Prioritized fuzzy logic based information processing in relational databases

Srdjan Škrbić, Miloš Racković, Aleksandar Takači

University of Novi Sad, Faculty of Science, Department of Mathematics and Informatics, Trg Dositeja Obradovića 4, Novi Sad, Serbia

Article history:
Available online 8 February 2012

Keywords:
Fuzzy
Relational
Database
Application
Development

ABSTRACT

Many years of research related to fuzzy logic and fuzzy set theory extensions to relational databases have not lead to stable implementations, standardized languages or fuzzy relational database application development tools and methods. The main goal of this paper is the modelling and the implementation of a set of tools that allow usage of fuzzy logic enriched with priorities in relational database applications. In order to achieve that goal, at first, the relational data model is extended with the elements of fuzzy set theory. After that, a fuzzy extension of the SQL query language, called the PFSQL, is defined. An interpreter for that language is integrated inside an implementation of the fuzzy JDBC driver. An implementation of the CASE tool for modelling of fuzzy relational database schemas rounds up a set of tools for the implementation of Java fuzzy database applications. In this sense, this paper presents a step towards a methodology for the fuzzy relational database application development.

© 2012 Elsevier B.V. All rights reserved.

1. Introduction

Most of the real world information contains imprecise and incomplete values. The relational data model does not facilitate support for this kind of information. Attribute values are, by definition, absolute. Fuzzy sets and fuzzy logic have been successfully used in many similar applications, where management of imprecise and incomplete data was necessary. That is why the idea to incorporate fuzzy set theory and fuzzy logic to relational databases seems feasible and useful.

The main goal of the five-year long research conducted at the University of Novi Sad was the implementation of a system capable of using priority fuzzy logic with databases. Moreover, we have defined and implemented a complete fuzzy relational database application development solution. The system can be divided into four main parts:

- fuzzy query language PFSQL (Prioritized Fuzzy Structured Query Language),
- new fuzzy relational data model based on fuzzy extensions of the relational model,
- CASE (Computer Aided Software Engineering) tool for fuzzy relational database modelling and
- interpreter for the PFSQL query language weaved inside a fuzzy extension of the JDBC (Java Database Connectivity) driver that allows PFSQL querying from Java programs.

In an effort to create such a system, we encountered two main problems. The first one is a question of a method for storing fuzzy values using a relational database management system. The second, much more complex, is related to the design of the SQL language extensions that allow usage of prioritized fuzzy logic. Implementation of an appropriate interpreter that uses a fuzzy database is an integral part of the second problem. In this paper, we offer solutions to these problems and compare these solutions to the previous ones.

Giving solutions to these fundamental problems, we round them up by implementing the CASE tool that simplifies fuzzy database modelling and the fuzzy JDBC driver that allows easy querying. In this way, we obtain a system for the fuzzy relational database application development, to the best of our knowledge, the first of its kind.

The implementation of the priority fuzzy logic language interpreter is heavily dependent on the mathematical background. It is first necessary to formally explain how the actual calculations are done. The needed mathematical model is defined as a generalization of the known result – the Prioritized Fuzzy Constraint Satisfaction Problem. In this way, we obtain the concept of the Generalised Prioritized Fuzzy Constraint Satisfaction Problem (GPFCSP) that we describe in this paper as the needed mathematical background.

In the next section we give an overview of the existing results and solutions related to the use of fuzzy logic with databases. The third section contains the definition and other details related to the GPFCSP system. The next section gives the details related to the PFSQL language design, the fuzzy relational data model related to it and describes the main ideas in the PFSQL interpreter and fuzzy JDBC driver implementation. The fifth section describes how we applied the described system and gives some examples.
of its usage. At the end, we compare our approach to the most successful competitors – the FSQl (Fuzzy Structured Query Language) language and the FIRST-2 fuzzy relational data model, and draw some conclusions.

2. Related work

The first model that introduces similarity relations in the relational model is the Buckles–Petry model [1]. This paper gives a structure for representing inexact information in the form of a relational database. The structure differs from ordinary relational databases in two important aspects: components of tuples need not be single values and a similarity relation is required for each domain set of the database. Zvieli and Chen [2] offered a first approach to incorporate fuzzy logic in the ER (Entity–Relationship) model. Their model allows fuzzy attributes in entities and relationships.

Fuzzy functional dependencies and fuzzy normal forms, as well as algorithms for dependency preserving and lossless join decompositions of fuzzy relations in specific fuzzy extensions of the relational model are investigated in [3,4].

Umano and Fukami proposed the FREEDOM-O, a fuzzy database system which is an extension of the relational data model [5]. This system supports a fuzzy data model, and querying. It is the first implementation of a fuzzy database system. After that result, other researchers have proposed similar fuzzy extensions to the relational model such as in [6–9].

Another serious attempt to implement a fuzzy database system is given in [10,11]. The authors propose fuzzy extensions of the classical SQL and implement a system that allows using fuzzy conditions in the place of Boolean ones.

The GEFRED (Generalised Model of Fuzzy Relational Databases) model [12] is a probabilistic model that refers to generalised fuzzy domains and admits the possibility distribution in domains. This is a fuzzy relational database model that has the representational 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 in [13–15].

Chen and Kerre [16] introduced the fuzzy extension of several major EER (Extended Entity–Relationship) concepts. Fuzzy logic was applied to some of the basic EER concepts connected to the notion of subclass and super class. Chaudhry, Moine and Rundensteiner [17] 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 the FRDBs (Fuzzy Relational Databases) that 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 [18] describe a way to use the fuzzy EER model to model the database and represent the modeled fuzzy knowledge using a relational database in detail. This work gives insight into some new semantic aspects and extends the EER model with the fuzzy capabilities. The model is called the FuzzyEER model. Also, a way to translate the FuzzyEER model to the FIRST-2, a database schema that allows representation of fuzzy attributes in relational databases is given. The FIRST-2 schema introduces a concept of the Fuzzy Meta-knowledge Base (FMB). In addition, in this work, authors introduce and describe the specification and the implementation of the FSQL – an SQL language with fuzzy capabilities in great detail.

The current state of the art in this area includes mature fuzzy EER model extensions that describe a wide range of modelling concepts for full flavoured fuzzy database modelling. These conceptual models are supported by the robust models for the fuzzy data representation in relational databases, such as the FIRST-2. The possibilities to translate the conceptual models to the relational-based ones are also studied in detail. In addition, the FSQL is the first implementation of the fuzzy database query language that incorporates the majority of fuzzy logic concepts. Fuzzy logic has also been often applied to enrich various search and data mining algorithms [19,20].

In [21] the authors studied the possibilities to extend the relational model with the fuzzy logic capabilities. The subject was elaborated in [22,23], where a detailed model of fuzzy relational databases was given. One of the main features of the model is that it allows any fuzzy subset of the domain to be the attribute value which was not the case in previous FRDB models.

Moreover, using the concept of GPFCSP from [24,25], the authors found a way to introduce priority queries into fuzzy relational databases, which resulted in the PFSQL query language [26]. The PFSQL allows conditions in WHERE clause of a query to have a different priority i.e. importance degree. It is one of the first languages with such capabilities. The GPFCSP gives the theoretical background for the implementation of priority queries [26]. The authors have also succeeded in formalising the PFSQL queries by obtaining their interpretation in an existing fuzzy logic. They found that the Liti logic provides enough elements [27,28].

In difference to the earlier research attempts, the basic goal of our research is to develop an integrated set of tools that allows the fuzzy relational database application implementation. The main purpose of this paper is to round up all results obtained in this five-year long research and to describe the final solution as a whole.

3. Theoretical background

3.1. PFCSPP

A prioritized fuzzy constraint satisfaction problem (PFCSPP) is a type of the fuzzy constraint satisfaction problem (FCSP) in which the notion of priority is introduced. The PFCSPPs are first introduced by Dubois et al. [29]. An axiomatic framework was given in [24] and applied in the agent-based automated negotiation [30]. The most important factors in that implementation are the Schur-concave t-norms. They are introduced in such a way that the value with the biggest priority has the largest impact on the result given by a Schur-concave t-norm. For more information on the Schur-concave t-norms and t-norms in general see [31].

Takači [32] gives an alternative axiomatic framework for the PFCSPP, similar to the one given in [24], but with a stricter notion of priority. We recall that definition here.

Definition 3.1. When given $X,D,C,\rho$, where

1. $X = \{x_i | i = 1,2,\ldots,n\}$ is a set of variables,
2. $D = \{d_i | i = 1,2,\ldots,n\}$ is a finite set of domains. Each domain $d_i$ is a finite set containing the possible values for the corresponding variable $x_i$ in $X$,
3. $C$ is a set of fuzzy constraints, that is,
   $$C^f = \{R^f | \mu^f_i : d_{i1} \times \cdots \times d_{in} \to [0,1], \ i = 1,\ldots,m, 1 \leq k \leq n\}.$$ (1)
4. $\rho : C^f \to [0,\infty)$

and a simultaneous valuation $v_k(x_1,\ldots,x_n), x_i \in d_i$ of all variables in $X$, shortly denoted $v_k$ and an operator $*: [0,1]^n \to [0,1], g:[0,\infty) \times [0,1] \to [0,1], constraints R^f \in C^f$ for $i = 1,\ldots,n$ and a satisfaction degree $a^f_k(v_k)$ which is calculated in the following way:

$$a^f_k(v_k) = \oplus_{i=1}^n \left( g\left( \rho\left( R^f \right), \mu^f_i(v_k) \right) \right),$$ (2)

this system is a PFCSPP if the following axioms are satisfied:
1. If for the fuzzy constraint \( R'_{\text{max}} \), we have
\[
\rho_{\text{max}} = \rho\left( R'_{\text{max}} \right) \leq \max \{ \rho\left( R'_{i} \right) | R'_{i} \in C', \ i = 1, \ldots, n \},
\]
then
\[
\mu_{f_{\text{max}}} = 0 \Rightarrow \alpha_{f}(v_{k}) = 0.
\]

2. If \( \forall R' \in C', \rho(R') = \rho_{0} \neq 0 \), then
\[
\alpha_{f}(v_{k}) = \frac{\rho_{f}(v_{k})}{\rho_{\text{max}}}.
\]

3. For \( R'_{1}, R'_{2} \in C' \), assume \( \rho(R'_{1}) \geq \rho(R'_{2}) \), \( \delta > 0 \) and there are two different valuations \( v_{k} \) and \( v_{\delta} \) such that:
- if \( R'_{1} \neq R'_{2} \) and \( R'_{2} \neq R'_{1} \), then \( \mu_{p_{1}}(v_{k}) = \mu_{p_{2}}(v_{\delta}) \).
- if \( R'_{1} = R'_{2} \), then \( \mu_{p_{1}}(v_{k}) = \mu_{p_{2}}(v_{\delta}) + \delta \).
- if \( R'_{1} = R'_{2} \), then \( \mu_{p_{1}}(v_{k}) = \mu_{p_{2}}(v_{\delta}) + \delta \).
then \( \alpha_{f}(v_{k}) \geq \alpha_{f}(v_{\delta}) \) holds.

4. For two different valuations \( v_{k} \) and \( v_{\delta} \) such that \( \forall R' \in C' \), if
\[
\mu_{p_{1}}(v_{k}) \geq \mu_{p_{2}}(v_{\delta}),
\]
then \( \alpha_{f}(v_{k}) \geq \alpha_{f}(v_{\delta}) \) holds.

5. If there exists a compound label such that \( \forall R' \in C' \), \( \mu_{p_{1}}(v_{k}) = 1 \) then \( \alpha_{f}(v_{k}) = 1 \).

It is important to notice that the PFCSP is a generalization of the known concept of CSP (Constraint Satisfaction Problem). Constraints in the CSP are functions mapping the Cartesian product of the variable domains to the \( \{0,1\} \) set. In this case, we use the unit interval \( [0,1] \) instead of \( \{0,1\} \). In this way, the constraints are transformed to the membership functions of fuzzy sets.

The function \( \rho \) represents the priority of each constraint. Greater value of \( \rho(R') \) means that the constraint \( R' \) has a larger priority. The function \( g \) aggregates the priority of each constraint with the value of that constraint. These aggregated values of each constraint are then aggregated by the operator \( \oplus \), which results in the satisfaction degree of that valuation.

The first axiom states that a zero value of the local satisfaction degree of the constraint with the maximum priority implies a zero value of the local satisfaction degree. The second axiom states that, in the case of equal priorities, the PFCSP becomes a F CSP. The third axiom captures the notion of the priority. If one constraint has a larger priority then the increase of the value on that constraint should result in a bigger increase of the global satisfaction degree than when the value with the smaller priority has the same increase. The fourth axiom is the monotonicity property, and finally the fifth is the upper boundary condition. A more detailed explanation of the axioms given in this definition can be found in [32]. The next theorem gives actual PFCSP that satisfies this definition.

**Theorem 3.1.** The following system is a PFCSP:
\[
\alpha_{f}(v_{k}) = \oplus_{i=1}^{n} \left( \diamond_{\mu} \left( \frac{\rho(R'_{i})}{\rho_{\text{max}}} \right) \mu_{p_{i}}(v_{k}) \right),
\]
where \( R'_{i} \in C' \) for \( i = 1, \ldots, n \), \( \oplus \) is the Lukasiewicz t-norm given by:
\[
T_{l}(x_{1}, \ldots, x_{n}) = \max(0, \sum_{i=1}^{n} x_{i} - n + 1)
\]
and \( \diamond_{\mu}(x,y) = S_{l}(1 - x, y) \), where \( S_{l} \) is the product norm given by:
\[
S_{l}(x,y) = x + y - xy.
\]
The proof is given in [32].

As mentioned above, the PFCSP is a concept known for years. At this point we introduce the idea of similarity between the FCSP systems and the SQL SELECT clause. The basic structure of the SELECT clause consists of SELECT, FROM and WHERE constructs. The variables that follow after the SELECT keyword can be viewed as FCSP variables. The table names, that follow after the FROM keyword, can be viewed as their domains. The WHERE clause contains a sequence of constraints connected with logical operators.

Similarly, the FCSP also contains a set of constraints. The membership degree of each constraint indicates the local degree to which the constraint is satisfied. In order to obtain the global satisfaction degree, local degrees are aggregated using a certain t-norm.

In difference to the standard SQL WHERE clause, constraints in the FCSP are and they even have a priority assigned. It is becoming clear that it is possible to extend the SELECT clause by allowing fuzzy variables and fuzzy constraints with priorities in the WHERE clause. The FCSP could be used as a mathematical background that defines how to calculate the fuzzy membership degrees of the result set tuples. But it has one limitation – it only allows the usage of t-norm, i.e. conjunction in connecting constraints. The fuzzy where clause in which it is allowed only to use a conjunction, and not a negation or a disjunction is not acceptable. That is why the concept of the FCSP had to be generalised first.

### 3.2. GPFCSP

The FCSP concept is generalised by allowing the possibilities to combine the t-norm, t-conorm and negation operators to connect the prioritized fuzzy constraints. The definition is very similar to the definition of the FCSP. We extend the previous definition by adding the possibility to use a disjunction or a negation. In the process, we conserve the desired behaviour of priority.

**Definition 3.2.** Let \( X, D, C', \rho, g, \oplus \) be defined as in Definition 3.1. The generalised FCSP is defined as a tuple \( \langle X, D, C', \rho, g, \land, \lor, \neg \rangle \) which satisfies the following.

An elementary formula in the GPFCSP is a pair \( \langle x, \rho_{i}(C) \rangle \), where \( C_{i} \in C', x \in \text{Dom}(C_{i}) \) represents the satisfaction degree of a constraint \( C_{i} \) and \( p_{i} = \rho_{i}(C_{i}) \) represents its priority.

A formula in the GPFCSP is defined in the following way:

1. An elementary formula is a formula.
2. If \( f_{1} \) and \( f_{2} \) are formulas then also \( \land(f_{1}, f_{2}), \lor(f_{1}, f_{2}) \) and \( \neg(f_{1}) \) are formulas.

For each valuation \( v_{k} \), a satisfaction degree \( \alpha_{f}(v_{k}) \) of a formula \( F \) is calculated depending on the interpretation of connectives.

A system is a GPFCSP if:

1. Let \( F = \langle \land(1, \ldots, n) \rangle \) be a formula in the GPFCSP where \( f_{i} \) \( i \in \{1, \ldots, n\} \) are elementary formulas and let \( C' \) be a set of constraints that appear in the formula. If for the fuzzy constraint \( R'_{\text{max}} \) we have
\[
\rho_{\text{max}} = \rho\left( R'_{\text{max}} \right) = \max(\rho(R') | R' \in C'),
\]
then for each formula \( F \) we have:
\[
\mu_{f_{\text{max}}} = 0 \Rightarrow \alpha_{f}(v_{k}) = 0.
\]

2. If \( \forall R' \in C', \rho(R') = \rho_{0} \in [0,1] \), then for each formula \( F \) the following holds:
\[
\alpha_{f}(v_{k}) = F_{\text{C}}(v_{k}),
\]
where \( F_{\text{C}} \) is the interpretation of the logical formula \( F \) in fuzzy logic \( L_{\land, \lor, \neg, \land} \).

3. For \( R'_{1}, R'_{2} \in C' \), assume \( \rho(R'_{1}) \geq \rho(R'_{2}), \delta > 0 \) and assume that there are two different valuations \( v_{k} \) and \( v_{\delta} \) such that:
- if \( R'_{1} \neq R'_{1} \) and \( R'_{1} \neq R'_{2} \), then \( \mu_{p_{1}}(v_{k}) = \mu_{p_{2}}(v_{\delta}) \).
Theorem 3.2. The following system \( (X, D, C^f, \rho, x, \wedge, \vee, \neg, \circ) \), where \( x = T_x \), \( \circ = S_x \), \( x = 1 - x \) and finally \( \circ(x_i, c_i) = S_p(x_i, 1 - p_i) \) is a GFPCSP, where \( f_i, i \in \{1, \ldots, n\} \) are elementary formulas then
\[
\alpha_x(v_x) = 1.
\]
The fact that the \( T_x - S_p \) combination can be used to obtain a GFPCSP as it was used to obtain a PFCS system earlier, will be expressed as another very important theorem.

Let us also define the following constraints:
1. \( R_1 \) = "excellent high school GPA",
2. \( R_2 \) = "about 20 years old",
3. \( R_3 \) = "good test results".

The constraints are fuzzy subsets of the corresponding domains (Fig. 1). Let us model the first constraint as a right-shoulder that ascends from 4 to 5. The second constraint is modeled as a triangle fuzzy number that has a peak in 20, with the left offset of 1 and the right offset of 6. Finally, the third constraint is a right-shoulder that ascends from 40 to 60.

Let the criterion on how to pick a candidate (formula \( F \)) be: “student has to have an excellent high school GPA \( (R_1) \) or has to be about 20 years of age \( (R_2) \) and has to have a good test results \( (R_3) \).” In addition, we define the priorities like this: \( \rho(R_1) = 0.4 \), \( \rho(R_2) = 0.1 \) and \( \rho(R_3) = 0.6 \). The valuation \( v_x \) given in the Table 1 defines three candidates.

First, we calculate the constraint satisfaction degree for every constraint and every student. These degrees are obtained directly as values of the corresponding membership functions in the given points. The results are given in the Table 2.

Now we can calculate the global constraint satisfaction degree for every student in the following way:
\[
\alpha = T_x \left( S_p \left( \mu_{R_1}(v), 1 - \rho(R_1) \right), S_p \left( \mu_{R_2}(v), 1 - \rho(R_2) \right) \right).
\]
If we use the values for the first student, we obtain the following:
\[
\alpha = T_x \left( S_p(0.8, 0.983), S_p(0.5, 0.9), S_p(0.5, 0.4) \right).
\]
Finally, we obtain the satisfaction degree for the first student:
\[
\alpha = T_x (0.8, 0.983, 0.7) = T_x (1, 0.7) = 0.7.
\]
The global constraint satisfaction degrees for the other students are calculated in the same way and are given in the Table 3. These results are a measure of how much the students satisfy our criteria. Obviously, Sarah is the best candidate.

Clearly, the GFPCSP concept can be used as a mathematical background that defines how to calculate the fuzzy membership degrees of the result set tuples because, like the standard SQL, it allows the usage of t-norms (conjunction), t-conorms (disjunction) and negation. It should be clear by now that the given example can easily be connected to the query from the Listing 1 given in some pseudocode priority fuzzy SQL language.

4. The fuzzy relational database application development system

In this section we first discuss details related to the PFSQL language design. We introduce it formally and illustrate its possibilities on a set of examples. Next, we describe our fuzzy relational

---

1. \( X_1 \) represents the high school GPA, \( d_1 = [2, 5] \).
2. \( X_2 \) represents the age, \( d_2 = [0, 130] \).
3. \( X_3 \) represents the test score, \( d_3 = [0, 60] \).
The constraint satisfaction degrees for every constraint and every student.

<table>
<thead>
<tr>
<th>Name</th>
<th>GPA</th>
<th>Age</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>4.5</td>
<td>21</td>
<td>50</td>
</tr>
<tr>
<td>Sarah</td>
<td>4.8</td>
<td>19</td>
<td>55</td>
</tr>
<tr>
<td>Vanessa</td>
<td>4.7</td>
<td>20</td>
<td>52</td>
</tr>
</tbody>
</table>

Table 2

The constraint satisfaction degrees for all students.

<table>
<thead>
<tr>
<th>Name</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>0.7</td>
</tr>
<tr>
<td>Sarah</td>
<td>0.85</td>
</tr>
<tr>
<td>Vanessa</td>
<td>0.76</td>
</tr>
</tbody>
</table>

The nested queries are yet another problem that we encountered in our effort to extend the SQL with the fuzzy capabilities. We can divide the nested queries in two categories – the ones that do not depend on the rest of the query and the ones that do. The independent SQL queries are not problematic, they can be calculated separately, and the resulting values can be used in the remainder of the query as constants. The dependent SQL queries with the dependence expressions that do not use the 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 the fuzzy values or operators, then it remains unclear how to evaluate such a query and what does this dependence even mean.

Having stated the facts and considerations we had in mind, we give the complete EBNF syntax of the PFSQL language in the Appendix A.

Although it is not possible to present all the features of the language just by giving its EBNF syntax, this grammar gives an overview of the possibilities and extensions built into the PFSQL. To illustrate the possibilities further, we give four PFSQL query examples. The queries are executed against the database containing the student entrance examination data that was mentioned in the Section 3, and described in detail later in this section. The first query (Listing 2) returns the names and surnames of the students whose high school GPA is greater than the given triangular fuzzy number.

The # symbol is chosen to mark the fuzzy constants. If we defined a linguistic label “average GPA” that has a value triangle(4.1,0.4), the previous query could be simplified (Listing 3).

The queries can be enriched with the additional constraints. The next query (Listing 4) returns the names and surnames of the students that have the GPA greater than average with the priority 0.4, and are 20 years old with the priority 0.1. The query also contains the threshold clause that limits the results and removes the tuples with the fuzzy satisfaction degree smaller than 0.2.

As we already mentioned, the aggregate functions MAX, MIN and COUNT can take a fuzzy value as an argument. The query at the Listing 5 illustrates the usage of the aggregate function MIN. It returns the minimal GPA.

If we assume that the variable s.highSchoolGPA 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: triangle(4.2,0.7,0.4).

4.2. The fuzzy relational data model

It is clear now that the PFSQL implementation has to rely upon a meta data about the fuzzy attributes that reside inside the database. For these purposes, a fuzzy relational data model has been defined. Our model stores the crisp values in the same way as the relational data model. At the end, we explain details related to the CASE tool and the implementation of the PFSQL interpreter integrated inside the fuzzy JDBC driver.

4.1. The PFSQL language

In order to allow the use of fuzzy values in SQL queries, we extended the standard SQL with several new elements. In addition to fuzzy capabilities, we add the possibility to specify the priorities for fuzzy statements. We named the query language constructed in this manner the priority fuzzy SQL – PFSQL. This appears to be the first language that has such capabilities.

The question is what elements of the standard SQL should and can be extended. It is necessary to allow the comparison between different types of fuzzy values. In addition, variables can have both crisp and fuzzy values, so comparison between fuzzy and crisp values has to be included. In other words, the PFSQL has to be able to calculate expressions like $c_1 = \text{triangle}(a, b, c)$, regardless of what value of height is in the database – fuzzy or crisp. The expression $\text{triangle}(a, b, c)$ denotes a triangular fuzzy number with the peak at $a$, with the left offset $b$, and the right offset $c$. Next, we demand the possibility to set the conditions like $\text{height} \geq \text{triangle}(180, 11, 9)$. The ordering and addition operations on the set of fuzzy numbers give grounds to introduce the aggregate functions like MIN, MAX and SUM in the PFSQL. Moreover, it is possible to define a fuzzy GROUP BY clause in combination with the aggregate functions on fuzzy values.

The standard SQL includes possibilities to combine the conditions using logical operators. This possibility also has to be a part of the fuzzy extensions, thus combining the fuzzy conditions is also a feature of our implementation. The values are calculated using the $t$-norms, $t$-conorms, and the strict negation. The queries are handled using priority fuzzy logic which is based on the GPCSP systems.

SELECT s.firstName, s.lastName
FROM Student s
WHERE (s.highSchoolGPA > triangle(4.1,0.4))

Listing 2. An example of a PFSQL query that returns names and surnames of students whose GPA is greater than the given triangular fuzzy number.

SELECT s.firstName, s.lastName
FROM Student s
WHERE (s.highSchoolGPA > ling(averageGPA))

Listing 3. An example of a PFSQL query with a linguistic label.

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

The standard SQL includes possibilities to combine the conditions using logical operators. This possibility also has to be a part of the fuzzy extensions, thus combining the fuzzy conditions is also a feature of our implementation. The values are calculated using the $t$-norms, $t$-conorms, and the strict negation. The queries are handled using priority fuzzy logic which is based on the GPCSP systems.
model does, while, for fuzzy values, we define the fuzzy meta data model. In addition, here we provide an insight into the process of transformation of an example of the classical relational model with the fuzzy attributes to the corresponding fuzzy relational data model.

If we wish to store a fuzzy value, we need to find a way to store the data related to its characteristic function. Theoretically, in this way, we could store any fuzzy value. But, in practice, only a handful of characteristic functions are in use. Let us name them the fuzzy data types from this aspect. That is why we cover only a limited number of fuzzy data types and obtain an efficient and a relatively simple data model.

An example relational model shown in Fig. 2 contains the tables Student and StudyProgram as well as the intersection table Score that we use to model this many-to-many relationship. The table Student contains data about the students taking an entrance examination. It contains the two fuzzy attributes: the highSchoolGPA and the age. The table StudyProgram contains data related to the study programs and contains no fuzzy values. The intersection table – Score, contains data about the score of a particular student on a particular study program in a particular examination term (we suppose that there can be more than one entrance examination term). Its attribute score is also fuzzy.

The corresponding fuzzy relational data model is shown in Fig. 3. The tables Student, StudyProgram and Score are shown at the top of the figure. They are the same as they were before except for the data type of fuzzy columns. In this schema, they are of type INTEGER. Moreover, they became the foreign keys that originate from the attribute valueID belonging to the FuzzyValue table. In order to represent the fuzzy values in the database, we extend this model with some additional tables that make the fuzzy meta data model.

The table IsFuzzy simply stores the information whether an attribute is fuzzy or not. All attribute names in the database are stored here, and beside the table and the attribute name (attributes tableName and attributeName), the information whether the attribute is fuzzy (value of the attribute isFuzzy is TRUE) or not (value of the attribute isFuzzy is FALSE) is present.

The table FuzzyValue represents a connection between the fuzzy data model and the fuzzy data meta model. Every fuzzy column in every table is a foreign key that references the attribute valueID – the primary key of the table FuzzyValue. Thus, we have one record in the table FuzzyValue for every record with the fuzzy value in the database. The attribute code is a foreign key from the table FuzzyType. This table stores the name of every possible type of fuzzy value allowed in the model. These types are as follows: interval – the fuzzy value is an interval, triangle – the fuzzy value is a triangular fuzzy number, trapezoid – the fuzzy value is a trapezoidal fuzzy number, fuzzyShoulder – the fuzzy value is a fuzzy shoulder, linguisticLabel – the fuzzy value is a linguistic label, crisp – the fuzzy value is actually a crisp value.

For every value in this list, there is a separate table in the meta model that stores the data for all fuzzy values of the specific fuzzy type. All of these tables have the attribute valueID, a foreign key from the table FuzzyValue. In this way, the value for the specific fuzzy attribute is stored in one of these tables depending on its type.

The attribute forValueID in the table FuzzyValue is a foreign key that represents a recursive relationship and references the primary key of the FuzzyValue table. This attribute is used to represent the linguistic labels. It has a value different than null if the type of the attribute that it represents is a linguisticLabel. The linguistic labels only represent names for the previously defined fuzzy values. In this fashion, if an attribute is a linguistic label, then its name is stored in the table LinguisticLabel. In this case, the attribute forValueID has the value of valueID of a fuzzy value that this linguistic label represents. We conclude that, in order to represent a linguistic label, two records in the table FuzzyValue are needed.

For example, let us suppose that the student Jovan Jankovic has a high school GPA of “approximately 4.85”. In accordance to this, in the Student table there is a corresponding record containing some basic information and values for the student’s GPA – highSchoolGPA = 1. This value is a foreign key referring to the valueID attribute of the FuzzyValue table from the fuzzy meta model that contains actual data. For example, in case of the highSchoolGPA attribute, the FuzzyValue table contains a record with valueID = 1 and with the code attribute value of 2 that signifies a triangular fuzzy number. The table containing the fuzzy type codes is the FuzzyType table. Finally, the table TriangularFN contains the actual value for the student’s high school GPA – a triangular fuzzy number with the peak in 4.85 and with the left offset of 0.65 and the right offset of 0.07.

The rest of the values, for the other fuzzy types, are stored in the database in a similar way. Additional information and details related to this topic can be found in [23,26].

The presented fuzzy meta model has been put through the normalization algorithm that guarantees that the resulting model conforms to the 3rd normal form. Of course, the fulfillment of the theoretical conditions for the 3rd normal form depends on the ground database model that we are creating too. In any case, this feature guarantees that if a database model is at least in the 3rd normal form, then the addition of the presented fuzzy meta model will result in a complete model at least in the 3rd normal form. In this way, the presented fuzzy meta model significantly improves...
its theoretical and practical performance. The main reason for insisting on the 3rd normal form in this model is the efficiency of the complete software system relying on a fuzzy relational database that conforms to our model.

4.3. The CASE tool

The requirements set in the process of modelling of the CASE tool include the functions for simplified building of a fuzzy relational data model, as well as the functions for its transformation to the SQL script. Our intention was to implement a CASE tool capable for the visual modelling and easy administration of all components of a fuzzy relational data model – tables, attributes, fuzzy data types and relationships. The application is implemented using Java programming language and the Swing platform for the GUI.

The CASE tool’s GUI works in the similar way as in all modern tools of this type that allow the modelling of the classical relational models. In the model building process all the automation related to the migration of keys through the relationships is included. This feature includes the cascade deletion process, migration of keys during the relationship creation process and the circular reference detection and prevention. In addition, the CASE tool is required to allow easy SQL script generation for the specific database management system. In this sense, the capabilities to specify the data types used by the DBMS and rules for mapping of the types used in the model (together with the fuzzy data types) to these types had to be included.

The details on how this CASE tool is implemented may be found in [23].

4.4. The PFSQL interpreter and the fuzzy JDBC driver

The most difficult problem in building of the system that we are describing is related to the implementation of an interpreter for the presented PFSQL query language. If we consider a classical SQL query, it is clear how to assign a truth value to every elementary condition in the WHERE clause. 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 implement a set of algorithms that calculate the 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 an algorithm that calculates the compatibility degree 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 the operator AND, and its dual \( t \)-conorm in case of the operator OR. For the negation, we use the strict negation: \( N(x) = 1 - x \). In case of the priority statements, we use the GPFCSP system rules to calculate the result.

We will now describe the processes that allow the PFSQL queries to be executed. The basic idea is to first transform a PFSQL query into an SQL query. Namely, conditions with fuzzy attributes are removed from the WHERE clause and moved up in the SELECT clause. In this way, the 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 obtained results are interpreted using the 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 the satisfaction degree below the threshold are removed.

The implementation of these processes contains many subtle points and solutions to technical problems. They are all listed and described in detail in [26].

The need to ease the PFSQL usage from Java programs and still keep the database independence is resolved with the implementation of the fuzzy JDBC driver. This driver acts as a wrapper for the PFSQL processing mechanisms and for the JDBC API implemented by the driver for a specific RDBMS. The JDBC driver for the database used simply becomes a parameter that the fuzzy JDBC driver uses.

**Fig. 3.** An example of a fuzzy relational data model.
5. Testing

The described implementation is tested on the fuzzified database subschema of the student affairs information system of the Faculty of Science in Novi Sad. This subschema is related to the faculty entrance examination management. It consists of twenty relational tables with more than 5000 records in most of them. We add to that nine tables that make the fuzzy meta data model (Fig. 3), explained in detail in the Section 4.2.

First we describe how the sample database was made. We copied the entrance examination subschema with real data from the Faculty’s information system to the test database. For the purpose of testing, we used the MaxDB database management system. After that, we selected a number of table attributes suitable for fuzzification. We added the fuzzy meta model to our database, and connected all the selected attributes to the fuzzy meta model using approach described in the Section 4.2. Finally, we implemented a small ad hoc application that fuzzifies data related to all selected attributes by randomly putting some fuzzy values instead of the real values. The inserted values are random, but they are selected carefully for every attribute so that they are meaningful. For instance, the high school GPA takes values from 2 to 5 in the real database. The inserted values also respect this constraint. An example of the inserted fuzzy value would be the fuzzy interval [3.6, 4.2], or the triangular fuzzy number with the peak in 4.2, with the left offset of 0.8 and with the right offset of 0.35.

5.1. The example database subschema

The database subschema is shown in Fig. 5. This figure shows only the main thirteen tables of the model. For the sake of simplicity, we omitted the utility tables, as well as the tables that belong to the fuzzy meta model.

The central table in this subschema is the Student table that contains the most important data related to an entrance examination candidate. This data is appended with the detailed information contained in the Application table. The utility table Language contains the data on foreign languages that a candidate learned as part of his previous education.

The table Application profile contains information related to the study profiles that candidates may choose. These profiles do not have to be equal with those they will choose after the successful entrance examination. For instance, a candidate may choose to apply to the Mathematics study profile, but after the examination, he may choose to enroll to the Mathematics teacher or the Financial mathematics study profile. We use the tables Profile and Contains-Profile to model these facts. The table Priority contains information on all application profiles a candidate has applied to, sorted by their priority.

Every application profile is related to its quota – the maximum number of students that can be enrolled to that profile. A quota may be related to more than one application profile. For every application profile, we define a group of subjects included in the entrance examination for that profile (table Group). The tables Subject and BelongsToGroup contain information about subjects in every specific group.

The candidate’s score is contained in the table Score. It contains scores for every group of subjects, that constitute an entrance examination for the specific application profile a candidate has chosen. RankList stores information on candidate rank lists for every application profile.

5.2. Fuzzification

Fuzzification of described model includes adding the fuzzy meta model, selection of the attributes that will be fuzzified and their connection to the fuzzy meta model. In essence, the final example database contains tables shown in Fig. 5, the utility tables from the model that are removed from the figure for the sake of simplicity, and the fuzzy meta model. The attributes that we selected for fuzzification are the following:

- GPA1, GPA2, GPA3 and GPA4 from the Application table, representing the candidate’s GPA’s for all four years of high school,
- highSchoolGPA from the Student table, representing the candidate’s overall GPA,
- score from the Score table, representing the candidate’s entrance examination score for a given group of subjects,
- score, entranceExamScore and GPA Score from the Ranklist table, representing the candidate’s overall score, the entrance examination score and the score related to the high school GPA,
- budget and selfFinance from the Quota table, representing available space for a give application profile and
- budgetThreshold, selfFinanceThreshold and entranceExamThreshold, from the ApplicationProfile table, representing the minimum score for budget and self finance studying and the minimum score needed to pass the entrance examination.

As stated in the Section 4.2, these attributes are transformed to the foreign keys from the valueID attribute of the FuzzyValue table, that relate them to the fuzzy meta model. Therefore, their actual (possibly fuzzy) values are contained and described in the fuzzy meta model. The Data from the real faculty’s information system database is removed and new random fuzzy values are generated for testing purposes. For example, the Table 4 contains a few records from the Student table. We do not show the citizenID column because we do not want to expose any private information. This column contains 13 digit values.

The values for the column highSchoolGPA are not actual GPA values, they are the foreign keys referencing the valueID attribute from the FuzzyValue table. These values connect our database subschema to the fuzzy meta model. For instance, the fuzzy meta model contains the trapezoidal fuzzy number with the left maximum in 4.55, the right maximum in 4.82 and with the left offset of 1.02 and the right offset of 0.07, as the value for the second candidate from the example table, Stevan Vasiljević.

5.3. Test query results

In this section we use the described example fuzzy database to set three queries used as the examples for the PFSQL syntax in the Section 4.1. We add to that another, more complex query example.
For every query we give an illustrative fragment of the obtained result set.

The first query (Listing 6) lists the first and last names of students whose GPA is greater than the given triangular fuzzy number. The Table 5 shows only a small fragment of the result set. Clearly, the result set contains an additional column, the membership degree, that takes values from the unit interval and represents a fuzzy measure of how much does a tuple belongs to the result set. The same observation holds for the following two examples.
The previous query can be enriched with additional constraints. The query from the Listing 7 returns names and surnames of students that have the GPA greater than average with the priority of 0.4, and are born before 1991, with the priority of 0.1. The query also contains the threshold clause that limits the results and removes the tuples with the fuzzy satisfaction degree smaller than 0.2. This query is more complex than the corresponding query from the Section 4.1, because the attribute that represents date of birth does not belong to the Student table. It belongs to the Application table, and therefore, it was necessary to join these two tables.

The Table 6 shows a fragment of the result set that contains the same tuples shown in the Table 5. In this way, we can observe the changes made to the membership degree caused by the additional constraints.

The Listing 8 shows the most complex query chosen for the testing purposes. It returns the first and last names of the students that:

- have the high school GPA greater than the given triangular fuzzy number with the priority of 0.9,
- whose first name is Milan, with the default priority of 1 and
- that have the entrance examination score equals to the given interval with the priority of 0.6.

Additionally, the threshold of 0.6 for the membership degrees is specified.

The Table 7 lists a fragment of the result set that the system returned.

The last example illustrates the use of the aggregate functions with fuzzy arguments. The well known aggregate functions MAX, MIN and COUNT can take fuzzy value as an argument, as defined in the PFSQL syntax. The query at the Listing 9 returns the minimal high school GPA for all candidates. The high school GPA contains fuzzy values, so the result that the system returned was also fuzzy – a fuzzy interval:

\[
\#[2.109256895095825, 3.480854190734863]\#
\]

5.4. Performance

Detailed performance analysis of the presented system is out of the scope of this paper. Nevertheless, we present some basic observations related to the query execution performance that can serve to get an impression of overall system behaviour.

The test database presented in this section contains information on approximately 5000 entrance examination candidates. So, most of the tables presented in Fig. 5 contain approximately 5000 records. As a consequence, the central table in the fuzzy meta model, FuzzyValue, contains exactly 59608 records. For the testing purposes, we used the MaxDB database management system, version 7.6. The testing machine was a PC containing the Intel Core i7 processor with the 12 GB of RAM running the 64 bit Ubuntu server operating system.

Using described environment, the first three presented queries (Listings 6–8) had an execution time below 0.5 s, which resulted in an instantaneous answer. However, the fourth query (Listing 9) includes the complex calculations related to comparison of approximately 5000 fuzzy values of different types. All these calculations are done outside the database, and here we observed a drop in performance. The time needed to process this query was 9.78 s.

6. Comparison to the other approaches

If we compare our fuzzy relational data model to the most advanced fuzzy relational data model available today – the FIRST2 [18], we conclude that there are several similarities between

<table>
<thead>
<tr>
<th>Table 4</th>
<th>Example values from the Student table.</th>
</tr>
</thead>
<tbody>
<tr>
<td>appNumber</td>
<td>firstName</td>
</tr>
<tr>
<td>00746</td>
<td>Marko</td>
</tr>
<tr>
<td>00747</td>
<td>Stevan</td>
</tr>
<tr>
<td>00748</td>
<td>Milica</td>
</tr>
<tr>
<td>00749</td>
<td>Jelena</td>
</tr>
<tr>
<td>00751</td>
<td>Danijela</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5</th>
<th>The results of the first example query.</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstName</td>
<td>lastName</td>
</tr>
<tr>
<td>Jovan</td>
<td>Janković</td>
</tr>
<tr>
<td>Jelena</td>
<td>Stanojević</td>
</tr>
<tr>
<td>Petar</td>
<td>Kristanović</td>
</tr>
<tr>
<td>Milica</td>
<td>Marinković</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 6</th>
<th>The results of the second example query.</th>
</tr>
</thead>
<tbody>
<tr>
<td>First name</td>
<td>Last name</td>
</tr>
<tr>
<td>Jovan</td>
<td>Janković</td>
</tr>
<tr>
<td>Jelena</td>
<td>Stanojević</td>
</tr>
<tr>
<td>Petar</td>
<td>Kristanović</td>
</tr>
<tr>
<td>Milica</td>
<td>Marinković</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing 6</th>
<th>An example of a PFSQL query.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT s.firstName, s.lastName FROM Student s WHERE s.highSchoolGPA&gt;’tri(4,1,0.4)#’</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing 7</th>
<th>An example of a PFSQL query.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT s.firstName, s.lastName FROM Student s, Application a WHERE s.highSchoolGPA&gt;’tri(4,1,0.4)#’ PRIORITY 0.4 AND a.birthDate&lt;’1991-01-01’ PRIORITY 0.1 AND s.applicationNumber=a.applicationNumber AND s.termCode=a.termCode AND THRESHOLD 0.2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Listing 8</th>
<th>An example of a PFSQL query.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT s.firstName, s.lastName FROM Student s, Score sc WHERE NOT s.highSchoolGPA&gt;’tri(4,1,0.4)#’ PRIORITY 0.9 OR s.firstName=’Milan’ AND s.applicationNumber=sc.applicationNumber AND s.termCode=sc.termCode AND sc.score&gt;’i(45,55)#’ PRIORITY 0.6 AND THRESHOLD 0.6</td>
<td></td>
</tr>
</tbody>
</table>
them. Although the methods for the 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 most widely used fuzzy concepts, and stores values as effectively as possible without too much overhead.

The fuzzy attributes of the type 1 in the FIRST-2 model are crisp values that our model also supports. The fuzzy types that our model covers are a subset of those represented by the fuzzy attributes type 2 and 3. Null values, intervals and trapezoidal fuzzy numbers in the FIRST-2 are represented by the structures that have these same names. A subset of the set of triangular fuzzy numbers, isosceles triangle, is represented by the approximate value with the explicit margin in the FIRST-2 model. All other types of triangular fuzzy numbers, as well as fuzzy quantities can be represented by the possibility distributions with 2 and with 4 values in the FIRST-2, although these distribution types are more general.

Moreover, the FIRST-2 model describes a wider range of other possibilities for the fuzzy values and combines atomic values according to their respective structure. In this paper we described the advanced version of our model that treats the fuzzy values similarly. Although, functionally, our model is a subset of the FIRST-2, it gives the theoretical contribution in modelling from the aspect of the relational model theory because it conforms to the 3rd normal form. The basic disadvantage of the FIRST-2 model is non-conformance even to the 1st normal form.

The fuzzy database query language FSQL is built on top of the FIRST-2 model using the Oracle DBMS and the PL/SQL stored procedures [18]. Similarly, we used the fuzzy relational data model described in this paper to build an interpreter for the PFSQL language. We have developed the PFSQL query language from ground up, extending the features of the SQL into the fuzzy domain. The PFSQL language is an extension of the SQL language that allows fuzzy logic concepts to be used in queries. Among other features described in [26,23] in detail, this query language allows the priority statements to be specified for the query conditions. For calculating the membership degree of the query tuples when the priority is assigned to the conditions, we use the GPFCSP systems mentioned in the introduction.

Although the FSQL language has more features than the PFSQL, it does not allow the 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. On the other hand, in this way, most of the calculations are done outside the database which yields a drop in performance comparing to the systems that use the database and its stored procedures to do the calculations.

### Table 7

<table>
<thead>
<tr>
<th>First name</th>
<th>Last name</th>
<th>Membership degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Milan</td>
<td>Amadžić</td>
<td>1.0</td>
</tr>
<tr>
<td>Milan</td>
<td>Brkić</td>
<td>0.9156248444431065</td>
</tr>
<tr>
<td>Milan</td>
<td>Korač</td>
<td>0.93025398896355</td>
</tr>
<tr>
<td>Milan</td>
<td>Rosić</td>
<td>0.9469450772437487</td>
</tr>
<tr>
<td>Milan</td>
<td>Vujinović</td>
<td>0.7428571671835702</td>
</tr>
</tbody>
</table>

### Listing 9. An example of a PFSQL query.

```sql
SELECT MIN(s.highSchool GPA) FROM Student s
```

### Listing 10. The EBNF syntax of the PFSQL language.

---

Following the stated ideas, we implemented the CASE tool described here in order to ease the fuzzy relational database schema development and its usage in the real world applications with the PFSQL using fuzzy JDBC from Java programs. To the best of our knowledge, this is the first attempt to introduce a solution, an integrated set of tools, that allows the fuzzy relational database application development. Indeed, the system described in this paper is the only one publicly available open source system that serves this purpose. A working version of our model, the CASE tool and the PFSQL interpreter are available for download from the following web site http://www.is.pmf.uns.ac.rs/fuzzydb.

7. Conclusion

Fuzzy logic and fuzzy set theory have been proven to be the natural approach to model uncertainty in the relational databases. The most important result of this paper is a set of tools that allows usage of the priority fuzzy logic mechanisms in the database application development.

Detailed analysis of possibilities and related attempts to extend the SQL language using fuzzy set theory concepts has lead to the PFSQL language. This language is the first to allow the queries with the arbitrary logical expression containing the fuzzy set theory constructs and the priorities. The mathematical model needed to calculate the results of such queries has been formulated in the form of the GPF CSP system that represents a generalisation of the already known construct – the PF CSP.

A new model for the fuzzy information storage has been defined as an extension of the relational model using elements of fuzzy set theory. This data model represents the basis for the PFSQL query language interpreter. The CASE tool that enables easy database design based on the introduced model has been constructed. The idea to encapsulate the process of the query execution inside a conservative fuzzy extension of the JDBC driver is also new. This result allows the database independence. The presented set of tools supports the idea to specify the complete methodology for fuzzy relational database applications development.

The 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 the entrance examination management. It consists of about 30 relational tables with more than 5000 records in most of them.

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 the 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. The authors intend to study and solve these problems in the future.

Acknowledgement

The work is partially supported by Ministry of Education and Science of the Republic of Serbia, through Project OI174023: “Intelligent techniques and their integration into wide-spectrum decision support” and Project OI174009: “Mathematical models of non-linearity, uncertainty and decision making”.

Appendix A. The PFSQL language EBNF grammar

Listing 10.

References


