A Server for Fuzzy SQL Queries

José Galindo[1] **Juan M. Medina**[2] **Olga Pons**[2] **Juan C. Cubero**[2]

[1] Dpto. Lenguajes y Ciencias de la Computación, Universidad de Málaga. 29071 Málaga (Spain). ppgg@lcc.uma.es

[2] Dpto. Ciencias de la Computación e I.A., Universidad de Granada. 18071 Granada (Spain). {medina,opc,carlos}@decsai.ugr.es

Abstract. The client-server model is being used mostly in the actual DataBase Management Systems (DBMS). However, these DBMS do not allow either to make flexible queries to the database or to store vague information in it. We have developed a FSQL Server for a Fuzzy Relational Database (FRDB). The FSQL language (Fuzzy SQL) is an extension of the SQL language that allows us to write flexible conditions in our queries. This Server has been developed for Oracle, following the model GEFRED, a theoric model for FRDB that includes fuzzy attributes to store vague information in the tables. The FSQL Server allows us to make flexible queries about traditional (crisp) or fuzzy attributes and we can use linguistic labels defined on any attribute.

KEYWORDS: Information Storage and Retrieval, Flexible and Fuzzy Queries, Fuzzy Relational Databases, Fuzzy SQL.

1 Introduction

The relational model was developed by E.F. Codd of IBM and published in 1970 in [6]. This model is the most used at present. In a theoric level, there exists many FRDB systems that, based on the relational model, they extend it in order to allow storing and/or treating vague and uncertain information.

In [16], Tahani presented a system to carry out fuzzy queries in a classic relational database. More recently, this topic has been widely studied by P. Bosc et al. [1].

Buckles and Petry presented in [2] their database model, in which they used similarity relations among a finite set of possible values for an attribute. This model was extended in [3] to store fuzzy numbers.

In order to store possibility distributions (in Zadeh's sense [20]) Umano and Fukami published their model in [17, 18], with a fuzzy relational algebra [19]. Later, other models appeared like the Prade-Testemale model [14, 15] and the Zemankova-Kandel model [21].

In [12, 13], the GEFRED model was proposed for FRDB. The GEFRED model represents a synthesis among the different models which have appeared to deal with the problem of the representation and management of fuzzy information in relational databases. One of the main advantages of this model is that it consists of a general abstraction which allows us to deal with different approaches, even when these may seem disparate.

Together with GEFRED, a fuzzy relational algebra was defined and in [9] a fuzzy relational calculus is defined. Both query languages allow the performance of fuzzy queries and they retrieve a table in which every attribute of every tuple may have a fulfilment degree associated. This fulfilment degree indicates the level with which this concrete value has satisfied the query condition and it is between 0 (condition not satisfied) and 1 (condition totally satisfied).

We present here an architecture that implements the model GEFRED for one of the commercial DataBase Management Systems (DBMS) most frequently used at present: Oracle. We have programmed a FRDB server that answers queries in a particular language, FSQL (Fuzzy SQL). This query language allows the writing of flexible queries, fulfilment degrees and thresholds, fuzzy constants... and other options that are very useful in making flexible queries in a FRDB.

2 Preliminary concepts about GEFRED

The GEFRED model is based on the definition which is called *Generalized Fuzzy Domain* (D) and *Generalized Fuzzy Relation* (R), which include classic domains and classic relations respectively.

Definition 1. *If U is the discourse domain or universe, $\widetilde{\mathcal{P}}(U)$ is the set of all possibility distributions defined for U, including those which define the* Unknown *and* Undefined *types (types 8 and 9), and* NULL *is another type defined below (type 10) therefore, we define the* ***Generalized Fuzzy Domain*** *as $D \subseteq \widetilde{\mathcal{P}}(U) \cup$* NULL. *The* **Unknown**, **Undefined** *and* **NULL** *types are defined according to Umano, Fukami et al. [7, 18]. Then, all data types that can be represented are in table 1.*

Definition 2. *A* ***Generalized Fuzzy Relation***, *R, is given by two sets: "Head" ($\mathcal{H}$) and "Body" ($\mathcal{B}$), $R = (\mathcal{H}, \mathcal{B})$, defined as: The "Head" consists of a fixed set of attribute-domain-compatibility terns (where the last is optional), $\mathcal{H} = \{(A_1 : D_1\ [, C_1]), \ldots, (A_n : D_n\ [, C_n])\}$ where each attribute A_j has an underlined fuzzy domain, not necessarily different, D_j (j=1,2,...,n). C_j is a "compatibility attribute" which takes values in the range [0,1]. The "Body" consists of a set of different generalized fuzzy tuples, where each tuple is composed of a set of attribute-value-degree terns (the degree is optional), $\mathcal{B} = \{(A_1 : \widetilde{d}_{i1}\ [, c_{i1}]), \ldots, (A_n : \widetilde{d}_{in}\ [, c_{in}])\}$ with $i = 1, 2, \ldots, m$, where m is the number of tuples in the relation, and where $\widetilde{d}_{ij}$ represents the domain value for the tuple i and the attribute A_j, and c_{ij} is the compatibility degree associated with this value.* □

The comparison operators are also redefined in order to be adapted to the fuzzy nature of our data. For example, the "Possibly Equal" comparator is defined in the following equation for possibility distributions where $\widetilde{p}, \widetilde{p'} \in D$, and their associated possibility distributions are $\pi_{\widetilde{p}}$ and $\pi_{\widetilde{p'}}$ respectively. U is the discourse domain underlying the generalized fuzzy domain D (see definition 1):

1.	A single scalar (e.g. Size=Big, represented by the possibility of distribution 1/Big).
2.	A single number (e.g. Age=28, represented by the possibility of distribution 1/28).
3.	A set of mutually exclusive possible scalar assignations (e.g. Behaviour={Bad,Good}, represented by {1/Bad,1/Good}).
4.	A set of mutually exclusive possible numeric assignations (e.g. Age= $\{20, 21\}$, represented by $\{1/20, 1/21\}$).
5.	A possibility distribution in a scalar domain
6.	A possibility distribution in a numeric domain (e.g. Age= $\{0.4/23, 1.0/24, 0.8/25\}$, fuzzy numbers or linguistic labels).
7.	A real number belonging to $[0, 1]$, referring to the degree of matching (e.g. Quality=0.9).
8.	An **Unknown** value with possibility distribution **Unknown**=$\{1/d : d \in D\}$ on domain D, considered.
9.	An **Undefined** value with possibility distribution **Undefined**=$\{0/d : d \in D\}$ on domain D, considered.
10.	A **NULL** value given by **NULL**={1/Unknown,1/Undefined}.

Table 1. Data types for the FRDB.

$$\Theta^{=}(\widetilde{p}, \widetilde{p'}) = \sup_{d \in U} \min\left(\pi_{\widetilde{p}}(d), \pi_{\widetilde{p'}}(d)\right) \tag{1}$$

We can define a similarity relation between two scalars (with a non ordered domain), in order to compare them.

On these definitions, GEFRED redefines the relational algebraic operators in the so-called **Generalized Fuzzy Relational Algebra**: Union, Intersection, Difference, Cartesian Product, Projection, Selection, Join and Quotient. These operators are defined giving the head and body of a generalized fuzzy relation which will be the result of the operation. All these operators are defined in [12] and [13], but the quotient is defined in [8].

In a relation (definition 2), the **compatibility degree** for an attribute value is obtained by manipulation processes performed on that relation and it indicates the degree to which that value has satisfied the operation performed on it.

3 FSQL (Fuzzy SQL)

The FSQL language extends the SQL language [4, 5] to allow flexible queries. We have extended the `SELECT` command in order for it to express flexible queries and, due to its complex format, we only show here an abstract with the main extensions to this command:

- **Linguistic Labels**: If an attribute is capable of fuzzy treatment then linguistic labels can be defined on it. These labels will be preceded with the symbol `$` to distinguish them easily. There are two types of labels and they will be used in different fuzzy attribute types:

1. Labels for attributes with an ordered underlined fuzzy domain: Every label of this type has associated a trapezoidal possibility distribution like the figure 1. So, for example, we can define labels `$Very_Short`, `$Short`, `$Normal`, `$Tall` and `$Very_Tall` on the `Height` attributed to a person. For example, we can set a possibility distribution for label `$Tall` with the values `a=175`, `b=180`, `c=195` y `d=200` (in centimeters).

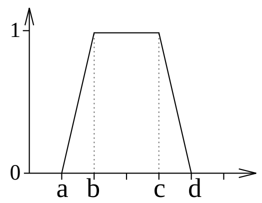

Fig. 1. Trapezoidal possibility distribution.

2. Labels for attributes with a non ordered domain (scalars of types 1, 3 and 5 in table 1): Here, there is a similarity relation defined between each two labels in the domain. The similarity degree is in the interval [0,1]. For example, for the `Hair_Colour` attributed to a person, we can define that labels `$Fair` and `$Red_Haired` are similar with a 0.6 degree.

– **Fuzzy Comparators**: Besides the typical comparators (`=`, `>`...), FSQL includes the fuzzy comparators in table 2. Like in SQL, fuzzy comparators compare one column with one constant or two columns of the same type.

Comparator for:		
Possibility	Necessity	Significance
`FEQ`	`NFEQ`	Fuzzy EQual (Possibly/Necessarily Equal)
`FGT`	`NFGT`	Fuzzy Greater Than
`FGEQ`	`NFGEQ`	Fuzzy Greater or Equal
`FLT`	`NFLT`	Fuzzy Less Than
`FLEQ`	`NFLEQ`	Fuzzy Less or Equal
`MGT`	`NMGT`	Much Greater Than
`MLT`	`NMLT`	Much Less Than

Table 2. Fuzzy Comparators for FSQL (Fuzzy SQL).

Possibility comparators are more general than necessity comparators are. Then, necessity comparators retrieve less tuples and these comply with the conditions necessarily. In [11], the definitions of all the comparators can be found with some examples of queries and their results. For example, `FEQ` uses de equation 1 and `NFEQ` uses the following equation:

$$\Theta^{\mathtt{NFEQ}}(\widetilde{p}, \widetilde{p'}) = \inf_{d \in U} \max\left(1 - \pi_{\widetilde{p}}(d), \pi_{\widetilde{p'}}(d)\right) \tag{2}$$

- **Fulfilment thresholds** (γ): For each simple condition a fulfilment threshold may be established (default is 1). with the format:

  <condition> THOLD γ

  indicating that the condition must be satisfied with minimum degree $\gamma \in [0,1]$ to be considered. The reserved word THOLD is optional and may be substituted by a traditional crisp comparator ($=$, $\leq$...), modifying the query meaning. The word THOLD is equivalent to using the crisp comparator $\geq$.

  Example 1. Give me all persons with fair hair (in minimum degree 0.5) that they are possibly taller than label $Tall (in minimum degree 0.8):

  ```
  SELECT * FROM Person WHERE Hair   FEQ $Fair THOLD 0.5 AND
                             Height FGT $Tall THOLD 0.8
  ```

  If we are looking for persons that are necessarily taller than label $Tall, then we use the fuzzy comparator NFGT instead of FGT. □

- **Function** CDEG(<attribute>): It shows a column with the fulfilment degree of the condition of the query, for a specific attribute, which is expressed between brackets as the argument. If logic operators appear, the calculation of this compatibility degree is carried out following table 3. We use the minimum T-norm and the maximum T-conorm, but the user may change these values by default modifying only a view (FSQL_NOTANDOR). In this view the user can set the function to use for every logic operator (NOT, AND, OR). Obviously, that function must be implemented in the FSQL Server or may be implemented by the user himself.

<Condition>	CDEG(<Condition>)
<cond1> AND <cond2>	min(CDEG(<cond1>),CDEG(<cond2>))
<cond1> OR <cond2>	max(CDEG(<cond1>),CDEG(<cond2>))
NOT <cond1>	1 - CDEG(<cond1>)

Table 3. Default computation for function CDEG with logic operators in FSQL.

  If the argument of the function CDEG is an attribute, then the function CDEG only uses the conditions that includes that attribute. We can use CDEG(*) to obtain the fulfilment degree of each tuple (with all of its attributes, not just one of them) in the condition.
- **Character** %: It is similar to the character * of SQL but this one also includes the columns for the fulfilment degrees of the attributes in which they are relevant. In the result you will also find the function CDEG applied to each and every one of the fuzzy attributes which appear in the condition. If we want to obtain in example 1 two more columns with the degrees of CDEG(Hair) and CDEG(Height), we would only have to replace * by %. Of course, this character may be also used with the format [[scheme.]table.]%, as for example: Person.%.

– **Fuzzy Constants**: In FSQL we can use the fuzzy constants as detailled in table 4. See examples 1 and 2.

F. Constant	Significance
`UNKNOWN`	Unknown value but the attribute is applicable (type 8 in table 1).
`UNDEFINED`	The attribute is not applicable or it is meaningless (type 9 in table 1).
`NULL`	Total ignorance: We know nothing about it (type 10 in table 1).
`$[a,b,c,d]`	Fuzzy trapezoid (a≤b≤c≤d): See figure 1.
`$label`	Linguistic Label: It may be a trapezoid or a scalar (defined in FMB).
`[n,m]`	Interval "Between n and m" (a=b=n and c=d=m).
`#n`	Fuzzy value "Approximately n" (b=c=n and n–a=d–n=margin).

Table 4. Fuzzy constants that may be used in FSQL queries.

– **Condition with** `IS`: Another kind of condition we can use has the next format:

```
<Fuzzy_Attribute>  IS  [NOT]  (UNKNOWN | UNDEFINED | NULL)
```

Remarks concerning the condition with `IS`:
- This condition (without `NOT`) will be true if the left fuzzy attribute value (`<Fuzzy_Attribute>`) is the fuzzy constant placed on the right.
- If the attribute is not fuzzy and the constant is `NULL`, this constant will be understood in the way given by the DBMS.
- If `FEQ` is used instead of `IS`, the compatibility degree between attribute and constant is compared (equation 1) and not only when the attribute is *equal* to the constant.

Example 2. A query, which shows a compatibility degree, uses a trapezoidal constant and avoids the `UNKNOWN` values, could possibly be:

```
SELECT City,CDEG(Inhabitants)
FROM   Population
WHERE  Country='Spain' AND
       Inhabitants FGEQ $[200,350,650,800] .75 AND
       Inhabitants IS NOT UNKNOWN
```

Remarks concerning the example:

– The minimum threshold is set at 0.75. The word `THOLD` not appear because it is optional.
– As we are using the comparator `FGEQ`, the `c` and `d` values of the trapezoid will not be used, i.e., if inhabitants equals or exceeds 350, the degree will be 1. Naturally, if the inhabitants is equal to or less than 200 the degree will be zero.

– If a spanish city has in the `Inhabitants` attribute the possibility distribution `$[50,150,200,350]`, its fulfilment degree of the condition will be 0.5 and it will not appear in the final result, because the minimum has been set at 0.75. □

Thus, in order to make FSQL queries are especially flexible, in every simple condition, we can modify the following three already defined important elements:

1. **Fuzzy Comparators** (table 2), overcoat changing between possibility and necessity.
2. **Thresholds**, in order to retrieve only the *most important* items.
3. **Comparators instead of word** `THOLD`, to modify the query meaning, e.g. to retrieve the *least important* items.
4. **Fuzzy constant** (Table 4): If the right part of a simple condition is a fuzzy constant this may be flexibly modified in order to better achieve the target.

4 FRDB Architecture

The FRDB and the FSQL Server [10] have been implemented using an already existing DBMS. Basically this involves three consequences:

1. The system will be slower than if it were programmed at a lower level.
2. The task is made much more simple (we do not have to program the DBMS).
3. We obtain all the advantages of the DBMS host (security, efficiency...) without our Server having to taking them into account.

The DBMS chosen was Oracle, because of its changeability, its large extension and its ability to program packages (with functions and procedures) internal to the system, in its own language, PL/SQL, which turns out to be quite efficient (more than others we have tested: Pro*C...). Of course, this architecture can be implemented in other DBMS. Basically, the architecture of the FRDB with the FSQL Server is made up by:

1. **Data**: Traditional Database and Fuzzy Meta-knowledge Base.
2. **FSQL Server**.
3. **FSQL Client**.

4.1 Data: Traditional Database and Fuzzy Meta-knowledge Base

The data can be classified in two categories:

– **Traditional Database**: They are data from our relations with a special format to store the fuzzy attribute values. The fuzzy attributes are classified by the system in 3 types:
  - Fuzzy Attributes **Type 1**: These attributes are totally crisp (traditional), but they have some linguistic trapezoidal labels defined on them, which allow us to make the query conditions for these attributes more flexible. Besides, we can use all constants in table 4 in the query conditions with these fuzzy attributes.

  - Fuzzy Attributes **Type 2**: These attributes admit crisp data as well as possibility distributions over an ordered underlying domain. With these attributes we can store and use all the constants we see in table 4.
  - Fuzzy Attributes **Type 3**: They are the types 1, 3 and 5 of table 1. These attributes have not an ordered underlying domain, for instance the hair colour in example 1. On these attributes some labels are defined and on these labels a similarity relation has yet to be defined. With these attributes we can only use the fuzzy comparator `FEQ`, as they have no relation of order. Obviously we cannot store or use the constants fuzzy trapezoid, interval and approximate value of table 4.
- **Fuzzy Meta-knowledge Base** (FMB): It stores information about the FRDB in a relational format. It stores attributes which admit fuzzy treatment and it will store different information for each one of them, depending on their type:
  - Fuzzy Attributes **Type 1**: In order to use crisp attributes in flexible queries we will only have to declare them as being a fuzzy attribute type 1 and store the following data in the FMB:
    - ⋆ Trapezoidal linguistic labels: Name of the label and `a`, `b`, `c` and `d` values (as in figure 1).
    - ⋆ Value for the margin of the approximate values (see table 4).
    - ⋆ Minimum distance in order to consider two values *very* separated (used in comparators `MGT/NMGT` and `MLT/NMLT`).
  - Fuzzy Attributes **Type 2**: As well as declare them as being a fuzzy attribute type 2, these attributes have to store the same data in the FMB as the fuzzy attributes type 1.
  - Fuzzy Attributes **Type 3**: They store in the FMB their linguistic labels, the similarity degree amongst themselves and the compatibility between attributes of this type, i.e., the attributes that use the same labels and that can be compared amongst themselves.

4.2 FSQL Server

It has been programmed entirely in PL/SQL and it includes 3 kinds of functions:

- **Translation Function** (`FSQL2SQL`): It carries out a lexical, syntactic and semantic analysis of the FSQL query. If errors, of any kind whatsoever, are found, it will generate a table with all the found errors. If there are no errors, the FSQL query is translated into a standard SQL sentence. The resulting SQL sentence includes reference to the following kinds of functions.
- **Representation Functions**: These functions are used to show the fuzzy attributes in a comprehensible way for the user and not in the internally used format.
- **Fuzzy Comparison Functions**: They are utilized to compare the fuzzy values and to calculate the compatibility degrees (`CDEG` function).

The Translation Function, summarizing, what it does is to replace the fuzzy attributes of the `SELECT` by calls to Representation Functions, the fuzzy conditions by calls to the Fuzzy Comparison Functions and the `CDEG` functions by calls to the Fuzzy Comparison Functions and other functions if there exists some logic operators (default functions are shown in table 3).

4.3 FSQL Client

It is an independent program which serves as an interface between the user and the FSQL Server. The user introduces the FSQL query and the client program communicates with the server and with the database in order to obtain the final results. The Translation Function of the FSQL Server is the only function that is to be executed directly by the client. We have developed a FSQL Client for Windows 95 and NT, called FQ (Fuzzy Queries).

5 Calling to the FSQL Server

We summarized the use process of the FSQL Server in the diagram in figure 2. In short, for a FSQL query the following steps are taken:

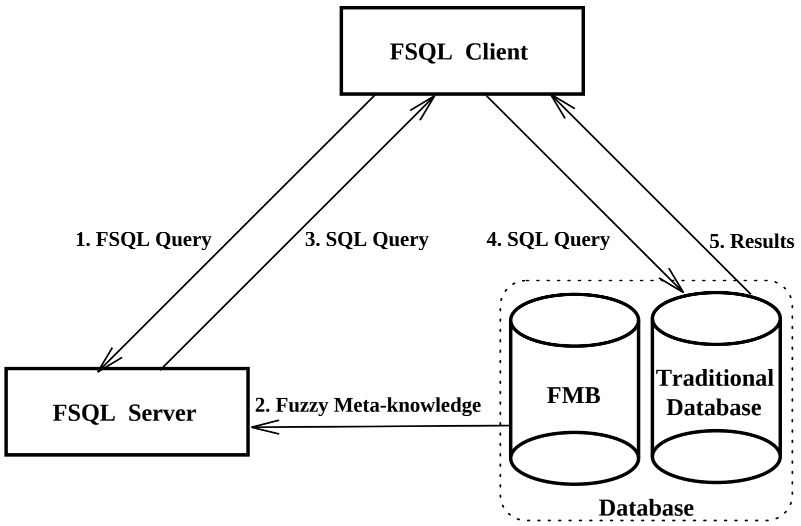

Fig. 2. Basic Architecture for the FRDB with the FSQL Server.

1. The FSQL Client program sends the FSQL query to the FSQL Server.
2. The FSQL Server analizes the query and, if it is correct, generates a SQL sentence, starting from the original query. In this step, the FSQL Server uses the information of the FMB.

3. Once the query has been generated in SQL, the client program will read it.
4. The client program will send the SQL query to any database, which is coherent with the FMB. In the execution of this query, functions of the FSQL Server will be used (Representation and Fuzzy Comparison Functions).
5. Finally, the client will receive the resulting data and it will show them.

Steps number 3 and 4 could have been eliminated to increase the efficiency but, in the way we have presented, we achieve an independence between the Traslation phase (steps number 1, 2 and 3) and the Consultation phase (steps number 4 and 5). That way, if we make use of a local database with FSQL Server and FMB, we will be able to translate our sentences locally and send the translated queries to a remote database, avoiding this way that the network is overcharged with error messages, translated queries... This way, the remote database would not have to have the Translation Function installed.

6 Conclusions

We have presented a FSQL Server, which extends an Oracle database so that it can be consulted in a flexible way through a language that is very similar to the popular SQL: The Fuzzy SQL (**FSQL**) language.

Besides, the main advantage of the used model is that it allows storing many types of fuzzy data (table 1). Then, the FSQL Server does the representation and the fuzzy comparison of these fuzzy data possible.

The presented architecture is not ideal for a final product, but it allows us to evaluate the possibilities of a FRDB on a practical level and not only on a theoretical one. Let us hope the DBMS incorporate soon new types of internal data which allow storing fuzzy values and fuzzy processing of these values. In this line of thinking the object-oriented databases could cooperate quite a lot.

However, the carried out tests prove that the FSQL Server is very fast, being able to be used as a real time server. Obviously, if the query is very complicated and we have a very large database, even if the translation is fast, the information recovery may be a little slow, since if the condition is long then the DBMS would carry out many operations.

Presently, we are working on the creation of a FSQL Client that can be executed through Internet (in Java). We are also working to apply this architecture in some real applications.

The usefulness of this Server is clear, since it can even be used in traditional databases, using only fuzzy attributes type 1, but we would like to underline the possibilities it could offer when used in combination with Data Mining techniques (classification...).

References

1. P. Bosc, O. Pivert, K. Farquhar, "Integrating Fuzzy Queries into an Existing Database Management System: An Example". International Journal of Intelligent Systems, Vol. 9, 475–492 (1994).

2. B.P. Buckles, F.E. Petry, "A Fuzzy Representation of Data for Relational Databases". Fuzzy Sets and Systems 7, 213–226 (1982).
3. B.P. Buckles, F.E. Petry, "Extending the Fuzzy Database with Fuzzy Numbers". Information Sciences 34, 145–155 (1984).
4. D.D. Chamberlin, R.F. Boyce, "SEQUEL: A Structured English Query Language". Proc. ACM SIGMOD Workshop on Data Description. Access and Control, 1974.
5. D.D. Chamberlin et al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control". IBM J. R&D 20, num. 6, November, 1976.
6. E.F. Codd, "A Relational Model of Data for Large Shared Data Bases". Communications of the ACM, 13, no. 6, 377–387, June 1970.
7. S. Fukami, M. Umano, M. Muzimoto, H. Tanaka, "Fuzzy Database Retrieval and Manipulation Language", IEICE Technical Reports, Vol. 78, N° 233, pp. 65–72, AL-78-85 (Automata and Language) (1979).
8. J. Galindo, J.M. Medina, M.C. Aranda, "Una solución al problema de la División Relacional Difusa". VII Congreso Español sobre Tecnologías y Lógica Fuzzy (ESTYLF'97). Tarragona (Spain), September 1997.
9. J. Galindo, J.M. Medina, J.C. Cubero, "How to Obtain the Fulfilment Degrees of a Query using Fuzzy Relational Calculus". To appear in "Knowledge management in Fuzzy Databases". Eds. O. Pons, M.A. Vila and J. Kacprzyk. Ed. Physica-Verlag, 1998.
10. J. Galindo, J.M. Medina, O. Pons, M.A. Vila, J.C. Cubero, "A prototype for a fuzzy relational database". Demo session in the 6th International Conference on Extending Database Technology, Valencia (Spain), March 1998.
11. J. Galindo, J.M. Medina, A. Vila, O. Pons, "Fuzzy Comparators for Flexible Queries to Databases". Iberoamerican Conference on Artificial Intelligence, IBERAMIA'98, Lisbon (Portugal), October 1998.
12. J.M. Medina, O. Pons, M.A. Vila, "GEFRED. A Generalized Model of Fuzzy Relational Databases". Information Sciences, 76(1–2), 87–109 (1994).
13. J.M. Medina R., "Bases de Datos relacionales difusas. Modelo teórico y aspectos de su implementación". Ph. Doctoral Thesis, University of Granada, May 1994.
14. H. Prade, C. Testemale, "Generalizing Database Relational Algebra for the Treatment of Incomplete or Uncertain Information and Vague Queries". Information Sciences 34, 115–143 (1984).
15. H. Prade, C. Testemale, "Fuzzy Relational Databases: Representational issues and Reduction Using Similarity Measures". J. Am. Soc. Information Sciences 38(2), 118–126 (1987).
16. V. Tahani, "A Conceptual Framework for Fuzzy Query Processing–A Step toward Very Intelligent Database Systems". Information Process. Management, 13, 289–303 (1977).
17. M. Umano, S. Fukami, M. Mizumoto, K. Tanaka, "Retrieval Processing from Fuzzy Databases". Technical Reports of IECE of Japan, Vol 80, No. 204 (on Automata and Languages), pp. 45–54, AL80-50 (1980).
18. M. Umano, "Freedom-O: A Fuzzy Database System". Fuzzy Information and Decision Processes. Gupta-Sanchez edit. North-Holand Pub. Comp. (1982).
19. M. Umano, S. Fukami, "Fuzzy Relational Algebra for Possibility-Distribution-Fuzzy Relational Model of Fuzzy Data". Journal of Intelligent Information Systems, 3, 7–27 (1994).
20. L.A. Zadeh, "Fuzzy Sets as a Basis for a Theory of Possibility". Fuzzy Sets and Systems, 1, 3–28 (1978).
21. M. Zemankova-Leech, A. Kandel, "Fuzzy Relational Databases – A Key to Expert Systems". Köln, Germany, TÜV Rheinland (1984).