FREDDI: A Fuzzy RElational Deductive Database Interface

J. M. Medina,* O. Pons, J. C. Cubero, and M. A. Vila
Department of Computer Sciences and Artificial Intelligence, University of Granada, 18071 Granada, Spain

This paper shows an architecture for Deductive Fuzzy RDBMS which integrates two approaches of fuzzy databases: the relational and the logical ones. It uses the first one to represent and manipulate imprecise information and the second one to obtain intensional information. Besides, it shows how it is possible to use the resources offered by a conventional RDBMS together with a deduction module to build a complete DFRDBMS capable of handling queries of imprecise and/or deductive nature. This task is carried out using an extension of SQL language. © 1997 John Wiley & Sons, Inc.

I. INTRODUCTION

Since the appearance of the Relational Data Base model (RDB) proposed by Codd, several approaches have tried to provide a theoretical environment for the representation and handling of fuzzy information. These approaches, which are based on the use of the fuzzy sets theory as a tool of representation and manipulation, are grouped mainly into two tendencies: Unification Models by Similarity Relations, and Possibilistic Models.

The first approach is described in Buckles and Petry's proposal.[1,2] The second approach covers several proposals, the most important of which were introduced by Umano,[3] Prade and Testemale,[4] and Zemankova.[5]

In Ref. 6 we present a model, named GEFRED, which attempts to synthesize the most outstanding aspects of previous approaches within a common theoretical framework.

Theoretical models of fuzzy databases require mechanisms to build relational systems (FRDBMS), which operate in accordance with these principles. Most relevant proposals have been provided by the proponents of different theoretical models.[3–9]

In Refs. 10–12, we present some ideas on how to represent fuzzy information, and how to implement it on a conventional RDBMS. This has allowed us to

*To whom correspondence should be addressed. E-mail: medina@decsai.ugr.es

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 12, 597–613 (1997)
 CCC 0884-8173/97/080597-17

formulate an implementation scheme that makes it possible to incorporate all these capacities into existing systems. On the other hand, it still lacks something, i.e., the possibility of defining information through rules. This could allow us to define even recursive-defined concepts, e.g., the well-known `ancestor` computation. In Ref. 13 we present a logic approach to fuzzy databases which allows us to deduce fuzzy (or not) information from a set of fuzzy rules using "*fuzzy facts.*" So, our interface is developed from these basic elements:

1. A suitable theoretical model of Fuzzy Relational Database.
2. A suitable theoretical model of Logic Fuzzy Database.
3. A suitable and general enough architecture for organizing the modules which form the whole system.
4. Operative criteria for representing fuzzy data and their manipulation operations.
5. The use of the resources provided by a conventional RDBMS to implement fuzzy handling.
6. The use of conventional deductive systems (Prolog in this case) to implement fuzzy deduction.

Therefore, what we intend is to be able to solve queries involving a deductive process which may not be solved by a relational system. In this case, it is necessary that our Deductive-Fuzzy Relational DBMS (DFRDBMS) must be able to distinguish between two types of relations:

- *Extensive Tables,* which are physically stored in the database or can be computed by relational methods (p.e. views).
- *Intensional Tables,* whose data must be computed by an independent deduction module.

To avoid inconsistency and redundancy between the relational and the logic systems, the information must be uniformly organized and as centralized as possible. In Section 4 we describe how all the information is expressed in relational terms.

To illustrate the problem we are tackling, let us consider a database with the following scheme:

PARENTS(Name,Father,Mother)
DEAD_PEOPLE(Name,Age,Date)
ALIVE_PEOPLE(Name,Address,)

and a query *"Get person names together with their ancestor names such that the ancestor died young".* In this query there are some interesting points:

- There is no explicit information about *ancestors* in the DB and this information is not computable in relational terms, but we have all the necessary information to know who the ancestors of a person are.
- There is an attribute value *young,* which implies a fuzzy treatment of information.

However, the intensional table *ancestors* can be defined by the following rule:

ANCESTOR(X,Y) if PARENTS(X,Y,_).
ANCESTOR(X,Y) if PARENTS(X,_,Y).
ANCESTOR(X,Y) if PARENTS(X,Z,_) and ANCESTOR(Z,Y).
ANCESTOR(X,Y) if PARENTS(X,_,Z) and ANCESTOR(Z,Y).

so, what we need first is a relational implementation for this rule, in order to represent it in the database. Once this representation is achieved, the DFRDBMS should build the rule into its logical format and send it to the deduction module. All this process will be amply explained in the following sections.

Once the deduction module has received the rule, it computes all possible solutions for X and Y variables and saves them in a table, called ANCESTOR, which is then used by the DFRDBMS to solve the rest of the query.

This paper is organized as follows: In Section 2, we summarize the theoretical models on which this implementation is based: A fuzzy relational database model and a logic fuzzy database model. In Section 3, the most important features of FREDDI architecture are shown. We give its modular description, paying special attention to the Deductive Fuzzy SQL Server. In Section 4, we explain in detail how imprecise and intensional information is implemented in the Deductive Fuzzy Relational DBMS (DFRDBMS) and give an example to illustrate it. Section 5 summarizes and classifies the most important contributions of this work and, finally, Section 6 points out the open problems and the future avenues for research.

2. THEORETICAL MODELS

In this section, we summarize the main theoretical models on which we have founded this work. On the one hand, the model for fuzzy relational databases and, on the other hand, the logic model for fuzzy databases.

2.1. Fuzzy Relational Database Model

In this section we introduce the basic elements of a Fuzzy extension of the Relational Model, called GEFRED, described in Ref. 6. This extension includes some additional elements to the previously reviewed Fuzzy Relational Models.

The main contributions are:

- Handling more kinds of imprecise information.
- A different way of organizing information. The same relational structure is used to represent the initial information, the information resulting from algebraic operations, the rules and the final results.
- The precision with which any simple condition involved in a query is satisfied can be controlled independently.

2.1.1. Data Structure

The information the model handles is organized as follows:

- The domain D_G underlying every attribute of the relation contains some of the data in Table I.

Table I. Types of data.

1.	A single scalar (Behavior = good, represented by the possibility distribution, 1/good)
2.	A single number (Age = 28, represented by the possibility distribution, 1/28)
3.	A set of possible scalar assignments (Behavior = {good, bad}, represented by {1/good, 1/bad})
4.	A set of possible numeric assignments (Age = {20, 21}, represented by {1/20, 1/21})
5.	A possibility distribution in a scalar domain (Behavior = {0.6/bad, 0.7/normal})
6.	A possibility distribution in a numeric domain (Age = {0../23, 1.0/24, 0.8/25}, fuzzy numbers or linguistic labels)
7.	A real number belonging to [0, 1], referring to degree of matching (Quality = 0.9)
8.	An **Unknown** value with possibility distribution, **Unknown** = $\{1/u : u \in U\}$
9.	An **Undefined** value with possibility distribution, **Undefined** = $\{0/u : u \in U\}$
10.	A **NULL** value given by **NULL** = {1/Unknown, 1/Undefined}

- We structure the data through a relation model, R_{FG}, given by:

$$R_{FG} \subset (D_{G1}, C_1), \times, \ldots, \times (D_{Gn}, C_n)$$

where every D_{Gj} is a domain of the type previously described, and C_j is a *''compatibility attribute''* that takes its values from [0, 1]. Every attribute is associated with a *''compatibility attribute.''* In base relations, *''compatibility attribute''* does not appear. This relation represents the initial information as well as that resulting from the Fuzzy Algebra operations. Handling these relations through Fuzzy Relational Algebra can modify, for every tuple, the compatibility attribute values.

2.1.2. Data Handling

The Fuzzy Algebra used in this model is an extension of classical relational algebra. In this extension specific comparison operators are used for handling fuzzy information.

Fuzzy querying receives special handling, based on the following points:

- *''Atomic Selection''* is a query, on a relation type R_{FG}, in which the satisfaction of a simple condition is sought.
- When an attribute, an operator and a fuzzy constant are involved in an *''Atomic Selection,''* such condition will be satisfied to a degree for every attribute value. This degree takes a value in [0, 1].
- In an *''Atomic Selection''* we can establish a threshold for the degree of satisfaction of a condition. Thanks to this threshold in the *''Atomic Selection,''* we can eliminate those tuples that do not satisfy the condition to a degree greater or equal to that of the threshold.
- The result of an *''Atomic Selection''* with a threshold for the degree is, once again, a relation of the type described in point 1 of this section. In this relation, the degree of satisfaction of a condition for every value of the attribute involved appears in the *compatibility attribute*.

Compound conditions are those obtained combining simple conditions through logic connectives (negation, conjunction, and disjunction). Compound conditions are solved as follows:

- From every simple condition we obtain the resulting relation by applying the *''Atomic Selection''* with a given threshold.
- For simple conditions connected with a conjunctive operator, we obtain the intersection of the relations derived from every condition. Then the values of the *''compatibility attribute''* associated with every attribute involved in the simple conditions are computed. Computing consists of giving a value to the compatibility attribute of every tuple in the intersection, i.e., equal to the minimum value in the respective initial simple conditions.
- For simple conditions connected with a disjunctive operator, we obtain the union of the relations derived for every condition and update the *compatibility attribute* with the maximum value.
- For a negated simple condition, we update the *compatibility attribute* value with the complement to 1 of the present value in every tuple.

2.2. Logic Fuzzy Database Model

In this section we summarize the most relevant elements of a Logic Fuzzy Database Model.[13,14] The data types considered are the same as those appearing in Table I. In this model, information is represented as follows:

- **Facts** or *Logic Database:* Predicates with the form of:

  Student('Perez','Young','Granada').

  (whose arguments correspond to NAME, AGE, and CITY attributes) where fuzzy arguments (such as 'Young') can appear.
- **Labels:** These labels correspond to both attribute fuzzy values and fuzzy values you can query the database about, although they do not correspond to a current attribute value.
  - *Labels* are stored with the format:

    LABEL(Age,'Middle',30,35,40,45).

    for a normalized trapezoidal label called *''Middle''* on domain *''Age''*.
  - *Possibility distributions* are stored in the following format.
    Suppose the possibility distribution A for a student's age is: $A = 15/1 + 16/1 + 17/0.9$ establishing that he/she is 15 with possibility 1, 16 with possibility 1 or 17 with possibility 0.9. Then, the facts involved are:

    STUDENT('Sanchez','A','Sevilla').
    POSS(Age,'A',15,1).
    POSS(Age,'A',16,1).
    POSS(Age,'A',17,0.9).

  All the predicates used in this module are defined and justified in Ref. 13.
- **Rules:** Used to deduce new information from that stored in the Logic Database. Rules are Prolog rules but a certainty degree can be attached to them. Therefore, we will impose on every rule that they have an output argument that will be computed in the body of the rule from the certainty values of all the rules activated in the process.

  For example, we can use a rule to define what a *difficult course* is, in the following way:

```
difficult-course(Course,Pos):-
        num_subj(Course,NumSubj),
        NumSubj>=4,
        num_cred(Course,NumCred),
        comp(cred,NumCred,''High'',Pos1),
        CD is 0.8, min(CD,Pos1,Pos).
```

The aim of the predicates involved in the definition is the following:

- Predicates `num_subj(Course,NumSubj)` and `NumSubj≥4` are used to compute the number of subjects in the course `Course` and this number is required to be greater than or equal to 4.
- Predicates `num_cred(Course,NumCred)` and `comp(cred,NumCred, ''High'',Pos1)` compute the number of credits of the considered course and compare it with "High". The result of this comparison is a certainty degree stored in `Pos1`.
- Finally, `CD=0.8` is the *"goodness"* of the definition, which combines the results of the previous predicates to compute the value of `Pos`.

- **Comparison Operators:** Which are predicates for defining all fuzzy comparison operators used to solve a query. In our case, we have defined the operators:
  - **Comp(X,Y,Z):** Computes the degree (Z) to which X and Y are equal, where X and Y can be fuzzy sets, crisp values or possibility distributions.
  - **Greater_Eq(X,Y,Z), Less_Eq(X,Y,Z), Greater(X,Y,X), Less(X,Y,Z):** Compute, respectively, to what extent X is {greater or equal, less or equal, strictly greater or strictly less} than Y and store that degree in Z.

  The expression used to compute all these operators need not be unique or concrete, but most coincide with the ones used by the DFRDBMS. It may also occur that different forms of comparison are used for different attributes. We have chosen a unique expression for the sake of simplicity. More details about fuzzy comparison operators can be found in Refs. 15 and 16.

To sum up, all the information to be handled must be expressed from the logical point of view, i.e., as facts and rules, which in general may be imprecise.

3. ARCHITECTURE OF FREDDI

Figure 1 shows the general scheme of FREDDI; it provides a deductive and fuzzy extension of a conventional RDBMS, adopting a modular structure which permits a Client/Server organization. The main element of FREDDI architecture is the RDBMS, which works as a host; around the RDBMS there are modules that perform the deductive and fuzzy information handling. Below, we describe the modules which compose FREDDI in terms of their functional character, structure and implementation.

3.1. Modular Description

- The **Host RDBMS** is the system used as the basis of an implementation of FREDDI. It provides the management resources for RDB and a group of basic tools for programming SQL applications in the host language (by an API or an Embedded SQL specification). The incorporated SQL syntax, it is provided with a deductive and fuzzy extension and is used in the implementation of the conversion routines of DFSQL into SQL sentences.
- **DB** stores all the extensive information, "fuzzy" or not, with a relational scheme.

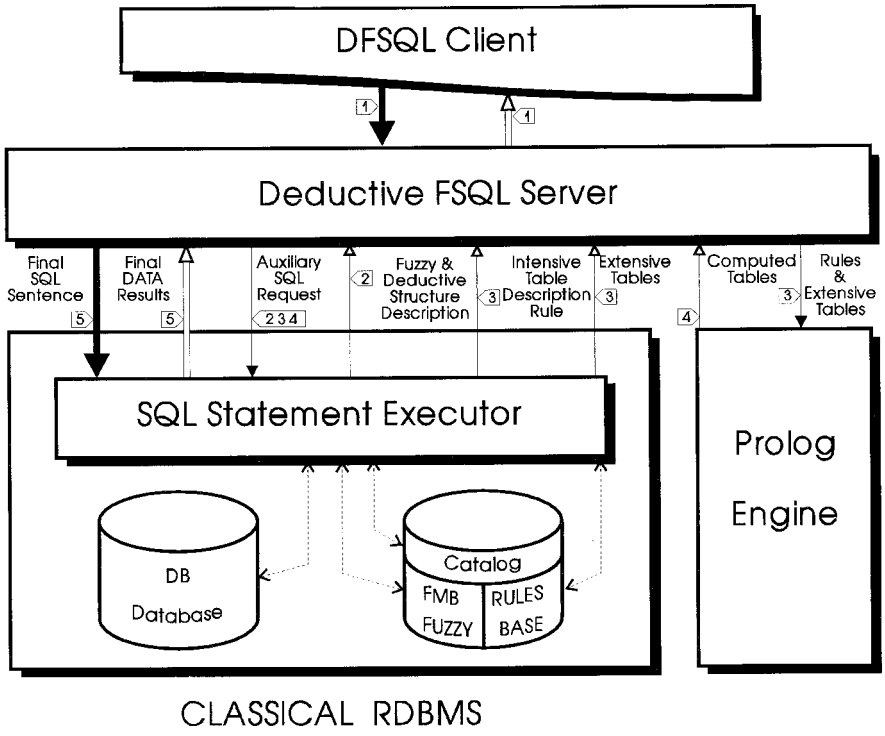

Figure 1. General scheme of FREDDI.

Certain representation criteria are used to implement the fuzzy data in this data scheme.

- **FMB,** the Fuzzy Meta-knowledge Base, is an extension of the host RDBMS catalog, whose mission is to provide FREDDI with the information about all fuzzy structures, data and definitions contained in the DB. FMB refers also to the intensional tables described in the DB. All the information is organized in accordance with the catalog scheme of the host RDBMS.
- **RB,** the Rules Base is another extension of the host RDBMS catalog, which contains all the information related to intensional tables defined in the DB. RB organization is shown in Section 4.3.
- **The Deductive FSQL Server** module is responsible for processing deductive and fuzzy requests to the system. To carry out this job, it is equipped with a parser for the Deductive FSQL (DFSQL) syntax. In Section 3.2 we describe its organization and functioning in depth.
- **The Prolog Engine,** which performs the inference processes which allow the intensional information requested to be obtained. The DFSQL Server has to determine which rules, extensive tables and definitions are to be sent to the Prolog Engine for it to solve these requests.

Among the previous modules, it is only left to implement the DFSQL Server, as the rest of them are based on components available at this moment. According to FREDDI architecture, the implementation of DFSQL Server can profit all

the advantages of the RDBMS Host on which it is built. Specifically, if the RDBMS Host supports procedural extensions of SQL, then the DFSQL Server can be built as a stored program executable in the RDBMS. This choice offers FREDDI implementation a better coupling with the Host RDBMS. Another important choice a RDBMS of these characteristics offers is the building of an ad hoc inference engine to process intensional tables without the use of an external prolog engine.

3.2. Deductive Fuzzy SQL Server

As we have seen in the previous section, the central module of FREDDI is the Deductive FSQL Server.

This module serves the requests of client applications expressed in DFSQL: it analyzes, verifies and processes them to return the results in a table. To carry out this task, it is supported by the rest of the modules in FREDDI architecture. In Figure 1 we show how this module performs a query.

1. It obtains a DFSQL request (black arrow 1) and separates those clauses involving fuzzy treatment or intensional table. To do this, it must explore the FMB through auxiliary SQL queries (black arrow 2 3 4).
2. It gets from the FMB the structure of the fuzzy clauses, from the Rules Base (RB) the rules that define the intensional tables and, from the DB, the extensive tables involved (white arrows 2 and 3, respectively).
3. It sends all the information involved in the deduction process to the Prolog Engine (black arrow 3), gets the results (white arrow 4) and updates the DB with these results (black arrow 2 3 4).
4. It translates fuzzy clauses into SQL clauses and constructs a definitive SQL sentence (black arrow 5).
5. It executes that sentence, obtains the results (white arrow 5) and processes them before sending them to the client application (white arrow 1).

3.2.1. DFSQL Syntax

As we have seen, FREDDI functioning is based on an extended version of SQL syntax provided by the Host RDBMS. Next, we show in YACC format some of the new clauses that DFSQL syntax incorporates.

- **Data Definition Language.** This sub-language incorporates into SQL new sentences to define tables containing fuzzy attributes and intensional tables. FREDDI supports intensional tables with both classical and fuzzy attributes. Among all the sentences needed, we only pay attention, for the sake of shortness, to the syntax adopted for the definition of these tables.

```
create_table   : CREATE INTENSIONAL TABLE table_item
                 '(' column_list ')' RULE rule_descpt ;

table_item     : id_user '.' id_table | id_table ;

column_list    : column_list ',' column_list | column_descpt ;
```

```
column_descpt  : column_id datatype ;

data_type      : classical_datatype | fuzzy_datatype ;

fuzzy_datatype : crisp | possibilistic | nearness ;

nearness       : SCALAR ')' NATURAL_NUMBER ')' | SCALAR ;

rule_descpt    : '(' rule_list ')' ;

rule_list      : rule_list ';' rule_list | rule_item ;

rule_item      : rule_item AND rule_item | predicate ;

predicate      : table_descpt | condition ;

table_descpt   : table_item '(' table_arg ')' ;

table_arg      : table_arg ',' table_arg | arg_item ;

arg_item       : '_' | var_id ;

var_id         : 'X'NATURAL_NUMBER ;

condition      : var_id comp var_id ;

comp           : '=' | '<' | '>' | '<=' | '=>' | '<>' ;
```

Nonterminal symbols `user_id, table_id` and `column_id` identify the object user, table and attribute, respectively, and follow the formation rules established in the syntax of the host RDBMS. Nonterminal symbols `crisp, possibilistic` and `nearness` identify fuzzy datatype for "fuzzy" attributes. The terminal symbol `NATURAL_NUMBER` identifies a datum with natural format; the semantics parser is responsible for verifying the range and type for each occurrence in the entry. The symbols in simple quotation marks, which belong to the production rule, are literals. The remaining terminal symbols are reserved words in the new syntax.

- **Data Manipulation Language.** In FREDDI, the query is the operation that involves all expressive power of DFSQL. Next, we detail the new clauses attached to SELECT in order to make queries which allow intensional and fuzzy information to be obtained.

```
fcond_simp   : fcond_wtout threshold | fcond_wtout ;

fcond_wtout  : column_item fcomp fuz_constant ;

threshold    : THOLD NUMBER | THOLD '$' ID ;

fuz_constant : '$' ID
             | NUMBER
             | '#' NUMBER
             | '[' NUMBER ',' NUMBER ']'
```

```
             | '$' '[' NUMBER ',' NUMBER ',' NUMBER ','
               NUMBER ']' ;

fcomp        : FEQ | FGT | FLT | FGEQ | FLEQ ;

comp_deg     : CDEG '(' fuz_arith_op ')' ;

agre_funct   : FMAX  '('  column_item  ')'  |  FMIN  '('
               column_item ')'
             | FSUM  '('  column_item  ')'  |  FAVG  '('
               column_item ')' ;

fuz_arith_op : fuz_arith_op '+' fuz_arith_op
             | fuz_arith_op '-' fuz_arith_op
             | fuz_arith_op '*' fuz_arith_op
             | fuz_arith_op '/' fuz_arith_op
             | '(' fuz_arith_op ')'
             | column_item
             | fuz_constant ;

column_item  : table_id   '.'   column_id   |   table_alias
               '.' column_id
             | column_id ;
```

The terminal symbol NUMBER identifies a datum with numerical format. The remaining terminal symbols are reserved words in the next syntax, which permit "fuzzy" conditions, modifiers and functions to be expressed.

The nonterminal symbol fcomp represents the set of fuzzy comparators in FSQL syntax. The complete syntax of an atomic fuzzy comparison takes the form:

⟨column_item⟩ fcomp ⟨fuz_constant⟩ [⟨threshold⟩]

where square brackets indicate that the threshold may or may not appear in the condition; in the latter case, the default threshold value is equal to 0.5 here.

Composite conditions are obtained by the use of NOT, AND and OR connectives, and fuzzy atomic conditions can be connected with other conditions which are not fuzzy.

The following sentences would be valid statements of the FSQL syntax adopted (assuming the existence of the tables and labels used):

```
CREATE INTENSIONAL TABLE ancestor (X char(30),
Y char(30)) RULE (
parents(X,Y,_);
parents(X,_,Y);
parents(X,Z,_) and ancestor(Z,Y);
parents(X,_,Z) and ancestor(Z,Y)).

SELECT FACOUNT,FAVG(sal),FMAX(sal),FMIN(sal) FROM
employees
GROUP BY age THOLD 0.8

SELECT a.emp#, name, education, CDEG(education) FROM
employees a, capacity b
```

Table II. Representation of Type 2 attributes.

Data Type	F_TYPE	F_1	F_2	F_3	F_4
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[A,B]	5	A	0	0	B
APPROX(d)	6	d-margin	margin	-margin	d + margin
FUZZY	7	$a - \alpha$	α	β	b + β

```
WHERE a.emp#=b.emp# AND education FEQ $graduate THOLD 0.6

SELECT emp#,dept#,job#,salary+commission,CDEG
(salary+commission)
FROM employees
WHERE salary FEQ $high AND commission FEQ $low THOLD 0.8
```

4. IMPLEMENTATION OF INTENSIONAL AND IMPRECISE INFORMATION IN THE DFRDBMS

This section describes how the resources proposed by the host RDBMS can be used to implement the intensional and fuzzy data and operations. The details of this process on fuzzy data can be found in Ref. 10.

Implementation of intensional and imprecise information is done considering three levels:

1. The Database level. As we are dealing with the representation of imprecise and intensional data, we must determine how we can store it. The data representation must therefore be extended to deal with this kind of information.
2. The System Catalog level. The DFRDBMS must contain information about the elements in the Database which involves imprecise and/or intensional data as well as their nature and representation.
3. The DFSQL Server level. The system possesses knowledge about the treatment of the available fuzzy operations and about the deduction process.

4.1. Imprecise Data in the Host Database

The representation used for the imprecise data allows us to distinguish three types of ''fuzzy'' attributes at the Database level:

Type 1 Attributes with *''crisp data''* having *linguistic labels* defined on them.
Type 2 Attributes with *''imprecise data on an ordered domain.''*
Type 3 Attributes with *''discrete domains with analogies.''*

Type 1 attributes do not need a specific implementation scheme in the DB of the host RDBMS. Tables II and III show how we use the classical relational scheme to represent the data in the **Type 2** and **Type 3** attributes, respectively.

Table III. Representation of Type 3 attributes.

Data Type	F_TYP	F_P1	F_1	F_P2	F_2	F_P3	F_3	. . .
UNKNOWN	0	NULL	NULL	NULL	NULL	NULL	NULL	. . .
UNDEFINED	1	NULL	NULL	NULL	NULL	NULL	NULL	. . .
NULL	2	NULL	NULL	NULL	NULL	NULL	NULL	. . .
SIMPLE	3	1	d	NULL	NULL	NULL	NULL	. . .
POS. DISTR.	4	p_1	d_1	p_2	d_2	p_3	d_3	. . .

For each type 2, an attribute F is created: an F_TYPE attribute with the type code and attributes F_1, F_2, F_3, F_4 representing the parameters for each data. NULL values appearing in the attributes have the meaning assigned by the host RDBMS. For the LABEL type, the FUZZY_ID code represents an identifier for the linguistic hedge defined in the Meta-knowledge Base. ''Margin'' is another parameter stored in the Meta-knowledge Base.

4.2. The Fuzzy Meta-Knowledge Base

We use the term **Fuzzy Meta-Knowledge Base** (FMB) to refer to that extension of the System Catalog which captures all the necessary information about imprecise data in the Database. Besides, it contains references to the intensional tables of the DB. Accordingly, we shall organize all this information in tables or relations. The elements stored in the Meta-knowledge Base are the following:

- The intensional tables present in the Database.
- The attributes in the Database that will be treated as imprecise.
- The type of these attributes. The table Fuzzy_col codifies the nature of an attribute with a value in the column column_type, according to the following table:

Attribute_Type	Column_type Value
Extensive Crisp	0
Extensive Possibilistic	1
Extensive Nearness	2
Intensional Crisp	4
Intensional Possibilistic	5
Intensional Nearness	6

- Elements defined in the Database scope, i.e., query fuzzy quantifiers.
- The fuzzy objects defined on each attribute:
  - Linguistic labels.
  - Approximate values.
  - Proximity relations.
  - Threshold labels.

The organization of the tables of FMB is shown in Figure 2.

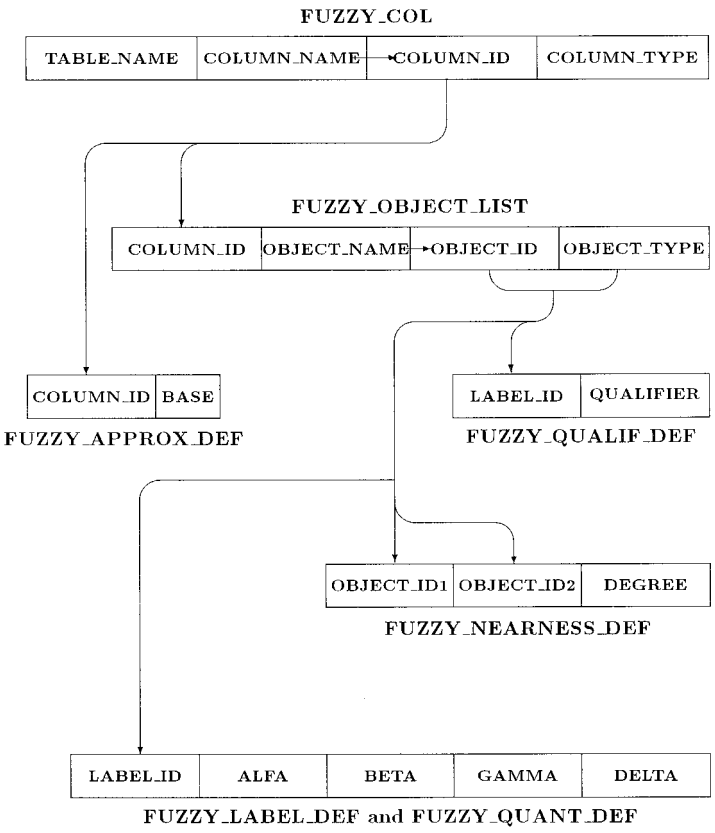

Figure 2. Meta-knowledge base scheme.

4.3. Rules Representation in the Host Database

To deal with the problem of representing rules in the RDBMS, the creation and manipulation of four tables are necessary. These tables are represented in Figure 3 and their descriptions are the following:

- **Intensional_Table_Description:** This table contains the name of the intensional table we are defining and the number of rules needed to be defined. The attributes are:
  - TABLE_ID: It contains the intensional table name or, what is the same, the head of the rules defining such table. The definition of this intensional table (at least, name and attribute names and types) must already be stored in the DB catalog.
  - RULE_ID: It contains a different identifier for each rule defining a concrete intensional table. These identifiers are automatically assigned by the system.
- **Rule_Description:** This table contains the description of each rule, i.e., the predicates involved. Its attributes are:
  - TABLE_ID, RULE_ID: These two attributes have the same meaning as the

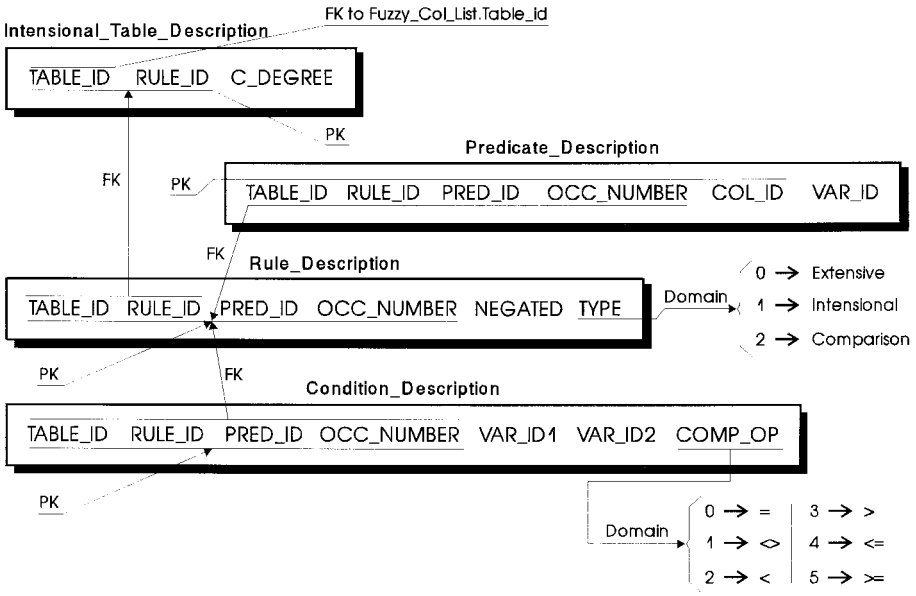

Figure 3. Relational representation of rules.

ones in the previous table and must coincide with them, i.e., both of them constitute a foreign key to the Intensional_Table_Description table.

- PRED_ID: This column contains the name of the predicates involved in a concrete rule.
- OCC_NUMBER: Should the same predicate name appear more than once in the body of a rule, we need to distinguish different occurrences by attaching a number in the column OCC_NUMBER.
- NEGATED: This column contains value 1 if the corresponding predicate occurrence is negated in the rule and 0, otherwise.
- TYPE: This column contains a value in the set $\{0, 1, 2\}$ with the following meaning:
  - $0 \rightarrow$ Extensive, i.e., the predicate name corresponds to a stored table of the database.
  - $1 \rightarrow$ Intensional, i.e., the predicate name corresponds to a rule that must be described in the Intensional_Table_Description table.
  - $2 \rightarrow$ Comparison, i.e., the expression is not a predicate but a comparison expression between the variables appearing in the rule.

- **Predicate Description.** This table contains the description of every predicate occurrence, i.e., the variables used, the order in which they appear, etc. . . . Its attributes are:
  - TABLE_ID, RULE_ID, PRED_ID and OCC_NUMBER: These columns must coincide with the corresponding ones in the table Rule_Description, i.e., they constitute a foreign key to this table.
  - VAR_ID: This column contains the identifiers of the variables that appear in a predicate expression. If the variables are called, p.e. X_i, this column will contain the number i.
  - COL_ID: This column indicates the position of every variable inside a predicate expression (whether X_i is in first, second, . . . place in the predicate P expression).
- **Condition_Description.** This table contains the description of comparison expressions appearing in the rules. Its attributes are:

Table IV. Intensional_Table_Description table.

TABLE_ID	RULE_ID
P	P_1
P	P_2

- TABLE_ID, RULE_ID, PRED_ID and OCC_NUMBER: These columns must coincide with the corresponding ones of table Rule_Description, i.e., they constitute a foreign key to this table.
- VAR_ID1 and VAR_ID2: Are the identifiers of the variables involved in a comparison operation.
- COMP_OP: This column indicates the comparison operator used in the expression. This column can contain one of the values from the set {0, 1, 2, 3, 4, 5} standing for a $=$, $\langle\rangle$, $\langle$, $\rangle$, $\langle=$ and $>=$, respectively.
  If instead of a classical comparison operator, a fuzzy one is needed (see fuzzy comparison predicates in Section 2.2), the predicate associated to this operator must be stored in the same way as any other predicate. It is not stored as a comparison expression because the result is not just *true* or *false,* but rather an accomplishment degree.

- **Example.** Let us see, by way of an example, how rules are represented in the DFRDBMS. Let us suppose an intensional table (predicate) definition like:

  $P(X_1, X_2, X_3)$ *if* $Q_1(X_1,\_)$ *and not* $(Q_2(X_2, X_3))$ *and* $(X_1 < X_3)$.
  $P(X_1, X_2, X_3)$ *if* $Q_1(X_1, X_4)$ *and* $P(X_4, X_2, X_1)$ *and* $Q_3(X_3,\_)$.

  with the following characteristics:
  - Predicate P has been defined using more than one rule (in fact, two rules).
  - In P definition extensive predicates Q_1, Q_2 and Q_3 appear, which correspond to tables of the DB.
  - In P definition, P itself appears, i.e., it is a recursive definition.
  - In P definition, a comparison expression is considered $(X_1 < X_3)$.

  The representation of the rules defining an intensional table P is shown in Tables IV through VII.

5. CONCLUSIONS

The main characteristics of the proposed architecture may be classified as follows:

- **Theoretical Contributions.** Integration of the two theoretical models of Fuzzy

Table V. Rule_Description table.

TABLE_ID	RULE_ID	PRED_ID	OCC_NUMBER	NEGATED	TYPE
P	P_1	Q_1	1	0	0
P	P_1	Q_2	1	1	0
P	P_1	comp1	1	0	2
P	P_2	Q_1	1	0	0
P	P_2	P	1	0	1
P	P_2	Q_3	1	0	0

Table VI. Predicate_Description table.

TABLE_ID	RULE_ID	PRED_ID	OCC_NUMBER	COL_ID	VAR_ID
P	P_1	Q_1	1	1	1
P	P_1	Q_2	1	1	2
P	P_1	Q_2	1	2	3
P	P_2	Q_1	1	1	1
P	P_2	Q_1	1	2	4
P	P_2	P	1	1	4
P	P_2	P	1	2	2
P	P_2	P	1	3	1

Relational DB: a logic model,[13] which allows us to represent, manipulate and infer imprecise data, and a generalized relational model GEFRED,[6] which extends the representation and manipulation mechanisms of the relational model for the treatment of fuzzy information.

This philosophy tries to take advantage, on the one hand, of the efficiency of the relational model with respect to the representation used and the extensive information searching and, on the other hand, of the logical model capacity to infer intensional information. For this reason, joining both models allows a balanced compromise to be reached between flexibility, in the treatment of all kinds of information, and processing speed.

- **Representation Criteria.** It adopts the classical relational scheme as representation tool for precise, fuzzy and intensional information.
- **Implementation Features.** We have proposed an open architecture, which can be implemented on commercial RDBMS and permits the use of different kinds of inference engines. This approach brings some benefits:
  - It reduces the time for the development of DFRDBMS prototypes.
  - It is easy to transport to a different environment.
  - It uses the resources offered by conventional RDBMS to implement all additional elements of FREDDI architecture.
  - It allows co-existence with classical DB and applications, preserving previous architecture and data.
  - It is easily integrated inside the actual processing schemes (Client/Server architecture, Distributed Databases, etc.).

6. FUTURE AVENUES FOR RESEARCH

Further work is needed on the following aspects of the proposed implementation model:

- The development of a complete FREDDI prototype which would incorporate all the modules.
- To increase the variety and applicability of implemented operators.
- The development of a complete DFSQL extension, including all operations of DDL, DML and DCL sublanguages.

Table VII. Condition_Description table.

TABLE_ID	RULE_ID	PRED_ID	OCC_NUMBER	VAR_ID1	VAR_ID2	COMP_OP
P	P_1	comp1	1	1	3	2

- To incorporate tools for the development of applications for Deductive Fuzzy Databases.
- To adapt the FREDDI specification to widely used conventional RDBMS.
- To study the viability of developments at low levels of implementations based on GEFRED.
- In the case of RDBMS with procedural extensions of SQL, the use of these characteristics to build an ad hoc inference engine that processes intensional queries without the need for an external Prolog module.

The adaptation of the theoretical models to the framework of object databases and their implementation on OODBMS, also merits further investigation.

References

1. B.P. Buckles and F.E. Petry, ''A fuzzy representation of data for relational databases,'' *Fuzzy Sets and Systems,* **7,** 213–226 (1982).
2. S. Shenoi and A. Melton, ''An extended version of the fuzzy relational database model,'' *Information Sci.,* **52,** 35–52 (1990).
3. M. Umano, ''Freedom—0: A fuzzy database system,'' *Fuzzy Information and Decision Processes,* Gupta-Sanchez, Ed. North-Holland, Amsterdam, 1982.
4. H. Prade and C. Testemale, ''Generalizing database relational algebra for the treatment of incomplete/uncertain information and vague queries,'' *Information Sci.,* **34,** 115–143 (1984).
5. M. Zemankova and A. Kandel, *Fuzzy Relational Data Bases—A Key to Expert Systems,* Verlag TUV, Rheinland, 1984.
6. J.M. Medina, O. Pons, and M.A. Vila, ''GEFRED. A Generalized Model of Fuzzy Relational Databases,'' *Information Sci.,* **76**(1–2), 87–109 (1994).
7. P. Bosc, M. Galibourg, and G. Hamon, ''Fuzzy querying with SQL: Extensions and implementation aspects,'' *Fuzzy Sets and Systems* **28,** 333–349 (1988).
8. D. Li and D. Liu, *A Fuzzy Prolog Database System,* Wiley, New York, 1990.
9. Nakajima Hiroshi, Sogoh Taiji, and Arao Masaki, ''Fuzzy database language and library—fuzzy extension to SQL,'' *Proceedings of Second IEEE International Conference on Fuzzy Systems,* 1993, pp. 477–482.
10. J.M. Medina, M.A. Vila, J.C. Cubero, and O. Pons, ''Towards the implementation of a generalized fuzzy relational database model,'' *Fuzzy Sets & Systems,* **75,** 273–289 (1995).
11. M.A. Vila, J.C. Cubero, J.M. Medina, and O. Pons, ''Some recent advances in fuzzy relational and fuzzy deductive databases,'' *European Research Consortium for Informatics and Mathematics,* Barcelona, 1–2 November, 1994, pp. 161–176.
12. M.A. Vila, J.C. Cubero, J.M. Medina, and O. Pons, ''Towards the computer implementation of a fuzzy relational and deductive database system,'' *Proceedings of the FUZZ-IEEE/IFES'95 workshop on Fuzzy Relational Systems and Information Retrieval,* Yokohama, Japan, March, 1995.
13. M.A. Vila, J.C. Cubero, J.M. Medina, and O. Pons, ''Logic and fuzzy relational databases: A new language and a new definition,'' in *Fuzzy Sets and Possibility Theory in Databases Management Systems,* P. Bosc and J. Kacprzyk Eds. Physica-Verlag, Heidelberg, 1994.
14. O. Pons, M.A. Vila, and J.M. Medina, ''Handling imprecise medical information in the framework of logic fuzzy databases,'' *Fuzzy Systems & A. I.,* **3**(1), 5–25. Ed. Academiei Romane, 1994.
15. K.M. Lee and H. Lee-Kwang, ''Fuzzy matching and fuzzy comparison in fuzzy expert systems, *on Fuzzy Logic and Neural Networks,* Iizuka, Japan, 1992, pp. 313–316.
16. R. Zwick, E. Carlstein, and D.V. Budescu, ''Measures of similarity among fuzzy concepts: A comparative analysis, *Int. J. Approximate Reasoning,* **1,** 221–242 (1987).