An Interpreter for Priority Fuzzy Logic Enriched SQL

Srdan Škribić
Faculty of Science
Novi Sad, Serbia
Email: shkrba@uns.ac.rs

Aleksandar Takači
Faculty of Technology
Novi Sad, Serbia
Email: stakaci@tehnol.ns.ac.yu

Abstract—In this paper we describe a query language called PFSQL (Priority Fuzzy Structured Query Language) that, beside basic fuzzy capabilities, adds the possibility to specify priorities for fuzzy statements. We give an analysis of possibilities to extend the SQL language and briefly describe the GPFCSP (Generalized Prioritized Fuzzy Constraint Satisfaction Problem) systems, a theoretical concept vital for the implementation of the PFSQL interpreter. In the following, we describe the most important ideas used in the process of the interpreter implementation. As one of the most innovative ideas we present the fuzzy JDBC (Java Database Connectivity) driver as a framework for the PFSQL interpreter.

I. INTRODUCTION

In this section we will shortly describe the notion of FRDB (Fuzzy-Relational Databases) and PFSQL, and give some pointers on how the fuzzy logic enriched databases have been developing in the past. Relational databases are not suitable for representation of imprecise, uncertain or inconsistent information, thus they require add-ons to handle these types of information. The most common add-on is to allow the attributes to have values that are fuzzy sets on the attribute domain, which results in FRDB. The idea to use fuzzy sets and fuzzy logic to extend existing database models to include these possibilities has been utilized since the 1980s. Although this area has been researched for a long time, implementations are rare.

Literature contains references to several models of fuzzy knowledge representation in relational databases. For example, the Buckles-Petry model [1] is the first model that introduces similarity relations in the relational model. The GEFRED (Generalized Model of Fuzzy Relational Databases) model [2] is a probabilistic model that refers to generalized fuzzy domains and admits the possibility distribution in domains. It includes the case where the underlying domain is not numeric, but contains scalars of any type. Also, it contains the notion of unknown, undefined and null values. It experienced subsequent expansions, such as [3], [4]. Zvieli and Chen [5] offered a first approach to incorporate fuzzy logic in ER (Entity-Relationship) model. Their model allows fuzzy attributes in entities and relationships. Chen and Kerre [6] introduced the fuzzy extension of several major EER concepts. Chaithry, Moyne and Rundensteiner [7] proposed a method for designing fuzzy relational databases following the extension of the ER model of Zvieli and Chen. They propose a way to convert a crisp database into a fuzzy one.

Galindo, Urrutia and Piattini [8] describe a way to use fuzzy EER (Extended Entity-Relationship) to model the database and how to represent modeled fuzzy knowledge using relational database in detail. In addition they describe specification and implementation of the FSQL - an SQL language with fuzzy capabilities.

In [9] and [10] authors have studied the possibilities to extend the relational model with fuzzy logic capabilities. The subject was elaborated in [11], where a detailed model of fuzzy relational database was given. Moreover, using the concept of GPFCSP from [12] and [13] authors have found a way to introduce priority queries into FRDB, which resulted in the PFSQL query language. In [14] and [15] authors introduce similarity relations on the fuzzy domain which are used to evaluate FRDB conditions.

In this paper we present the basics of the PFSQL - an SQL extended using priority fuzzy logic. We give an overview of the GPFCSP - a theoretical background for the construction of the interpreter for the PFSQL. Next, we give some pointers and ideas about the process of the interpreter implementation. We present the fuzzy JDBC driver as a framework for the PFSQL interpreter. In the end, the most important differences between the PFSQL and the FSQL, one of the most advanced languages of this kind today, are explained.

II. THE PFSQL

In order to allow the use of fuzzy values in SQL queries, we extended the classical SQL with several new elements. In addition to SQL fuzzy capabilities already described and implemented by other authors, we add the possibility to specify priorities for fuzzy statements. We named the query language constructed in this manner priority fuzzy SQL - PFSQL. This appears to be the first implementation that has such capabilities.

A. PFSQL features

The question is what elements of the classical SQL should be extended. Because variables can have both crisp and fuzzy values, it is necessary to allow comparison between different types of fuzzy values as well as between fuzzy and crisp values. In other words, PFSQL has to be able
to calculate expressions like \( \text{height} = \text{triangle}(180, 11, 8) \) regardless of what value of height is in the database - fuzzy or crisp. Expression \( \text{triangle}(a, b, c) \) denotes triangular fuzzy number with peak at \( a \), with left offset \( b \), and right offset \( c \). Next, we demand the possibility to set the conditions like: 

\[ \text{height} > \text{triangle}(180, 11, 8) \].

As we mentioned before, PF- SQL allows priorities to be specified for every fuzzy constraint. In other words, every condition, like the ones we stated in this paragraph, can be added that defines an expression that adds its priority: 

\[ \text{height} > \text{triangle}(180, 11, 8) \ \text{priority} \ 0.6 \].

Priorities are normalized, so the minimum priority is 0, and maximum priority is 1.

Beside the aggregate function COUNT, ordering operations on the set of fuzzy numbers give grounds for the introduction of aggregate functions MIN and MAX in the PFSQL. The classical SQL includes possibilities to combine conditions using logical operators. This possibility also has to be a part of fuzzy extensions, thus combining fuzzy conditions is also a feature of PFSQL. Values are calculated using triangular norms, conorms, and strict negation [12]. Queries are handled using priority fuzzy logic which is based on GPFCSP systems.

Nested queries are yet another problem that we encountered in our effort to extend SQL with fuzzy capabilities. We can divide nested queries in two categories - ones that do not depend on the rest of the query and the ones that do. Independent SQL queries are not problematic, they can be calculated separately, and resulting values can be used in the remainder of the query as constants. Dependent SQL queries with dependence expressions that do not use fuzzy values or operators are also easy to handle - they can be evaluated by a classical SQL interpreter. However, if a nested query is dependent and dependence conditions contain fuzzy values or operators, then it remains unclear how to evaluate such a query and what does this dependence mean.

Complete EBNF (Extended Backus-Naur Form) syntax of the PFSQL language can be found in [16]. For the sake of brevity, we do not present all of the features of the language here. Instead, we present some examples of the queries that give an overview of the possibilities and extensions built into the PFSQL. The queries are executed against a hypothetical student database.

The first query returns names and surnames of students whose GPA is greater than the given triangular fuzzy number:

\[
\text{SELECT msd.name, msd.surname FROM MainStudentData msd WHERE (msd.GPA}\geq\#\text{ling(averageGPA)}\# \text{ PRIORITY 0.7)}
\]

The \# symbol is chosen to mark fuzzy constants. If we defined a linguistic label “average GPA” that has value triangle(9,1,0.4), the previous query could have been simplified like this:

\[
\text{SELECT msd.name, msd.surname FROM MainStudentData msd WHERE (msd.GPA}\geq\#\text{ling(averageGPA)}\#}
\]

Queries like these can be enriched with additional constraints. The next query returns names and surnames of students that have GPA greater than average with priority 0.7, and GPA on the fourth year greater than 8.5 with priority 0.4. The query also contains the threshold clause that limits the results and removes tuples with fuzzy satisfaction degree smaller than 0.2.

\[
\text{SELECT msd.name, msd.surname FROM MainStudentData msd WHERE (msd.GPA}\geq\#\text{ling(averageGPA)}\# \text{ PRIORITY 0.7) AND (msd.GPA}\geq\#8.5\# \text{ PRIORITY 0.4) THRESHOLD 0.2}}
\]

As we already mentioned, aggregate functions MAX, MIN and COUNT can take fuzzy value as an argument. The next query illustrates the usage of aggregate function MIN. It returns the minimal GPA.

\[
\text{SELECT MIN(msd.GPA) FROM MainStudentData msd}
\]

If we assume that the variable msd.GPA is fuzzy, execution of this query becomes complex because it includes the ordering of fuzzy values. As a result, for example, we could get this value: \( \text{triangle}(6.9, 0.4, 0.7) \).

### III. The Interpreter

The basic difference between SQL and PFSQL mechanisms is in the way the database processes records. In a classical relational database, queries are executed so that a tuple is either accepted in the result set, if it fulfills the conditions given in a query, or removed from the result set, if it does not fulfill the conditions. In other words, in classical databases, every tuple is given a value true (1) or false (0). On the other hand, as the result set, the PFSQL returns a fuzzy relation on the database. Every tuple considered in the query is given a value from the unit interval. This value is calculated using GPFCSP systems.

#### A. GPFCSP

If we interpret fuzzy constraints as conditions in a WHERE clause of an SQL query, then we can use PFCSP (Priority Fuzzy Constraint Satisfaction Problem) [13] as formal grounds to introduce priorities in PFSQL queries. The problem lies in the fact that PFCSP systems only allow and describe the use of conjunction, i.e., t-norm, over constraints. That means that corresponding PFSQL statements could only use the AND operator. Obviously, we need to generalize the PFCSP idea in order to allow usage of disjunction and negation. The result of this generalization is GPFCSP [17]. Only GPFCSP systems give theoretical backgrounds for fuzzy SQL language with priorities - PFSQL. Formal definition and detailed explanation of GPFCSP systems can be found in [17] and [18].

#### B. Query execution

The PFSQL implementation relies upon a meta data about fuzzy attributes that reside inside the database. For these purposes, a FRDB data model has been defined. Here we only give some pointers necessary to understand the basics about the model. The model introduces possibilities to store well known standard types of fuzzy sets (triangular, trapezoidal etc.) as attribute values. If a type of a fuzzy set is introduced, then we only need to store the parameters that are necessary
to calculate the value of its membership function. In order to represent these fuzzy values in the database, we extend every data model with additional tables that make fuzzy meta data model. Several tables are introduced to cover all described needs. Every fuzzy value in every table in a model is a foreign key that references a table in the fuzzy meta model suitable for storing fuzzy data of its type. For every type of fuzzy value there is a separate table in the meta model that stores data for the specific fuzzy type. Complete description of the model can be found in [11] and [16].

We will now describe processes that allow PFSQL queries to be executed. The basic idea is to first transform the PFSQL query into something that a classical SQL interpreter understands. Namely, conditions with fuzzy attributes are removed from the WHERE clause and moved up in the SELECT clause. In this way, conditions containing fuzzy constructs are eliminated, so that the database will return all the tuples - ones that fulfill fuzzy conditions as well as the ones that do not. As a result of this transformation, we get a classical SQL query (Fig. 1).

Then, when this query is executed against the database, results are interpreted using fuzzy mechanisms. These mechanisms assign a value (membership degree) from the unit interval to every tuple in the result set. If a threshold is given, all the tuples in the result set that have satisfaction degree below the threshold are removed.

In the classical SQL it is clear how to assign truth value to every elementary condition. With the fuzzy attributes, the situation becomes more complex. The original PFSQL query tree is traversed recursively, and in the process, fuzzy satisfaction degrees are added to every tuple in the result set. At first, we assign a truth value from the unit interval to every elementary condition. The only way to do this is to give a set of algorithms that calculate truth values for every possible combination of values in a query and values in the database. For instance, if a query contains a condition that compares a fuzzy quantity value to a triangular fuzzy number in the database, we must have an algorithm that calculates the compatibility of the two fuzzy sets. After the truth values from the unit interval are assigned, they must first be aggregated with their priorities.

The GPFCSP system describes a way to do this. In our implementation, this truth value is calculated as follows: \( v = S_P(x, 1 - p) \), where \( x \) is the truth value of the condition, \( p \) is the priority, and \( S_P \) is the product t-conorm. Values obtained in this way can be connected with logical operators. In that case they are further aggregated using fuzzy logic. We use the \( T_L \) t-norm in case of operator AND, and its dual t-conorm \( (S_L) \) in case of operator OR. For negation we use the strict negation. The whole process is presented as activity diagram at Fig. 2.

C. Fuzzy JDBC driver

There are two main requirements set before the implementation of described PFSQL interpreter. The first one is database independence. The system has to work with any given relational database that supports standard SQL. This immediately implies that all fuzzy processing has to be done outside the database. The second important requirement is easy access from Java that is as similar as possible to JDBC API. These requirements are resolved with the implementation of the fuzzy JDBC driver. This driver acts as a wrapper for the PFSQL processing mechanisms described in the second section and for the JDBC API implemented by the driver for a specific DBMS (Database Management System). JDBC driver for the database used simply becomes a parameter that the fuzzy JDBC driver uses to access the database. The architecture of the system built in this way is shown at Fig. 3.

Java program uses interfaces offered by the fuzzy JDBC driver as a front end component. These interfaces include possibilities to:

- initialize driver class,
create database connection,
create and execute PFSQL statements and
read result set.

When executed, PFSQL statements are preprocessed in the way described earlier, and sent to the database as ordinary SQL statements using a JDBC driver. Result returned from the database is processed again by the PFSQL mechanisms (membership degrees are added), and returned to the Java program using front end classes. In this way, user gets all the tuples that satisfy non-fuzzy part of the query, and every one of them has associated fuzzy membership degree. This membership degree specifies how much does the tuple belong to the result set. All tuples that have membership degree equal or greater than the given threshold (0 by default) are shown. If a tuple has membership degree 0 it is still shown, because it belongs to the fuzzy set with membership degree 0.

In the process of implementation we used Java programming language, together with JavaCC, a Java compiler-compiler. A working version of the interpreter is available for download from http://www.is.pmf.uns.ac.rs/fuzzydb.

IV. CONCLUSION AND COMPARISON

Described implementation is tested on the fuzzified database segment of the student affairs information system of the Faculty of Science in Novi Sad. This segment is related to examination management. It consists of about 30 relational tables with more than 5000 records in most of them. We do not give detailed analysis of the results here, but it can be concluded that performance of the system is satisfactory. Of course, the system is slower than direct SQL query execution against the database, but it is still very usable. The queries that have less than five fuzzy conditions were processed in less than 5 seconds. Significant slowdown is observed only when fuzzy ordering is used.

Here we give a brief comparison of the PFSQL to the FSQL, one of the most advanced fuzzy SQL languages today. The fuzzy database query language FSQL is built on top of the FIRST-2 model using Oracle DBMS and PL/SQL stored procedures [8]. Similarly, we used our own fuzzy-relational data model described in [10] and [11] to build an interpreter for the PFSQL language. We have developed the PFSQL query language from ground up, extending the features of SQL into the fuzzy domain. This query language allows priority statements to be specified for query conditions, which is a new feature that FSQL does not contain. For calculating the membership degree of query tuples when priority is assigned to conditions, we use the GPFCSP, theoretical concept developed just for these purposes. Although the FSQL language has more features than the PFSQL, it does not allow usage of priority statements. The PFSQL is the first query language that does. Moreover, the PFSQL is implemented using Java, outside the database, which makes our implementation database independent.

In order to offer a more complete solution for the fuzzy relational database application development, it is necessary to enrich the PFSQL language with more features of a regular SQL, such as insert, update and delete statements. In addition, the fuzzy JDBC driver has to be augmented with other interfaces and possibilities offered by the JDBC specification. Authors intend to study and solve these problems in the future.

ACKNOWLEDGMENT

Authors would like to acknowledge the support of the Serbian Ministry of Science, project "Mathematical models of non-linearity, uncertainty and decision making", No. 144012 and project "Abstract Methods and Applications in Computer Science" No. 144017A, and also the support of the Provincial Secretariat for Science and Technological Development of Vojvodina.

REFERENCES


