(a) = 1\} =\ core\ (T(P)) = \left\{a \mid \mu_{T(P)}(a) = 1\right\}.$$

Transformation preserves the core of condition. The transformed query cannot retrieve more tuples which fully meet the relaxed condition than the initial query condition.

An example of the original predicate (P) and the transformed one ($T(P)$) according to requirements C1 – C3 is depicted in Figure 5.

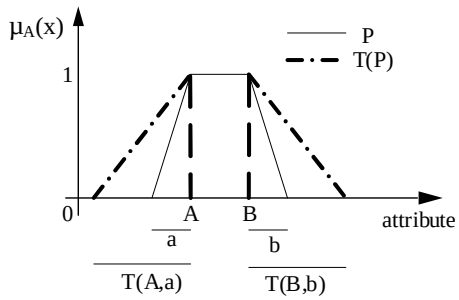

Fig. 5. Weakening of a query condition.

The important issue in weakening is the stopping criterion otherwise the query may retrieve tuples which are semantically far from the initial one or even in an extreme situation the query may retrieve all tuples. In case of the first and third approaches mentioned above no intrinsic semantic limit is provided. The user has to specify a crisp set C_a (set of non-adequate data) by its characteristic function $\varphi_c(a)$ on the domain A. Therefore, the additional condition C4 is as follows (Figure 6):

$$C4: \quad \min(\mu_{T(P)}(a), 1 - \varphi_c(a)) = 0.$$

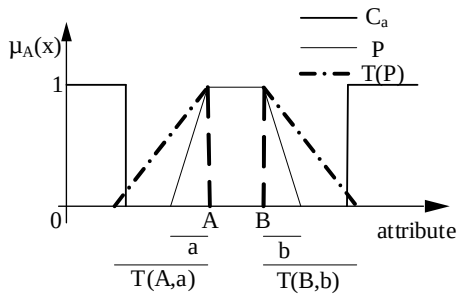

Fig. 6. Constraint of relaxation.

Concerning the requirements C1 – C3, the stopping criterion says that the weakening process stops when the answer to modified Q is not empty. Constraint C4 ensures that tuples which are semantically far from the initial query are not selected. When in the weakening process the transformation $T \ldots T(P) \ldots)$ enters the core of C_a, it causes that C4 $= 0$ (because of $1 - \varphi_c(a) = 0$) and therefore, weakening process stops. The purpose of this condition is controlling the relaxation process.

Instead of the set C_a we can construct fuzzy set F_a, a set of more or less non-adequate data.

Aforementioned approaches deal with the local weakening, that is, the basic weakening transformation applies to each elementary condition. The result of repeated weakening processes is a tree of relaxed queries [7]. In the case of two elementary conditions P_1 and P_2 a tree is depicted in Figure 7. When the number of elementary conditions increases, the size of a tree significantly increases as well. The final step consists of searching the tree in order to find minimal relaxed query condition which provides us with a non-empty answer.

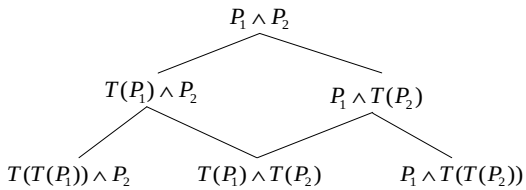

Fig. 7. Tree of relaxed queries (limited to two levels).

Generally, it is valid that the processing of fuzzy queries introduces an additional computation burden due to the substantial amount of calculations concerning data [34]. Moreover, query weakening approaches contain an additional significant calculation burden consisting of executing a series of semantically similar queries and exploitation of the obtained tree. Further research in this field should be focused on reducing this additional burden especially in the execution of a high number of semantically similar queries. Let us assume a query consisting of two elementary conditions. If the query relaxes to a second level of transformation only, software will realize 6 similar queries. A high number of elementary conditions significantly increase the number of nodes in a tree (Figure 7).

5.2. Overabundant answer problem

The opposite situation arises when a large amount of tuples is retrieved from a database. The query Q_f results in overabundant answers if the cardinality of A_{Q_f} (3) is too large.

To recognize the empty answer problem is quite easy. In the case of an overabundant answer problem, it is not easy to unambiguously recognize it. The answer to a question about where lies the boundary between non overabundant and overabundant answers is not easy to give. It depends on how many records user wants to obtain and on the current content of the database. Two situations might arise: too many tuples fully satisfy a query condition and/or too many tuples partially satisfy a query condition. The first situation requires query intensification; that is, reduction of the core of fuzzy sets in order to less tuples fully meet the query condition. At first glance the second situation is easy to solve because the α – cut is an appropriate facility. For example, the threshold clause [13, 55] could solve this problem. However, situation when very large number of answers has the same maximal score (different from value of 1) might occur. In this case the intensification of condition is not the solution. These records will have again the same, although lower membership degree to the answer. The solution could be

adding additional elementary condition semantically close to the initial query condition.

From a theoretical point of view problems of empty and overabundant queries are dual. Ways how to solve overabundant answer problems are examined in [7]. Smits, Pivert and Hadjali [48] address both problematic situations: empty and overabundant answers by the precomputation of summary of database in order to retrieve information about distribution of data in respective domains. In this way the single scan of databases provides relevant information about fuzzy cardinality. Another approach, adding a semantically similar attribute to the query condition to obtain more restrictive solution is suggested in [4]. In this direction fuzzy functional dependencies [26, 54] could be useful in the process of mining related attributes.

From a practical point of view, the empty answer problem is more relevant for users. The empty sheet of data says nothing about stored data in databases but the sheet containing a large amount of data could be examined by several other methods.

6. SOME EFFECTS OF USAGE OF FUZZY LOGIC IN FLEXIBLE QUERYING

It is usually stated that fuzzy data selection expresses an informational need better than crisp data selection. Although this is generally true, several aspects should be considered during the query realization.

Let us recall the known facts that the operations of two-valued logic meet all axioms of Boolean algebra, namely excluded middle, contradiction and idempotency whereas operations of fuzzy logic do not meet all of them. We can find several situations when queries could lead to, from the users' point of view, unexpected results.

Let us look at two queries: *a is High* and *a is High and a is High* where $\mu(a) < 1$. Some t-norms provide different answers to the first and the second query, namely the product (7) and Lukasiewicz (8) t-norms. Moreover, in the case when $\mu(a) < 0.5$ and using Lukasiewicz t-norm the result is 0. This kind of query is not a realistic one but if the user has the freedom to create fuzzy queries over a list of database attributes, these situations might lead (for example in the testing of a software application) to speculation of general applicability of fuzzy queries in practice. This kind of query could appear either as a mistake during the construction of the condition or by purpose during testing application. There is a simple solution: before query realization, check whether user doubled the same elementary condition (semantics of query condition). If it is true, then eliminate doubled elementary condition from a query (it especially holds when other than min t-norm is used). Developers of classical queries do not need to focus their attention on this problem because classical conjunction is idempotent.

This is the consequence of the generalization of truth functionality principle from the two-valued logic to the fuzzy logic. Logic is truth functional if the truth value of a compound sentence depends only on the truth values of the constituent atomic sentences, not on their meaning or structure [43]. In case of the two-valued logic this principle satisfies all axioms. But it is not the case for the fuzzy logic.

Interestingly, in [3] it is stated that " ... the symbols of the (logic) calculus do not depend for their interpretation upon the idea of quantity..." and only "in their particular application..., conduct us to the quantitative conditions of inference". So, according to the above statement the principle of truth functionality is just not enough for generalization.

At any rate, there is a possible answer about how to solve the above-mentioned inconsistency in fuzzy queries. The promising area for research in this field is the Interpolative Realization of Boolean Algebra (IBA) [43]. IBA consists of symbolic level (atomic functions) and value level (intensity $[0, 1]$). It is in line with the ideas expressed in [3]. We have not found any deeper research or realisation in this direction. Therefore, there is a space for evaluating pros and cons of the IBA in this field.

We have seen in previous sections that mathematics based on fuzzy sets and fuzzy logic has greater expressive power than classical mathematics based on crisp sets and crisp logic. But the usefulness depends critically on semantics of data retrieval task, converting task into fuzzy conditions and choosing appropriate aggregation functions. Finally, user awareness of advantages and limitations of fuzzy logic is a prerequisite for an efficient use of fuzzy queries in everyday tasks. Therefore, an application should contain short demonstrating part expressing efficient work with flexible queries.

7. AN EXAMPLE OF PRACTICAL REALISATION

The purpose of an interface is to offer unspecialized users the ability to ask for data without having to know the complexity of a relational database and to lead them through the flexible querying process. Several articles dealing with user-friendly interface can be found e. g. [42] and [52]. In [36] a detailed description of FQUERY user interface is provided. The PostgreSQL_f [46] offers a solution for construction of fuzzy sets by graphical interface through the ReqFlex. Although this topic is a matter of design and programming, the appropriate user interface is crucial for acceptance of flexible queries by end users. It means that items discussed in this paper (dynamic modelling of membership functions, defining preferences, applying non commutative operators, quantifiers, dealing with empty or overabundant answers) need to be offered to users and managed in an appropriate way. In addition, presenting retrieved records and their satisfaction degrees in useful and understandable ways is also very important.

Through the interface users can communicate with the whole flexible data selection process. Each application of a flexible query needs to be optimized according to the field where it is used. Lets assume flexible queries on territorial units as in Figure 8 [25]. In the phase of query creation navigation through a list of query-able attributes needs to be easily manageable. The result of a query could be presented to users in a form of a table or thematic map. The latter one is suitable for dealing with territorial units. For example, territorial units which fully satisfy the query criterion can be marked with one colour, territorial units which do not satisfy the query can be marked with a second colour and territorial units which partially meet the query condition could be marked with a third colour having a colour gradient from a faint hue to a deep hue. In this way the ordinary flexible query application has a high power of information representation. Merging results and suggestions from [28, 36, 42, 46, 52] could create a powered tool.

Although the interface in Figure 8 is adjusted to the specific database, there are parts which are universal. Selecting relevant attributes from the list of all attributes appearing in the database is the first prerequisite. Users should have option to decide which attributes will appear in fuzzy part of a query and which in crisp part of a query. Furthermore, users could define parameters of fuzzy sets either by sliders [46] or directly inserting values (Figure 8). Attributes bearing categorical values could be fuzzified by

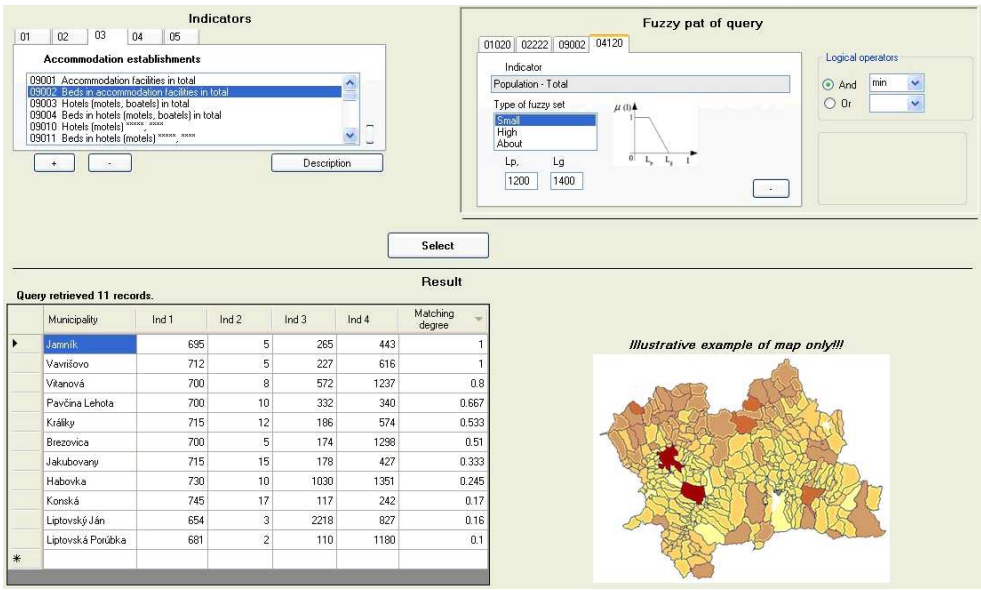

Fig. 8. Interface for flexible queries [25].

assigning membership degree to each relevant category. For example, attribute A_{opin} expressing opinion about something gets values from the set {very low, low, medium, high, very high} for every single tuple. If the query condition is *opinion is positive* then categorical values could be fuzzified as follows: term *high* gets membership degree of 0.75 and term *very high* gets membership degree of 1.

Combo boxes for selecting aggregation operators should contain commutative and non-commutative ones. In case of t-norm operators we should offer functions which do not require more computational burden when larger number of attributes is involved in query condition [45] and cover main users requirements for aggregation discussed in section 4.1 In case of non-commutative aggregation users should be careful in creating the order of attributes (e. g. for the tab control of interface in Figure 8). In bipolar queries wishes and constraints should be clearly separated. Putting constraint into wish and vice versa leads to an inappropriate solution.

The "*brain*" of software tool is the application layer. It communicates with the database and the interface. The application layer should communicate with the database in a such way that no database modification is required. An example is the solution based on the generalized logical condition (GLC) [28]. This solution contains the usual steps for flexible querying: (i) converting fuzzy conditions to classical ones from supports of each fuzzy set and applied aggregation operator; (ii) connecting to database, selecting all candidates (tuples which have membership degree greater than zero) and releasing database connection; (iii) calculating satisfaction degree for each tuple to each elementary condition and finally calculating overall satisfaction degree. The detailed

explanation of this approach can be found in [27]. Although this solution currently supports only the fuzzified *where* clause, it could be adopted to solve majority of queries discussed in Section 4. The next step should be comparison with powerful solutions like SQLf [12] and FQUERY [31].

A powerful scenario for using flexible queries consists of all of the above mentioned items. In the first step software could suggest parameters of linguistic terms according to the current content of a database. Later, user is able to modify these parameters according to his/her opinion about linguistic terms (or just define parameters without suggestion from the software), include preferences, priorities and bipolarities between elementary conditions, set threshold values and choose appropriate aggregation functions. In cases when the result is empty or overabundant, the software and user should cooperate to solve these problems.

8. CONCLUSION

In the last several decades, fuzziness has been studied in the context of the selection of relevant data from relational databases. Flexible query languages provide more human oriented data retrieval in comparison to the classical query languages. Although mathematics based on fuzzy sets and fuzzy logic has greater expressive power than classical mathematics based on crisp sets and crisp logic, the usefulness depends critically on ability to construct appropriate membership functions [38]. Furthermore, in commutative queries different t-norm operators calculate a different satisfaction degree which might cause different sorting of retrieved tuples. The same holds for different functions for quantifiers and implications in bipolar queries and for other non-commutative queries. Flexible querying gives us more freedom in data retrieval processes but also requires us to carefully define parameters and aggregations.

This paper discussed some of the relevant issues of flexible queries namely, the construction of a membership function for each elementary condition, the calculation of matching degree by aggregation functions for commutative and non-commutative queries, the creation of preferences between elementary conditions, the merging of constraints and wishes, how to deal with the empty and overabundant answers, the effects of different aggregation operators on results and the realization focused on the user-friendly interface with touch on evaluation of queries. These issues are not independent. For example, support of querying by construction of membership functions from the current database content could reduce issues of empty or overabundant answers and improve non-commutative aggregation. In addition, this paper outlines ideas for further development.

Database querying tools based on fuzzy logic need additional calculations in comparison with the traditional SQL counterpart. Although this added amount of calculation is balanced with an additional information obtained from databases it could be useful to reduce this burden to a feasible level. Embedding fuzzy querying engine into a relational database management system is a solution which brings improved performances [47]. In this way we can adjust the engine to the specific needs of a particular database management system. On the other hand, if company migrates its data into another database management system, the query engine should be adjusted.

We believe that this contribution has given some relevant information for further

research and helped designers and practitioners in understanding some issues in flexible querying relevant for building applications that will meet particular needs.

(Received January 23, 2014)

REFERENCES

[1] T. Andreasen and O. Pivert: On the weakening of fuzzy relational queries. In: Proc. 8th International Symposium on Methodologies for Intelligent Systems, Charlotte 1994, pp. 144–151. DOI:10.1007/3-540-58495-1_15

[2] T. Bilgiç and I. B. Türkşen: Measurement and elicitation of membership functions. In: Handbook of Granular Computing (W. Pedrycz, A. Skowron and V. Kreinovich, eds.), Wiley-Interscience, Chichester, West Sussex 2008, pp. 141–153. DOI:10.1002/9780470724163.ch6

[3] G. Boole: The calculus of logic. Cambridge and Dublin Math. J. *III* (1848), 183–198.

[4] P. Bosc, A. Hadjali, O. Pivert, and G. Smits: An approach based on predicate correlation to the reduction of plethoric answer sets. In: Advances in Knowledge Discovery and Management. Studies in Computational Intelligence, Volume 398 (F. Guillet, B. Pinaud, G. Venturini and D.A. Zighed, eds.), Springer-Verlag, Heidelberg 2012, pp. 213–233. DOI:10.1007/978-3-642-25838-1_12

[5] P. Bosc, C. Brando, A. Hadjali, H. Jaudoin, and O. Pivert: Semantic proximity between queries and the empty answer problem. In: Proc. Joint IFSA-EUSFLAT Conference, Lisbon 2009, pp. 259–264.

[6] P. Bosc, D. Kraft, and F. Petry: Fuzzy sets in database and information systems: Status and opportunities. Fuzzy Sets and Systems *156* (2005), 418–426. DOI:10.1016/j.fss.2005.05.039

[7] P. Bosc, A. Hadjali, and O. Pivert: Empty versus overabundant answers to flexible relational queries. Fuzzy Sets and Systems *159* (2008), 1450–1467. DOI:10.1016/j.fss.2008.01.007

[8] P. Bosc, A. Hadjali, and O. Pivert: Weakening of fuzzy relational queries: and absolute proximity relation-based approach. Mathware and Soft Comput. *14* (2007), 35–55.

[9] P. Bosc, O. Pivert and G. Smits: On a fuzzy group-by and its use for fuzzy association rule mining. In: Proc. 14th East-European Conference on Advances in Databases and Information Systems (ADBIS'10), Novi Sad 2010, pp. 88–102. DOI:10.1007/978-3-642-15576-5_9

[10] P. Bosc and O. Pivert: On a fuzzy bipolar relational algebra. Inform. Sci. *219* (2013), 1–16. DOI:10.1016/j.ins.2012.07.018

[11] P. Bosc and O. Pivert: On four noncommutative fuzzy connectives and their axiomatization. Fuzzy Sets and Systems *202* (2012), 42–60. DOI:10.1016/j.fss.2011.11.005

[12] P. Bosc and O. Pivert: SQLf query functionality on top of a regular relational database management system. In: Knowledge Management in Fuzzy Databases (M. Pons, M. Vila and J. Kacprzyk, eds.), Physica-Verlag, Heidelberg 2000, pp. 171–190. DOI:10.1007/978-3-7908-1865-9_11

[13] P. Bosc and O. Pivert: SQLf: a relational database language for fuzzy querying. IEEE Trans. Fuzzy Systems *3* (1995), 1–17. DOI:10.1109/91.366566

[14] P. Bosc, O. Pivert, and A. Mokhtari: On fuzzy queries with contextual predicates. In: Proc. International Conference on Fuzzy Systems (FUZZ-IEEE 2009), Jeju Island 2009, pp. 484–489. DOI:10.1109/fuzzy.2009.5277136

[15] E. Cox: Fuzzy Modeling and Genetic Algorithms for Data Mining and Exploration. Morgan Kaufman, San Francisco 2005. DOI:10.1016/b978-012194275-5/50002-5

[16] D. Dubois and H. Prade: Handling bipolar queries in fuzzy information processing In: Handbook of Research on Fuzzy Information Processing in Databases (J. Galindo, ed.), Information Science Reference, Hershey 2008, pp. 97–114. DOI:10.4018/978-1-59904-853-6.ch004

[17] D. Dubois and H. Prade: Using fuzzy sets in flexible querying: Why and how? In: Flexible Query Answering Systems (T. Andreasen, H. Christiansen and H. L. Larsen, eds.), Kluwer Academic Publishers, Dordrecht 1997, pp. 45–60. DOI:10.1007/978-1-4615-6075-3_3

[18] D. Dubois and H. Prade: Weighted minimum and maximum operations. Inform. Sci. *39* (1986), 205–210. DOI:10.1016/0020-0255(86)90035-6

[19] J. M. Garibaldi and R. I. John: Choosing membership functions of linguistic terms. In: Proc. 12th IEEE International Conference on Fuzzy Systems (FUZZ'03), St. Louis 2003, pp. 578–583. DOI:10.1109/fuzz.2003.1209428

[20] R. George and R. Srikanth: Data summarization using genetic algorithms and fuzzy logic. In: Genetic Algorithms and Soft Computing (F. Herrera and J. L. Verdegay, eds.), Physica Verlag, Heidelberg 1996, pp. 599–611.

[21] I. Glöckner: Quantifier selection for linguistic data summarization. In: Proc. IEEE International Conference on Fuzzy Systems, Vancouver 2006, pp. 720–727. DOI:10.1109/fuzzy.2006.1681790

[22] M. Gupta and J. Qi: Theory of t-norms and fuzzy inference methods. Fuzzy Sets and Systems *40* (1991), 431–450. DOI:10.1016/0165-0114(91)90171-l

[23] M. Hudec, M. Vučetić, and M. Vujošević: Synergy of linguistic summaries and fuzzy functional dependencies for mining knowledge in the data. In: Proc. 18th IEEE International Conference on System Theory, Control and Computing (ICSTCC 2014), Sinaia 2013, pp. 335–340.

[24] M. Hudec: Issues in construction of linguistic summaries. In: Proc. Uncertainty Modelling 2013 (R. Mesiar and T. Bacigál, eds.), STU, Bratislava 2013, pp. 35–44.

[25] M. Hudec: Improvement of data collection and dissemination by fuzzy logic. In: Joint UNECE/Eurostat/OECD Meeting on the Management of Statistical Information Systems (MSIS 2013), Paris – Bangkok 2013.

[26] M. Hudec, M. Vučetić, and M. Vujošević: Comparison of linguistic summaries and fuzzy functional dependencies related to data mining. In: Biologically-Inspired Techniques for Knowledge Discovery and Data Mining (S. Alam, G. Dobbie, Y. Sing Koh and S. ur Rehman, eds.), Information Science Reference, Hershey 2014, pp. 174–203.

[27] M. Hudec: Fuzzy improvement of the SQL. Yugoslav J. Oper. Res. *21* (2011), 2, 239–251. DOI:10.2298/yjor1102239h

[28] M. Hudec: An approach to fuzzy database querying, analysis and realisation. Computer Sci. Inform. Systems *6* (2009), 2, 127–140. DOI:10.2298/csis0902127h

[29] M. Hudec and F. Sudzina: Construction of fuzzy sets and applying aggregation operators for fuzzy queries. In: Proc. 14th International Conference on Enterprise Information Systems (ICEIS 2012), Wroclaw 2012, Proceedings volume 1, pp. 253–257. DOI:10.5220/0003968802530258

[30] J. Kacprzyk and S. Zadrożny: Protoforms of linguistic database summaries as a human consistent tool for using natural language in data mining. Int. J. Software Sci. and Comput. Intel. *1* (2009), 100–111. DOI:10.4018/jssci.2009010107

[31] J. Kacprzyk and S. Zadrożny: FQUERY for Access: Fuzzy querying for windows-based DBMS. In: Fuzziness in Database Management Systems (P. Bosc and J. Kacprzyk, eds.), Physica-Verlag, Heidelberg 1995, pp. 415–433. DOI:10.1007/978-3-7908-1897-0_18

[32] J. Kacprzyk, S. Zadrożny, and A. Ziółkowski: FQUERY III +: A "human-consistent" database querying system based on fuzzy logic with linguistic quantifiers. Information Systems *14* (1989), 6, 443–453. DOI:10.1016/0306-4379(89)90012-4

[33] J. Kacprzyk and A. Ziółkowski: Database queries with fuzzy linguistic quantifiers. IEEE Trans. Systems, Man and Cybernetics *SMC-16* (1986), 3, 474–479. DOI:10.1109/tsmc.1986.4308982

[34] J. Kacprzyk, G. Pasi, P .Vojtáš, and S. Zadrożny: Fuzzy querying: issues and perspectives. Kybernetika *36* (2000), 6, 605–616.

[35] J. Kacprzyk and R. R. Yager: Linguistic summaries of data using fuzzy logic. International Journal of General Systems *30* (2001), 133–154. DOI:10.1080/03081070108960702

[36] J. Kacprzyk and S. Zadrożny: Computing with words in intelligent database querying: standalone and internet-based applications. Inform. Sci. *134* (2001), 71–109. DOI:10.1016/s0020-0255(01)00093-7

[37] E. Klement, R. Mesiar, and E. Pap: Triangular Norms. Kluwer Academic Publishers, Dordrecht 2000. DOI:10.1007/978-94-015-9540-7

[38] G. Klir and B. Yuan: Fuzzy Sets and Fuzzy Logic, Theory and Applications. Prentice Hall, New Jersey 2005.

[39] M. Lacroix and P. Lavency: Preferences: putting more knowledge into queries. In: Proc. 13th International Conference on Very Large Databases, Brighton, 1987 pp. 217–225.

[40] O. Pivert and P. Bosc: Fuzzy Preference Queries to Relational Databases. Imperial College Press, London 2012. DOI:10.1142/9781848168701

[41] D. Rasmussen and R. Yager: Summary SQL - A fuzzy tool for data mining. Intelligent Data Analysis *1* (1997), 49–58. DOI:10.1016/s1088-467x(98)00009-2

[42] R. Ribeiro and A. Moreira: Fuzzy query interface for a business database. Int. J. of Human-Computer Studies *58* (2003), 363–391. DOI:10.1016/s1071-5819(03)00010-7

[43] D. Radojević: Interpolative realization of Boolean algebra as a consistent frame for gradation and/or fuzziness. In: Forging New Frontiers: Fuzzy Pioneers II Studies in Fuzziness and Soft Computing (M. Nikravesh, J. Kacprzyk and L. Zadeh, eds.), Springer-Verlag, Berlin Heidelberg 2008, pp. 295–318. DOI:10.1007/978-3-540-73185-6_13

[44] A. Rosado, R. Ribeiro, S. Zadrożny, and J. Kacprzyk: Flexible query languages for relational databases: An overview In: Flexible Databases Supporting Imprecision and Uncertainty. Studies in fuzziness and soft computing, Vol. 203 (G. Bordogna and G. Psaila, eds.), Springer-Verlag, Berlin Heidelberg 2006, pp. 3–53. DOI:10.1007/3-540-33289-8_1

[45] W. Siler and J. Buckley: Fuzzy Expert Systems and Fuzzy Reasoning. John Wiley and Sons, New Jersey 2005. DOI:10.1002/0471698504

[46] G. Smits, O. Pivert, and T. Girault: ReqFlex: Fuzzy queries for everyone. In: Proc. 39th International Conference on Very Large Data Bases, Trento 2013, pp. 1206–1209. DOI:10.14778/2536274.2536277

[47] G. Smits, O. Pivert, and T. Girault: Towards reconciling expressivity, efficiency and user-friendliness in database flexible querying. In: Proc. 22th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2013), Hyderabad 2013, pp. 1–8. DOI:10.1109/fuzz-ieee.2013.6622356

[48] G. Smits, O. Pivert, and A. Hadjali: Fuzzy cardinalities as a basis to cooperative answering. In: Flexible Approaches in Data, Information and Knowledge Management (O. Pivert and S. Zadrożny, eds.), Studies in Computational Intelligence, volume 497, Springer, Berlin Heidelberg 2013, pp. 261–289. DOI:10.1007/978-3-319-00954-4_12

[49] V. Tahani: A conceptual framework for fuzzy query processing: a step toward very intelligent database systems. Inform. Processing and Management *13* (1977), 5, 289–303. DOI:10.1016/0306-4573(77)90018-8

[50] C. Tudorie, S. Bumbaru, and L. Dumitriu: Relative qualification in database flexible queries. In: Proc. 3rd International IEEE Conference on Intelligent Systems, London 2006, pp. 83–88. DOI:10.1109/is.2006.348398

[51] C. Tudorie: Qualifying objects in classical relational database querying In: Handbook of Research on Fuzzy Information Processing in Databases (J. Galindo, ed.), Information Science Reference, Hershey 2008, pp. 218–245. DOI:10.4018/978-1-59904-853-6.ch009

[52] C. Tudorie: Intelligent interfaces for database fuzzy querying. The annals of Dunarea de Jos University of Galati, Fascicle III *32* (2009), 2.

[53] J. Verkulien: Assigning membership in a fuzzy set analysis. Sociological Methods Res. *33* (2005), 462–496. DOI:10.1177/0049124105274498

[54] M. Vučetić and M. Vujošević: A literature overview of functional dependencies in fuzzy relational database models. Technics Technologies Education Management *7* (2012), 4, 1593–1604.

[55] T. C. Wang, H. D. Lee, and C. M. Chen: Intelligent queries based on fuzzy set theory and SQL. In: Proc. Joint Conference on Information Science, Salt Lake City 2007, pp. 1426–1432. DOI:10.1142/9789812709677_0203

[56] N. Werro, A. Meier, C. Mezger, and G. Schindler: Concept and implementation of a fuzzy classification query language. In: Proc. International Conference on Data Mining, Las Vegas 2005, pp. 208–214.

[57] H. C. Wu: Fuzzy Systems and Neural Networks. National Chi Nan University, Puli, Nantou 2002.

[58] R. Yager: Higher structures in multi-criteria decision making. International Journal of Man-Machine Studies *36* (1992), 553–570. DOI:10.1016/0020-7373(92)90096-4

[59] R. R. Yager: On ordered weighted averaging operators in multicriteria decision making. IEEE Trans. Systems, Man and Cybernetics *SMC-18* (1988), 183–190. DOI:10.1109/21.87068

[60] R. R. Yager: A new approach to the summarization of data. Information Sciences *28* (1982), 69–86. DOI:10.1016/0020-0255(82)90033-0

[61] M. Ying: Implication operators in fuzzy logic. IEEE Trans. Fuzzy Systems *10* (2002), 1, 88–91. DOI:10.1109/91.983282

[62] L. Zadeh: A computational approach to fuzzy quantifiers in natural languages. Computers and Math. Appl. *9* (1983), 149–184. DOI:10.1016/0898-1221(83)90013-5

[63] L. Zadeh: Fuzzy sets. Information and Control *8* (1965), 338–353. DOI:10.1016/s0019-9958(65)90241-x

[64] S. Zadrożny and J. Kacprzyk: Issues in the practical use of the OWA operators in fuzzy querying. J. Intell. Inform. Systems *33* (2009), 307–325. DOI:10.1007/s10844-008-0068-1

[65] S. Zadrożny and J. Kacprzyk: Bipolar queries: a way to enhance the flexibility of database queries In: Advances in Data Management, Studies in Computational Intelligence, Vol. 223 (Z. W. Ras and A. Dardzinska, eds.), Springer-Verlag, Berlin Heidelberg 2009, pp. 49–66. DOI:10.1007/978-3-642-02190-9_3

[66] S. Zadrożny, G. de Tré, R. de Caluwe, and J. Kacprzyk: An overview of fuzzy approaches to flexible database querying In: Handbook of Research on Fuzzy Information Processing in Databases (J. Galindo, ed.), Information Science Reference, Hershey 2008, pp. 34–55. DOI:10.4018/978-1-59904-853-6.ch002

[67] S.-M. Zhou, F. Chiclana, R. I. John, and J. M. .Garibaldi: Fuzzification of the OWA operators for aggregating uncertain information with uncertain weights. In: Recent Developments in the Ordered Weighted Averaging Operators: Theory and Practice (R. R. Yager, J. Kacprzyk and G. Beliakov, eds.), Studies in Fuzziness and Soft Computing Volume 265, Springer-Verlag, Berlin Heidelberg 2011, pp. 91–109. DOI:10.1007/978-3-642-17910-5_5

[68] S.-M. Zhou, F. Chiclana, R. I. John, and J. M. .Garibaldi: Type-1 OWA operators for aggregating uncertain information with uncertain weights induced by type-2 linguistic quantifiers. Fuzzy Sets and Systems *159* (2008), 3281–3296. DOI:10.1016/j.fss.2008.06.018

[69] H. J. Zimmerman and P. Zysno: Latent connectives in human decision making. Fuzzy Sets and Systems *4* (1980), 37–51. DOI:10.1016/0165-0114(80)90062-7

Miroslav Hudec, Faculty of Economic Informatics, University of Economics in Bratislava, Dolnozemska cesta 1, 852 35 Bratislava. Slovak Republic.
e-mail: miroslav.hudec@euba.sk

Miljan Vučetić, Research and Development Institute "Vlatacom", Milutina Milankovica 5, 11070 Belgrade. Serbia.
e-mail: miljanvucetic@gmail.com