Extending Relational Data Access Programming Libraries for Fuzziness: The FJDBC Framework

Miguel-Angel Sicilia Urbán[1] and Elena García Barriocanal[2]

[1] DEI Laboratory, Carlos III University Avda. de la Universidad, 30
28911 Leganés, Madrid, Spain
msicilia@inf.uc3m.es
http://www.dei.inf.uc3m.es
[2] Alcala University,
Alcala de Henares, Madrid, Spain
elena.garciab@uah.es

Abstract. Fuzzy relational databases have been extensively studied in recent years, resulting in several models and constructs, some of which are implemented as software layers on top of diverse existing database systems. Fuzzy extensions to query languages and end-user query interfaces have also been developed, but the design of programming interfaces has not been properly addressed. In this paper, we describe a software framework called FJDBC that extends the *Java Database Connectivity API* by allowing fuzzy queries on existing relational databases, using externally-stored metadata. Since the main design objective of this extension is *usability* for existing database programmers, only a restricted subset of extensions (supported also by an extended object modelling notation) has been included. The overall design of the framework and some experimental data are also described.

1 Introduction: Programming Interfaces for Extended-For-Fuzziness Databases

1.1 Motivation and Related Work

The research area of fuzziness in database management systems (DBMS) has resulted in a number of models aimed at the representation of imperfect information in databases, or at enabling non-precise queries (often called *flexible queries*) on conventional database schemas. The former include both relational models (in which the two major approaches are similarity/proximity relation-based models [2, 8], and possibility distribution-based ones [1]) and object-oriented models [3]. Flexible querying is in many cases achieved by extending the syntax and semantics of the SQL language [7]. These models extend existing systems and have been applied in real-world situations. Nonetheless, as a matter of fact, no widespread commercial implementation of a 'native' fuzzy database management system (FDBMS) currently exists. Since internally extending commercial relational systems seems to be a difficult to justify effort for database vendors

(despite of the fact that even fuzzy physical database access mechanisms have been proposed, for example [12]), the approach taken by many researchers is that of building some form of software layer on top of an existing DBMS (for example, Sonayst's *Fuzzy Query* [3] product can add flexible querying capabilities to any ODBC-compliant DBMS).

In this work, we concentrate on extending programming interfaces, rather than on extending the declarative SQL language or end-user querying capabilities. Our ultimate aim is that of devising an easy-to-understand and useful set of fuzzy database constructs delivered as an extension of existing database programming libraries. Our target programming interfaces are JDBC libraries, although our approach can be easily migrated to similar database access frameworks like Microsoft's *ActiveX Data Objects*(ADO). We have called our libraries FJDBC ('f' standing for fuzzy), since they're closely tied to JDBC libraries and follow strictly their programming semantics.

Related work in the specific field of relational systems includes extensions to small desktop database engines like FQUERY for Microsoft Access [13], and proprietary extensions for large RDBMS like FSQL for Oracle [5].

1.2 JDBC Essentials

Sun's JDBC is an application programmer's interface (API) that provides access to relational databases through existing or new drivers in Java programs [4]. The latest JDBC specification (version 3.0) has adopted those features found in the SQL99 standard that are already widely supported, but they're not covered here, and therefore, we'll restrict ourselves to discuss 2.0 version features.
JDBC provides... [[[TO-DO]]]

1.3 Design Principles

Since the ultimate aim of our research is to effectively bring fuzzy programming to common database software developers, we have departed from the human understanding of the existing JDBC interfaces, and not from fuzzy database models. This approach tries to accommodate only those fuzzy modelling notions that are close to the semantics of the classes that are part of the API, and is not concerned with achieving maximum coverage of the diverse fuzzy database notions. Our work represents a novel approach in fuzzy extensions of relational databases due to the following facts:

- We focus on application programmer's technology, and not on command-based or graphical query interfaces, so that the task model of the database programmer is the driving design input.

[3] http://fuzzy.sonalysts.com/
[4] http://java.sun.com/products/jdbc/

- As a result of its design, it can be used with any JDBC-compliant database (this includes the vast majority of commercial systems) by adding some simple *external* metadata, allowing for a fuzzy (re-)interpretation of existing legacy databases.
- It was also designed to give explicit support conceptual notions that arise in some extensions of fuzzy conceptual modeling, specifically, it allows for the conversion of some forms of extended-with-fuzziness Unified Modeling Language (UML) models [9].

The main design objective of our work is *usability* [5], and more specifically the following general principles or heuristics:

- *Consistency*, that is, avoiding the break of the original JDBC semantics.
- *Simplicity*, minimizing the number of added programming constructs.
- *Ease of learning*, measured in terms of time to master the new features of the interfaces.

The design of FJDBC follows the principle of "minimal change" from existing database programming structures, namely, minimal added interfaces and/or operations and minimal extensions to query languages. In this sense, we have designed the extension as a thin software layer on top of JDBC.

The rest of this paper is organized as follows. [[[TO-DO]]]

2 FJDBC Underlying Model

We'll describe the underlying JDBC model using Chen's notation [4].

The Fuzzy Relation Model [] relaxes the relational framework by allowing a tuple to belong to a particular relation in a degree between [0,1], instead of the classical 0,1. This 'fuzziness at the object or tuple level' can intuitively be implemented in a straightforward manner by using a "special" attribute tm storing the belonging grade of every tuple in its same relation. A second design choice is storing the fuzzy subset in a separate relation, modelling its relationship with its crisp domain as a standard relational integrity constraint.

3 Framework Description

3.1 Extending Existing Drivers Through Delegation

JDBC driver implementers must provide a small footprint class that implements `java.sql.Driver` interface. A driver class is responsible for its self-registration by providing an instance of itself to `java.sql.DriverManager` class. Our solution is that of providing a wrapper for an existing JDBC driver and delegating behavior to the internal instance, as an application of the *Proxy* design pattern [6]. Our driver can be loaded into a *Java Virtual Machine* (JVM) just as any other one, for example, with the typical fragment of code:

[5] Ease of use was also one of the objectives of JDBC design

```
try{
    Class.forName("es.uc3m.inf.dei.fjdbc.FuzzyDriver");
}catch(ClassNotFoundException e)
    {System.out.println(e);}
```

It should be noted that the JDBC driver that will be wrapped by our driver is internally loaded in the same fashion as a result of connection establishment. Connections are obtained by using a special protocol, as showed in the following code fragment.

```
Connection con=null;
try{
  con = DriverManager.getConnection("jdbc:fuzzy:prueba.xml",
                                    "", "");
  }catch(Exception e)
          {e.printStackTrace(); ...}
```

The JDBC protocol we have defined begins with the `jdbc:fuzzy` specifier followed by an string that should indicates the location of the metadata file for the database. That location must be specified as a currently a relative path pointing to the location of an XML file [11](this approach will be subject to change in the near future to a more flexible one that uses an URI [10] instead).

3.2 Metadata Specifications

The previously mentioned XML file contains metadata about the storage of fuzzy relations. An example of such a file could be the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<fuzzydb>
    <driver>sun.jdbc.odbc.JdbcOdbcDriver</driver>
    <fuzzy_relations>
        <relation label="sporadic" table="USERS" oidTable="oid" childTable="esporadicos"
                 muAtt="mu" childForeignKey="oiduser"/>
        <relation label="frequent_visitors" table="USERS" oidTable="oid"
                 childTable="habituales" muAtt="mu" childForeignKey="oiduser"/>
    </fuzzy_relations>
    <connString>jdbc:odbc:users</connString>
</fuzzydb>
```

A fuzzy database is described by specifying the JDBC connection string and driver of the real database, and the metadata through a list of (fuzzy) relation tags. The rationale for the configuration data is described as follows:

- Label: the linguistic label that (uniquely) identifies the fuzzy set in the underlying data store.
- Table: the table that stores the elements of the domain of the membership function that defines the fuzzy set.

- OidTable: the key attribute that identifies the elements of the domain in Table.
- ChildTable
- MuAtt
- ChildForeignKey

Note that a fuzzy relation defined by the addition of an attribute to a table is just a specific case in which Table and ChildTable are the same. Note that FuzzyConnection is designed as a case of the Proxy pattern [5], in which "intelligence" can be understood as the maintenance of metadata. Figure 2 shows the corresponding class diagram, where FuzzyConnection acts as Proxy, the underlying connection (the generic aConnection in the diagram) acts as the real subject, and java.sql.Connection is the pattern's Subject.

3.3 Queries and Iteration

3.4 Fuzzy Metadata Access

4 Usability and Performance Facts

5 Conclusions and Future Work

References

1. Bosc, P. and Pivert, O.: Fuzzy querying in conventional databases, In Fuzzy Logic for the Management of Uncertainty, Zadeh, L. and Kacprzyk, J. (Eds), John Wiley, New York, (1992) 645–671
2. Buckles, B.P. and Petry, F.E: A fuzzy representation of data for relational databases. Fuzzy Sets and Systems, 7, (1982) 213–226
3. De Caluwe R. (ed.) Fuzzy and Uncertain Object-Oriented Databases. Concepts and Models. World Scientific, Singapore (1997)
4. Chen, G.: Fuzzy Logic in Data Modelling : Semantics, Constraints, and Database Design. The Kluwer International Series on Advances in Database Systems (1998)
5. Galindo, J., Medina,J.M., Pons, O., Cubero,J.C.: A Server for Fuzzy SQL Queries. In T. Andreasen, H. Christiansen and H.L. Larsen (Eds.), Lecture Notes in Artificial Intelligence (LNAI) 1495, Springer, (1998) 164–174
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Elements of Reusable Object Oriented Design. Addison Wesley (1995)
7. Rasmussen, D. Yager, R.R.: SummarySQL - A Flexible Fuzzy Query Language. In Proceedings of the 1996 Workshop on Flexible Query-Answering Systems (FQAS'96) (1996) 1–18
8. Shenoi,S. and Melton,A. Proximity relations in the fuzzy relational database model, Fuzzy Sets and System, 31, (1989) 285–296.
9. Sicilia, M.A., Garca, E. and Gutirrez, J.A., Integrating Fuzziness in Object-Oriented Modelling Languages: Towards a Fuzzy-UML. Proceedings of the sixth International Conference on Fuzzy Sets Theory and its Applications (FSTA'02) (2002).
10. Berners-Lee, Fielding, Masinter, Uniform Resource Identifiers (URI): Generic Syntax; Internet Draft Standard August, RFC2396 1998

11. W3C, Extensible Markup Language (XML) 1.0; World Wide Web Consortium Recommendation, available at http://www.w3.org/TR/REC-xml.
12. Yazici,A. and Cibiceli, D.: An Access Structure for Similarity-Based Fuzzy Databases. Information Sciences **115(1-4)** (1999) 137–163
13. Zadrozny,S., Kacprzyk, J: FQUERY for Access: towards human consistent querying user interface. Proceedings of the 1996 ACM Symposium on Applied Computing (SAC'96) (1996) 532–536