Mathware & Soft Computing 12 (2005) 129-153

A First Approach to the Multipurpose Relational Database Server

I.J. Blanco[1], C. Martínez-Cruz[1], J.M. Serrano[2] and M.A. Vila[1]
[1]Dept. Comp. Science and Art. Intelligence, University of Granada
C/ Periodista Daniel Saucedo Aranda S/N, 18071, Granada, SPAIN
{*iblanco,cmcruz,vila*}*@decsai.ugr.es*
[2]Dept. Computer Science, University of Jaen
Las Lagunillas Campus, 23071, Jaen, SPAIN
jschica@ujaen.es

Abstract

In this paper, an architecture and an implementation of a multipurpose relational database server are proposed. This architecture enables classical queries to be executed, deductions to be made, and data mining operations to be performed on fuzzy or classical data. The proposal of this integration is to combine several ways of querying different types of data. In order to achieve this, a combination of existing meta-knowledge bases and new data catalog elements is presented. We also introduce a language for handling all these data coherently and uniformly on the basis of classical SQL sentences.

1 Introduction

In order to increase the expressiveness of Codd's relational model [14], various extensions have been proposed. While some of these improvements focused on the integration of fuzzy logic into the model so that imprecise and flexible values can be represented [6, 21], another extension of the model was based on the logical representation of a relational database[17] and the possibility of making deductions using classical inference mechanisms. Following the introduction of these extensions, it was obvious that it would be interesting to merge fuzzy relational databases and logical relational databases [1, 4, 5].

Data mining is the process by which information is extracted from a potentially large volume of data. The data mining community is currently well aware of the importance of databases in the process. Because of this, Imielinski et al.[18] proposed merging databases with data mining, and for a *Knowledge Discovery and Data Mining System* (*KDD System*) to be converted into a *Knowledge Discovery and Database Mining System* (*KDDB System*). On the basis of this, Carrasco [11, 9, 10, 8] has proposed the integration of data mining operations with a fuzzy extension of the relational database model GEFRED[21].

All architectures and implementations shown in this paper have been suggested on the basis of a specific proposal for fuzzy relational databases, called GEFRED[21]. Each of these has been developed to deal with a specific type of information (fuzzy, logical or referring to data mining processes). When the capabilities of all these implementations are merged, the power of the relational model is increased since complex queries can be executed. With such a system, data mining on relations that are defined using logical rules is possible.

The aim of this paper is therefore to combine data mining, and flexible and logical approaches in order to enhance the query capability of the system. This proposal of an extended RDBMS enables new architectures and implementations to be included due to its scalability.

The paper is organized in the following way: Section 2 describes previous implementations and architectures; Section 3 examines the architecture for the unified server and the information which needs to be added so that existing servers can be connected; Section 4 presents an example of how the data are stored in the database when a query has been run on the new architecture; and finally, Section 6 presents the conclusions and future lines of research.

2 Previous Implementations, Architectures and Models

As we mentioned above, we propose to combine existing implementations for fuzzy querying, making deductions with imprecise data, and data mining querying the relational model into a unified architecture. These existing implementations are described in this section.

2.1 Representing Imprecise Information in the Relational Model

2.1.1 Theoretical Model (*GEFRED*)

This model was introduced by Medina et al. [21] to represent flexible and imprecise information within the relational model. In order to achieve this, the concept of domain is extended into the *generalized fuzzy domain.* This extension of the concept enables a relational attribute to store values such as non-numerical discrete values, numerical values, a set of possible non-numerical discrete values, a set of possible numerical values, possibility distributions on the basis of a non-numerical domain and possibility distributions on the basis of a numerical domain. This definition enables three special values to be represented: *unknown* (when no concrete value is known), *undefined* (when the attribute is not applicable), and *null* (when the value can be unknown or undefined).

The second extension introduced by this model is the concept of the *generalized fuzzy relation* as a pair $(\mathcal{H}, \mathcal{B})$. $\mathcal{H}$ is called the *head* (the scheme of the relation where each attribute can take a value in a generalized fuzzy domain with a compatibility value in the $[0, 1]$ interval) and $\mathcal{B}$ is called the *body* (the instance of the relation as a set of tuples).

Due to the extension of the domain concept, the concept of the comparison operator is also extended into a *generalized fuzzy comparison operator* concept (see Table 3).

2.1.2 An Architecture (*FIRST*)

In order to represent all the values in a generalized fuzzy domain, various authors entered three new data types into a classic RDMBS [22]. These data types are:

- *Fuzzy data type 1*, or *CRISP data type*, which represents those attributes storing classical data which can be fuzzy queried;
- *Fuzzy data type 2*, or *POSSIBILISTIC data type*, which represents those attributes storing fuzzy data represented using trapezoid possibility distributions defined on a numerical domain, (values for the attributes are shown in Table 1); and
- *Fuzzy data type 3*, or *SCALAR data type*, which allows attributes storing fuzzy data to be represented using possibility distributions defined on a non-numerical domain (see Table 2).

Table 1: Fuzzy Type 2 Representation

Value Types	**FT**	**F1**	**F2**	**F3**	**F4**
Unknown	0	NULL	NULL	NULL	NULL
Undefined	1	NULL	NULL	NULL	NULL
Null	2	NULL	NULL	NULL	NULL
Crisp	3	d	NULL	NULL	NULL
Label	4	Fuzzy_ID	NULL	NULL	NULL
Interval[n,m]	5	n	NULL	NULL	m
Approx(d)	6	d	d-margin	d+margin	margin
trapezoid$[\alpha, \beta, \gamma, \delta]$	7	α	β	γ	δ

Table 2: Fuzzy Type 3 Representation

Value Types	**FT**	**FP1**	**F1**	**...**	**FPn**	**Fn**
Unknown	0	NULL	NULL	...	NULL	NULL
Undefined	1	NULL	NULL	...	NULL	NULL
Null	2	NULL	NULL	...	NULL	NULL
Simple	3	p	d	...	NULL	NULL
Distribution	4	p_1	d_1	...	p_n	d_n

Type	Comparator	Meaning
Possibility	FEQ	Fuzzy Equal (scalars or fuzzy numbers)
	FGT	Fuzzy More Than
	FGEQ	Fuzzy More or Less Than
	FLT	Fuzzy Less Than
	FLEQ	Fuzzy Less Than or Equal To
	MGT	Fuzzy Much More Than
	MLT	Fuzzy Much Less Than
Necessity	NFEQ	Necessary Equal To
	NFGT	Fuzzy Necessary More Than
	NFGEQ	Fuzzy Necessary More Than or Equal To
	NFLT	Fuzzy Necessary Less Than
	NFLEQ	Fuzzy Necessary Less Than or Equal To
	NMGT	Fuzzy Necessary Much More Than
	NMLT	Fuzzy Necessary Much Less Than

Table 3: Fuzzy Comparators

The aim of this extension is to provide a new set of capabilities to a classical RDBMS. So that this may be achieved, all accesses to relations in the data catalog must be intercepted so that new types and relations can be processed. Some new data catalog relations involved in this processing have therefore been defined, and this new set has been named the *Fuzzy Meta-knowledge Base (FMB)*. Each FMB relation is described below:-

- *FUZZY_COL_LIST relation*, storing information about attributes of relations that can contain fuzzy data or can be fuzzy queried;
- *FUZZY_OBJECT_LIST relation*, storing common information about all the *fuzzy concepts* stored in the database such as labels;
- *FUZZY_LABEL_DEF relation*, storing the possibility distribution related to every fuzzy label defined in the database;
- *FUZZY_APPROX_MUCH relation*, storing information for designing possibility distributions on predefined fuzzy concepts in the database such as *greater than*, *much greater than*;
- *FUZZY_NEARNESS_DEF relation*, storing information about similarity relations between every pair of values of a fuzzy type 3 attribute;
- *FUZZY_COMPATIBLE_COL relation*, storing those attributes whose domain is copied from another fuzzy attribute in the database; and
- *FUZZY_QUALIFIERS_DEF relation*, storing the minimum threshold that is assigned to every qualifier and is defined on a linguistic label.

A set of fuzzy comparators has also been proposed and implemented for making fuzzy queries. All these comparison operators, which are described in Table 3, are implemented as procedures in the database system.

2.1.3 An Implementation (*FSQL*)

In order to manage all the concepts and relations defined by the *FIRST* architecture, an extension of the *Data Description Language* (*DDL*) was introduced. This extension includes a syntax and an implementation of several BNF expressions such as the ones shown below.

The following sentence shows how to define the previously described fuzzy attributes:

```
CREATE TABLE table_name
   '(' { col_name (clasic_type |
                  ( FTYPE1  '('margin ','much')' clasic_type |
                    FTYPE2 '('margin ','much')' clasic_type |
                    FTYPE3 ['('number')'][DOMAIN col_reference]
                   ) [ NULL |ONLY LABEL | NOT NULL | NOT UNDEFINED |
                       NOT UNKNOWN | NOT LABEL | NOT CRISP |
                       NOT TRAPEZOID | NOT INTERVAL | NOT APPROX]) }+
   ')'
   table_clauses;
```

where *margin* represents the triangular base in an approximate (*approx*) value, *much* represents the minimum distance required in order for two values to be considered different. *Col_reference* is a reference to a previously defined FTYPE3 attribute and the remaining commands (i.e. NOT LABEL, NOT CRISP, etc.) restrict the types allowed in the definition.

This expression defines linguistic labels as values for the fuzzy domain of fuzzy types 1 and 2:

```
CREATE LABEL
    (    * ON column FROM colum   |
          ID ON column VALUES alfa,beta, gamma, delta
    ) ;
```

This expression states symbols as values for a fuzzy type 3 attribute and establishes the values of the similarity function for each pair of domain values.

```
CREATE NEARNESS ON column
    LABELS label_name, label_name {',' label_name}* VALUES
    similarity_value{',' similarity_value}*
```

Furthermore, the sentences for altering (ALTER) and dropping (DROP) the structures mentioned above have been implemented.

An extension of the *Data Management Language* (*DML*) was introduced to handle tuples within this relation [2, 15], with an extended syntax for the *INSERT* sentence:

```
INSERT INTO table_name VALUES
  '(' {  clasic_value |
        UNDEFINED | UNKNOWN |NULL |
        \$'[' number , number, number, number ']' |
        id | # number   }+
  ')'
```

where the symbol *$* enables the representaion of trapezoidal and interval values, the symbol *#* that of triangular values, and *id* allows a linguistic label to be referenced.

In order to query all the data defined by the *GEFRED* Model and implemented by the *FIRST* architecture, Galindo[16, 15, 2, 3] introduced an extension of the *SELECT* relational sentence (*DML* sentence) which incorporates management of the fuzzy comparison of the extended model and allows classical or fuzzy data to be queried.

2.2 Representing Logical Information in the Relational Model

2.2.1 Theoretical Model: *The Logical View of a Relational Database*

The translation of a relational database into a logic database was first proposed by Gallaire et al.[17]. In this proposal, the authors provided translations for every element in the relational model into the logic language and the necessary axioms for the correct and complete description of a database. This correspondence enabled the application of logic programming languages such as the *Prolog*[7, 19] language to a database and the development of a specific language for deduction in relational databases called *Datalog*[13].

After the first attempt at introducing fuzzy data into the relational model, Vila et al.[23] proposed a logical representation for a fuzzy relational database. As in the classic relational case, this correspondence includes the translation for every element in the fuzzy relational database and the necessary axioms for the complete representation of data in the logical model.

On the basis of this logical equivalence, Blanco et al.[4] defined two new kinds of relations depending on whether the instance of the relation is explicitly stored (*extensional relation, either fuzzy or classical*) or if it is calculated using logical rules (*intensional relation, either fuzzy or classical*). The classical rules are extended into the concept of *a generalized fuzzy rule with matching degree* in order to deal with fuzzy values and fuzzy comparisons.

2.2.2 An Architecture (*Extended FREDDI*)

The *FREDDI* architecture was proposed by Medina et al.[20] to incorporate a way of storing and executing logical rules within an RDBMS. In order to combine *FREDDI* with the implementation for a fuzzy relational database *FIRST*, Blanco et al.[4] introduced an extension of *FREDDI*. All of these extensions are represented in a set of catalog relations called the *Rule Base (RB)* and described below:

- *DED_INTENSIONAL_CATALOG relation*, storing an index of all the intensional relations in the database;
- *DED_INT_TABLE_DESCRIPTION relation*, storing the index of all the rules for the calculation of the instance of an intensional relation;
- *DED_RULE_DESCRIPTION relation*, representing the definition of every rule defining the content of an intensional relation as a conjunction of predicates (related to other extensional and intensional predicates) and comparisons;
- *DED_PREDICATE_DESCRIPTION relation*, describing the structure of every predicate in every rule (variable, position, ...); and
- *DED_COMPARISON_DESCRIPTION relation*, describing the structure of every predicate that is present in any rule (variables, comparison operator and threshold).

2.2.3 An Implementation (*DFSQL*)

Intensional relations, extensional relations, and generalized rules in the *Extended FREDDI* architecture are managed by means of an extension of the *DDL* language provided by Blanco et al.[2]. This extension includes the syntax and management of expressions such as:

```
CREATE INTENSIONAL TABLE table_name
 '(' list_columns ')' ;
```

this expression enables an intensional table to be created, specifying the attributes of the relation to replace *list_columns*.

```
CREATE RULE table_name FOR '('
   list_identifier  AS [NOT]
        { ( predicate_name '(' identifier
          (FEQ|NFEQ) identifier.identifier [THOLD number] ')' |
          predicate_name '(' identifier SOURCE colum
          (FEQ|NFEQ) identifier.identifier [THOLD number] ')' |
          identifier.identifier (FEQ|FGT|FGEQ|FLT|FLEQ|MGT|MLT|
          NEQL|NFGT|NFGEQ|NFLT|NFLEQ|NMGT|NMLT)
          identifier.identifier [THOLD number])
          [AND]}*  ')'
```

where *list_identifier* identifies the name of the predicate or predicates that are being defined. The meaning of FEQ, NFEQ, FGT, FGEQ, FLT, FLEQ, MGT, MLT, NEQL, NFGT, NFGEQ, NFLT, NFLEQ, NMGT, NMLT is described in Table 3 and all of these represent fuzzy comparators.

The sentences for dropping these objects are also defined.

¿From the functional point of view, the calculus of an intensional relation instance can be performed by two implementations of the basic algorithms *Prolog* and *Datalog* introduced by Blanco et al.[4, 5].

2.3 Data Mining in the Relational Model

2.3.1 Theoretical Model (*G*EFRED*)

Based on the GEFRED model, this model redefines the previous concept of fuzzy domain, giving it a more universal sense: the *Complex Generalized Fuzzy Domain*(GEFRED*) [12, 11] where its definition is not limited to a specific domain. In addition, GEFRED* defines complex data types where more than one classic attribute can be included. The concept of *Generalized Fuzzy Comparator* is also extended into the concept of *Complex Generalized Fuzzy Comparator* and this provides the mechanism for comparing values in this domain. On the basis of this complex domain, a *Complex Generalized Fuzzy Relation* has been defined in the same way as the *Generalized Fuzzy Relation.*

2.3.2 Architecture: (*DmFIRST*)

A new data type is introduced to represent the new set of values that were not contained in the *Generalized Fuzzy Domain.* This data type is a super-type that can contain values of any classical data type and the three data types introduced by *F*IRST (*Fuzzy data type 1, Fuzzy data type 2*, and *F*uzzy data type 3). *Fuzzy data type 4*[11] consists of:

- a part storing information about the value, and
- another part storing meta-data about the type of value.

This type is included in an intermediate modification of the *FIRST* architecture, called *FIRST*.* The representations of the attributes defined in the *FIRST* architecture are now collected in *Fuzzy data type 4.*

Following the idea expressed in FIRST, the data catalog relations that are involved in this processing are made up of an extension of the Fuzzy Meta-knowledge Base (FMB*)[11]. These relations provide the capability of representing the new fuzzy data type and executing data mining operations:-

- *DMFSQL_LABEL_DEFINITION relation*, storing information about the fuzzy labels defined for the new fuzzy data type;
- *DMFSQL_COL_COL relation*, storing information about all the attributes capable of containing or handling this new type;
- *DMFSQL_FUNCTIONS relation*, storing the index of functions that define the extended comparators;

- *DMFSQL_FUNCTIONS_COL relation*, storing relations between an attribute and its related comparison function;
- *DMFSQL_COL_PAR relation*, storing the interface to call functions implementing extended comparisons.

There are also another two relations in the data catalog that are part of the so-called DmFMB. This extension of the catalog enables the system to define certain required parameters for executing some data mining operations such as:

- clustering
- fuzzy classification
- characterization
- fuzzy global dependencies searching: Fuzzy Functional Dependencies(FFD) and Fuzzy Gradual Dependencies (FGD)

The data mining fuzzy meta-knowledge base (DmFMB) maintains the necessary information for performing a data mining operation. Some of these relations store temporary data, determine the default function for an operation, parameters such as the minimum confidence or support, etc. and these are:

- *DMFSQL_PROJECT relation*, storing general information about the data mining process, such as the source relation, the clustering process status;...
- *DMFSQL_COL_LIST relation*, storing information about source attributes that are significant for the data mining process, the type of operation, and the relation created for storing the results, called *project*.

These relations enable table names to be defined where the results of data mining operations will be stored. These tables are created when a data mining project is defined. The definition of a data mining project is described in the following section.

2.3.3 An Implementation (*DmFSQL*)

In order to manage all the concepts and relations defined by the FIRST* and the DmFIRST architecture, in [11], Carrasco proposed the syntax of an extended DDL language. This extension includes the syntax of expressions such as *CREATE_MINING*, *ALTER_MINING*, *DROP_MINING*, *GRANT_MINING*, or *REVOKE_MINING* to enable the user to create, drop, alter, or revoke constraints in a data mining project. A data mining project stores the required information for performing a data mining process. The syntax of the *Create_mining project* is described below:

```
CREATE_MINING PROJECT id_project [ON OWNER id_owner]
   ON TABLE id_table
   WITH COLUMNS FOR
```

```
    [CLUSTERING '(' list_columns_clu |list_columns_clu_cen')']
    [CLASSIFICATION '('list columns_cla')']
    [FGLOBAL_DEPENDENCIES '('
        {ANTECEDENT list_columns_fgd [THOLD_ANT thold]
         CONSEQUENT list_columns_fgd [THOLD_CON thold] }
         ...
    ')']
     [STORAGE_SPECIFICATIONS storage_stm]
```

This BNF sentence declares a data mining project, and fixes the parameters of the mining process. A data mining project is associated with a relation, *id_table* (and sometimes with a schema or owner). Each data mining process requires the definition of different parameters:

- A clustering process specifies in *list_columns_clu* and *list_columns_clu_cen* the set of relevant attributes in the process and certain specifications for obtaining the distance matrix such as weight establishment, specification of the comparison operator, etc. If a classification process is applied after the clustering process, parameters for obtaining the attribute centroids will also be established.

- A classification process specifies in *list_columns_cla* the set of relevant attributes in the process and some specifications about the comparison operator, if it is necessary to use a t-norm or a t-conorm, weight establishment, etc.

- A search of functional dependencies involves defining some attributes as antecedents and some as consequents in *list_columns_fgd* and whether they must accomplish a certain threshold.

Carrasco[11] also implements a sentence for managing *Fuzzy data type 4*.

Furthermore, the DML defined for the data mining process is included as an extension of the SELECT sentence[11]. This sentence incorporates operations for clustering, classifying, or extracting global functional dependencies. One example of the syntax of this sentence when a clustering process is executed is:

```
SELECT MINING CLUSTERING id_project
    INTO TABLE_CLUSTERING id_table_result_clu
    [',' TABLE_CENTROIDS id_table_result_cen ]
    OBTAINING {n_clusters|OPTIMAL_ABS|OPTIMAL_H3|OPTIMAL_MED}
    CLUSTERS
```

where *id_project* identifies a previously defined data mining project, *id_table_result_clu* represents the name of a new relation created for storing the clustering results, *id_table_result_cen* is the name of the relation where the characterization process stores all the sets and centroids obtained. The parameter *n_clusters* specifies the number of clusters to be obtained and *OPTIMAL_ABS,OPTIMAL_H3 and OPTIMAL_MED* allow the system to obtain the number of clusters using other implemented processes.

The example in Section 4 shows a sentence for obtaining some fuzzy global dependencies.

3 Architecture for a Unified Server

Using the previously described architectures, we propose the infrastructure for a unified server that integrates the capabilities of all of these architectures and enables their functionalities to be combined.
These architectures are defined below and include:

- a database server for fuzzy querying;
- a database server for making deductions with classical and fuzzy data; and
- a database server for executing data mining operations.

This integration is capable of processing several types of queries in the same sentence. For example, queries about deductions with fuzzy data or deductions using the results of a data mining process can be included in a single sentence.
Moreover, the proposed architecture establishes a mechanism to add more implementations of new servers with different functionalities on the GEFRED extension of the relational model.

The elements to be described are:

- the combination of the different meta-knowledge bases; and
- the way to process a query so that the server can decide which modules must participate in its execution and the order for this participation.

3.1 Meta-knowledge Base (MB)

This set of catalog relations stores the definition of objects and data types used by the different modules and by the unified server. It is divided into:

- FMB, which represents fuzzy data types, fuzzy domains, fuzzy labels; ...
- RB, which stores intensional predicates and their definition, described by means of several logical rules;
- FMB*, which defines a new data type capable of representing text, XML, objects,... and the operations that can be applied to this data type;
- DMFMB, which stores information about clustering, classification and fuzzy global dependencies, for applying data mining operations on classic data types of the classical RDBMS, or the fuzzy types defined in FMB.

The FMB must be defined in any system that uses either FIRST, FREDDI or DMFIRST, because the FMB implements the basic structure of fuzzy data. The fuzzy comparison operators implemented in FIRST are outside the FMB. Furthermore, the RB stores all the rules included in the system, referencing attributes that can be fuzzy or not, so that the FMB must be visible to this architecture. On the other hand, the FMB* is only an extension of the FMB for managing a new data type. The DMFMB works with all the attributes of the database defined in FIRST and in DMFIRST, so it is necessary to include the FMB relations for managing fuzzy data.

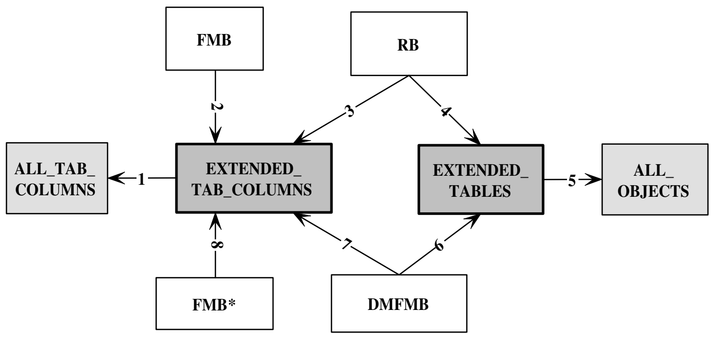

Figure 1: Extended Knowledge Base

In Figures 1 and 2, the different meta-knowledge bases are interconnected by means of two new relations. The connections introduced in order for cross-queries to be processed are:

- *EXTENDED_TABLES*, storing the relations (classical or extended) defined in the DB that can be used by fuzzy, deductive or data mining queries. These relations stored in *ALL_OBJECTS* (only those tuples referring to tables) are included in this relation (connection (5)). Consequently, this relation is a specialization of *ALL_OBJECTS* and all the relations included in it have some of the previously mentioned special characteristics. Table 5 shows the attributes of this relation and the values that it can take.

- *EXTENDED_TAB_COLUMNS*, providing information about all of the attributes (classical or extended) accessible to the user. This includes some of the columns stored in *ALL_TAB_COLUMNS* (connection (1)) and a description of this. As in the previous item, this relation is a specialization of *ALL_TAB_COLUMNS* and the tuples that this relation can contain could be the fuzzy attributes described in FIRST, the intensive attributes described in FREDDI, the attributes that can be used in a data mining process, or the attributes that can store data mining temporary information or results. The *TYPE* attribute (see Table 4) therefore stores the type of data that the attribute contains (a rule, fuzzy data, etc.).

These new relations must be related to a specific relation of the Relational Database Management System (RDBMS) catalog containing information about all the columns and tables defined in the database. In this proposal, the specific relations of the RDBMS (Oracle©) catalog (*All_Tab_Columns* and *All_Object*) are shown as an example.

The remaining connections with the new relations in Figure 1 are:

- Connections 2 and 8 enable the FMB and the FMB* to be related to *EXTENDED_TAB_COLUMNS* since they extend the definition of attributes and their domains;
- Connection 3 enables the RB to be related to *EXTENDED_TAB_COLUMNS* since the RB extends the definition of relations;
- Connection 4 makes the RB be related to *EXTENDED_TABLES* because the schema of this extension must be obtained from attributes in other relations;
- Connection 7 enables the DMFMB to be related to *EXTENDED_TAB_COLUMNS* because data mining operations are applied on attributes;
- Connection 6 enables DMFMB to be related to *EXTENDED_TABLES* because the results of these operations must be stored in other relations.

Table 4: Extended_Tab_Columns Relation

OBJ#	**COL#**	**Type**
		0 (Fuzzy Column) 1 (Logic Column) 2 (Data Mining Column)

Table 5: Extended_Tables Relation

OBJ#	**Type**	**Orig**
	0 (Extensive) 1 (Intensive)	0 (Classic Data) 1 (Fuzzy Data) 2 (Rule Description) 3 (DM Data) 4 (DM Description)

The inclusion of these unifying catalog relations makes the extended MB easily scalable.

3.2 Server Architecture and Operation

Figure 3 shows the relations between the different modules within the RDBMS. As the figure shows, there is at this moment an independent client for every module

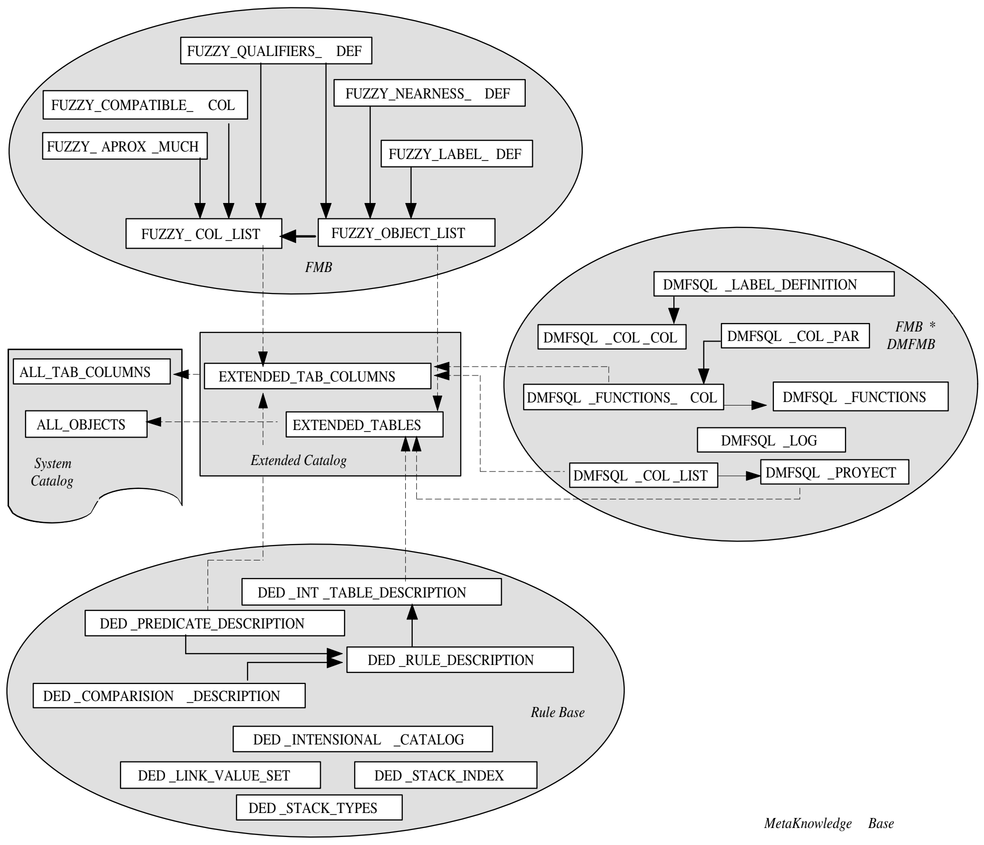

Figure 2: MetaKnowledge Base

providing the specific functionality. Not only will a simple SQL query be sent to the SQL executor directly, but also fuzzy, deductive, or data mining queries will be parsed by the appropriate module so that they may be translated into a classic SQL sentence or a set of sentences.

As Figure 3 shows, Galindo[16, 15] implemented lexical and syntactical parsers for *FSQL* language so that it could be easily modified to increase the number of processed sentences. In this way, developers of *DFSQL* and *DMFSQL* took advantage of this feature when developing analyzers for their extensions. In addition, semantic parsers are implemented separately and each translates a different query into a classical SQL sentence to be processed by the classical RDBMS.

Compilation between *FSQL* and *DFSQL* is tight because the second is implemented using the first as a basis and re-using some of its functions and modules.

Our proposal consists of a unified server that enables all flows of information to be integrated from and to a single client. This client communicates to only one server which is capable of identifying the type of relations involved in every

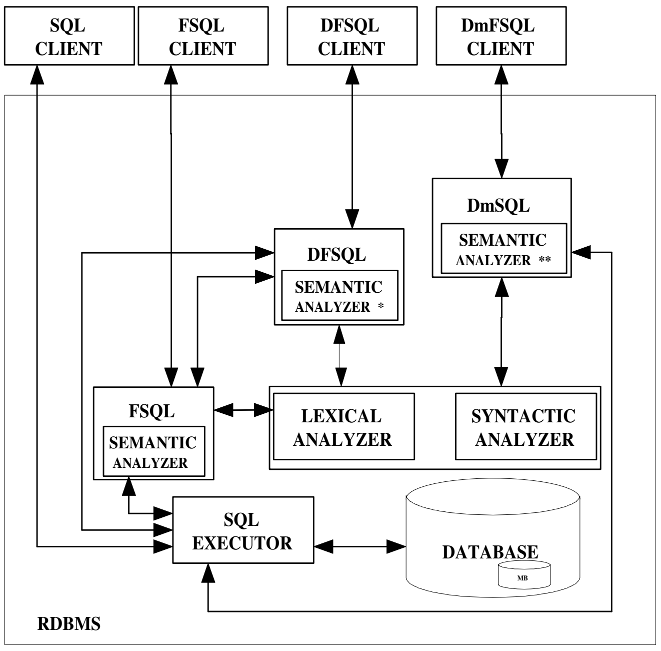

Figure 3: Independent Servers

query. Once the query has been parsed, the server controls the execution of all the modules that must translate some parts of the query, and integrates their responses. There is also another module inside the server, the *Querying Strategy Planner*, which schedules the order that the queries must be executed in order to increase the efficiency of the server. The solution strategy of this planner is to analyze a complex query (a query that implies different modules to be solved) and to determine the order of execution of each subquery that this complex query involves.

The modifications provided by this architecture are shown in Figure 4 using solid lines.

The processing of a query can be resumed as follows:

- the client application sends a query to the server;
- the query is analyzed by the server using lexical and syntactic analyzers in order to determine the modules involved in the query resolution;
- the sentence is divided (if necessary) and sent to the corresponding module (here the Querying Strategy Planner schedules the execution of the different subqueries);
- every module analyzes the assigned part of the query semantically and translates it into a classical sentence;

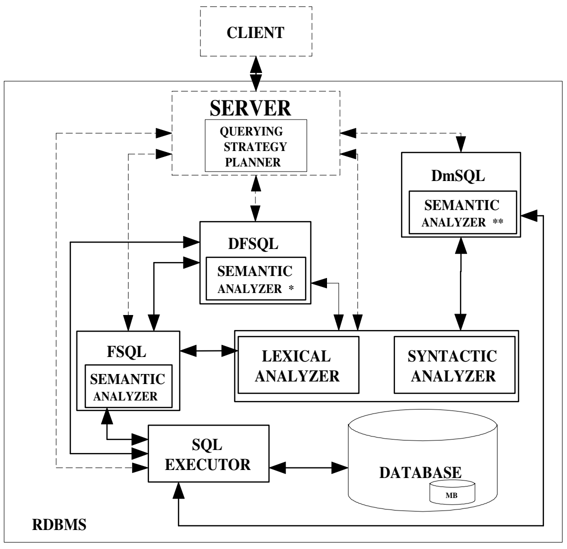

Figure 4: Multipurpose Server

- the processed part of the original query is returned to the server which integrates the translations provided by the modules into a single classical query that is sent to the SQL Executor; and
- the server formats the resulting set of tuples provided by the SQL Executor before sending it to the client application.

As Figure 4 shows, both the server and all the modules need to query the *MB*.

4 Example

As suggested in the previous sections, the integration of the different architectures provides both the capacity for parsing several types of queries and for storing the results of independent queries as relations, rules, calculated data, etc. that other processes can subsequently use.

This section shows how a typical data mining process may be combined with the fuzzy logic rules handling. More specifically, the search for fuzzy functional dependencies (FFDs) can (when they are found) generate rules that can be stored in the database as *generalized fuzzy rules with a matching degree* (described in Section 2.2).

In order to illustrate this example, we suggest the following query: we want to know if there is a relationship between a student's marks and his/her behavior. The real problem to solve is whether there is a fuzzy functional dependency (FFD) between the attribute *mark* and the attribute *behavior* in the *STUDENT* relation (Table 6).

In order to achieve this, it is necessary to explain how the information is stored in the database, and, more specifically, the structure of the data in the meta-knowledge base (MB).

Table 6: STUDENT relation

Name	**Mark**	**Behavior**
John	#8	good
Beth	10	excellent
Mary	[3,4]	bad
Tom	C	normal
Charles	2	bad
Susan	Unknown	Unknown
...	...	...

The *STUDENT* relation shown in Table 6 has been stored in the database with the structure described in Table 8, i.e. in order to represent the distinct fuzzy data types (defined in Section 2).

As we can see in the attribute F_TYPE in Table 12, *Mark* is a Fuzzy Type 2 and *Behavior* is a Fuzzy Type 3. The similarity relation of *Behavior* (described in Table 7) is stored in the MB in the *Fuzzy_Nearness_Def Relation* (Table 10).

Labels such as mark 'C' are described in the *Fuzzy_Label_Def Relation* (Table 11) of the MB. The *Fuzzy_Object_List Relation* (Table 9) stores the labels used in the attribute *Behavior* and all the labels that can be used to describe a value of the attribute *Mark*. Furthermore, this table establishes a single identifier for each label so as to avoid confusion.

Table 7: Similarity Relation s_r over BEHAVIOR

S_r**(d,d')**	**Bad**	**Normal**	**Good**	**Excellent**
Bad	1	0.8	0.5	0.1
Normal	0.8	1	0.7	0.5
Good	0.5	0.7	1	0.8
Excellent	0.1	0.5	0.8	1

This is only the specification of the data structures in the FMB, however, and the data must be defined in other relations, more specifically in the new relations of the MB. These definitions are shown in Tables 14 and 15. Table 14 contains a reference to the attributes used in this example and the type of data that they represent (fuzzy data). Table 15 keeps the description of the relations used in this example, an extensive table *STUDENT* and an intensive table *FDD* that will be described subsequently. The '-' in Table 15 means that this value is not relevant. The tuple *STUDENT* is inserted into this table once it has been defined in the database as a fuzzy table, and the tuple *FDD* inserted when the data mining process has successfully finished and an conclusion extracted from it.

Before performing any data mining operation, a new *project* must be defined on the database. This project definition generates a few new rows in certain relations of the MB (corresponding to the previous DmFMB). The sentence for defining this

Table 8: STUDENT relation

Name	**MarkT**	**Mark1**	**Mark2**	**Mark3**	**Mark4**	**. . .**	**BehaviorT**	**BehaviorP1**	**Behavior1**
John	6	8	7	9	1	. . .	3	1	2
Beth	3	10	NULL	NULL	NULL	. . .	3	1	3
Mary	5	3	NULL	NULL	4	. . .	3	1	0
Tom	4	2	NULL	NULL	NULL	. . .	3	1	1
Charles	3	2	NULL	NULL	NULL	. . .	3	1	0
Susan	0	NULL	NULL	NULL	NULL	. . .	0	NULL	NULL
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

Table 9: Fuzzy_Object_List Relation

OBJ#	**COL#**	**FUZZY_ID**	**FUZZY_NAME**	**FUZZY_TYPE**
STUDENT	Mark	0	'A'	0
STUDENT	Mark	1	'B'	0
STUDENT	Mark	2	'C'	0
STUDENT	Mark	3	'D'	0
STUDENT	Behavior	0	'BAD'	1
STUDENT	Behavior	1	'NORMAL'	1
STUDENT	Behavior	2	'GOOD'	1
STUDENT	Behavior	3	'EXCELLENT'	1

Table 10: Fuzzy_Nearness_Def Relation

OBJ#	**COL#**	**FUZZY_ID1**	**FUZZY_ID2**	**DEGREE**
STUDENT	Behavior	0	1	0.8
STUDENT	Behavior	0	2	0.5
STUDENT	Behavior	0	3	0.1
STUDENT	Behavior	1	2	0.7
STUDENT	Behavior	1	3	0.5
STUDENT	Behavior	2	3	0.8

Table 11: Fuzzy_Label_Def Relation

OBJ#	**COL#**	**FUZZY_ID**	**ALFA**	**BETA**	**GAMMA**	**DELTA**
STUDENT	Mark	0	9	9	10	10
STUDENT	Mark	1	6	6	8	9
STUDENT	Mark	2	4.5	5	6	6
STUDENT	Mark	3	0	0	4.5	5

Table 12: Fuzzy_Col_List Relation

OBJ#	**COL#**	**F_TYPE**	**LEN**	**COM**
STUDENT	Mark	2	NULL	”STUDENT.MARK”
STUDENT	Behavior	3	1	”STUDENT.BEHAVIOR”

Table 13: Fuzzy_Aprox_Much Relation

OBJ#	**COL#**	**MARGEN**	**MUCH**
STUDENT	Mark	1	3

Table 14: Extended_Tab_Column Relation

OBJ#	**COL#**	**COL_TYPE**
STUDENT	Mark	0
STUDENT	Behavior	0

Table 15: Extended_Tables Relation

OBJ#	**TAB_TYPE**	**ORIG**
STUDENT	0	1
FDD	1	-

project has the following structure:

```
CREATE_MINING PROJECT STUDENT_PRJ
   ON TABLE STUDENT
   WITH COLUMNS FOR
    FGLOBAL_DEPENDENCIES '('       {
     ANTECEDENT MARK FCOMP_GLOBAL_DEPENDENCIES FEQ THOLD_ANT 0.6
     CONSEQUENT BEHAVIOR FCOMP_GLOBAL_DEPENDENCIES FEQ THOLD_CON 0.7}
```

where STUDENT_PRJ is an identifier of a data mining project that keeps information about the parameters of the data mining process: fuzzy functional dependencies search. Some of the parameters that this project contains are the type of functional dependencies to search, the matching degree of the attributes, etc. This project has been defined as the first step in a data mining process. The structure of the DmFMB is described in Tables 16 and 17. Table 16 stores the general specification about the fuzzy functional dependence proposed and Table 17 stores information about each column involved in the fuzzy functional dependence search.

Table 16: DmFsql_Project Relation

PROJECT_NAME	**OWNER**	**OBJ#**	**STATUS_FGD**	**THOLD_ANT_FGD**	**THOLD_CON_FGD**	**CONFIDENCE_FGD**	**SUPPORT_FGD**	**...**
STUDENT_PRJ	OWNER	STUDENT	-	0.6	0.7	c	s	...

Table 17: DmFsql_Col_List Relation

PROJECT_NAME	**COL_TYPE**	**COL#**	**FUZZY_COMP_FGK**	**THOLD_FGD**
STUDENT_PRJ	A	Mark	FEQ	-
STUDENT_PRJ	Q	Behavior	FEQ	-

The fuzzy functional dependence we are searching for has been called FDD and it is described in the following expression:

0.6 - 0.7 FDD MARK $_{FEQ*FEQ} \longrightarrow$ BEHAVIOR with confidence c and support s

The goal is obviously to discover whether a student's mark influences his/her behavior, where 0.6 is the matching degree of the mark, and 0.7 is the matching degree of the behavior. In order to accomplish this operation, a DML sentence is run on the data mining server:

```
SELECT_MINING FGLOBAL_DEPENDENCIES STUDENT_PRJ
  USING T_NORM THOLD_ANT_CON
```

This query is actually the last in a set of basic queries such as the following:

```
SELECT COUNT(*) FROM STUDENT A1, STUDENT A2
WHERE(A1.NAME<>A2.NAME) AND
   (A1.MARK FEQ A2.MARK
   THOLD 0.6)   AND NOT
   (A1.BEHAVIOR NFEQ A2.BEHAVIOR
   THOLD 0.7)
```

Fuzzy functional dependence (FDD), support and confidence are calculated with a similar sentence, counting the number of antecedent and consequent appearances. If support and confidence are high enough, the FDD will be accepted and automatically stored in the database as a logic rule. If not, the functional dependence must be rejected or its matching degrees must be decreased.

The structure of this rule is:

FDD(MARK,BEHAVIOR) :- STUDENT(X,Y) $\wedge$
(X $=_{0.6}$ MARK)$\wedge$ (Y $=_{0.7}$ BEHAVIOR)

The sentence which enables us to define the rule on the database is:

```
CREATE INTENSIONAL TABLE FDD
   (MARK FTYPE2 (2,3) NUMBER (3,2)
   BEHAVIOUR FTYPE3 );

CREATE RULE FOR FDD (Mark, Behavior) AS
   STUDENT (X SOURCE Mark, Y SOURCE Behavior) AND
   ( X FEQ MARK THOLD 0.6) AND (Y FEQ BEHAVIOR THOLD 0.7)
```

where *Create Intensional Table* inserts a new tuple into Table 18 and makes a new relation *FDD* in the database, without tuples because this is an intensive relation. This process also involves inserting tuple *FDD* into the Table 15. *Create Rule* stores information in the remaining tables of the rule base (RB), i.e. Tables 19, 20, 21 and 22. In these tables, the structure of the rule, each predicate and variable that conforms to it are defined.

Table 18: Ded_Intensional_Catalog Relation

ID_PRED	**MARCADO**	**NVARS**
FDD	1	2

Table 19: Ded_Int_Table_Description Relation

Table_ID	**Rule_Id**
FDD	1

Table 20: Ded_Rule_Description Relation

Table_ID	**Rule_Id**	**Pred_Id**	**Occ_Number**	**Negated**	**Type**
FDD	1	2	1	0	0
FDD	1	-	2	0	2
FDD	1	-	3	0	2

Once we have defined this rule in the database, for each insertion run on the database, the system checks whether this tuple keeps the threshold for each attribute (the rule antecedent and consequent). If it does, the new tuple will validate the FDD and will be inserted into the database increasing the confidence and support of the rule.

The previous query has only shown how the mixed architecture has increased the number of queries that can be made on the database. Other possibilities for querying the unified database are described in the following section.

5 Operations in the New Integrated System

Once the architecture of the unified system has been designed, a number of new capabilities are introduced. In general, the purpose of this union is:

- to increase the number of operations and data types that a fuzzy RDBMS (FRDBMS) can manage, these operations include:
  - making deductions on data mining structures
  - making data mining on logical structures (such as intensive relations)
  - storing data mining results as association rules, or as logical rules, in the case of fuzzy functional dependencies
- to convert this architecture into a new, more scalable one, since the system must be extended to enable new operations and data types.

Table 21: Ded_Predicate_Description Relation

Table_ID	**Rule_Id**	**Pred_Id**	**Occ_Number**	**Var_Id**	**Col_Id**	**Source_Col**
FDD	1	2	1	3	1	Mark
FDD	1	2	1	4	2	Behavior

Table 22: Ded_Condition_Description Relation

Table_ID	**Rule_Id**	**Pred_Id**	**Occ_Number**	**Var_Id1**	**Var_Id2**	**ComOp**	**Thold**
FDD	1	-	2	3	1	6(FEQ)	0.6
FDD	1	-	3	4	2	6(FEQ)	0.7

The possibility of using fuzzy data, however, is presented in every operation of the unified system since all the architectures have been developed with this functionality.

This unified system represents a single system which is capable of making any type of query with fuzzy or classic data on a database, while enabling new processes to be implemented in the system when necessary.

In order to implement this unified architecture, the entire *Querying Strategy Planner* must be implemented so as to enable the query to be separated and scheduled depending on the operation to be performed. FIRST, FREDDI, and DMFIRST, the initial architectures, have been developed and they are currently running.

6 Conclusions and Future Work

As we have shown in this paper, this architecture connects several existing servers that are demoted to modules. It also unifies the way of communicating with a single extended RDBMS. From our point of view, the structure of *MB* allows new servers to be easily incorporated to the extension.

Added connections enable the result of a query to be used as an input for another module execution and to calculate the content of certain relations prior to a query. The automatic storing of the retrieved knowledge without an explicit query writing is another feature of the system proposed. In short, results of the different integrated server can be combined, and additional types of data can be stored and handled.

The syntax of the *FSQL* and *DFSQL* languages has been designed in line with the *SQL* philosophy, but the language incorporated by *DmFSQL* must be unified with this philosophy and some sentences for the management of data types must be completed.

Acknowledgments

This research has been partially supported by the Spanish "Ministerio de Ciencia y Tecnología" (MCYT) with projects TIC2002-04021-C02-02 and TIC2002-00480.

References

[1] J. F. Baldwin and S. Q. Zhou. A fuzzy relational inference language. *Fuzzy Sets and Systems*, (14):155–174, 1984.

[2] I. Blanco, J. C. Cubero, O. Pons, and M. A. Vila. An implementation for fuzzy relational databases. In G. Bordogna and G. Passi, editors, *Recent Research Issues on the Management of Fuzziness in Databases*, Studies in Fuzziness and Soft Computing, pages 183–207. Physica-Verlag, 2000.

[3] I. Blanco, N. Marín, O. Pons, and M. A. Vila. An extension of data description language (ddl) for fuzzy data handling. In Larsen, Kacprzyk, Zadrozny, Andreasen, and Christiansen, editors, *Flexible Query Answering Systems, Recent Advances*, Advances in Soft Computing, pages 54–64. Physica-Verlag, 2000.

[4] I. Blanco, M. J. Martin-Bautista, O. Pons, and M. A. Vila. A mechanism for deduction in a fuzzy relational database. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 11:47–66, September 2003.

[5] I. Blanco, O. Pons, J. M. Serrano, and M. A. Vila. Deduction in a gefred database using datalog. In *International Conference in Fuzzy Logic and Technology EUSFLAT 2003*, pages 550–553, September 2003.

[6] P. Bosc, M. Galibourg, and G. Hamon. Fuzzy querying with sql: Extensions and implementation aspects. *Fuzzy Sets and Systems*, 28:333–349, 1988.

[7] I. Bratko. *Prolog, Programming for Artificial Intelligence.* Addison Wesley, 2 edition, 1990.

[8] R.A. Carrasco, J. Galindo, M.C. Aranda, J.M. Medina, and M.A. Vila. Classification in databases using a fuzzy query language. In *9th International Conference on Management of Data, COMAD'98*, 1998.

[9] R.A. Carrasco, J. Galindo, M.A. Vila, and J.C. Cubero. Fsql: a tool for obtaining fuzzy dependencies. In *8th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU'2000*, pages 1916–1919, 2000.

[10] R.A. Carrasco, J. Galindo, M.A. Vila, and J.M. Medina. Clustering and fuzzy classification in a financial data mining environment. In *3rd International ICSC Symposium on Soft Computing, SOCO'99*, pages 713–720, 1999.

[11] Ramón Alberto Carrasco, María Amparo Vila, and José Galindo. Fsql: a flexible query language for data mining. *Enterprise information systems IV*, pages 68–74, 2003.

[12] Ramón A. Carrasco. *Lenguajes e Interfaces de Alto Nivel para Data Mining con Aplicación Práctica a Entornos Financieros.* PhD thesis, E. T. S. I. Informática, Universidad de Granada, Spain, 2003.

[13] S. Ceri, G. Gottlob, and L. Tanca, editors. *Logic Programming and Databases.* Surveys in Computer Science. Springer-Verlag, 1990.

[14] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[15] J. Galindo, J.M. Medina, A. Vila, and O. Pons. Fuzzy comparators for flexible queries to databases. In *Iberoamerican Conference on Artificial Intelligence, IBERAMIA'98*, pages 29–41, 1998.

[16] José Galindo, Juan Miguel Medina, Olga Pons, and Juan C. Cubero. A server for fuzzy sql queries. In *Proceedings of the Third International Conference on Flexible Query Answering Systems*, pages 164–174, 1998.

[17] H. Gallaire, J. Minker, and J. M. Nicholas. Logic and databases: A deductive approach. *Computing Surveys*, 16(2):153–185, June 1984.

[18] T. Imielinski and Heikki Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, 1996.

[19] D. Li and D. Liu. *A Fuzzy Prolog Database System.* Computing Systems Series. John Wiley & Sons, 1990.

[20] J. M. Medina, O. Pons, J. C. Cubero, and M. A. Vila. Freddi: A fuzzy relational deductive database interface. *International Journal of Intelligent Systems*, 12:597–613, 1997.

[21] J. M. Medina, O. Pons, and M. A. Vila. Gefred. a generalized model of fuzzy relational databases. *Information Sciences*, 76(1-2):87–109, 1994.

[22] J. M. Medina, M. A. Vila, J. C. Cubero, and O. Pons. Towards the implementation of a generalized fuzzy relational database model. *Fuzzy Sets and Systems*, 75:273–289, 1995.

[23] M. A. Vila, J. C. Cubero, J. M. Medina, and O. Pons. Some recent advances in fuzzy relational and fuzzy deductive databases. In *European Research Consortium for Informatics and Mathematics*, pages 161–176, Barcelona, Spain, Noviembre 1994.