Handbook of Research on Fuzzy Information Processing in Databases

José Galindo

University of Málaga, Spain
Chapter XIV

How to Achieve Fuzzy Relational Databases Managing Fuzzy Data and Metadata

Mohamed Ali Ben Hassine
Tunis El Manar University, Tunisia

Amel Grissa Touzi
Tunis El Manar University, Tunisia

José Galindo
University of Málaga, Spain

Habib Ounelli
Tunis El Manar University, Tunisia

ABSTRACT

Fuzzy relational databases have been introduced to deal with uncertain or incomplete information demonstrating the efficiency of processing fuzzy queries. For these reasons, many organizations aim to integrate flexible querying to handle imprecise data or to use fuzzy data mining tools, minimizing the transformation costs. The best solution is to offer a smooth migration towards this technology. This chapter presents a migration approach from relational databases towards fuzzy relational databases. This migration is divided into three strategies. The first one, named “partial migration,” is useful basically to include fuzzy queries in classic databases without changing existing data. It needs some definitions (fuzzy metaknowledge) in order to treat fuzzy queries written in FSQL language (Fuzzy SQL). The second one, named “total migration,” offers in addition to the flexible querying, a real fuzzy database, with the possibility to store imprecise data. This strategy requires a modification of schemas, data, and eventually programs. The third strategy is a mixture of the previous strategies, generally as a temporary step, easier and faster than the total migration.

INTRODUCTION

New enterprise information systems are requested to be flexible and efficient in order to cope with rapidly changing business environments and advancement of services. An information system that develops its structure and functionality in a continuous, self-organized, adaptive, and interactive way can use many sources of incoming information and can perform intelligent tasks such as language
learning, reasoning with uncertainty, decision making, and more. According to Bellman and Zadeh (1970), “much of the decision making in the real world takes place in an environment in which the goals, the constraints, and the consequences of possible actions are not known precisely.” Management often makes decisions based on incomplete, vague, or uncertain information. In our context, the data which are processed by the application system and accumulated over the lifetime of the system may be inconsistent and may not express the reality. In fact, one of the features of human reasoning is that it may use imprecise or incomplete information and in the real world, there exists a lot of this kind of fuzzy information. Hence, we can assert that in our every day life we use several linguistic labels to express abstract concepts such as young, old, cold, hot, cheap, and so forth. Therefore, human–computer interfaces should be able to understand fuzzy information, which is very usual in many human applications. However, the majority of existing information systems deal with crisp data through crisp database systems (Elmasri & Navathe, 2006; Silberschatz, Korth, & Sudarshan, 2006). In this scenario, fuzzy techniques have proven to be successful principles for modeling such imprecise data and also for effective data retrieval. Accordingly, fuzzy databases (FDBs) have been introduced to deal with uncertain or incomplete information in many applications demonstrating the efficiency of processing fuzzy queries even in classical or regular databases. Besides, FDBs allow storing fuzzy values, and of course, they should allow fuzzy queries using fuzzy or nonfuzzy data (Bosc, 1999; De Caluwe & De Tré, 2007; Galindo, Urrutia, & Piattini, 2006; Petry, 1996).

Facing this situation, many organizations aim to integrate flexible querying to handle imprecise data or to use fuzzy data mining tools, minimizing the transformation costs. A solution of the existing (old) systems is the migration, that is, moving the applications and the database to a new platform and technologies. Migration of old systems, or legacy systems, may be an expensive and complex process. It allows legacy systems to be moved to new environments with the new business requirements, while retaining functionality and data of the original legacy systems. In this context, the migration towards FDBs, which constitutes a step to introduce imprecise data in an information system, does not only constitute the adoption of a new technology, but also, and especially, the adoption of a new paradigm. Consequently, it constitutes a new culture of development of information systems, and this book is evidence of the current interest and the promising future of this paradigm and its multiple fields.

However, with important amounts invested in the development of relational systems, in the enrollment and the formation of “traditional” programmers, and so forth, enterprises appear reticent to invest important sums in the mastery of a new fuzzy paradigm. The best solution is to offer a smooth migration toward this technology, allowing them to keep the existing data, schemas, and applications, while integrating the different fuzzy concepts to benefit of the fuzzy information processing. It will lower the costs of the transformations and will encourage the enterprises to adapt the concept of fuzzy relational databases (FRDBs). Moreover, although the migration of the information systems constitutes a very important research domain, there is a limited number of migration methods between two specific systems. We mention some examples (e.g., Behm, Geppert, & Dittrich, 1997; Henrard, Hick, Thiran, & Hainaut, 2002; Menhoudj & Oualiha, 1996). To our knowledge, the migration of relational databases (RDB) towards FRDB is not even studied.

FDBs allow storing fuzzy values and, besides, they allow making fuzzy queries using fuzzy or nonfuzzy data. It should be noted that classic querying is qualified by “Boolean querying,” although some systems use a trivalued logic with the three values true, false, and null, where null indicates that the condition result is unknown because some data is unknown. The user formulates a query usually with a condition, for example, in SQL, which returns a list of rows, when the condition is true. This querying system constitutes a hindrance for
several applications because we cannot know if one row satisfies the query better than another row. Besides, the traditional querying does not make it possible for the end user to use some vague linguistic terms in the query condition or to use fuzzy quantifier such as “almost all” or “approximately the half.” Many works have been proposed in the literature to introduce the flexibility into the database querying both in crisp and fuzzy databases (Bosc, Liétard, & Pivert, 1998; Bosc, & Pivert, 1995, 1997, 2000; Dubois & Prade, 1997; Galindo, Medina, & Aranda, 1999; Galindo, Medina, Pons, & Cubero, 1998; Galindo et al., 2006; Kacprzyk & Zadrozny, 1995, 2001; Tahani, 1977; Umano & Fukami, 1994). The essential idea in these works consists in adding an additional layer to the classic DBMS (database management systems) to evaluate fuzzy predicates. In this book, the reader can find a chapter by Zadrozny, de Tré, de Caluwe, and Kacprzyk with an interesting review about fuzzy querying proposals. Also, this book includes other chapters with new applications and new advances in the field of fuzzy queries. Some examples are the chapter by Takači and Škrbič about priorities in queries, the chapter by Dubois and Prade about bipolar queries, and the chapter by Barranco, Campana, and Medina using a fuzzy object-relational database model.

Among various published propositions for different fuzzy database models, we mention the one by Medina, Pons, and Vila (1995) who introduced the GEFRED model, an eclectic synthesis of other previous models. In 1995, Bosc and Pivert introduced the first version of a language handling the flexible queries named SQLf. In their turn, Medina, Pons, and Vila (1994b) proposed the FSQl language, which was later extended (Galindo, 1999, 2005; Galindo et al., 1998, 2006). Although the basic target in FSQl is similar to the SQLf language, FSQl allows fuzzy queries both in crisp and fuzzy databases and it presents new definitions such as many fuzzy comparators, fuzzy attributes (including fuzzy time), fuzzy constants. It allows the creation of new fuzzy objects such as labels, quantifiers, and so forth. There is another chapter by Urrutia, Tineo, and González studying both proposals.

This chapter presents a new approach for the migration from RDB towards FRDB with FSQL. The aim of this migration is to permit an easy mapping of the existing data, schemas, and programs, while integrating the different fuzzy concepts. Therefore, all valid SQL queries remain useful in the fuzzy query language FSQl (fuzzy SQL). This approach studies the RDB transformations essentially at the level of the schemas (physical and conceptual), the data, and, less specifically, the applications.

First, we present a very brief overview about fuzzy sets and then we present basic concepts about FRDB. After, we present our three migration strategies. The first one, named “partial migration,” is useful only to include fuzzy queries in classic databases without changing existing data. The second one, named “total migration,” offers in addition to the flexible querying the possibility to store imprecise data. The third strategy is a mixture of the previous strategies. Finally, we outline some conclusions and suggest some future research lines.

**Introduction to Fuzzy Sets**

The fuzzy sets theory stems from the classic theory of sets, adding a membership function to the set, which is defined in such a way that each element is assigned a real number between 0 and 1. In 1965, professor L.A. Zadeh defined the concept of fuzzy sets and then many works and applications have been made (Pedrycz & Gomide, 1998). We give here the most basic notions, and for a better introduction, read the first chapter of this handbook.

A fuzzy set (or fuzzy subset) $A$ is defined by means of a membership function $\mu_a (u)$, which indicates the degree to which the element $u$ is included in the concept represented by $A$. The fuzzy set $A$ over a universe of discourse $U$ can also be represented with a set of pairs given by:

$$A = \{(\mu_a (u) /u : u \in U, \mu_a (u) \in [0,1])\}$$  (1)
where \( \mu \) is the membership function and \( \mu_{A}(u) \) is the membership degree of the element \( u \) to the fuzzy set \( A \). If \( \mu_{A}(u) = 0 \), it indicates that \( u \) in no way belongs to the fuzzy set \( A \). If \( \mu_{A}(u) = 1 \), then \( u \) belongs totally to the fuzzy set \( A \). For example, if we consider the linguistic variable \( \text{height_of_a_person} \), then three fuzzy subsets could be defined identified by three labels, \( \text{Short} \), \( \text{Medium-height} \), and \( \text{Tall} \), with membership functions \( \mu_{\text{Short}}(u) \), \( \mu_{\text{Medium-height}}(u) \), and \( \mu_{\text{Tall}}(u) \), respectively, where \( u \) takes values in the referential of this attribute (or underlying domain), that would be the real positive numbers (expressing the centimetres of height).

On the other hand, for domains with a non-ordered referential, a similarity function can be defined, that can be used to measure the similarity or resemblance between every two elements of the domain. Usually, the similarity values are normalized in the interval \([0,1]\), where 0 means “totally different” and 1 means “totally alike” or equal. Thus, a similarity relationship is a fuzzy relation that can be seen as a function \( s_r \), so that:

\[
s_r : D \times D \rightarrow [0,1] \quad s_r(d_i, d_j) \rightarrow [0,1] \quad \text{with } d_i, d_j \in D
\]

(2)

where \( D \) is the domain of the defined labels. We can assume that \( s_r \) is a symmetrical function, this is that \( s_r(d_i, d_j) = s_r(d_j, d_i) \), as this is the most usual, although it does not necessarily have to be this way.

We can also construct possibility distributions (or fuzzy sets) on the labels of \( D \), extending the possibilities for expressing imprecise values (Zadeh, 1978), in such a way that each value \( d_i \in D \) has a degree of truth or possibility \( p_i \) associated to it, obtaining expressions for specific values that can be expressed generically as:

\[
\{ p_i/d_i : p_i \in [0,1], d_i \in D \}
\]

(3)

The domains with ordered and non-ordered referentials can adequately represent concepts of “imprecision” using fuzzy sets theory. It should be noted that many of these natural concepts depend, in a greater or lesser degree, on the context and on the person that expresses them.

From this simple concept, a complete mathematical and computing theory has been developed which facilitates the solution of certain problems. Fuzzy logic has been applied to a multitude of objectives such as control systems, modeling, simulation, patterns recognition, information or knowledge systems (databases, expert systems, etc.), computer vision, artificial intelligence, artificial life, and so forth.

**Introduction to Fuzzy Relational Databases**

The first chapter of this handbook includes a brief introduction to this topic, explaining some basic models. We give here a brief overview in order to facilitate the reading of this chapter.

The term “imprecision encapsulates various meanings, which might be interesting to highlight. It alludes to the facts that the information available can be incomplete (vague), that we don’t know whether the information is true (uncertainty), that we are totally unaware of the information (unknown), or that such information is not applicable to a given entity (undefined). Usually, the total ignorance is represented with a NULL value. Sometimes these meanings are not disjunctive and can be combined in certain types of information” (Galindo et al., 2006, p. 45).

This imprecision was studied in order to elaborate systems, databases, and, consequently, applications which support this kind of information. Most works studying the imprecision in information have used possibility, similarity, and fuzzy techniques. The research on FDBs has been developed for about 20 years and concentrated mainly on the following areas: flexible querying in classical databases, extending classical data models in order to achieve fuzzy databases (including, of course, fuzzy queries on these fuzzy databases and fuzzy conceptual modeling tools), fuzzy data mining.
How to Achieve Fuzzy Relational Databases Managing Fuzzy Data and Metadata

techniques, and applications of these advances in real databases. All these different issues have been studied in different chapters of this volume and also in many other publications (De Caluwe & De Tré, 2007; Bosc, 1999; Bosc et al., 1998; Galindo et al., 2006; Petry, 1996).

The querying of a FRDB, contrary to classical querying, allows the users to use fuzzy linguistic labels (also named linguistic terms) and express their preferences to better qualify the data that they wish to get. An example of flexible query, also named in this context fuzzy query, would be “list of the young employees, well paid and working in department with big budget.” This query contains the fuzzy linguistic labels “young,” “well paid,” and “big budget.” These labels are words, in natural language, that express or identify a fuzzy set that may or may not be formally defined.

In fact, the flexibility of a query reflects the preferences of the end user. This is manifested by using a fuzzy set representation to express a flexible selection criterion. The extent to which an object in the database satisfies a request then becomes a matter of degree. The end user provides a set of attribute values (fuzzy labels), which are fully acceptable for the user, and a list of minimum thresholds for each of these attributes. With these elements, a fuzzy condition is built for the fuzzy query. Then, the fuzzy querying system ranks the answered items according to their fulfillment degree or level of acceptability. Some approaches, the so-called bipolar queries, need both the fuzzy condition (or fuzzy constraint) and the positive preferences or wishes, which are less compulsory. (A very interesting chapter about bipolar queries may be found in this volume in the chapter by Dubois and Prade.) Hence, the interests of fuzzy queries for a user are twofold:

1. A better representation of the user’s preferences while allowing the use of imprecise predicates.
2. Obtaining the necessary information in order to rank the answers contained in the database according to the degree to which they satisfy the query. It contributes to avoid empty sets of answers when the queries are too restrictive, as well as too large sets of answers without any ordering when queries are too permissive.

This preface led us to establish the definition of FRDB as an extension of RDB. This extension introduces fuzzy predicates or fuzzy conditions under shapes of linguistic expressions that, in flexible querying, permits to have a range of answers (each one with its membership degree) in order to offer to the user all intermediate variations between the completely satisfactory answers and those completely dissatisfactory (Bosc et al., 1998). Yoshikane Takahashi (1993, p. 122) defined FRDB as “an enhanced RDB that allows fuzzy attribute values and fuzzy truth values; both of these are expressed as fuzzy sets”.

Then, a fuzzy database is a database which is able to deal with uncertain or incomplete information using fuzzy logic. There are many forms of adding flexibility in fuzzy databases. The simplest technique is to add a fuzzy membership degree to each record, an attribute in the range [0,1]. However, there are others kind of databases allowing fuzzy values to be stored in a fuzzy attribute using fuzzy sets or possibility distributions or fuzzy degrees associated to some attributes and with different meanings (membership degree, importance degree, fulfillment degree...). The main models are those of Prade-Testemale (1987), Umano-Fukami (Umano, 1982; Umano & Fukami, 1994), Buckles-Petry (1982), Zemankova-Kaendel (1985) and GEFRED by Medina-Pons-Vila (1994a).

This chapter deals mainly, with the GEFRED model (GÉneralised model for Fuzzy RElational Database), and some later extensions (Galindo et al., 2006). This model constitutes an eclectic synthesis of the various models published so far with the aim of dealing with the problem of representation and treatment of fuzzy information by using RDB. One of the major advantages of this model is that it consists of a general abstraction that allows for the use of various approaches,
How to Achieve Fuzzy Relational Databases Managing Fuzzy Data and Metadata

regardless of how different they might look. In fact, it is based on the generalized fuzzy domain and the generalized fuzzy relation, which include respectively classic domains and classic relations. The original data types supported by this model are showed on Table 1.

### PRELIMINARY CONCEPTS

In order to implement a system which represents and manipulates “imprecise” information, Medina et al. (1995) developed the FIRST (fuzzy interface for relational systems) architecture, which has been enhanced with FIRST-2 (Galindo, Urrutia, & Piattini, 2004b; 2006). It has been built on some DBMS client-server architecture, such as Oracle and PostgreSQL (Galindo, 2007; Maraboli & Abarzua, 2006). It extends the existing structure and adds new components to handle fuzzy information. This architecture adds a server, named FSQL server, assuring the translation of flexible queries written in FSQL in a comprehensible language for the host DBMS (SQL). FSQL is an extension of the popular SQL language, in order to express fuzzy characteristics, especially in fuzzy queries, with many fuzzy concepts (fuzzy conditions, fuzzy comparators, fulfillment degrees, fuzzy constants, fuzzy quantifiers, fuzzy attributes, etc.). The first versions of FSQL were developed during the last decade of the 20th century (Galindo et al., 1998; Medina et al., 1994b), and the more recent version is defined by Galindo et al. (2006).

In the following subsections, we present this language and the supported fuzzy attributes types. The RDBMS (relational DBMS) dictionary or catalog which represents the part of the system allowing the storage of information about the data collected in the database, and other information (such as users, data structures, data control, etc.), is prolonged in order to collect the necessary information related to the imprecise nature of the new collection of data processing (fuzzy attributes, their type, their objects such as labels, quantifiers, etc.). This extension, named fuzzy metaknowledge base (FMB), is organized following the prevailing philosophy in the host RDBMS catalog. In this chapter, we designate by fuzzy RDBMS (FRD-BMS) the addition of the FSQL server and the FIRST-2 methodology to the RDBMS.

### Fuzzy Attributes

In order to model fuzzy attributes, we distinguish between two classes of fuzzy attributes: Fuzzy attributes whose fuzzy values are fuzzy sets (or possibility distributions) and fuzzy attributes whose values are fuzzy degrees. Each class includes some

---

**Table 1. Data types in the GEFRED model**

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>A single scalar (e.g., Behavior=Good, represented by the possibility of distribution 1/Good).</td>
</tr>
<tr>
<td>2.</td>
<td>A single number (e.g., Age=28, represented by the possibility of distribution 1/28).</td>
</tr>
<tr>
<td>3.</td>
<td>A set of mutually exclusive possible scalar assignations (e.g., Behavior={Bad, Good}, represented by {1/Bad, 1/Good}).</td>
</tr>
<tr>
<td>4.</td>
<td>A set of mutually exclusive possible numeric assignations (e.g., Age={20, 21}, represented by {1/20, 1/21}).</td>
</tr>
<tr>
<td>5.</td>
<td>A possibility distribution in a scalar domain (e.g., Behavior={0.6/Bad, 1.0/Regular}).</td>
</tr>
<tr>
<td>6.</td>
<td>A possibility distribution in a numeric domain (e.g., Age={0.4/23, 1.0/24, 0.8/25}, fuzzy numbers or linguistic labels).</td>
</tr>
<tr>
<td>7.</td>
<td>A real number belonging to [0, 1], referring to a degree of matching (e.g., Quality=0.9).</td>
</tr>
<tr>
<td>8.</td>
<td>UNKNOWN value with possibility distribution {1/u: u ÎU}, where U is the considered domain.</td>
</tr>
<tr>
<td>9.</td>
<td>UNDEFINED value with possibility distribution {0/u: u ÎU}, where U is the considered domain.</td>
</tr>
<tr>
<td>10.</td>
<td>NULL value, given by NULL={1/Unknown, 1/Undefined}.</td>
</tr>
</tbody>
</table>
different fuzzy data type (Galindo et al., 2006; Urrutia, Galindo, & Piattini, 2002).

**Fuzzy Sets as Fuzzy Values**

These fuzzy attributes may be classified in four data types. This classification is performed taking into account the type of referential or underlying domain. In all of them, the values Unknown, Undefined, and Null are included:

- **Fuzzy Attributes Type 1 (FTYPE1):** These are attributes with "precise data," classic or crisp (traditional with no imprecision). However, we can define linguistic labels over them, and we can use them in fuzzy queries. This type of attribute is represented in the same way as precise data, but they can be transformed or manipulated using fuzzy conditions. This type is useful for extending a traditional database, allowing fuzzy queries to be made about classic data. For example, enquiries of the kind “Give me employees that earn a lot more than the minimum salary.”

- **Fuzzy Attributes Type 2 (FTYPE2):** These are attributes that gather “imprecise data over an ordered referential.” These attributes admit, like Table 2 shows, both crisp and fuzzy data, in the form of possibility distributions over an underlying ordered dominion (fuzzy sets). It is an extension of Type 1 that does, now, allow the storage of imprecise information, such as “he is approximately 2 metres tall.” For the sake of simplicity, the most complex of these fuzzy sets are supposed to be trapezoidal functions (Figure 1).

- **Fuzzy Attributes Type 3 (FTYPE3):** They are attributes over “data of discrete non-ordered dominion with analogy.” In these attributes, some labels are defined (e.g., “blond,” “red,” “brown,” etc.) that are scalars with a similarity (or proximity) relationship defined over them, so that this relationship indicates to what extent each pair of labels resemble each other. They also allow possibility distributions (or fuzzy sets) over this dominion, for example, the value (1/dark, 0.4/brown), which expresses that a certain person is more likely to be dark than brown-haired. Note that the underlying domain of these fuzzy sets is the set of labels, and this set is non-ordered.

- **Fuzzy Attributes Type 4 (FTYPE4):** These attributes are defined in the same way as Type 3 attributes without it being necessary for a similarity relationship to exist between the labels.

**Fuzzy Degrees as Fuzzy Values**

The domain of these degrees can be found in the interval [0,1], although other values are also permitted, such as a possibility distribution (usually over this unit interval). The meaning of these degrees is varied and depends on their use. The processing of the data will be different depending on the meaning. The most important possible meanings of the degrees used by some authors are the fulfillment degree, uncertainty degree, possibility degree, and importance degree.

The most typical kind of degree is a degree associated to each tuple in a relation (Type 7) with the meaning of membership degree of each tuple to the relation. Another typical degree is the fulfillment degree associated to each tuple in the resulting relation after a fuzzy query. In this volume, there are some chapters about these kinds of relations (see for example the ranked tables in the chapter by Belohlavek and Vychodil or the fulfillment degrees in the chapter by Voglozin, Raschia, Ughetto and Mouaddib).
Sometimes it is useful to associate a fuzzy degree to only one attribute (Type 5) or to only a concrete set of attributes (Type 6), for example, in order to measure the truth, the importance, or the vagueness. Finally, in some applications, a fuzzy degree with its own fuzzy meaning (Type 8) is useful in order to measure a fuzzy characteristic of each item in the relation like the danger in a medicine or the brightness of a concrete material.

Representation of Fuzzy Attributes

The representation is different according to the fuzzy attribute type. Fuzzy attributes Type 1 are represented as usual attributes because they do not allow fuzzy values. Fuzzy attributes Type 2 need five (or more) classic attributes: One stores the kind of value (Table 2), and the other four store the crisp values representing the fuzzy value. Note in Table 2 that trapezoidal fuzzy values (Figure 1) need the other four values. An approximate value (approximately \(d, d±\text{margin}\)) is represented with a triangular function centered in \(d\) (degree 1) and with degree 0 in \(d–\text{margin}\) and \(d+\text{margin}\), where the value margin depends on the context, as we will see later). Other approximate values (number 8) use their own margin \(m\).

Finally, we can also represent possibility distributions in the Type 2 attributes. Some of them (number 9 and 10) use only the four attributes defined previously, but we define here two new and more flexible possibilities:

- **Number 11:** Discontinuous possibility distribution, given a list of point with the format \(p_1/v_1, \ldots, p_n/v_n\), where the \(p_i\) are the possibility degrees and the \(v_i\) are the values with such degrees. Note that we need \(2n\) attributes (instead of the four) for storing a possibility distribution with \(n\) terms. The rest of the values have a degree of zero.

- **Number 12:** Continuous possibility distribution, given a list of point with the format \(p_1/v_1, \ldots, p_n/v_n\), where the \(p_i\) are the possibility degrees and the \(v_i\) are the values with such degrees. Note that we need \(2n\) attributes for storing a possibility distribution with \(n\) terms. Now the stored possibility distribution represents a continuous linear function, and between \(v_i\) and \(v_{i+1}\), there is a straight line joining each consecutive two points.

Fuzzy attributes Type 3 need \(2n+1\) attributes: One stores the kind of value (Table 3) and the others \((2n)\) may store a possibility distribution where \(n\) is the maximum number of elements (degree/label). Note in Table 3 that number 3 needs only two values, but number 4 needs \(2n\) values. Value \(n\) must be defined for each fuzzy attribute Type 3, and it is stored in the FMB (see following section).

Fuzzy attributes Type 4 are represented just like Type 3. The difference between them is shown in the next section. Fuzzy degrees (Types 5, 6, 7, and 8) are represented using a classic numeric attribute because their domain is the interval \([0,1]\).
The FSQL Language

The FSQL language (Galindo, 2005; Galindo et al., 2006; Galindo, Aranda, Caro, Guevara, & Aguayo, 2002) is an authentic extension of SQL which allows fuzzy data manipulation like fuzzy queries. It means that all the valid statements in SQL are also valid in FSQL. In addition, FSQL incorporates some novelties to permit the inexact processing of information. This chapter will only provide a summary of the main extensions added to this language:

- **Linguistic labels:** If an attribute is capable of fuzzy treatment, then linguistic labels can be defined on it. These labels will be preceded with the symbol $ to distinguish them easily. There are two types of labels, and they will be used in different fuzzy attribute types:
  
  1. Labels for attributes with an ordered underlined domain (Fuzzy Attributes Type 1 and 2): every label of this type has associated a trapezoidal possibility distribution in the FMB. This possibility distribution is generally trapezoidal, linear, and normalized, as shown in Figure 1.
  
  2. Labels for attributes with a non-ordered fuzzy domain (Fuzzy Attributes Type 3 and 4). Here, a similarity relation may be defined between each two labels in the domain, and it should be stored in the FMB.

- **Fuzzy comparators:** Besides the typical comparators (=, >, etc.), FSQL includes all the fuzzy comparators shown in Table 4. As in SQL, fuzzy comparators compare one column with one constant or two columns of the same (or compatible) type. As possibility comparators are more general (less restrictive) than necessity comparators, necessity comparators retrieve fewer tuples, and these tuples necessarily comply with the conditions (whereas with possibility comparators, the tuples only possibly comply with the condition, without any absolute certainty). Is it necessary to note that fuzzy attributes Type 2 can be compared with crisp values but always with the FSQL language.

- **Function CDEG:** The function CDEG (compatibility degree) may be used with an attribute in the argument. It computes the fulfillment degree of the condition of the query for the specific attribute in the argument. We can use CDEG(*) to obtain the fulfillment degree of each tuple (with all of its attributes, not just one of them) in the condition. If logic operators (NOT, AND, OR) appear in the condition, the calculation of this compatibility degree is carried out, by default, using the traditional negation, the minimum t-norm,

### Table 4. Fuzzy comparators for FSQL (Fuzzy SQL), 16 in the Possibility/Necessity Family, and 2 in the Inclusion Family

<table>
<thead>
<tr>
<th>Possibility</th>
<th>Necessity</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEQ or F=</td>
<td>NFEQ or NF=</td>
<td>Possibly/Necessarily Fuzzy Equal than…</td>
</tr>
<tr>
<td>FDIF, F!# or F&lt;&gt;</td>
<td>NFDF, NF!# or NF&lt;&gt;</td>
<td>Possibly/Necessarily Fuzzy Different to…</td>
</tr>
<tr>
<td>FG&gt; or F&gt;</td>
<td>NFGT or NF&gt;</td>
<td>Possibly/Necessarily Fuzzy Greater than…</td>
</tr>
<tr>
<td>FGEQ or F&gt;=</td>
<td>NFGEQ or NF&gt;=</td>
<td>Possibly/Necessarily Fuzzy Greater or Equal than…</td>
</tr>
<tr>
<td>FLT or F&lt;</td>
<td>NFLT or NF&lt;</td>
<td>Possibly/Necessarily Fuzzy Less than…</td>
</tr>
<tr>
<td>FLEQ or F&lt;=</td>
<td>NFLEQ or NF&lt;=</td>
<td>Possibly/Necessarily Fuzzy Less or Equal than…</td>
</tr>
<tr>
<td>MGT or F&gt;&gt;</td>
<td>NMGT or NF&gt;&gt;</td>
<td>Possibly/Necessarily Much Greater than…</td>
</tr>
<tr>
<td>MLT or F&lt;&lt;</td>
<td>NMLT or NF&lt;&lt;</td>
<td>Possibly/Necessarily Much Less than…</td>
</tr>
<tr>
<td>FINCL</td>
<td>INCL</td>
<td>Fuzzy Included in… / Included in…</td>
</tr>
</tbody>
</table>
and the maximum s-norm (or t-conorm), but the user may change these values.

- **Fulfillment thresholds**: For each simple condition, a fulfillment threshold \( \tau \) may be established (default is 1) with the format: \(<\text{condition}> \text{THOLD} \ \tau \) indicating that the condition must be satisfied with minimum degree \( \tau \in [0,1] \) to be considered. The reserved word \text{THOLD} (threshold) is optional and may be substituted by a traditional crisp comparator (\(=, <, >, \geq, \text{etc.}\)), modifying the query meaning (e.g., to retrieve the less interesting items).

- **Fuzzy constants**: Besides the typical constants (numbers, NULL, etc.), FSQL included many constants such as fuzzy trapezoidal \([a,b,c,d]\), approximate values using the expression \#n (approximately \(n\)), fuzzy predefined labels using \$\text{LabelName}\), crisp intervals with \([n,m]\), \text{UNKNOWN}, \text{UNDEFINED}, NULL, and so forth.

- **Fuzzy quantifiers**: There are two types: absolute and relative. They allow us to use expressions like “most,” “almost all,” “many,” “very few,” and so forth.

**Example 1**: “Give me all persons with fair hair (in minimum degree 0.5) that are possibly taller than label $Tall$ (with a high degree as qualifier)”:

```sql
SELECT * FROM Person
WHERE Hair FEQ $Fair THOLD 0.5
AND Height FGT $Tall THOLD $high;
```

The FuzzyEER Model

A database design methodology has three phases: conceptual design, logical design, and physical design. This study looks at conceptual design, which is the first phase during which the analysis of the database requirements takes place. The base requirements are independent of the data model that we use. For the conceptual design, the entity/relationship model (ER model) or the enhanced ER model (EER) are usually used. Originally, the conceptual level allows the use of elementary types of data which are called classical or crisp. These data types include numerical, alphanumerical, and binary data. However, the conceptual model does not always include these data types because they are usually not very important, so their definition is normally included in the data dictionary model.

On the other hand, several works have been proposed in the literature to introduce the fuzzy concepts in database modeling. The conceptual modeling tool used in this work is the fuzzy enhanced entity relationship (FuzzyEER) model (Galindo, Urrutia, Carrasco, & Piattini, 2004c; Galindo, Urrutia, & Piattini, 2004a, 2006; Urrutia et al., 2002). This model extends the enhanced entity relationship (EER) model with fuzzy semantics and fuzzy notations to represent imprecision and uncertainty in the entities, attributes, and relationships using fuzzy sets and necessity-possibility measures. The basic concepts introduced in this model are fuzzy attributes, fuzzy entities, fuzzy relations, fuzzy degrees, and fuzzy constraints to mention those only here. We present in Example 5 (Figure 9) a simplified FuzzyEER conceptual schema. The book *Fuzzy Databases: Modeling, Design and Implementation* (Galindo et al., 2006) presents more details about this model.

**A MIGRATION APPROACH TOWARDS FRDBs**

Designing a system that is able to make use of quantitative and qualitative data for real-world applications is a challenging problem. Traditional systems produce representational descriptions that are often not very useful to the human expert. Using classic logic, it is possible to deal only with information that is totally true or totally false; it is not possible to handle information inherent to a problem that is imprecise or incomplete, but this type of information contains data that would allow a better solution to the problem.

In the section titled Introduction to Fuzzy Sets, we saw that fuzzy logic is an extension of
How to Achieve Fuzzy Relational Databases Managing Fuzzy Data and Metadata

the classic systems (Zadeh, 1992). Fuzzy logic is the logic behind approximate reasoning instead of exact reasoning. Its importance lies in the fact that many types of human reasoning, particularly the reasoning based on expert knowledge, are by nature approximate. Note the great potential that the use of membership degrees represents by allowing something qualitative (fuzzy) to be expressed quantitatively. Besides, a better communication can be attained through fuzzy logic because of its ability to utilize natural languages in the form of linguistic variables (Zadeh, 1975, 1983).

Closer to our context, database technology has an extremely successful track record as a backbone of information technology throughout the last three decades. To introduce imprecise data, global information should be managed as fuzzy. The best solution is to offer a smooth migration toward this technology. The migration towards FDBs, or fuzzy migration, does not only constitute the adoption of a new technology but also, and especially, the adoption of a new paradigm. Consequently, it constitutes a new culture of development of information systems. In fact, the fuzzy migration of information systems consists in modifying or replacing one or more of their components: database, architecture, interfaces, applications, and so forth, and generally the modification of one of these components can generate modifications of some others.

This chapter is about the migration from crisp databases (relational) towards FDBs in order to introduce imprecise information in current information systems. This fuzzy migration consists in deriving a new database from a legacy database and in adapting data, metadata, and the software components accordingly. This migration, due generally to the apparition of new needs in the enterprise, must answer these requirements while maintaining the content of the information unaltered. Once the ex-database is emigrated, the ex-programs must be changed in such a manner that they reach the new database instead of the ex-data.

The definition of the fuzzy migration concept cited above may involve several problems such as:

• The schemas modification requires a very detailed knowledge on the data organization (data types, constraints, etc.).
• The database source is generally badly documented.
• The difficulty of correspondences establishment between the two databases.
• The database models, source, and target can be incompatible.
• The values in the FMB (metadata) must be chosen after thorough studies.
• The communication protocols between the database and their applications are generally hidden.
• The administrator and at least some database users need some knowledge about fuzzy logic.
• Software using fuzzy information must be designed with care, especially if it will be utilized by regular users.

Related Work

Although the information systems migration constitutes a very important research domain, there is a limited number of migration methods. For example, Tilley and Smith (1996) discuss current issues and trends in legacy system re-engineering from several perspectives (engineering, system, software, managerial, evolutionary, and maintenance). The authors propose a framework to place re-engineering in the context of evolutionary systems. The butterfly methodology (Wu, Lawless, Bisbal, Richardson, Grimson, Wade, & O’Sullivan, 1997) provides a migration methodology and a generic toolkit for the methodology to aid engineers in the process of migrating legacy systems. Different from the incremental strategy, this methodology eliminates the need of interoperability between the legacy and target systems.

Closer to our subject, the Varlet project (Jahnke & Wadsack, 1999) adopts a process that consists of two phases. In the first one, the different parts of the original database are analyzed to obtain a logical schema for the implemented physical schema. In the second phase, this logical schema
is transformed into a conceptual one, which is the basis for modification or migration activities. The approach of Jeusfeld and Johnen (1994) is divided into three parts: mapping of the original schema into a metamodel, rearrangement of the intermediate representation, and production of the target schema. Some works also address the migration between two specific systems. Among those, Menhoudj and Ou-Halima (1996) present a method to migrate the data of a legacy system into a RDB management system. Behm et al. (1997) describe a general approach to migrate RDBs to object technology. Henrard et al. (2002) and Cleve (2004) present strategies to migrate date-intensive applications from a legacy data management system to a modern one and take the example of conversion of COBOL files into a SQL database.

In the context of fuzzy databases, few fuzzy databases implementations have been developed in real and running systems (Galindo et al., 2006; Goncalves & Tineo, 2006; Kacprzyk & Zadrozny, 1995, 1999, 2000, 2001; Tineo, 2000). More information about these approaches is available in this handbook in the chapters by Zadrożny et al. and Urrutia et al. However, we do not know studies about the migration to fuzzy databases from classical or regular ones (Ben Hassine, Ounelli, Touzi, & Galindo, 2007). This chapter covers this gap.

Presentation of Our Approach

Basically, our approach consists in achieving some fuzzy characteristics in any existing database, mainly fuzzy queries. We study how to optionally migrate the data stored in RDBs towards FRDBs. This approach is addressed mainly to database administrators (DBA), and it is intended to meet the following requirements:

- to provide for methodical support of the migration process,
- to assist DBA in transforming relational schemas and database,
- to allow DBA to choose the attributes able to store imprecise data or/and be interrogated with flexible queries,
- to assist DBA in the list of required metadata,
- to exploit the full set of FRDBs features,
- to cover properties of the migration itself such as correctness and completeness.

We adopted in our migration approach three strategies answering users’ needs:

- **Partial migration**: Its goal is to keep the existing data, schema, and applications. The main benefit in this migration is the flexible querying, but also some fuzzy data mining methods could be implemented on crisp data.
- **Total migration**: Its goal is to store imprecise values and to benefit from the flexible querying, fuzzy data mining on fuzzy data.
- **Easy total migration**: Its goal is to store imprecise values, to benefit from the flexible querying, fuzzy data mining on fuzzy data, and to keep the existing data and applications with the minimum required modifications.

Consequently, all users’ needs are assured. In the first strategy, the existing schemas and data of the database remain unchanged. There is only the addition of the FMB and the FSQL server to treat the flexible queries. In the second strategy, two operations in one are aimed: model imprecise data and interrogate the database with flexible queries. This strategy requires a modification of the schemas, the data, and eventually the programs of the database. The third strategy is a mix of the previous strategies, and the basic idea is only to add the required fuzzy attributes, but these fuzzy attributes do not replace the previous existing classic attributes. Then, we have a redundancy problem, but if the space is not a problem, then it is easy to manage.
PARTIAL MIGRATION

The principle of this migration is based on the fact that all what is valid in SQL remains also valid in FSQL. This migration is destined to the designers who want to keep their RDB while taking advantage of fuzzy queries. The existing schemas and data of the database remain unchanged. There is only the addition of at least two elements (Galindo et al., 2006):

1. The metabase, named FMB, consisted of 12 tables with metadata about the fuzzy information (for some applications, we can use less than these 12 tables).
2. The FSQL server to utilize fuzzy queries. This server assures the translation of FSQL statements to SQL, a comprehensible language by the DBMS.

Managing Fuzzy Metadata

At the level of the database, the tables of the FMB are created in order to store the information about the attributes susceptible to be interrogated by flexible queries. These attributes, and only these attributes, must be declared in the FMB of type FTYPE1. The choice of this type is justified by two reasons:

A. **Consistency**: FTYPE1 attributes store only the previous existing crisp data. We must respect the following rules:

   - All the attributes remain unchanged, including the attributes identifiers.
   - The quantifiable attributes (with ordered domain, where we can define trapezoidal possibility distributions, that is, of type number, real, etc.) which will be interrogated by fuzzy queries must be declared like FTYPE1 in the FMB: The attributes are not modified but they must be included in the FMB as FTYPE1.
   - To include in the FMB the fuzzy attribute characteristics, detailed in Table 6, rows 1 and 2. This is optional, but it is mandatory if we can use such characteristics.

B. **Flexibility**: Fuzzy attributes FTYPE1 permit fuzzy queries allowing the use of:

   - Fuzzy constants (see The FSQL Language section) like linguistic labels ($Hot$), approximate values (#30), the

### Table 5. Examples of fuzzy querying on a fuzzy attribute Type 1 (Salary)

<table>
<thead>
<tr>
<th>Type</th>
<th>Query</th>
<th>Signification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classic (SQL)</td>
<td>SELECT * FROM Employee WHERE Salary = 2200;</td>
<td>List of the employees with a salary equal to 2200 euros.</td>
</tr>
<tr>
<td>Fuzzy (FSQL)</td>
<td>SELECT * FROM Employee WHERE Salary FEQ #2200;</td>
<td>List of the employees with a salary near to 2200 euros.</td>
</tr>
<tr>
<td>Fuzzy (FSQL)</td>
<td>SELECT % FROM Employee WHERE Salary FEQ $High;</td>
<td>List of the employees with a high salary.</td>
</tr>
</tbody>
</table>
values \texttt{UNKNOWN}, \texttt{UNDEFINED}, and \texttt{NULL}, trapezoidal possibility distributions as well as, of course, crisp values.

- Fuzzy comparators, useful in this kind of attribute (the entire Table 4).
- Fulfillment thresholds, fuzzy set operators (fuzzy union, fuzzy intersection, and fuzzy minus), and functions to modify fuzzy constants (concentration, dilatation, contrast intensification, etc.).

It is necessary to note at this level that the use of these different concepts is based on many rules. For example, the linguistic labels defined for the \texttt{FTYPE1} attributes must belong to a numerical domain, and this permits defining the associated trapezoidal possibility distributions in the FMB.

\textbf{Example 2}: The attribute Salary can be transformed in a fuzzy attribute Type 1. We can also define the labels \texttt{Low}, \texttt{Medium}, and \texttt{High}, for example, as the trapezoidal possibility distributions described in Figure 2. This attribute is quantifiable and have, for example, the euros as unit of measure. An attribute Productivity cannot be quantifiable. It can take the values “bad,” “regular,” and “good,” which cannot be represented by trapezoidal functions. For this reason, we cannot transform it to fuzzy attribute \texttt{FTYPE1}. We will show in the following section that this attribute can be \texttt{FTYPE3}.

\textbf{Example 3}: Table 5 shows three types of queries (classic and fuzzy) that can be applied to the attribute Salary (\texttt{FTYPE1}) of Example 2. It should be noted that the FSQL queries must be preceded by the creation of the label \texttt{High} and the margin for approximate values defined on this attribute in the FMB.

Finally, the partial migration allows three extra characteristics that may be very useful for many enterprises:

- \textbf{Fuzzy degrees (Table 6, row 4)}: We can add some different fuzzy degrees to each table. These attributes were explained in the subsection titled Fuzzy Degrees As Fuzzy Values.
- \textbf{Fuzzy quantifiers (Table 6, row 5)}: We could answer to questions like: “Give me the departments in which most of their employees have a high salary,” and we can add a minimum threshold to the condition and also to the quantifier. Note that in this example, quantifier “most” is associated to the context of employee table. FSQL defines four forms to use fuzzy quantifiers. See Galindo et al. (2006) for details about fuzzy quantifiers in FSQL.
- \textbf{Fuzzy data mining}: The definition of fuzzy attributes Type 1 allows using many fuzzy data mining methods. In this volume, there is a chapter by Feil and Abonyi summarizing these methods. In particular, FSQL is a powerful tool for these purposes (Carrasco, Vila, & Galindo, 2003; Galindo et al., 2006), and it is shown in the chapter by Carrasco et al. in this handbook.

\section*{The FSQL Server}

At the level of the DBMS, the FSQL server is placed over the DBMS to treat the flexible statements (queries, deletions, updates, etc.) written in FSQL language. Figure 3 shows the DBMS architecture with the FSQL server.
TOTAL MIGRATION

This strategy offers, in addition to the flexible querying, the possibility to store imprecise data at the level of the fuzzy attributes. Therefore, it will be a total migration towards the FRDB. Contrary to the previous case, the attributes susceptible to store imprecise data are of type FTYPE2, FTYPE3, and FTYPE4, and besides, some degrees may be included (types 5-8). The modification concerns only the tables defined or referenced to these attributes. This strategy comprises three main steps: (Step 1) schemas conversion, (Step 2) data conversion, and (Step 3) programs conversion. Note that this type of decomposition has already been used by Henrard et al. (2002); Henrard, Cleve, and Hainaut (2004); Cleve (2004); and Cleve, Henrard, and Hainaut (2005) to integrate some systems at the time of the reengineering of a database.

Figure 4 shows, in the left part, the main parts of the legacy system, comprising programs that interact with the legacy data through the legacy DBMS and through the legacy schema. The right part shows the state of the new system after the legacy DBMS has been extended with the FSQL server (Fuzzy DBMS) and the database with the FMB. The new database comprises the converted schema and data that have been transformed and migrated according to the new schema. Legacy programs have been transformed in such a way that they now access the data through the API of the new technology and through the new schema. When the converted system is deployed, new programs can be developed, that use the database through the native interface of the new fuzzy DBMS. Later on, if and when needed, the legacy programs could be rewritten according to the new technology.

Step 1: Database Schemas Migration

The schema conversion is the translation of the legacy database structure, or schema, into an equivalent database structure expressed in the new technology (Henrard et al., 2002). In our context, it consists in modifying the table schemas with fuzzy attributes which are going to store fuzzy values. Moreover, the FMB, which stores the fuzzy
How to Achieve Fuzzy Relational Databases Managing Fuzzy Data and Metadata

Table 6. Fuzzy metadata stored in the FMB

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 1. | Information concerning fuzzy attributes (see Fuzzy Attributes section):  
  - Fuzzy type, unit of measurement, comments, etc. |
| 2. | Fuzzy attributes FTYPE1 and FTYPE2:  
  - Fuzzy objects defined on these attributes: trapezoidal labels, qualifiers, quantifiers, etc.  
  - Margin for approximate values (the meaning of \#n in the context of each attribute).  
  - MUCH value: The minimum distance in order to consider two values as very separated: This is necessary for comparators MGT, NMGT, MLT, and NMLT (see Table 4).  
  - n value: Maximum number of points in the possibility distributions of new types 11 and 12 of Table 2. |
| 3. | Fuzzy attributes FTYPE3 and FTYPE4:  
  - Fuzzy objects defined on these attributes: linguistic labels, qualifiers, quantifiers, etc.  
  - n value: Maximum number of elements in the possibility distributions (see Table 3).  
  - Compatible attributes.  
  - Similarity measures between labels, only for FTYPE3. |
| 4. | Fuzzy degrees:  
  - Meanings of these fuzzy degrees (importance, membership, etc.).  
  - Association: Table, column, set of columns or without association. |
| 5. | Quantifiers associated to tables or general quantifiers for the general system. |
Algorithm 1. Physical schema transformation

Input: SPS (RDB)
Output: FPS (FRDB)
Begin
To create the FMB tables.
for each attribute A of SPS do
  if A remains classic then
    no modification in its definition;
  else /* This treatment is divided in two under-treatments: in the database and in the FMB */
    switch (type of A) { /* Modify the tables structure according to each fuzzy type */
      case FTYPE1: A remains unchanged.
      case FTYPE2:
        create at least 5 attributes with the following names:
        concatenate the first one with the letter 'T': AT;
        concatenate the others ones respectively with 1, 2, 3, 4, ..., 2n: A1, A2, A3, A4, ...;
        /* Note that 2n must be at least 4. Then the minimum value for n is 2. */
      case FTYPE3 and FTYPE4:
        create 2n+1 attributes with the following names:
        concatenate the first one with the letter 'T' (AT);
        for each i = 1, ..., n do { /* n = max. number of elements in the pos. distributions */
          concatenate the first attribute with Pi;
          concatenate the following with fi. /* (AP1, A1, AP2, A2, ..., APn, An) */
        }
    }
  Update the FMB tables with the metadata about all fuzzy attributes: see Table 6.
End.

Figure 6. Example of physical schema conversion at the level of the database

Algorithm 1 modifies the database schema for each fuzzy attribute. Note that we do not use the DDL statements of FSQL, because we want to show the inner schema in this conversion.

Example 4: Suppose the physical schema constituted of the tables EMPLOYEE, PROJECT, and WORKS_ON. Figure 6 shows the modifications done at the level of the fuzzy attributes:

- **Salary**: FTYPE1 with the linguistic terms low, medium, and high.
- **Age**: FTYPE2 with the linguistic terms young, adult, and old.
- **Budget**: FTYPE2 with the linguistic terms small, medium, and big.
- **Productivity**: FTYPE3 with the linguistic terms bad, regular, and good.

Note that, for example, attribute Budget is represented now with five classic attributes, and the Productivity attribute is transformed in three attributes. Figure 7 shows the modifications done.
Figure 7. Example of physical schema conversion at the level of the FMB

<table>
<thead>
<tr>
<th>OBJ#</th>
<th>COL#</th>
<th>P_TYPE</th>
<th>LEN</th>
<th>COLUMN_NAME</th>
<th>UM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>USER</td>
<td>EMPLOYEE.AGE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>USER</td>
<td>EMPLOYEE.SALARY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>USER</td>
<td>EMPLOYEE.PRODUCTIVITY</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>USER</td>
<td>PROJECT.BUDGET</td>
</tr>
</tbody>
</table>

**FUZZY.COL_LIST (FCL)**

<table>
<thead>
<tr>
<th>OBJ#</th>
<th>COL#</th>
<th>MARGIN</th>
<th>MUCH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
<td>1000</td>
</tr>
</tbody>
</table>

**FUZZY.OBJEC IList (FOL)**

<table>
<thead>
<tr>
<th>OBJ#</th>
<th>COL#</th>
<th>FUZZY ID</th>
<th>FUZZY_NAME</th>
<th>FUZZY_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>'YOUNG'</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>'ADULT'</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>'OLD'</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>'LOW'</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>'MEDIUM'</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>'HIGH'</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>'BAD'</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>'REGULAR'</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>'GOOD'</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>'SMALL'</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>'MEDIUM'</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>'BIG'</td>
<td>4</td>
</tr>
</tbody>
</table>

**FUZZY_LABEL_DEF (FLD)**

<table>
<thead>
<tr>
<th>OBJ#</th>
<th>COL#</th>
<th>FUZZY ID</th>
<th>ALFA</th>
<th>BETA</th>
<th>GAMMA</th>
<th>DELTA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>25</td>
<td>22</td>
<td>30</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>25</td>
<td>22</td>
<td>45</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>50</td>
<td>55</td>
<td>62</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>850</td>
<td>1200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1000</td>
<td>1200</td>
<td>1700</td>
<td>2200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1700</td>
<td>2200</td>
<td>100000</td>
<td>100000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1000</td>
<td>1500</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1000</td>
<td>1500</td>
<td>6000</td>
<td>7000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>6000</td>
<td>7000</td>
<td>100000</td>
<td>100000</td>
</tr>
</tbody>
</table>

**FUZZY_MEANNESS_DEF (FND)**

<table>
<thead>
<tr>
<th>OBJ#</th>
<th>COL#</th>
<th>FUZZY ID1</th>
<th>FUZZY ID2</th>
<th>DEGREE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>2</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>0.7</td>
</tr>
</tbody>
</table>

368
How to Achieve Fuzzy Relational Databases Managing Fuzzy Data and Metadata

Figure 8. Conceptual schema conversion

Figure 9. Examples of conceptual schema conversion
How to Achieve Fuzzy Relational Databases Managing Fuzzy Data and Metadata

Conceptual Schema Conversion

The conceptual schema conversion consists in extracting the physical schema conversion of the legacy database (SPS) and transforming it into its correspondent conceptual schema through a database reverse engineering (DBRE) process. Figure 8 describes this process.

First of all, the source DDL script written in SQL is parsed in order to extract the source physical schema (SPS). This last is refined through an in-depth inspection of the way that the program uses and manages the data. The final DBRE step is the conceptualization that interprets the physical schema into the source conceptual schema (SCS).

Then, a phase for FDB design makes a schema conversion, introducing new concepts (fuzzy constraints, fuzzy attributes, etc.) (Galindo et al., 2004a, 2004c, 2006; Urrutia et al., 2002). This transformation produces the fuzzy conceptual schema (FCS), and then the fuzzy physical schema (FPS), which is coded in the final DDL script of the FRDB.

At this level, we have two options:

Algorithm 2. Data transformation

```java
Input: Source data (RDB)
Output: Target data (FRDB)
Begin
for each attribute A of DML script do
    for each value v in A (for each row in the database table) do
        switch (Type of A) do {
            case Classic and FTYPE1:
                Insert v in its reserved field (no change);
            case FTYPE2:
                if v=NULL then
                    Attribute AT=2 and A1=A2=A3=A4=NULL;
                else /* v is a crisp value */
                    AT=3, A1=v and A2=A3=A4=NULL;
            case FTYPE3 and FTYPE4:
                if v=NULL then
                    Attribute AT=2 and AP1=A1=NULL;
                else /* v is a crisp value as text-label */
                    AT=3, AP1=1 and A1=v;
        }
End
```

in the FMB tables. We include here a brief explanation of these tables: Table FCL stores all the fuzzy attributes (type, unit of measurement, etc.). Table FAM stores the margin for approximate values and the minimum distance for two values considered very separated for all FTYPE1 and FTYPE2 attributes. Table FOL includes all fuzzy objects belonging to all fuzzy attribute types (only labels in our example). Table FLD includes the definition of trapezoidal labels for FTYPE1 and FTYPE2 attributes (the four basic values, like in Figure 1). Table FND stores the similarity relation between each labels of a FTYPE3 attribute. Of course, the FMB needs more tables with different information (Galindo et al., 2006).
How to Achieve Fuzzy Relational Databases Managing Fuzzy Data and Metadata

1. To CREATE new tables, copy values from old tables and, optionally, drop old tables and rename new tables (the best option).

2. To ALTER old tables and to preserve current information, we must only store old attributes values (which will be converted in fuzzy attributes) in temporary tables, modify the structures of this tables, convert old attributes values to fuzzy attributes values, and copy them according to the new fuzzy attributes structure (Tables 2 and 3).

It is necessary to note that during the database design step, the choice of the most suitable fuzzy attribute type is a delicate task. The presence of an expert in FRDB design must be counseled strongly due to the complexity of the assimilation of the different fuzzy concepts (Ben Hassine, Touzi, & Ounelli, 2007).

Example 5: Figure 9 shows an example of conceptual schema conversion from a legacy conceptual schema to a FuzzyEER schema. Attribute Salary is transformed to FTYPE1, Age and Budget to FTYPE2, and Productivity to FTYPE3. The other attributes and primary keys remain unchanged. The constraints of the source ER schema can be transformed also to fuzzy constraints using the fuzzy (min, max) notation as this presented in the relationship Works_On. For example, in the PROJECT side, the (min, max) constraint indicates that in each project must work a minimum of approximately two employees (with a minimum degree of 0.5). At the same time, the number of employees in each project must not exceed approximately 30 (with a minimum degree of 0.25).

Step 2: Data Migration

The data conversion consists in transforming the data of the RDB (crisp) to the format of the data defined by the fuzzy schema. It involves data transformations that materialize the schema transformations described above. These transformations include three stages as shown in Figure 10. The first step consists in extracting the data from the database. This extraction takes in account the different kind of constraints, data coherence, security, and so forth. The second step consists in converting these data in such a way that their structures coincide with the new format of the target FRDB schema. This transformation is made automatically using algorithm 2, or manually using expert knowledge as we will explain later. Finally, the data will be stored in the FRDB.

However, during this transformation, there is a modification in the data representation for fuzzy attributes at the level of the database tables (see Tables 2 and 3 and the example in Figure 6) while introducing the data values according to their types

<table>
<thead>
<tr>
<th>Matriculate</th>
<th>Name</th>
<th>Salary</th>
<th>AgeT</th>
<th>Age1</th>
<th>Age2</th>
<th>Age3</th>
<th>Age4</th>
</tr>
</thead>
<tbody>
<tr>
<td>005201</td>
<td>Habib</td>
<td>2000</td>
<td>3</td>
<td>50</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>005202</td>
<td>Mohamed Ali</td>
<td>2000</td>
<td>4</td>
<td>1 (Id. for Adult)</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>005203</td>
<td>José</td>
<td>2100</td>
<td>6</td>
<td>40</td>
<td>30</td>
<td>50</td>
<td>10</td>
</tr>
<tr>
<td>005204</td>
<td>Amel</td>
<td>2200</td>
<td>0</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
How to Achieve Fuzzy Relational Databases Managing Fuzzy Data and Metadata

It should be noted that if we want to “fuzzify” some previous data, then the transformation is not automated. Fuzzy information may be more real than crisp information. For this reason, the intervention of an expert in FRDB design and in the database domain is strongly counseled in order to choose the most suitable type among the different types of fuzzy values mentioned previously (Ben Hassine et al., 2007). Sometimes, the crisp data can be kept. In other cases, they will be transformed, using some standard rules, in linguistic terms, intervals, approximate values, and so forth, Especially in some contexts, NULL values may be transformed into UNKNOWN values.

Algorithm 2 presents the automatic modification. This step shows the advantages of the migration towards the FRDBs in terms of imprecise data modeling. It is important to note that in the legacy database, all attributes are crisp: Then, the translation is very easy; we have to choose between two values: NULL or crisp. Another easy transformation is, for example, to store approximate values for FTYPE2 attributes: Attribute A1=6, A2=v−margin, A3=v+margin, and A4=margin (values in A2, A3, and A4 are only for increasing the efficiency).

Example 6: Suppose the relation Employee described in Table 7, schema in Figure 6, where we assigned the FTYPE1 to the attribute Salary, and the FTYPE2 to Age. Since the attribute salary does not have any transformation at the level of the database, only the attribute Age is transformed, as shown in Table 8.

All fuzzy data are represented by a number. Also, every fuzzy object has an identifier in the FMB. In our example, the attribute AgeT stores the kind of data stored about the Age (see Table 2). The parameters of this data are stored in the remaining attributes.

- The employee Habib keeps the crisp value 50 years. This value (50) and its type (3) are stored respectively in the fields Age1 and AgeT.
- The age of Mohamed Ali receives the linguistic label adult, which must be first of all created (with the command CREATE LABEL) and stored in the FMB as shown in Figure 7. The identifier of this label will be stored in the field Age1, while specifying its type (4) in the field AgeT.
- The employee José stores an approximate value of 40 years old with a margin of 10. We store 40 in the field Age1 and its identifier in AgeT (6).
- The age of Amel stores NULL value, but in this case we can translate it to UNKNOWN (because we know that every person has some Age).

Step 3: Programs Migration

The modification of the structure of the database requires, in the majority of the cases, propagation to the level of their related programs. Since that is not the first interest of this work, we present an overview of programs conversion. In fact, if we want to use flexible queries and to store imprecise data, the communication between programs and the database must be through the FSQL server. The programs must be modified according to the representation, interrogation, and storage of the new data. Moreover, we must decide what to do with fuzzy values in each program. Note that emigrating these programs not only means to convert DBMS calls in programs, but in addition requires the reengineering of imperative programs to accept fuzzy values and surely the reconstruction of user interfaces.

We draw our inspiration from the strategies of programs conversion proposed by Henrard et al. (2002, 2004) and Cleve (2004). One of these strategies relies on wrappers that encapsulate the FRDB. This strategy allows interaction of the application programs with the legacy data ac-
cess logic through these wrappers instead of the legacy DBMS. These wrappers are in the form of a software layer permitting the translation of the previous statements written in SQL to the FSQL language (fuzzification in the statement, not in the data). The FSQL server translates in turn these queries to SQL language (defuzzification in the statement). The fuzzy returned answers will be defuzzified in order to be treated by the application programs. This process is depicted in Figure 11. The second strategy consists in rewriting the access statements in order to make them process the new data through the new fuzzy DBMS-DML. Each DML statement must be located, its parameters must be identified, and the new fuzzy DML statement sequence must be defined and inserted in the code. This task may be complex because legacy program will manage imprecise data instead of legacy crisp ones.

The third strategy generalizes the problems met in the second strategy. In fact, the change of paradigm when moving from standard crisp data in RDB to imprecise ones in FRDB induces problems such as whether the user wants now to use fuzzy information in FSQL statements and the manipulation of the imprecise data returned by these statements. The program is rewritten in order to use the new fuzzy DBMS-DML at its full power and take advantage of the new data system features. This strategy is much more complex than the previous one since every part of the program may be influenced by the schema transformation. The most obvious steps consist of:

1. Identifying the statements and the data objects that depend on these access statements,
2. Deciding whether each statements will be now fuzzy or not,
3. Rewriting these statements and redefining its data objects, and
4. Treating the possibly fuzziness in returned answers.

**EASY TOTAL MIGRATION**

As we have shown, the migration of programs may be a very hard task in this process, but it is mandatory and essential in the total migration. With the *Easy Total Migration*, we achieve a total fuzzy database (storing fuzzy values, fuzzy querying, and fuzzy data mining on fuzzy data) and also keep the existing data and applications with the minimum required modifications. The basic idea is to mix partial and total migration; that is, fuzzy attributes with fuzzy values are duplicated: one with fuzzy values and the other with only crisp values. In this process, we use the three
How to Achieve Fuzzy Relational Databases Managing Fuzzy Data and Metadata

Steps of the total migration with some modifications: (Step 1) schemas conversion, (Step 2) data conversion, and (Step 3) programs conversion. The steps 1 and 2 are equal but now we preserve the old attributes. For example, if the Age attribute is a fuzzified attribute and converted to FTYPE2, then we preserve the existing Age attribute and add the new attributes AgeT, Age1, Age2, Age3, and Age4 (like in Table 8).

The program conversion is now easier, but we must manage the new fuzzy attributes in some DML statements in order to achieve legacy programs running exactly like before the migration:

1. **SELECT**: No modifications required (except if the `SELECT` uses the asterisk, `*`, because it represents all the attributes and in the new FRDB there are more attributes).
2. **DELETE**: No modifications required.
3. **INSERT**: The values of the fuzzified attributes must be inserted again in the same row in the corresponding new fuzzy attributes (Algorithm 2).
4. **UPDATE**: The values of the fuzzified attributes must also be updated in the same row in the corresponding new fuzzy attributes (Algorithm 2).

The main drawback of this migration strategy is the redundancy in the fuzzified attributes (except the FTYPE1). The main advantage is the easy program migration. In some situations, this is the best option, using this strategy as an intermediate and temporary step to a total migration.

**CONCLUSION**

Several real applications need to manage imprecise data and to benefit its users from the flexible querying (Bosc et al., 1998). Several theoretical solutions have been proposed (Bosc & Pivert, 1995; Buckles & Petry, 1982; Galindo, 1999; Medina et al., 1994; Umano, 1982; Zemankova-Leech & Kandel, 1985). Unfortunately, the repercussions of these works on the practical level are negligible, even with the existence of some prototypes as FSQL server (Galindo et al., 1998, 2006). In this chapter, we proposed a migration approach from RDBs towards the generation of FRDBs. This approach is addressed mainly to database administrators and enterprises interested in such a fuzzy migration.

We proposed three possible strategies for this migration. The first strategy, is called **partial migration** and enjoys some of the advantages of the FRDBs while preserving the existing schema, data, and applications. The second approach, named **total migration**, consists in benefiting from all FRDBs advantages (imprecise data storage, flexibility in queries, etc.). It allows an easy mapping of the existing data, schemas, and programs, while integrating the different fuzzy concepts. This strategy, based on the Henrard et al. (2002) approach, has three levels of conversions: conversion of the physical schema that generally can be preceded by a conceptual schema conversion, data conversion, and applications conversion. We studied in detail the first two levels. The applications conversion, or migration of programs, may be a very hard task in this process, but it is mandatory and essential in the total migration and must be made with experts in fuzzy databases and in the database domain. The third strategy, called **easy total migration**, is a mixture of the previous two strategies, generally as a temporary step. This option is easier and faster than the total migration and may be a good option in order to make the migration of programs slowly but painstakingly.

As for perspectives on the future of this work, we mention (1) the automation of conversion of the schema, data, and applications, following the algorithms defined in this chapter and (2) the addition of an expert system to help the designer choose the appropriate fuzzy attributes type and other fuzzy objects (labels, quantifiers, etc.). This last point also contributes to the use of the FRDBs easier in real applications. In fact, one of the meted problems during the FRDBs design is to determine the attributes (columns) susceptible to store fuzzy
data and to choose their respective data types. The type assignment to these attributes is not an obvious task. This choice asks the designer, on the one hand, to know in a detailed way the properties of every fuzzy attribute type and, on the other hand, to really qualify the attribute in order to affect it the most suitable type among the four types of fuzzy attributes mentioned previously. We started working in order to solve this problem, and we have implemented an expert system to choose the suitable attribute type. It can also easily treat other fuzzy models by enriching its knowledge base. We are working now to combine this expert system in such a way that the migration process will be automated. We think that the automation of this migration will be a useful starting point in order to generalize the FDBs in the real database world.

**ACKNOWLEDGMENT**

This work has been partially supported by the “Ministry of Education and Science” of Spain (projects TIN2006-14285 and TIN2006-07262) and the Spanish “Consejería de Innovación Ciencia y Empresa de Andalucía” under research project TIC-1570.

**REFERENCES**


How to Achieve Fuzzy Relational Databases Managing Fuzzy Data and Metadata

(Eds.), Proceedings of the Workshop on Software-Reengineering, Bad Honnef, Germany.


**KEY TERMS**

**CDEG Function:** Function defined in FSQL to compute the Compatibility DEGree of each row. This value is the fulfillment degree of each row to the fuzzy condition included in the WHERE clause of a SELECT statement in FSQL language. This function may be used with an attribute in the argument and then it computes the fulfillment degree for the specific attribute. If the argument is the symbol asterisk, *, then it computes the fulfillment degree using the whole condition, even whether it includes fuzzy conditions on different attributes.

**Fuzzy Attribute:** In a database context, a fuzzy attribute is an attribute of a row or object in a database, which allows querying by fuzzy information and/or storing this kind of information.

**Fuzzy Database:** If a regular or classical database is a structured collection of records or data that is stored in a computer, a fuzzy database is a database which is able to deal with uncertain or incomplete information using fuzzy logic. There are many forms of adding flexibility in fuzzy databases. The simplest technique is to add a fuzzy membership degree to each record, that is, an attribute in the range [0,1]. However, there are other kinds of databases allowing fuzzy values to be stored in fuzzy attributes using fuzzy sets, possibility distributions, or fuzzy degrees associated to some attributes and with different meanings (membership degree, importance degree, fulfillment degree, etc.). Of course, fuzzy databases should allow fuzzy queries using fuzzy or nonfuzzy data, and there are some languages that allow these kinds of queries, like FSQL or SQLf. In synthesis, the research in fuzzy databases includes the following four areas: flexible querying in classical or fuzzy databases, extending classical data models in order to achieve fuzzy databases (fuzzy relational databases, fuzzy object-oriented databases, etc.), fuzzy data mining techniques, and applications of these advances in real databases.

**FSQL (Fuzzy SQL):** Extension of the popular language SQL that allows the management of
fuzzy relational databases using the fuzzy logic. Basically, FSQL defines new extensions for fuzzy queries, extending the \texttt{SELECT} statement, but it also defines other statements. One of these fuzzy items is the definition of fuzzy comparators using mainly the possibility and necessity theory. Besides, FSQL allows the definition of linguistic labels (like hot, cold, tall, short, etc.) and fuzzy quantifiers (most, approximately 5, near the half, etc.). The more recent publication about FSQL is the book \textit{Fuzzy Databases: Modeling, Design and Implementation} by Galindo et al. (2006).

**Fuzzy Comparators:** They are different techniques to compare two values using fuzzy logic. FSQL defines fuzzy comparators like FEQ (fuzzy equal), NFEQ (necessarily fuzzy equal), FGT (fuzzy greater than), NFGT (necessarily fuzzy greater than), and so forth.

**Fuzzy Metaknowledge Base (FMB):** In a fuzzy database, the FMB is the extension of the data dictionary in order to store the fuzzy metadata, that is, information about fuzzy objects: fuzzy data type of each fuzzy attribute, the definition of labels, the margin for approximate values, the minimum distance for very separated values, fuzzy quantifiers, and so forth.

**Fuzzy Migration:** Migration from crisp databases towards fuzzy databases in order to introduce imprecise/fuzzy information in current information systems. This fuzzy migration consists in deriving a new database from a legacy database and in adapting data, metadata, and the software components accordingly. It does not only constitute the adoption of a new technology, but also the adoption of a new paradigm.

**Fuzzy Query:** Query with imprecision in the preferences about the desired items. These preferences may be set usually using fuzzy conditions in the queries. These fuzzy conditions include many possible forms like fuzzy preferences (e.g., I prefer bigger than cheaper), fuzzy labels (e.g., hot and cold), fuzzy comparators (e.g., approximately greater or equal than), fuzzy quantifiers (e.g., most or approximately the half), and so forth. One basic target in a fuzzy query is to rank the resulting items according to their fulfillment degree (usually a number between 0 and 1).

**FuzzyEER Model:** Conceptual modeling tool, which extends the Enhanced Entity Relationship (EER) model with fuzzy semantics and fuzzy notations to represent imprecision and uncertainty in the entities, attributes, and relationships. The basic concepts introduced in this model are fuzzy attributes, fuzzy entities, fuzzy relations, fuzzy degrees, fuzzy degrees in specializations, and fuzzy constraints. A complete definition of this model is published in the book \textit{Fuzzy Databases: Modeling, Design and Implementation} (Galindo et al., 2006).

**Legacy System:** Existing system in a concrete context, for example, an existing database.

**SQL (Structured Query Language):** A computer language used to create, retrieve, update, and delete data from relational database management systems. SQL has been standardized by both ANSI and ISO. It includes DML (Data Management Language) and DDL (Data Definition Language). The statement for querying is the \texttt{SELECT} command.

**ENDNOTES**

1. \textbf{Oracle} is possibly the most powerful database system. The last versions are object-relational databases and designed for grid computing. Some distributions are free but with some limits (such as, to store up to 4GB of user data). It began three decades ago with only one relational database and actually it runs on all major operating systems, including Linux, UNIX (AIX, HP-UX, Mac OS X, Solaris, Tru64), and Windows. Official web page: http://www.oracle.com

2. \textbf{PostgreSQL} is a powerful, open source relational database system. It has more than
15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. It runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows. Official web page: http://www.postgresql.org

A DBRE is a technology used to recover the conceptual schema that expresses the semantics of the source data structure.