On Development of Fuzzy Relational Database Applications

Srdjan Skrbic
Faculty of Science
Trg Dositeja Obradovica 3
21000 Novi Sad
Serbia
shkrba@uns.ns.ac.yu

Aleksandar Takači
Faculty of Technology
Bulevar Cara Lazara 1
21000 Novi Sad
Serbia
stakaci@tehnol.ns.ac.yu

Abstract

In this paper we examine possibilities to extend relational data model with mechanisms that can handle imprecise, uncertain and inconsistent attribute values using fuzzy logic. We present a model for fuzzy knowledge representation in relational databases and describe PFSQL – a priority fuzzy logic enriched SQL. We give a brief description of fuzzy JDBC driver and FRDB CASE tool that make complete set of tools needed to develop Java fuzzy relational database application. In addition, a brief survey of research related to application of fuzzy logic in relational databases is given in the introduction. Authors propose several points in which this research and implementation can be continued and extended, contributing to better understanding of fuzzy database concepts and techniques.

Keywords: fuzzy relational database, priority fuzzy SQL, fuzzy JDBC driver.

1 Introduction

One of the disadvantages of the relational model is its disability to model uncertain and incomplete data. 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.

The Buckles-Petry model [6] is the first model that introduces similarity relations in the relational model. This paper gives a structure for representing inexact information in the form of a relational database. Zvieli and Chen [5] offered a first approach to incorporate fuzzy logic in the ER model. Their model allows fuzzy attributes in entities and relationships.

The GEFRED (Generalized Model of Fuzzy Relational Databases) model [11] is a possibilistic model that refers to generalized fuzzy domains and admits the possibility distribution in domains. This is a fuzzy relational database model that has representation capabilities for a wide range of fuzzy information. In addition, it describes a flexible way to handle this information. The GEFRED model experienced subsequent expansions, such as [8] and [9].

Chen and Kerre [7] introduced the fuzzy extension of several major EER concepts. Fuzzy logic was applied to some of the basic EER (Extended Entity-Relationship) concepts connected to the notion of subclass and superclass. Chaudhry, Moyne and Rundensteiner [12] proposed a method for designing fuzzy relational databases following the extension of the ER model of Zvieli and Chen. They also proposed a design methodology for FRDBs, which contains extensions for representing the imprecision of data in the ER model, and a set of steps for the derivation of a FRDB from this extended ER model.

Galindo, Urrutia and Piattini [10] describe a way to use fuzzy EER model to model the database and represent modelled fuzzy knowledge using relational database in detail. This work gives insight into some new semantic aspects and extends the EER model with fuzzy capabilities. Devised model is called FuzzyEER model. Also, a way to translate FuzzyEER model to the FIRST-2, a database schema that allows representation of fuzzy attributes in relational databases is given. In addition, in this work, authors introduce and describe specification and implementation of the FSQL – an SQL language with fuzzy capabilities in great detail.

In [2] authors have studied the possibilities to extend the relational model with fuzzy logic capabilities. The subject was elaborated in [4], where a detailed model of fuzzy relational database was given. Moreover, using the concept of Generalized Priority Constraint Satisfaction Problem (GPFCSP) from [1] and [13] authors have found a way to introduce priority queries into FRDB, which resulted in the PFSQL query language. In [3] authors introduce similarity relations on the fuzzy domain which are used to evaluate FRDB conditions. PFSQL allows the conditions in the WHERE clause of the query to have different priority i.e. importance degree. The GPFCSP gives the theoretical background for the implementation of priority queries. This is one of the first languages with such capabilities.

In this paper, we focus on an effort to produce a complete solution for a fuzzy relational database application development. We describe the architecture of the PFSQL implementation, and the data model that this implementation is based on. Furthermore, we give a brief description of a fuzzy JDBC driver and a FRDB CASE tool. Together, these components make a set of tools that allow and facilitate development of FRDB applications. We describe the features and basic principles of every component of this system, but technical details about the implementation are far beyond the scope of this paper.

2 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 fuzzy capabilities that make the fuzzy SQL - FSQL, 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.

The basic difference between SQL and PFSQL 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, 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 fuzzy logic operators.

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

height=triangle(180,11,8)

regardless of what value of height is in the database – fuzzy or crisp. Expression *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

height<triangle(180,5,5).

The Ordering and addition operations on the set of fuzzy numbers give grounds for the introduction of set functions like MIN, MAX and SUM in the PFSQL. Moreover, it is possible to define the fuzzy GROUP BY clause in combination with the aggregate functions on fuzzy values.

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 our implementation. Values are calculated using t-norms, t-conorms, and so called “strict” negation. Queries are handled using priority fuzzy logic which is based on the 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.

We do not present the PFSQL EBNF syntax here because of its voluminosity, but it can be acquired in electronic form from authors.

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. At first, we assign a truth value from the unit interval to every elementary condition. The only way to do this is to give 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 with a triangular fuzzy number in the database, we must have algorithm to calculate the compatibility of the two fuzzy sets. After the truth values from the unit interval are assigned, they are aggregated using fuzzy logic. We use a t-norm in case of operator AND, and its dual t-conorm in case of operator OR. For negation we use strict negation: $N(x) = 1 - x$. In case of priority statements, we use the GPFCSP system rules to calculate the result.

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. 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.

A more detailed description of the PFSQL language and mechanisms of its implementation can be found in [4].

3 FRDB Data Model

It is clear now that the PFSQL implementation has to rely upon a meta data about fuzzy attributes that reside inside the database. For these purposes, a FRDB data model has been defined. In this section we give a brief description of this model.

Our FRDB data model allows data values to be any fuzzy subset of the attribute domain. User only needs to specify a membership function of a fuzzy set. Hypothetically, for each fuzzy set we should have an algorithm on how to calculate the values of its membership function. This would lead to a large spatial complexity of the database. Most often, this is solved by introducing 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 the membership function. This is the most common way to implement FRDB, and we used it in our model also.

On the other hand, we did not want to restrict ourselves to these particular fuzzy sets, so we allow users to specify a general membership function for each attribute value. Our idea is to have the most common fuzzy set types implemented and that the attribute values in FRDB are most often standard fuzzy sets, and only a small percentage of attribute values are generalized fuzzy sets specified by user, though our model works with general fuzzy sets in every aspect of FRDB - storing, querying, etc.

We introduce one more extension of the attribute value, the linguistic label. Linguistic labels are used to represent the most common and widely used expressions of a natural language such as “tall people”, “small salary” or “mediocre result”. Linguistic labels are in fact named fuzzy values from the domain.

Considering these extensions, we can define a domain of a fuzzy attribute as follows:

$$D = D_C \cup F_D \cup L_L$$

where D_C is a classical attribute domain, F_D is a set of all fuzzy subsets of the domain, and L_L is the set of linguistic labels.

In order to represent these fuzzy values in the database, we extend this model with additional tables that make fuzzy meta data model. Several tables are introduced to cover all described needs.

One of these tables is created for the purpose of storing the data whether an attribute is fuzzy or not. All attribute names in the database are stored here, and beside the table and attribute name, we have information whether the attribute is fuzzy or not. The main table in the meta model represents a connection between fuzzy data model and fuzzy data meta model. Every fuzzy value in every table is a foreign key that references table’s primary key attribute. Thus, we have one record in this table for every record with the fuzzy value in the database. Another one of its attributes is a foreign key from the table that stores information on fuzzy types. This table stores names of every possible types of fuzzy values allowed in the model.

For every type of fuzzy value there is a separate table in the meta model that stores data for a specific fuzzy type. Every one of these tables has a foreign key attribute from the main table in the meta model. In this way, a value for a specific fuzzy attribute is stored in one of these tables depending on its type.

In order to represent linguistic labels, we introduce another attribute in the main table as a foreign key that represents recursive relationship and references the table’s primary key. This attribute is used to represent linguistic labels. It has a value different then *null* if the type of the attribute that it represents is a linguistic label. As mentioned before, linguistic labels only represent names for previously defined fuzzy values. In this fashion, if an attribute is a linguistic label, then its name is stored in the table specialized for storage of linguistic labels.

Complete description of all values and types that can be stored in the database can be found in [4].

4 Fuzzy JDBC Driver and FRDB CASE Tool

The need to ease the PFSQL usage from Java programs and still keep database independence is 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 RDBMS. 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 Figure 1.

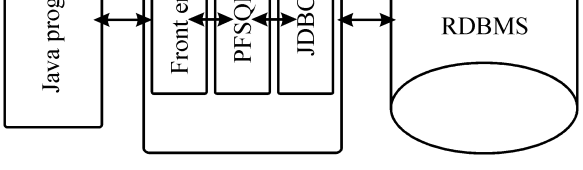

Figure 1: Fuzzy JDBC driver.

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 pre-processed in the way described in the section 2, 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.

The Fuzzy JDBC driver with PFSQL mechanisms and the FRDB data model described above offer a complete solution to develop database applications when a database model exists in the database. In order to ease the development of data models enriched with fuzzy elements as described in section 3, a CASE tool is implemented.

This CASE tool is a standalone Java application that offers automatic DDL (Data Definition Language) code generation for fuzzy relational data models with respect to the fuzzy meta data described in section 3.

Fuzzy relational tables supported by the CASE tool can contain regular relational attributes of any (user defined) data type and fuzzy attributes introduced as a new data type. Primary key consisting of non fuzzy attributes can be specified for every table. There are two types of relationships between tables (foreign keys) – identifying and non-identifying. Identifying relationship makes foreign key attributes part of the child table primary key, while non-identifying relationship does not.

For a given fuzzy data model constructed in this way, the CASE tool generates complete fuzzy meta data structure, integrates it into the model and generates DDL file. The generation process is parameterized so that it can generate DDL file using SQL dialect supported by any specific RDBMS.

5 Conclusion

In this paper we give a brief overview of research conducted in the field of fuzzy databases. We present a variant of the SQL language enhanced with fuzzy logic and a concept of priority. Implementation of this PFSQL is in close connection with the data model that extends the relational model with capabilities to store fuzzy values.

Comparing our data model with one of the most advanced fuzzy relational data models – the FIRST-2, leads to a conclusion that there are similarities between the two. Although the methods for fuzzy value representation are completely different, functionally, our model is a subset of the FIRST-2 model. Our intention was to define the simplest possible model that supports the most widely used fuzzy concepts, and stores values as effectively as possible without too much overhead. At the same time, the model had to include all the features necessary to implement the PFSQL interpreter.

We have developed the PFSQL query language from ground up, extending the features of SQL with fuzzy logic. Among other features already present in other fuzzy query languages, this query language allows priority statements to be specified for query conditions. Membership degrees of query tuples are calculated using the GPFCSP system. The PFSQL is the first query language that introduces such capabilities. Moreover, the PFSQL is implemented using Java, outside the database, which makes our implementation database independent.

A set of tools that facilitate fuzzy relational database applications development consisting of a fuzzy JDBC driver and a FRDB CASE tool is described in continuation. To the best of our knowledge, these tools are the only ones with such capabilities today.

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 API specification. Authors intend to study and solve these problems in the future.

Acknowledgements

Authors would like to acknowledge the support of the Serbian Ministry of Science and Environmental Protection, 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 Ministry of Science, Technology and Environmental Protection of Vojvodina.

References

[1] A. Takači (2005). Schur-concave triangular norms: characterization and application in PFCSP. *Fuzzy Sets and Systems,* no. 155, volume 1, pages 50-64.

[2] A. Takači, S. Skrbic (2005). How to Implement FSQL and Priority Queries. *Proceedings of the 3rd Serbian-Hungarian Joint Symposium on Intelligent Systems*. Pages 261-267, Subotica, Serbia.

[3] A. Takači, S. Skrbic (2007). Measuring the similarity of different types of fuzzy sets in FRDB. *Proceedings of the EUSFLAT-LFA*. Pages 247-252, Ostrava, Czech Republic.

[4] A. Takači, S. Skrbic (2008). Data Model of FRDB with Different Data Types and PFSQL. *In Handbook of Research on Fuzzy Information Processing in Databases*. Ed. J. Galindo. Pages 403-430, Hershey, PA, USA, Information Science Reference.

[5] A. Zvieli, P. Chen (1986). ER modelling and fuzzy databases. In *Proceedings of the Second International Conference on Data Engineering*. pages 320-327, LA, USA.

[6] B.P. Buckles, F.E. Petry (1982). A fuzzy representation of data for relational databases. *Fuzzy Sets and Systems* no. 7, pages 213-226.

[7] G.Q. Chen, E.E. Kerre (1998). Extending ER/EER concepts towards fuzzy conceptual data modelling. *Proceedings of the IEEE International Conference on Fuzzy Systems*. Pages 1320-1325, Anchorage, AK, USA.

[8] J. Galindo, J.M. Medina, M.C. ARANDA, (1999). Querying fuzzy relational databases through fuzzy domain calculus. *International Journal of Intelligent Systems*, no. 14, volume 4, pages 375-411.

[9] J. Galindo, J.M. Medina, J.C. Cubero, M.T. Garcia, (2001): Relaxing the universal quantifier of the division in fuzzy relational databases. *International Journal of Intelligent Systems*, no. 16, volume 6, pages 713-742.

[10] J. Galindo, A. Urrutia, M. Piattini (2006). *Fuzzy Databases: Modelling Design and Implementation*. Hershey, USA: IDEA Group.

[11] J.M. Medina, O. Pons, M.A. Vila (1994). GEFRED: A Generalized Model of Fuzzy Relational Databases. *Information Sciences*, no. 76, pages 87-109.

[12] N. Chaudhry, J. Moyne, E. Rundensteiner (1994). A design methodology for databases with uncertain data. *Proceedings of the Seventh International Working Conference on Scientific and Statistical Database Management*. Pages 32-41, Charlottesville, VA, USA.

[13] X. Luo, J.H. Lee, H. Leung, N.R. Jennings (2003): Prioritized fuzzy constraint satisfaction problems: axioms, instantiation and validation. *Fuzzy Sets and Systems,* no. 136, pages 151-188.