



ELSEVIER

Information Sciences 134 (2001) 71–109

INFORMATION
SCIENCES

AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

Computing with words in intelligent database querying: standalone and Internet-based applications

Janusz Kacprzyk *, Sławomir Zadrozny

Systems Research Institute, Polish Academy of Sciences, ul. Newelska 6, 01-447 Warsaw, Poland

Abstract

We present how computing with words, meant as a set of fuzzy-logic-based tools for an effective and efficient handling of imprecise elements of natural language, can be implemented for fuzzy querying via a user-friendly interface to Microsoft Access, FQUERY for Access. The system accommodates fuzzy (imprecise) terms and linguistic quantifiers allowing for queries exemplified by “find (all) records such that *most* of the (important) clauses are satisfied (to a degree from $[0, 1]$)”. L.A. Zadeh’s [Comput. Math. Appl. 9 (1983) 149] fuzzy logic based calculus of linguistically quantified propositions, and R.R. Yager’s [IEEE Trans. Syst. Man Cybernet. 18 (1988) 183] ordered weighted averaging (OWA) operators are employed to deal with fuzzy linguistic quantifiers. It is then shown how FQUERY for Access, which is a standalone application, may be extended to support fuzzy querying via the Internet (or, analogously, Intranet). It is shown how WWW browsers, both the Netscape Navigator and Microsoft Explorer, can be employed for developing a fuzzy querying interface for handling imprecise natural language elements in database queries following Zadeh’s computing with words paradigm. © 2001 Published by Elsevier Science Inc.

Keywords: Database querying; Flexible querying; Fuzzy querying; Fuzzy logic; Microsoft Access; Fuzzy querying via Internet; Computing with words

* Corresponding author.

E-mail addresses: kacprzyk@ibspan.waw.pl (J. Kacprzyk), zadrozny@ibspan.waw.pl (S. Zadrozny).

1. Introduction

Database management systems (DBMSs) are extremely relevant and useful software products which are used in all kinds of systems supporting human activities. For a long time people have been aware of an inherent discrepancy between the “hard” machine and the “soft” human being. One of the reasons has been the fact that all our computing machinery, in the sense of both hardware and software, employs explicitly or implicitly some artificial, precise formalisms which are unable to handle ambiguity, vagueness, imprecision, etc. of natural language, which is the only fully natural human communication means. Some natural language interfaces to DBMSs have been proposed to overcome this inherent difficulty (cf. [3]).

Fuzzy logic has played here a crucial role making it possible to considerably improve those interfaces by providing formal means to handle vagueness resulting from the use of natural language. A further impetus has been provided by a general paradigm of *computing with words*, advocated by Zadeh in recent years, which is closely related to fuzzy logic. In this paper we will show how fuzzy logic, in the sense of a formal tool on which computing with words is based, can provide new qualities to database querying. We will present some linguistic, fuzzy-logic-based interfaces to DBMSs both for standalone applications, and for querying over the Internet (in fact, we will mean here and later on over the Internet and Intranet).

This paper is based, first of all, on our previous works [12–19,23–25,38,39]. Basically, we assume that a commercial, widely available and popular DBMS – Microsoft Access®, to be more specific – is employed, and fuzzy querying is implemented as an add-on (or add-in). This is, in our view, the only practically relevant solution at present as, first, there are no commercial “fuzzy DBMSs” on the market, second, the development of such a fuzzy database would be prohibitively costly and time-consuming, and, third, testing of fuzzy querying may proceed on popular (non-fuzzy) DBMSs which are relatively inexpensive and widely available. These works have resulted in the FQUERY for Access, a fuzzy querying systems for Microsoft Access.

There are two basic lines of research in the use of fuzzy sets and logic in DBMSs [30]. The first is to assume that the database is conventional and fuzzy sets, possibility theory, fuzzy logic, etc. are used to make its use easier and more human consistent (mainly in the sense of handling vagueness resulting from the use of natural language). This is mainly done by constructing some add-ons, and may be exemplified by [1,2,4,5], Buell (1982), [8,9,12–19,23–25,27,31], Yager (1980), [38,39]; see also [42,30].

The second approach is to build a DBMS which can involve imprecision and vagueness represented by fuzzy or possibilistic elements. Clearly, querying, updating, etc. is also based on fuzzy logic. This approach is exemplified by [7], Vila et al. (1995), [42]; see also [30], and [36] for an excellent review and perspective.

This paper belongs to the first class. We assume a conventional (non-fuzzy) DBMS, a commercial and popular one (Microsoft Access[®] 7), and construct an add-on to it which may help the human operator (e.g., decision maker) retrieve information related to some highly aggregated and vaguely defined concepts, relations, etc. For instance, in a water quality related context, in which the first source [24,25] works have been developed, the user may be interested in knowing all points in a watershed where the environmental pollution is *severe*. It is quite clear that it is very difficult, if not impossible, to adequately represent *severe* by a conventional query involving the ANDing and ORing of attributes' values only as may be done by using conventional querying tools (e.g., SQL). On the other hand, it may be subjectively felt that in such a case an adequate representation of “*severe* (pollution)” is when, e.g., “*most* of the *important* pollution indicators *considerably* exceed some (possibly fuzzily specified) limits”. Notice that a linguistic quantifier (*most*) which is used here cannot be accommodated in an adequate way in a conventional DBMS.

In this paper we show how basic elements of Zadeh's paradigm of computing with words: linguistic values (e.g., *high*), linguistic relations (e.g., *much more than a half*), linguistic modifiers (e.g., *very*), and linguistic quantifiers (e.g., *most*) can be used in devising more human-consistent and human-friendly querying interfaces to DBMSs.

Basically, our querying system allows for queries of the type “find (all) records such that *most* (almost all, much more than a half, ... or any other suitable linguistic quantifier) of the *important* attributes are as specified (e.g., equal to 5, greater than 10, much less than 100, low, etc.)”.

First, we discuss the use of fuzzy sets (logic) in standalone database querying showing where such fuzzy elements may be employed. Second, we briefly present how to define fuzzy elements appearing in the queries. Third, we discuss the implementation of the querying scheme proposed, mainly the transformation of a fuzzy query into its equivalent legitimate Access' 7 query. We present an example which, for readability and clarity, uses the NWIND.MDB database of a fictitious trading company which is provided in Access.

In next parts we describe how this querying system can be implemented using a WWW browser (Netscape Navigator[®] and Microsoft Internet Explorer[®]) as interface as proposed and implemented by [16,20].

2. Flexible querying in a DBMS using the paradigm of computing with words

In querying users are often interested in answers to imprecisely specified questions that are natural to humans but strange to the “machine”. For instance, in case of a personnel database the user may wish to retrieve *all younger*

much_better_than_average_paid_employees which is inconsistent with conventional (precise) querying languages.

The problem is how to extend a query language so as to allow for the use of such linguistic (fuzzy) terms “low”, “much greater than”, “most”, etc. This involves four issues to be dealt with:

- extension of the syntax of a query language,
- semantics of an extended language,
- elicitation and manipulation of linguistic (fuzzy) terms and queries containing them, and
- embedding fuzzy querying engine in a native querying environment of a host DBMS.

We will deal with the two former in this section, and with the two latter in the next section. Needless to say that the above issues are relevant for all kinds of fuzzy querying, for standalone and Internet applications.

Internally, Access represents a query using SQL, hence we focus on the simple query of the type:

SELECT ⟨list of fields⟩ FROM ⟨list of tables⟩ WHERE ⟨condition⟩

and propose the following extension to the syntax of its most interesting – from our point of view – part, i.e., WHERE clause:

⟨condition⟩ ::= ⟨linguistic quantifier⟩
 ⟨sequence of subconditions⟩;

⟨sequence of subconditions⟩ ::=
 ⟨subcondition⟩|
 ⟨subcondition⟩ OR ⟨sequence of subconditions⟩

⟨subcondition⟩ ::=
 ⟨linguistic quantifier⟩ ⟨importance coefficient⟩
 ⟨sequence of atomic conditions⟩

⟨sequence of atomic conditions⟩ ::=
 ⟨atomic condition⟩|
 ⟨atomic condition⟩ AND ⟨sequence of atomic conditions⟩

⟨atomic condition⟩ ::=
 ⟨attribute⟩ = ⟨fuzzy value⟩|
 ⟨attribute⟩ ⟨classical relational operator⟩
 ⟨numerical-attribute⟩|
 ⟨attribute⟩ ⟨fuzzy relation⟩ ⟨attribute⟩|
 ⟨attribute⟩ ⟨fuzzy relation⟩ ⟨number⟩|
 ⟨single-valued-attribute⟩ IN ⟨fuzzy-set constant⟩|
 ⟨multi-valued-attribute⟩ ⟨compatibility operator⟩ ⟨fuzzy-set constant⟩|

⟨attribute⟩ ::= ⟨numeric field⟩

$\langle \text{linguistic quantifier} \rangle ::= \langle \text{OWA-tag} \rangle \langle \text{quantifier name} \rangle$

$\langle \text{OWA-tag} \rangle ::= \text{OWA} |$

$\langle \text{classical relational operator} \rangle ::= \langle | \langle = | \rangle | \rangle = | =$

The above query syntax supported is very general allowing for virtually all kinds of linguistic elements.

The main entities used in the system, their linguistic (fuzzy) representation, elicitation and handling are:

- *Attributes*

First, to use a (numerical) field in a query in connection with a fuzzy value or relation, it has to be declared as an attribute. For each attribute the user specifies: the lower limit (LL) and upper limit (UL) which determine the value interval in which the field's values are. They are used for scaling the values of the attributes for the particular records while calculating the degree of matching with a linguistic (fuzzy) value used, or the degree of membership in a fuzzy relation. In fact, they need not describe the real value interval of the attributes.

- *Single-valued attributes*
- *Multi-valued attributes*

Both types of attributes may be used along with the fuzzy set constant in a query. A single-valued attribute may be considered as a special case of a multi-valued attribute. Namely, only a special case of one type of the compatibility operators (IN) is meaningful in the case of the former one, whereas various types of compatibility indices may be employed in the case of the latter. Both types of attributes may be exemplified by "Country" and "Main_products_purchased" fields in a database of a trading company, respectively. In case of the former one a fuzzy set may be used in a corresponding atomic condition of a query as, e.g., in "find customers from Central Europe". In case of the latter one a fuzzy set may be used in a query as well as in a record as a value of the attribute. The value of such an attribute will be a list of relevant products. The direct use of such types of data is inconsistent with the relational database paradigm. Still, such an attribute may exist in the user's view of the database, even if the real arrangement of the data is different. In the original database scheme, a list of fields corresponds to such a virtual multi-valued attribute. Each of this fields is of logical or real type, corresponding to the characteristic function of a crisp set or the membership function of a fuzzy set, respectively.

The matching degree for an atomic condition involving a single-valued attribute (AT) and a fuzzy set (FS) is calculated for each record in a straightforward manner as the value of the membership function of the fuzzy set FS for the element being the value of the attribute AT in a given record.

The calculation of matching degree for multi-valued attributes is discussed in what follows with regards to the compatibility operator.

- *Linguistic (fuzzy) values*

These imprecise linguistic terms as, e.g., *large* in “salary is *large*”, are defined by membership functions on the interval $[-10, +10]$, to allow for context-independent definitions. The membership functions are assumed trapezoidal as in Fig. 1 which is simple and sufficient in practice.

Thus, to define a fuzzy value the four points *A*, *B*, *C*, and *D* are needed, and their interpretation is obvious.

The matching degree, $\text{md}(\cdot, \cdot)$, of an *atomic condition* $\text{AT} = \text{FV}$ and a record *R* is

$$\text{md}(\text{AT} = \text{FV}, R) = \mu_{\text{FV}}(\tau(R(\text{AT}))), \quad (1)$$

where $R(\text{AT})$ is the value of the attribute *AT* in record *R*, μ_{FV} is the membership function of the fuzzy value *FV*, and $\tau : [\text{LL}_{\text{AT}}, \text{UL}_{\text{AT}}] \rightarrow [-10, 10]$ is the mapping from the variability interval of the attribute *AT* onto the unified $[-10, 10]$ interval.

- *Linguistic (fuzzy) relations*

Linguistic (fuzzy) relations, exemplified by “Amount_in_Stock Is MUCH_GREATER_THAN Amount_on_Orders”, is represented by a binary fuzzy relation. The interpretation of a fuzzy relation is similar to that for a fuzzy value. The main difference is that in case of a fuzzy value just one universe of discourse (i.e. the set of possible values of a particular attribute) is to be employed. On the other hand, in case of a binary fuzzy relation two attributes are involved, that is we are dealing with an atomic conditions like

$$\text{FR}(\text{AT1}, \text{AT2}). \quad (2)$$

A natural approach is to assume the universe of discourse to be the set of possible values of the difference of the values of two attributes, that is $[\text{LL}_{\text{AT1}} - \text{UL}_{\text{AT2}}, \text{UL}_{\text{AT1}} - \text{LL}_{\text{AT2}}]$. Mapping again the resulting range of variability onto the unified interval $[-10, 10]$, a fuzzy relation, *FR*, may be equated with a fuzzy set, *FRS*, defined on this interval, i.e., $\mu_{\text{FR}}(x, y) = \mu_{\text{FRS}}(x - y)$.

The user should therefore define its membership function, assumed to be trapezoidal, analogously as for fuzzy values (cf. Fig. 1).

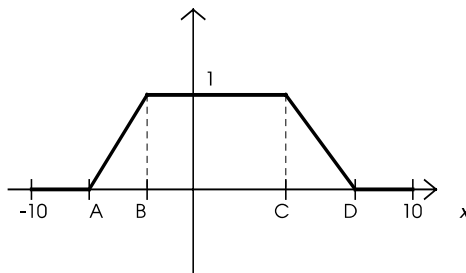


Fig. 1. Trapezoidal membership function of a linguistic (fuzzy) value.

Thus, to define a membership function the user should give four points: A , B , C and D . Then, to evaluate the fulfillment of the atomic condition (2), for the attributes $AT1$ and $AT2$ values in record R the interval $[LL_{AT1} - UL_{AT2}, UL_{AT1} - LL_{AT2}]$ is linearly mapped onto the interval $[-10, 10]$. Then, the mapped difference of values of $AT1$ and $AT2$ is calculated, and:

- if it is below A , then the fuzzy relation FR is not fulfilled at all, i.e. to degree 0.0,
- if it is between A and B , then FR is fulfilled to a degree between 0.0 and 1.0, the closer to B the more;
- if it is between B and C , then FR is fully satisfied, i.e. to degree 1;
- if it is between C and D , then FR is satisfied to a degree between 1.0 and 0.0, the closer to D the less;
- if it is above D , then FR is not satisfied at all, i.e. to degree 0.0.

Thus, the matching degree, $md(\cdot, \cdot)$, of an *atomic condition* (2), and a record R is

$$md(FR(AT1, AT2), R) = \mu_{FR}(\tau_1(R(AT1)), \tau_2(R(AT2))), \quad (3)$$

where $R(AT1)$, $R(AT2)$ are the values of the attributes $AT1$ and $AT2$, in R , μ_{FR} is the membership function of the fuzzy relation FR , and $\tau_i : [LL_{ATi}, UL_{ATi}] \rightarrow [-10, 10]$, $i = 1, 2$, are the mappings from the variability intervals of $AT1$ and $AT2$ onto $[-10, 10]$.

For example, if we have the following two attributes with the same range of variability (from 0 to 1000): $AT1$: Amount_In_Stock and $AT2$: Amount_On_Orders, a fuzzy relation IS_ABOUT defined by four points $(-2, -1, 1, 2)$ and a query:

Amount_In_Stock IS_ABOUT Amount_On_Orders

then, for a record R with Amount_In_Stock = 500 and Amount_On_Orders = 1000 we obtain the matching degree equal to 0.0. On the other hand, for a record with Amount_In_Stock = 910 and Amount_On_Orders = 1000 the matching degree is equal 1.0.

- *Linguistic quantifiers*

The linguistic quantifiers are used in statements like “most clauses of the query are to be satisfied” where clauses are meant either as subconditions or atomic conditions. In the FQUERY for Access system the fuzzy linguistic quantifiers are defined in the sense of [37], with the $[0.0, 10.0]$ interval assumed for technical, not conceptual reasons. On the other hand, they may be interpreted either using original Zadeh’s approach or via the ordered weighted averaging (OWA) operators (cf. [34,35]). We assume that the membership function of the fuzzy linguistic quantifier is piecewise linear as sketched in Fig. 2.

To define a fuzzy linguistic quantifier it is therefore needed to provide two numbers corresponding to A and B meant as follows. If a query is composed of

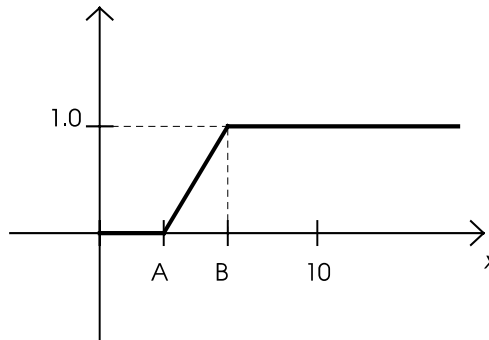


Fig. 2. Example of the membership function of a fuzzy linguistic quantifier.

N subconditions (atomic conditions), we first map $[0, N]$ and the number of actually satisfied clauses onto $[0, 10]$. Notice that the above should be properly understood because the particular clauses may be satisfied to a degree which is between 0.0 and 1.0. Then:

- if the number of clauses satisfied is less than A , then the query is not satisfied (by a particular record) at all or, is satisfied to degree 0.0.
- if the number of clauses satisfied is between A and B , then the query is satisfied to a degree between 0.0 and 1.0, the closer to B the higher.
- if the number of clauses satisfied is above B , then the query is fully satisfied, that is to degree 1.0.

Thus, the matching degree, $\text{md}(\cdot, \cdot)$, for the query “ Q of N clauses are satisfied” for record R is equal to

$$\text{md}(Q\{\text{clause}_{i=1,\dots,N}\}, R) = \mu_Q \left(\tau \left(\frac{1}{N} \sum_i \text{md}(\text{clause}_i, R) \right) \right), \quad (4)$$

where $\text{md}(\text{clause}_i, R)$ is the matching degree for clause i , for record R , μ_Q is the membership function of the linguistic quantifier Q , and $\tau: [0, N] \rightarrow [0, 10]$ is the mapping onto the unit interval.

The above formula may be extended to the case when the importances of particular clauses are to be taken into account (cf. [37] for tools).

Moreover, the OWA operator interpretation of a linguistic quantifier may also be employed [cf. [35 volume]. In this case, we start with [37] classic definition of a fuzzy linguistic quantifier of the type shown in Fig. 2. Then, the weights of an OWA operator of dimension N determined for such a regular monotone non-decreasing linguistic quantifier Q , are calculated due to [34]:

$$w_i = \begin{cases} Q(i/N) - Q((i-1)/N) & \text{for } i = 1, \dots, N, \\ 0 & \text{for } i = 0. \end{cases} \quad (5)$$

The user may modify the particular weights, and to support the fine tuning of the OWA weights, the measures of ORness and dispersion are provided. These measures were introduced by [34] and are calculated for a given OWA operator F with the weight vector $[w_i]_{i=1,\dots,N}$ as follows:

$$\text{ORness}(F) = \left(\sum_{i=1}^N (N - i) * w_i \right) / (N - 1), \tag{6}$$

$$\text{disp}(F) = - \sum_{i=1}^N w_j * \ln(w_j). \tag{7}$$

The measure of $\text{ORness}(F)$ says how similar is the OWA operator F to the logical connective OR (in terms of its aggregation behavior), which may be written symbolically as

$$\text{ORness}(F) = \begin{cases} 1 & \text{iff } F \text{ is OR } (F \equiv \text{OWA}^{\text{OR}}), \\ a \in (0, 1) & \text{iff } F \text{ is "between" OR and AND,} \\ 0 & \text{iff } F \text{ is AND } (F \equiv \text{OWA}^{\text{AND}}), \end{cases} \tag{8}$$

where OWA^{OR} is the OWA operator corresponding to OR, i.e. $F = [1, 0, \dots, 0]$, and OWA^{AND} is the OWA operator corresponding to AND, i.e. $F = [0, 0, \dots, 1]$.

By employing the user interface described in the following section, a fine tuning of the OWA operator may be done in the following way. Instead of dealing with the particular weights separately, the user may request to increase or decrease the ORness of the currently defined operator which is done using the following simple algorithm (the algorithm shown applies when the increase of ORness is required, and analogously for the decrease. Let:

- the OWA operator F be defined by the vector of weights $[w_1, \dots, w_N]$, and
- z^0 be required, increased value of ORness measure for F ; $z^0 \in (\text{ORness}(F), 1)$.

Then:

$$\begin{aligned} \text{Step 1. } \Delta z &:= z^0 - \text{ORness}(F) \\ \text{Step 2. } k &:= \arg \max_i \{w_i : w_i > 0\} \\ x &:= 2(N - 1)\Delta z / (4N - 3k). \end{aligned} \tag{9}$$

Step 3. If $x > w_k$ then $x := w_k$

Step 4.

$$w_k := w_k - x \quad w_i := w_i + x / (k - 1) \quad \forall i \in [1, k - 1]. \tag{10}$$

Step 5. If $\text{ORness}(F) < z^0$ then Go to Step 1.

STOP

- *Fuzzy set constant*

Fuzzy set constant represents in a query the user's requirement as to the value of a single-valued attribute or a multi-valued attribute. Its use may be exemplified in the following atomic conditions:

1. COUNTRY IN 1.0/Bulgaria
2. COUNTRY IN 1.0/Belarus + 1.0/Russia + 1.0/Ukraine
3. COUNTRY IN 1.0/CzechRepublic + 1.0/Hungary + 1.0/Poland + 1.0/Slovakia + 0.8/Belarus + ... ,

where IN is a simple compatibility operator corresponding to the classical set theory operator \in .

The user, e.g., looking for a customer from Bulgaria only, will employ the first condition. If a few countries are relevant, the second condition may be used. Finally, if the choice of a customer's country of origin refers to a vague concept like, e.g., the Central Europe or the "developing countries", the third form should be employed.

- *Compatibility operators*

Compatibility operators make possible to express a relation that should be met by a single-valued attribute or a multi-valued attribute and a fuzzy set constant in an atomic condition.

The matching degree of an atomic condition involving a single-valued AT and a FS is calculated as equal to $\mu_{FS}(R(AT))$, where $R(AT)$ is the value of the attribute AT in a given record R .

Let FS (in a query) and D (in a database record) be two fuzzy sets defined in the same universe of discourse U , i.e., $FS, D \in F(U)$. Let $md(FS, D)$ denote a compatibility operator to be defined. Then the following definitions may be employed:

1. Degree of possibility of matching

For the general case when both FS and D are fuzzy, we have

$$md(FS, D) = \Pi(FS|D) = \sup_{u \in U} \min(\mu_{FS}(u), \mu_D(u)) \quad (11)$$

while in case when both these sets are crisp, we obtain

$$md(FS, D) = \begin{cases} 0 & \text{if } FS \cap D = \emptyset, \\ 1 & \text{otherwise} \end{cases} \quad (12)$$

and, finally, if $D = \{d\}$ is a single-element crisp set, then

$$md(FS, D) = \mu_{FS}(d). \quad (13)$$

2. Degree of necessity of matching

For the general case of fuzzy FS and D , we have

$$md(FS, D) = N(FS|D) = \inf_{u \in U} \max(1 - \mu_{FS}(u), \mu_D(u)) \quad (14)$$

while in case when both these sets are crisp, we obtain

$$\text{md}(\text{FS}, D) = \begin{cases} 1 & \text{if } \text{FS} \subseteq D, \\ 0 & \text{otherwise} \end{cases} \quad (15)$$

and, finally, if $\text{FS} = \{x\}$ is a single element crisp set, then

$$\text{md}(\text{FS}, D) = \mu_D(x). \quad (16)$$

3. Generalized Jaccard coefficient

For the general case of fuzzy and/or crisp FS and D , we have

$$\text{md}(\text{FS}, D) = |\text{FS} \cap D| / |\text{FS} \cup D| \quad (17)$$

and this is one of most popular operators used in the classical, crisp framework.

The matching degree for an atomic condition involving a multi-valued AT and a FS is calculated for each record in a straightforward manner as the value of the selected compatibility operator for FS and the element being the value of the attribute AT in a given record.

3. Declaring attributes and defining fuzzy terms

The user declares attributes and defines fuzzy terms by double-clicking appropriate push-buttons in the special toolbar (see Fig. 3). The definition of each fuzzy term boils down to the specification of a number of parameters which proceeds interactively.

The declaration of an attribute consists in adding an appropriate record to the table of attributes. A special form is displayed on the monitor screen (see Fig. 4). The user has to choose a table from which there comes the field serving as a base for an attribute being declared. To make it easier, a special menu listing all the tables in a given database (i.e. a currently opened “.MDB” file) is displayed. When a table has been picked up, the user has to select the field. Again, the list of all numerical fields in the table just selected appears on the screen. Finally, the user has to enter LL and UL, as discussed in the previous section.

Definition of a multi-valued attribute requires the selection of the set of fields from underlying table and assigning a name to it. Single-valued attributes do not require any special definition, i.e., any field of character (text) type may be used as a single-valued attribute.

In the case of fuzzy values, relations and linguistic quantifiers interaction with the user is quite similar. Appropriate records have to be added to tables.

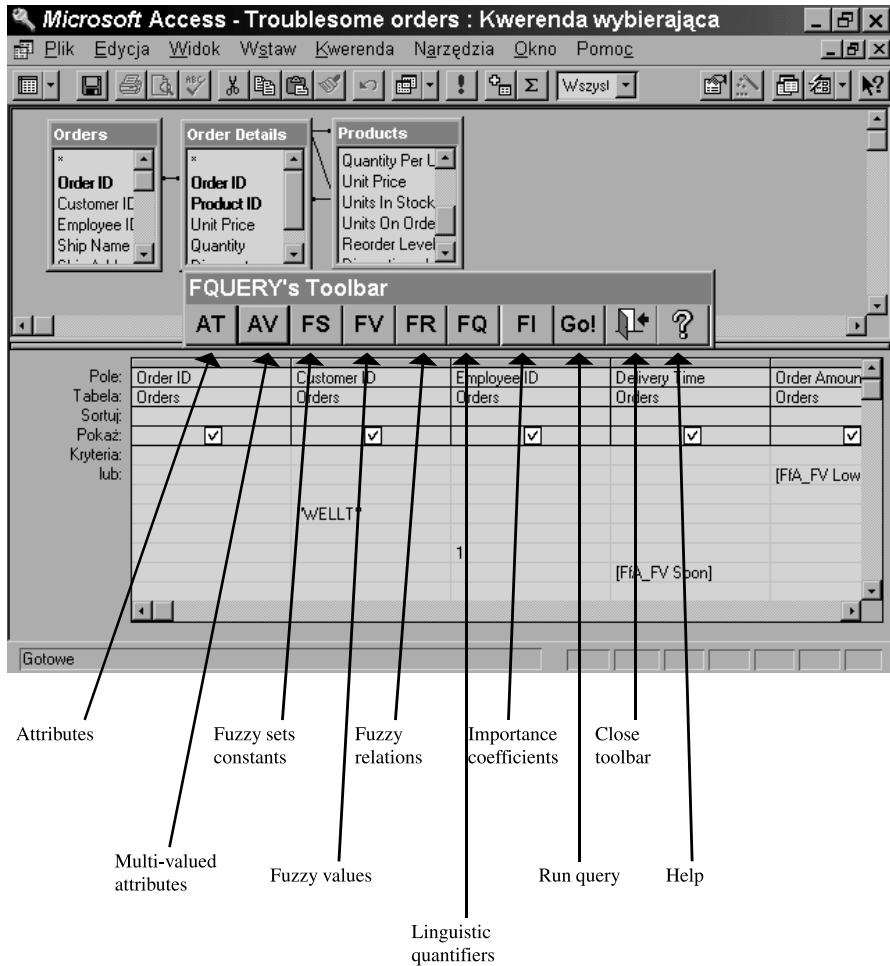


Fig. 3. FQUERY for Access's toolbar.

The fuzzy terms defined are illustrated by appropriate graphs corresponding to the Figs. 1 and 2 – see Figs. 5 and 6, respectively.

Defining a fuzzy set constant the user has to select a field for which it is planned to be used. Due to the fact, that fuzzy set constants may be employed in the context of single-valued as well as multi-valued attributes two separate registers of them are maintained. After choosing a single-valued attribute some of its values currently used in the underlying table have to be selected and assigned the membership degree. In case of multi-valued attribute some of its “component” fields have to be selected and assigned 0/1, Yes/No or belonging to the interval $[0, 1]$ values depending on the declared type of the multi-valued attribute.

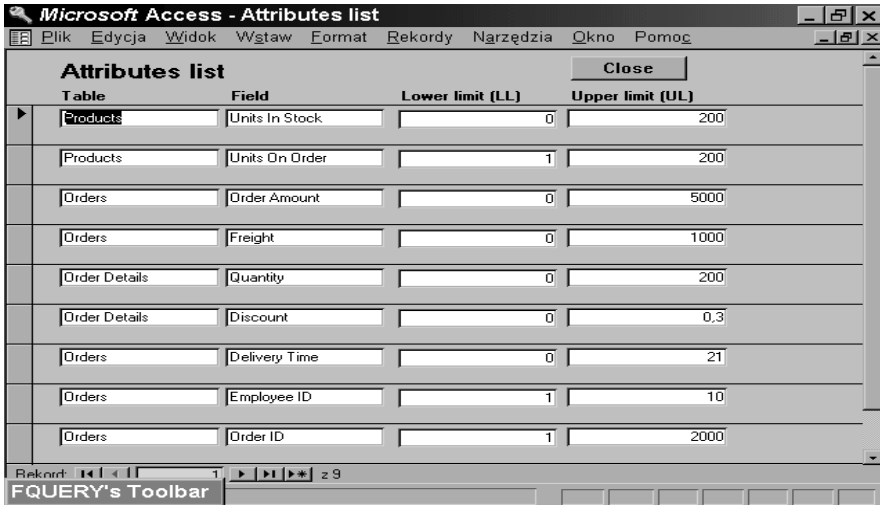


Fig. 4. Declarations of attributes.

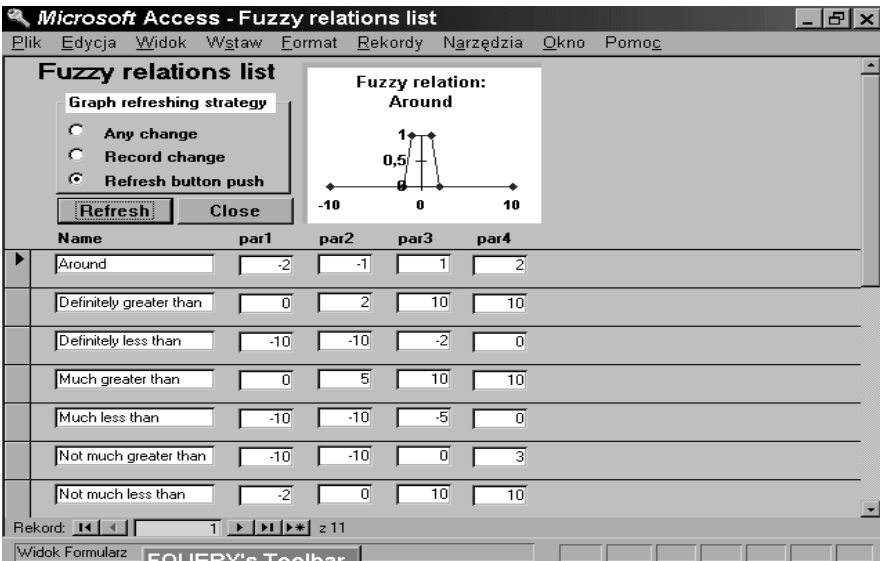


Fig. 5. Definitions of fuzzy relations.

4. Query construction and processing in FQUERY for Access

The user composes a query in the query design window, enhanced with tools provided by FQUERY for Access. It is a “legitimate” query of Access in which

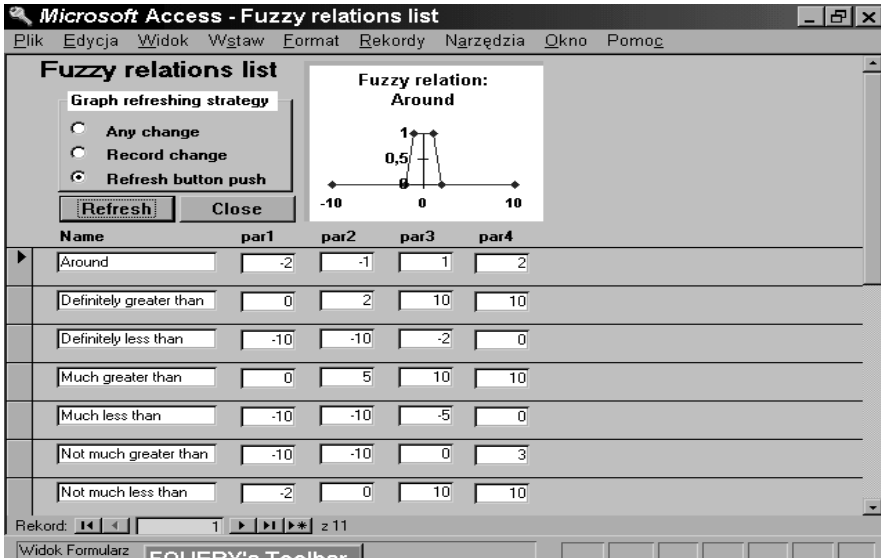


Fig. 6. Definitions of linguistic quantifiers.

fuzzy (linguistic) terms appear in the form of parameters. For example, if one wishes to use a fuzzy value labeled “Low” one needs to put a parameter [FfA_FVLow] into a criteria cell (the prefix FfA_FV is added to differentiate between “fuzzy” parameters and “normal” parameters which can still be employed by the user). All fuzzy terms are inserted from the pop-up menus provided by the FQUERY’s toolbar.

As already mentioned, the OWA operator may be introduced into the query through the selection of a linguistic quantifier. Thus, inserting a quantifier into the query, the user has to decide if the original Zadeh’s or OWA-based semantics should be used for that quantifier during the query execution, see Fig. 7. Obviously, a linguistic quantifier has to be defined before it may be inserted into the query. In order to make the interface more flexible in respect to the manipulation of OWA operators, we introduce still another possibility to handle them inside a query. Namely, if there is no linguistic quantifier specified by the user in a query, a *default* OWA operator is placed there. In particular, if a global linguistic quantifier is omitted, the OWA^{OR} operator is put into the query by default. On the other hand, if a linguistic quantifier is omitted in a subcondition, the default operator is assumed to be the OWA^{AND} operator (cf. (8)). These default OWA operators are not visible during the query construction. They are available for the user’s modifications only at the stage of fine-tuning of the OWA operators to be described later.

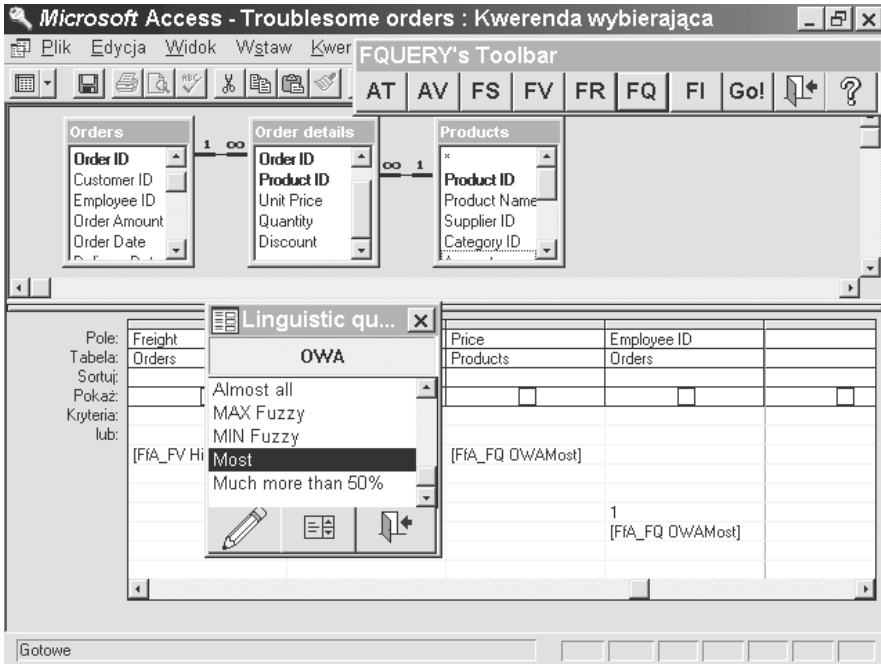


Fig. 7. The query construction screen while employing the OWA operators.

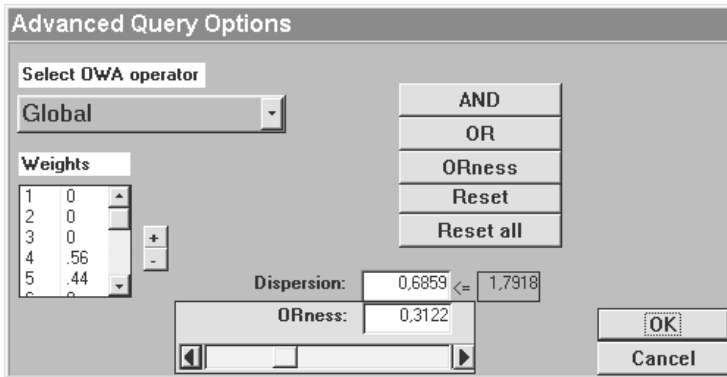


Fig. 8. Fine tuning of the OWA operators.

When the construction of the query is completed, the user can initiate the execution of querying by pressing the GO button, what is briefly described later on. Then, at the user’s request, an additional step may be performed during the processing of the query. Namely, the screen, shown in Fig. 8 is displayed

allowing the user to specify more precisely the particular weights of the OWA operator. At the top of this screen all OWA operators appearing in the query, including the default ones, are listed. The user has to select one of them and, then, the weights of this operator are displayed below. Initially, these weights are calculated according to (5) for the explicitly introduced linguistic quantifiers (in terms of their related OWA operators), or correspond to particular default OWA operators.

Then, the user can modify them in several ways:

- “manually”, by setting each weight separately,
- by pressing the AND button which sets the OWA weights to $[0, 0, \dots, 1]$,
- by pressing the OR button which sets the OWA weights to $[1, 0, \dots, 0]$,
- by automatically increasing/decreasing the ORness of the operator by a specified amount.

The last modification is performed by the system automatically due to (9, and 10).

Finally, after the GO button is pressed and fine-tuning of the OWA operators is finished, the query is automatically transformed, and then run as a native (legitimate) query of Access. The transformation (after the GO button is pressed) is done by the appropriate FQUERY’s routines according to the following rules in two main steps:

I. New, temporary query is created. Depending on the form of the conditions appearing in the WHERE clause of the original query, a call to FQUERY for Access’ function FfA_MD, is added to the SELECT clause of the new query along with the list of all fields used in the original query. The arguments in the FfA_MD function call correspond to particular conditions appearing in the WHERE clause of the original query. Together with the composition of the query recorded in FQUERY for Access’ internal data structure, this allows to compute a matching degree. For example, each fuzzy value appearing in a condition yields one parameter in the FfA_MD function call, corresponding to the actual value of an attribute involved in a given expression, e.g., the atomic condition `Products.[Units_In_Stock] = [FfA_FV Low]`, which means that the amount of a given product in stock is low, adds the value of `[Units_In_Stock]` in the current record to the list of parameters of the FfA_MD function call.

The INTO clause is added to the new query and the WHERE clause is left empty. Thus, we obtain the new query:

```
SELECT list_of_fields, FfA_MD(parameter) As MD INTO newTable
```

which is executed in the background. For each record, Access evaluates the arguments of the FfA_MD function call because they appear in the SELECT clause of the query. Using their values, a matching degree of the current record against the query is calculated.

II. The original query is transformed in the following way: all condition in the WHERE clause are replaced with `MD > threshold`, where threshold is a

value defined by the user. This means that the query will display only records matching its conditions to the degree higher than the threshold specified. The FROM clause is replaced with FROM newTable. The ORDER BY MD clause is added. Thus, the transformed query looks like follows:

```
SELECT list_of_fields, MD FROM newTable ORDER BY MD
```

As a result, each record displayed by the query is accompanied with the value of its matching degree (all fields selected originally are preserved in the SELECT clause).

So, after pressing the GO button the new, temporary query is executed and then the original SQL string of the query is translated according to the above rules and replaced by the modified one. Then, the query is run by Access as usually. Records matching the query to a degree higher than a prespecified threshold value are displayed along with the value of their matching degree. The SQL string in its original form is restored into the query, so that the user cannot even see its modified form on which the currently displayed information is based.

5. Example

Basically, we will consider here a modification of the standard example that is included in Microsoft Access – a slightly modified version of the NWIND.MDB database. Therefore, the users can run this example on their own, and – on the other hand – could find more information on the example in the documentation of Access.

Suppose we have a database of a small trading company and we wish to retrieve a list of *troublesome* order items requiring special attention. The first problem is vagueness of the linguistic term a “troublesome order item”. Suppose that in our case there are the following factors (conditions) that may imply potentially troublesome order item:

1. because the company is based in the USA, orders coming from abroad may require more attention (transportation formalities, customs duties, etc.),
2. a *short* delivery time,
3. a *low profitability* of the order, that in turn can be defined as having three sources:
 - a *low* amount of ordered product,
 - *high* freight charges,
 - *high* discount rates
4. the ordered amount is *much greater than* that now available in stock,
5. the order is placed by a certain customer (named, e.g., Wellt) with whom we have experienced some problems in the past,
6. an employee responsible for a given order (e.g., labeled 1) has recently had a few unsuccessful transactions.

Obviously the list of factors that should be taken into account depends on a given case and can be much longer. A requirement that *all* of the above difficulties (conditions) occur in full scale is often unreasonable. One can claim that an order fulfilling, e.g., *most* of the conditions listed above, may surely be treated as potentially troublesome.

One can easily recognize in the above formulation several linguistic (fuzzy) concepts supported by FQUERY for Access: fuzzy values (*short*, *low*, *high*), a fuzzy relation (*much greater than*) and a linguistic quantifier (*most*).

Let the following database tables and fields convey information required:

- In the table Orders information about the whole orders is stored; in particular the following fields are of interest for our purposes:

Orders.[Order ID] – identifier of an order,
 Orders.[Customer ID] – identifier of an customer,
 Orders.[Employee ID] – identifier of an employee dealing with the order,
 Orders.[Delivery Time] – time, in days, when the order has to be completed
 Orders.[Order Amount] – the value, in dollars, of the whole order,
 Orders.Freight – the freight fare for the order,

- In the table [Order Details] information about the particular items (ordered products) of each order is stored; among others, there are the following fields:

[Order Details].Discount – discount given for a given product,
 [Order Details].Quantity – the ordered amount of a given product,

- In the table Products information about particular products is stored:

Products.[Units In Stock] – amount of a given product in stock.
 Products.[Units On Order] – amount of a given product, required by all current orders.

As it has already been mentioned, to use fuzzy terms concerning a given database field one has to declare this field as an attribute in the sense of FQUERY for Access. Such declarations of attributes for our example may look as in Fig. 4. Next, we have to define fuzzy values and fuzzy relations which may proceed as in Fig. 5. Finally, we need definitions of linguistic quantifiers involved. They are shown in Fig. 6.

Then, the query considered may be expressed in SQL of Microsoft Access as:

```
SELECT DISTINCTROW
Orders.[Order ID], Orders.[Customer ID],
Orders.[Employee ID], Orders.[Delivery Time],
Orders.Amount, Orders.Freight,
[Order Details].Discount, [Order Details].Quantity,
Products.[Units In Stock], Products.[Units On Order]
```

FROM

Orders INNER JOIN [Order Details] ON
 Orders.[Order ID] = [Order Details].[Order ID],
 [Order Details] INNER JOIN Products ON
 [Order Details].[Product ID] = Products.[Product ID]

WHERE

((Orders.[Ship Country](<'USA')) OR
 ((Orders.[Delivery Time] = [FfA_FVSoon])) OR
 ((Orders.[Order Amount] = [FfA_FVLow]) AND
 (Orders.Freight = [FfA_FVHigh]) AND
 ([Order Details].Discount = [FfA_FVHigh]) AND
 ([Order Details].Quantity = [FfA_FQMost])) OR
 ((Products.[Units On Order] = [FfA_FRMuch greater than – Products:Units
 In Stock])) OR
 ((Orders.[Customer ID] = “WELLT”)) OR
 (([Order Details].Discount = [FfA_FQA Lot of])) OR
 ((Orders.[Employee ID] = 1));

In the QBE window of Access this query looks as shown in Fig. 9 (only a part of it is visible).

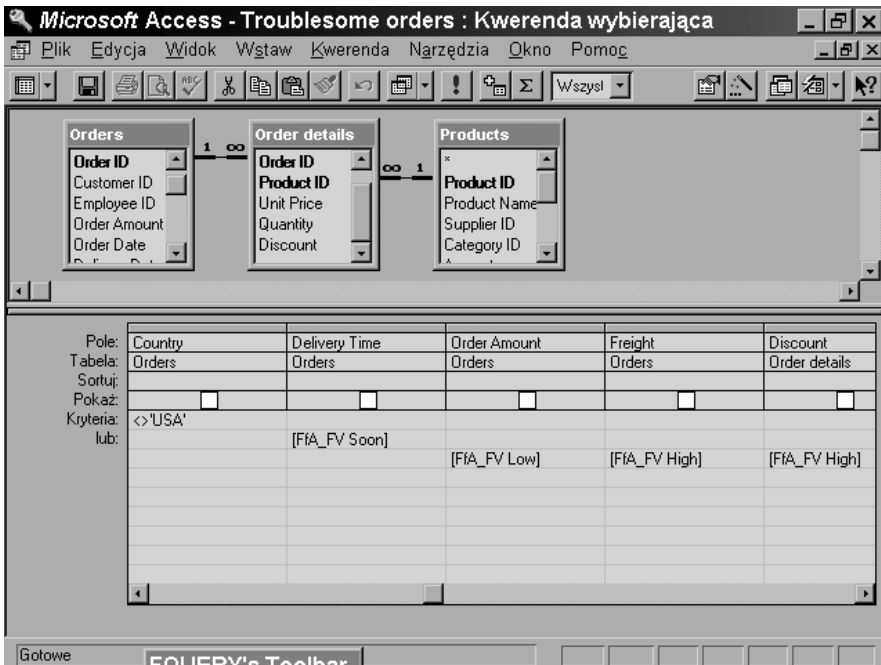


Fig. 9. Composition of a query.

The screenshot shows a Microsoft Access window titled "Microsoft Access - [Select Query: Troublesome orders]". The menu bar includes File, Edit, View, Records, Layout, Window, and Help. The toolbar contains icons for Find, Print, and Help. The main area displays a table with the following columns: Order ID, Customer ID, Employee ID, Delivery Time, Order Amount, Freight, Discount, and Units On Ord. The data is sorted by Order ID in descending order. The status bar at the bottom shows "Record: 1" and "Unique order number from invoice."

Order ID	Customer ID	Employee ID	Delivery Time	Order Amount	Freight	Discount	Units On Ord
10900	WELLT	1	5	\$33.75	\$1.66	25.00%	1
10061	EASTC	1	1	\$142.80	\$4.34	0.00%	10
10773	EMPIT	1	4	\$2,030.40	\$96.43	20.00%	7
10968	EMPIT	1	4	\$1,408.00	\$74.60	0.00%	8
11027	BOTTM	1	3	\$877.73	\$52.52	25.00%	
10690	HANOP	1	2	\$862.50	\$15.80	25.00%	
10690	HANOP	1	2	\$862.50	\$15.80	25.00%	1
11027	BOTTM	1	3	\$877.73	\$52.52	25.00%	
10400	EASTC	1	3	\$3,063.00	\$83.93	0.00%	6
10859	FRASD	1	9	\$1,078.69	\$76.10	25.00%	8
10011	WELLT	6	6	\$589.05	\$31.54	25.00%	
10011	WELLT	6	6	\$589.05	\$31.54	25.00%	
10717	FRASD	1	2	\$1,270.75	\$59.25	5.00%	4
10224	RICHS	1	7	\$303.75	\$24.58	25.00%	1
10182	DUNNH	1	5	\$272.00	\$20.06	0.00%	4
10792	WDLVH	1	1	\$399.85	\$23.79	0.00%	4
10068	PICAF	1	2	\$312.55	\$19.15	5.00%	4
10224	RICHS	1	7	\$303.75	\$24.58	25.00%	
10567	HUNGO	1	8	\$2,519.00	\$33.97	20.00%	7
10258	EMPIT	1	6	\$1,614.88	\$140.51	20.00%	4
10420	WELLT	3	1	\$1,707.84	\$44.12	10.00%	1
10258	EMPIT	1	6	\$1,614.88	\$140.51	20.00%	4
10800	SEVES	1	5	\$1,468.94	\$137.44	10.00%	3
10542	KINGG	1	5	\$469.11	\$10.95	5.00%	3
11038	SUGAA	1	5	\$732.60	\$29.59	20.00%	
10022	LAPLA	1	1	\$671.50	\$46.00	25.00%	6

Fig. 10. Results of the query.

After pressing the GO button (and selecting the ordering according to the matching degree value and the threshold equal to 0.4) the above query is transformed (“behind the scene”) according to the rules presented earlier.

As a result we obtain the list of records shown in Fig. 10.

This concludes our brief description of FQUERY for Access, an add-on to Microsoft Access which, by using some tools available in the context of computing with words, makes it possible to effectively and efficiently handle imprecise and vague linguistic terms in database querying.

6. Computing with words in fuzzy querying in DBMSs via the Internet

The FQUERY for Access is a standalone application. Its capabilities may be considerably enhanced by allowing it to be used via a WWW browser as proposed and implemented by [10]. This will hopefully be a first step in a more human-consistent and human friendly use of vast information resources available there which are so difficult to locate and used for a normal user whose only natural means of communication and expression is natural language. Fortunately enough, fuzzy querying in the sense described in previous sections can also be implemented for querying over the Internet, and this will be

presented in the next sections. We will start with some general remarks, and this part is based on [21], and then present implementations as proposed in [10,20].

6.1. General remarks on fuzzy querying over the Internet and its implementation

Although DBMSs may be useful for dealing with data for individual use, as it has been the case in standalone fuzzy querying described in previous sections, only a possibility to share data with other users over a network may release full power from a DBMS. Needless to say that a global network spanning the whole world, i.e. the Internet, creates virtually unlimited opportunities for a user, say a company, to provide its data to a general public.

The most popular solution for accessing a database over the Internet is via the World Wide Web (WWW) service. WWW creates a new framework for applications following the client–server paradigm. This framework features, e.g., a cross-platform availability, cheaper software maintenance and a unified user interface. With the advent of WWW and fast development of its underlying technologies, the population of potential database users has grown to an unprecedented scale. Owing to the fast development of telecommunication infrastructure and sophisticated, but relatively cheap software tools, notably Web browsers, the access and navigation through the Internet become easier and easier. More and more people consider the Internet as an important source of information being competitive with traditional media. Moreover, this new medium is more flexible, allowing to interactively specify the users' needs.

A growing popularity of the Internet spreading across whole societies inevitably changes the profile of a typical network user. Such a user is no longer an IT expert. He or she expects more human consistent user interfaces making it possible to communicate with information servers using, in addition to graphical controls, also flexible terms from natural language. As DBMSs will surely still be the workhorses of many WWW-based applications, the need for simple, highly flexible and human-consistent interfaces to databases seems to be obvious. One of possible approaches is certainly the one employing the computing with words paradigms, in particular those related to fuzzy logic and fuzzy sets theoretic tools.

The WWW service follows the client–server architecture, i.e. there are basically two types of WWW-related applications: the WWW server and the WWW client. The WWW server, also known as a HTTP server, may be regarded as a kind of a file server providing on request various files (documents, also known as pages), primarily those written in the HyperText Markup Language (HTML), the native format of WWW.

Currently available implementations of the WWW servers provide a much more extended functionality. The most important feature is the ability to

execute other applications and deliver to the user appropriately encoded results (e.g., by using HTML). Thus, the WWW server may also be treated as an application server. This is attained by particular software vendors using different techniques. Most of the servers support a mechanism, called a Common Gateway Interface (CGI). This approach is standard, but rather ineffective in terms of the host computer's resources usage. The proposed replacements, on the other hand, are specific for particular products.

The WWW browser is a client application for this Internet service. The user initiates the connection providing the browser with the address of a required document, a so-called Uniform Resource Locator (URL), and the browser forms and sends to the WWW server a request for this document. Both, the required document and the server address are specified within the URL. The communication is regulated by the HTTP protocol. Then, the server checks if a required document exists, and if so, delivers it to the browser. The browser renders the document to the required form by, e.g., interpreting HTML commands (tags) contained in the text, and displays the document on the user's screen. HTML allows marking selected parts of a document text so as to inform the browser that this parts should be emphasized, underlined etc. The most characteristic feature of the HTML are hyperlinks allowing the user to quickly "jump" from a given document to other related documents, somewhere on the WWW.

To provide for a more dynamic communication in the WWW environment, some enhancements were introduced to the basic request-and-display rules. First of all, to make the aforementioned CGI protocol really useful, some means of sending input data for server-based CGI-compliant applications have to be provided. This may be done through an extension to the HTML language, so-called *forms*. The browser displays a HTML form using Microsoft Windows' *controls* or X-windows' *widgets*, such as a push button, listbox etc. The data entered using these controls will be sent to the server along with the request for a document – usually referring to some CGI application. Then, the server will fire this application with the data provided by the user as an input. Such data may be, e.g., a query which is used by the above CGI-compliant program to fetch matching records from a database. The server catches the output of the executed application and sends it back to the browser.

Thus, the support of HTML forms provides a basic level of interaction which may be sufficient for the implementation of a database querying interface including fuzzy extensions – see, e.g., [16], but still lacks some important features. For example, a browser-side verification of the data introduced by the user is not assumed in this approach. In consequence, e.g., the user's spelling errors may be discovered only at the server-side and their correction will be connected with some extra network transmissions.

Forms have become part of the HTML standard soon after its introduction and are now widely recognized by different browser implementations, even in a

non-GUI environment (cf. the Lynx browser). Recently proposed enhancements are much more sophisticated though much less standardized. Three of the most important should be discussed here. All of them allow to embed in a HTML page a code, which is to be executed on the browser-side rather than on the server-side, as it is the case of CGI-compliant applications. The simplest technique consists in the use of scripting languages, such as Microsoft's Visual Basic Script or Netscape's Javascript. The routines written in these languages and embedded in the HTML text, although rather restricted in their capabilities, are powerful enough to serve as a glue between all of the elements of the user interface provided by a browser. For example, they can check data entered by the user in a form and signal potential mistakes already at the browser-side, thus avoiding unnecessary network transmissions.

The second novel technique is connected with the Java language. It is a general purpose programming language designed to meet the requirements of a distributed environment such as the Internet. Thus, specially prepared Java programs, so-called *Java applets*, may be embedded in the HTML text. They are more powerful than the script routines, however still somehow limited in comparison to standalone Java programs, for security reasons. Anyway, using Java applets a developer can create advanced user interfaces while still taking advantage of working within the WWW environment. Java is designed to be platform independent and is supported by leading software vendors.

Finally, the third hot Internet solution is referred to as *ActiveX controls*. Functionally, they are similar to Java applets in that ActiveX are small programs which may be embedded in a HTML page. On the other hand, from the software engineering point of view, they are quite different. The ActiveX controls are sent over the network in the binary ready-to-execute form, thus are faster in execution than applets which have to be interpreted by a Java machine (or at least initially compiled at the browser side). However, the ActiveX controls are currently limited to the Microsoft Windows platform, while Java applets are working also on other platforms, notably UNIX.

Basically, the HTTP protocol used for communication between WWW agents is state-less making it possible to keep the protocol simple. On the other hand, there is no built-in mechanism to employ the concept of a session which is quite important for some applications, including database querying. Some remedy for this drawback is offered by so-called *cookies*. A cookie is a piece of text which may be sent by a server with the document requested by a browser. Whenever the browser requests a related page from the same server, it adds to the request the cookie previously received. This makes possible to maintain the state of a session.

While the core of the HTML language is pretty well standardized and recognized by all WWW browsers, it is unfortunately no longer true for the extensions mentioned above.

6.2. *Integration of a DBMS with WWW*

The diversity of existing WWW-related software and protocols implies diverse solutions for the integration of a database with a WWW service. A WWW browser might be considered as an attractive alternative for specialized database clients as mention in Section 6.1. We can expect that this way of accessing a database over the Internet (or Intranet) will gain more and more popularity. Before looking for opportunities and challenges provided by a fuzzy logic based approach in this respect, we will first try to classify the existing solutions for the integration of WWW and DBMS. Basically, the criteria for this classifications will refer to the way the data flow is arranged between all the agents involved, i.e. the WWW server and browser, DBMS software and other intervening pieces of software. We will trace the flow of information starting from the browser side.

First, we can differentiate the solutions taking into account the division of work done at the browser and server sides. Thus, we can distinguish the approaches where:

- the browser side processing of data is kept at the minimal level, or
- a preliminary processing is already done at the browser side.

The former approach allows for full independence of the type of browser used by the end users. Only a basic level of interaction with the user is maintained at the browser side, usually through the HTML forms. The data is checked and processed only at the server side. Thus, some additional data transmissions may be required in case of errors in input data or if a more complex user interaction is necessary (cf. [16]). Anyway, this solution may be adequate when the targeted population of the end users is large and/or spatially and organizationally distributed, i.e. no assumptions may be made about the type of browsers used. The latter approach requires the use of, e.g., scripting languages routines, which will carry out some preliminary data checking and maybe also provide some more advanced functionality as, e.g., operations on cookies – see [20].

Second, the access to a database may be accomplished as follows:

- within the regular communication between the WWW server and the client,
or
- using a separate connection specifically arranged for that purpose.

The first solution would usually consist in using a HTML form on the browser side and some additional application executed by the server. An advantage of such an approach is its portability – virtually all browsers and WWW servers support forms and some ways to execute external applications, respectively. The second solution may be accomplished using a Java applet or an ActiveX control communicating directly over the network with a target DBMS. It may be much more efficient but there are also some aspects which should be carefully considered. For example, from the viewpoint of security, applets and

ActiveX controls in particular are considered potentially dangerous. Obviously, the question of security is more general and there are tremendous efforts to solve it so that we can expect some satisfactory solutions fairly soon. On the other hand, this approach has little to do with a real integration of WWW and DBMS. In fact, the WWW browser plays here a rather marginal role and may be substituted by, e.g., a standalone Java application.

The third issue is related to how a WWW server cooperates with some “middle-ware” usually sitting between the server and DBMS. Here, two main classes may be distinguished:

- using the CGI protocol,
- using specialized *Application Programming Interfaces* (APIs).

The former solution is supported by virtually all WWW server software (in Microsoft Windows environment its variant, called *WinCGI*, is sometimes employed). CGI-compliant applications may be written using popular languages, like C, Perl, script languages etc. but this way of executing an internal application by the WWW server is considered inefficient. The latter solution extends the WWW server dynamically so as to provide a required functionality – in this case to make possible an access to the database. This is a more efficient solution but the standard for APIs does not exist.

Many commercial DBMS packages offer now some means for the integration of their products with the Internet. On the other hand, some WWW server software vendors include in their products some tools essentially facilitating the integration with a database (see, e.g., the Microsoft Internet Information Server and Active Server Pages mechanism, or the AOL server with its SQL Server or Illustra DBMSs connectivity).

6.3. Processing of linguistic data via fuzzy logic over the Internet

In this paper fuzzy querying is meant in the spirit of the previous works of the authors exemplified by [14,17]) and discussed in the previous sections. Basically, the vocabulary of querying languages is extended with linguistic (fuzzy) terms to allow for query conditions to contain clauses like: ‘temperature is high’, ‘income is much greater than expenditures’, ‘most of subconditions are met’, etc. Then, these terms are to be interpreted during the query evaluation against subsequent records from a database. As a result we obtain a *matching degree* stating how well a given record meets conditions specified in the query. Hence, we need some methods for the modelling, definition and maintenance of fuzzy terms. The former issue is dealt with extensively in the previous sections. Basically, fuzzy sets, fuzzy relations and linguistic quantifiers are used to represent particular linguistic terms. This approach may be directly applied also in the Internet environment. On the other hand, the ways particular fuzzy entities are defined and stored depends on the user interface capabilities and general structure of the querying system. Thus, in case of an Internet-based

querying application we have to devise appropriate procedures fitting the specifics of the WWW, notably those discussed in the previous sections. Following the criteria used for the aforementioned classifications of the WWW–DBMS integration approaches, we will now briefly look how the processing of fuzzy terms may be accomplished.

Obviously, the maintenance of fuzzy terms, i.e. their definition, storage and interpretation, requires some additional measures to be undertaken in comparison to a non-fuzzy querying system. It may be advantageous to have a more sophisticated user interface provided already on the browser side. For example, the definition of a fuzzy set representing a fuzzy value or linguistic quantifier, requires a membership function to be defined. It may be rather cumbersome to create a sufficiently flexible user interface for that purpose using the HTML forms only. Certainly, this is the only solution if full independence of the browser software capabilities is sought. On the other hand, using, e.g., a Java applet may make the interface more user friendly, cf. [20] for an example.

The definitions of linguistic (fuzzy) terms may be stored on the server or browser side. The first solution is more consistent with the WWW standards. The parameters of particular fuzzy terms may be stored in the same database as the data to be retrieved, in a separate database or even in flat files. It should be relatively easy for a CGI- or API-compliant application to handle this information at the server side. It may be a little bit cumbersome to achieve personalization of fuzzy terms in this approach. All definitions have to be stored at the server side so that, in case of multiple users, this may require an excessive amount of extra disk space. Thus, the storing of this information at the browser side would solve the question of personalization. Unfortunately, the manipulating of data on the local disk at the browser side is considered as risky for security. Presently, this may be achieved by using the ActiveX controls or cookies. The first option provides a full access to a local file system and may become acceptable in the future when security problems will be solved. The second option is fairly secure but local data handling is rather limited and even here standards do not exist.

Anyway, the parameters of available fuzzy terms, predefined or defined by current and/or other users, have to be embedded in the user's querying interface. This may be done through HTML form controls or Java applets and ActiveX controls parameters. In case of HTML forms, it may be convenient to store definitions of fuzzy terms in cookies and/or so-called controls properties. They are not visible to the user but their content may be easily manipulated by, e.g., a JavaScript script.

The actual database search and calculation of matching degrees for the particular records against a query containing fuzzy terms may take place at the browser or server side. The former alternative results in a solution rather loosely connected with the WWW services and we will omit it here. The latter

approach may be implemented using a CGI-compliant application (see, e.g., [16]) or a WWW server extensions using API (see, e.g., [10]). From the viewpoint of functionality of this “middle-ware” (gateway) application, the following classification may be proposed:

- the gateway fetches the rows from the database and only then evaluates the query against them,
- the handling of fuzziness is built into the DBMS, and the gateway just forwards the query and its results,
- the gateway itself manages the database, and the handling of fuzziness is a part of its regular query processing.

The first approach may be applied for virtually any DBMS. The whole fuzzy querying logic has to be implemented in the gateway, and also possibly in the optimization of the query sent to the underlying DBMS. The second approach would be definitely most efficient though commercial DBMSs accepting fuzzy queries are not widely available yet. For example of such a solution one can consult [10]. The third approach may be well-suited for simple databases as, e.g., an XBase file (see [20]).

Finally, results of the query are to be displayed. Here, the features of the HTML perfectly fit the needs. The results may have the form of a list where each line consists of a value of a key field of the record with its matching degree. Each such a line contains a hyperlink which brings to the screen the content of the whole record.

We will now present an implementation of fuzzy querying over the Internet using the paradigm of computing with words, and some WWW-related tools and techniques mentioned above.

7. Implementation of fuzzy querying over the internet

We will briefly describe now two linguistic (fuzzy) querying interfaces based on the WWW, and show that tools available in WWW, in particular Java applets and JavaScript, may be effectively and efficiently employed for implementing advanced, multi-platform and user-friendly fuzzy-logic-based querying and other applications following general remarks provided in Section 6, and the philosophy and solutions adopted in our FQUERY for Access add-on described in Section 2.

The first interface, employing advanced client-side techniques will be illustrated on a WWW-based fuzzy querying interface for a database of Research, Technology and Development (RTD) institutions from Central and Eastern Europe (cf. [16,20]). The second Web based interface presented makes it possible to submit a query over the Internet to the fuzzy querying engine implemented by the FQUERY for Access add-on, and to display the results back on the user screen in the window of a WWW browser (cf. [20]).

We will start with the description of the first interface and the second will be presented in Section 7.4.

The DBMS is still the main element of a fuzzy querying system considered here. It may be a full-fledged, network-based product like Oracle, Sybase, Microsoft SQL Server etc. This will be the best choice for large systems serving many customers. For intranet-oriented systems, much smaller DBMSs may be sufficient as, e.g., Microsoft Access. Most of the existing commercial DBMSs offer now means of the integration with the Internet. It confirms a wide recognition of the importance of this network technology by leading vendors of DBMSs. For our implementation we use a relatively simple database consisting of a set of Xbase format and text files.

The second element of our fuzzy querying system is a WWW server which follows a simple CGI-based scheme in order to access the database. More precisely, a CGI-compliant C program performs the actual querying of a Xbase file, including the processing of fuzzy terms. Obviously, it has to be provided by a WWW server with a query which in turn is formulated by a user and sent to the server using the next component, i.e. a WWW browser.

In our first pilot implementation we have chosen the Netscape Navigator but a similar functionality may be attained by using the Microsoft Explorer which was also used in further cases, and an example will be mentioned here too. As the processing of fuzzy terms essentially increases the scope of interaction with a user, the limitations of standard HTML form serious obstacles for creation of an effective and efficient WWW-based fuzzy querying interface. Thus, we will employ some new enhancements of the standard HTML mentioned earlier in Section 6.

7.1. The database component

The core of our first implementation of fuzzy querying over the Internet is a relatively simple database, an XBase file, which contains descriptions of some RTD institutions from Central and Eastern Europe constructed as a result of a research project for the support of cooperation between the scientific communities of the European Union and other European countries. Each organization in the database is described by many attributes (fields).

For our fuzzy querying system we selected the following attributes which demonstrate the use of various fuzzy terms in queries and may be useful for real querying:

1. number of employees (*numerical*),
2. country (*single-valued*),
3. subjects of activity related to RTD (*multi-valued*),
4. specific topics/know-how (*textual*).

The first field is numerical. Regular (non-fuzzy) queries referring to this field would use it with a classical relational operator as \langle , \rangle , $=$ etc. Our fuzzy

querying scheme extends them with fuzzy values and fuzzy relations, and then fuzzy linguistic quantifiers. Here we will only illustrate them with examples of simple queries exemplified by: “Number of employees IS *small*”, “Number of employees IS *much greater than 100*”.

The second field is a text (string). Typically, a non-fuzzy query may contain a test for equality with some fixed text (here: the name of a selected country of origin of the RTD organization sought). However, very often (also here) it may be natural to pose a question like: “Country IS IN *Central Europe*”. The concept of “Central Europe” should be represented by a set, preferably fuzzy, as one can claim, that, e.g., Slovenia belongs to this set but only to some degree.

The third field is a virtual one, represented by a set of logical (Yes/No) fields. More precisely, there is a separate field corresponding to each possible subject of activity related to RTD in a given organization. A query may refer collectively to all these fields. Thus, we propose to treat all this fields as one, virtual multi-valued field. Then, its value would be a set of subjects of activity and, consequently, in a query it may be compared to a, possibly fuzzy, set. Dealing with a set of values, both in a record and a query, calls for soft measures of matching as the requirement that both sets are exactly equal is usually too rigid. Thus, we employ a compatibility index which measures how well a particular record fits a query in respect to a field of this type – cf. Section 2.

Finally, the fourth field is typical for information retrieval, i.e. it contains purely textual data. We do not propose any fuzzy extensions in this respect. There are promising approaches with fuzzy thesauri to enhance querying capabilities, and they may well be applied in Internet-based systems requiring some additional processing, mainly at the server-side.

7.2. The browser side processing

A WWW browser provides a framework for the user interface of our querying system. Basically, we use the following features:

- HTML forms,
- Java applets,
- JavaScript scripting,
- frames, and
- cookies.

As to the frames, basically they make possible to divide the screen into more or less independent parts which may display different documents. At the same time, the script code embedded in a document in one frame may manipulate other frames, i.e. make them display some documents or even dynamically create new documents in other frames. This is very convenient for our purposes.

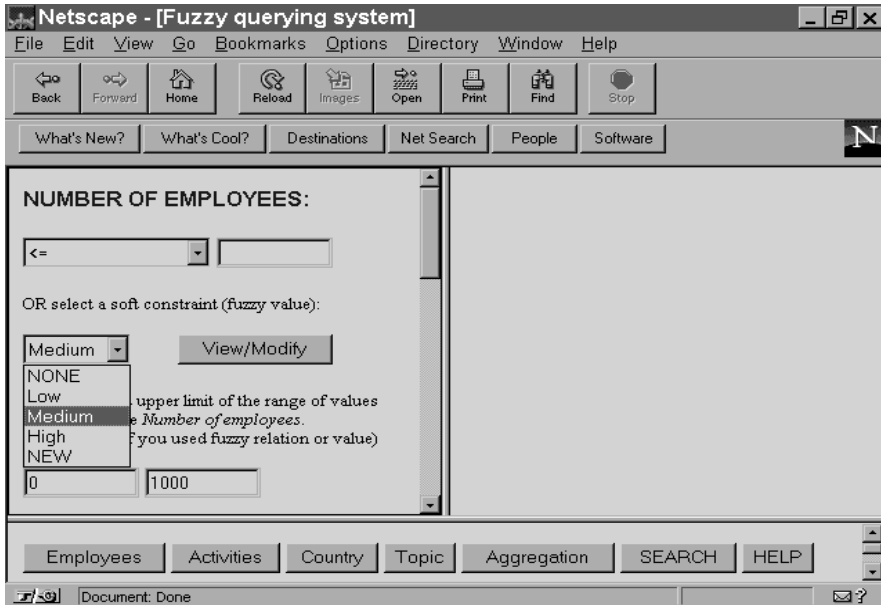


Fig. 11. Main screen of the interface.

The main screen of our system is shown in Fig. 11. There are three frames:

- the main frame, at the left, for specifying query criteria,
- the second frame for specifying membership functions for a fuzzy set employed in a query and for displaying help information,
- the third frame, at the bottom, for quickly moving around the first frame.

In the main frame there is a HTML form with separate controls corresponding to particular fields mentioned in Section 7.1. Now we will briefly discuss how they may be associated with various fuzzy terms.

The numerical field “Number of employees” may be used in a query with fuzzy values or fuzzy relations. Fig. 11 shows how a fuzzy value may be introduced into a query. Thus, the user may select a fuzzy value from a list which may be easily manipulated via a GUI. The content of the list depends on the history of previous interactions with the system. If a user contacts the system for the first time – what manifests itself with the lack of the cookie at the client side – the list is short as in Fig. 11. The user can also define “own” fuzzy values which are stored at the browser side within the cookie and are available during next sessions. The definition is initiated by selecting the “NEW” option and is supported by a graphical interface shown in Fig. 12. This is implemented as a Java applet.

The definitions of fuzzy values, relations and quantifiers are stored and defined similarly using cookies and Java applets, respectively. Thus Java applet, when used to modify the parameters of an existing fuzzy term will fetch its

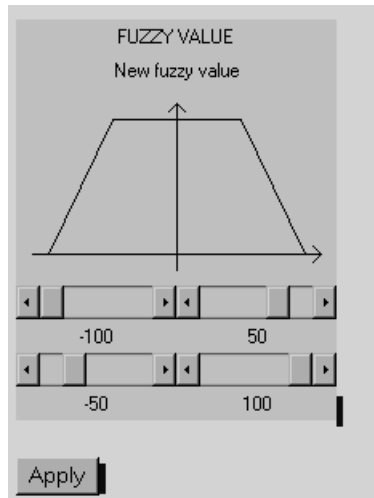


Fig. 12. Java applet for the definition of a fuzzy value.

parameters from the cookie and store them after modification at the same place. Hence, the definitions of fuzzy terms each time will go to the server along with the query.

The single-valued attribute “Country” may be manipulated by the user in a similar way as the multi-valued “Subjects of activity” (cf. Fig. 13). As the latter one requires more user-interaction, we will focus on it. The user looking for an RTD organization active in some areas should select names of these areas in the main frame. Then, if the user is interested in various areas to a different degree, he or she may fuzzify the set of selected activities. For doing this, an appropriate button is to be pressed and the list of earlier selected activities will be displayed in the right frame along with the textboxes for changing the membership degree value from (assumed initially) to a chosen one. Then, after the “OK” button is pressed, values assigned to particular members of the set are stored “inside” the HTML form displayed in the main frame. The VALUE property of the HTMLs SELECT control is employed to store these values and send them back to the server with the query.

As mentioned earlier, this type of fields requires some compatibility indices. An appropriate index may be selected from another control located below the ones described. The following indices are currently available in the system:

1. degree of possibility of matching,
2. degree of necessity of matching 1,
3. degree of necessity of matching 2,
4. generalized Jaccard coefficient.

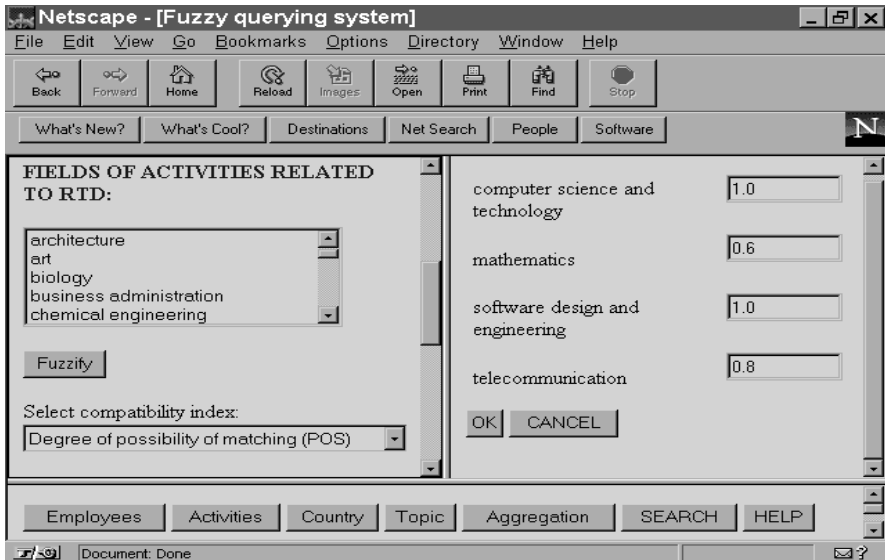


Fig. 13. Multi-valued attribute and fuzzy set constant.

In case of our database, the following reasoning may help choose a proper compatibility index. The first should be used when looking for an organization dealing with at least one subject of activity specified in a query. The second one is appropriate, when looking for an organization dealing with all activities specified in a query (and maybe some more). The third one is suitable for the search of an organization dealing only with activities specified in the query, not necessarily all. Finally, the fourth one is adequate when looking for an organization dealing with most activities specified in the query and not many other.

The last field, that may be used in the construction of a query, i.e. “Special topic (of activity)/know how” does not require any special treatment. The user just enters some keywords in a textbox which need no preprocessing or manipulation at the browser side.

Classically, all conditions specified for the particular fields mentioned would be combined using the AND and OR logical connectives. In our querying system we allow for more flexible aggregation operators, representing linguistic quantifiers. Thus, the user may request that, e.g., “most” of the conditions are met. These linguistic quantifiers are represented and interpreted due to Zadeh (cf. Section 2). As in case of other fuzzy terms, there is a list of linguistic quantifiers, or more generally a list of the aggregation operators – including the classical logical connectives AND and OR – available to the user. The user may define “own” quantifiers by an interaction with a graph implemented using a Java applet – cf. Fig. 14.

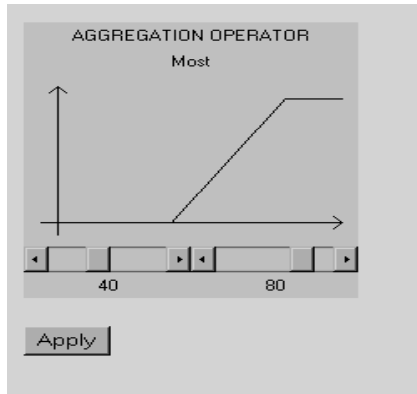


Fig. 14. Java applet for the definition of a linguistic quantifier.

7.3. The server side processing

At the server side we can distinguish two types of operations:

- a regular communication with a WWW browser due to the HTTP protocol done by the WWW server, and
- the generation of dynamic HTML documents and processing of a query by a set of CGI-compliant programs.

The whole processing is initiated by a browser requesting the “main document” of our querying system containing the query form, as shown in Fig. 11. The browser request may be accompanied by the cookie comprising the definitions of user-specific fuzzy terms. The server simply sends the query form to the browser together with the cookie if one was present in the request. Now the browser side processing takes place, as described in the previous section. Finally, the user presses the “SEARCH” button and all information entered by the user is transmitted to the server. Then, the server initially processes the data according to the CGI specification and calls our main C program which makes the actual query processing and database search.

In fact, all information about fuzzy terms possibly used in the query are contained in the data forwarded to this program by the WWW server. Thus, it has only to extract particular pieces of data and place them in its internal data structures and then start the search of the database. It fetches subsequent records and computes the matching degree. A partial fulfillment of the criteria by a record is possible. We start with the matching of particular conditions against the values of the attributes in a record. Then, the partial matching degrees are combined using an aggregation operator and we obtain an overall matching degree sought. Then, the records are ordered according to their matching degrees, and finally a HTML document containing the list of hyperlinks to the

particular records is produced. More precisely, each hyperlink consists of the matching degree and the content of a pre-specified field of the record. All these hyperlinks lead to another C program which delivers the full content of the records selected (cf. Fig. 15).

To summarize, at the server side we have the following components of our fuzzy querying system:

- a database file in the XBase format,
- HTML pages for: query form (enhanced with JavaScript functions and applets), list of records and the contents of a selected record,
- the searching program, and
- the program displaying (reporting) records.

7.4. WWW-based interface to FQUERY for Access add-on

Briefly speaking, the Web based interface presented in this paper accepts a query containing fuzzy elements according to the syntax presented in Section 2 and submit a query over the Internet to the fuzzy querying engine implemented by the FQUERY for Access add-on. Then, the query is processed in exactly the same way as in the standalone case. The results of the query are send back to

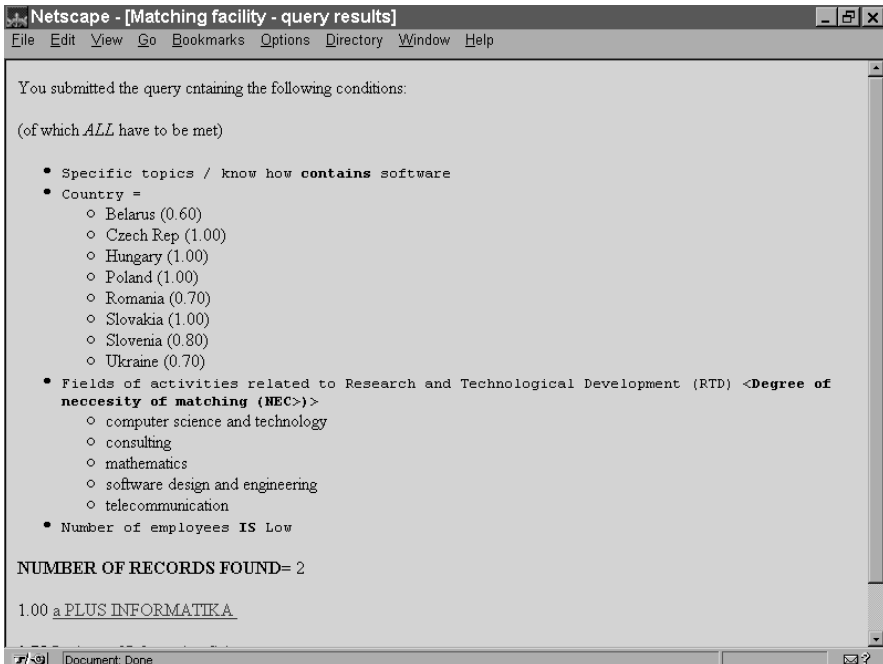


Fig. 15. Example of querying results.

the client and displayed by the browser. The process of query formation and processing may be described in the following steps using an example for querying a database of houses for sale in a real estate agency. Basically, in that kind of application virtually all customers' requirements concerning preferred apartments and houses are imprecise as, e.g., "possibly low price", "more or less two bedrooms", "in the city center", etc. Due to lack of space, we will just present a simple example of how a single imprecise condition "price should be low" is formulated using the Explorer, and then how the results of such a query, i.e. the addresses of apartments and or houses with their major descriptions and the values of a matching degree, are displayed via the Explorer's window.

Firstly, the user, using a WWW browser, opens the main document of the querying system shown in Fig. 16. It contains a simple query form, making it possible to define a SQL query. Such a query may contain fuzzy elements, which are put there as query parameters. The same syntax and semantics applies as presented in Sections 2 and 4. The form shown in Fig. 16 is deliberately kept very simple so as to make it possible to use any WWW browser available on the market. The user can learn which fuzzy elements are at his or her disposal by opening another window which lists all fuzzy values, relations etc. currently defined in the copy of FQUERY for Access, to which the query will be delivered.

When the query is created, the user sends it to the WWW server pressing SEND button. After receiving the request along with the query definition data,

Fuzzy querying system (pilot version) - Microsoft Internet Explorer

File Edit View Go Favorites Help

Back Forward Stop Refresh Home Search Favorites Print Font Mail Edit

Address <http://localhost/aspsamp/fquery/fquery97.asp> Links

SPECIFY THE QUERY:

SELECT
x

FROM
houses

WHERE
price=[FIA_FV Low]

SEND RESET

[Help information on available query components](#)
(Hint: open it in a "New window" after clicking the right mouse button on this hyperlink)

Fig. 16. Formulation of a simple condition in a query using the Microsoft Explorer.

the server starts script, written in VBScript, which are embedded in the requested document. Notice, that this time script is run at the server side rather than at the client side as it was the case for the first WWW-based interface presented. This script starts an instance of Microsoft Access opens appropriate database, and then executes the query submitted by the user over the Internet. This execution is initiated through the call to the FQUERY for Access routines securing proper interpretation of all fuzzy terms used – exactly as in the case when the user is directly interacting with Microsoft Access package with FQUERY for Access add-on installed. The results of the query are captured by the script and used to create an HTML page which is then send back to the browser – see, e.g., Fig. 17.

Concluding, the same fuzzy querying tool, FQUERY for Access, may be used in the desktop and Internet environment as well.

This completes our brief description of implementations of linguistic (fuzzy) querying over the Internet by employing two major WWW browsers, the Netscape Navigator and Microsoft Explorer. We have only mentioned here some implementation-specific elements and solutions since the other ones are implemented analogously as for the standalone application described in Sections 2.

We hope that the Internet is a challenge for fuzzy querying and, at the same time, may give momentum to a further research in this relevant field.

Query results:

SELECT * FROM houses WHERE price=[FfA_FV Low]

MD	ADDRESS	PRICE	BEDROOMS	BATHROOMS	RECEPTION ROOMS	CAR SPACE	CELLAR ROOMS	LIVING AREA	LAND AREA
1.0000	7 Bridge Street	223200	2	1	1	1	0	244	1744
1.0000	32 Low Street	219700	1	1	2	1	0	236	1725
1.0000	89 Eastgate Street	218700	2	2	2	1	0	348	1491
1.0000	15 Masons Yard	261200	2	1	1	2	0	232	2148
0.9869	56 Beauchamp Place	290600	4	3	2	2	1	472	1962

PgUp PgDn

Fig. 17. Results of fuzzy querying over the Internet using the Microsoft Explorer.

8. Concluding remarks

The purpose of this paper has been to show, first, how Zadeh's newly advocated paradigm of computing with words can be employed to make querying in DBMSs more flexible, and human-consistent by allowing for the use of natural language.

First, we concentrated on standalone applications, and have presented the idea, technical solution, and implementation of FQUERY for Access [12–18], a flexible and human-consistent querying system for Microsoft Access, a popular Windows-based relational DBMS. We have shown that a combination of the flexibility of fuzzy querying and a modern user-friendly Windows-based GUI provides a synergistic effect, and gives a new quality in database querying.

Then, we have shown an implementation of fuzzy querying over the Internet using its WWW service, and modern Internet-related tools and techniques. This may hopefully be a first step in providing a flexible and human-consistent fuzzy querying interface which may help an average user formulate his or her queries in natural language, and hence more easily make productive use of vast information resources of the Internet.

9. For further reading

[6,11,22,26,28,29,32,33,40,41]

References

- [1] G. Bordogna, P. Carrara, G. Pasi, Fuzzy approaches to extend Boolean information retrieval, in: P. Bosc, J. Kacprzyk (Eds.), *Fuzziness in Database Management Systems*, Physica-Verlag, Heidelberg, 1995, pp. 231–274.
- [2] P. Bosc, M. Galibourg, G. Hamon, Fuzzy querying with SQL: extensions and implementations aspects, *Fuzzy Sets Syst.* 28 (1988) 333–349.
- [3] P. Bosc, J. Kacprzyk (Eds.), *Fuzziness in Database Management Systems*, Physica-Verlag, Heidelberg, 1995.
- [4] P. Bosc, O. Pivert, Fuzzy querying in conventional databases, in: L.A. Zadeh, J. Kacprzyk (Eds.), *Fuzzy Logic for the Management of Uncertainty*, Wiley, New York, 1992, pp. 645–671.
- [5] P. Bosc, O. Pivert, SGL^f: a relational database language for fuzzy querying. *IEEE Trans. on Fuzzy Systems* 3 (1995) 1–17.
- [6] P. Bosc, L. Lietard, O. Pivert, Quantified statements and database fuzzy querying, in: P. Bosc, J. Kacprzyk (Eds.), *Fuzziness in Database Management Systems*, Physica-Verlag, Heidelberg, 1995, pp. 275–308.
- [7] B.P. Buckles, F.E. Petry, A fuzzy representation of data for relational databases, *Fuzzy Sets Syst.* 7 (1982) 213–226.
- [8] S.K. Chang, J.S. Ke, Database skeleton and its application to fuzzy query translation, *IEEE Trans. Software Eng.* SE-4 (1978) 31–43.

- [9] S.K. Chang, J.S. Ke, Translation of fuzzy queries for relational database systems, *IEEE Trans. Pattern Anal. Machine. Intel. PAMI-1* (1979) 281–294.
- [10] W. Dobrzyński, J. Kacprzyk, S. Zadrozny, An example of fuzzy querying using Microsoft' Active Server Pages tools, in: *Proceedings of Fifth European Congress on Intelligent Techniques and Soft Computing – EUFIT'97*, Aachen, Germany, 1997, vol. 2, pp. 1181–1185.
- [11] J. Kacprzyk, Fuzzy logic in DBMSs and querying, in: N.K. Kasabov, G. Coghill (Eds.), *Proceedings of the Second New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems* (Dunedin, New Zealand), IEEE Computer Society Press, Los Alamitos, CA, USA, 1995, pp. 106–109.
- [12] J. Kacprzyk, S. Zadrozny, Fuzzy querying for Microsoft Access, in: *Proceedings of the Third IEEE Conference on Fuzzy Systems*, Orlando, USA, 1994a, vol. 1, pp. 167–171.
- [13] J. Kacprzyk, S. Zadrozny, Fuzzy queries in Microsoft Access: toward a 'more intelligent' use of Microsoft Windows based DBMSs, in: *Proceedings of the 1994 Second Australian and New Zealand Conference on Intelligent Information Systems – ANZIIS'94*, Brisbane, Australia, 1994b, pp. 492–496.
- [14] J. Kacprzyk, S. Zadrozny, FQUERY for Access: fuzzy querying for a Windows-based DBMS, in: P. Bosc, J. Kacprzyk (Eds.), *Fuzziness in Database Management Systems*, Physica-Verlag, Heidelberg, 1995a, pp. 415–433.
- [15] J. Kacprzyk, S. Zadrozny, Fuzzy queries in Microsoft Access v. 2, in: *Proceedings of the Sixth International Fuzzy Systems Association World Congress*, Sao Paulo, Brazil, 1995, vol. 2, pp. 341–344.
- [16] J. Kacprzyk, S. Zadrozny, A fuzzy querying interface for a WWW-server-based relational DBMS, in: *Proceedings of IPMU'96 – Sixth International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Granada, Spain, vol. 1, 1996, pp. 19–24.
- [17] J. Kacprzyk, S. Zadrozny, Fuzzy queries in Microsoft Access v. 2, in: D. Dubois, H. Prade, R.R. Yager (Eds.), *Fuzzy Information Engineering – A Guided Tour of Applications*, Wiley, New York, 1997a, pp. 223–232.
- [18] J. Kacprzyk, S. Zadrozny, Implementation of OWA operators in fuzzy querying for Microsoft Access, in: R.R. Yager, J. Kacprzyk (Eds.), *The Ordered Weighted Averaging Operators: Theory and Applications*, Kluwer Academic Publishers, Boston, 1997b, pp. 293–306.
- [19] J. Kacprzyk, S. Zadrozny, Flexible querying using fuzzy logic: An implementation for Microsoft Access, in: T. Andreasen, H. Christiansen, H.L. Larsen (Eds.), *Flexible Query Answering Systems*, Kluwer Academic Publishers, Boston, 1997c, pp. 247–275.
- [20] J. Kacprzyk, S. Zadrozny, A fuzzy querying interface for a WWW environment, in: *Proceedings of the IFSA'97 – Seventh International Fuzzy Systems Association World Congress* (Prague, Czech Rep.), Academia, Prague, 1997d, vol. 4, pp. 285–290.
- [21] J. Kacprzyk, S. Zadrozny, Issues and solutions for fuzzy database querying over internet, in: *Proceedings of the Fifth European Congress on Intelligent Techniques and Soft Computing EUFIT'97*, Aachen, Germany, 1997e, vol. 2, pp. 1191–1195.
- [22] J. Kacprzyk, S. Zadrozny, On linguistic approaches in flexible querying and mining of association rules, in: H.L. Larsen, J. Kacprzyk, S. Zadrozny, T. Andreasen, H. Christiansen (Eds.), *Flexible Query Answering Systems. Recent Advances*, Physica-Verlag (Springer), Heidelberg and New York, 2001, pp. 475–484.
- [23] J. Kacprzyk, S. Zadrozny, A. Ziółkowski, FQUERY III+: a 'human consistent' database querying system based on fuzzy logic with linguistic quantifiers, *Inf. Syst.* 6 (1989) 443–453.
- [24] J. Kacprzyk, A. Ziółkowski, Retrieval from databases using queries with fuzzy linguistic quantifiers, in: H. Prade, C.V. Negoita (Eds.), *Fuzzy Logics in Knowledge Engineering*, Verlag TUV Rheinland, Cologne, 1986a, pp. 46–57.
- [25] J. Kacprzyk, A. Ziółkowski, Database queries with fuzzy linguistic quantifiers, *IEEE Trans. Syst., Man Cybernet. SMC-16* (1986b) 474–479.

- [26] P.C. Kim, A taxonomy on the architecture of database gateways for the Web [URL:<http://grigg.chungnam.ac.kr/~uniweb/documents/taxonomy/text.html>] (1996).
- [27] H.L. Larsen, R.R. Yager, The use of fuzzy relational thesauri for classificatory problem solving in information retrieval and expert systems, *IEEE Trans. Syst., Man Cybernet. SMC-23* (1993) 31–41.
- [28] H.L. Larsen, J. Kacprzyk, S. Zadrozny, T. Andreasen, H. Christiansen, (Eds.), *Flexible Query Answering Systems. Recent Advances*. Physica-Verlag (Springer-Verlag), Heidelberg and New York, 2001.
- [29] S. Miyamoto, in: *Fuzzy Sets in Information Retrieval and Cluster Analysis*, Kluwer Academic Publishers, Dordrecht, Boston, London, 1990.
- [30] F.E. Petry, in: *Fuzzy Databases: Principles and Applications*, Kluwer Academic Publishers, Boston, 1996.
- [31] V. Tahani, A conceptual framework for fuzzy query processing: a step toward very intelligent data systems, *Inf. Process. Manage.* 13 (1977) 289–303.
- [32] M.A. Vila, J.C. Cubero, J.M. Medina, O. Pons, Logic and fuzzy relational databases: a new language and a new definition, in P. Bosc, J. Kacprzyk (Eds.), *Fuzziness in Database Management Systems*. Physica-Verlag, Heidelberg, 1994, pp. 114–138.
- [33] M.A. Vila, J.C. Cubero, J.M. Medina, O. Pons, *artukul z Pragi*, 1997.
- [34] R.R. Yager, On ordered weighted averaging aggregation operators in multi-criteria decision making, *IEEE Trans. Syst., Man Cybernet.* 18 (1988) 183–190.
- [35] R.R. Yager, J. Kacprzyk, Eds. *The Ordered Weighted Averaging Operators: Theory and Applications*. Kluwer Academic Publishers, Boston, 1997.
- [36] A. Yazici, R. George, B.P. Buckles, F.E. Petry, A survey of conceptual and logical data models for uncertainty management, in: L.A. Zadeh, J. Kacprzyk (Eds.), *Fuzzy Logic for the Management of Uncertainty*, Wiley, New York, 1992, pp. 607–643.
- [37] L.A. Zadeh, A computational approach to fuzzy quantifiers in natural languages, *Comput. Maths. Appl.* 9 (1983) 149–184.
- [38] S. Zadrozny, J. Kacprzyk, Fuzzy querying using the ‘query-by-example’ option in a Windows-based DBMS, in: *Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing-EUFIT’95*, Aachen, Germany, 1995, vol. 2, pp. 733–736.
- [39] S. Zadrozny, J. Kacprzyk, Multi-valued fields and values in fuzzy querying via FQUERY for Access, in: *Proceedings of FUZZ-IEEE’96 – Fifth International Conference on Fuzzy Systems*, New Orleans, USA, 1996, vol. 2, pp. 1351–1357.
- [40] S. Zadrozny, J. Kacprzyk, K. Floisand, Internet and WWW – new opportunities for information technology and soft computing, in: *Proceedings of IFSA’97 – Seventh International Fuzzy Systems Association World Congress* (Prague, Czech Rep.), Academia, Prague, 1997, vol. 4, pp. 316–319.
- [41] M. Zemankova, J. Kacprzyk, The roles of fuzzy logic and management of uncertainty in building intelligent information systems, *J. Intell. Inf. Syst.* 2 (1993) 311–317.
- [42] M. Zemankova-Leech, A. Kandel, in: *Fuzzy Relational Databases – a Key to Expert Systems*, Verlag TÜV Rheinland, Cologne, 1984.