fuzzy sql,
fuzzy database,
fuzzy sets

Przemysław KUDŁACIK [1]

FUZZY SQL QUERIES IN STANDARD SQL DATABASE

Uncertain queries are very common in many areas of human activity. The problem can be seen particularly in medicine, where expressions like ”very high”, ”low”, ”normal” are commonly used in order to describe different information. However, the most popular data repositories do not allow to form imprecise queries in order to filter information. Therefore, the paper proposes an extension to the standard SQL language allowing anybody to profit from fuzzy database using any SQL engine. The existing approaches employ different mechanisms in order to allow the user to perform fuzzy queries on a database. The most complex solutions modify the database engine itself. However, such approach is strongly bound to the modified server version and must be updated with any development of the original server. Nevertheless, there is possible to store fuzzy information using for instance columns of regular relational database. Therefore, this approach proposes extensions to the query language allowing to use fuzzy information in a query and provides a parser transforming a fuzzy query into a standard SQL. Thus, the database server version is irrelevant. The solution is provided as a module written in multi-platform Java language using popular JDBC database connection.

1. INTRODUCTION

Fuzzy sets introduced by Zadeh in 1965 [14] are widely used as a flexible representation of uncertain information. Therefore, it was only a matter of time when first implementations in database systems had been developed. The advantages of fuzzy database queries can be noticed in many scientific and industrial areas. However, the solution can be particularly helpful in medicine, where expressions like ”high”, ”very low” or ”normal” are used to describe different states. Thus, the possibility of making a fuzzy query allowing the user to retrieve i.e. a list of patients with high level of liver fibrosis or even very low blood pressure, would be helpful and practical.

The state of the art indicates solutions in the area of fuzzy databases using different approaches [1], [7], [6], [3], [2], [10], [11], [12], [8]. Generally, the approaches can be divided into two groups. The first one assumes significant modifications in the existing database system, usually open-source, or is based on dedicated implementation. Such approach gives developers more possibilities, however in case of an open-source database servers it is strongly bound to the modified sever version. When new version is developed the modifications must be updated and implemented into server again. Another option considers dedicated systems, which must

[1]University of Silesia, Institute of Computer Science, Będzińska 39, 41-200 Sosnowiec, Poland

be implemented individually and because of constant need of development are much more complex to maintain.

The second group of solutions assume using regular SQL server and SQL language as a basis. Nevertheless, the system allows to form fuzzy queries and retrieve results basing on fuzzy data filtering (conditions on columns storing fuzzy information). Generally speaking, the approach provides a parser translating a fuzzy SQL language (extended SQL) into regular SQL. Therefore, it should not be limited in the context of a particular server vendor or version. Such solution is sufficient to store information in the easiest, but functional forms, like fuzzy sets with trapezoidal or triangular membership functions.
Utilizing an existing SQL server must be considered as an advantage, because the user can choose preferred vendor and can perform upgrades of the server without the impact on the Fuzzy SQL translator.

There are a couple of problems with the existing approaches. First of all, solutions bound to the particular DataBase Management System (DBMS), like extended approach described in [12], [8] only for Oracle DBMS, create a significant limitation of usability. The second problem is the availability of the proposed systems. In some cases approaches are simply out-of-date because they were designed in a deprecated environment (like [3]) or represent just purely theoretical models without implementation (like [13], [2]). Therefore, the idea of a fuzzy SQL presented years ago becomes a big problem in modern platforms and needs re-implementation.

The aim of this work was to develop as simple as possible solution, provided as a programming library of modern platform, allowing any user to use the advantages of a fuzzy database. The proposed approach is designed and tested on the open source MySQL server. However, any SQL server could be used, because the solution is based on the JDBC standard (Java DataBase Connectivity), which makes this approach very flexible. The designed Java module is an independent part of the Fuzzlib library, which in general provides a set of tools for fuzzy analysis in Java language.

In subsequent sections the proposed approach is described beginning with the structure of designed library, followed by a detailed specification of functionality. The specification is focused on the SQL extensions and backed up with examples.

2. DESIGNING A FUZZY SQL - INITIAL ASSUMPTIONS

The main functionality of the designed approach is a cooperation with a relational database in order to create a standard SQL query from a fuzzy query. The fuzzy query extends the standard SQL in terms of:

- definition of fuzzy columns
- definition of linguistic values (fuzzy sets)
- using fuzzy columns and linguistic values in queries with fuzzy operators (i.e. comparison).

Defining fuzzy columns allows storing fuzzy information using regular database table. In this work it was assumed that a fuzzy expression can be described with four floating point values, allowing to define trapezoidal membership function of a fuzzy set. Therefore, a fuzzy set V, defined with a membership function μ_V can be described with the following ordered set of four values a, b, c, d with the restriction that $a <= b <= c <= d$

$$\mu_V(x,a,b,c,d) = \begin{cases} 0\,, & x <= a \\ \frac{x}{b-a} - \frac{a}{b-a}\,, & a < x <= b \\ 1\,, & b < x <= c \\ \frac{x}{c-d} + \left(1 - \frac{c}{c-d}\right)\,, & c < x <= d \\ 0\,, & x > d \end{cases} \qquad x,a,b,c,d \in X, \tag{1}$$

where X in a database system is mapped into a floating point type. Such structure is simple enough to be represented using an existing database system (four columns) and flexible enough to describe wide range of different states. Basically, each subsequent value is designed to represent subsequent nodes of trapezoidal membership function of a fuzzy set. However, also other types of functions can be described, like triangular or rising/falling slopes. Examples of such membership functions are shown in fig. 1

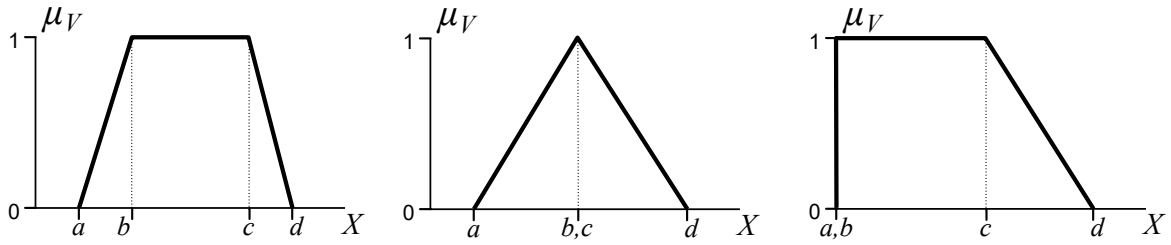

Fig. 1. Different membership functions defined by a, b, c, d values.

Another mentioned problem is definition of linguistic terms in order to use them in queries. It was solved by providing a special table called $fuzzyvalues$ containing the following columns: $name, a, b, c, d$. The name is a primary key of type $char(50)$ and the rest of columns, representing described trapezoidal function, are of type real. The table is used in queries to obtain stored linguistic values in a form of defined fuzzy sets. Records stored in the table are retrieved in queries by name and other tables do not use it (related by a foreign key). Therefore, an additional integer primary key is not needed.

The last functionality is designing a set of fuzzy operators allowing to compare different fuzzy columns with discrete and linguistic values. The following sections precisely describe all the stages of creating and using the designed library.

2.1. DATABASE CONNECTION AND INITIALIZATION

The library is created as a standard Java package named $fuzzlib.fuzzysql$ and is a subset of the larger $fuzzlib$ library [9].

The most important module of the designed library and first to describe is FuzzyDatabaseConnection class. It serves three functions: database connect, database disconnect and sending a query, where the translation from fuzzy SQL to standard SQL is performed. An example of database connection, query execution and disconnection is presented in listing 1.

Listing 1. Connecting to the database

```
FuzzyDatabaseConnection connection =
        new FuzzyDatabaseConnection("com.mysql.jdbc.Driver");

connection.connect("jdbc:mysql://localhost/database",
                   "user", "password");
...
connection.execute("select fc:fuzzy_column from some_table")
...
connection.disconnect()
```

The ”execute” method of a FuzzyDatabaseConnection accepts a fuzzy query string, which is further translated by inner layers of the module to convert it into regular SQL and execute. It is important to mention, that the method returns an object of FuzzyResult class, which allows to access either a whole set of results (for select statements) or the integer number of modified rows for update and other types of queries.

The following sections of the paper focus on different aspects of that process.

2.2. DEFINING FUZZY VALUES

For the purposes of representing fuzzy expressions, the designed module needed additional keywords extending the regular SQL language. The first such functionality are fuzzy prefixes, describing fuzzy columns and fuzzy values: $fc:$ and $fv:$ respectively. However, is is important to notice that the user can define those default prefix names.

The prefixes allow to use fuzzy expressions in select statements. The module also allows to add, modify and delete fuzzy sets. All fuzzy values are stored in one predefined table named $fuzzyvalues$ and defined by the following columns:

- name, CHAR(50), PRIMARY KEY
- a, REAL
- b, REAL
- c, REAL
- d, REAL

The table name can also be configured if default $fuzzyvalues$ is not suitable in particular implementation. As it was mentioned in section 2, the $name$ column is not used as a foreign key in any table, therefore, a primary key of this type is acceptable.

As it was mentioned in the introduction, the simple set of four nodes allows to define several popular types of membership functions. The approach is flexible enough to describe imprecise values and ranges, which is sufficient for the defined purposes.

Listing 2 presents three simple commands available in the module allowing to create, update and delete fuzzy values, simplifying the process of their management.

Listing 2. Fuzzy data definition commands

```
add <fuzzy value name> (<a>, <b>, <c>, <d>);
modify <fuzzy value name> (<a>, <b>, <c>, <d>);
remove <fuzzy value name>;
```

Listing 3 presents the same commands in sample usage.

Listing 3. Data definition example

```
add high_temperature (37, 38, 100, 100);
modify high_temperature (37, 38, 40, 41);
remove high_temperature;
```

2.3. OPERATORS

The module allows to use different operators between fuzzy and precise values. However, in order to perform comparison containing a fuzzy argument, a threshold t have to be defined.

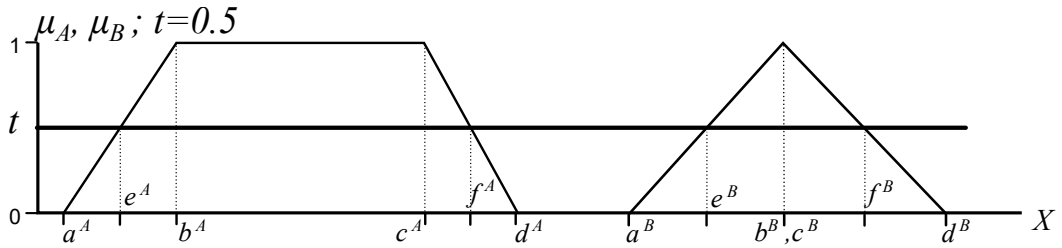

Fig. 2. Membership functions of two sample fuzzy values A,B and the threshold level t. The points e and f indicate the intersection of sets' membership functions with the threshold level.

Fig. 2 depicts two sample fuzzy values A and B, which are described respectively by the membership functions $\mu_A(x, a^A, b^A, c^A, d^A)$ and $\mu_B(x, a^B, b^B, c^B, d^B)$. As it can be observed, the threshold t, in this case equal 0.5, generates additional points e, f for each fuzzy value A and B. The points are obtained according to an intersection of the threshold level t and given membership function - as it is shown for μ_A and μ_B in fig. 2. The segment of X indicated by e and f points is nothing else than an α-cut of a fuzzy set defined by the level t [4]. Further description, if not specified, considers an α-cut obtained for the level t.

Therefore, changing the level t influences the position of e, f points. For low values of t the e and f get closer to a and d points respectively. Analogically, setting the threshold higher (near 1.0) moves e and f closer to b and c respectively. Using the definitions available within the fuzzy set theory, for $t = 0$ the e, f segment of X equals the support of the fuzzy set and for $t = 1$, the e, f indicates the core.

The threshold level is adjustable in the library and by default it is set to 0.5. The parameter directly influences the comparison operators, because the calculations are based mostly on α-cuts. Thus, the following definitions are presented using a, b, c, d, e, f points.

Operator $=$ (equal), $A = B$

Equality of two given fuzzy values A and B is assumed as any overlapping of their α-cuts, and can be defined as follows:

$$f^A \geq e^B \wedge e^A \leq f^B \tag{2}$$

Operator $!=$ (not equal), $A! = B$

Two fuzzy values A and B are assumed as not equal when their α-cuts do not overlap and can be defined as follows:

$$f^A < e^B \vee e^A > f^B \tag{3}$$

Operator $>$ (greater than), $A > B$

A fuzzy value A is greater then B when:

$$e^A > f^B \tag{4}$$

Operator $<$ (less than), $A < B$

A fuzzy value A is less then B when:

$$f^A < e^B \tag{5}$$

Operator $\geq$ (greater or equal), $A \geq B$

A combination of operators $>$ and $=$. A fuzzy value A is greater or equal B when:

$$f^A > e^B \tag{6}$$

Operator $\leq$ (less or equal), $A \leq B$

A combination of operators $<$ and $=$. A fuzzy value A is less or equal B when:

$$e^A < f^B \tag{7}$$

Operator $<<$ (contains), $A << B$

A fuzzy value A contains B when:

$$e^A \leq e^B \wedge f^A \geq f^B \tag{8}$$

And an inverse version $A >> B$, defined as follows:

$$e^B \leq e^A \wedge f^B \geq f^A \tag{9}$$

2.3.1. STRICT OPERATORS

Operators presented in previous section are designed to compare two fuzzy values (fuzzy sets) or fuzzy and precise value. The strict operators allow to compare a fuzzy value with a precise value in terms of classical logic. To distinguish such operators an exclamation mark is added as a suffix. In future implementations of the library the strict versions of operators will also be used for two fuzzy arguments and will modify the behavior defined in previous section.

For the purposes of further definitions let the A represent a fuzzy value and the x is a precise value.

Operator $=!$ **(strict equal),** $x =!A$ when

$$x \geq b^A \wedge x \leq c^A. \tag{10}$$

Operator $!=!$ **(strict not equal),** $x! =!A$ when

$$x \leq a^A \vee x \geq d^A. \tag{11}$$

Operator $>!$ **(strict greater than),** $x >!A$ when

$$x \geq d^A. \tag{12}$$

Operator $<!$ **(strict less than),** $x <!A$ when

$$x \leq a^A. \tag{13}$$

Operator $\geq!$ **(strict greater or equal),** $x \geq!A$ when

$$x \geq b^A. \tag{14}$$

Operator $\leq!$ **(strict less or equal),** $x \leq!A$ when

$$x \leq c^A. \tag{15}$$

2.4. CONFIGURATION

In order to configure the presented above variables like the threshold level, name of the table storing fuzzy values, as well as prefixes for fuzzy columns and fuzzy values, the following command was created

Listing 4. Set command

```
set <key> <value>;
```

All possible options available for the command are presented in table 1

Table 1. Module variables and their properties

variable	key	default value	range/limitations
threshold (activation level)	treshold	0.5	0.0 - 1.0
fuzzy column prefix	columnprefix	fc	max 10 characters
fuzzy value prefix	valueprefix	fv	max 10 characters
table name storing fuzzy values	fvtname	fuzzyvalues	(table name limitations)

3. QUERIES

This section presents syntax of extended SQL commands. Basically, the functionality allows to use fuzzy columns and values preserving the standard SQL syntax.

The first set of commands allow the user to create and modify tables: commands ”create” and ”alter”. The listing 5 presents the general syntax. It is important to mention, that columns used in the following definition can be fuzzy.

Listing 5. Create and alter syntax

```
CREATE TABLE <table name> ( <column name> <type> <attributes>
             [, <column name> <type> <attributes>] ... );
ALTER TABLE <table name> ADD ( <column name> <type> <attributes>
             [, <column name> <type> <attributes>] ... );
ALTER TABLE <table name> MODIFY ( <column name> <type> <attributes> );
ALTER TABLE <table name> RENAME <column name> <different name>;
ALTER TABLE <table name> DROP <column name>;
```

The approach defines a new column type named ”fuzzy”, which is nothing more than four a,b,c,d columns defined in previous sections. Listing 6 presents a couple of examples using the new type.

Listing 6. Create and alter examples

```
CREATE TABLE some_table (id INT PRIMARY KEY AUTO_INCREMENT,
                         sample_fuzzy_column fuzzy);
ALTER TABLE some_table ADD (age TINYINT NOT NULL);
ALTER TABLE some_table ADD (another_fuzzy_column fuzzy);
ALTER TABLE some_table MODIFY (age TINYINT NULL);
ALTER TABLE some_table DROP fc:another_fuzzy_column;
```

As it was mentioned in section 2-2, fuzzy columns are denoted in queries by $fc:$ prefix. Such description allows the translation mechanism to precisely recognize a fuzzy column and transform it into a query performing an operation on four physical columns (not one) storing a,b,c and d values. Without such prefix the system would have to detect if the column is fuzzy or not. The problem is much more complex in select statements, which are considered in further sections. With the prefix the detection is not necessary, the mechanism is less complicated and faster.

Most examples considering fuzzy columns shown in listing 6 does not use the prefix. It comes from the fact that in these cases there is another new keyword $fuzzy$, which is not present in regular SQL. As well as the $fc:$ prefix, it indicates the necessity of transforming a regular column into four columns of type real.

3.1. DATA MANIPULATION

Listing 7 presents three standard commands from the DML (Data Manipulation Language) of standard SQL. Similarly to the previously shown commands, fuzzy columns can be used.

Listing 7 shows only examples of fuzzy information usage, because generally the SQL syntax is preserved.

Listing 7. Create and alter syntax

```
INSERT INTO some_table (fc:fuzzy_column) values (1, 2, 3, 4);
UPDATE some_table SET fc:fuzzy_column = fv:high_pressure;
DELETE FROM some_table WHERE fc:fuzzy_column = fv:low_humidity;
```

The example presents usage of both fuzzy columns and fuzzy information. Fuzzy values, like "high_pressure", are defined and stored in table $fuzzyvalues$. Therefore, for the purposes of a proper translation, the second prefix $fv:$ is used.

The INSERT statement requires four values for each fuzzy column being inserted (the a, b, c, d describing trapezoidal membership function). UPDATE and DELETE statements present the assignment of stored fuzzy value and comparison of fuzzy column with stored fuzzy value. The WHERE part of the queries is described more precisely in the next section, concerning SELECT statement.

The examples show that the $fc:$ and $fv:$ prefixes are not an onerous demand but a clear description where fuzzy information is used within a query. The following section, considering the SELECT statement, reveals how this not complicated approach simplifies and clarifies a fuzzy query.

3.2. FUZZY SELECT

The SELECT statement is the most compound part of implementation. Fuzzy columns can occur in the list of retrieved columns (immediately after select), as well as in ORDER BY and GROUP BY clauses. However, the WHERE and HAVING clauses can contain fuzzy columns, fuzzy values and fuzzy operators defined earlier in section 2-3.

The translator was designed in such a way that the process does not influence other SQL syntax than operators and defined fuzzy columns and values. These elements occurring in the query are simply converted into real database fields and do not influence other, not fuzzy parts.

Listing 8 presents an example of retrieving an information with a condition on a fuzzy column and the resulting SQL after translation by the module.

Listing 8. Translation of fuzzy select

```
SELECT fc:col FROM fuzzytable WHERE fc:col < 6

SELECT col_f1, col_f2, col_f3, col_f4 FROM fuzzytable
  WHERE col_f4-0.5*(col_f4-col_f3)<6
```

It can be observed how a fuzzy column is converted into four values and a simple condition being transformed into an adequate expression using the default threshold level 0.5. Translation reveals the output column names of a fuzzy column, which is a set of four columns with $\_f1, \_f2, \_f3$ and $\_f4$ suffixes.

Another example, concerning a little bit more complex situation, is presented in listing 9. Again, the fuzzy SQL is shown first and the result of translation is below. It is worth to notice that the fuzzy query contains at the very end a new keyword WITH, which is an additional functionality of changing the threshold level only for that one particular query. It is set to 0.3 in this case.

Listing 9. Translation of fuzzy select, mode compound example

```
SELECT id FROM fuzzytable WHERE fc:temp = fv:cold
  OREDR BY fc:temp asc WITH 0.3
```

```
SELECT id FROM fuzzytable WHERE
  temp_f4-0.3*(temp_f4-temp_f3) >= (SELECT 0.3 * (b-a)
  + a FROM fuzzyvalues WHERE name = 'cold')
  AND 0.3*(temp_f2-temp_f1)+temp_f1<=
  (SELECT d - 0.3 * (d-c) FROM fuzzyvalues
  WHERE name = 'cold')
  ORDER BY temp_f1 asc, temp_f2 asc, temp_f3 asc, temp_f4 asc
```

4. PERFORMANCE

In order to analyze the additional time of computation provided by the translator, several tests were performed. Examinations compared execution time of a query with fuzzy SQL and regular SQL, already translated and sent to the DBMS directly. The measurement considered performing hundreds of statements launched with different parameters to avoid output caching.

First group of tests involved statements like INSERT, UPDATE and SELECT with small number of fuzzy elements like columns or values (two or three). The execution time of queries in fuzzy SQL was higher by 2%-5%. Therefore, the translation time is irrelevant in simple cases.

The highest impact was noticed in case of compound WHERE phrase of SELECT, containing around 10 fuzzy comparisons (a fuzzy column with a fuzzy value) which gives around 20 instances of fuzzy information. The execution time was in this situation higher by 8%-10%. Thus, the translation time in comparison with the overall work of a DBMS is just a small acceptable fraction.

5. CONCLUSION

The article presented the extension to the $fuzzlib$ Java library giving the functionality of a fuzzy database system based on a standard SQL. Such solution, in contrast to other publicly available approaches, is independent on the particular version or even vendor of SQL database. The tests were performed using MySQL database, however the implementation should work properly also in case of other systems.
As it was presented in the section concerning queries, the proposed syntax allows to create fuzzy expressions based on the same rules as regular precise values in standard SQL.
The user can define fuzzy values, create fuzzy columns and perform a comparison of those with discrete or fuzzy information. Fuzzy sets mapped by four floating point values will not significantly impact overall computational complexity of database queries (comparing to queries using discrete information). The additional overhead is involved only with retrieving named fuzzy values from dedicated table, however, the complexity of this process is generally much smaller than other actions performed within queries (like joins).

The main advantage of this work is involved with implementation on very popular platform, available for every advanced operating system and even contemporary mobile solutions. In addition to its low complexity and the fact that it is publicly available [5], it creates a very interesting alternative to other implementations.

Future work will focus on converting the fuzzy results of the SELECT statements, which is four columns for each one fuzzy column, into a list of dedicated objects representing fuzzy sets. Generally retrieving fuzzy columns was not the most important goal of the project, however, this functionality would be very helpful. In this context the solution can be integrated with an Object-Relational Mapping mechanism (ORM).

At the time of this publication the FUZZLIB library is publicly available for download at [5]. In case of any difficulties in accessing the module please contact the author.

6. ACKNOWLEDGEMENT

The author would like to thank Miłosz Szulik, who implemented and tested the idea of the author. Student's master thesis titled "Module of fuzzy databases development and fuzzy SQL query translation for FUZZLIB library" is available in the archives of the University of Silesia.

BIBLIOGRAPHY

[1] BALAMURUGAN V., KANNAN K. S. A framework for computing linguistic hedges. International Journal of Database Management Systems, 2010, Vol. 2.
[2] BOSC P., LIÉTARD L., PIVERT O. Sugeno fuzzy integral as a basis for the interpretation of flexible queries involving monotonic aggregates. Information Processing and Management, 2003, Vol. 39. pp. 287–306.
[3] BOSC P., PIVERT O. Fuzzy queries and relational databases. In *SAC '94 Proceedings of the 1994 ACM symposium on Applied computing*, 1994. pp. 170–174.
[4] CZOGAŁA E., ŁĘSKI J. Fuzzy and neuro-fuzzy intelligent systems. 2000. Physica-Verlag, Springer-Verlag Comp.
[5] FUZZLIB. http://fuzzlib.eu. 2016.
[6] GRISSA-TOUZI A., HASSINE M. B. New architecture of fuzzy database management systems. The International Arab Journal of Information Technology, 2009, Vol. 6.
[7] HUDEC M. An approach to fuzzy database querying, analysis and realisation. Computer Science and Information Systems, 2009, Vol. 6.
[8] J. GALINDO, J. M. MEDINA O. P. J. C. A server for fuzzy sql queries. Lecture Notes in Computer Science, 2006, Vol. 1495. pp. 164–174.
[9] KUDŁACIK P. Structure of a knowledge base in the fuzzlib library (polish). Studia Informatica, 2010, Vol. 31. pp. 469–478.
[10] MA Z. Fuzzy database modeling with xml. 2005. Springer.
[11] MA Z., YAN L. Generalization of strategies for fuzzy query translation in classical relational databases. Information and Software Technology, 2007, Vol. 49. pp. 172–180.
[12] PONS J. M. O., VILA M. Gefred. a generalized model of fuzzy relational data bases. Information Sciences, 1994, Vol. 76. pp. 87–109.
[13] S. SHENOI A. M. An extended version of the fuzzy relational database model. Information Sciences, 1990, Vol. 52. p. 35–52.
[14] ZADEH L. A. Fuzzy sets. Information and Control, 1965, Vol. 8. pp. 338–353.