How to Implement FSQL and Priority Queries

Aleksandar Takači, Srđan Škrbić

Faculty of Technology, Faculty of Natural Sciences
University of Novi Sad
atakaci@uns.ns.ac.yu, shkrba@uns.ns.ac.yu

Abstract: Intelligent databases are a developing field since the early nineties. Using fuzzy numbers to model imprecise information is a well-known concept. In this paper, we will show how to implement FSQL queries that can also handle priority. FSQL is actually a superset of SQL, thus queries that deal with fuzzy values need to be pre and post processed. We will only sketch some technical aspects of the on-going implementation process.

Keywords: intelligent databases, FSQL, priority

1 Introduction

An intelligent database system combines the functionality expected from a database system with better semantic support: new data models and, as a consequence, new functionalities. This paper deals with the application of fuzzy logic in a relational database environment. A fuzzy relational data model can be used to represent impreciseness in data values. One of the concepts used to model imprecise and uncertain information are fuzzy numbers and fuzzy quantities. As it is well known fuzzy numbers and quantities are used to model information of type "approximately 10" and "tall people". In this paper we will not discus technical details of the storing updating and deleting values from fuzzy relational databases (FRDB). Our focus will be on FSQL i.e. how to handle queries on FRDB. In addition, we will incorporate priority into our queries, which is our contribution to state of the art, because many other authors have implemented FSQL. This will be done by using a Priority Fuzzy Constrain Satisfaction Problem (pFCSP) obtained from [4].

2 Preliminaries

2.1 Fuzzy Sets and Fuzzy Logic

First, we will give a well-known definition of fuzzy sets.

Definition 2.1 A fuzzy subset P of a universe X *is described by its membership function* $\mu_P : X \to [0,1]$. *The value of the membership function denoted by* $\mu_P(x)$ *in the point x is a membership degree of the element x in the fuzzy set P.*

If a fuzzy set has a continuous unimodal (up to a point non-decreasing, then non-increasing) membership function it is called a fuzzy quantity. Fuzzy quantities are used to represent linguistic labels such as "tall people", "average salary", and "small tip". Special kind of fuzzy quantities are fuzzy numbers.

A fuzzy set that is convex, normalized and has a limited kernel is called a fuzzy number (see Figure 2.1). If a fuzzy number is upper-continuous and has a limited support then it is an L-R fuzzy number or fuzzy interval. L-R fuzzy numbers are denoted by (*L,R,F,G*) where *L* and *R* are endpoints of the kernel interval [*L*;*R*] and *F* and *G* are functions that describe the shape of the fuzzy number left and right of the kernel interval [*L*,*R*] respectively. If *F* and *G* are linear functions then we have a trapezoidal fuzzy number that we can denote (*L, ML, MR*, *R*). If *L* = *R* then we have a triangular fuzzy number with the notation (*L, M, R*) (see Figure 2.1)

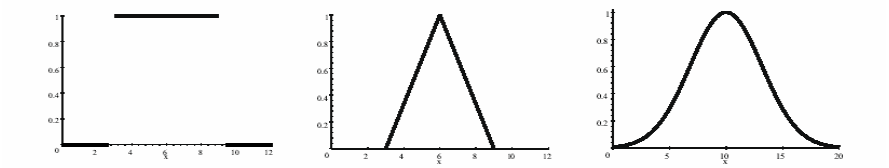

Figure 2.1

In fuzzy logic, t-norms are used as a conjunction operator. They are obtained from the following definition.

Definition 2.2 *A mapping* $T : [0,1]^2 \to [0,1]$ *is called a t-norm if the following conditions are satisfied for all* $x, y, z \in [0,1]$:

(T1) T(x, y) = T(y, x)
(T2) T(x, T(y, z)) = T(T(x, y), z)
(T3) if $y \geq z$ *then T(x, y)≥ T(x, z)*
(T4) T(x, 1) = x.

The most common t-norms are $T_m(x, y) = \min(x, y)$ the minimum, $T_p(x, y) = xy$ the product, and $T_L(x, y) = \min(x + y - 1, 0)$ the Lukasiewicz t-norm. Similarly, t-conorms are used as a disjunction operator. For more information on t-norms see [1].

Definition 2.3 *A mapping* $S : [0,1]^2 \to [0,1]$ *is called an s-norm or a t-conorm if the following conditions (T1), (T2), (T3) from the previous definition and the condition (S4) are satisfied for* $x, y, z \in [0,1]$.

(S4) S(x, 0) = x.

The most common t-conorms are $S_M(x, y) = \max(x, y)$ the maximum, $S_P(x,y) = x + y - xy$ probabilistic sum and $S_L(x, y) = \min(x + y, 1)$ the Lukasiewicz S-norm or truncated addition.

Since t-norms and t-conorms are associative, they can easily be extended to n-ary operators. For more information on t-norms see [1].

2.2 Priority Fuzzy Constraint Satisfaction Problem

The pFCSP have evolved from constraint satisfaction problems (CSP). First, fuzzy values of a constraint have been introduced into CSP's and fuzzy constraint satisfaction problems (FCSP) were obtained. Constrains are model as fuzzy sets over a particular domain. The global satisfaction degree is obtained by aggregating the values of each constraint. Adding the concept of constraint priority to FCSP, we obtain pFCSP.

Each constraint is modeled as a fuzzy number. The membership degree of each constraint value indicates the local degree to which the constraint is satisfied with a compound label. In order to obtain the global satisfaction degree, local degrees are aggregated together with the priority of each constraint. In order for a system to be a pFCSP, it needs to satisfy some axioms. For more details on axioms, see [2]. A system that is a pFCSP is given in the following definition:

Definition 2.4 Let $C = \{C_i, i \in 1 \ldots n\}$ be the set of constraints. Let c_i be the local satisfaction degree of constraint C_i and p_i be the priority of constrain C_i. Then the global satisfaction degree in is a pFCSP is obtained by using this formula:

$$\alpha = T_L(S_P(c_i, p_i), i \in \{1, \ldots, n\})$$

2.3 Fuzzy Relational Databases and FSQL

The relational model uses a collection of tables to represent data and relationships among those data. In our model, data values need not be exact. We can handle imprecise and uncertain information using fuzzy numbers and quantities. First, for each attribute we specify weather it can have fuzzy values or not. Obviously, keys cannot have fuzzy values. Fuzzy numbers are linguistic labels on the domain of the attribute e.g. “tall people” and “average salary”. Before using a linguistic label, it needs to be predefined. Besides linguistic labels our attributes can have interval values from the domain e.g. "height= [173,180]".

SQL is the most influential commercially marketed database query language. It uses a combination of relational algebra and relational calculus constructs to retrieve desired data from a database (see [3]). FSQL is SQL that can handle fuzzy attribute values. The main difference between SQL and FSQL is that SQL returns a subset of the database as the query result. When attributes with fuzzy values appear in the query, it is transformed into a query that can be handled by SQL and finally results obtained from the SQL query are then post processed in order to obtain the desired information. More details are given in the following section.

3 Processing FSQL Queries

First, we will give the syntax of our FSQL query.

SELECT atributeList

FROM tableNameList

[WHERE conditionList]

[GROUP BY atributeList

HAVING conditionList]

[TRESHOLD number]

We will discuss the case when in the WHERE condition list fuzzy values appear without the GROUP BY clause. The idea is to push fuzzy attributes that appear together with the AND operator in the WHERE clause, up into the SELECT clause. When fuzzy attributes appear together with the AND operator in the where clause, they are pushed up into the SELECT clause. Take an example:

SELECT name

FROM students

WHERE height="tall" AND age="young" AND name="Peter"

This query is transformed into

SELECT name, age, height

FROM students

WHERE name="Peter"

If it is possible to isolate the non-fuzzy conditions, they are put into the transformed SQL query. Then this transformed query runs on the database returning a data set as the result. Then this data set is post-processed by counting the values of the remaining conditions. Each dataset gets a value in the unit interval, which is actually its satisfaction degree of the query, and they are displayed in the result. If the threshold clause is active then the rows in the dataset with smaller satisfaction degrees are cut i.e. they are not displayed in the result.

In case of the OR, NOT operator we cannot do much but process the SQL query without conditions and then check each condition whether it is fuzzy or not.

One of the main technical aspects of the implementation is how do we obtain the satisfaction degree for each row in the data set. If we have more that one condition connected by AND, OR operators we need to calculate satisfaction degrees for each condition separately and then these local satisfaction degrees are aggregated using t-norms in case of the AND operator and t-conorms in case of the OR operator.

When we count the local satisfaction degree of each constraint many cases can appear. Our calculations are based on the surface intersection method for determining the compatibility of fuzzy sets. First, we will consider the case when in the condition a crisp attribute value appears e.g. "AGE=25". Obviously when there is a crisp value in the database, there is no problem. On the other hand, if there is an interval value in the database we return the probability that if the value is in the interval it is actually equal to our crisp value. This value is actually counted depending of the precision of our crisp data e.g. if integer is the domain the probability that "age=25" if the age is in the interval [24, 26] is 1/3. Finally, if the data value is a fuzzy number or quantity the result is calculated by determining the compliance of the set determined from our crisp value with the fuzzy quantity ex. "age=25" transforms into [24.5,25.5] complies with "young" 0.1.

If the value of the fuzzy attribute in the query is an interval value, we have also three cases. First, if the data value is crisp the satisfaction degree is one. Then, if the data value is the interval we count the result using the formula

$$\text{degree} = \frac{\text{length}(A \cap B)}{\text{length}(A)}$$

e.g. "age= [24, 26]" has the satisfaction degree of 0.5 of the condition "age= [25, 27]". Finally, if the data is a fuzzy quantity we transform the interval into a fuzzy set and then counts the compliance degree.

The most interesting case in the queries will be when the value of the fuzzy attribute is a fuzzy quantity. If the value of the data is crisp then the satisfaction degree is actually the membership degree of that value. If the value is an interval it is transformed in a uniformly distributed fuzzy number where each point has the membership degree of $\frac{1}{\text{length}(I)}$ and then the compliance degree of two fuzzy sets is calculated. Finally, if the attribute value is a fuzzy set the compliance degree of two fuzzy sets is calculated.

Now let us discuss the case when functions (MIN, MAX, AVG) appear in the select statement together with fuzzy attribute values. In this case, we have to demand that the threshold clause exists because we need the actual data set to find the function values. If there is no threshold, we can simply remove the fuzzy condition from the query since it holds no restrictions. In order to process these types of queries, we first need to remove the functions from SELECT statement and calculate them later on the dataset obtained from the actual query. For example the query:

SELECT MIN(age)

FROM students

WHERE gpa="good"

TRESHOLD 0.5

is transformed into

SELECT age, gpa

FROM students.

If there is a GROUP BY clause present in the query we remove it but later we calculate function results for each specified values in the clause. If the HAVING condition exists, then calculated values are checked at the end to obtain the proper result.

3.1 Adding Priority to FSQL

One of the contributions of the authors to state of the art is adding the priority of the condition into database queries. Although we can put individual thresholds on each condition this is not the same as having different priorities for each condition in the where and having statement. A theoretical background to implement a system that calculates the global satisfaction degree of prioritized conditions is

given in [4]. The syntax of pFSQL is the same as FSQL only the terminal symbol *conditionList* has the following form:

conditionList=conditionList| *condition*

condition= identifier operator value [PRIORITY number].

This is just an illustration of general much more complicated syntax of SQL.

Conclusions

In this paper an idea how to implement FSQL is given. We have shown only a sketch of a basic idea not going into the technical details. In addition, in our implementation we did not discuss nested queries. This can be done but, in this faze, we decided not to do this.

Implementation of pFSQL will be one of the first of its kind. Using pFCSP many interesting queries can be processed adding a new perspective to data mining.

Acknowledgements

This paper was written with the support of the project "Mathematical Models for Decision Making under Uncertain Conditions and Their Applications" by Academy of Sciences and Arts of Vojvodina supported by Vojvodina Provincial Secretariat for Science and Technological Development.

References

[1] E. Klement, R. Mesiar, E. Pap: Triangular norms, Series: Trends in Logic, Kluwer Academic Publishers, Vol. 8, Dordrecht, 2000

[2] X. Luo, J. H. Lee, H. Leung, and N. R. Jennings: Prioritized fuzzy constraint satisfaction problems: axioms, instantiation and validation, Fuzzy Sets and Systems 136 (2003) 151-188

[3] A. Silberschatz, H. F. Korth, S. Sudarshan: Database System Concepts, McGraw-Hill Higher Education, New York, 2002

[4] A. Takači, Schur-concave triangular norms: characterization and application in pFCSP, Fuzzy Sets and Systems (in press)