Abstract: In this paper, we present features of the PFSQL, as well as the implementation of the fuzzy JDBC driver that contains the interpreter for this language. The PFSQL is an extension of the SQL with some elements of fuzzy logic and fuzzy set theory. Directions on how the interpreter for this language is implemented are given in the continuation. Simple access to the PFSQL interpreter is achieved by introducing a fuzzy JDBC driver. In the conclusion, we give a brief comparison of the language with the FSQL – one of the most advanced fuzzy SQL languages today.

Key Words: Fuzzy relational database; Priority fuzzy SQL; Fuzzy JDBC driver

1 INTRODUCTION

In this section we will shortly describe the notion of FRDB 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 fuzzy relational databases (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.

One of the early works, the Buckles-Petry model [1], 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. 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. It defines three levels of fuzziness. At the first level, entity sets, relationships and attribute sets may be fuzzy, i.e. they have a membership degree to the model. The second level is related to the fuzzy occurrences of entities and relationships, and on notion which instances belong to the entity or relationship with different membership degrees. Finally, the third level is concerned with the fuzzy values of attributes of special entities and relationships.

The GEFRED (Generalized Model of Fuzzy Relational Databases) model [3] 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. Also, it contains the notion of unknown, undefined and null values. The GEFRED model experienced subsequent expansions, such as [5] and [6].

Chen and Kerre [6, 7] 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 superclass. Chaudhry, Moyne and Rundensteiner [8] 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 (Fuzzy Relational Databases), 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 [9] describe a way to use the 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 EER model with fuzzy capabilities. The 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. The FIRST-2 schema introduces a concept of Fuzzy Metaknowledge Base (FMB). For each attribute type, it defines how to represent values and what information about them has to be stored in the FMB. In addition, in this work, authors introduce and describe specification and implementation of the FSQL - an SQL language with fuzzy capabilities in great detail. This language is an extension of the SQL language that allows users to write flexible conditions in queries, using all extensions defined by the FuzzyEER model.

We conclude that 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 robust models for fuzzy data representation in relational databases, such as the FIRST-2. The possibilities to translate conceptual models to the relational-based ones are also studied in detail. In addition, the FSQL is the first implementation of fuzzy database query language that incorporates the majority of fuzzy logic concepts.

In [10] authors have studied the possibilities to extend the relational model with the fuzzy logic capabilities. The subject was elaborated in [11], where a detailed model of Fuzzy Relational Databases (FRDB) 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 the Generalized Priority Constraint Satisfaction Problem (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] authors introduce similarity relations on the fuzzy domain which are used to evaluate the FRDB conditions. The PFSQL allows the conditions in the WHERE clause of the query to have 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.

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.

PFSQL features

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: $height = \text{triangle}(180,11,8)$.

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:

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

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.

Having stated facts and considerations in mind, we give the complete EBNF syntax of the PFSQL language at Figure 1.

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 a hypothetical student database.

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

```
SELECT name, surname
FROM MainStudentData
WHERE GPA > #triangle(9,1,0.4)#
```

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 be simplified like this:

```
SELECT name, surname
FROM MainStudentData
WHERE GPA > #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.

```sql
SELECT msd.name, msd.surname
FROM MainStudentData msd
WHERE (msd.GPA>#ling(averageGPA)# > 0.7) AND (msd.GPA4>8.5 > 0.4)
```

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.

```sql
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: triangle(6.9,0.4,0.7).

### Query execution

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 the strict negation. In case of priority statements, we use the GPFCS 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.

### 3 FRDB DATA MODEL

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. In this section we give a brief description of this model.

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.

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:

$$D = D_C \cup F_D \cup L_L$$  \hspace{1cm} (3)

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 all 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 [11].
4 Fuzzy JDBC Driver

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 2.

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.

Figure 3 shows class diagram that illustrates the structure of the fuzzy JDBC driver.

There are four main classes – FuzzyDriverImpl, FuzzyConnectionImpl, FuzzyStatementImpl and FuzzyResultSetImpl that represent wrappers for JDBC API interfaces: DriverManager, Connection, Statement and ResultSet. These classes utilize mechanisms offered by the JDBC API in background and add fuzzy JDBC driver functionalities to them. Users do not have direct access to these classes, they can be called indirectly using interfaces shown at the Figure 3. These interfaces hide details of the implementation of the fuzzy JDBC driver and offer only certain directly usable methods to end users. Class FuzzyMetaDataException represents a Java exception that can be generated when an error occurs.

Classes Transformer and FRSGenerator contain implementation of vital mechanisms offered by the fuzzy JDBC driver. The PFSQL query execution has two main phases. The first one is the transformation of the query syntax tree, and the second one is the fuzzy satisfaction degree calculation. Both processes are essentially recursive tree traversals. The first process changes the structure of the tree in the way described in section 2, so that the resulting tree represents an ordinary SQL query. Class Transformer contains the implementation of this process. Resulting SQL query is then executed against the database using ordinary JDBC driver. It is essential to note that the result of the execution of this query contains values for all fuzzy attributes mentioned in the original PFSQL query (see section 2).

Results returned from the database are processed again by the PFSQL mechanisms. The original PFSQL query tree is traversed recursively, and in the process, fuzzy satisfaction degrees are added to every tuple in the result set. This altered result set is then returned to the Java program using front end classes. This process is implemented in the FRSGenerator class.

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.

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.

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. Here we give a brief comparison of PFSQL to the FSQL, one of the most advanced fuzzy SQL languages. The fuzzy database query language FSQL is built on top of the FIRST-2 model using Oracle DBMS and PL/SQL stored procedures [9]. 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 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 [11] in detail, this query language allows priority statements to be specified for query conditions. For calculating the membership degree of query tuples when priority is assigned to conditions, we use the GPFCSP. 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 API specification. Authors intend to study and solve these problems in the future.

6 ACKNOWLEDGEMENT
Authors would like to acknowledge the support of the Serbian Ministry of Science, project “Abstract Methods and Applications in Computer Science” No. 144017A.

7 REFERENCES