Fuzzy Databases
Modeling, Design and Implementation

JOSE GALINDO
ANGELICA URRUTIA
MARIO PIATTINI

Fuzzy Databases: Modeling, Design and Implementation

José Galindo
University of Málaga, Spain

Angélica Urrutia
Catholic University of Maule, Chile

Mario Piattini
University of Castilla-La Mancha, Spain

IDEA GROUP PUBLISHING
Hershey • London • Melbourne • Singapore

Acquisitions Editor: Renée Davies
Development Editor: Kristin Roth
Senior Managing Editor: Amanda Appicello
Managing Editor: Jennifer Neidig
Copy Editor: Eva Brennan
Typesetter: Amanda Kirlin
Cover Design: Lisa Tosheff
Printed at: Yurchak Printing Inc.

Published in the United States of America by
Idea Group Publishing (an imprint of Idea Group Inc.)
701 E. Chocolate Avenue, Suite 200
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@idea-group.com
Web site: http://www.idea-group.com

and in the United Kingdom by
Idea Group Publishing (an imprint of Idea Group Inc.)
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 0609
Web site: http://www.eurospanonline.com

Fuzzy databases : modeling, design and implementation / Jose Galindo, Angelica Urrutia and Mario Piattini, editors.
p. cm.
Summary: "This book includes an introduction to fuzzy logic, fuzzy databases and an overview of the state of the art in fuzzy modeling in databases"--Provided by publisher.
Includes bibliographical references and index.
ISBN 1-59140-324-3 (h/c) -- ISBN 1-59140-325-1 (pbk.) -- ISBN 1-59140-326-X (ebook)
1. Database management. 2. Fuzzy sets. I. Galindo, Jose, 1970- II. Urrutia, Angelica. III. Piattini, Mario, 1966-
QA76.9.D3F8935 2005
005.74--dc22
2005013546
British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Dedication

To my family and to everyone who finds this book interesting (with a fuzzy degree of at least 0.2).

José Galindo

To my parents, Hugo and Angélica.

Angélica Urrutia

To the Velthuis family.

Mario Piattini

Fuzzy Databases: Modeling, Design and Implementation

Table of Contents

Foreword

Since Zadeh's Fuzzy Sets Theory was formulated, a lot of efforts have been devoted to extend databases with mechanisms to represent and handle information in a flexible way. The proposals appearing in the literature to deal with this aim are mainly supported in the possibilistic models, similarity relationship models, or the combination of both perspectives. This fact, together with the variety of database models susceptible of extension (i.e., relational model, object oriented models, logic model, object-relational model, etc.), has given rise to many approaches of fuzzy database models.

The materialization of these models in Fuzzy DBMS has not been so fructuous, and the development of applications supported by these systems is in an exploratory stage.

The implementation of Fuzzy DBMS will be determined by the development of applications that take advantage of the capabilities of these ones to operate with flexible information when solving real-life problems. In this sense, different areas of application have appeared, and in this book, some examples are collected, such as data mining, information retrieval, content-based image retrieval, and classical applications in the management field, improved with the possibility of manipulating flexible information (see, for example, http://idbis.ugr.es/immosoftweb for an online real-estate portal based on flexible search. It is built on the FSQL server developed by José Galindo and other members of the IDBIS group).

One issue that, from my point of view, has not been paid enough attention from the scientific community has been the extension of the conceptual models for the design of databases to the ambit of the representation of incomplete

information. In this sense, this book put together the most important proposals present in the literature. This study is completed with a deep analysis of the features of modeling susceptible of fuzzy treatment to present, next, a fuzzy extension of the EER model, which gives a notation for each of these features. The fuzzy concepts identified in the ambit of modeling require, in a similar way as in the classical case, a DBMS that permits the representation and handling of this type of information. The authors have incorporated these new characteristics to previous models and prototypes of fuzzy databases. The new model, the new data structures, and the new capabilities of handling have given as a result FIRST-2 and a new extension of FSQL (Fuzzy SQL), both of them thoroughly described in this book. The creation of an algorithm that permits the translation of the conceptual definition in terms of FuzzyEER into FSQL sentences completes an important cycle in relation to the conceptual design oriented to fuzzy databases.

Though the central argument of the book is the description of a notation for the conceptual design in an imprecise environment, this volume collects and proposes many worthy resources in the area of fuzzy databases, which makes it an important reference for those people interested in this field in general.

Dr. Juan Miguel Medina
Senior Researcher
Member of the IDBIS Group
Granada, Spain, January 2005

Preface

In 1965 at the University of California, Berkeley, also called the "Athens of the Pacific," Lotfi A. Zadeh[1] introduced the theory of fuzzy sets and fuzzy logic, two concepts that laid the foundation of possibility theory in 1977. These terms were coined by him to deal with the phenomenon of vagueness, in the cognition process of the human being. According to Zadeh, "the theory of fuzzy sets is a step toward a rapprochement between the precision of classical mathematics and the pervasive imprecision of the real world… a rapprochement born of the incessant human quest for a better understanding of mental processes and cognition[2]."

Since then, an enormous quantity of congresses and publications around the world has intended to explore and develop this basic idea of vagueness and its industrial application. Zadeh also said: "at present, we are unable to design machines that can compete with humans in the performance of such tasks as recognition of speech, translation of languages, comprehension of meaning, abstraction and generalization, decision-making under uncertainty and, above all, summarization of information."

When we look at the growth of the Japanese industry in the 1980s we can understand the relevant impact of "fuzzy technologies" in the modeling and design of new products[3].

In these aspects, the gap between the industrial domain and the research domain can be seen in books, journals, articles, cases studies, proceedings, and so forth. In fact, these are the greatest tools to put the theoretical knowledge in action (i.e., in Idea Group Publishing you can find the latest advance in the research of information science, technology, and management). But it is diffi-

cult to find a pedagogical book to help the learning process of the students in computer science in the area of fuzzy databases.

I am glad to tell you that the book you have in yours hands has the courage to attack the problem of fuzzy databases, with a clear and direct approach guiding the reader step-by-step through the understanding process. Indeed this book has the ability to help you in the modeling, design, and implementation processes of fuzzy databases. This book gives you a first glance at a systematic exposition of the three issues (modeling, design, and implementation). Perhaps the only regret I have in this book is the use of Oracle platform, which, in my view, has the influence of the industrial software of the 1990s. However, the definitions, ideas, and new approaches are platform independent.

Before I say something about the features of the book, I would like to explain some historical aspects that I find interesting to being taken into account by readers. First, in Europe there are two cities well known by the implication of the database in the Zadeh legacy: Toulouse and Granada.

In 1985 Didier Dubois[4] and Henry Prade[5] published *Théorie des Possibilités — Applications à la représentation des connaissances en informatique*, which was translated into English three years later as *Possibility Theory: An Approach to Computerized Processing of Uncertainty*. In Chapter VI of this book the authors introduce the use of the possibility distribution to represent incomplete and uncertain dates in a relational database. This chapter was the result of a PhD thesis written in Toulouse by Claude Testemale[6] and co-directed by Prade. In this work you can see the original code in MACLISP for fuzzy query processing.

Some years later, in Granada, the book of Dubois and Prade, in particular Chapter VI, had a great impact on the PhD thesis of Juan Miguel Medina[7]. In that work Medina summarized the main fuzzy database models in three families (Chapter III): The fuzzy relational model (with a fuzzy degree in each row or tuple), the model based in similarity relations by Buckles and Petry, and the relational models with possibility distributions by Umano, Fukami, Prade, Testemale, Zemankova, Kaendel and other authors. Medina's PhD thesis also embraced the generalizations of fuzzy models. Medina proposed a conceptual framework for fuzzy representation called GEFRED (Generalized Model for Fuzzy Relational Databases) and a language called FSQL (Fuzzy SQL). In the same research group a young mathematician and informatic José Galindo[8] started his PhD research under the supervision of Medina, in order to improve the relational algebra of the GEFRED model, to define a fuzzy relational calculus and to implement other fuzzy comparators. In fact, the possibility and necessity measures, shown by Dubois and Prade, do not only allow the con-

struction of two fuzzy comparators, but 14 of them. The implementation of a new FSQL server running in Oracle and a new GUI interface of the FSQL language was included too.

In these two theses, part of the job was concluded; that is, the physical and logical approaches for development of fuzzy databases. Nevertheless, the conceptual design of fuzzy entities and relations was still missing.

This last step was achieved in 2003 by Angelica Urrutia[9], in her PhD research under the supervision of José Galindo and Mario Piattini. In this work, you find a conceptual fuzzy model, so-called FuzzyEER, and a case tool (FuzzyCASE), to help the database engineers to build the conceptual model for fuzzy databases.

Herein lie the roots of this book, the logical fuzzy models of Medina (1994) and Galindo (1999) on one hand, and, on the other hand, the conceptual fuzzy model of Urrutia (2003).

Personally, I find the name of the book *Fuzzy Databases: Modeling, Design and Implementation* quite right because the work of Galindo, Urrutia, and Piattini is a highly important contribution to understanding the fuzzy database process, not only by professionals of software engineering, but also by computer science students. I hope this book has a real influence in the orientation of the databases courses.

Chapter I, dedicated exclusively to the fuzzy logic, should be appreciated. This chapter could be very useful to new students in this area.

Chapter II brings up to date the classification of fuzzy database models, including some ideas about fuzzy object-oriented database models centered in the relational model, even though these ideas are not used in this book. In spite of this, the contributions of this book will turn out to be very useful for the definition of a complete fuzzy object-oriented database model.

Chapter III is focused on fuzzy database modeling, showing some of the more important approaches by other authors. This chapter is important in order to understanding the importance of the FuzzyEER model defined in Chapter IV, an extension of an EER model to create a model with fuzzy semantics and notations. Although the model has numerous characteristics, the main components of this data modeling tool are: imprecise attributes; fuzzy attributes associated to one or more attributes or with an independent meaning; degrees of fuzzy membership to the model itself, such as fuzzy aggregation, fuzzy entity, weak fuzzy entity, fuzzy relationship; and defined specialization with fuzzy degrees.

Chapter V describes how to represent fuzzy knowledge in relational databases. This methodology is debatable. Nevertheless, as the authors said, it is

complete enough for the immense majority of the applications. On the other hand, the possible lacks in that methodology may be easily solved in each specific application. Chapter VI gives the steps of an algorithm for mapping FuzzyEER models to that methodology. This algorithm relates Chapter IV and V.

Chapter VII describes the more important statements of the FSQL language. This definition improves upon the previous version of this language in many aspects. The educational experience of the authors is noted also in this chapter, which includes a multitude of examples that permits understanding of the utility of each definition.

With all the tools defined in previous chapters, Chapter VIII studies some applications of fuzzy databases. These applications show that fuzzy databases are useful in areas other than management applications (storing and querying information). Of course, FSQL may be used for fuzzy querying, but it can also be used for fuzzy clustering and fuzzy classification, for defining fuzzy dependencies, and for the fuzzy characterization of images in a system of fuzzy image retrieval. The last chapter, the appendices, and references close the book, giving additional information. The open research lines are especially interesting, because they prove that this matter is not closed.

Finally, I borrow the words said by Zadeh in May of 1972, in his Preface of Kaufmann's book, "Professor Kaufmann's treatise is clearly a very important accomplishment. It may a well exert a significant influence on scientific thinking in the years ahead and stimulate much further research on the theory of fuzzy sets and their applications in various field of science and engineering." Well, I think these words match the aim of this book too.

Endnotes

1 See http://www.cs.berkeley.edu/~zadeh

2 This proposal was mentioned by Zadeh, in the he wrote for the preface of the book written by A. Kaufmann in 1977, *Introduction à la théorie des sous-ensembles flous à l'usage des ingénieurs*.

3 The reader can find some ideas related to fuzzy control of engineering systems in the book by Kazuo Tanaka in 1996, *An Introduction to Fuzzy Logic for Practical Applications*.

[4] Dubois, D. (1983). *Modèles mathématiques de l'imprécis et de l'incertain en vue d'applications aux techniques d'aide à la decision*. Doctoral dissertation. Medical and Scientific University, Grenoble, France.

[5] Prade, H. (1982). *Modèles mathématiques de l'imprécis et de l'incertain en vue d'applications au raisonnement naturel*. Doctoral dissertation. Paul Sabatier University, Toulouse, France.

[6] Testemale, C. (1984). *Un système de traitement d'informations incomplètes ou incertaines dans une base de données relationnelles*. Doctoral dissertation. Paul Sabatier University, Toulouse, France.

[7] Medina, J.M. (1994). *Bases de Datos Relacionales Difusas: Modelo teórico y aspecto de su implementación*. Doctoral dissertation. University of Granada, Spain.

[8] Galindo, J. (1999). *Tratamiento de la imprecisión en bases de datos relacionales: Extensión del modelo y adaptación de los SGBD actuales*. Doctoral dissertation. University of Granada, Spain.

[9] Urrutia, A. (2003). *Una definición de un modelo conceptual a una base de datos difusa. Doctoral dissertation*. University of Castilla-La Mancha, Spain.

[10] On February 22, 2005, Dr. Jiménez was awarded with the Trophy Fernand Gallais, who grants the *Ecole Nationale Supérieure des Ingénieurs in Arts Chimiques et Technologiques* of the Polytechnic Institute of Toulouse, France, for his PhD thesis *Gestion des connaissances imparfaites dans les organisations industrielles: Cas d'une industrie manufacturière en Amérique Latine*.

Dr. Leoncio Jiménez[10]

Talca, Chile, June 2005

Acknowledgments

The authors would like to thank the CyTED project VII-J-RITOS2 (and its main researcher, Professor Nieves R. Brisaboa) and MCYT project TIC2002-00480 for their partial support, which contributed to the final formulation of this work.

In addition, IDBIS Research Group (Intelligent DataBases and Information Systems; see http://idbis.ugr.es or http://frontdb.ugr.es) has been one of our inspirations. The works of these researchers have been and are very important in the field of fuzzy logic and fuzzy databases. Thank you very much.

Of course, many people collaborated directly or indirectly so that this book could finally be published. They know who they are. Thanks to all, and especially to Dra. M. Carmen Aranda for her useful technical revisions in Chapter VIII.

Chapter I

Introduction to Fuzzy Logic

This book mixes concepts of different areas of knowledge or technologies, such as databases, system architecture design, SQL language, programming concepts and logic, mathematics, and so forth. These concepts are introduced where they correspond, although we have not intended this book to be an introduction to databases or information systems. An important part of this book utilizes the fuzzy logic. For that reason we will begin by introducing some basic concepts of the theory of fuzzy sets as well as the notation used in this book. In this summary we will focus on the semantic aspects and those of representation associated with this important theoretical tool. In written sources we can find a large number of papers dealing with this theory, which was first introduced by L.A. Zadeh[1] in 1965 (Zadeh, 1965). A compilation of some of the most interesting articles published by Zadeh on the theme can be found in Yager et al. (1987). Dubois and Prade (1980, 1988) and Zimmerman (1991) bring together the most important aspects behind the theory of fuzzy sets and the theory of possibility.

A more modern synthesis of fuzzy sets and their applications can be found in Kruse, Gebhardt, and Klawonn (1994), Mohammad, Vadiee, and Ross (1993), Piegat (2001), and particularly in Pedrycz and Gomide (1998). A complete introduction in Spanish is given in Galindo (2001) and Escobar (2003).

Fuzzy Sets

The original interpretation of fuzzy sets arises from a generalization of the classic concept of a subset extended to embrace the description of "vague" and "imprecise" notions. This generalization is made in the following way:

1. The membership of an element to a set becomes a "fuzzy" or "vague" concept. In the case of some elements, the issue of whether they belong to a set may not be clear.
2. The membership of an element may be measured by a degree, commonly known as the "membership degree" of that element to the set, and it takes a value in the interval [0,1] by agreement.

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, which would allow a better solution to the problem. In classic logic the membership of an element to a set is represented by 0 if it does not belong and 1 if it does, having the set $\{0,1\}$. On the other hand, in fuzzy logic this set extends to the interval [0,1]. Therefore, it could be said that fuzzy logic is an extension of 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 common sense, are by nature approximate.

Note the great potential that the use of membership degrees represents by allowing something qualitative (fuzzy) to be expressed quantitatively by means of the membership degree. A fuzzy set can be defined more formally as follows:

Definition 1.1: *A **fuzzy set** A over a universe of discourse* X *(a finite or infinite interval within which the fuzzy set can take a value) is a set of pairs*

$$A = \{\mu_A(x) / x : x \in X, \mu_A(x) \in [0,1] \in \Re\} \qquad (1.1)$$

*where $\mu_A(x)$ is called the **membership degree** of the element x to the fuzzy set A. This degree ranges between the extremes* **0** *and* **1** *of the dominion of the real numbers*:

- $\mu_A(x) = 0$ *indicates that x in no way belongs to the fuzzy set A.*
- $\mu_A(x) = 1$ *indicates that x completely belongs to the fuzzy set A.*

*Sometimes, instead of giving an exhaustive list of all the pairs that make up the set (discreet values), a definition is given for the function $\mu_A(x)$, referring to it as characteristic function or **membership function**.*

*

The universe X may be called *underlying universe* and, in a more generic way, a fuzzy set A can be considered as a function μ_A that matches each element of the universe of discourse X with its membership degree to the set *A*:

$$\mu_A(x): X \rightarrow [0,1] \tag{1.2}$$

If the membership function produces only values of the set {0,1}, then the set that it generates is not fuzzy, but "crisp" (specific, exact, or precise).

As mentioned previously, the universe of discourse X or the set of values being considered can be of two types:

- Finite or discreet universe of discourse $X = \{x_1, x_2, ..., x_n\}$, where a fuzzy set *A* can be represented by:

$$A = \mu_1 / x_1 + \mu_2 / x_2 + ... + \mu_n / x_n \tag{1.3}$$

where μ_i with $i = 1, 2, ..., n$ represents the membership degree of the element x_i. Normally the elements with a zero degree are not listed. Here, the + does not have the same significance as in an arithmetical sum but rather has the meaning of aggregation, and the / does not signify division but rather the association of both values.

- Infinite universe of discourse, where a fuzzy set A over X can be represented by:

$$A = \int \mu_A(x) / x \qquad (1.4)$$

A **linguistic label** is the word, in natural language, that expresses or identifies a fuzzy set that may or may not be formally defined. Thus, the membership function $\mu_A(x)$ of a fuzzy set A expresses the degree in which x verifies the category specified by A.

With this definition, we can assure that in our everyday life we use several linguistic labels for expressing abstract concepts such as young, old, cold, hot, cheap, expensive, and so forth.

The intuitive definition of these labels not only varies from one person to another and depends on the moment, but also it varies with the context in which it is applied. For example, a "high" person and a "high" building do not measure the same.

Example 1.1: If we express the qualitative concept "young" by means of a fuzzy set, where the x-axis represents the universe of discourse "age" (in natural whole numbers) and the y-axis represents the membership degrees in the interval [0,1], then, following Equation 1.3, the fuzzy set that represents that concept could be expressed in the following way (considering a discreet universe):

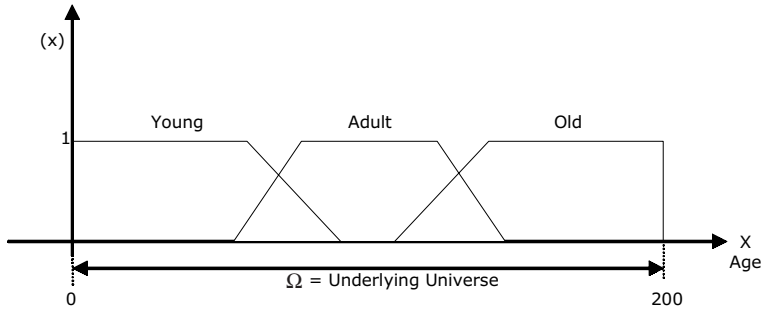
$$\text{Young} = 1/0 + ... + 1/25 + 0.9/26 + 0.8/27 + 0.7/28 + 0.6/29 + 0.5/30 + ... + 0.1/34$$

The "age" (in whole years) would be the universe of discourse of "young." The linguistic label "young" would identify this fuzzy set represented by a membership function if we consider a nondiscrete universe of discourse, of others such as "adult," "old," and so forth in this way according to Figure 1.1.

*

This logic is a multivalued logic, whose main characteristics are as follows (Zadeh, 1992):

Figure 1.1. Three linguistic labels of Example 1.1



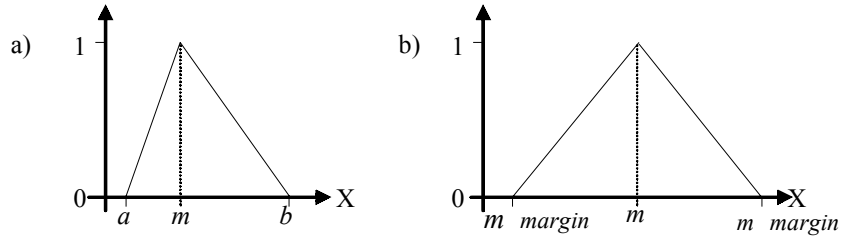
- In fuzzy logic, exact reasoning is considered as a specific case of approximate reasoning.
- Any logical system can be converted into terms of fuzzy logic.
- In fuzzy logic, knowledge is interpreted as a set of flexible or fuzzy restrictions over a set of variables.
- Inference is considered as a process of propagation of those restrictions. It is understood to be the process by which a result is reached, consequences are obtained, or one thing is deduced from another.
- In fuzzy logic, all problems are problems of degree.

From this simple concept, a complete mathematical and computing theory has been developed that facilitates the solution of certain problems (Pedrycz & Gomide, 1998). Fuzzy logic has been applied to a multitude of objectives, such as control systems, modeling, simulation, pattern recognition, information or knowledge systems (including databases, knowledge management systems, case-based reasoning systems, expert systems, etc.), computer vision, artificial intelligence, artificial life, and so forth.

Types of Membership Functions

Depending on the type of membership function, different types of fuzzy sets will be obtained. Zadeh proposed a series of membership functions that could be classified into two groups: those made up of straight lines being "linear" ones, and to the contrary the Gaussian forms, or "curved" ones.

Figure 1.2. Triangular fuzzy sets: a) general and b) symmetrical.



We will now go on to look at some types of membership functions. These types of fuzzy sets are those known as convex fuzzy sets in fuzzy set theory, with the exception of what is known as extended trapezium, which does not necessarily have to be convex, although for semantic reasons this property is always desirable.

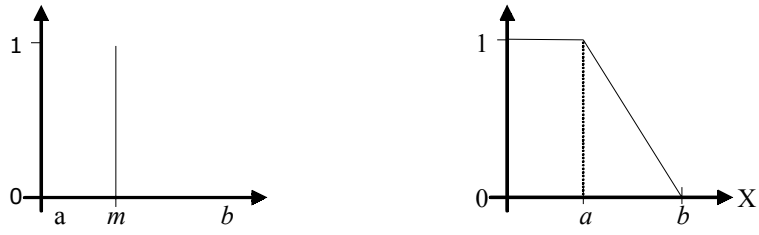
1. **Triangular (Figure 1.2)**: Defined by its lower limit *a*, its upper limit *b*, and the modal value *m*, so that $a < m < b$. We call the value *b-m margin* when it is equal to the value $m - a$.

$$A(x) = \begin{cases} 0 & \text{if } x \le a \\ (x-a)/(m-a) & \text{if } x \in (a, m] \\ (b-x)/(b-m) & \text{if } x \in (m, b) \\ 1 & \text{if } x \ge b \end{cases} \tag{1.5}$$

2. **Singleton (Figure 1.3)**: It takes the value 0 in all the universe of discourse except in the point *m,* where it takes the value 1. It is the representation of a crisp value.

$$SG(x) = \begin{cases} 0 & \text{if } x \ne m \\ 1 & \text{if } x = m \end{cases} \tag{1.6}$$

Figure 1.3. Singleton fuzzy set (left) and Figure 1.4. L fuzzy set (right)



3. **L Function (Figure 1.4)**: This function is defined by two parameters **a** and **b**, in the following way:

$$L(x) = \begin{cases} 1 & \text{if } x \le a \\ \dfrac{a-x}{b-a} & \text{if } a < x \le b \\ 0 & \text{if } x > b \end{cases} \tag{1.7}$$

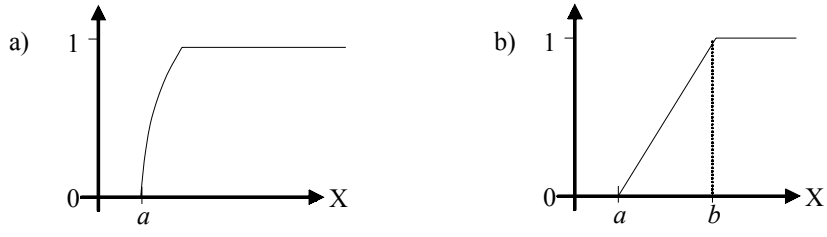
4. **Gamma Function (Figure 1.5)**: It is defined by its lower limit a and the value $k > 0$. Two definitions:

$$\Gamma(x) = \begin{cases} 0 & \text{if } x \le a \\ \dfrac{k(x-a)^2}{1+k(x-a)^2} & \text{if } x > a \end{cases} \tag{1.8}$$

$$\Gamma(x) = \begin{cases} 0 & \text{if } x \le a \\ \dfrac{k(x-a)^2}{1+k(x-a)^2} & \text{if } x > a \end{cases} \tag{1.9}$$

- This function is characterized by rapid growth starting from a.
- The greater the value of k, the greater the rate of growth.
- The rate of growth is greater in the first definition than in the second.
- Horizontal asymptote in 1.
- The gamma function is also expressed in a linear way (Figure 1.5 b):

Figure 1.5. Gamma fuzzy sets: a) general and b) linear



$$\Gamma(x)=\begin{cases}0 & \text{if } x\le a\\ \dfrac{x-a}{b-a} & \text{if } a<x\le b\\ 1 & \text{if } x>b\end{cases} \tag{1.10}$$

5. **Trapezoid Function (Figure 1.6)**: Defined by its lower limit **a** and its upper limit **d,** and the lower and upper limits of its nucleus, **b** and **c** respectively.

$$T(x)=\begin{cases}0 & \text{if } (x\le a) \text{ o } (x\ge d)\\ (x-a)/(b-a) & \text{if } x\in(a,b]\\ 1 & \text{if } x\in(b,c)\\ (d-x)/(d-c) & \text{if } x\in(b,d)\end{cases} \tag{1.11}$$

6. **S Function (Figure 1.7)**: Defined by its lower limit a, its upper limit b, and the value m or point of inflection so that $a<m<b$**.** A typical value is $m=(a+b)/2$**.** Growth is slower when the distance $a-b$ increases.

$$S(x)=\begin{cases}0 & \text{if } x\le a\\ 2\{(x-a)/(b-a)\}^2 & \text{if } x\in(a,m]\\ 1-2\{(x-a)/(b-a)\}^2 & \text{if } x\in(m,b)\\ 1 & \text{if } x\ge b\end{cases} \tag{1.12}$$

Figure 1.6. Trapezoidal fuzzy set (left) and Figure 1.7. S fuzzy set (right)

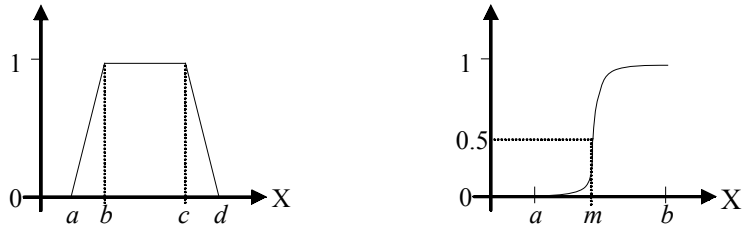


Figure 1.8. Gaussian fuzzy set (left) and Figure 1.9. Pseudo-exponential fuzzy set (right)

7. **Gaussian Function (Figure 1.8)**: This is the typical Gauss bell, defined by its midvalue **m** and the value of $k > 0$. The greater k is**,** the narrower the bell.

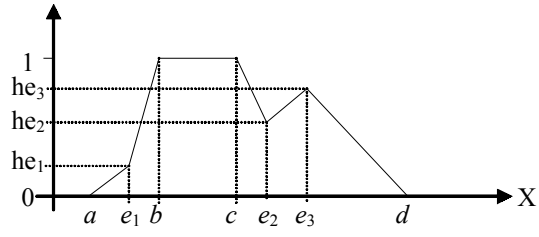
$$G(x) = e^{-k(x-m)^2} \tag{1.13}$$

8. **Pseudo-Exponential Function (Figure 1.9)**: Defined by its midvalue m and the value $k > 1$. As the value of k increases, the rate of growth increases, and the bell becomes narrower.

$$P(x) = \frac{1}{1+k(x-m)^2} \tag{1.14}$$

9. **Extended Trapezoid Function (Figure 1.10)**: Defined by the four values of a trapezoid (a, b, c, d) and a list of points between a and b and/

Figure 1.10. Extended trapezoidal fuzzy set



or between *c* and *d*, with its membership value (height) associated to each of these points. (e_i, he_i).

Comments:

- In general, the **trapezoid** function adapts quite well to the definition of any concept, with the advantage that it is easy to define, easy to represent, and simple to calculate.
- In specific cases, the **extended trapezoid** is very useful. This allows greater expressiveness through increased complexity.
- In general, the use of a **more complex function does not give increased precision**, as we must keep in mind that we are defining a **fuzzy** concept.
- Concepts that require a **nonconvex function** can be defined. In general, a nonconvex function expresses **the union** of two or more concepts whose representation is convex.

In fuzzy control, for example, the aim is to express the notions of increase, decrease, and approximation, and in order to do so, the types of membership functions previously mentioned are used. The membership functions Gamma and S would be used to represent linguistic labels such as "tall" or "hot" in the dominion of height and temperature. Linguistic labels such as "small" and "cold" would be expressed by means of the L function. On the other hand, approximate notions are sometimes difficult to express with one word. In the dominion of temperature, the label would have to be "comfortable," which would be expressed by means of the triangle, trapezoid, or the Gaussian function.

Membership Function Determination

If the system uses badly defined membership functions it will not work well; therefore, these functions must be carefully defined. The membership functions can be calculated in several ways. Which method is chosen will depend on the application in question, the manner in which the uncertainty is to be represented, and how this is to be measured during the experiments. The following points give a brief summary of some of these methods (Pedrycz & Gomide, 1998).

1. **Horizontal Method**: It is based on the answers of a group of **N** "experts"
   - The question takes the following form: "Can $\boldsymbol{x}$ be considered compatible with the concept $\boldsymbol{A}$?"
   - Only "Yes" and "No" answers are acceptable, so

     $A(x) = (\text{Affirmative Answers}) / \text{N}$

2. **Vertical Method**: The aim is to build several α-cuts (Definition 1.5), for which several values are selected for α.
   - Now the question that is formulated for these predetermined α values is as follows: "Can the elements of **X** that belong to $\boldsymbol{A}$ to a degree that is not inferior to α be identified?"
   - From these α-cuts, the fuzzy set $\boldsymbol{A}$ can be identified, using the so-called Identity Principle or Representation Theorem (Definition 1.6).

3. **Pair Comparison Method** (Saaty, 1980): Supposing that we already have the fuzzy set A, over the universe of discourse X of n values ($x_1, x_2, \ldots, x_n$) we could calculate the Reciprocal Matrix $M = [a_{hi}]$, a square matrix $n \times n$ with the following format:

$$M = \begin{bmatrix} \frac{A(x_1)}{A(x_1)} & \frac{A(x_1)}{A(x_2)} & \cdots & \frac{A(x_1)}{A(x_n)} \\ \frac{A(x_2)}{A(x_1)} & \frac{A(x_2)}{A(x_2)} & \cdots & \frac{A(x_2)}{A(x_n)} \\ \cdots & \cdots & \frac{A(x_i)}{A(x_j)} & \cdots \\ \frac{A(x_n)}{A(x_1)} & \frac{A(x_n)}{A(x_2)} & \cdots & \frac{A(x_n)}{A(x_n)} \end{bmatrix}$$

- This matrix has the following properties:
  - The principal diagonal is always 1.
  - Property of Reciprocity: $(a_{hi}, a_{ih})=1$.
  - Transitive Property: $(a_{hi}, a_{ik}) = a_{hk}a$.

- If what we wish to calculate is the fuzzy set A, the process is reversed:
  - The matrix M is calculated.
  - A is calculated from M.

- In order to calculate M, the level of priority or the highest membership degree of a pair of values is numerically quantified: x_i with respect to x_j.
  - The number of comparisons: $n\,(n\text{-}1)\,/\,2$.
  - Transitivity is difficult to achieve (the eigenvalue of the matrix is used to measure the consistency of the data, so that if it is very low, the experiments should be repeated).

4. **Method Based on Problem Specification**: This method requires a numerical function that should be approximate. The error is defined as a fuzzy set that measures the quality of the approximation.

5. **Method Based on the Optimization of Parameters**: The shape of a fuzzy set A depends on some parameters, indicated by the vector p, which is represented by $A(x; p)$.

- Some experimental results in the form of pairs are needed (element, membership degree) (E_k, G_k) with $k = 1, 2...., N$.
- The problem consists of optimizing the vector p, for example, minimizing the cuadratic error:

$$\min_p \sum_{k=1}^{N} [G_K - A(E_K; p)]^2$$

6. **Method Based on Fuzzy Clustering**: This method is based on clustering the objects of the universe in overlapping groups whose levels of membership to each group are considered as fuzzy degrees. There are several Fuzzy Clustering algorithms, but the most widely used is the algorithm of "fuzzy isodata" (Bezdek, 1981).

Concepts About Fuzzy Sets

A series of concepts regarding fuzzy sets is defined, which allow us to deal with and compare fuzzy sets. In this section, the most important concepts about fuzzy sets are defined.

Definition 1.2: *Let A and B be two fuzzy sets over* X. *Then A is* ***equal*** *to B if*

$$A = B \Leftrightarrow \forall\, x \in \mathrm{X},\ \mu_A(x) = \mu_B(x) \qquad (1.15)$$

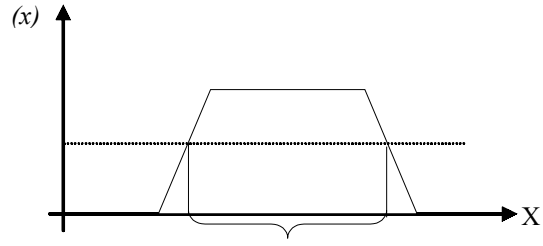
*

Definition 1.3: *Taking two fuzzy sets A and B over* X, *A is said to be* ***included*** *in B if*

$$A \subseteq B \Leftrightarrow \forall\, x \in \mathrm{X},\ \mu_A(x) \leq \mu_B(x) \qquad (1.16)$$

*

Figure 1.11. α-cut in a trapezium



Definition 1.4: *The **support** of a fuzzy set A defined over X is a subset of that universe that complies with*:

$$\text{Supp}(A) = \{x \in X, \mu_A(x) > 0\} \quad (1.17)$$

*

Definition 1.5: *The **α-cut** of a fuzzy set A, denoted by A_α (Figure 1.11) is a classic subset of elements in X, whose membership function takes a greater or equal value to any specific α value of that universe of discourse that complies with*:

$$A_\alpha = \{x : x \in X, \mu_A(x) \geq \alpha, \alpha \in [0,1]\} \quad (1.18)$$

*

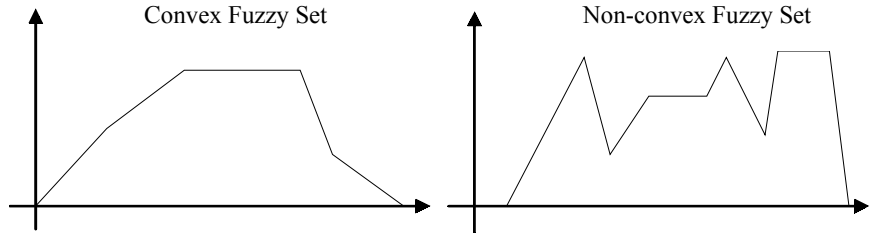
Definition 1.6: *The **Representation Theorem** states that any fuzzy set A can be obtained from the union of its α-cuts.*

$$A = \bigcup_{\alpha \in [0,1]} \alpha A_\alpha \quad (1.19)$$

*

Definition 1.7: *By using the Representation Theorem, the concept of **convex fuzzy set** (Figure 1.12) can be established as that in which all the α-cuts are convex*:

Figure 1.12. Examples of convex (left) and nonconvex (right) fuzzy sets



$$\forall x, y \in X, \forall \lambda \in [0,1]: \mu_A(\lambda \cdot x + (1 - \lambda) \cdot y) \geq \min(\mu_A(x), \mu_A(y)) \quad (1.20)$$

*

This definition means that any point situated between two other points will have a higher membership degree than the minimum of these two points.

Definition 1.8: *A **concave fuzzy set** (Figure 1.12) complies with*:

$$\forall x, y \in X, \forall \lambda \in [0,1]: \mu_A(\lambda \cdot x + (1 - \lambda) \cdot y) \leq \min(\mu_A(x), \mu_A(y)) \quad (1.21)$$

*

Definition 1.9: *The **kernel** of a fuzzy set A, defined over X, is a subset of that universe that complies with*:

$$\text{Kern}(A) = \{x \in X, \mu_A(x) = 1\} \quad (1.22)$$

*

Definition 1.10: *The **height** of a fuzzy set A, defined over X, is defined as*:

$$\text{Hgt}(A) = \sup_{x \in X} \mu_A(x) \quad (1.23)$$

*

Definition 1.11: *A fuzzy set A is* ***normalized*** *if and only if*

$$\exists x \in X, \mu_A(x) = \text{Hgt}(A) = 1 \quad (1.24)$$

*

Definition 1.12: *The* ***cardinality*** *of a fuzzy set A, with finite universe X, is defined as*:

$$\text{Card}(A) = \sum_{x \in X} \mu_A(x) \quad (1.25)$$

If the universe is infinite, the addition must be changed for an integral defined within the universe.

*

Fuzzy Set Operations

The fact that the theory of fuzzy sets generalizes the theory of classic sets means that the fuzzy sets allow operations of union, intersection, and complement. These and other operations can be found in Petry (1996) and Pedrycz and Gomide (1998), such as *concentration* (the square of the membership function), *dilatation* (finding the square root of the membership function), *contrast intensification* (see Equation 7.1), and *fuzzification* (see Equation 7.2), which can be used when linguistic hedges such as "very" or "not very" are used.

Union and Intersection: t-conorms and t-norms

Definition 1.13: *If A and B are two fuzzy sets over a universe of discourse X, the membership function of the* ***union*** *of the two sets* $A \cup B$ *is expressed by*

$$\mu_{A\cup B}(x) = f(\mu_A(x), \mu_B(x)), x \in X \quad (1.26)$$

where f is a t-conorm (Schweizer & Sklar, 1983).

*

Definition 1.14: *If A and B are two fuzzy sets over a universe of discourse* X*, the membership function of the* ***intersection*** *of the two sets* $A \cap B$*, is expressed by*

$$\mu_{A\cap B}(x) = g(\mu_A(x), \mu_B(x)), x \in X \quad (1.27)$$

where g is a t-norm (Schweizer & Sklar, 1983).

*

Both s-norms and t-norms establish **generic models** respectively for the operations of **union** and **intersection,** which must comply with certain basic properties (commutative, associative, monotonicity, and border conditions). They are concepts derived from Menger (1942) and Schweizer and Sklar (1983), and have been looked at in-depth more recently (Butnario & Klement, 1993).

Definition 1.15: *Triangular Norm,* ***t-norm****: binary operation, t:* $[0,1]^2 \rightarrow [0,1]$ *that complies with the following properties*:

1. Commutativity: $x\ t\ y = y\ t\ x$.
2. Associativity: $x\ t\ (y\ t\ z) = (x\ t\ y)\ t\ z$.
3. Monotonicity: If $x \leq y$, and $w \leq z$ then $x\ t\ w \leq y\ t\ z$.
4. Boundary conditions: $x\ t\ 0 = 0$, and $x\ t\ 1 = x$.

*

Definition 1.16: *Triangular Conorm,* ***t-conorm*** *or* ***s-norm****: Binary operation, s:* $[0,1]^2 \rightarrow [0,1]$ *that complies with the following properties*:

1. Commutativity: $x\ s\ y = y\ s\ x$.
2. Associativity: $x\ s\ (y\ s\ z) = (x\ s\ y)\ s\ z$.
3. Monotonicity: If $x \leq y$, and $w \leq z$ then $x\ s\ w \leq y\ s\ z$.
4. Boundary conditions: $x\ s\ 0 = x$, and $x\ s\ 1 = 1$.

*

The most widely used of this type of functions are the **t-norm of the Minimum** and the **t-conorm or s-norm of the Maximum** as they have retained a large number of the properties of the Boolean operators, such as the property of idempotency ($x\ t\ x = x$; $x\ s\ x = x$). In Figure 1.13 we can see the intersection and union, using respectively the minimum and maximum, of two trapezoid fuzzy sets.

There is an extensive set of operators, called **t-norms (triangular norms)** and **t-conorms (triangular co-norms)**, that can be used as connectors for modeling the intersection and union respectively, as detailed in Dubois and Prade (1980), Yager (1980), Predycz and Gomide (1998), and Piegat (2001). The most important are shown in Tables 1.1 and 1.2.

A relationship exists between t-norms (t) and t-conorms (s). It is an extension of De Morgan's Law:

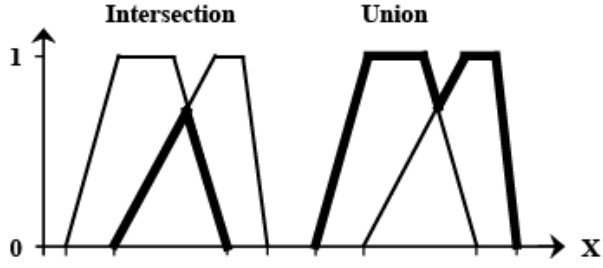
$$x\ s\ y = 1 - (1 - x)\ t\ (1 - y)$$
$$x\ t\ y = 1 - (1 - x)\ s\ (1 - y) \qquad (1.28)$$

When a t-norm or a t-conorm complies with this property, it is said to be **conjugated** or **dual**.

T-norms and t-conorms cannot be ordered from larger to smaller. However, it is easy to identify the largest and the smallest t-norm and t-conorm:

- Largest t-norm: Minimum Function.
- Smallest t-norm: Drastic Product.
- Largest t-conorm: Drastic Sum.
- Smallest t-conorm: Maximum Function.

Figure 1.13. Intersection (minimum) and union (maximum)



Note that if two fuzzy sets are convex, their intersection (but not necessarily their union) will also be.

Complements or Negations

The notion of the complement can be constructed by using the concept of strong negation (Trillas, 1979).

Definition 1.17: *A function* C: [0,1] → [0,1] *is a strong* ***negation*** *if it fulfils the following conditions*:

1. *Boundary conditions:* C(0) = 1 *and* C(1) = 0.
2. *Involution:* C(C(x)) = x.
3. *Monotonicity:* C *is nonincreasing.*
4. *Continuity:* C *is continuous.*

*

Although several types of operators satisfy such properties or relaxed versions of them, we will mainly use Zadeh's version of the complement (1965). Thus, for a fuzzy set *A* in the universe of discourse *X,* the membership function of the complement, denoted by $\neg A$, or by $\overline{A}$ is shown as

$$\mu_{\neg A}(x) = 1 - \mu_A(x), x \in X \quad (1.29)$$

Table 1.1. t-norms functions: f(x,y) = x t y

t-norms	Expression
Minimum	$f(x,y) = \min(x,y)$
Product (Algebraic)	$f(x,y) = x \bullet y$
Drastic Product	$f(x,y) = \begin{cases} xy, & \text{if } y = 1 \\ y, & \text{if } x = 1 \\ 0, & \text{otherwise} \end{cases}$
Bounded Product (bounded difference)	$f(x,y) = \max[0,(1+p)(x+y-1) - pxy]$, where $p \geq -1$
Hamacher Product	$f(x,y) = \dfrac{xy}{p+(1-p)(x+y-xy)}$, where $p \geq 0$
Yager Family	$f(x,y) = 1 - \min(1, \left[(1-x)^p + (1-y)^p\right]^{1/p})$, where $p > 0$
Dubois-Prade Family	$f(x,y) = \dfrac{xy}{\max(x,y,p)}$, where $0 \leq p \leq 1$
Frank Family	$f(x,y) = \log_p \left(1 + \dfrac{(p^x - 1)(p^y - 1)}{p-1}\right)$, where $p > 0;\ p \neq 1$
Einstein Product	$f(x,y) = \dfrac{xy}{1+(1-x)+(1-y)}$
Others	$f(x,y) = \dfrac{1}{1+\left[((1-x)/x)^p + ((1-y)/y)^p\right]^{1/p}}$, where $p > 0$ $f(x,y) = \dfrac{1}{\left[1/x^p + 1/y^p - 1\right]}$ $f(x,y) = \left[\max(0, x^p + y^p - 1)\right]^{1/p}$

Table 1.2. s-norms functions: f(x,y) = x s y

t-conorms or s-norms	Expression
Maximum	$f(x,y) = \max(x,y)$
Sum-Product (Algebraic sum)	$f(x,y) = x + y - xy$
Drastic sum	$f(x,y) = \begin{cases} x, & \text{if } y = 0 \\ y, & \text{if } x = 0 \\ 1, & \text{otherwise} \end{cases}$
Bounded sum	$f(x,y) = \min(1, x + y + pxy)$, where $p \geq 0$
Einstein sum	$f(x,y) = \frac{x + y}{1 + xy}$
Sugeno Family	$f(x,y) = \min(1, x + y + p - xy)$, where $p \geq 0$
Yager Family	$f(x,y) = \min(1, \left[x^p + y^p\right]^{1/p})$, where $p > 0$
Dubois-Prade Family	$f(x,y) = \frac{(1-x)(1-y)}{\max(1-x,\ 1-y,\ p)}$, where $p \in [0,1]$
Frank Family	$f(x,y) = \log_p\left(1 + \frac{(p^{1-x} - 1)(p^{1-y} - 1)}{p - 1}\right)$ where $p > 0;\ p \neq 1$
Others	$f(x,y) = \frac{x + y - xy - (1-p)xy}{1 - (1-p)xy}$, where $p \geq 0$ $f(x,y) = 1 - \max(0, \left[(1-x)^p + (1-y)^p - 1\right]^{1/p}$, where $p > 0$ $f(x,y) = \frac{1}{1 - \left[x/(1-x)^p + y/(1-y)^p\right]^{1/p}}$, where $p > 0$ $f(x,y) = \frac{1}{1 - \left[1/(1-x)^p + 1/(1-y)^p - 1\right]^{1/p}}$, where $p > 0$

Comparison Operations on Fuzzy Sets

The fuzzy sets, defined by using a membership function, can be compared in different ways. We will now list several methods used to compare fuzzy sets (Pedrycz & Gomide, 1998).

Distance Measures

A distance measure considers a distance function between the membership functions of two fuzzy sets in the same universe. In such a way it tries to indicate the proximity between the two fuzzy sets. In general, the distance between *A* and *B,* defined in the same universe of discourse, can be defined by using the Minkowski Distance:

$$d(A,B) = \left[\int_X |A(x) - B(x)|^p \, dx \right]^{1/p} \tag{1.30}$$

where $p \geq 1$ and we assume that the integral exists. Several specific cases are typically used:

1. Hamming Distance ($p = 1$):

$$d(A,B) = \int_X |A(x) - B(x)| \, dx \tag{1.31}$$

2. Euclidean Distance ($p = 2$):

$$d(A,B) = \left[\int_X |A(x) - B(x)|^2 \, dx \right]^{1/2} \tag{1.32}$$

For a discrete universe of discourses, integration is replaced with sum. The more similar are the fuzzy sets; the distance between them is smaller. Therefore, it is convenient to normalize the function of distance, denoted by $d_n(A, B)$, and use this form to express the similarity as a direct complementation: $1 - d_n(A, B)$.

Equality Indexes

This is based on the logical expression of equality; that is, two sets A and B are equals if $A \subset B$ and $B \subset A$. In fuzzy sets, a certain degree of equality can be found. With that the following expression is defined:

$$(A \equiv B)(x) = \frac{[A(x)\varphi B(x)] \wedge [B(x)\varphi A(x)] + [\overline{A}(x)\varphi \overline{B}(x)] \wedge [\overline{B}(x)\varphi \overline{A}(x)]}{2} \tag{1.33}$$

where the conjunction ($\wedge$) is modeled on the minimum operation, and the inclusion is represented by the operator φ (phi), induced by a continuous t-norm t:

$$A(x)\ \varphi\ B(x) = \sup_{c \in [0,1]} [A(x)\ t\ c \leq \mathrm{B(x)}] \tag{1.34}$$

Taking the Selected Product with p = 0 as an example:

$$A(x)\varphi B(x) = \begin{cases} 1 & \text{if } A(x) < B(x) \\ B(x) - A(x) + 1 & \text{if } A(x) \geq B(x) \end{cases}$$

$$\Rightarrow (A \equiv B)(x) = \begin{cases} A(x) - B(x) + 1 & \text{if } A(x) < B(x) \\ B(x) - A(x) + 1 & \text{if } A(x) \geq B(x) \end{cases} \tag{1.35}$$

Three basic methods can be used to obtain a single value ($\forall x \in X$):

- Optimistic Equality Index:

$$(A \equiv B)_{opt} = \sup_{x \in X} (A \equiv B)(x) \tag{1.36}$$

- Pessimistic Equality Index:

$$(A \equiv B)_{pes} = \sup_{x \in X} (A \equiv B)(x) \tag{1.37}$$

- Medium Equality Index:

$$(A \equiv B)_{avg} = \left({}^{1}\!/_{Card(x)} \right) \int_{x} (A \equiv B)(x)dx \tag{1.38}$$

Thus, the following relationship is satisfied:

$$(A \equiv B)_{pes} \leq (A \equiv B)_{avg} \leq (A \equiv B)_{opt} \tag{1.39}$$

Possibility and Necessity Measures

These use the fuzzy sets as possibility distributions where $A(x)$ measures the possibility of the required figure being X (Zadeh, 1978), that is, the possibility of value A being equal to value B. It measures the extent to which A and B superpose each other, denoted as Poss(A, B) and defined as:

$$\text{Poss}\ (A, B) = \sup_{x \in X} \left[\min \left(A(x), B(x) \right) \right] \tag{1.40}$$

The necessity measure describes the degree to which B is included in A and is denoted by Nec(A, B):

$$\text{Nec}(A, B) = \inf_{x \in X} \left[\max \left(A(x), 1 - B(x) \right) \right] \tag{1.41}$$

In Figures 1.14 and 1.15 we can see graphically how these measurements for two concrete fuzzy sets are calculated.

As we have already mentioned, the measurement of possibility measures the extent to which measurement A is superposed on B. In the light of the previously introduced definitions, it can be stated that Poss (A, B) = Poss (B, A). On the other hand, the measurement of necessity is asymmetrical, Nec $(A, B) \neq$ Nec (B, A). However, the following relation is fulfilled:

$$\text{Nec } (A, B) + \text{Poss } (\overline{A}, B) = 1 \tag{1.42}$$

Example 1.2: Let there be a motorway where the speed limit is 100 km/h. In this specific context, the concept of high speed could be represented by a fuzzy set B, defined in the speed space, as is shown in Figure 1.16. The membership function for a vehicle moving at a speed of around 80 km/h is a triangular shape, denoted by the fuzzy set A. In this scenario several points must be considered. To what extent is the vehicle traveling at a high speed? What degree of high speed is around 80km/h? Based on the interpretation of the concept of the possibility measurement, we could quantify the extent to which A is dependent on B, that is, the extent to which the speed at which the vehicle travels (around 80 km/h) is high:

Poss (around 80km/h, high speed) = Poss(A, B) = Poss (B, A) = 0.6

*

Other equivalences are:

$$\text{Poss } (A \cup B, C) = \max \{\text{Poss } (A, C), \text{Poss } (B, C)\} \tag{1.43}$$

$$\text{Poss } (A \cap B, C) = \min \{\text{Nec } (A, C), \text{Nec } (B, C)\} \tag{1.44}$$

Figure 1.14. General illustration of the Poss(A,B) concept using the minimum t-norm

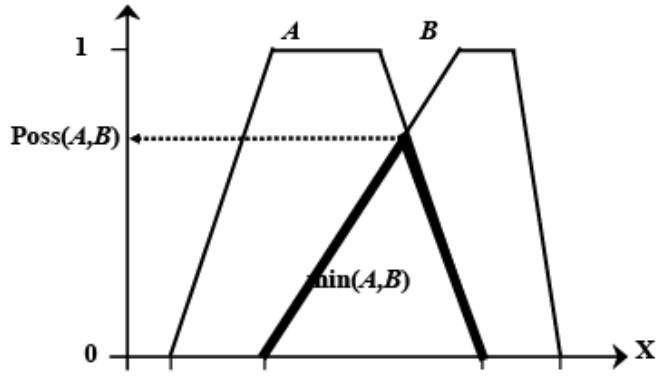
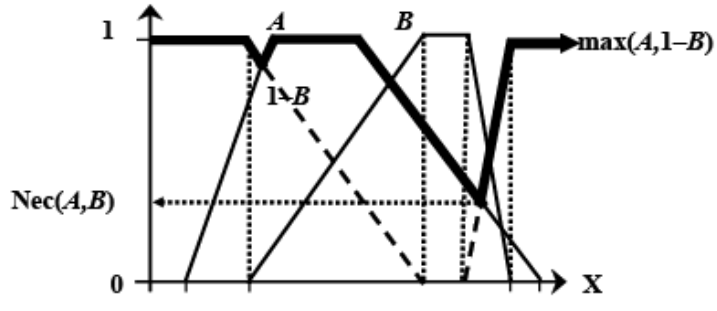


Figure 1.15. General illustration of the Nec(A,B) concept using the maximum t-conorm



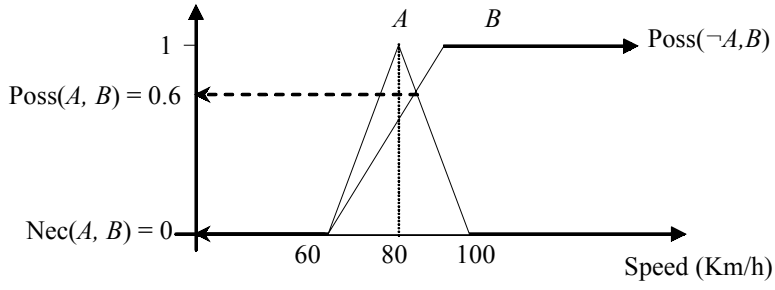
The generalization of the possibility and necessity measurements use triangular t-norms or t-conorms instead of min and max functions, respectively.

If the concept is extended, the possibility of a fuzzy set A (or of a possibility distribution) in the universe X can be defined as:

$$\Pi(A) = \text{Poss}(A, \text{X}) = \sup_{x \in \text{X}} \left[\min\left(A(x), 1\right) \right] = \sup_{x \in \text{X}} A(x) \qquad (1.45)$$

This possibility measures whether a determined event (the fuzzy set A) is possible in universe X. It would not measure uncertainty, because if $\Pi(A) = 1$, we know that event A is possible, but:

Figure 1.16. Illustration of Example 1.2 about possibility and necessity



- if $\Pi(\overline{A}) = 1$, then the certainty is indeterminate.
- if $\Pi(\overline{A}) = 0$, then the occurrence of A is certain.

Therefore, the following two equalities are always satisfied:

- $\Pi(X) = 1$ (possibility of an element of the universe).
- $\Pi(\phi) = 0$ (possibility of an element *not* in the universe).

Similarly, the necessity of a fuzzy set $N(A)$ in X can be defined, and then we can set some equivalences of possibility and necessity:

$$N(A) = \inf_{x \in X} \{A(x)\} = 1 - \sup_{x \in X} \{1 - A(x)\} = 1 - \Pi(\neg A): N(A) = 1 - \Pi(\neg A) \quad (1.46)$$

$$\Pi(A) = \sup_{x \in X} \{A(x)\} = 1 - \inf_{x \in X} \{1 - A(x)\} = 1 - N(\neg A): \Pi(A) = 1 - N(\neg A) \quad (1.47)$$

These equivalences explain why the necessity complements the information about the certainty of event A:

- The greater $N(A)$, the smaller the possibility of opposite event $(\neg A)$.
- The greater $\Pi(A)$, the smaller the necessity of the opposite event $(\neg A)$.

- $N(A) = 1 \Leftrightarrow \neg A$ is totally impossible (if an event is totally necessary, then the opposite event is totally impossible).
- $\Pi(A) = 1 \Leftrightarrow \neg A$ is not necessary at all $N(\neg A) = 0$ (if an event is totally possible, then the opposite event cannot be necessary in any way).
- $N(A) = 1 \Rightarrow \Pi(A) = 1$ (if A is a totally necessary event, then it must be totally possible). Note that the opposite is not satisfied.
- $A \subseteq B \Rightarrow N(A) \leq N(B)$ and $\Pi(A) \leq \Pi(B)$.

Compatibility Measures

This comparison operation measures the extent to which a certain fuzzy set is compatible with another (defined in the same space). The result is not a single number but instead a fuzzy set defined in the interval unit, [0, 1], $Comp(B, A)(u) = \sup_{u=A(x)} \{B(x)\}$, $u \in [0, 1]$ known as the Fuzzy Set of Compatibility. Therefore, the compatibility of B with A can be defined as:

$$Comp(B, A)(u) = \sup_{u=A(x)} \{B(x)\}, u \in [0,1] \quad (1.48)$$

Set B can be seen as a "fuzzy value" and set A as a "fuzzy concept." Therefore, $Comp(B, A)$ measures the compatibility with which B is A.

Example 1.3: Let B be the value "approximately 70 years" and A be the concept "very old." The fuzzy set $Comp(B, A)$ is represented in Figure 1.17 and the fuzzy set $Comp(A, B)$ in Figure 1.18.

*

The compatibility measurement has the following properties:

- It measures the degree to which B can fulfill concept A. That degree will be greater the more similar the fuzzy set $Comp(B, A)$ is to the singleton "1" value (maximum compatibility).
- Supposing A is a normalized fuzzy set: $Comp(A, A)(u) = u$ (linear membership function).

Figure 1.17. Example 1.3: Illustration of Comp(B, A)

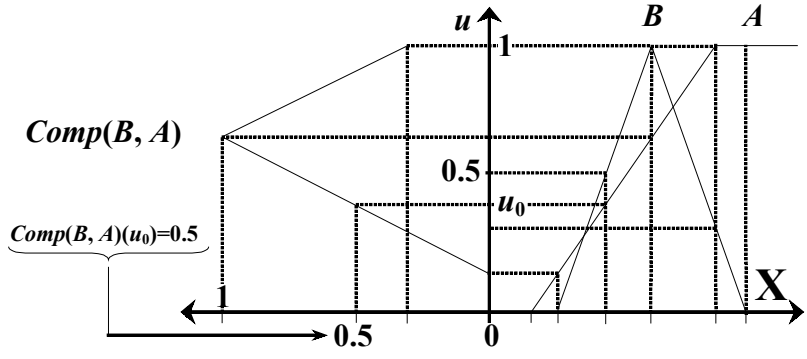
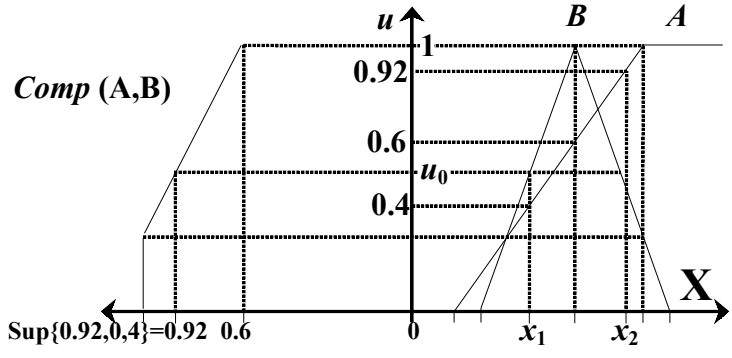


Figure 1.18. Example 1.3: Illustration of Comp(A, B)



- If A is not normalized, the function will be the same between 0 and the height of set A: If $u >$ Height(A), $Comp(A, A)(u)$= indeterminate (0).
- If B is a number x ("singleton" fuzzy set), the result will also be another "singleton" in the $A(x)$ value:

$$Comp(B, A)(u) = \begin{cases} 1, & \text{if } u = A(x) \\ 0, & \text{otherwise} \end{cases} \tag{1.49}$$

If B is not normalized, the result will not be either, its height being the same as that of set B.

- If Support (A) $\cap$ Support(B) = ϕ, then

$$Comp\ (B, A)(u) = Comp\ (A, B)(u) = \begin{cases} 1, & \text{if } u = 0 \text{ (minimum compatibility)} \\ 0, & \text{otherwise} \end{cases} \tag{1.50}$$

If B is not normalized, the result will not be either, and its height will be the same as that height of B, for $u = 0$.

In order to have a clearer vision of what this measurement means, we can look at the examples shown in Figures 1.19, 1.20, and 1.21, where several typical cases can be seen. We can conclude that fuzzy set B is more compatible with another A the closer that $Comp(B,A)$ is to 1 and the further it is from 0 (the less area it has). So, in Figure 1.20, we see that $B2$ is more compatible with A than $B1$.

Other properties of the compatibility measurement are:

- The compatibility is asymmetrical: $Comp(B, A) \neq Comp(A, B)$.
- $B \subset B' \Rightarrow Comp(B, A)\ (u) \leq Comp(B', A)\ (u) = u$.
- $B(x) = \{1, \forall\ x \in X\} \Rightarrow Comp(B, A)(u) = \{1, \forall\ u \in [0,1]\}$.
- $B(x) = \{0, \forall\ x \in X\} \Rightarrow Comp(B, A)(u) = \{0, \forall\ u \in [0,1]\}$.
- $B \subset A$ and they are normalized $\Rightarrow Comp(B, A)\ (0) = 0$ and $Comp(B, A)\ (1) = 1$. Of course there may be more points with 0 and 1 compatibility.
- $A \subset B$ and they are normalized $\Rightarrow Comp(B, A)\ (1) = 1$ and $Comp(B, A)\ (u) = 0 \Leftrightarrow u = 0$.
- The possibility and necessity measurements between A and B are included in the support of $Comp(B,A)$. For example, if B is an interval (Figure 1.22), then:

$$\text{Poss}(B, A) = \text{Sup}\{\text{Support}(Comp(B, A))\}$$
$$\text{Nec}(B, A) = \text{Inf}\{\text{Support}(Comp(B, A))\}$$

Figure 1.19. Three sets (B1, B2 and B3) with the same shape placed in different positions and compared to A

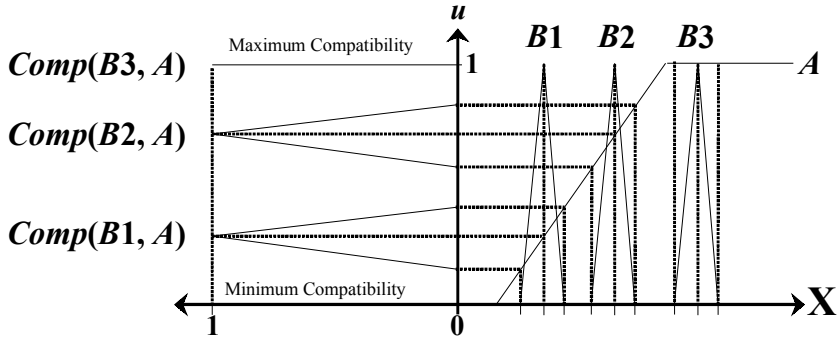


Figure 1.20. Two sets (B1 and B2) with the same height as A at one point, but with its nucleus included in A's nucleus (B2) or not included in A (B1).

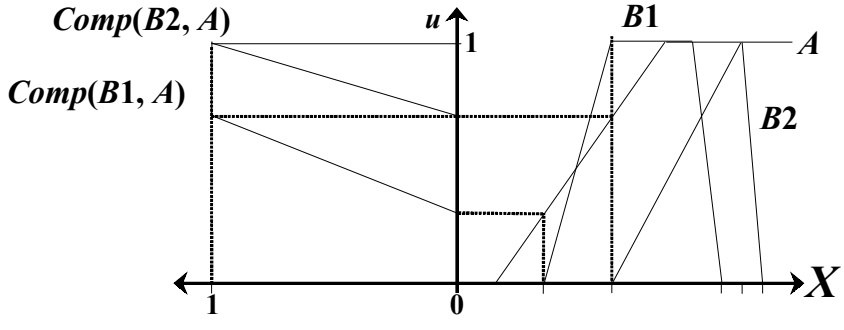


Figure 1.21. Two triangular sets (A and B) compared to each other

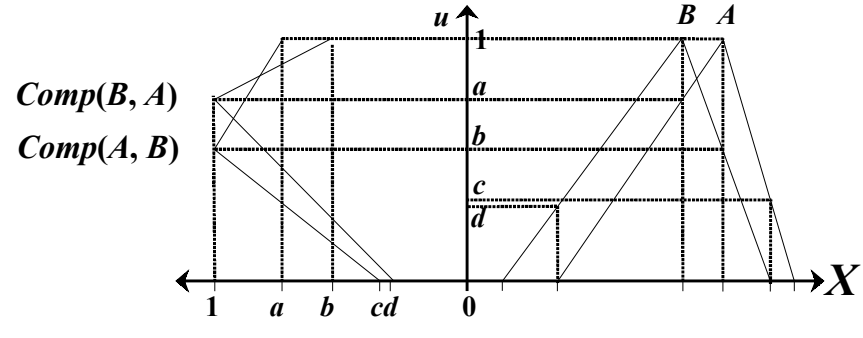
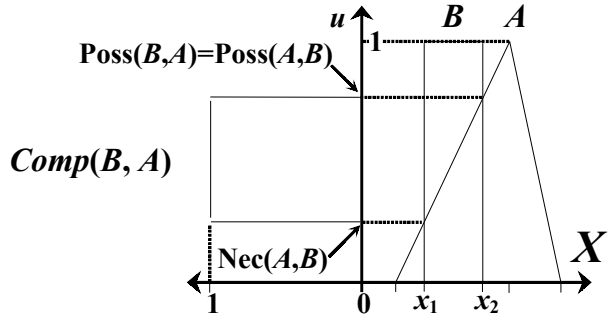


Figure 1.22. Possibility and necessity measurements included in the compatibility support



Fuzzy Relations

A classic relation between two universes X and Y is a subset of the Cartesian X´Y product. Like the classic sets, classic relation can be described by using a characteristic function. In the same way, a fuzzy relation *R* is a fuzzy set of tuples, where this characteristic function is extended to the interval. In the event of a binary relation, the tuple has two values.

Definition 1.18: *Let* U *and* V *be two infinite (continuous) universes and* m_R: $U \times V \rightarrow [0, 1]$. *Then, a* ***fuzzy relation*** *R is defined as*

$$R = \int_{U \times V} \mu_R (u, v) / (u, v) \qquad (1.51)$$

*

The function μ_R may be used as a similarity or proximity function. It is important to stress that not all functions are relations and not all relations are functions. Fuzzy relations generalize the generic concept of relation by allowing the notion of partial belonging (association) between points in the universe of discourse.

Example 1.4: Take as an example the fuzzy relation in $\Re^2$ (binary relation), "approximately equal," with the following membership function in $X \subset \Re$, with X^2

$= \{1, 2, 3\}^2$: 1/(1, 1) + 1/(2, 2) + 1(3, 3) + 0.8/(1, 2) + 0.8/(2, 3) + 0.8/(2, 1) + 0.8/(3, 2) + 0.3/(1, 3) + 0.3/(3, 1). This fuzzy relation may be defined as:

$$\text{x approximately equal to y}: \quad R(\text{x}, \text{y}) = \begin{Bmatrix} 1, & \text{if } \text{x} = \text{y} \\ 0.8, & \text{if } |\text{x} - \text{y}| = 1 \\ 0.3, & \text{if } |\text{x} - \text{y}| = 2 \end{Bmatrix}$$

where $\text{x}, \text{y} \in \Re$.

When the universe of discourse is finite, a matrix notation can be quite useful to represent the relation. So this example would be shown as:

X	X		
	1	0.8	0.3
	0.8	1	0.8
	0.3	0.8	1

Operations and Compositions of Fuzzy Relations

Definitions of basic operations with fuzzy relations are closely linked to operations of fuzzy sets (see the "Fuzzy Set Operations" section). Let R and W be two fuzzy relations defined in $\text{X} \times \text{Y}$:

- Union: $(R \cup W)(x, y) = R(x, y)\ s\ W(x, y)$, using a s-norm s.
- Intersection: $(R \cup W)(x, y) = R(x, y)\ t\ W(x, y)$, using a t-norm t.
- Complement: $(\neg R)(x, y) = 1 - R(x, y)$.
- Inclusion: $R \subseteq W \Leftrightarrow R(x, y) \leq W(x, y)$.
- Equality: $R = W \Leftrightarrow R(x, y) = W(x, y)$.

Fuzzy relations can be composed with the addition of different operators to sets. If G and *W* are fuzzy relations defined in $X \times Z$ and $Z \times Y$, respectively, then we can define the fuzzy relation *R* defined in $X \times Y$ as follows:

- **Sup-t Composition**:

$$R(x, y) = \underset{z \in Z}{\text{Sup}}[G(x, z) \text{ t } W(z, y)] \tag{1.52}$$

- **Inf-s Composition**:

$$R(x, y) = \underset{z \in Z}{\text{Inf}}[G(x, z) \text{ s } W(z, y)] \tag{1.53}$$

Fuzzy Numbers

The concept of fuzzy numbers was first introduced in Zadeh (1975a, 1975b, 1975c) with the purpose of analyzing and manipulating approximate numeric values, for example "near 0," almost 5," and so forth. The concept has since been refined (Dubois & Prade, 1980, 1985b), and several definitions exist.

Definition 1.19: *Let A be a fuzzy set in* X *and* $\mu_A(x)$ *be its membership function with* $x \in$ X. *A is a* ***fuzzy number*** *if its membership function satisfies that*:

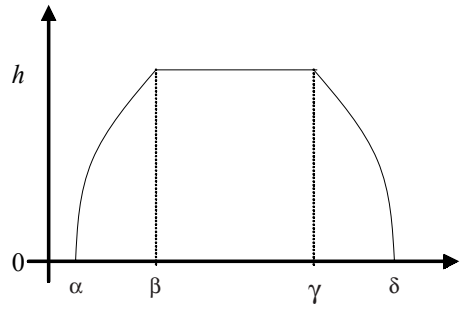
1. $\forall x, y \in X, \forall \mu_A(t) \geq \min(\mu_A(x), \mu_A(y))$, *i.e.* $\mu_A(x)$ *is convex.*
2. $\mu_A(x)$ *is upper semicontinuity.*
3. *Support of A is bounded.*

*

These requirements can be relaxed. Some authors include the necessity for the fuzzy set being normalized in the definition.

The general form of the membership function of a fuzzy number *A* can be seen in Figure 1.23, which can be defined as:

Figure 1.23. General fuzzy number



$$\mu_A(x) = \begin{cases} r_A(x) & \text{if } x \in [\alpha, \beta) \\ h & \text{if } x \in [\beta, \gamma] \\ s_A(x) & \text{if } x \in (\gamma, \delta] \\ 0 & \text{otherwise} \end{cases} \tag{1.54}$$

where r_A, s_A: X $\rightarrow$ [0,1], r_A is not decreasing, s_A is not increasing, and

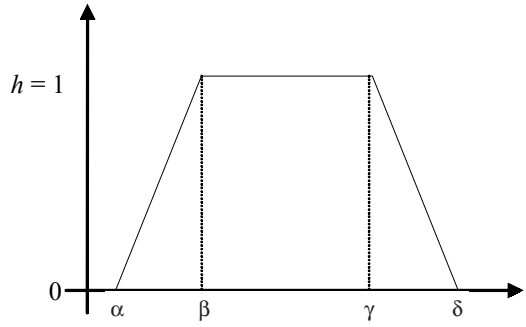
$$r_A(\beta) = h = s_A(\gamma) \tag{1.55}$$

with $h \in (0, 1]$ and $\alpha, \beta, \gamma, \delta \in$ X. The number h is called the height of the fuzzy number, the interval $[\beta, \gamma]$ is the kernel or modal interval, and the numbers $\beta - \alpha$ and $\delta - \gamma$ are the left and right spaces, respectively.

Throughout this study we will often use a particular case of fuzzy numbers that is obtained when we consider the functions r_A and s_A as linear functions. In this case the membership function adopts the following form:

$$\mu_A(x) = \begin{cases} h + \dfrac{(x-\beta)h}{\beta - \alpha} & \text{if } x \in [\alpha, \beta) \\ h & \text{if } x \in [\beta, \gamma] \\ h - \dfrac{(x-\gamma)h}{\delta - \gamma} & \text{if } x \in (\gamma, \delta] \\ 0 & \text{otherwise} \end{cases} \tag{1.56}$$

Figure 1.24. Normalized trapezoidal fuzzy number



We will call this type of fuzzy number *triangular* or *trapezoidal,* and it takes the form shown in Figure 1.24. We will usually work with normalized fuzzy numbers due to which $h = 1$, and in this case we will be able to characterize normalized trapezoidal fuzzy number A, using the four really necessary numbers: $A \equiv (\alpha, \beta, \gamma, \delta)$.

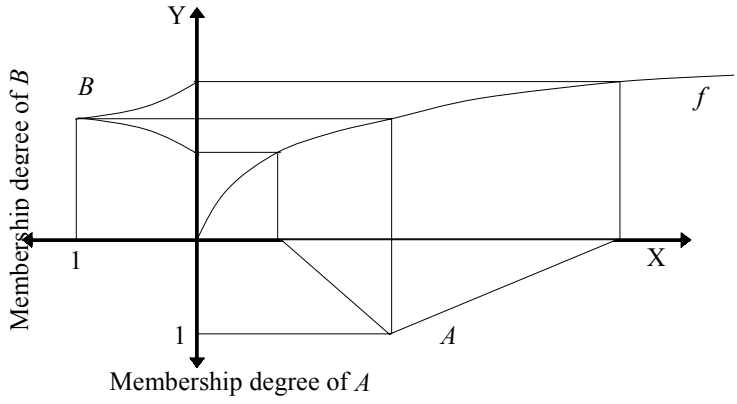
The Extension Principle

One of the most important notions in the fuzzy sets theory is the extension principle, proposed in Zadeh (1975a, 1975b, 1975c). It provides a general method that allows nonfuzzy mathematical concepts to be extended to the treatment of fuzzy quantities. It is used to transform fuzzy quantities, which have the same or different universes, according to a transformation function between those universes.

Let A be a fuzzy set, defined in universe of discourse X, and f a nonfuzzy transformation function between universes X and Y, so that f: X $\rightarrow$ Y. The purpose is to extend f so that it can also operate on the fuzzy sets in X. The result must be fuzzy set B in Y: $B = f(A)$. In Figure 1.25 this transformation is represented. It is achieved with the use of the Sup-Min composition, which will now be described in a general way in the case of the Cartesian product in a universe of discourses.

Definition 1.20: *Let* X *be a cartesian product of n universes such as* $X = X_1 \times X_2 \times ... \times X_n$, *and* $A_1, A_2, ..., A_n$ *are n fuzzy sets in those n universes, respectively. Moreover, we have a function f from* X *to the universe* X', *so a fuzzy set B from* X' *is defined by the* ***extension principle*** *as*:

Figure 1.25. Graphic representation of the extension principle, where f carries out its transformation from X to Y



$$B = f(A_1, A_2, \ldots, A_n) \tag{1.57}$$

defined as

$$\mu_B(y) = \sup_{x \in X, y = f(x)} \min(\mu_{A_1}(x_1)), \ldots (\mu_{A_n}(x_n)) \tag{1.58}$$

*

Example 1.5: Let both *X* and *Y* be the universe of natural numbers.

- **Sum 4** function: $y = f(x) = x + 4$;
  - $A = 0.1/2 + 0.4/3 + 1/4 + 0.6/5$;
  - $B = f(A) = 0.1/6 + 0.4/7 + 1/8 + 0.6/9$.
- **Sum:** $y = f(x_1, x_2) = x_1 + x_2$:
  - $A_1 = 0.1/2 + 0.4/3 + 1/4 + 0.6/5$;
  - $A_2 = 0.4/5 + 1/6$;
  - $B = f(A_1, A_2) = 0.1/7 + 0.4/8 + 0.4/9 + 1/10 + 0.6/11$.

*

We can conclude that the extension principle allows us to extend any function (for example, arithmetic) to the field of fuzzy sets, making possible what we will see in the next section as fuzzy arithmetic.

Fuzzy Arithmetic

Thanks to the Extension Principle (Definition 1.20), it is possible to extend the classic arithmetical operations to the treatment of fuzzy numbers (see Example 1.5). In this way the four main operations are:

1. **Extended Sum**: Given two fuzzy quantities A_1 and A_2 in X, the membership function of the sum $A_1 + A_2$ is found using the expression

$$\mu_{A_1+A_2}(y) = \sup\{\min(\mu_{A_1}(y-x), \mu_{A_2}(x)) / x \in X\} \quad (1.59)$$

   In this way the sum is expressed in terms of the supreme operation. The extended sum is a commutative and associative operation, and the concept of the symmetrical number does not exist.

2. **Extended Difference**: Given two fuzzy quantities A_1 and A_2, in X, the membership function of the difference $A_1 - A_2$ is found using the expression

$$\mu_{A_1+A_2}(y) = \sup\{\min(\mu_{A_1}(y+x), \mu_{A_2}(x)) / x \in X\} \quad (1.60)$$

3. **Extended Product**: The product of two fuzzy quantities $A_1 * A_2$ is obtained as follows:

$$\mu_{A_1*A_2}(z) = \begin{cases} \sup\{\min(\mu_{A_1}(z/y), \mu_{A_2}(y) / y \in X - \{0\}\} & \text{if } z \neq 0 \\ \max(\mu_{A_1}(0), \mu_{A_2}(0)) & \text{if } z = 0 \end{cases} \quad (1.61)$$

4. **Extended Division**: The division of two fuzzy quantities A_1, A_2 is defined as follows:

$$\mu_{A_1 \div A_2}(z) = \sup\{\min(\mu_{A_1}(y \cdot z), \mu_{A_2}(y)) / y \in X\} \quad (1.62)$$

From these definitions we can easily conclude that A_1 and A_2 have a discrete universe (with finite terms), and they have *n* and *m* terms, respectively; the number of terms of $A_1 + A_2$ and of $A_1 - A_2$ is $(n - 1) + (m - 1) + 1$, that is, $n + m - 1$.

Based on a particular expression from the uncertainty principle, adapted to the use of a-cuts and in a type of numbers similar to the previously described, called LR fuzzy numbers, in Dubois and Prade (1980), rapid calculus formulae for the previous arithmetical operations are described.

It is important to point out that if we have two fuzzy numbers, then the sum or remainder of both fuzzy numbers will be fuzzier (it will have greater cardinality) than the fuzziest of the two (that which has greatest cardinality). This is logical, because if we add two approximate values, then the result can be as varied as the initial values are. The same thing happens with division and multiplication, but on a larger scale.

Possibility Theory

The Possibility Theory is based on the idea of linguistic variables and how they are related to fuzzy sets (Zadeh, 1978; Dubois & Prade, 1988). In this way, we can evaluate the possibility of a determinate variable X being (or belonging to) a determinate set *A*, like the membership degree of the X elements in *A*.

Definition 1.21: *Let there be a fuzzy set A defined in* X *with membership function* $\mu_A(x)$ *and a variable x in* X *(the value of which we do not know). So, the proposition "x is A" defines a* ***Possibility Distribution*** *in such a way that it is said that the possibility that* $x = u$ *is* $\mu_A(u)$, $\forall\ u \in X$.

*

The concepts of fuzzy sets and membership functions are now interpreted as linguistic labels and possibility distributions. Instead of membership degrees, we have possibility degrees, but all the tools and properties defined for fuzzy sets are also applicable to possibility distributions.

Fuzzy Quantifiers

Fuzzy or linguistic quantifiers (Yager, 1983; Zadeh, 1983; Liu and Kerre, 1998a, 1998b; Galindo, 1999; Galindo, Medina, Cubero, & García, 2001) allow us to express fuzzy quantities or proportions in order to provide an approximate idea of either the number of a subset's elements fulfilling a certain condition or the proportion of this number in relation to the total number of possible elements.

Fuzzy quantifiers can be *absolute* or *relative*:

- **Absolute quantifiers** express quantities over the total number of elements of a particular set, stating whether this number is, for example, "much more than 10," "close to 100," "a great number of," and so forth. Generalizing this concept, we can consider fuzzy numbers as absolute fuzzy quantifiers in order to use expressions such as "approximately between 5 and 10," "approximately -8," and so forth. Note that the expressed value may be positive or negative. In this case, we can see that the truth of the quantifier depends on a single quantity. For this reason, the definition of absolute fuzzy quantifiers is, as we shall see, very similar to that of fuzzy numbers.
- **Relative quantifiers** express measurements over the total number of elements, which fulfill a certain condition depending on the total number of possible elements (the proportion of elements). Consequently, the truth of the quantifier depends on two quantities. This type of quantifier is used in expressions such as "the majority" or "most," "the minority," "little of," "about half of," and so forth. In this case, in order to evaluate the truth of the quantifier, we need to find the total number of elements fulfilling the condition and to consider this value with respect to the total number of elements that could fulfill it (including those that do fulfill it and those that do not).

Some quantifiers, such as "many" and "few," can be used in either sense, depending on the context (Liu & Kerre, 1998a).

In Zadeh (1983) absolute fuzzy quantifiers are defined as fuzzy sets in the positive real numbers and relative quantifiers as fuzzy sets in the interval [0,1]. We have extended the definition of absolute fuzzy quantifiers to all real numbers.

Definition 1.22: *A **fuzzy quantifier** named* Q *is represented as a function Q whose domain depends on whether it is absolute or relative*:

$$Q_{abs}: \Re \rightarrow [0, 1]$$
$$Q_{rel}: [0, 1] \rightarrow [0, 1] \quad (1.63)$$

where the domain of Q_{rel} *is* [0,1] *because the division* $a / b \in [0, 1]$*, where a is the number of elements fulfilling a certain condition and b is the total number of existing elements.*

*

In order to know the fulfillment degree of the quantifier over the elements that fulfill a certain condition, we apply the function Q of the quantifier to the value of quantification ϕ (phi):

$$\Phi = \begin{cases} a & \text{if } Q \text{ is absolute} \\ a / b & \text{if } Q \text{ is relative} \end{cases} \quad (1.64)$$

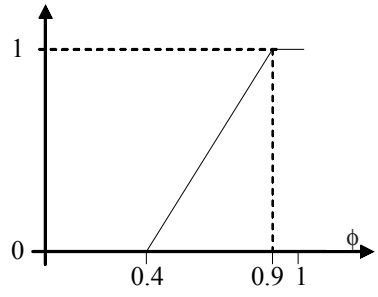
Thus, the fulfillment degree is $Q(\phi)$. If the function of the quantifier (absolute or relative) $Q(\phi)$ has the value 1, this quantifier is completely satisfied. The value 0, on the other hand, indicates that the quantifier is not fulfilled at all. Any intermediate value indicates an intermediate fulfillment degree for the quantifier.

Example 1.6: "Approximately 8" is an absolute fuzzy quantifier, defined as a triangular and symmetrical function (Figure 1.2b) with $m = 8$ and $margin = 2$, for example. "Almost all" is a relative fuzzy quantifier, defined as shown in Figure 1.26.

*

Two classic quantifiers are very important: The universal quantifier (for all, $\forall$) and the existential quantifier (exist, $\exists$). The first is relative, and the second is absolute. They are discretely defined as

Figure 1.26. Relative fuzzy quantifier "almost all": $Q(\phi) = 0 \Leftrightarrow \phi \leq 0.4$, $Q(\phi) = 1 \Leftrightarrow \phi \geq 0.9$ and $Q(\phi) = 2(\phi - 0.4) \Leftrightarrow \phi \in (0.4, 0.9)$



$$Q_{\forall}(x) = \begin{cases} 1 & \text{if } x = 1 \\ 0 & \text{otherwise} \end{cases} \tag{1.65}$$

$$Q_{\exists}(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{otherwise} \end{cases} \tag{1.66}$$

The existential quantifier may be also defined with a nondiscrete trapezoidal form: $[0, 1, \infty, \infty]$.

Some quantifiers (absolute or relative) may have arguments, and in these cases, the function is defined by using the arguments. Most of quantifier with arguments are absolute, because relative are rare.

Example 1.7: Some fuzzy quantifiers with one and two arguments:

- Absolute quantifiers:
  - "Much greater than x": Represented with the function in Figure 1.27a.
  - "About half of x": Represented with the function in Figure 1.27b.
  - "Approximately between x and y" ($x < y$): Represented with a trapezoidal function (Figure 1.24) with $[\alpha, \beta, \gamma, \delta] = [x - 5, x, y, y + 5]$. Another form is $[0.75x, x, y, 1.25y]$.

- "Approximately between half of x and half of y" ($x < y$): Represented with a trapezoidal function (Figure 1.24) with $[\alpha, \beta, \gamma, \delta] = [0.25x, 0.5x, 0.5y, 0.75y]$.

- Relative quantifiers:
  - "Approximately an x-th part" ($x \in [0, 1]$): $[\alpha, \beta, \gamma, \delta] = [x - 0.2, x, x, x + 0.2]$.
  - "Less than an x-th part" ($x \in [0,1]$): $[\alpha, \beta, \gamma, \delta] = [0, 0, x, 1.25x]$.
  - "Approximately between an x-th and a y-th part" ($x < y$ and $x, y \in [0, 1]$). This is a rare relative quantifier with arguments: $[\alpha, \beta, \gamma, \delta] = [0.75x, x, y, 1.25y]$ or $[x - 0.1, x, y, y + 0.1]$. For example, "approximately between a quarter and the half" is represented with $x = 0.25$ and $y = 0.5$.
  - "Approximately between half of an x-th and half of a y-th part" ($x < y$ and $x, y \in [0, 1]$). $[\alpha, \beta, \gamma, \delta] = [0.4x, 0.5x, 0.5y, 0.6y]$.

Observe that in relative quantifiers it does not matter if d > 1, because the important part is the quantifier definition in [0, 1].

*

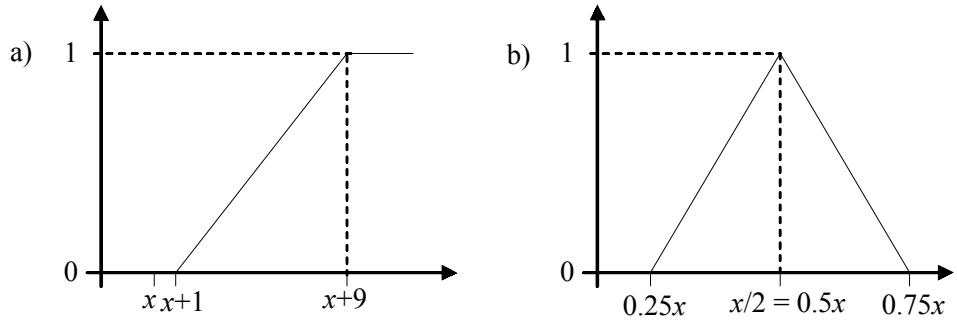
A general classification of trapezoidal quantifiers $[\alpha, \beta, \gamma, \delta]$, attending to its arguments and the building type, is the following one:

1. Without arguments: See Equation 1.63.

2. With one argument x:
   - Type Product: $[\alpha * x, \beta * x, \gamma * x, \delta * x]$.
   - Type Sum: $[\alpha + x, \beta + x, \gamma + x, \delta + x]$.

3. With two arguments x and y:
   - Type Product: $[\alpha * x, \beta * x, \gamma * y, \delta * y]$.
   - Type Sum: $[\alpha + x, \beta + x, \gamma + y, \delta + y]$.

Figure 1.27. Absolute fuzzy quantifiers with one argument (type sum and product): a) "Much greater than x*":* [1 + x, 9 + x, ∞, ∞]*, and b) "About half of* x*":* [0.25x, 0.5x, 0.5x, 0.75x]



In relative quantifiers it is not necessary to warranty that all values are in [0, 1], but if the condition is important, we can use the function min. For example, a type product relative quantifier with one argument is built with [min{1, $\alpha * x$}, min{1, $\beta * x$}, min{1, $\gamma * x$}, min{1, $\delta * x$}].

A survey of methods for evaluating quantified sentences and some new methods is shown in Delgado, Sánchez, and Vila (1999, 2000) and Sánchez (1999).

Endnote

[1] L.A. Zadeh has been named "doctor honoris causa" in universities worldwide, one of them being the University of Granada in 1996, in recognition of his important contribution in this scientific field.

Chapter II

Fuzzy Database Approaches

Both the problem of representation and the treatment of imprecise information have been widely discussed. Many references can be found in the corresponding bibliography. Nevertheless, all the known models aimed at solving this problem have their own advantages, disadvantages, and constraints.

The term *imprecision* encompasses various meanings, which might be interesting to highlight. It alludes to the facts that the information available can be incomplete, 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). Sometimes these meanings are not disjunctive and can be combined in certain types of information.

This chapter deals with the main published models aimed at solving the problem of representation and treatment of imprecise information in relational databases. This problem is not trivial, because it requires relations structure modification, and thus, the operations on these relations also need to be modified. To allow the storage of imprecise information and the making of an inaccurate query of such information, a wide variety of case studies that do not occur in the classic model, without imprecision, is required.

The models exposed include various approaches that do not utilize the fuzzy logic, such as the "classical" Codd approach (1979, 1986, 1987, 1990), the Prade-Testemale model (1984, 1987a, 1987b; Prade, 1984), the Umano-Fukami model (Umano, 1982, 1983; Umano & Fukami, 1994), the Buckles-Petry model (1982a, 1982b, 1984), the Zemankova-Kandel model (Zemankova-Leech & Kandel, 1984, 1985), and the GEFRED model of Medina-Pons-Vila (1994; Medina, 1994; Galindo, Medina, Cubero, & Pons, 1999, 2001b).

Other models deal with database uncertainty as well but have not been widely accepted, such as the ones based on *rough* sets, which were introduced by Pawlak (1982, 1991).

The GEFRED model, which the development of this work is based on, will be the main focus of our discussion. 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 relational databases. One of the major advantages of this model is that it consists of a general abstraction that allows for the use of various approaches, regardless of how different they might look.

A more detailed discussion of each one of these models can be seen in the corresponding references, some of them in Medina (1994) and Petry (1996). In Medina, a comparative study of GEFRED and other models can also be examined.

Imprecision Without Fuzzy Logic

In this section, we will summarize some ideas allowing for imprecise information treatment *without* utilizing either the fuzzy set theory or the possibility distributions. In the bibliography, these models are dealt with globally in the section on imprecision in conventional databases, although some of the ideas discussed here have not been implemented in any of the models.

The Codd Approach

The first attempt to represent imprecise information on databases was the introduction of NULL values by Codd in 1979, which was further expanded

Table 2.1. Truth tables for the tri-valued logic: True, false, and maybe

NOT	
T	F
m	m
F	T

AND	T	m	F
T	T	m	F
m	m	m	F
F	F	F	F

OR	T	m	F
T	T	T	T
m	T	m	m
F	T	m	F

Table 2.2. Truth tables for the tetra-valued logic

NOT	
T	F
A	A
I	I
F	T

AND	T	A	I	F
T	T	A	I	F
A	A	A	I	F
I	I	I	I	F
F	F	F	F	F

OR	T	A	I	F
T	T	T	T	T
A	T	A	A	A
I	T	A	I	F
F	T	A	F	F

(1986, 1987, 1990). This model did not use the fuzzy set theory. A NULL value in an attribute indicates that such a value was any value included in the domain of such an attribute.

Any comparison with a NULL value originates an outcome that is neither true (T) nor false (F) called *maybe* (m) (or unknown, in the SQL of Oracle). The truth tables of the classical comparators NOT, AND, and OR can be seen in Table 2.1.

Later on, another nuance was added, differentiating the NULL value in two marks, the *A-mark* representing an absent or unknown value, although it was applicable, and the *I-mark* representing the absence of the value because it is not applicable (undefined). An I-mark may be situated, for instance, in the car plate attribute of someone who does not have a car. This is a tetra-valued logic, where the A value, having a similar meaning to that of the m in the tri-valued logic, is generated by comparing any value containing an A-mark, and a new I value is added as a result of the comparison of any value containing an I-mark. The tetra-valued logic is shown in Table 2.2.

Default Values

In 1982, Date presented an approach for the treatment of null values, included in his 1986 book. He starts from the notion that the null values treatment problem is not properly defined; therefore, this feature should not be included

in the Relational Model. Based on this premise, Date presents an alternative grounded on the "default values" concept.

In this model, a value of such a domain is labeled "default value" in the declaration of each domain. The same thing is done in SQL with the `DEFAULT` clause. Thus, when a new tuple becomes part of the relations, the user has to provide a value for each attribute that doesn't have a default value, and the system will assign the default value to those attributes that have no value.

Interval Values

Grant (1980) expands the relational model in order to allow that a possible value range/interval be stored in one attribute, in addition to a precise value as well as the NULL value in case no information is available. The problem about repeated tuples is solved by allowing the tuples repetition, because even though they may seem identical, they could be different.

The relational operators are redefined in two versions, true and maybe. For example, the operator < is defined as

$$[a, b] <_T [n, m] \quad \text{if } b < n \qquad (2.1)$$
$$[a, b] <_M [n, m] \quad \text{if } a < m$$

For queries in this model, Lipski's proposal (1979) can be used (Prade, 1984), in which each tuple is placed in one of these three categories: surely belongs to the result (surely-set), likely to belong to the result (possibly-set), and surely does not belong to the result (eliminated-set).

Statistical and Probabilistic Databases

The main proposal related to this issue was published by Wong in 1982, in which a lot of cases of statistical inference uncertainty were discussed. This formula assumes that incomplete information can be statistically compared. This method treats queries as statistical experiments in which information is incomplete and focuses on calculating the response to the query as a set of those tuples that minimize both types of statistical errors. In Shoshani and Wong (1985), research on statistical databases and their usefulness are summarized.

Barbara, Garcia-Molina, and Porter (1992) published the best developed probabilistic databases, in which probabilities are associated with attribute values. In this model, each probabilistic attribute is dealt with as a discrete probability distribution. In one tuple, probabilities must be standardized: The sum total of the probabilities of all probable values must be equal to 1. Nevertheless, determining the probabilities for all the possible domain values can be difficult. So they developed the notion of *missing probabilities* in order to introduce the rest of the probabilities in a more general value, including all the domain values without specifying how the probability is distributed in those values. That is, each value of the domain of which we know its probability is stored in the database, and for all the values of the domain, the remaining value of such a probability is stored (1 minus the sum total of the known probabilities).

When recuperating (queries), a mechanism could be established in order to recuperate only the values having a probability greater than certain threshold. Also, in relational operations (like the Natural Join), probabilities may need to be calculated. To calculate these probabilities, two methods are utilized. In the first one the user introduces probabilities based on user-defined criteria or user-performed calculations. In the second one, the system calculates probabilities based on a set of examples, including survey-collected data or direct analysis of a population sample.

In the Cavallo and Pittarelli (1987) model, the probability is associated to each tuple, signaling the probability that such a tuple may belong to the relation. On the other hand, Fuhr's work (1990) focuses more on how to specify imprecise queries.

Basic Model of Fuzzy Databases

The basic model of fuzzy relational databases is considered the simplest one, and it consists of adding a *grade,* normally in the [0, 1] interval, to each instance (or tuple). This makes keeping database data homogeneity possible. Nevertheless, the semantic assigned to this grade will determine its usefulness, and this meaning will be utilized in the query processes.

This grade may have the meaning of *membership degree* of each tuple to the relation (Giardina, 1979; Mouaddib, 1994). But it may mean something different, such as the *dependence strength level* between two attributes, thus representing the relation between them (Baldwin, 1983), the *fulfillment*

degree of a condition, or the *importance degree* (Bosc, Dubois, Pivert, & Prade, 1997) of each tuple in the relation, among others.

The main problem with these fuzzy models is that they do not allow for the representation of imprecise information about a certain attribute of a specific entity (such as the "tall" or "short" values for a height attribute). Besides, the fuzzy character is assigned globally to each instance (tuple), making it impossible to determine the specific fuzzy contribution from each constituting attribute.

Similarity Relations: The Buckles-Petry Model

The Buckles-Petry Model is the first model that utilizes similarity relations (Zadeh, 1971) in the relational model. It was proposed by Buckles and Petry (1982a, 1982b, 1984). In this model, a *fuzzy relation* is defined as a subset of the following Cartesian product: $P(D_1) \times ... \times P(D_m)$, where $P(D_i)$ represents the parts set of a D_i domain, including all the subsets that could be considered within the D_i domain (having any number of elements). The data types permitted by this model are the following:

- Finite set of scalars (labels)
- Finite set of numbers
- Fuzzy number set

The meaning of these sets is disjunctive, that is, the real value is one belonging to the set.

The equivalence types on a domain are constructed from a *similarity relation*, in which the user provides values taken by such a relation. Typically, these similarity values are standardized in the [0,1] interval, where 0 corresponds to "totally different" and 1 to "totally similar."

A *similarity threshold* can be established, with a value between 0 and 1, in order to get the values whose similarity is greater than the threshold or to consider those values undistinguishable.

Possibilistic Models

Under this denomination, models using the possibility theory (see "Possibility Theory" section in Chapter I) to represent imprecision are included.

The most important models in this group are:

1. Prade-Testemale Model
2. Umano-Fukami Model
3. Zemankova-Kaendel Model

Prade-Testemale Model

Prade and Testemale published an FRDB model that allows the integration of what they call *incomplete* or *uncertain* data in the possibility theory sphere (Prade & Testemale, 1984, 1987a, 1987b; Prade, 1984).

An attribute A, having a D domain, is considered. All the available knowledge about the value taken by A for an x object can be represented by a *possibility distribution* $\pi_{A(x)}$ about $D \cup \{e\}$, where e is a special element denoting the case in which A is not applied to x. In other words, $\pi_{A(x)}$ is an application that goes from $D \cup \{e\}$ to the [0, 1] interval. From this formulation, all value types adopted by this model can be represented.

In every possibilistic model, one must take into account that for a value $d \in D$, if $\pi_{A(x)}(d) = 1$, then the d value is totally possible for A(x), and the d value is not necessarily true for A(x), unless this is the only possible value, that is, $\pi_{A(x)}(d') = 0$, for every $d' \neq d$. Both the information and representation of this model are shown in Table 2.3.

The way that two possibility distributions can be compared was discussed in the "Comparison Operations on Fuzzy Sets" section in Chapter I. In general, the most commonly used measurements are possibility and necessity (see the subsection "Possibility and Necessity Measures" in Chapter I). For a more accurate definition of these fuzzy database comparators, refer to Chapter VII.

Table 2.3. Representation of information in two possibilistic models

Information	Prade-Testemale Model	Umano-Fukami Model
The precise data is known and this is *crisp*: c	$\pi_{A(x)}(e) = 0$ $\pi_{A(x)}(c) = 1$ $\pi_{A(x)}(d) = 0, \quad \forall d \in D, d \neq c$	$\pi_{A(x)}(d) = \{1 / c\}$
Unknown but applicable	$\pi_{A(x)}(e) = 0$ $\pi_{A(x)}(d) = 1, \quad \forall d \in D$	Unknown (Equation 2.2)
Not applicable or nonsense	$\pi_{A(x)}(e) = 1$ $\pi_{A(x)}(d) = 0, \quad \forall d \in D$	Undefined (Equation 2.3)
Total ignorance	$\pi_{A(x)}(d) = 1, \quad \forall d \in D \cup \{e\}$	Null (Equation 2.4)
Range [m, n]	$\pi_{A(x)}(e) = 0$ $\pi_{A(x)}(d) = 1$ if $d \in [m, n] \subseteq D$ $\pi_{A(x)}(d) = 0$ in other case	$\pi_{A(x)}(d) = 1$ if $d \in [m, n] \subseteq D$ $\pi_{A(x)}(d) = 0$ in other case
The information available is a possibility distribution μ_a	$\pi_{A(x)}(e) = 0$ $\pi_{A(x)}(d) = \mu_a(d) \quad \forall d \in D$	$\pi_{A(x)}(d) = \mu_a(d) \quad \forall d \in D$
The possibility that it may not be applicable is λ and, in case it is applicable the data is μ_a	$\pi_{A(x)}(e) = \lambda$ $\pi_{A(x)}(d) = \mu_a(d) \quad \forall d \in D$	Without representation

Umano-Fukami Model

This proposal also utilizes the possibility distributions in order to model information knowledge. In this model, the nonapplicable information may be modeled by a possibility distribution about a given domain, where each domain value has a possibility equal to 0. That is, if D is the discourse universe of A(x), and $\pi_{A(x)}(d)$ represents the possibility that A(x) takes the value $d \in U$, then for the unknown and applicable values, the following representation is used:

$$\text{Unknown} = \pi_{A(x)}(d) = 1 \quad \forall d \in D \tag{2.2}$$

The *nonapplicable* values have a special case of possibility distribution named *undefined,* which is represented as:

$$\text{Undefined} = p_{A(x)}(d) = 0 \quad \forall d \in D \tag{2.3}$$

To represent a situation in which a *lack* of information is *applicable* or *nonapplicable,* a special value called *Null* is used as follows:

$$\text{Null} = \{1 / \text{Unknown}, 1 / \text{Undefined}\} \quad (2.4)$$

For the remaining cases of imprecise information, a similar model to the preceding one is adopted.

Besides, every instance of a relation in this model has a possibility distribution associated with it in the [0, 1] interval, thus indicating the membership degree of that particular instance to such a relation. In other words, a fuzzy relation R, with m attributes, is defined as the following membership function:

$$\mu_R: P(U_1) \times P(U_2) \times ... \times P(U_m) \longrightarrow P([0, 1]) \quad (2.5)$$

where the × symbol denotes the Cartesian product, $P(U_j)$ with $j = 1, 2, ..., m$ is the collection of all the possibility distributions in the discourse universe U_j of the j-th R attribute.

The function m_R associates a P([0, 1]) value to every instance of the relation R, which corresponds to all the possibility distributions in the [0, 1] interval; this shall be considered as an R membership degree of such an instance.

Finally, in the query process, expressed either in fuzzy or precise terms, the model solves the query problem by dividing the set of instances involved in the relation into three subsets, where the first subset contains the instances completely satisfying the query, the second subset groups those instances that might satisfy the query, and the third subset consists of those instances that do not satisfy the query. The information as well as the representation of this model are shown in Table 2.3.

Zemankova-Kandel Model

This Zemankova-Kandel Model was published in 1984 and 1985. It consists of three parts:

- A value database, in which data are organized in a similar way as in the possibilistic models
- An explanatory database, in which both the fuzzy subsets and fuzzy relation used are stored
- A set of translating rules for the handling of adjectives and modifiers

The query is posed in a similar way as to that in the Prade-Testemale Model, except the possibility measure used to find the compatibility of the fuzzy subset F of the condition, with an attribute A value for each tuple in the relation, is given by the following equation:

$$P_A(F) = \sup_{u \in D} \{\mu_F(u) \cdot \pi(u)\} \quad (2.6)$$

The certainty measure is given by this equation:

$$C_A(F) = \max_{u \in D} \{0, \inf \{\mu_F(u) \cdot \pi_A(u)\}\} \quad (2.7)$$

Certainty is used instead of the necessity in the Prade-Testemale Model. However, the interpretation of the certainty degree is unclear, and no relationship exists between possibility and certainty as it does between possibility and necessity: $N(X) = 1 - P(\neg X)$ (see Equations 1.46 and 1.47 in Chapter I).

The result of a query is presented as fuzzy relations containing two fields, in which both possibility and certainty values for each instance are included for a given query. Minimum thresholds can be established for these relations.

Upon imposing conditions when selecting, the starting point is a defined similarity relations on $D \times D$, from which any other comparative relation is built. Nevertheless, this has some important limitations, which make it an incomplete model.

The GEFRED Model by Medina-Pons-Vila

The GEFRED Model dates back to 1994, and it experienced subsequent expansions (Medina, Pons, & Villa, 1994; Medina, 1994; Galindo, Medina, & Aranda, 1999; Galindo, Medina, Cubero, & García, 2001). This model is an eclectic synthesis of some of the previously discussed models. Being a possibilistic model, it particularly refers to generalized fuzzy domains, thus admitting the possibility distribution in the domains, but it also includes the case where the underlying domain is not numeric but scalars of any type. It includes unknown, undefined, and null values as well, having the same sense as that in Umano and Fukami.

Table 2.4. Data types in the GEFRED Model

1.	A single scalar (e.g., Size = Big, represented by the possibility distribution 1/Big).
2.	A single number (e.g., Age = 28, represented by the possibility distribution 1/28).
3.	A set of mutually exclusive possible scalars (e.g., Behavior = {Bad, Good}, represented by {1/Bad, 1/Good}).
4.	A set of mutually exclusive possible numbers (e.g., Age = {20, 21}, represented by {1/20, 1/21}).
5.	A possibility distribution in a scalar domain (e.g., Behavior = {0.6/Bad, 1.0/Regular}).
6.	A possibility distribution in a numeric domain (e.g., Age = {0.4/23, 1.0/24, 0.8/25}, fuzzy numbers or linguistic labels).
7.	A real number belonging to [0,1], referring to the degree of matching (e.g., Quality = 0.9).
8.	An Unknown value with possibility distribution: Unknown = {1/d : d ∈ D}.
9.	An Undefined value with possibility distribution: Undefined = {0/d : d ∈ D}.
10.	A NULL value given by: NULL = {1/Unknown,1/Undefined}.

The GEFRED model is based on the generalized fuzzy domain (D) and generalized fuzzy relation (R), which include classic domains and classic relations, respectively.

Definition 2.1: If U is the discourse domain or universe, P(U) is the set of all possibility distributions defined for U, including those that define the Unknown and Undefined types (types 8 and 9 in Table 2.4), and NULL is another type defined in Table 2.4 (type 10). Therefore, we define the generalized fuzzy domain as $D \subseteq P(U) \cup NULL$. The unknown, undefined, and NULL types are defined according to Umano and Fukami (1994; see the "Umano-Fukami Model" section, earlier in this chapter). All data types that can be represented are shown in Table 2.4.

*

Definition 2.2: The generalized fuzzy relations of the GEFRED Model are relations whose attributes have a generalized fuzzy domain. The following points are also true:

- Each attribute A_j may be associated to a "compatibility attribute" C_j, where we can store a compatibility degree. The compatibility degree for an attribute value is obtained by manipulation processes (such as queries)

performed on that relation and indicates the degree to which that value has satisfied or met the operation performed on it.

- Generalized fuzzy relations are given by two sets: Head H and Body B. The head includes the name of each one of the *n* attributes, their domains, and their compatibility attributes (which are optional). The body includes the values of the *m* tuples:

$$R=\begin{cases} H = \{(A_1:D_1[,C_1], ..., A_n:D_n[,C_n])\} \\ B = \{(A_1:d_{i1}[,c_{i1}], ..., A_n:d_{in}[,c_{in}])\} \text{ con } i= 1,..., m \end{cases} \quad (2.8)$$

*

The GEFRED Model defines fuzzy comparators, which are general comparators based on any existing classical comparator (>, <, =, etc.), but it does not consolidate the definition of each one. The only requirement established is that the fuzzy comparator should respect the classical comparators' outcomes when comparing possibility distributions expressing *crisp* values (such as 1/x with x belonging to X).

For example, the "approximately equal" comparator, "possibly equal" or "fuzzy equal" (FEQ), may be defined in the following equation, where values $p, p' \in D$, and their associated possibility distributions are π_p and $\pi_{p'}$, respectively. U is the discourse domain underlying the generalized fuzzy domain D (see Definition 2.1):

$$\text{FEQ}(p, p') = \sup_{d \in U} \min(\pi_p(d), \pi_{p'}(d)) \quad (2.9)$$

Note that this definition is the possibility measure given in Equation 1.40 in Chapter I.

On these definitions, GEFRED redefines the relational algebraic operators in the so-called generalized fuzzy relational algebra: union, intersection, difference, Cartesian product, projection, selection, join, and division. These operators are defined by giving the head and body of a generalized fuzzy relation, which is the result of the operation. All these operators are defined in the definition of GEFRED, but the fuzzy division is defined in Galindo, Medina, Cubero, and García (2001). Fuzzy relational calculus is defined in Galindo,

Medina, and Aranda (1999). An extension for fuzzy deductive relational databases was presented in Blanco, Cubero, Pons, and Vila (2000).

Fuzzy Object-Oriented Database Models

Although the object-oriented database model has not been very successful on a practical level, it has become increasingly important on a theoretical and scientific level. With respect to the scope of this model when working with fuzzy information, some proposals have emerged. We briefly discuss some of the latest ones here. An overview of other approaches can be found in De Caluwe (1997).

A Generalized Object-Oriented Database Model

Tré, de Caluwe, and Van der Cruyssen (2000) propose a generalized object-oriented database model, a formal framework for the definition of a fuzzy and/or uncertain object-oriented database model. This model is obtained as a generalization of a crisp object-oriented database model compliant with the ODMG de facto standard and built upon an algebraic type system, and a constraint system. The ODMG proposal still suffers from some shortcomings, such as the absence of formal semantics and its limited ability to deal with constraints, despite the fact that a thorough support of constraints is the most obvious way to guarantee the integrity of a database.

Thus, this framework is based on a type system and a related constraint system, which is meant to guarantee database integrity. The constraint system includes a variety of constraint types. The constraint definition is so formal that is difficult to set and understand. As you will see in Chapter IV, the FuzzyEER Model includes an easy way to set fuzzy constraints.

A Fuzzy Object-Oriented Database Management System

Bordogna, Leporati, Lucarella, and Pasi (1999), as well as Bordogna, Lucarella, and Pasi (2000), present a prototypal implementation of the Fuzzy Object-Oriented Data Model (FOOD), which allows the representation and instantiation of vague attribute values and uncertain and strengthened relations.

The FOOD Model is defined as an extension of a graph-based object model in order to manage both crisp and imperfect information by using fuzzy set theory and possibility theory (see Chapter I). In this model a conceptual scheme is defined as a quintuple {C, T, A, P, N}:

1. C is a finite set of class names (crisp and fuzzy classes). Fuzzy classes collect objects, which can have a partial membership to the class.
2. T is a finite set of type names (crisp and vague types). Vague types elements denote sets of vague and imprecise values.
3. A is a set of attribute names. Attributes are simple when their domain is a type and are complex when their domain is a class. Moreover, both single-valued and multivalued attributes exist.
4. P is the property relation. P relates one class with its attribute names and both with the domain. This domain may be another class or type.
5. H is the inheritance relation. If the inheritance relation is fuzzy, then one label specifies the extent to which the instances of the subclass are also instances of the superclass.

Thus, the imperfection in data representation is modeled in FOOD in order to define the following aspects:

1. Vague attribute values are defined to model situations in which the precise value of a given object attribute is not precisely known. These authors use possibility distributions. Note that the FuzzyEER Model (Chapter IV) allows other interesting fuzzy attribute types.
2. Uncertain property and uncertain link relationships allow us to associate an uncertainty degree, which can be interpreted as a tolerance threshold to violate the constraint imposed on the actual attribute value.
3. Strengthened property and strengthened link relationships allow us to use uncertainty degrees in the relationship between two objects.
4. Fuzzy classes are useful when we want to represent the partial membership of an object to a class.
5. Fuzzy class hierarchies represent the vagueness in a hierarchy defined for classification purposes. This situation occurs when both the superclass and the subclass are fuzzy. In fact, due to this fact, the vague concepts

represented in the superclass can be specialized or generalized by the vague concepts represented in the subclass to a vague extent, which can be specified by a fuzzy quantifier (see the "Fuzzy Quantifiers" section in Chapter I).

The implementation has been conceived as an extension of an object-oriented database management system, the commercial product O_2, which is programmable by means of its own language (O_2C), an object-oriented extension of the C language, with direct access to data types and constructs defined.

This prototype lacks many of the characteristics that are usually found in a full-featured application. However, it is the first step toward achieving a good fuzzy object-oriented database.

Chapter III

State of the Art in Fuzzy Database Modeling

On occasion, the term *imprecision* embraces several meanings that we should differentiate. For example, as you saw in Chapter II, the information you have may be incomplete or *fuzzy* (diffuse or vague), you may not know whether it is certain (uncertainty), perhaps you are totally ignorant of the information (unknown), you may know that the information cannot be applied to a specific entity (undefined), or you may not even know whether the data can be applied to the entity in question (total ignorance or a value of null) (Umano & Fukami, 1994). Each of these terms depends on the context in which it is applied.

The management of uncertainty in database systems is a very important problem (Motro, 1995), as the information is often vague. Motro states that fuzzy information is content-dependent, and he classifies it as follows:

- **Uncertainty**: It is impossible to determine whether the information is true or false. For example, "John may be 38 years old."
- **Imprecision**: The information available is not specific enough. For example, "John may be between 37 and 43 years old," "John is 34 or 43 years old" (disjunction), "John is not 37 years old" (negative), or even a simple unknown.

- **Vagueness**: The model includes elements (predicates or quantifiers) that are inherently vague, for example, "John is in his early years" or "John is at the end of his youth." However, after these concepts have been defined, this case would match the previous one (imprecision).
- **Inconsistency**: It contains two or more pieces of information that cannot be true at the same time. For example, "John is 37 and 43 years old, or he is 35 years old"; this is a special case of disjunction.
- **Ambiguity**: Some elements of the model lack complete semantics (or a complete meaning). For example, "It is unclear whether the salaries are annual or monthly."

Zadeh (1965) introduces the fuzzy logic, as explained in Chapter I, in order to deal with this type of data. Traditional logic, because it is bi-valued, can operate only with concepts such as yes or no, black or white, true or false, or 0 or 1, which allow for a very limited knowledge representation. Although other logics take more truth values, namely multivalued logics (see the "Imprecision Without Fuzzy Logic" section in Chapter II), fuzzy logic is one extension that takes endless truth levels (or degrees), associating the concept of membership degree or truth degree in an interval [0,1] within the fuzzy logic theory.

Fuzzy databases have also been widely studied (see Chapter II), with little attention being paid to the problem of conceptual modeling (Chaudhry, Moyne, & Rundensteiner, 1999). This does not mean that there are no publications, however, but that they are sparse and have no standard. Therefore, there have also been advances in modeling uncertainty in database systems (Buckles & Petry, 1985; Kerre & Chen, 1995; Chen, 1998; Yazici & George, 1999) including object-oriented database models (Van Gyseghem, de Caluwe, & Vandenberghe, 1993; George, Srikanth, Petry, & Buckles, 1996; de Caluwe, 1997; Bordogna, Lucarella, & Pasi, 1999; Yazici and George, 1999).

At the same time, the extension of the ER model for the treatment of fuzzy data (with vagueness) has been studied in various publications (Zvieli & Chen, 1986; Ruspini, 1986; Vandenberghe, 1991; Chaudhry, Moyne, & Rundensteiner, 1994, 1999; Chen & Kerre, 1998; Chen, 1998; Kerre & Chen, 2000; Vert, Morris, Stock, & Jankowski, 2000; Ma, Zhang, Ma, & Chen, 2001), but none of these publications refer to the possibility of expressing constraints by using the tools by fuzzy sets theory. In Kerre and Chen (1995), you can find a summary of some of these models.

On the other hand, the main methodologies of databases design (Batini, Ceri, & Navathe, 1994; Connolly, Begg, & Strachan, 1998; Elmasri & Navathe, 2000; de Miguel, Piattini, & Marcos, 1999; Atzeni, Ceri, Paraboschi, & Torlone, 1999; Gardarin, 1999) have not paid attention to the modeling of data with uncertainty, although the intent of uncertainty modeling of the real world is rarely absent.

Based on these concepts, in this chapter we discuss different approaches, by various authors, related to the uncertainty conceptual modeling problem in database models.

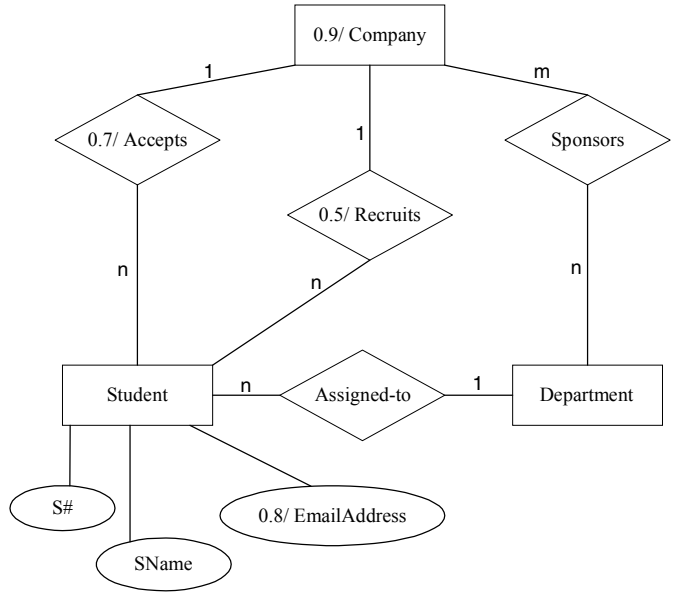
Closing the modeling stage, in Chapter IV we present a Fuzzy Enhanced Entity-Relationship model, also known as FuzzyEER, a tool for fuzzy database modeling with many advantages with respect to the modeling tools presented in this chapter: fuzzy values in the attributes, the degree in each value of an attribute, the degree in a group of values of diverse attributes, as well as fuzzy entities, fuzzy relationships, fuzzy aggregation, fuzzy constraints, and so on. We also include a comparison of FuzzyEER and some other fuzzy models.

The Zvieli and Chen Approach

Zvieli and Chen (1986) offered the first great approach in ER modeling. They allowed fuzzy attributes in entities and relationships and introduced three levels of fuzziness in the ER model:

1. At the first level, entity sets, relationships, and attribute sets may be fuzzy, namely, they have a membership degree to the model. For example, in Figure 3.1, the fuzzy entity "Company" has a 0.9 membership degree, the relationship "Accepts" has a 0.7 membership degree, and the fuzzy attribute "EmailAddress" has a 0.8 membership degree.
2. The second level is related to the fuzzy occurrences of entities and relationships. For example, an entity "Young_Employees" must be fuzzy, because its instances, its employees, belong to the entity with different membership degrees.
3. The third level concerns the fuzzy values of attributes of special entities and relationships. For example, attribute "Quality" of a basketball player may be fuzzy (the possibilities include bad, good, very good, and so on).

Figure 3.1. Example with membership degrees to the model in some sets (entities, relationships, or attributes): The first level of the Zvieli and Chen approach



The first level may be useful, but at the end you must decide whether such an entity, relationship, or attribute will or will not appear in the implementation. The second level is useful, too, but it is important to consider different degree meanings (membership degree, importance degree, fulfillment degree, and so on). A list of authors using different meanings may be found in Galindo, Medina, Cubero, and García (2001). The third level is useful and is similar to writing the data type of some attributes, because fuzzy values belong to fuzzy data types.

Proposal of Yazici and Merdan

Yazici and Merdan (1996) propose an extension of the IFO model, shown in Figure 3.2a, for the processing of imprecise data and special treatment of data where similarity exists in a label. They call this extension ExIFO, and by means of examples they explain the implementation and validation of the representation of a fuzzy conceptual scheme by looking at a representation of uncertain

attributes. In the model, three new constructors are added, and by using these new constructors, it is possible to represent attributes that explicitly have uncertain values.

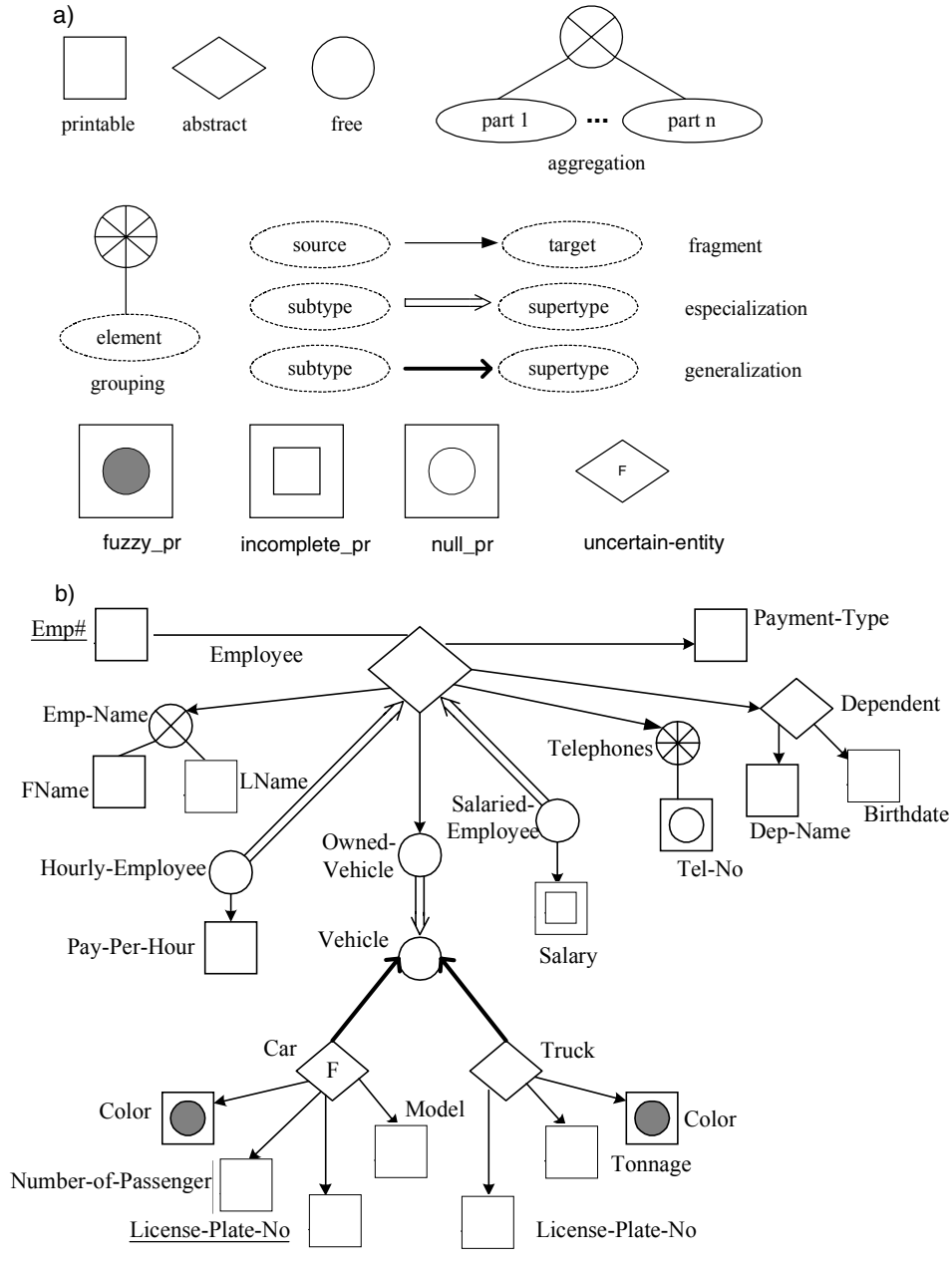
The ExIFO conceptual model (Yazici, Buckles, & Petry, 1999) allows imprecision and uncertainty in database models, based on the IFO conceptual model (Yazici & Merdan, 1996; Yazici & George, 1999). They use fuzzy-valued attributes, incomplete-valued attributes, and null-valued attributes. In the first case, the true data may belong to a specific set or subset of values; for example, the domain of this attribute may be a set of colors {red, orange, yellow, blue} or a subset {orange, yellow} where there is a similarity relation between the colors. In the second case, the true data value is unknown; for example, the domain of this attribute may be a set of years between 1990 and 1992. In the third case, the true data value is available but is not expressed precisely; for example, the domain of this attribute may be whether a certain telephone number exists. Each of these attribute types has a formal definition and a graphical representation. In this study, the authors introduce high-level primitives whose semantics are related to each other with logic operators OR, XOR, or AND to model fuzzy entity type. An example involving an employee-vehicle scheme is used in Figure 3.2b to illustrate the aggregation and composition of fuzzy entity types. The main contribution of this approach is the use of an extended NF^2 relation (Non First Normal Form) to transform a conceptual design into a logical design. Consequently, the strategy is to analyze the attributes that compose the conceptual model in order to establish an NF^2 model.

The study in Yazici and Cinar (2000) is mainly a conceptual modeling approach for the representation of complex-uncertain information (Yazici & Merdan, 1996) by using an object-oriented paradigm and an algorithm for transforming a conceptual schema specification of the ExIFO Model into a logical schema of the Fuzzy Object-Oriented Databases Model (FOOD). ExIFO attempts to preserve the acquired strengths of semantic approaches while integrating concepts from the object paradigm and fuzziness by adding new constructors.

The Chen and Kerre Approach

In Chen and Kerre (1998), Chen (1998), and Kerre and Chen (2000) these authors introduce the fuzzy extension of several major EER concepts (super-

Figure 3.2. Fuzzy ExIFO Model proposed by Yazici and Merdan (1996): a) notation b) example employee-vehicle



class, subclass, generalization, specialization, category, and shared subclass) without including graphical representations. The basic idea is that if E1 is a superclass of E2 and $e \in E2$, then $E1(e) \leq E2(e)$, where E1(e) and E2(e) are the membership functions of e to E1 and E2, respectively. They discuss three kinds of constraints with respect to fuzzy relationships (but they do not study fuzzy constraints): (1) The inheritance constraint means that a subclass instance inherits all relationship instances in which it has participated as a superclass entity, (2) the total participation constraint for entity E is defined when for any instance in E, $\exists\, \alpha_i$ such that $a_i > 0$, where a_i is one membership degree in the fuzzy relationship, and (3) the cardinality constraints 1:1, 1:N, and N:M are also studied with fuzzy relationships.

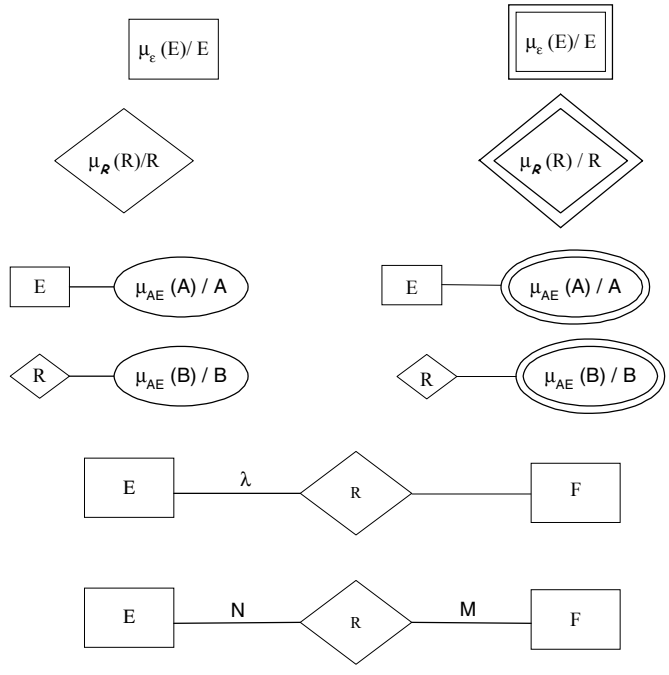
The fuzzy ER model (Chen, 1998) proposes a model generated by M = (***E***, ***R***, ***A***) expressed by E as entity type, R as interrelation type, and A as attributes, also including label types that generate, at the first level, L1(M) = (***E***, ***R***, $\boldsymbol{A}_E$, $\boldsymbol{A}_R$), and proposes four set types, with notation shown in Figure 3.3 (see an example in Figure 3.1), and where μ_X is the membership function to the set X (one Entity, one Relationship or one Attribute) and $D_{\boldsymbol{E}}$ is the domain of ***E*** composed of all possible entity types concerned:

- $\boldsymbol{E} = \{\mu_{\boldsymbol{E}}(E) / E : E \in D_{\boldsymbol{E}} \text{ and } \mu_{\boldsymbol{E}}(E) \in [0,1]\}$.
- $\boldsymbol{R} = \{\mu_{\boldsymbol{R}}(R) / R : R \text{ is a relationship type involving entity types in } D_{\boldsymbol{E}} \text{ and } \mu_{\boldsymbol{R}}(E) \in [0,1]\}$.
- $\boldsymbol{A}_E = \{\mu_{AE}(A)/A : A \text{ is an attribute type of entity type E and } \mu_{AE}(A) \in [0,1]\}$.
- $\boldsymbol{A}_R = \{\mu_{AR}(B)/B : B \text{ is an attribute type of relationship type R and } \mu_{AR}(B) \in [0,1]\}$.

The participation constraint (Figure 3.3) is modeled setting that an entity E λ-participates in R if for every e of E, there exists an f in F such that $\mu_R(e,f) >= \lambda$. The cardinality constraint is shown at the end of Figure 3.3, where N and M are fuzzy sets. The concept of fuzzy quantifier is not used in this approach.

At the second level, for each entity type E and relationship type R, the sets of their values can be fuzzy sets, reflecting possible partial belonging of the corresponding values to their types. The third level of fuzzy extensions concerns attributes and their values. For each attribute type A, any of its values can be a fuzzy set.

Figure 3.3. ER fuzzy notation proposed by Chen (1998)



Later on, in another section, an attribute-defined specialization is defined with $FS_i \in F(Dom(A))$, where all the FS_i are fuzzy sets on Dom(A), the domain of the attribute A. Graphically, this kind of attribute-defined specialization can be represented as shown in Figure 3.4a. Figure 3.4b shows the entity Employee and the fuzzy attribute Age with the labels "Young Employee," "Middle-Aged Employee," and "Old Employee." He also includes the fuzzy definition for categories and shared subclass, that is, union and intersection (see Figure 3.4c). This proposal always makes reference to linguistic labels and to the trapezoidal function over an attribute or specific entity, not to a set of different attributes or different entities. This author, just like Yazici and Merdan (1996), establishes his data models from the attributes and creates the object class or entity by using generalization and specialization tools.

Chen (1998) defines that a linguistic variable X is composed of the tuple (T, U, G, M), where T is the set of linguistic terms of X, U is the universe of discourse, G is the set of syntactic rules that generate the element T, and M is the set of semantic rules translated from T that correspond to the fuzzy subset of U. With this definition, he defines a conceptual model and its mathematical representa-

Figure 3.4: Notation proposed by Chen (1998):
a) An attribute-defined overlapping specialization with $FS_i \in F(Dom(A))$ at the first level; b) Employee in an overlapping specialization with the fuzzy attribute Age; and c) Shared subclass Intersection

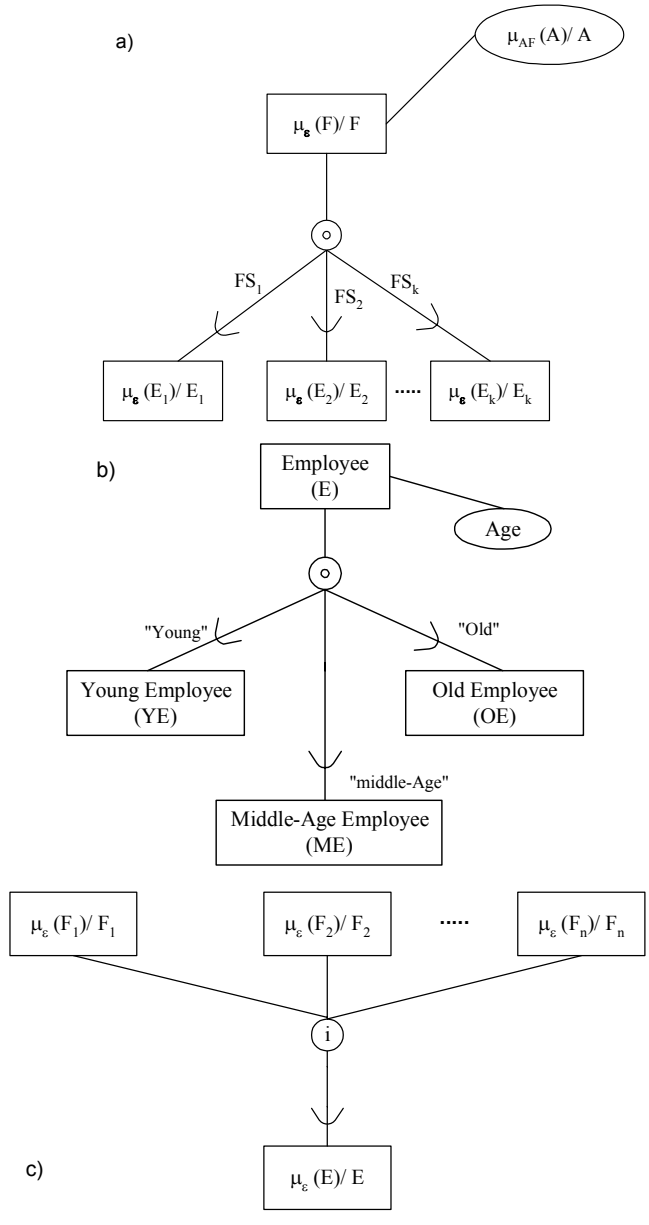
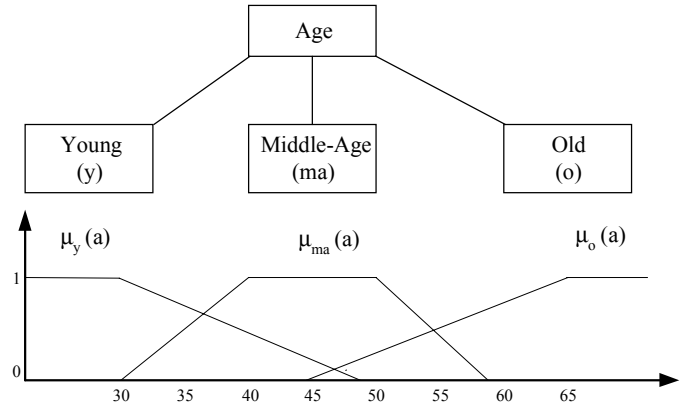


Figure 3.5. Linguistic variable "Age" with its corresponding values and conceptual model, according to Chen (1998)

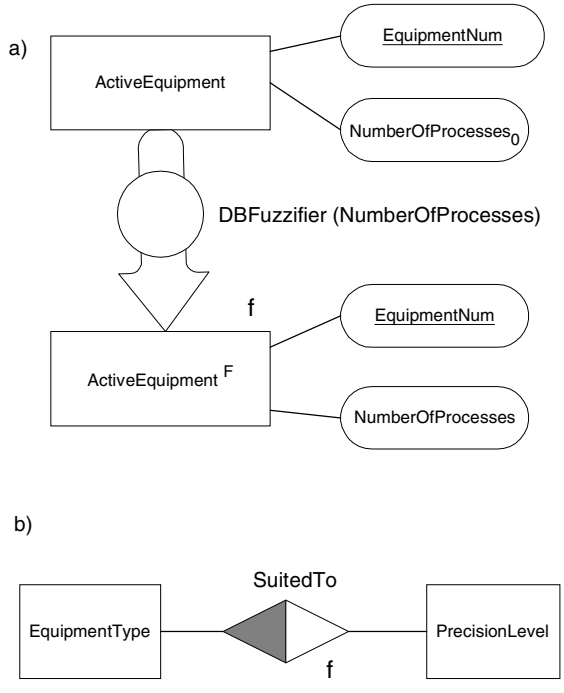


tion. For example, let X = Age in Figure 3.5, and T is generated via G by the set {Young, Middle-Aged, Old}. Each term of T is specifically handled by M by fuzzy sets. The type of correspondence between an entity and a fuzzy entity is also established, as well as the set of values that a membership degree obtains from a fuzzy set: 1:1, 1:N, N:M, incorporating fuzziness to the ER model.

The Chaudhry, Moyne, and Rundensteiner Approach

Chaudhry, Moyne, and Rundensteiner (1994, 1999) propose a method for designing Fuzzy Relational Databases (FRDB) following the extension of the ER model of Zvieli and Chen (1986), taking special interest in converting crisp databases into fuzzy ones. The way to do so is to define *n* linguistic labels as *n* fuzzy sets over the universe of an attribute. Then, each tuple in the crisp entity is transformed to up to *n* fuzzy tuples in a new entity (or *n* values in the same tuple). Each fuzzy tuple (or value) does not store the crisp value but a linguistic label and a grade of membership, giving the degree to which the corresponding crisp entity belongs to the new entity. Finally, the crisp entity and the new fuzzy entity are mapped to separate tables.

Figure 3.6. Model proposed by Chaudhry, Moyne, and Rundensteiner (1994): a) example of DBFuzzifier transformation and b) fuzzy interrelation for labels



Their ER model includes fuzzy relationships as relationships with at least one attribute, namely, the membership grade. They propose FERM, a design methodology for mapping a fuzzy ER data model to a crisp relational database in four steps (constructing a fuzzy ER data model, transforming it to relational tables, normalization, and ensuring correct interpretation of the fuzzy relational operators). They also present the application of FERM to build a prototype of a fuzzy database for a discreet control system for a semiconductor manufacturing process.

Chaudhry, Moyne, and Rundensteiner (1999) expand the model presented in their 1994 paper, focusing on their proposal for the *control processes* example. In each process, imprecise values are observed and associated to linguistic labels, and every value involves a process called "DBFuzzifier construct," as shown in Figure 3.6.

Proposal of Ma, Zhang, Ma, and Chen

Ma, Zhang, Ma, and Chen (2001) work with three levels of the work of Zvieli and Chen (1986) and incorporate into the Fuzzy Extended Entity-Relationship model (FEER model) a way of managing complex objects in the real world at a conceptual level, associating an importance degree of each of the components (attributes, entities, etc.) to the scheme. However, their definitions of generalization, specialization, category, and aggregation impose very restrictive conditions. They also provide an approach to mapping an FEER model to a Fuzzy Object-Oriented Database scheme (FOODB).

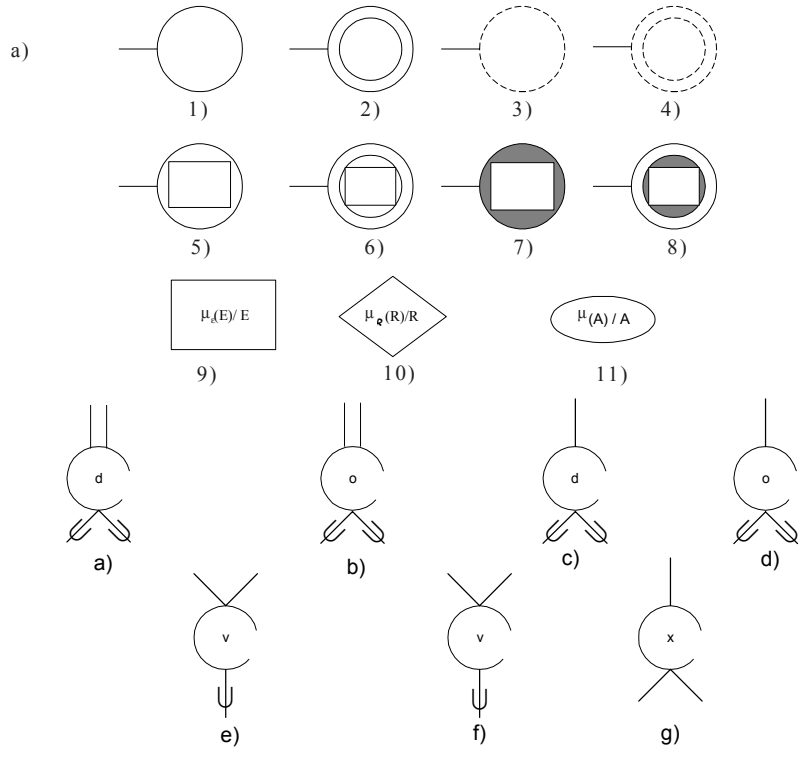
Figure 3.7a shows the following: 1) single-valued attribute type, 2) multivalued attribute type, 3) disjunctive fuzzy attribute type, 4) conjunctive fuzzy attribute type, 5) null attribute type, 6) open or null attribute type, 7) disjunctive imprecise attribute type, 8) conjunctive imprecise attribute type, 9) entity with grade of membership, 10) relationship with grade of membership, and 11) attribute with grade of membership.

In addition, Figure 3.7b shows the following notations: 1) fuzzy total and disjoint specialization, 2) fuzzy total and overlapping specialization, 3) fuzzy partial and disjoint specialization, 4) fuzzy partial and overlapping specialization, 5) fuzzy subclass with fuzzy multiple superclasses, 6) fuzzy category, and 7) fuzzy aggregation.

Figure 3.8 shows an example of the EER fuzzy model utilizing some of the notions proposed by Ma, Zhang, Ma, and Chen (2001). Thus, the "car" entity is a superclass with two fuzzy subclasses "new car" and "old car" in an overlapping specialization. Besides, the "young employee" fuzzy entity, having fuzzy instances from the "company" entity, consists of the "union" category from the fuzzy entity "buyer." Also, "young employee" has a fuzzy relationship, "like." Finally, the "car" entity is an aggregation of some entities: "engine," "chassis," "interior," and "radio" (with an associated fuzzy degree of 0.7). Note that "engine" has some fuzzy attributes, such as *size* and *turbo*.

Ma, Zhang, and Ma (2004) introduce an extended object-oriented database model to handle imperfect as well as complex objects. They extend some major notions in object-oriented databases such as objects, classes, objects-classes relationships, subclass/superclass, and multiple inheritances.

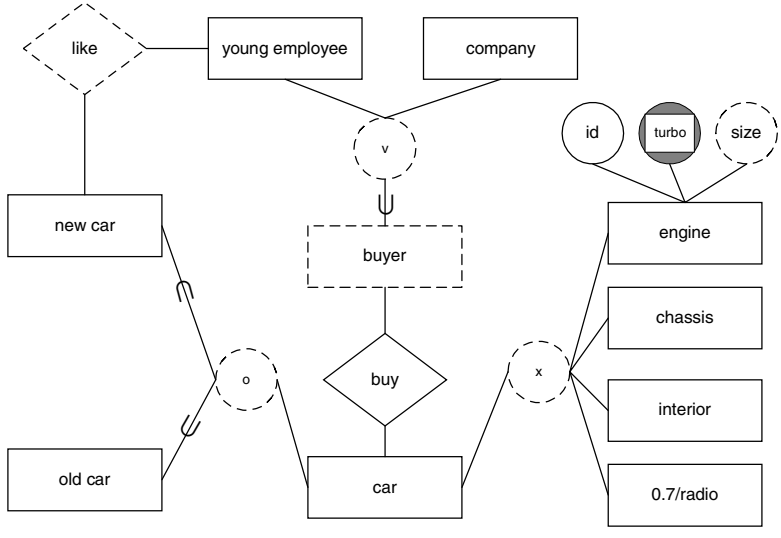
Figure 3.7. FEER notation by Ma, Zhang, Ma, and Chen (2001): a) fuzzy attributes, entities, and interrelations, and b) specialization, aggregation, and fuzzy categories



Approaches by Other Authors

Ruspini (1986) proposes an extension of the ER model with fuzzy values in the attributes, and a truth value can be associated with each relationship instance. In addition, some special relationships such as same-object, subset-of, member-of, and so on are also introduced. Vandenberghe (1991) applied Zadeh's extension principle to calculate the truth value of propositions. For each proposition, a possibility distribution is defined on the doubleton true, false of the classical truth values. In this way, the concepts such as entity, relationship, and attribute, as well as subclass, superclass, category, generalization, and specialization, have been extended.

Figure 3.8. Example of Ma, Zhang, Ma, and Chen notation (2001) for a car assembly company case



The proposal of Vert, Morris, Stock, and Jankowski (2000) is based on the notation used by Oracle and uses the fuzzy sets theory to treat data sets as a collection of fuzzy objects, applying the result to the area of Geospatial Information Systems (GIS).

Another line of work in fuzzy conceptual data modeling (without using the Entity-Relationship model) is reported in Fujishiro et al. (1991), using a graph-oriented schema for modeling a fuzzy database. Fuzziness is handled by defining various links between records of the value database (actual data values) and the explanatory database (semantic interpretation of fuzzy attributes, symmetries, and so on). Extensions carried out to allow modeling imprecision in semantic data models are also described in Rundensteiner and Bic (1989), focusing on exploring the potential of semantic data models to allow fuzziness to be represented.

Van Gyseghem and de Caluwe (1997) discussed two types of imperfect information appearing in database applications: fuzzy information representing information with inherent gradations, for which it is impossible to define sharp or precise borders, and uncertain or imprecise information, representing information that is (temporarily) incomplete due to a lack of sufficient or more precise knowledge. Dealing with this kind of imperfect information within the

formal and crisp environment of a computer, is based in this paper upon the fuzzy set theory and its related possibility theory, which offers a formal framework to model imperfect information, and upon the object-oriented paradigm, which offers flexible modeling capabilities. The result is the UFO database model, a "fuzzy" extension of a full-fledged object-oriented database model.

This research discusses the UFO database model in detail in three steps. First, it shows how fuzzy information is handled: Meaningful fuzzifications of several object-oriented concepts are introduced in order to store and maintain fuzzy information and to allow a flexible or "soft" modeling of database application. Then, it discusses how uncertainty and imprecision in the information are handled: Possible alternatives for the information are stored and maintained by introducing role objects, which are tied like shadows to regular objects in the database and allow the processing of uncertainty and imprecision to the user in an implicit and transparent way, and they also allow the modeling of tentative behavior and of hypothetical information in the database application. Both the static and the dynamic aspects of (imperfect) information are developed in the UFO database model, and imperfect information is considered at the data level as well as at the metalevel of a database application. The process of "extending" an object-oriented database model to the UFO database model, as discussed here, adheres, as closely as possible, to the original principles of the object-oriented paradigm to allow a flexible and transparent, but semantically sound, modeling of imprecise information. The object-oriented database model, from which the extension process starts, adheres to the standard proposal ODMG-93 to allow for practical implementations of the UFO database model. For the same purpose, the authors' paper also discusses an interface of the UFO database model to an extended relation database model that is capable of handling some imperfect information and for which some prototypes are already available.

Chapter IV

FuzzyEER: Main Characteristics of a Fuzzy Conceptual Modeling Tool

In this chapter we present the FuzzyEER Model, which is an extension of the EER Model with fuzzy semantics and notations. The Entity-Relationship Model was introduced by Chen (1976). Since then, numerous modifications and extensions of its modeling capabilities have been suggested. We will mainly use the approach by Elmasri and Navathe (2000) because it is very popular, general, and has an international scope.

With regard to the fuzzy attributes, the following aspects will be defined in the FuzzyEER Model: imprecise attributes, fuzzy attributes associated to one or more attributes or with an independent meaning, as well as degrees of fuzzy membership to the model itself. Furthermore, the following concepts will also be defined: fuzzy aggregation, fuzzy entity, weak fuzzy entity, fuzzy relationship, and defined specialization with fuzzy degrees. Fuzzy constraints are very important, and we review them in this chapter as well.

Finally, we also include a table to compare FuzzyEER and the modeling tools for the currently most important published fuzzy databases. Appendix A summarizes the conventions for FuzzyEER diagrams.

A Brief Introduction to the ER/EER Model

The ER Model graphically represents data as entities, relationships, and attributes. Entities are objects that exist in the real world and are represented in the model by rectangles. Relationships relate different entities to each other and are represented with diamond shapes. Both entities and relationships can have different attributes, which identify or characterize them.

The EER Model allows us to extend the description of the entities with new types (superclasses, subclasses, and categories). A *subclass* is a specialization of a superclass, so that each member of a subclass must be a member of the superclass. A *superclass* is a generalization of one or several subclasses. A specialization to which the superclass and all its subclasses are connected is represented with a circle. Subclasses are marked with the inclusion symbol (Ì) in the connecting line. A shared subclass is a subclass with various superclasses so that every member (or instance) of the subclass must belong to all the superclasses. Naturally, a subclass inherits every attribute of all its superclasses. On the other hand, a category (union type) is similar to a shared subclass, in which every member of the subclass must belong to only one of the superclasses, inheriting only the attributes of that superclass.

Fuzzy Values: Fuzzy Attributes and Fuzzy Degrees

In this section we refer to the fact that the attributes of an entity may be fuzzy values, and it is possible to operate with these values. The imprecision may be expressed in some ways, and we have classified it in two basic types. The first one we will properly call *fuzzy attributes*, and the second one we will call *fuzzy degrees*. As you will see, we based our work on works by many

authors, but we especially want to remark about the GEFRED Model (refer to Chapter II).

Fuzzy Attributes

In order to model fuzzy attributes we classify data in two categories: ordered and nonordered referential (or underlying) domain. In an ordered referential we can express any type of possibility distribution or fuzzy set defined over this referential, but in short, we can synthesize the following types of possibility distribution: trapezoidal (such as those that you see in Figure 1.1 or Figure 4.2), linguistic labels (associated to specific possibility distributions), approximate values (triangular distributions), and intervals of possibility. In a nonordered referential there are simple "scalars" (or labels) and possibility distribution over scalars (such as Equation 1.3). In both contexts, the values Unknown, Undefined, and Null are allowed and are defined as in Umano and Fukami (1994).

With these concepts, we classify fuzzy attributes as the following four types, based on the definitions of Medina (1994) and Galindo (1999). This classification is performed by taking into account the type of referential or underlying domain. They allow the representation of imprecise information or if the values only fuzzy processing only on crisp data. There are four possible types of fuzzy attributes:

- **Type 1**: These are attributes with *precise data*, classic or crisp (traditional, with no imprecision). They can have linguistic labels defined over them. This type of attribute is represented in the database system in the same way as precise data but can be transformed or manipulated by using fuzzy conditions. This type is practical for extending traditional databases, allowing fuzzy queries to be made about classic data. In this type of attribute we will not be able to store imprecision, and therefore, strictly speaking, they are not fuzzy, although they do allow fuzzy queries or manipulations to be carried out — for example, queries of the kind, "Give me employees that earn a lot more than the minimum salary."
- **Type 2**: These are attributes that gather *imprecise data over an ordered referential*. These attributes admit both crisp and fuzzy data in the form of possibility distributions over an underlying ordered domain. It is an extension of the Type 1 that allows the storage of imprecise information, such as "he is approximately 2 meters tall."

- **Type 3**: They are attributes over *data of discreet nonordered domain with analogy*. In these attributes, some labels are defined (e.g., "blond," "ginger," "brown," etc.) that are scalars with a similarity relationship (or proximity) defined over them (Definition 1.8). Thus, this fuzzy relation indicates to what extent each pair of labels is similar. As we saw, they also allow possibility distributions over this domain (Equation 1.3), for example, the value (1/dark, 0.4/brown), which expresses that a certain person is more likely to be dark than brown-haired but will definitely not be blond or ginger.
- **Type 4**: These are attributes proposed in this model, and they are defined in the same way as Type 3 attributes, without it being necessary for a similarity relationship to exist between the labels (or values) of the domain. In this case, the main thing is the degree associated to each individual label, without evaluating the similarity between labels. For this reason, in this case, it will, in principle, be more usual to find non-normalized values. A possible example could be the type of role a client plays in a real estate agency, where the degree measures the importance with which a client is seeking or offering a property, without taking into account the similarity between the two roles.

It should be noted that the fuzzy set of possible values for an attribute is called *fuzzy domain*.

Fuzzy Degrees

In the previous section, different ways of including we explain fuzzy values in attributes. However, there is another way of incorporating uncertainty in a database, which consists of using fuzzy degrees (Galindo, 1999; Galindo, Medina, Cubero, & García, 2001). The domain of these degrees may be the interval [0,1], although other values are also permitted, such as possibility distributions (usually over this unit interval), which, in turn, may be related to specific linguistic labels (such as "a lot," "normal," etc.). In order to keep it simple, we will use only degrees in the interval [0,1], because the other option offers no great advantages and a greater technical and semantic complexity. In the next sections you see some examples that use the definitions in this section.

The following are some of the most important ways of using these fuzzy degrees:

- **Associated degrees**: These degrees are associated to a specific value and incorporate imprecision to the associated value. They may be associated to different concepts:
  - **Degree in each value of an attribute**: An instance can have several attributes. In turn, some of these attributes may have a fuzzy degree associated to them. This implies that each value of this attribute has an associated degree that measures the level of fuzziness of that value.
  - **Degree in a set of values of different attributes**: This is an intermediate case between the previous cases. Here, the degree is associated to some attributes. Although this case is unusual, it is sometimes very useful (Galindo, Medina, Cubero, & Pons, 1999).
  - **Degree in the whole instance of the entity**: This is similar to the previous case, but here the degree is associated to the whole instance of the entity and not exclusively to the value of a specific attribute of the instance. It can measure to what degree this tuple (or instance) belongs to this table (or entity) of the database (Umano & Fukami, 1994).
- **Nonassociated degrees**: There are cases in which we wish to express imprecise information, which can be represented by using only the degree, without associating this degree to another specific value or values.

On the other hand, the meaning of these degrees can be varied and depends on their use. The processing of the data will be different depending on the meaning. You will see some examples of each type when we use these concepts in the third section of this chapter. The following are the most important possible meanings of the degrees:

- **Fulfillment Degree**: A property can be fulfilled with a certain degree between two extremes: total fulfillment (usually degree 1) and no fulfillment at all (usually degree 0). This is usually used in fuzzy queries, when a condition has been established over the values of an entity or relationship (i.e., a selection with a fuzzy condition), and the degrees will express to what extent this condition has been fulfilled. For example, the GEFRED Model (Medina, Pons, & Vila, 1994; refer to Chapter II) uses an associated degree in each value of an attribute with this meaning. Some other authors use this meaning (Bosc, Dubois, Pivert, & Prade, 1997).

- **Uncertainty Degree**: The uncertainty degree expresses the certainty with which a specific piece of data is known. When certainty exists regarding the truth of the degree, then it will be 1, and if certainty exists that it is false, the degree will be 0. The values between 0 and 1 express different levels of uncertainty, indicating that there is doubt. This meaning is to some extent quite similar to the previous one, as this uncertainty may be in relation to the membership of an object to a set (Umano & Fukami, 1994).
- **Possibility Degree**: It measures the possibility of the information used. This meaning is similar to the previous one but can be seen to be a weaker degree because it contains information that is more or less possible and not more or less certain (Umano & Fukami, 1994; Medina, Pons, & Vila, 1994).
- **Importance Degree**: Different objects (i.e., instances, attributes, etc.) can have different importance, so that some objects are more important than others (Mouabdid, 1994; Bosc, Dubois, Pivert, & Prade, 1997).

When defining fuzzy sets, we speak of a **membership degree** to a fuzzy set. This meaning be added to the degrees previously described. Various authors have studied the degrees of fulfillment, uncertainty, possibility, importance, and membership (Medina, Pons, & Vila, 1994, Mouaddib, 1994; Petry, 1996; Bosc, Dubois, Pivert, & Prade, 1997; Dubois & Prade, 1998; Galindo, 1999; Galindo, Medina, Cubero, & García, 2001; Ma, Zhang, Ma, & Chen, 2001). In this chapter we do not aim to demonstrate the usefulness of these degrees and their different meanings, because the authors who have used these degrees with different meanings have already done so. Our objective is to show the need for representing it in a conceptual model based on the EER Model, and we do so in the following section.

Fuzzy Attributes in FuzzyEER Model

In the preceding section, we define several ways of allowing imprecise information. In this section we define the application of each of these cases in the conceptual model called FuzzyEER, including its representation or graphic notation. Examples to clarify each definition are also given. In this section do

not deal with the case of a degree associated to the whole instance, which we look at in later sections.

Fuzzy Values in Fuzzy Attributes

Fuzzy values in the attributes refer to the different kinds of attributes of an entity that can model fuzzy values (imprecise), and we classified them in the section on fuzzy attributes. This corresponds with the third level of Zviely and Chen (1986), but in a clearer and more complete way, according to the given classification.

The graphic representation of the attributes in the EER Model is made by means of a circle, labeled with the name of the attribute and joined by a line to the entity or relationship to which it belongs (De Miguel, Piattini, & Marcos, 1999). Some authors place the name of the attribute inside the circle (which can be an oval of differing sizes in order to accommodate the name). However, we believe that it is better to place the name outside the circle either on the left- or right-hand side, so that the circle can be of a set size and does not depend on the number of letters in the name of the attribute.

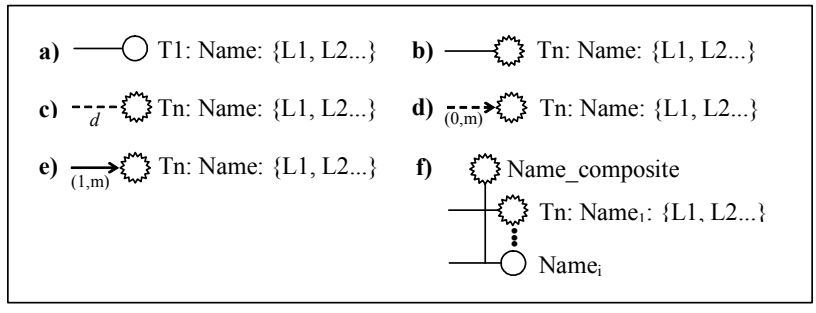
Definition 4.1: *A type of entity or a relationship E with attributes A_1, A_2, ..., A_n and D_1, D_2, ..., D_n being their respective domains. A fuzzy attribute is represented in the FuzzyEER Model depending on its type:*

- *Fuzzy attribute Type 1: It is graphically represented in the same way as a classic attribute, with a normal circle with a line that joins it to the entity, first placing the name of the attribute* T1 *followed by a colon. See* Figure 4.1a.
- *Fuzzy attributes Type 2, 3, and 4: Their representation is similar except that a circle of stars is used, and the marks* T2*,* T3*, or* T4 *are placed before the name of the corresponding attribute. See* Figure 4.1b.

Another option is to include next to the name of the attribute a list in brackets with the linguistic labels that are defined for this attribute: {L1, L2...}. These labels must be defined in the Data Dictionary of the model.

*

Figure 4.1. Graphic representation in FuzzyEER for a) fuzzy attribute Type 1 (simple); b) fuzzy attribute Type n, with n∈{2,3,4} (simple); c) derived fuzzy attribute, d) optional multivalued fuzzy attribute; e) multivalued fuzzy attribute with a minimum compulsory value; and f) generic example of a composite attribute with a fuzzy component



Note that in the case of classic attributes a normal circle is used, and although the Type 1 is included in fuzzy attributes, it does not in fact admit fuzzy information (it simply allows fuzzy processing of nonfuzzy data). For this reason Type 1 is not represented by a circle of stars like the rest of the fuzzy types, but it does allow the declaration of fuzzy labels that can be used in different operations (e.g., queries).

In De Miguel, Piattini, and Marcos (1999), a classification is proposed for the attributes defined in the EER Model:

1. **Identifying attribute** or **primary key**, denoted by a solid black circle. In order to avoid problems of ambiguity, distinguishability, or others, the primary key cannot be represented by fuzzy values (Medina, Pons, & Vila, 1994; Medina, 1994).
2. A **simple attribute** takes only a single value in a domain (for each instance). This is the most usual kind of attribute. For example, a person's weight or height is a simple attribute, as only a single value is possible, although it may or may not be fuzzy.
3. A **derived attribute** is obtained from already existing ones (these may or may not be fuzzy). For example, age can be obtained from the date of birth, and if that date is fuzzy, then age will be fuzzy. These attributes may be associated to a rule *d* for derivation that defines how to make the

calculation. Graphically, a derived attribute is represented by using a dashed or broken line on the line that joins the circle of the attribute with the entity or relationship to which it belongs. This dashed line may also be labeled with the rule of the derivation, *d*.

4. A **multivalued attribute** is one that takes more than one value in a domain (for each instance). For example, a person can have several values for the attribute "telephone." Graphically, this is expressed by changing the line of the attribute to an arrow, which points towards the circle. This arrow is labeled with two whole numbers, between round brackets and separated by commas, that define the minimum and maximum number of possible values, taking into account that
   - If the minimum value is 0, it indicates that this attribute is optional and that it cannot take any value. In this case the line of the arrow may be dashed (broken).
   - If a value is at least compulsory, then the minimum value will take a whole number that is equal to or greater than 1, and the line of the arrow will be normal (not dashed).
   - If the minimum or maximum number is unknown, we will use a letter.
5. A **composite attribute** is composed of other attributes that can be of any type. For example, an address can be composed of a street, number, city, and so forth. A composite attribute is denoted by a normal circle with a line that crosses all the lines of attributes that compose it. All components of a composite attribute are represented normally (i.e., joined to the type of entity or relationship to which it belongs).

Definition 4.2: *The simple fuzzy attributes (Definition 4.1), and the derived and multivalued attributes (optional and compulsory) preserve the same definition and graphic representation when they are fuzzy. That is to say, it is only necessary to place before the name of the attribute the mark that distinguishes the type of fuzzy attribute (T1, T2, T3, or T4) and change the circle to a circle of stars (except for T1). See Figures 4.1b, c, d, and e, respectively.*

In composite attributes, when one of the components is fuzzy, this component will be represented according to its type. See a generic example in Figure 4.1f.

*

Example 4.1: In a theatrical company let us take the entity Employee, which represents the actors of the company. In this entity, we can consider in a simplified form the following attributes: Employee_ID, Height, Age, Color_hair, and Color_skin. Some of these attributes can be characterized as fuzzy attributes:

- **Employee_ID**: This is a classic attribute, and its domain will correspond to numerical values, as this attribute is defined as a primary key for the entity Employee.
- **Height**: It is a fuzzy attribute Type 1, and its domain can be defined as the set of values in the interval (0, 2.5). We understand that it is a *precise attribute*, but we can define linguistic labels (such as "short," "medium_height," and "tall") to be used when manipulating these data (e.g., in queries). In this case, it is supposed that the exact height of an employee is known, and if we do not know that height, then we must store the null value (even though we know something about it). Note that this attribute could be Type 2 if we wish to store fuzzy information about the height of an actor, such as "about 1.72 meters tall," or "tall," for example.
- **Age**: It is a fuzzy attribute Type 2, and its domain corresponds to a fuzzy set of ages considered as numerical values (ordered referential), allowing linguistic labels such as "young," "mature," and "elderly." These labels are represented and defined as *possibility distributions*. In Figure 4.2 these labels are defined. Note, for example, how the label "young" is defined as a trapezoidal function with its four characteristic values (0/15, 1/20, 1/25, 0/30), and how the age 26 belongs to the label "young" (or to the set of young people) in a degree of 0.8.

Figure 4.2. Example 4.1: Possibility distributions for the linguistic labels of the fuzzy attribute T2: Age

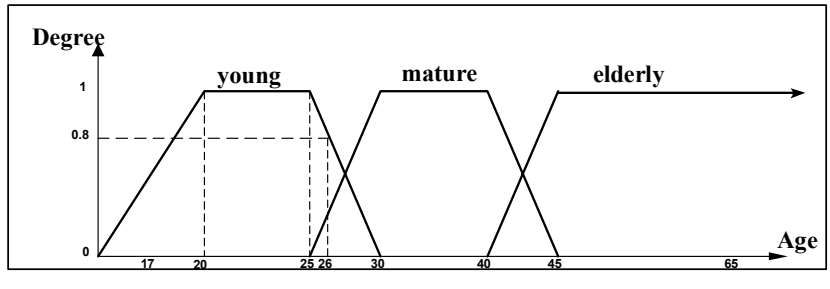


Table 4.1. Similarity function between the labels of the fuzzy attribute Type 3 Color_hair (Example 4.1)

	Blond	Dark	Ginger
Blond	1	0.1	0.8
Dark	0.1	1	0.3
Ginger	0.8	0.3	1

- **Color_hair**: It is a fuzzy attribute Type 3. Its underlying domain will correspond to the set {"blond," "dark," "ginger"}; each of these labels must be associated to a value of resemblance or similarity in the interval [0,1]. It should be noted that the values of this domain are not ordered, because no such relationship exists. Their associated similarity function is represented in Table 4.1. The real domain is made up of simple values in this referential (for example, 1/blond) or possibility distributions (for example, {1/blond, 0.6/ginger}).
- **Color_skin**: It is a fuzzy attribute Type 4, and its underlying domain has the labels {"white," "yellow," "brown," "black"}. The fuzzy domain is formed by adding a single degree to one or several labels of the underlying domain. For example, a valid value is {0.8/white, 0.6/yellow, 0.4/brown, 0.1/black}. This differs from the previous case, as in this attribute there is no table of similarity, because we assume that we are not interested in establishing the similarity between labels. Rather, for each actor, we wish to store information as precise as possible regarding the similarity degree between his/her own skin color and the labels defined for this attribute.

Figure 4.3. Example 4.1: Entity Employee for the actors of a theatrical company, with fuzzy attributes Type 1, Type 2, Type 3, and Type 4

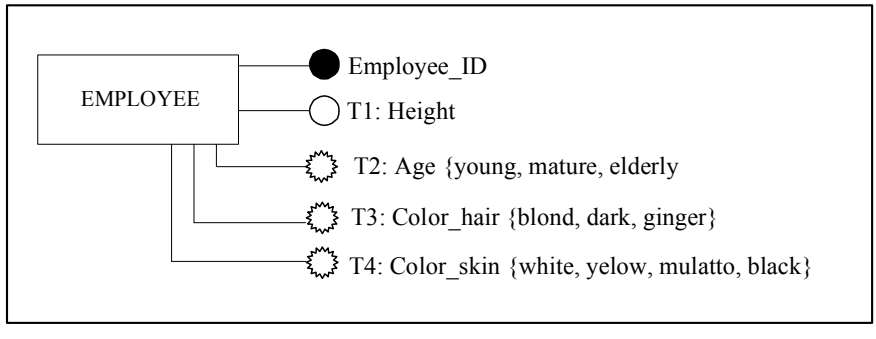


Figure 4.3 represents the entity Employee with the notation of the FuzzyEER Model. It defines fuzzy attributes Type 1, Type 2, Type 3, and Type 4, according to this example. This entity may be very useful because it is very common to seek an actor with particular features to represent a specific character that requires particular physical characteristics.

*

Fuzzy Degree Associated to Each Value of an Attribute

In this case, each value of an attribute (for each instance) can have an associated degree, generally in the interval [0,1]. The degree measures the level of vagueness of that value. The significance of these degrees can vary as explained in Chapter II.

Definition 4.3: *Let E be an entity or a relationship with attributes (A_1, A_2, ...,A_n). For each attribute A_i with $i \in \{1,..., n\}$ (whether fuzzy or not), there may be one or several fuzzy degrees associated to each value of each instance of E. These degrees are represented as normal attributes but with a circle with a dashed outline, joined by a line to the object E to which it belongs. This degree will be placed next to the attribute that has that degree associated to it, and in order to indicate it, an arrow originating from its line points towards the line of the degree.*

Furthermore, this degree may have different meanings that are expressed by labeling the dotted circle with the expression G^n, or $G^{Meaning}$, where "n," or its meaning, is one of those that are later defined or other new ones. This expression can be optionally followed by the name of the attribute A_i and, between brackets, any type of expression that can help to clarify the meaning of that degree in each specific context and/or its use. Generically, we can define the following meanings:

- ***Membership Degree**: The membership of a value to a specific instance can be quantified by a degree. It is represented by G^0 or $G^{Membership}$.*
- ***Fulfillment Degree**: In an instance, a property can be satisfied in an attribute with a certain degree between two extremes. It is represented by G^1 or $G^{Fulfillment}$.*

- ***Uncertainty Degree****: The uncertainty degree expresses to what degree we are certain that we know a specific piece of data for a specific instance. It is represented by G^2 or $G^{Uncertainty}$.*
- ***Possibility Degree****: It measures the possibility of the information that is being modeled for each piece of data. It is represented by G^3 or $G^{Possibility}$.*
- ***Importance Degree****: Different values of an attribute can have different importance, so that certain values of certain attributes are more important than others. This importance may or may not depend on the instance. It is represented by G^4 or $G^{Importance}$.*

If the meaning is not defined as 0, 1, 2, 3, or 4, by default the attribute will represent the membership degree G^0.

*

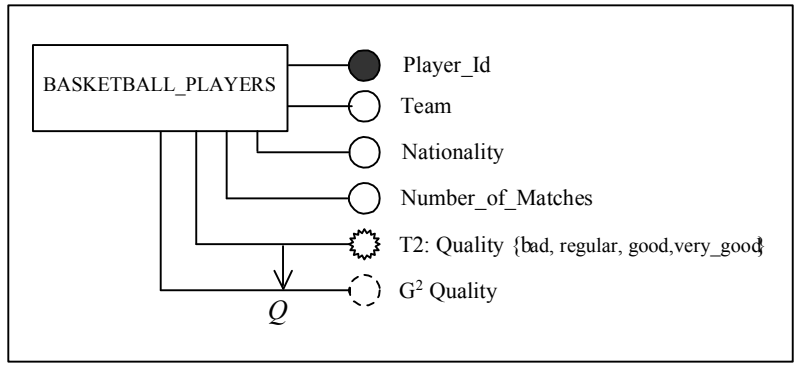
Definition 4.4: *Two types of fuzzy degrees associated to an attribute exist:*

1. ***Derived fuzzy degree****: If a function Q(x) exists that defines the calculus of those degrees. That is, the function allows the degrees to be automatically calculated based on the value of other attributes or on any other information stored or available in the database. This function could be, for example, a possibility distribution of a linguistic label, when x is the value of the attribute to which the degree is associated. In this case the function Q will be used by labeling the dashed line that joins the dashed circle to its owner.*
2. ***Nonderived fuzzy degree****: In this case a function for automatically calculating the degrees has not been defined. This case includes the cases in which those degrees are introduced manually by the user.*

*

It is important to distinguish between the value of the fuzzy attribute and the value of the degree associated to that value. For example, if for a certain person hair color is important (or a fulfillment degree of a particular property), then that importance degree is independent of the person's hair color. It is also independent of the type of that attribute (Type 3 in the hair color case). That is to say, the degree represents an importance degree of the whole value that

Figure 4.4. Example 4.2 case b): fuzzy derived degree and associated to the fuzzy attribute Type 2 Quality, with meaning of uncertainty degree (G^2)



hair color takes. Furthermore, it is important to point out that this attribute can have a fuzzy value. It can be considered as a double attribute because it has two related values (the value of the attribute, whether fuzzy or not, and the degree associated to it).

Example 4.2: Let us imagine an entity for representing Basketball Players. Each player has certain classic attributes, like Player_Id, Team, Nationality, Number_of_Matches, and so forth. Let us consider a fuzzy attribute Type 2 Quality whose referential is the average number of points scored per match (positive real numbers). The labels (Bad, Regular, Good, Very_Good) can be defined for this attribute. Moreover, a degree that can have different meanings can be associated to this attribute. The meanings may be as follows:

1. If we consider the derived type degree, and the function $Q(x)$ is the membership function of a label, such as "Good," then this degree represents the membership degree of each player to the fuzzy set composed of the "Good" players. In this case, the dashed circle could be labeled as G^0 Quality (degree of "Good").
2. The known quality of a player will be more certain if the player has played a lot of matches. Thus, if this degree is labeled as G^2 Quality, we are referring to the fact that this degree measures the uncertainty or reliability of the value of the attribute Quality. This can also be a derived degree if we define the function Q, which can depend on the attribute Number_of_Matches. This case is represented in the Figure 4.4.

3. For each player this evaluation of Quality may be more or less important, depending on his usual position in the team. In this case we can express it by labeling the degree with G^4 Quality.

*

The case (1) of the previous example allows us to store a value that may be useful for helping to speed up a query of the type "give me good players." However, it is clear that this degree is not necessary for this type of queries, and therefore the function that defines the calculus of the degree will not normally be a label defined for the same attribute to which the degree is associated. If this information is necessary, it can be calculated at any moment, although the conceptual model does not express it explicitly.

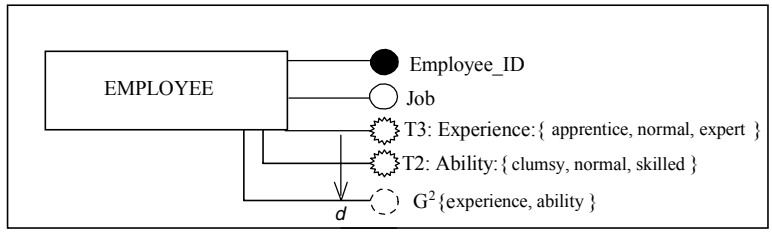
On the other hand, an attribute can be associated to several degrees, each of which may have a different meaning. In particular, the meanings of the sections (1), (2), and (3) of the previous example are perfectly compatible, and it could be useful to define two or three of them simultaneously in our FuzzyEER Model.

Fuzzy Degree Associated to Values of Some Attributes

In this case a degree associated to a set of attributes belonging to an entity is required. The degree is not associated to a single attribute but to a set of attributes. This case is unusual, but it can add expressiveness if it is incorporated to a data model. An example of this attribute can be found in the concept of fuzzy dependence proposed by Raju and Majumdar (1988).

Definition 4.5: *Let E be an entity (or relationship) with attributes (A_1, A_2, ...,A_n), and X a subset of these attributes: $X = \{A_i : i \in I\}$ of attributes, with I being a set of indices so that $I \subset \{1,..., n\}$. We can define a fuzzy degree associated to each value that the attributes of X take in each instance of E. This degree is named Degree associated to the values of different attributes. In a FuzzyEER Model this type of attribute is represented in a similar way to that explained in the Definition 4.3, but now the arrow will cross all the attributes of X (those that are associated with this degree). To make it clearer the attributes of X can be placed after the $G^{Meaning}$ of the degree. In general, this degree will be derived from the values of those attributes.*

Figure 4.5. Example 4.3: Fuzzy degree associated with two attributes (both being fuzzy)



Derived fuzzy degrees are represented like the Definition 4.4.

*

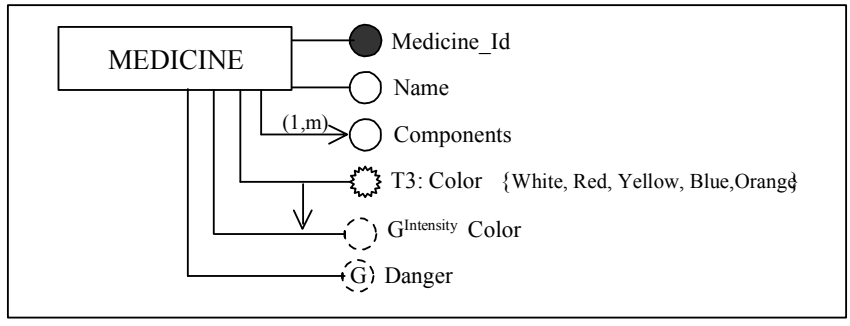
Example 4.3: Let us suppose the attributes (Employee_ID, Job, Ability, Experience) of an Employee entity. In this case, the attribute Job is considered to be classic or ordinary (alphabetical), and the attributes Ability and Experience are fuzzy attributes Type 2 and 3 respectively, with labels {Clumsy, Normal, Skilled} for the Ability, and labels {Apprentice, Normal, Expert} for Experience.

Now we add a fuzzy grade to the group of attributes X= {Experience, Ability}. We may call this degree "degree of competence or expertise," and its meaning can be varied according to the necessities of the model. For example:

1. **Uncertainty degree G^2**: This represents the extent to which the values of the attributes {Experience, Ability} are true. Depending on how the two values have been calculated (whether they have been based on much or little evidence) they can be more or less reliable. Figure 4.5 represents this case, where *d* expresses the formula for the automatic calculation of the fuzzy degrees.
2. **Importance degree G^4**: For an employee, those attributes may be more important than others. For example, a surgeon requires more experience and ability than an administrative worker.

*

Figure 4.6. Example 4.4: Fuzzy degree associated with the fuzzy attribute Type 3 Color, with meaning of Intensity degree, and one degree with its own meaning, known as Danger



Fuzzy Degree With Its Own Meaning

Another way of using the degrees is to consider the case in which a determined degree has no reason to be associated with any other attribute but instead can be fuzzy and have its own value (or meaning).

Definition 4.6: *An entity or a relationship E may have a fuzzy degree with its own meaning. This degree is graphically represented by a circle and a dashed line with a G inside it, joined to E. That circle will be labeled with the name of this degree. Here the meanings explained previously can also be applied, but in general, it is best to use a sufficiently descriptive name, because that degree has its own meaning, which should be expressed as clearly as possible with its name.*

Derived fuzzy degrees are represented like the Definition 4.4.

*

Example 4.4: The entity Medicine in Figure 4.6 has some attributes from which we can highlight Components, which is a multivalued (not fuzzy) attribute, Color, which is a fuzzy attribute Type 3, and two degrees:

- The Color attribute has a degree associated with each value that measures the intensity of that color.

- The attribute Danger measures how dangerous that substance is. This degree is not associated with any other attribute. It is a fuzzy degree with its own meaning.

*

Fuzzy Degree to the Model

This is a case of redefining the first level of Zvieli and Chen (1986), extending the advantage brought about by the power of being able to use degrees with different meanings.

Definition 4.7: *An entity, relationship, or attribute may have a fuzzy degree in respect to the model. This degree can have different meanings just as we expressed in Definition 4.3. The type of degrees are denoted by adding to the name of the object (entity, relationship, or attribute) the expression* $G^n = \alpha$, *where n is the number that indicates the meaning of that degree or the meaning in words and* α *is the associated fuzzy degree, with* $\alpha \in [0, 1]$.

*

With this tool we can define a complete generic model and for certain uses delimit the model to those entities, relationships, and attributes whose importance degree (for example) is greater than a value *x* and whose membership degree is greater than another value *y*. In this way, the entities, relationships, or attributes that do not reach those thresholds will be hidden (or eliminated). Thus, we may obtain a model that is as complete or as simplified as we want, paying attention to as many characteristics as we want.

Of course, if an entity is hidden from the model because it has a lower degree than the desired degree, then the attributes and relationships in which that entity is involved will also be hidden. And if a relationship is hidden, then its attributes will also be hidden. If an attribute is hidden, then the fuzzy degrees that are associated with it will also be hidden.

It is important to point out that by default, all the entities, relationships, and attributes have degree 1 (the maximum) for any meaning.

In the following section we present an example (Example 4.5) of the previous definition.

Fuzzy Aggregations

De Miguel, Piattini, and Marcos (1999) state that an aggregation allows us to represent types of compound entities, which are obtained through the union of other simpler ones. The compound type is referred to as the *all,* while the components are the *parts.* These authors define an aggregation as an entity, which is composed of one set of different elements. They define two kinds of aggregations:

1. **Aggregation of entities**: This aggregation expresses that each instance of an aggregated entity is composed of other instances of other entities, and it is denoted by a rhombus close to the aggregated entity (the *all*). The other entities (the *parts*) join the rhombus with a line.
2. **Aggregation of attributes**: This is the most common type of aggregation and expresses that an entity is a set of attributes.

Aggregation of entities may also be represented by creating relationships that link each part with the all, but we consider that this representation simplifies the understanding of the model. These concepts are also dealt with by other authors, such as Batini, Ceri, and Navathe (1994).

Definition 4.8: *A **fuzzy aggregation** is defined in two ways:*

1. *A fuzzy aggregation of types of entities allows us to represent that an all is obtained through the union of different parts that can be types of different objects, which belong to the aggregated entity (the all) with a certain degree and which carry out different roles in the aggregation. Following the notation of De Miguel, Piattini, and Marcos (1999), the graphic representation of the fuzzy aggregation of entities is a rhombus, which we show here with a dashed line, joined to the entity that forms the all, and the different entities, which*

form the parts that are joined with lines. Those parts with fuzzy degrees are joined with dashed lines, and beside these lines, the degree that associates it with the aggregation and its meaning is defined (using the format of Definition 4.3*).*

2. *A fuzzy aggregation of attributes allows us to represent an entity and a collection of attributes. These attributes represent the parts that compose it. For this notation we will use the proposal in* Definition 4.7.

*

Note that in Definition 4.7 an entity was represented as a being with diverse attributes that have a degree with regard to the model. On the other hand, in Definition 4.8 (2) the point of view changes to represent an entity as an aggregation of attributes. In other words, the difference is that although Definition 4.7 refers to the degree as the model, in Definition 4.8 the degree refers to the entity of the aggregation. However, for the sake of simplicity, we have not changed the notation; we consider that this small point, which causes the difference between the two concepts, is not important in practice. If it were, we could put a circle with a dashed line in Definition 4.8.

The classic aggregation of entities includes the (min, max) notation to express the cardinality of each aggregated entity. In the FuzzyEER Model we can use the fuzzy (min, max) notation (see the "Fuzzy Constrains" section, later in this chapter).

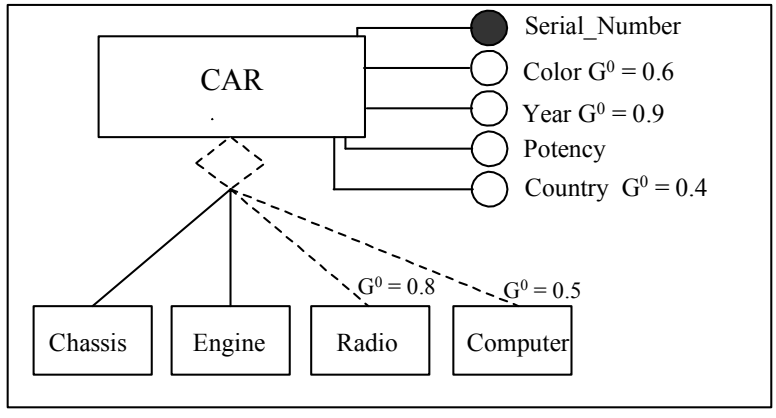
Consider that in Definition 4.8 classic and fuzzy aggregation is not exclusive, because normally only some of the parts will be fuzzy components. The same thing happens in the aggregation of attributes as, in general, only some attributes will be of fuzzy aggregation to the entity.

Example 4.5: Figure 4.7 models an entity Car with some attributes: serial number (the primary key), color, year, potency, and country (of production). On the other hand, a car is composed of a chassis, engine, radio, specialized computer, and other entities. Some of these elements (attributes and entities) have a membership degree to the model.

If we want a detailed model, we can use all elements, but if we do not need such a detailed model, then we can get an element with a membership degree greater than 0.7, for example. In this case, the model does not use some attributes (color and country) and some entities (computer).

*

Figure 4.7. Example 4.5: Fuzzy aggregations



It should be noted that the example in Figure 4.7 was also studied by Ma, Zhang, Ma, and Chen (2001). We consider that the notation proposed in the FuzzyEER Model is more representative than that of these authors as well as more complete. Also, the type of example of Figure 4.7 is dealt with in Chen and Kerre (1998), but this author does not use the concept of aggregation of entities.

Fuzzy Entities

There are two types of entities: the regular entities, which exist in the database on their own account, and the weak entities, whose existence depends on another entity (normally a regular one), known as owner entity.

On the other hand, in the section "Similarity Relations: The Buckles-Petry Model" in Chapter II we established that a fuzzy degree can be associated to each instance of an entity. This case expresses the concept of the fuzzy entity. In this section we study both types of fuzzy entities.

Fuzzy Entity as a Fuzzy Degree in the Whole Instance of an Entity

As we know, an entity is a real or abstract object about which we want to store information in the database. On the other hand, an instance (occurrence) of the

entity is each one of the concrete realizations (objects or values) of that type of entity. The graphic representation of the type of entities is a rectangle labeled with the name of the type of entity it represents (Elmasri & Navathe, 2000; De Miguel, Piattini, & Marcos, 1999; Silverschatz, Korth, & Sudarshan, 2002).

In the FuzzyEER Model, each instance in a fuzzy entity has a different degree for measuring the relationship between itself and its entity (its degree of pertinence to this type of entity, or its importance degree, its certainty degree).

Definition 4.9: *A (regular) fuzzy entity is defined in a FuzzyEER Model as an entity with an attribute that expresses a degree (with any meaning). In other words: If E is a fuzzy entity with n instances,* $e_1, e_2, \ldots, e_n$, *then at least one* m_E *function is defined about these instances so that*

$$\forall\ e_i \in E \text{ with } i = 1, 2, \ldots, n,\ \mu_E(e_i) \in [0, 1] \qquad (4.1)$$

The expression $\mu_E(e_i)$ *can measure the degree with which the instance* e_i *"belongs" to E, although it can have other meanings such as those expressed in Definition* 4.3.

The notation of a type of fuzzy entity is represented with a rectangle with dashed lines. The fuzzy attribute representing the degree must also be added with its meaning. This attribute is represented by a dashed circle with a dashed line (which distinguishes it from other degrees). The circle of that degree should be labeled with the symbol G^n*, which we explain in* Definition 4.3.

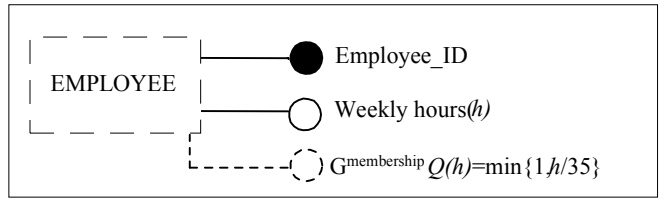
Optionally, the function that allows this degree to be calculated may also be added.

*

This is similar to the previous cases, but here the degree is associated with the whole instance of a certain type of entity and not exclusively to a particular value of an attribute.

Note that we can allow an entity to have several degrees associated with it. The semantic concept becomes more complicated, but it would not make sense to limit each entity to only one degree. Furthermore, constraints could be established between different degrees (for example, the importance of each

Figure 4.8. Example 4.6: Fuzzy entity with a membership degree for each instance, which depends on the number of hours (h) worked per week



employee in his department should be greater than his degree of capacity for a certain task).

Example 4.6: We may consider a fuzzy entity Employee with an attribute that stores the total number of hours worked per week. For each employee, a membership degree to the entity can be defined in such a way that the employees will belong to the Employee type of entity with a certain degree, according to the number of weekly hours. This degree will be calculated by dividing the total number of hours worked by the minimum number of hours, so that the belonging is total. Note that this is a derived fuzzy attribute in order to obtain the membership degree to the entity.

Figure 4.8 models this example, where $Q(h)$ is the calculus of the degree and h is the number of hours worked per week. We can see that $Q(h) = \min\{1, h/m\}$, where m is the minimum number of hours for the total membership.

If $m = 35$, then an employee who works in the company for 15 hours will be considered an employee with a degree of 0.43 (the result of the division 15/35) so that this degree can be maintained in diverse calculations (selections with different aims, gratifications, etc.).

*

Fuzzy Weak Entities

Weak entities depend on another entity known as owner entity. This dependency can be a relationship of dependency on identification (Elmasri & Navathe, 2000) or dependency on existence (De Miguel, Piattini, & Marcos, 1999). The graphic representation of a weak entity is a rectangle with double

line (two concentric rectangles) with its name inside connected with a double line to a type of relationship marked with an "E" (Existence) or an "ID" (Identification) in a corner.

Both types of weak entities are very similar, and with that of Identification, that of existence can be represented, although this second form helps us to better represent the model. Take a look at both types in detail:

- **Weak entity due to dependency on existence**: In this case the weak entity cannot exist if the type of owner entity disappears. Its existence makes no sense without the existence of the owner entity. In general, instances of this kind of weak entity do not exist in the real world if they are not associated to an instance of the owner entity. A weak entity due to dependency on existence is usually used to represent a multivalued attribute in a regular entity, especially if that attribute is composite or, in our case, is associated with a fuzzy degree.
- **Weak entity due to dependency on identification**: In this case, the weak entity lacks its own primary key and needs the key of the owner entity in order to identify each instance. Of course, the weak entity must have other attributes, especially those that are known as the *partial key*. The partial key distinguishes those instances that belong to the same instance in the owner entity. In this case, if the owner entity disappears, then the weak entity can continue to exist by simply looking for or creating a new primary key for that weak entity.

Based on these concepts, we can define the concept of the fuzzy weak entity by creating two new tools in order to amplify the expressiveness of the FuzzyEER Model.

Definition 4.10: *A **fuzzy weak entity** is defined in two cases, denoting the owner entity as E:*

1. *A fuzzy weak entity due to dependency on existence: This case allows us to express, in a different way, when an entity has a fuzzy multivalued attribute (or with a fuzzy component in composite attributes). However, the most useful thing about this case is that we can use it whenever we want to store the measurement in which the value of a*

certain fuzzy attribute B in each instance belongs to each one of the fuzzy sets defined by the labels of B. In order to do that, we identify the weak entity as E^F *with a rectangle with a dashed double line. Then* E^F *has the fuzzy attribute B (which is the partial key) and B has an associated fuzzy degree, which may optionally have a meaning (just like* Definition 4.3*).*

The instances of E^F *are defined by using the m instances of the owner entity E. In this way, for each instance* e_i *of E with* $i = 1, 2, ..., m$*, we deduce that* E^F *has n instances:*

$$z^F_{ij} = < k_i, b_j, \mu_{bj} (e_i [B]) > \text{ with } j = 1, 2, ..., n \qquad (4.2)$$

where k_i *is the key of the instance* $e_i \in E$*,* b_j *are the n labels with* $j = 1,..., n$ *defined for the attribute B,* μ_{bj} *is the membership function of the label* b_j *and* $e_i [B]$ *is the value of the attribute B in the instance* e_i*. In this case the key of the weak entity will be the union of* k_i *and the* b_j *values, which shows the label. Fuzzy attributes cannot be members of any key, but this is now possible because this attribute (B of* E^F*) only takes values in the labels (B of E).*

2. *A fuzzy weak entity due to dependency on identification: This happens when we have a fuzzy entity (*Definition 4.9*) that is weak. The graphic representation is denoted by a square with a dashed double line, and beside this an attribute with a circle with dashed lines, which indicates a type of degree associated with the weak entity.*

   This case supposes that the key of the weak entity will be the union of the key of the owner entity and an extra key, known as a partial key, that the weak entity must have. So the values of that partial key can be repeated in the weak entity if they belong to a different instance than E.

In both types of weak entities, the partial key will be represented in the same way as a primary key (solid black circle). Furthermore, the line joining the weak entity with the rhombus of the relationship must be a double line in order to express that all the instances of the weak entity must be related to an instance of the owner entity.

*

Some authors (Elmasri & Navathe, 2000) prefer to use a double line in the relationship diamond to the owner entity.

Example 4.7: There is an Employee entity with the attributes (Employee_ID, Employee_name, Contract_year, Evaluation). The Evaluation attribute is a qualification that each employee obtains for his or her work over a certain period of time. The attribute is defined as Type 2 and has the following labels associated with it: {Deficient, Mediocre, Good, Excellent}. From this point of view, Evaluation is an attribute that presents imprecision, and in some instances it could be treated like Type 1 with the same labels and may be mapped in a fuzzy domain.

For this case, an employee may have a membership degree to more than one label of the Evaluation attribute so that it obtains the most representative membership degree, which adjusts to the work of the employee. For example, if an employee is evaluated as Good, this value may be more or less similar to Excellent or Mediocre. Those similarity degrees can be important for some tasks.

In Figure 4.9a, this model with the weak entity Employee^F is expressed with the maximum cardinality M. Observe that the Evaluation attribute of Employee stores the evaluation that each employee obtains, while the Evaluation attribute of Employee^F stores each Evaluation label (of Employee attribute) with a membership degree. Note that this attribute is Type 2 and will form part of the key of Employee^F (denoted by a circle of black stars).

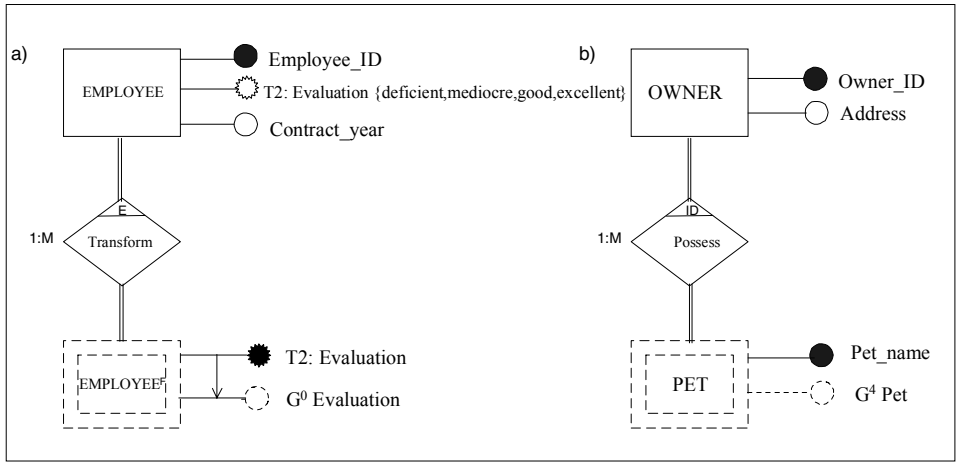
*

Example 4.8: Figure 4.9b shows an example of a fuzzy weak entity, which is due to dependency on identification. Imagine a pet hotel, in which the owner of each animal may express an importance degree for it. This importance degree, G^4, is identified in the fuzzy weak entity by means of an attribute that considers "the importance degree of each animal." The key of the Pet entity will be (Owner_ID, Pet_name).

*

This type of fuzzy weak entity due to dependency on existence covers and improves the definition of Chaudhry, Moyne, and Rundensteiner, (1994, 1999). Its main advantage is that it speeds up some types of fuzzy queries,

Figure 4.9. Weak fuzzy entities in FuzzyEER: a) with relationship of dependency on existence (Example 4.7) and b) with relationship of dependency on identification (Example 4.8)



because the E^F entity stores the similarity degrees already calculated. For example, to see the employees that are "excellent" (with a minimum degree of 0.8), you must look in Employee^F for instances with Evaluation = Excellent and $G^0 \geq 0.8$. Another option is that when necessary, those degrees can be calculated instead of being stored.

On the other hand, the fuzzy weak entities due to dependency on identification are very interesting and useful. Let us think that if we consider the fuzzy entities to be useful, then it may occur that a fuzzy entity is also weak, and for this reason a formal definition of this concept is required.

Fuzzy Relationships

Some authors, such as Connolly, Begg, and Strachan (1998) and Elmasri and Navathe (2000), define *relationship* as the generic structure of the set of existing connections between two or more types of entities. A relationship can associate an entity with itself, which is called a *recursive relationship*. On the other hand, the instance (or occurrence) of a relationship will be the existing link between the concrete occurrences of each type of entity that is involved in the

relationship. A relationship is represented by using a rhombus labeled with its name and joined to the associated types of entities with lines.

These authors also define the concept of relationship attributes. If a relationship 1:N has attributes, then they can be moved to the entity with the maximum cardinality (N).

Then, using the general concept of fuzzy relation (see Chapter I), we can define fuzzy relationships in the FuzzyEER Model.

Definition 4.11: *Let R be a relationship linking one or more entities* E_1, E_2, ..., E_t. *We will call it* ***fuzzy relationship*** *if it has at least a fuzzy degree with a* G^n *meaning (see* Definition 4.3*), which links the entities associated with R. A fuzzy relationship links or relates t entities (with* $t \geq 2$*), associating k degrees (with* $k \geq 1$*) to this union, for each group of related instances. A fuzzy relationship is represented by a function:*

$$R: E_1 \times E_2 \times ... \times E_t \rightarrow [0, 1]^k$$
$$R(e_1, e_2, ..., e_t) \rightarrow [0, 1]^k \qquad (4.3)$$

where $e_i \in E_i$ *with* $i = 1, 2, ..., t$, *are the instances of the entities in the relationship, and* $k \geq 1$ *is the number of fuzzy degrees associated with R. If R has other attributes, then each attribute may be represented by a similar function, changing the domain in the right part.*

The graphic representation of a fuzzy relationship is a rhombus with a dashed line with a degree attribute, similar to that in Definition 4.9*, with the circle with a dashed line.*

Similarly to Definition 4.9, the function that allows this degree to be calculated can be added (if it exists).

*

Of course, a relationship (whether or not it is fuzzy) can include fuzzy attributes (T1, T2, T3, or T4) or fuzzy degrees (associated or not associated), just as we showed in the section on fuzzy values, earlier in this chapter. Furthermore, in a fuzzy relationship we can use the (min,max) notation for the constraints of participation and cardinality. We can also use the fuzzy (min,max) notation (see the section on fuzzy constraints later in this chapter).

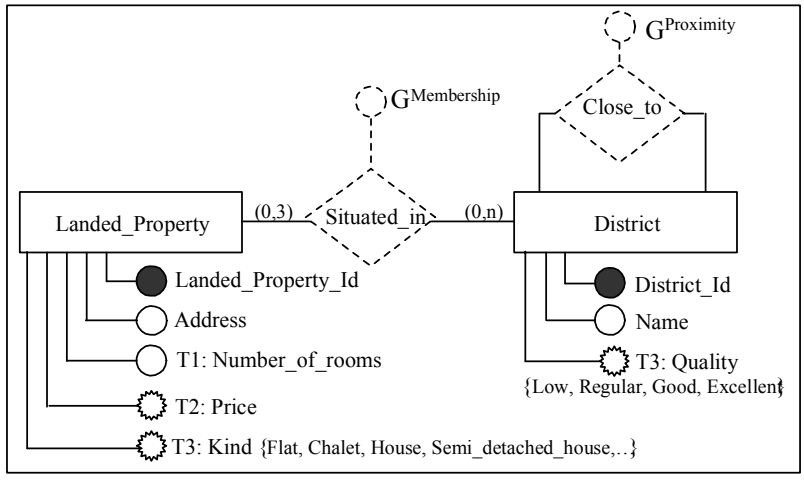
Example 4.9: For a real estate agency, the entity District can have the attributes (District_Id, Name, Quality). The attributes District_Id and Name are crisp. The attribute Quality of the district is defined as a fuzzy attribute Type 3 with the following labels: {Low, Regular, Good, Excellent}.

The relationship of proximity of the neighborhoods can be represented as the fuzzy relationship Close_to, which appears in Figure 4.10. This expresses that a proximity degree exists between any two districts.

Furthermore, the entity Landed_Property is modeled with some attributes, which you can also see in Figure 4.10. Each landed property can be situated in such a place that it belongs to several districts or to one district but is relatively close to another district. For example, for a property, it can be indicated that its neighborhood has the following possibility distribution (0.5/North, 1/East, 0.2/Plaza_España), indicating that it is situated in the eastern district and closer to the northern district than to the España square district.

If District were an attribute of Landed_ Property, it would be sufficient to define it as Type 3, to define each district as a label, and to establish a similarity relationship (or proximity in this case) for every two districts. But this case is special, because District is an entity with some attributes and is related to the Landed_Property entity, so that a property may be related to several districts (3 at most). At the same time, a district for a certain landed property may have a membership degree that measures to what extent that property belongs to that district. In our model it is represented by the degree $G^{Membership}$.

Figure 4.10. Example 4.9: Fuzzy relationships



The fuzzy relationship Close_to in fact generates a similar structure to a Type 3 fuzzy attribute. On the other hand, the fuzzy relationship Situated_in generates a similar structure in the Landed_Property entity, as if that entity had a Type 3 fuzzy attribute called District. The model reflects that the entities Landed_Property and District are related in such a way that each landed property may be situated in a maximum of three districts, and each one of those associations gives the degree at which that landed property belongs to the district.

Due to the fact that District has several attributes, it cannot be used as a fuzzy attribute Type 3 of Landed_Property.

*

A more detailed example of this case is found in Urrutia and Galindo (2002) and Urrutia (2003).

Fuzzy Degrees in Specializations

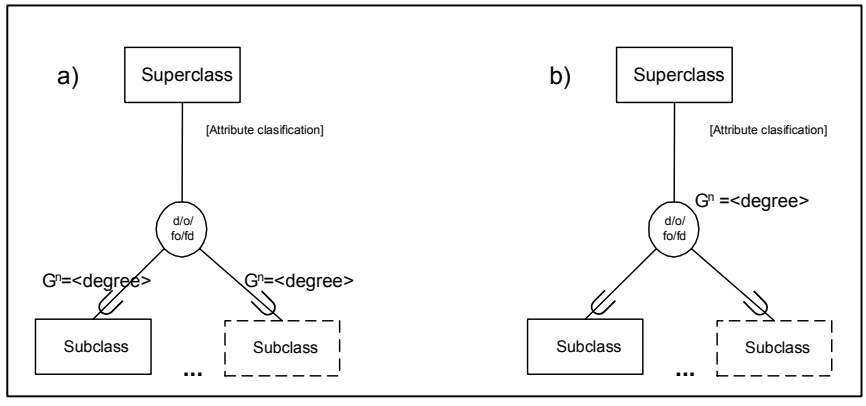
In the same way as the fuzzy degrees were incorporated in the aggregation in the "Fuzzy Aggregations" section, fuzzy degrees can be used in specializations.

Definition 4.12: *We can assign a degree to a specialization in two ways, and the meaning of this degree may be expressed in the model:*

1. ***Degree in the subclasses****: This degree expresses a fuzzy degree of some subclass in the specialization. It is denoted by* $G^n = \alpha$*, labeling the line joining the subclass with the circle referred to as specialization circle, where n is the meaning of this degree (just like Definition* 4.3*), and* α *is the associated fuzzy degree, with* $\alpha \in (0,1)$*. See* Figure 4.11a.
2. ***Degree in the specialization****: This degree expresses a fuzzy degree of all the specialization (or all its subclasses). It is denoted by* $G^n = \alpha$ *labeling the specialization circle. See* Figure 4.11b.

*

Figure 4.11. a) Degree in some subclasses of the specialization and b) degree in the specialization



Note that in Figure 4.11, inside the specialization circle, we can use "d" (for disjoint specializations), "o" (for overlapping specializations), "fd" (for fuzzy disjoint specializations), and "fo" (for fuzzy overlapping specializations). Disjoint and overlapping specializations are explained, for example, in Elmasri and Navathe (2000), whereas fuzzy disjoint and fuzzy overlapping specializations are explained in Galindo et al. (2003) and Galindo, Urrutia, and Piattini (2004a). In short, fuzzy specializations occur when a subclass is a fuzzy entity.

This type of concept (adding a degree) can be used for all the hierarchies of specialization, generalization, and aggregation in the same way as they were dealt with here. In Marín et al., (2000) there are similar examples to the notation of Figure 4.11. Other applications of the theory fuzzy sets in fuzzy specializations are found in Galindo, Urrutia, and Carrasco (2002), Galindo, Urrutia, and Piattini (2004a), and Galindo, Urrutia, Carrasco, and Piattini (2004b).

Fuzzy Constraints

In this section our aim is to relax the constraints, which can be expressed in a conceptual model by using the Enhanced Entity Relationship modeling tool, so that these constraints can be made more flexible. We also study new constraints that are not considered in classic EER models. We use the fuzzy quantifiers

(refer to Chapter I), which have been widely studied in the context of fuzzy sets and fuzzy query systems for databases. In addition, we examine the representation of these constraints in an EER model and their practical repercussions.

In Chapter III you see that only Chen (1998) fuzzifies only the participation and cardinality constraints in an ER relationship. The other approaches about fuzzy modeling tools do not include fuzzy constraints or some technique to relax the constraints expressed in the ER/EER model so that they can be made more flexible, because the constraints of the traditional model are either too restrictive or too permissive.

We can cite, in another line, the work by Davis and Bonnell (1989). They present a set of constructs for capturing certain types of semantic integrity constraints, based on the specific types of logic propositions that exist on a collection of relationships between a given entity set and the entity to which it is associated through these relationships. For example, let E be an entity with two relationships (P and Q). Using classical logic, we can then apply the following constraint based on the implication function: If an instance $e \in E$ uses relation P, then e uses relation Q. It can be observed that these constraints need only classical logic and that some of their cases are also solved by the EER Model.

We study the following constraints in this section: the fuzzy participation constraint, the fuzzy cardinality constraint, the fuzzy completeness constraint to represent classes and subclasses, the fuzzy cardinality constraint on overlapping specializations, fuzzy disjoint and fuzzy overlapping constraints on specializations, fuzzy attribute-defined specializations, fuzzy constraints in union types or categories, and fuzzy constraints in shared subclasses. We also demonstrate how fuzzy (min,max) notation can substitute the fuzzy cardinality constraint but not the fuzzy participation constraint. All these fuzzy constraints have a new meaning and offer greater expressiveness in conceptual design.

Constraints in the ER/EER Model

Using the ER/EER Model, constraints play a fundamental role: They express how the entities are related. Basically, we have the following types of constraints in a schema using the ER/EER Model:

1. **The participation constraint**: The participation of an entity in a relationship can be total or partial. If each instance must compulsorily relate to the

other instance or instances of the relationship, then participation is said to be *total*. If this relationship is not mandatory for all instances belonging to this type, then participation is said to be *partial*. In ER diagrams, total participation is displayed as a double line connecting the participating entity type to the relationship, whereas partial participation is represented by a single line. Figure 4.15 shows both constraint types. We discuss fuzzy participation constraints in a later section.

2. **The cardinality constraint**: This constraint expresses whether the relationship between entities is "one to one" (1:1), "one to many" (1:N), or "many to many" (N:M). We explain how to relax this constraint in the "Fuzzy Cardinality Constraint on Relationships" section.
3. **The completeness constraint on specializations**: This constraint may be *total* or *partial*. A total specialization constraint specifies that every instance in the superclass entity must be a member of one (or some) of the subclasses entity in the specialization. This is shown in EER diagrams by using a double line to connect the superclass to the circle referred to as the *specialization circle*, to which all the subclasses are joined by a single line with the inclusion symbol. A single line is used to display a partial specialization, which allows an instance to not belong to any of the subclasses. The inverse is not possible, because by definition each member of a subclass must be a member of the superclass. The "Fuzzy completeness constraint on specializations" section includes an explanation of this constraint in a fuzzy model.
4. **Disjoint or overlapping constraints on specializations**: A disjoint specialization occurs when subclasses are disjoint, that is, every member of the superclass must belong to a maximum of one of the subclasses. Disjoint specializations are shown in EER diagrams by using a circle with the letter "d." An overlapping specialization permits the subclasses to contain common elements, that is, each member of the superclass may belong to various subclasses. Overlapping specializations are shown in EER diagrams by using a circle with the letter "o." Later in this chapter, we study fuzzy disjoint and overlapping constraints on specializations as well as the cardinality constraint on overlapping specializations, a constraint that is not studied in classic EER models.
5. **Completeness constraint in union types**: A category (Elmasri & Navathe, 2000; Elmasri, Weeldreyer, & Hevner, 1985) can be total or partial. A category is *total* if every superclass instance must be a member of the category. This is shown in EER diagrams by using a double line to

connect the category with a circle with the union symbol (∪). This is a strange case, because this union type can be represented by using a total disjoint specialization (the *superclass* is the category, and the *subclasses* are all superclasses of the union type). A category is *partial* if every superclass instance may or may not be a member of the category. This is shown in EER diagrams by using a single line to connect the category to the circle with the union symbol. The classic model does not study the participation constraint of each superclass in the category. The section on fuzzy constraints in union types discusses these two constraints in a fuzzy model.

In addition, the **(min,max) notation** allows for the expression of the participation and cardinality of an entity in a relationship. In this notation, min and max indicate the minimum and maximum number of entity instances that take part in the relationship. The (min,max) notation is better, as it allows for the use of numbers other than 1 and N. It can clearly be seen that the (min,max) notation includes participation and cardinality in classic ER models. Later in this chapter we study fuzzy (min,max) notation on relationships.

If a relationship of the ER Model has a degree greater than 2, then the constraints are also applicable to each entity participating in such a relationship. In this case, each entity treats the rest of the entities, which participate in the relationship as if they were a single entity.

Thresholds and Fuzzy Quantifiers for Relaxing Constraints

Definition 1.22 in Chapter I defines the fuzzy quantifiers. Applied in the context of databases, the usefulness of fuzzy quantifiers is shown by the flexibility they offer when carrying out queries that involve these quantifiers, as in the division operation of relational algebra in fuzzy or classical databases, for example (Galindo, Medina, Cubero, & García, 2001). Applied in the context of conceptual data models, fuzzy quantifiers allow expressions about the number of instances satisfying a given condition, or the proportion with respect to the total. We shall study this in subsequent sections. Of course, the quantifier Q must be previously defined in the model's data dictionary (metadata).

In this context, we need a threshold $\gamma \in [0, 1]$ indicating the minimum fulfillment degree that must be satisfied. This threshold will be written in square brackets:

Q[γ]. For example, we may use "almost_all [0.2]" to indicate that this fuzzy quantifier must be satisfied at a minimum degree of 0.2. Consequently, the underlining constraint requires that

$$Q(\phi) \geq \gamma \tag{4.4}$$

Every time the database is modified, the DBMS computes ϕ and checks whether Equation 4.4 is satisfied. We define the meaning of ϕ in subsequent sections, because it depends on where the fuzzy quantifier is used. In order to simplify the expression, we can set a default value for γ at 0.5, for example. We consider 0.5 to be a good default value because it is in the middle of the interval [0, 1], but it may be changed.

If Q is an increasing function, then we can simplify Equation 4.4 because

$$\phi \geq Q^{-1}(\gamma) \tag{4.5}$$

Similarly, if Q is a decreasing function, then

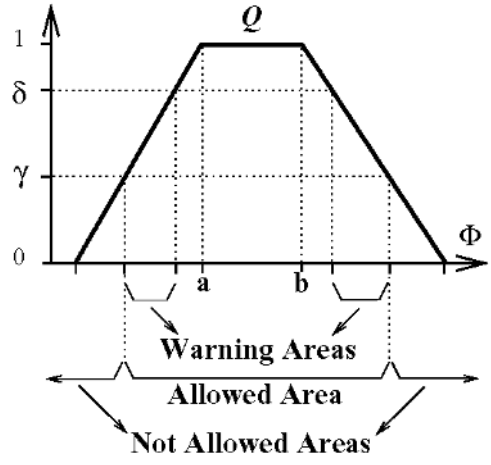
$$\phi \leq Q^{-1}(\gamma) \tag{4.6}$$

The last two equations may be useful because Q and γ are constants, whereas ϕ is a varying value. Value ϕ may change with every DML sentence `INSERT`, `DELETE`, or `UPDATE`. In this way, we can store $Q^{-1}(\gamma)$, avoiding the use of Q with those DML sentences.

In addition, another optional value δ can be established, which is greater than the threshold γ in the following way: Q[γ, δ] such that $\gamma < \delta$. The value δ is more restrictive than γ and establishes that when the constraint is unfulfilled with this higher value, the DBMS will inform the user but will permit the modification of the database that is underway. If the quantifier is unfulfilled with a value between γ and δ, then the DBMS must warn the user (or only the database administrator). Both values would be close in order to avoid too many warnings from the DBMS. Therefore, the warning message is generated when Equation 4.4 is satisfied and the following equation is *not* satisfied:

$$Q(\phi) \geq \delta \tag{4.7}$$

Figure 4.12. Thresholds γ and δ in a fuzzy quantifier "approximately between a and b" and its generated areas



In other words, the warning area is defined with

$$\delta > Q(\phi) \geq \gamma \qquad (4.8)$$

Finally, if Equation 4.4 is false, then it defines the area that is not allowed, and an error message must be generated.

Example 4.10: Figure 4.12 depicts a fuzzy quantifier with the thresholds γ and δ. We want to evaluate to what extent value ϕ satisfies the quantifier. This evaluation is carried out by $Q(\phi)$. It should be noted that these thresholds divide the domain of ϕ into three areas: the allowed area, the not allowed area, and the warning area. The warning area is included in the allowed area. Note that the not allowed area is defined when Equation 4.4 is false.

*

A fuzzy quantifier can be written in three ways:

1. Quantifier without a threshold g: Default threshold is γ= 0.5. For example, approx_2.

2. Quantifier with a threshold γ: For example, approx_8[0.25].
3. Quantifier with two thresholds γ and δ, with γ < δ: For example, approx_3[0.25,0.75].

Fuzzy Participation Constraint on Relationships

In addition to being either total or partial, in the fuzzy model that we propose here, the participation of an entity in a relationship can be fuzzy by using a relative fuzzy quantifier (principally).

Definition 4.13: *Let E_1 and E_2 be two entities and R a relationship between them. A* ***Fuzzy Participation Constraint*** *of E_1 in R is represented by using a zigzag line (or broken line) joining E_1 and R, indicating on this line which quantifier Q has been used, followed optionally by one or two thresholds,* [γ] *or* [γ, δ]*, with the meaning and default value having the one explained earlier. We propose another representation using a single line crossed with an arc labeled with Q.*

This constraint asserts Equation 4.4, with ϕ defined by Equation 1.64, which we reproduce here:

$$\Phi = \begin{cases} a & \text{if } Q \text{ is absolute} \\ a/b & \text{if } Q \text{ is relative} \end{cases} \tag{4.9}$$

where a is now the number of E_1 instances related to E_2, and b is the total number of instances in E_1.

If Q is used with two thresholds, then it defines a warning area (see the preceding section). A warning message must be generated when the condition is satisfied with γ and is not satisfied with δ. It should be noted that the warning area is included in the allowed area. The warning area is defined when Equation 4.4 is satisfied and Equation 4.7 is not *satisfied. In other words, the warning area is defined with Equation 4.8. Finally, if Equation 4.4 is false, then it defines the not allowed area.*

*

This fuzzy constraint implies that every DML sentence may generate an error or warning when the fuzzy quantifier is not satisfied. This message forces the user to maintain a "good" database or warns the user when the database is not "good enough."

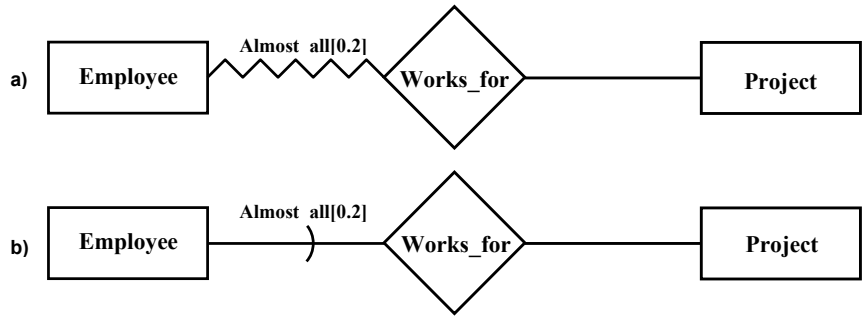
Example 4.11: Suppose we have an Employee entity and a Project entity linked by the relationship Works_for. The participation of Employee in this relationship can be represented by a relative fuzzy quantifier such as "almost all" (refer to Figure 1.26), indicating that "almost all employees work for some project." Figure 4.13 represents these two entities, the relationship between them, and the fuzzy constraint, using the two graphic representations proposed in Definition 4.13.

The threshold $\gamma = 0.2$ in "almost_all [0.2]" indicates the minimum degree with which this quantifier must be fulfilled in the database. If we divide the number of employees who work for a project (value *a*) by the total number of employees in the database (value *b*), the result ϕ should be, in accordance with Equation 4.5, a value greater than or equal to $Q^{-1}(0.2) = 0.5$, because this is the value (on the X axis) for which the quantifier "almost all" attains a degree 0.2: $Q(0.5) = 0.2$. From value 0.5 of ϕ, this quantifier obtains a degree greater than or equal to 0.2, which was the constraint imposed by the threshold γ in the initial quantifier. The constraint then establishes that $\phi \geq 0.5$ must be satisfied.

In this example, the value 0.5 obtained by the expression $Q^{-1}(0.2)$ indicates the constraint that, in our database, a minimum of 50% of the employees must work for some project.

*

Figure 4.13. Example 4.11: Fuzzy participation constraint in an ER model, using the fuzzy quantifier almost_all (with the two representations proposed in Definition 4.13)



In general, if Q is a relative fuzzy quantifier with an increasing function, then Equation 4.5 states that the constraint must be satisfied in the database in a minimum *percentage* of $100Q^{-1}(\gamma)$. In this case, it is also possible to express this percentage instead with the quantifier Q and the threshold γ. Although this may appear easier, it must be noted that the intuitive and natural expressiveness of the quantifier is lost, that this is not valid with absolute fuzzy quantifiers, and the method must be adapted for decreasing fuzzy quantifiers. Fuzzy quantifiers are easy and general as well as very expressive and intuitive.

On the practical level this will be implemented as a *trigger* that in each operation of the type `UPDATE`, `DELETE`, or `INSERT` checks the value ϕ, and if Equation 4.4 is false, then the DBMS must produce an error message indicating the unfulfillment of this fuzzy constraint, and the operation is aborted. On the other hand, if Equation 4.8 is true, then a warning message must be generated, but the operation is allowed. Finally, in any other case the operation is normally allowed.

It should be noted that fuzzy quantifiers expressed in this type of constraint can also be absolute; however, due to the significance of a participation constraint, this will generally be relative because the number of entity instances would vary too much. In the case of an absolute fuzzy quantifier, such as "many" or "approximately between 100 and 200," this quantifier will restrict the quantity of entity instances related to the other entity. In our example, the fuzzy quantifier would restrict the number of employees who are assigned to work on any project.

In some models it might even be useful to establish several fuzzy quantifiers as a constraint on fuzzy participation. In this case, all fuzzy quantifiers in the same constraint must be coherent, as two quantifiers can be contradictory.

A fuzzy participation constraint is not as restrictive as a total participation constraint, nor as permissive as the partial participation constraint, so that the fuzzy participation constraints extend the ER Model, allowing a new expressiveness that would have been impossible in the traditional model.

Fuzzy Cardinality Constraint on Relationships

Fuzzy participation constraints establish a condition globally on the entity instances. On the other hand, fuzzy cardinality constraints establish fuzzy conditions on each instance in a particular and individual way. In classical modeling, the cardinality constraint states whether the relationship between

entities is "one to one" (1:1), "one to many" (1:N), or "many to many" (N:M). The model we propose allows us to express cardinality as a fuzzy value by using an absolute fuzzy quantifier (principally).

Definition 4.14: *Let E_1 and E_2 be two entities and R be a relationship between them. We suppose that e_i with $i = 1, 2, ..., b_1$ are the instances of E_1, and w_j with $j = 1, 2, ..., b_2$ are the instances of E_2. A* ***Fuzzy cardinality constraint*** *is defined with two quantifiers, separated by the notation "**:**", that is, Q_1**:**Q_2, just below the diamond, which represents the relationship between both entities. The quantifier on the left of the separator "**:**" will correspond to the entity on the left (or above), and the quantifier on the right will correspond to the other entity (E_1 and E_2, respectively). This constraint establishes two conditions*:

1. *Condition of Q_1*:

$$Q_1(\phi_{1i}) \geq \gamma_1 \ \forall\ i = 1, 2, \ldots, b_1 \tag{4.10}$$

where γ_1 is the threshold for Q_1, b_1 is the total number of instances in E_1 and ϕ_{1i} with $i = 1, 2, ..., b_1$ is defined by

$$\Phi_{1i} = \begin{cases} a_i & \text{if } Q_1 \text{ is absolute} \\ a_i / b_2 & \text{if } Q_1 \text{ is relative} \end{cases} \tag{4.11}$$

with a_i being the number of E_2 instances related with the instance $e_i \in E_1$, and b_2 is the total number of instances in E_2.

2. *Condition of Q_2*:

$$Q_2(\phi_{2j}) \geq \gamma_2 \ \forall\ j = 1, 2, \ldots, b_2 \tag{4.12}$$

where γ_2 is the threshold for Q_2 and ϕ_{2j} with $j = 1, 2, ..., b_2$ is defined by

$$\Phi_{2j} = \begin{cases} a_j & \text{if } Q_2 \text{ is absolute} \\ a_j / b_1 & \text{if } Q_2 \text{ is relative} \end{cases} \quad (4.13)$$

with a_j being the number of E_1 instances related to the instance $w_j \in E_2$.

The warning area is similarly defined by using δ_1 and δ_2, respectively.

*

This fuzzy constraint has an effect on each instance and must be satisfied by each one. Generalizing, quantifier Q_1 must be satisfied for all quantification values ϕ_{1i} with $i = 1, 2, \ldots, b_1$, that is, ϕ_{1i} must be in the allowed area of Q_1 with its threshold γ_1. On the other hand, quantifier Q_2 must be satisfied for all quantification values ϕ_{2j} with $j = 1, 2, \ldots, b_2$, that is, ϕ_{2j} must be in the allowed area of Q_2 with its threshold γ_2.

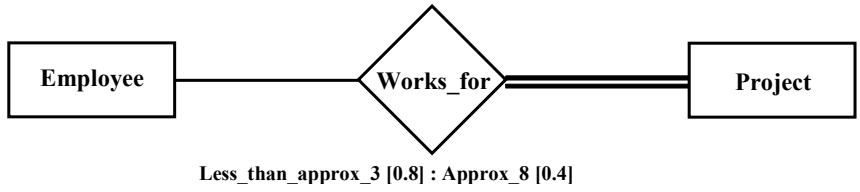
Example 4.12: Following the previous example, if we suppose that the entity Employee is on the left of the relationship Works_for, and the entity Project is on the right, a fuzzy cardinality constraint is shown in Figure 4.14.

These constraints express the condition that *each* employee will work for a maximum of approximately three projects, and each project will have approximately eight employees, requiring both constraints to be satisfied with the minimum fulfillment degrees indicated in square brackets.

*

It should be noted that the fuzzy quantifier of this type of constraint can also be relative; however, due to the meaning of a cardinality constraint, this quantifier will generally be absolute. In the case of a relative fuzzy quantifier, this quantifier

Figure 4.14. Example 4.12: Fuzzy cardinality constraint in a FuzzyEER model



will indicate the number of instances of the other entity to which each entity is related, with respect to the total number of instances of the other entity. Thus, if in Example 4.12 we use the quantifier "almost all" on the left, then we are saying that "each employee must work for almost all the existing projects."

In some models it might even be useful to establish various fuzzy quantifiers on one or in both sides of a fuzzy cardinality constraint. Of course, in this case all the fuzzy quantifiers in the same constraint must be coherent.

If a relationship joins three or more entities (a relationship with a degree greater than 2), then we can put the fuzzy cardinality quantifier associated with each entity at the side of the arc, which joins this entity with the relationship. If there is already a quantifier for the fuzzy participation constraint, then in order to avoid ambiguity, we must put the text "Card:" in front of the cardinality quantifier.

Fuzzy (min, max) Notation on Relationships

Nonfuzzy participation and cardinality of an entity in a relationship can be expressed with this notation, which is more expressive both in classical and fuzzy modeling. In fuzzy modeling both min and max can have values, which are fuzzy quantifiers in a similar way to the one explained above.

Definition 4.15: *Let E_1 and E_2 be two entities and R be a relationship between them. We denote the instances of E_1 as e_i* with *i* = 1, 2, ..., *b_1*, *and the instances of E_2 as w_j with j* = 1, 2, ..., *b_2*. *A **Fuzzy*** (min,max) ***Notation** of E_1 on R is represented by using two fuzzy quantifiers in parentheses* ($Q_{\min}, Q_{\max}$) *beside the line joining E_1 and R. This constraint establishes that*

$$\lambda_{\min} \le \phi_{\min,\,i} \wedge \lambda_{\max} \ge \phi_{\max,\,i} \;\forall\; i = 1, 2, \ldots, b_1 \tag{4.14}$$

where b_1 is the total number of instances in E_1 and

$$\Phi_{\min,i} = \begin{cases} a_i & \text{if } Q_{\min} \text{ is absolute} \\ a_i / b_2 & \text{if } Q_{\min} \text{ is relative} \end{cases} \tag{4.15}$$

$$\Phi_{\max,i} = \begin{cases} a_i & \text{if } Q_{\max} \text{ is absolute} \\ a_i / b_2 & \text{if } Q_{\max} \text{ is relative} \end{cases} \tag{4.16}$$

with a_i being the number of E_2 instances related with the instance $e_i \in E_1$, and b_2 is the total number of instances in E_2. Furthermore,

$$\lambda_{min} = \min\{\alpha : \alpha = Q_{min}^{-1}(\gamma_{min})\} \quad (4.17)$$

$$\lambda_{max} = \max\{\beta : \beta = Q_{max}^{-1}(\gamma_{max})\} \quad (4.18)$$

where γ_{min} and γ_{max} are the minimum thresholds for Q_{min} and Q_{max}, respectively. The allowed area is the interval $[\lambda_{min}, \lambda_{max}]$. The warning area is defined when the thresholds δ_{min} and δ_{max} are used for Q_{min} and Q_{max}, respectively. The warning message must be shown when Equation 4.14 is satisfied and the following equation is not *satisfied*:

$$\lambda'_{min} \leq \phi_{min,i} \wedge \lambda'_{max} \geq \phi_{max,i} \; \forall \; i = 1, 2, \ldots, b_1 \quad (4.19)$$

where

$$\lambda'_{min} = \min\{\alpha : \alpha = Q_{min}^{-1}(\delta_{min})\} \quad (4.20)$$

$$\lambda'_{max} = \max\{\beta : \beta = Q_{max}^{-1}(\delta_{max})\} \quad (4.21)$$

Hence, the warning area is the union of two intervals: $[\lambda_{min}, \lambda'_{min}] \cup [\lambda'_{max}, \lambda_{max}]$. We know that $\lambda_{min} < \lambda'_{min}$ and $\lambda'_{max} < \lambda_{max}$, because $\gamma_{min} < \delta_{min}$ and $\gamma_{max} < \delta_{max}$ respectively, and the quantifiers are defined with convex functions. Note that Equations 4.20 and 4.21 are similar to Equations 4.17 and 4.18, replacing γ_{min} and γ_{max} with δ_{min} and δ_{max}, respectively.

In other words, Equation 4.19 may be changed to obtain an equation that must be satisfied. Applying De Morgan's law, the warning area is defined when the following equation is satisfied:

$$\lambda'_{min} > \phi_{min,i} \vee \lambda'_{max} < \phi_{max,i} \; \forall \; i = 1, 2, \ldots, n \quad (4.22)$$

*

It should be noted that if a constraint exists on E_1 using (min,max) notation, then this constraint has an effect on each instance and must be satisfied by each one.

These two quantifiers indicate, respectively, the *minimum* and *maximum* number of E_2 instances related with each E_1 instance.

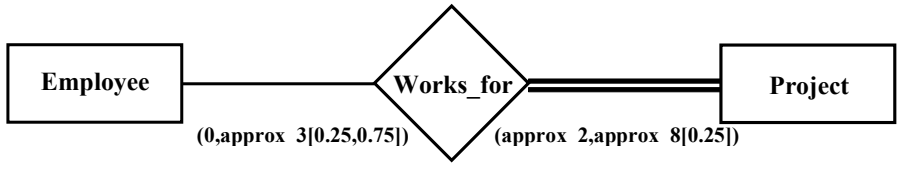
Observations:

- If $Q_{min}(0) \geq \gamma_{min}$, then we cannot use $Q_{min}[\gamma_{min}]$.
  In this case we must use 0 instead of Q_{min}: $[0, Q_{max}]$.
- If $Q_{max}(\psi) \geq \gamma_{max}$, where ψ is the maximum value in the underlying domain of Q_{max} (if Q_{max} is relative, $\psi = 1$), then we cannot use $Q_{max}[\gamma_{max}]$.
  In this case we must use the letter "N" instead of Q_{max}, expressing a cardinality constraint "to many": $[Q_{min}, N]$.
- If Q_{min} and Q_{max} are of the same type (absolute or relative), then $\phi_{min,i} = \phi_{max,i}$.
- In conclusion, we must use fuzzy quantifiers, which express a good constraint. An example of a bad constraint is [approx_3_or_less, almost_all].

Example 4.13: In the context of the previous examples, we can use the following constraints with the fuzzy (min,max) notation. These constraints are represented in Figure 4.15. On the employee side, the (min,max) constraint indicates that an employee may work for no projects (0 as *minimum*) and up to approximately three projects as a *maximum*. The two values after the quantifier indicate that if this is fulfilled to a degree greater than or equal to 0.75 the operation will normally be permitted; if it is fulfilled to a degree between 0.25 and 0.75, then the user will be informed but the operation will be allowed; and if the constraint is fulfilled to a degree of less than 0.25, this means that the constraint is not being fulfilled because it reaches an intolerable degree, and therefore the operation underway must not be permitted.

On the Project side in Figure 4.15, we can find a constraint indicating that each project must have a *minimum* of approximately two employees working for it (with a degree of 0.5 as a minimum). If the quantifier approx_2 is defined as a triangular function (refer to Figure 1.2) with a center in 2 ($m = 2$), and a *margin* equal to 2 ($a = 0$ and $b = 4$), then the value 0.5 is achieved with the minimum value 1, so that this quantifier with the minimum degree of 0.5 guarantees the total participation of the entity Project in the relationship Works_for. This possibility is also indicated by the double line, which connects the entity to the

Figure 4.15. Example 4.13: Fuzzy (min,max) notation in a FuzzyEER model



relationship. It should be noted that we are using two quantifiers for the min value, but both are coherent.

At the same time, the number of employees in each project is restricted to a *maximum* of approximately eight (with a minimum degree of 0.25).

*

In the classical ER Model, the (min,max) notation substitutes the other two notations for the participation and cardinality constraints, because if min = 0, then we are dealing with a partial participation, and if min > 0, we are dealing with a total participation. On the other hand, if max = 1, the relationship will be 1:1 or 1:N (on the side of 1), and if max > 1 (or max = N), we are dealing with a relationship N:M or 1:N (on the side of N).

However, in the ER Model with fuzzy constraints, the fuzzy (min,max) notation adds expressiveness to the conceptual model, but it can only substitute fuzzy cardinality constraints.

Fuzzy (min, max) Notation and Fuzzy Cardinality Constraints

With regard to cardinality constraints, the semantic of both notions is clearly different. It should be noted that in Example 4.12, the quantifier "approx_8" indicates that a Project must have approximately eight employees, but in Example 4.13, the same quantifier indicates that a Project must have a *maximum* of approximately eight employees.

In general, a fuzzy cardinality constraint with any type of quantifier can be represented by using the fuzzy (min, max) notation so that both values, min and max, have the value of this fuzzy quantifier. The fuzzy cardinality constraint expressed in Example 4.12 can be expressed in fuzzy (min,max) notation so that the minimum value is equal to the maximum and both have the value of "approx_8."

Expressiveness is also equivalent on the other side, with one exception depending on the types of both quantifiers:

- If the (min,max) notation uses **two quantifiers of the same type** (absolute or relative), then this constraint can be expressed by means of a fuzzy cardinality constraint using a quantifier that embraces both. For example, the constraint in Example 4.13 can be expressed with the fuzzy cardinality notation using a wider quantifier, instead of the quantifiers "approx_2" and "approx_8," such as "approx_between_2_and_8" (similar to the trapezoidal membership function in Figure 4.12 with a = 2 and b = 8). It can be observed that the resulting wider quantifier may be automatically generated starting from the other two quantifiers (min and max).
- If the (min,max) notation uses **two quantifiers of different types** (one absolute and one relative), then this restriction cannot be expressed with a single fuzzy cardinality quantifier, because different types of quantifiers have different domains and cannot be joined in another quantifier that embraces both.

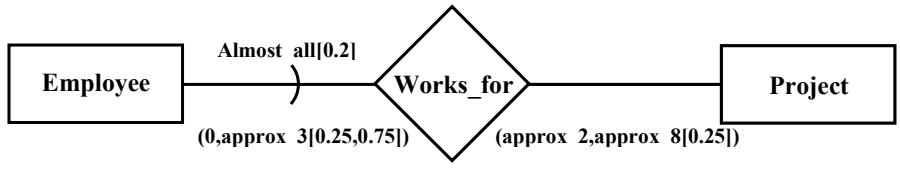
It is important to note that this second kind of (min,max) notation should be uncommon because perhaps it is not intuitive to check that the two fuzzy quantifiers are not contradictory. They can also be contradictory after a DML sentence. For example, if we use approx_half instead of approx_2 in Example 4.13 (refer to Figure 4.15), both quantifiers in (min,max) notation (approx_half and approx_8) are not contradictory when the number of employees is six, for example, because the constraint is (approx_3,approx_8). However, if the number of employees is 200, for example, both quantifiers are contradictory, because approx_100 is clearly greater than approx_8.

As in fuzzy cardinality constraints, because of their meanings, the (min,max) notation will preferably use two absolute quantifiers, although two relative quantifiers are also accepted here.

Fuzzy (min, max) Notation and Fuzzy Participation Constraints

On the other hand, the (min,max) notation and a fuzzy participation constraint are not exclusive. Although a fuzzy participation constraint establishes a global condition on all entity instances, the (min,max) notation individually restricts the relationship of each instance with the other participating entity.

Figure 4.16. Example 4.14: Fuzzy participation constraint with another constraint using fuzzy (min,max) notation



Example 4.14: If we merge Examples 4.11 and 4.13, then we obtain the model in Figure 4.16, which demonstrates that both constraints are different and coherent. It is also important to note the different meanings even though we use the following (min,max) notation: (almost_all,almost_all).

*

For these reasons, the most useful notations are the fuzzy (min,max) notation (mainly with absolute fuzzy quantifiers) and the notation for fuzzy participation constraints (mainly with relative fuzzy quantifiers). The notation for fuzzy cardinality restriction can be eliminated because this can be expressed with the fuzzy (min,max) notation.

In a new expression for fuzzy participation constraints, we can use fuzzy (min,max) notation instead of the quantifier Q in Definition 4.13. These minimum and maximum values restrict the quantity of entity instances related to the other entity. We must distinguish this notation for fuzzy participation and the usual fuzzy (min,max) notation. This new notation refers to the number of instances (in the constrained entity) related to the other entity. Usual fuzzy (min,max) notation refers to the number of instances in the other entity related to each instance in the constrained entity. This extension is not very useful, but the formal definition for it is easy, using Definitions 4.13 and 4.15.

Fuzzy Completeness Constraint on Specializations

In EER models, the relationship between a class and all its subclasses can be total or partial. In our fuzzy model, this constraint can be fuzzy mainly by utilizing a relative fuzzy quantifier, although as indicated in the case of participation constraints, they can also be absolute fuzzy quantifiers.

Definition 4.16: *Let E be a superclass, and S1, S2, ..., Sn the set of its n subclasses. A fuzzy completeness constraint is represented by an arc crossing the line between E and the specialization circle (or instead by a zigzag line), labeled with a quantifier Q and its required thresholds. This constraint asserts Equation 4.4, with ϕ defined by Equation 4.9 (or 1.64), where a is the number of E instances that belong to "any" subclass or subclasses, and b is the total number of instances in E.*

The warning area is defined when Equation 4.8 is satisfied.

*

Example 4.15: Let us consider the models in Figure 4.17 depicting an entity Employee, which is a superclass with two subclasses defined by the attribute Contract_Type: Permanent and Temporary. The relative fuzzy quantifier "almost all" (refer to Figure 1.26) indicates that "almost all employees must have a Permanent or Temporary contract, but other minority contract types may exist (work experience, grants, etc.)." These other contract types are not included in the model for various reasons (unknown types, types without own attributes, etc.).

*

Figure 4.17. Example 4.15: Fuzzy completeness constraint on an attribute-defined specialization with the defining attribute Contract_Type (with the two representations proposed in Definition 4.16)

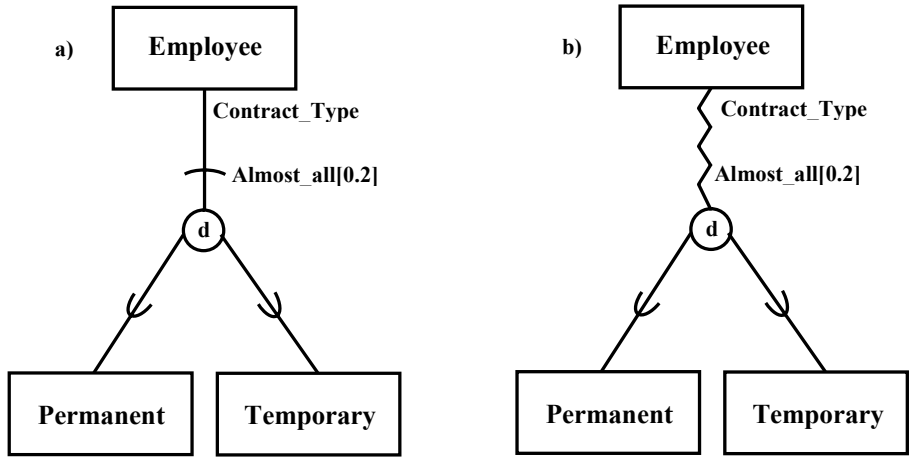
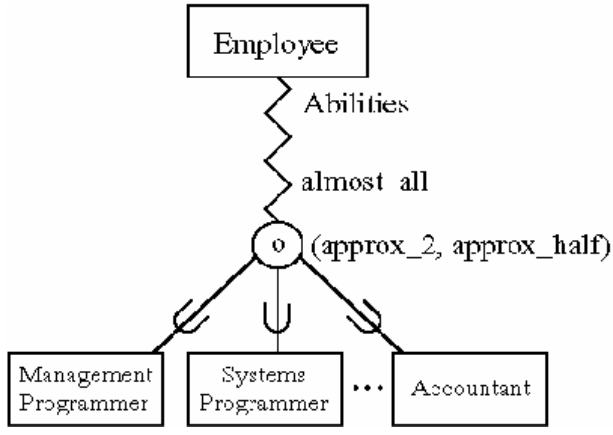


Figure 4.18. Examples 4.16 and 4.17: Fuzzy completeness constraint and fuzzy cardinality constraint on an overlapping specialization



In the previous example, the specialization is disjoint (with a "d" in the circle) because there cannot be an employee with various types of contracts. However, fuzzy constraints can also be applied to overlapping specializations (with an "o" in the circle), as shown in the following example.

Example 4.16: Let us consider an entity Employee that is a superclass with various subclasses defining the abilities of the employees: Management_Programmer, Systems_Programmer, Internet_Programmer, Analyst, Graphic_Designer, and Accountant, just as in Figure 4.18. A relative fuzzy quantifier such as "almost all" indicates that "almost all employees must have one or some of the abilities expressed in the subclasses."

*

In a new expression for fuzzy completeness constraints, we can use fuzzy (min,max) notation instead of the quantifier Q in Definition 4.16. These minimum and maximum values restrict the quantity of superclass instances that belong to "any" subclass. This extension is not very useful, but the formal definition for it is easy, using Definitions 4.15 and 4.16.

Fuzzy Cardinality Constraint on Overlapping Specializations

In an overlapping specialization we can also establish the minimum and maximum number of subclasses to which each member of the superclass can belong in a flexible manner. This can easily be expressed using the fuzzy (min,max) notation.

Definition 4.17: *Let E be a superclass and S1, S2, ...,Sb the set of b subclasses of an overlapping specialization. In addition, we suppose that ei with i = 1, 2, ..., n are the instances of E. A fuzzy cardinality constraint on this overlapping specialization is represented with a fuzzy (min,max) notation, (Qmin ,Qmax), next to the circle containing the letter "o" (overlapping). This constraint establishes that*

$$\lambda_{\min} \leq \phi_{\min,\,i} \wedge \lambda_{\max} \geq \phi_{\max,\,i} \;\forall\; i = 1, 2, \ldots, n \tag{4.23}$$

where

$$\Phi_{\min,i} = \begin{cases} a_i & \text{if } Q_{\min} \text{ is absolute} \\ a_i / b & \text{if } Q_{\min} \text{ is relative} \end{cases} \tag{4.24}$$

$$\Phi_{\max,i} = \begin{cases} a_i & \text{if } Q_{\max} \text{ is absolute} \\ a_i / b & \text{if } Q_{\max} \text{ is relative} \end{cases} \tag{4.25}$$

with a_i being the number of subclasses to which instance e_i belongs. Furthermore, $\lambda_{\min}$ and $\lambda_{\max}$ are defined in the same way as in Equations 4.17 and 4.18.

The warning area is defined when the thresholds $\delta_{\min}$ and $\delta_{\max}$ are used for $Q_{\min}$ and $Q_{\max}$, respectively. The warning message must be shown when Equation 4.23 is satisfied and the following equation is not satisfied:

$$\lambda'_{\min} \leq \phi_{\min,\,i} \wedge \lambda'_{\max} \geq \phi_{\max,\,i} \;\forall\; i = 1, 2, \ldots, n \tag{4.26}$$

where λ'_{min} *and* λ'_{max} *are defined in Equations 4.20 and 4.21. We can also apply De Morgan's law here.*

*

This fuzzy constraint has an effect on each superclass instance and must be satisfied by each one. In general, both min and max should be absolute quantifiers, although relative quantifiers will also be accepted (with regards to the total number of subclasses, value *b*).

Example 4.17: Continuing with Example 4.16, we can establish a fuzzy cardinality constraint on the overlapping specialization, such as (approx_2, approx_half).

This establishes the constraint whereby each employee must appear in a minimum of "approximately 2" skills and in a maximum of "approximately half" of the existing skills (or subclasses).

This schema is also depicted in Figure 4.18. It should be noted that the fuzzy quantifier almost_all is a fuzzy completeness constraint (zigzag line), and the (min, max) notation is used for a fuzzy cardinality constraint.

*

Finally, it is important to note that the quantifiers can be of any type (absolute or relative). In this case each quantifier can also be followed — optionally, of course — by one or two fulfillment degrees in square brackets [γ, δ], with the same meaning and default value as explained previously in the "Thresholds and Fuzzy Quantifiers for Relaxing Constraints" section.

Fuzzy Disjoint and Fuzzy Overlapping Constraints on Specializations

In specializations, the disjoint constraint specifies that the subclasses of the specialization must be disjoint. This means that an entity can be a member of at most one of the subclasses (zero or one). If the subclasses are not obliged to be disjoint, this is an overlapping specialization. Thus, it can be interesting to include to what extent the superclass instance belongs to each of the subclasses by using linguistic labels ("a lot," "a little," etc.) or, more simply, the membership degrees in the interval [0, 1].

It should be noted that each subclass is to be considered as a fuzzy subset of the superclass. As with all fuzzy sets, the elements of the subclass are not clearly defined, because each element can belong to the fuzzy set with a certain degree. We can use the concept of fuzzy entity (Definition 4.9). This definition allows for two new definitions, according to the specialization type:

Definition 4.18: *Let E be a superclass with n subclasses, S_1, S_2, ...,S_n, in a disjoint specialization. This specialization is a fuzzy disjoint specialization when at least one of the subclasses is a fuzzy entity (Definition 4.9), and for any instance e ∈ E, there is zero or one subclass S_i with i ∈ {1, 2, ..., n} such that*

$$\mu_i(e) > 0 \qquad (4.27)$$

where μi(e) is the membership degree of e to S_i. As with any disjoint specialization, $S_1 \cap S_2 \cap ... \cap S_n = \varnothing$.

This constraint will be represented by the letter "f" (fuzzy) before the letter "d" in the specialization circle, that is "fd".

*

Of course, if S_i is a nonfuzzy subclass, then μi(e) = 1 if e belongs to S_i, and μi(e) = 0 if e does not belong to S_i.

Definition 4.19: *Let E be a superclass with n subclasses, S_1, S_2, ..., S_n, in an overlapping specialization. This specialization is a fuzzy overlapping specialization when at least one of the subclasses is a fuzzy entity, and for any instance e ∈ E, there are zero or more subclasses S_i with i ∈ {1, 2, ..., n} such that μi(e) > 0, where μi(e) is the membership degree of e to S_i.*

This constraint will be represented by the letter "f" (fuzzy) before the letter "o" in the circle, that is, "fo".

*

Note that these definitions do not force all the subclasses to be fuzzy entities. Definition 4.19 is more flexible than Definition 4.18, because an instance e E may belong to some subclasses with different membership degrees.

These definitions have two points of view:

1. **From the point of view of subclasses**: Subclasses are fuzzy sets, and their underlying domain is all the superclass instances; that is, each superclass instance has a membership degree to each subclass (including the value 0). Let *S* be a subclass of *E*. Then the fuzzy set of *S* is represented by (using the format of Equation 1.1):

$$\{\mu_S(e_1) / e_1, \mu_S(e_2) / e_2, \ldots, \mu_S(e_m) / e_m\} \quad (4.28)$$

where e_i, with $i = 1, 2, ..., m$, are all the instances of superclass *E*, and $\mu_S(e_i)$ is the membership degree of e_i to subclass *S*.

2. **From the point of view of superclass instances**: Each superclass instance may belong to some subclasses. This membership is measured with a fuzzy set. The underlying domain of this fuzzy set is the set of all subclass names. Let S_j, with $j = 1, 2, ..., n$, be the *n* subclasses of *E*. Then the fuzzy set of instance e_i is

$$\{\mu_{S1}(e_i) / S_1, \mu_{S2}(e_i) / S_2, \ldots, \mu_{Sn}(e_i) / S_n\} \quad (4.29)$$

where $\mu_{Sj}(e_i)$, with $j = 1, 2, ..., n$, is the membership degree of e_i to subclass S_j. It is important to note that in disjoint specializations, the number of subclasses for a superclass instance is one.

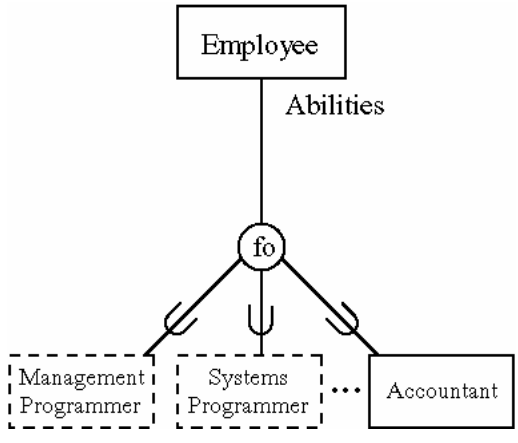
Both points of view work with fuzzy sets with a different discrete underlying domain. It may be represented with the format of Table 4.2, where the first point of view is represented by the fuzzy sets given by the columns, and the second point of view is given by the rows.

Example 4.18: Figure 4.19 indicates that our conceptual schema is also concerned with storing the extent to which each employee belongs to each of the subclasses. Thus, the set of system programmers is a fuzzy set (an employee can belong to this set with a certain membership degree), whereas we suppose that the set of accountants is not a fuzzy set (an employee can or cannot belong to this set). This is the first point of view.

Table 4.2. Representing fuzzy sets on a specialization with n subclasses and m superclass instances

Subclasses / Instances	S_1	S_2	...	S_n
e_1	$\mu_{S1}(e_1)$	$\mu_{S2}(e_1)$	...	$\mu_{Sn}(e_1)$
e_2	$\mu_{S1}(e_2)$	$\mu_{S2}(e_2)$	...	$\mu_{Sn}(e_2)$
⋮	⋮	⋮	...	⋮
e_i	$\mu_{S1}(e_i)$	$\mu_{S2}(e_i)$	...	$\mu_{Sn}(e_i)$
⋮	⋮	⋮	...	⋮
e_m	$\mu_{S1}(e_m)$	$\mu_{S2}(e_m)$	...	$\mu_{Sn}(e_m)$

Figure 4.19. Example 4.18: Fuzzy overlapping specialization



The second point of view starts with a particular employee; an employee who is an expert at programming management applications, although he or she may also be skilled in other types of applications and less skilled as an analyst, could be represented in the database by the following fuzzy set: {1/ Management_Programmer, 0.8/Systems_Programmer, 0.3/Analyst}. It should be noted that the underlying domain is the set of all the subclass names.

This database model allows us to make selections of the following type: "Find the name of the best management applications programmer amongst those who are not assigned to many projects and who is at least a *regular* analyst."

*

This constraint does not prevent the use of other fuzzy constraints (completeness or cardinality). These cases must be studied in order to define the method with which the DBMS ensures the fulfillment of these constraints:

1. If a **fuzzy completeness constraint** exists, then the DBMS must compute whether each superclass instance belongs to some subclass, for example, in order to decide if "almost all" superclass instances belong to some subclass. The problem is that membership is now fuzzy. The membership degree of an instance to the subclasses may be computed in various ways: (1) by using the greatest membership degree of this instance to any subclass, that is, the height (function Hgt in Definition 1.10) (Pedrycz & Gomide, 1998) of the second point of view fuzzy set; or (2) by using the fuzzy set cardinality (function Card in Definition 1.12) (Pedrycz & Gomide) of the second point of view fuzzy set (adding all the membership degrees), or by using generalized measures, such as the fuzzy set energy (Luca & Termini, 1974). We can certainly set a minimum threshold l in order to decide whether a superclass instance belongs to some subclass.

   Then, in order to compute the value of a in Definition 4.16, we must count how many instances of E have a fuzzy membership degree to "any" subclass or subclasses. The fuzzy membership degree is solved with the two previous options. The problem is then to count these elements. We propose the following four options, where b is the total number of instances in E, and a is computed by

$$a = \sum_{i=1}^{b} \alpha_i \tag{4.30}$$

   The definition of α_i gives the four options:

- Option 1:

$$\alpha_i = \begin{cases} \text{Hgt}(I_i) & \text{if Hgt}(I_i) \geq \lambda \\ 0 & \text{in any other case} \end{cases} \quad (4.31)$$

where I_i is the fuzzy set for instance $e_i \in E$ from the point of view of superclass instances. Value λ is the limit or minimum threshold to reject instances with a very small membership degree.

- Option 2:

$$\alpha_i = \begin{cases} \min(1, \text{Card}(I_i)) & \text{if Card}(I_i) \geq \lambda \\ 0 & \text{in any other case} \end{cases} \quad (4.32)$$

- Option 3:

$$\alpha_i = \begin{cases} 1 & \text{if Hgt}(I_i) \geq \lambda \\ 0 & \text{in any other case} \end{cases} \quad (4.33)$$

- Option 4:

$$\alpha_i = \begin{cases} 1 & \text{if Card}(I_i) \geq \lambda \\ 0 & \text{in any other case} \end{cases} \quad (4.34)$$

These four options are sufficiently efficient and allow the system to be very flexible. With a fixed λ, we can sort the four options according to the results of the count operation:

$$\text{Option 1} \leq \{\text{Option 2, Option 3}\} \leq \text{Option 4} \quad (4.35)$$

Options 2 and 3 cannot be sorted, because even though Card(I_i) ≥ Hgt(I_i), Option 3 adds 1 (if the height is greater than or equal to λ), whereas Option 2 adds a value less than or equal to 1 (if the cardinality is greater than or equal to λ).

2. If a fuzzy cardinality constraint exists (only on overlapping specializations), then the DBMS must compute the number of subclasses to which each superclass instance belongs. For example, in order to decide if the number of subclasses of a superclass instance is between "approximately 2" and "approximately half" of the existing subclasses, using fuzzy (min,max) notation. However, this number is not simple, as membership is now fuzzy. This problem may be solved in two ways: (1) by using the fuzzy set cardinality (Pedrycz & Gomide, 1998) of the second point of view fuzzy set or by using generalized measures, such as the fuzzy set energy (Luca & Termini, 1974); or (2) by counting the number of subclasses with a membership degree greater than a minimum value (usually 0). After the DBMS has computed this number, the system must check to see whether this number satisfies the fuzzy cardinality constraint.

The cardinality of a fuzzy set can be a complex problem and has been studied by various authors, especially Dubois and Prade (1985a) and Delgado, Sánchez, and Vila (2000). Nevertheless, in this application efficiency is very important (especially in large databases), but other methods can also be used.

This definition complements the definition of fuzzy types given in Marín et al. (2000).

Fuzzy Attribute-Defined Specializations

There are certain kinds of fuzzy attributes, summarized in the "Fuzzy Attributes" section, earlier in this chapter. We can define an attribute-defined specialization (Elmasri & Navathe, 2000) by using fuzzy attributes:

Definition 4.20: *A **fuzzy attribute-defined specialization** is exactly the same as an attribute defined specialization in EER models where this attribute is a fuzzy attribute. It is represented with an angled line joining*

*the superclass with the circle. This line will be labeled with the name of fuzzy attribute Type n, preceded by the text "***T*n*:***".*

*

This constraint establishes that every subclass instance has a valid value (in a certain fuzzy range) for that attribute, according to the subclass. In general, each subclass corresponds with one of the linguistic labels defined on this attribute. Each subclass would be a fuzzy entity, but this is not mandatory. For example, in Figure 4.19 the attribute Abilities would be considered as a fuzzy attribute Type 3. It should be noted that this makes it necessary to define a similarity relation on all the subclasses.

This definition is independent of all constraints such as fuzzy or crisp disjoint or overlapping specializations. The classification of each instance *e* of superclass *E* is then an automatic process, according to the characteristics of the specialization:

1. **Fuzzy disjoint (fd)**: Instance e is assigned to one subclass *S*. Subclass *S* is the subclass with a greater value of $\mu_S(e)$ (membership degree of *e* to *S*). This membership degree is stored only if *S* is a fuzzy entity.
2. **Fuzzy overlapping (fo)**: Instance *e* is assigned to all subclasses S_i, such that $\mu_{Si}(e) > 0$. These membership degrees are stored only in the fuzzy subclasses (subclasses that are fuzzy entities).
3. **Nonfuzzy disjoint (d)**: Instance *e* is assigned to one subclass *S*. Subclass *S* is the subclass with a greater value of $\mu_S(e)$, but this membership degree is not stored and is considered as 1.
4. **Nonfuzzy overlapping (o)**: Instance e is assigned to all subclasses S_i, such that $\mu_{Si}(e) > 0$, but these membership degrees are not stored and are considered as 1.

These four cases may be used with the four fuzzy attribute types. Then, 16 different possibilities are produced.

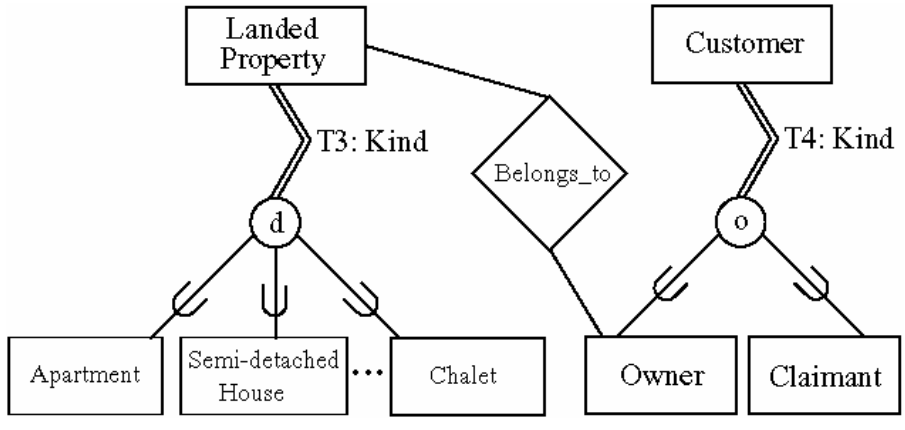
The following example shows two fuzzy attribute-defined specializations (disjoint and overlapping). In one specialization, each pair of subclasses has a fuzzy similarity degree between them (Type 3). This property is useful for comparing them and for searching the more important instances in some queries. In the other specialization there is no similarity relation (Type 4).

Example 4.19: The conceptual model represented in Figure 4.20 states that in a real estate agency, every landed property belongs to one subclass, which has its own attributes. Thus, this is a total disjoint specialization (a double line and a "d" inside the circle). The attribute Kind is a fuzzy attribute Type 3, because if one person is looking for a chalet, for example, then this customer is possibly interested in semidetached houses, because these two types are similar. This factor is taken into account in order to show all the relevant properties to our customer. In this sense, fuzzy queries are studied in Chapter VII and in this context in Galindo, Medina, and Cubero, and Pons (1999) and Urrutia and Galindo (2002). It should be noted that the subclasses are not fuzzy, because every landed property belongs to only one subclass.

Every landed property has an owner, who is a customer. Another kind of customer is a claimant who is looking for a landed property. The overlapping specialization results in the fact that one customer may be an owner and a claimant at the same time. The fuzzy attribute Type 4, Kind, allows us to store possibility distributions over the subclasses in order to express any fuzzy concept. In this example we are interested in measuring the urgency of the customer. Thus, a customer with the value {0.4/Owner, 1/Claimant} is a customer who is urgently looking for a landed property and who is offering some property without urgency. It can be seen that the subclasses are not fuzzy, because a customer is or is not an owner and/or a claimant.

*

Figure 4.20. Example 4.19: Fuzzy attribute-defined disjoint specialization with total participation constraint



Example 4.20: Figure 4.22 includes another three examples of fuzzy attribute-defined specializations using two fuzzy overlapping specializations and one disjoint specialization. The first one is a specialization with a total participation constraint (double line), and it establishes that all employees must belong to one or more categories. In addition, Category is a fuzzy attribute Type 3.

The second one is a specialization with a fuzzy participation constraint with the fuzzy quantifier almost_all in the labeled arc: Almost all researchers must belong to one or more research lines. In addition, Research_Line is a fuzzy attribute Type 3. We use a labeled arc instead of a zigzag line in the fuzzy participation constraint because in this case it is clearer.

The third one is a disjoint specialization with a total participation constraint, and it establishes that all temporary employees are beginners or seniors, according to their seniority (or antiquity). Subclasses are not fuzzy because we do not want to store the membership degree. In addition, a temporary employee cannot belong to both subclasses. The antiquity is a crisp and known value, but we can make flexible queries by using this attribute; that is, it is a fuzzy attribute Type 1.

*

Fuzzy Constraints in Union Types or Categories: Participation and Completeness

In the EER Model we can also find the union types or categories (Elmasri, Weeldreyer, & Hevner, 1985; Elmasri & Navathe, 2000). This represents the case when some different superclasses may or may not be members of a special subclass (called category). By definition, each member of the subclass or category must be a member of at least one of the superclasses. Union types are represented with the union symbol inside a circle. Superclasses are joined to that circle by a line. The subclass or category is joined to that circle by a single line with the inclusion symbol. Furthermore, in partial categories it is possible that superclass instances do not belong to the category, because the category is a subset of the union of all superclasses.

It should be noted that the total categories (double line) indicate that all the superclass instances belong to the category. In this case, the category may be represented by using a generalization in which the category is transformed into a superclass with total participation constraint.

In this type of specialization, it is possible to apply fuzzy constraints in two ways:

Definition 4.21: *Let C be a category (or subclass) of a union type, with n superclasses: E_i with i* = 1, 2, ..., n. *A **fuzzy participation constraint in one or more superclasses** is represented by an arc crossing the lines that join the selected superclasses with the circle. The arc must be labeled with its fuzzy quantifier or with the fuzzy (min, max) notation.*

The selected superclasses are those superclasses that are constrained. They are denoted by E_j, $\forall\, j \in J$ with $J \subseteq \{1, 2, \ldots, n\}$. The union of the selected superclasses is denoted by ∇:

$$\nabla = \bigcup_{j \in J} E_j \tag{4.36}$$

1. *If the arc is labeled with the quantifier Q, this constraint establishes Equation 4.4, with* ϕ *defined by Equation 4.9, where a is the number of instances in* ∇ *that belong to C, and b is the total number of instances in* ∇.
2. *If the arc is labeled with the fuzzy (min, max) notation (Q_{min}, Q_{max}), this constraint establishes that*

$$\lambda_{\min} \leq \phi_{\min} \wedge \lambda_{\max} \geq \phi_{\max} \tag{4.37}$$

where

$$\lambda_{\min} = \min\{\alpha : \alpha = Q_{\min}^{-1}(\gamma_{\min})\} \tag{4.38}$$

$$\lambda_{\max} = \max\{\beta : \beta = Q_{\max}^{-1}(\gamma_{\max})\} \tag{4.39}$$

where $\gamma_{\min}$ *and* $\gamma_{\max}$ *are the minimum thresholds for* $Q_{\min}$ *and* $Q_{\max}$, *respectively, and*

$$\Phi_{\min} = \begin{cases} a & \text{if } Q_{\min} \text{ is absolute} \\ a/b & \text{if } Q_{\min} \text{ is relative} \end{cases} \quad (4.40)$$

$$\Phi_{\max} = \begin{cases} a & \text{if } Q_{\max} \text{ is absolute} \\ a/b & \text{if } Q_{\max} \text{ is relative} \end{cases} \quad (4.41)$$

with a and b being the same values defined in previous case.

The warning area is similarly defined, using δ_1 *and* δ_2*, respectively.*

*

This constraint restricts the number of instances (in the union ∇ of any group of superclasses) that belong to the category. The fuzzy quantifier will normally be relative. For example, with the quantifier "almost all" on one superclass, the constraint states, "Almost all the superclass elements belong to the category." Another option is to join two or more superclasses with an arc, indicating that the union of instances of these superclasses is constrained in participation. This constraint allows the use of the (min, max) notation, indicating the minimum and maximum number of instances in ∇ that belong to the category (using absolute or relative fuzzy quantifiers), and in this case, we must perform the observations appearing in the "Fuzzy (min,max) Notation on Relationships" section in order to express a good constraint.

Definition 4.22: *A* ***fuzzy completeness constraint in the category*** *(on the union of all superclasses) is represented by an arc crossing the line that joins the category with the circle. The arc is labeled with one fuzzy quantifier, or with the fuzzy (*min, max*) notation. This constraint is a fuzzy participation constraint (Definition 4.21) embracing all superclasses: J =* {1, 2, ..., n}.

*

This constraint restricts the number of instances of all superclasses (the union) that belong to the category. This fuzzy quantifier will normally be relative. For example, with the quantifier "almost all" on the category, the constraint states, "Almost all elements of all superclasses belong to the category." This constraint

also allows the use of the fuzzy (min,max) notation, indicating the minimum and maximum number of all superclass instances that belong to the category. It should be noted that this second way always refers to all the superclasses instances, that is, to the union of all the superclasses. Consequently, relative fuzzy quantifiers are preferable in this constraint.

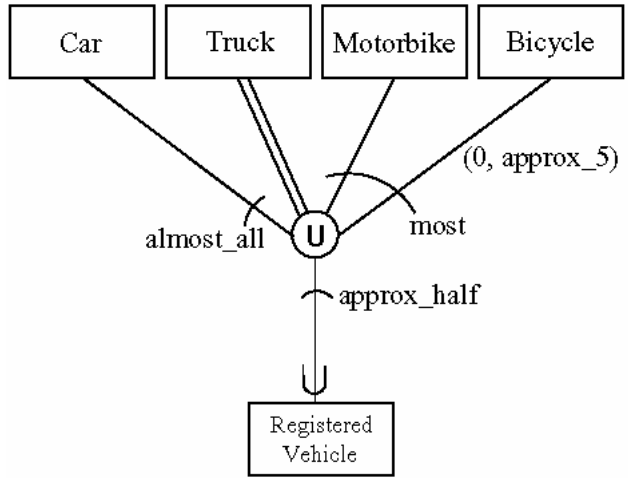
Example 4.21: Let us consider four entity types for vehicles: Car, Truck, Motorbike, and Bicycle. Some vehicles may belong to the Registered Vehicle entity. Figure 4.21 depicts this model with some participation constraints: Almost all the cars must be registered vehicles. All the trucks must also be registered. Moreover, the model allows a maximum of approximately five bicycles to be registered vehicles. The arc labeled with the fuzzy quantifier "most" indicates that most motorbikes or bicycles (its union) must be registered.

For the sake of simplicity, we introduce a fuzzy completeness constraint in the same specialization. This constraint establishes that approximately half of the existing vehicles must be registered vehicles.

*

In real models, fuzzy constraints in the same specialization must be mixed with care.

Figure 4.21. Example 4.21: Fuzzy constraints on a union type or category



Fuzzy Constraints in Intersection Types or Shared Subclasses: Participation and Completeness

A shared subclass (or intersection type) is a subclass with several superclasses (Elmasri & Navathe, 2000). Each member of the subclass must be a member of all the superclasses; that is, the subclass is a subset of the intersection of all the superclasses. A shared subclass is represented, joining it with all its superclasses by a single line and the inclusion symbol. Another representation utilizes the intersection symbol inside a circle: Superclasses are joined to that circle by a line, and the subclass is joined to that circle by using a single line with the inclusion symbol.

As with union types, in this type of specialization it is possible to apply fuzzy constraints in two ways:

Definition 4.23: *Let S be a shared subclass (of an intersection type), with n superclasses:* E_i *with* $i = 1, 2, ..., n$. *A* ***fuzzy participation constraint in one or more superclasses*** *is represented by an arc crossing the lines that join the selected superclasses with the circle. The arc must be labeled with its fuzzy quantifier or with the fuzzy* (min, max) *notation.*

The selected superclasses are those superclasses that are constrained. They are denoted by E_j, $\forall\, j \in J$ *with* $J \subseteq \{1, 2, \ldots, n\}$. *The intersection of the selected superclasses is denoted by* Δ:

$$\Delta = \bigcap_{j \in J} E_j \tag{4.42}$$

1. *If the arc is labeled with the quantifier Q, then this constraint establishes Equation 4.4, with* ϕ *defined by Equation 4.9, where a is the number of instances in* Δ *which belong to S, and b is the total number of instances in* Δ.
2. *If the arc is labeled with the fuzzy* (min, max) *notation* (Q_{min} , Q_{max}), *then this constraint establishes the constraint expressed in Equation 4.37, with* ϕ_{min} *and* ϕ_{max} *computed by using the values a and b defined in the previous case of this definition.*

The warning area is similarly defined by using δ_1 *and* δ_2, *respectively.*

*

This constraint restricts the number of instances in the intersection of any group of superclasses (Δ) that belong to the shared subclass. This fuzzy quantifier should be relative. For example, with the quantifier "almost all" on one superclass, the constraint expresses that "almost all the superclass elements belong to the shared subclass." Another option is to join two or more superclasses with the arc, indicating that the intersection of instances of those superclasses are constrained in participation. This constraint allows the use of the fuzzy (min, max) notation, indicating the minimum and maximum number of instances in Δ that belong to the shared subclass (using absolute or relative fuzzy quantifiers). Generally speaking, the participation constraint is not useful, as one constraint on one superclass (or on several superclasses) depends on the membership of its instances to the other superclasses (it should be remembered that the subclass is a subset of the intersection).

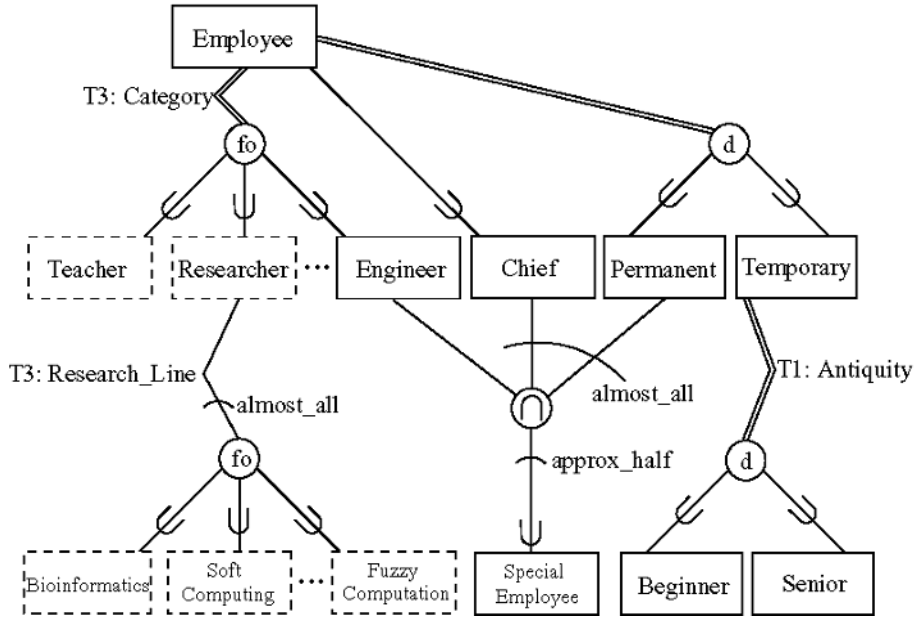
Definition 4.24: *A fuzzy completeness constraint in a shared subclass (on the intersection of all superclasses) is represented by an arc crossing the line that joins the shared subclass with the circle. The arc is labeled with one fuzzy quantifier, or with the fuzzy* (min, max) *notation. This constraint is a fuzzy participation constraint (Definition 4.23) embracing all superclasses:* $J = \{1, 2, ..., n\}$.

*

This constraint restricts the number of instances in the intersection of all the superclasses that belong to the shared subclass. This fuzzy quantifier will normally be relative. For example, with the quantifier "almost all" on the shared subclass, the constraint states, "almost all the elements of the intersection of all the superclasses belong to the shared subclass." This constraint also allows the fuzzy (min, max) notation to be used, indicating the minimum and maximum number of instances in the intersection (of all the superclasses) that belong to the shared subclass. Notice that this constraint is always referred to as the intersection of all superclasses.

Example 4.22: Let us consider an entity for Special Employees with its own attributes (extra payment, number of awards, motive, etc.). A member of this shared subclass must be an engineer, a chief (boss), and a permanent employee. Figure 4.22 depicts this model with the following participation constraint: Almost all the chiefs and permanent employees must be special

Figure 4.22. Examples 4.20 and 4.22: Three fuzzy attribute-defined specializations and fuzzy constraints in a shared subclass.



employees. It is interesting to note how this constraint means that almost all the chiefs and permanent employees must also be engineers (because all special employees belong to the engineer superclass).

On the other hand, the fuzzy completeness constraint establishes that approximately half of the employees who are engineers, chiefs, and permanent employees must be special employees.

*

Comparison of Some Fuzzy Models

In Chapter III we discussed some conceptual models proposed by other authors. None of these investigations uses a CASE support tool proposed to help in a system design that involves uncertainty. Our proposal has a tool called FuzzyCASE, which allows us to model by using EER and FuzzyEER tools. It incorporates all the notations shown in this work and in other works related to

the FuzzyEER Model (such as the fuzzy constraints that are not detailed in Table 4.3 but that have not been dealt with by any author in his or her publications).

Some of the most important models are those proposed by Chaudhry, Moyne, and Rundensteiner (1994, 1999), Yazici and Merdan (1996), Chen (1998), and Ma, Zhang, Ma, and Chen (2001). Table 4.3 shows a comparison of the FuzzyEER Model with those models. Each cell shows a "Yes" if the model has that component or modeling tool (even if it has another notation). In the opposite case, the cell is empty. On the other hand, if the cell has a "Yes*," the component has been confined in that model but with different characteristics than those of the FuzzyEER Model, or its characteristics are limited and more reduced than those of the FuzzyEER Model proposed here. In general this difference is caused by the use of different types of domains and treatment of imprecision, or by a type of degree.

Conclusion and Future Lines

Fuzzy databases have been widely studied, with the aim of allowing the storage of imprecise or fuzzy data and the imprecise queries about the existing data (Petry, 1996; Medina, 1994; Galindo, 1999).

However, the application of fuzzy logic to databases has traditionally paid little attention to the problem of the conceptual model, just as Chaudhry, Moyne, and Rundensteiner, (1999) affirm. Few investigations study a complete and exhaustive notation of the many characteristics, which may be improved by using fuzzy logic (see Chapter III), and none of these works refer to the possibility of extending constraints by using the tools offered by fuzzy sets theory. The FuzzyEER Model intends to do so, and in the beginning of this chapter we focused on the following: types of fuzzy attributes (T1, T2, T3, and T4), fuzzy degrees associated or not associated with different items and with different meanings, degrees with respect to the model, fuzzy aggregations, fuzzy entities and relationships, fuzzy weak entities, and degrees in a specialization.

All these concepts allow us to extend the EER Model to the FuzzyEER Model defined in this chapter. Therefore, it may be stated that a data model that contemplates fuzzy data allows us to represent a type of data in an information system, which the traditional systems do not deal with and therefore lose this information. This reduces the risk of obtaining empty answers from queries in the database, because fuzzy logic allows us to use a finer scale of discrimination,

Table 4.3. Comparison of some fuzzy models: FEER, FERM, ExIFO, Fuzzy ER, and FuzzyEER

Components \ Fuzzy Models	FEER Ma et al. (2001)	FERM Chaudhry et al. (1999)	ExIFO Yazici and Merdan (1996)	Fuzzy ER Chen (1998)	FuzzyEER
1. Fuzzy values in fuzzy attributes	Yes*	Yes*	Yes*	Yes*	Yes
Type 1					Yes
Type 2	Yes*	Yes*	Yes*	Yes*	Yes
Type 3	Yes*		Yes*		Yes
Type 4					Yes
2. Fuzzy degree associated to an attribute	Yes*		Yes*	Yes*	Yes
3. Fuzzy degree assoc. to some attributes					Yes
4. Fuzzy degree with its own meaning					Yes
5. Fuzzy degree to the model	Yes*		Yes*	Yes	Yes
6. Fuzzy entities	Yes*		Yes*	Yes*	Yes
7. Fuzzy weak entity (existence)		Yes*			Yes
8. Fuzzy weak entity (identification)					Yes
9. Fuzzy relationship	Yes*		Yes*	Yes*	Yes
10. Fuzzy aggregation of entities	Yes*		Yes*	Yes*	Yes
11. Fuzzy aggregation of attributes	Yes		Yes	Yes	Yes
12. Fuzzy degree in the specialization	Yes		Yes*	Yes	Yes
13. Fuzzy degree in the subclasses	Yes			Yes	Yes
14. Fuzzy constraints				Yes*	Yes
15. Graphic and CASE Tool					Yes

** The component has been defined in that model but with more limited characteristics than those of the FuzzyEER Model.*

as it considers the interval [0, 1] instead of the set {0, 1}. In other words, it allows us to recover instances that would not be obtained by using a precise method, as they only partially meet the imposed conditions. Furthermore, the set of instances can be ordered according to the level at which it satisfies the

conditions. This leads the way for creating queries and operations, which would be nonviable in a traditional system.

In the "Fuzzy Constraints" section, we presented a system for expressing flexible constraints, which can be used in a conceptual model utilizing the FuzzyEER Model. These restrictions can be represented by using fuzzy quantifiers (refer to Chapter I). The constraints studied are the fuzzy participation constraint, the fuzzy cardinality constraint, the fuzzy completeness constraint on specializations, the fuzzy cardinality constraint on overlapping specializations, fuzzy disjoint and fuzzy overlapping constraints on specializations, fuzzy attribute-defined specializations, fuzzy participation and completeness constraints in union types or categories, and fuzzy participation and completeness constraints in intersection types or shared subclasses.

In addition, we have studied the fuzzy (min, max) notation and shown how this notation can substitute fuzzy cardinality constraints and that a fuzzy cardinality constraint can only substitute the (min, max) notation if both quantifiers are of the same type (absolute or relative). Despite this equivalence in the majority of cases, we consider that it is preferable to use the (min, max) notation for greater clarity.

The studied constraints on specializations include and improve the types of constraints proposed in Varas, Contreras, and Campos, (1998), which are not considered in other models (Elmasri & Navathe, 2000). Our proposal improves on these constraints, because it uses the power and flexibility offered by fuzzy sets theory.

The fuzzy constraints have a novel meaning and offer great expressiveness to the conceptual model. Furthermore, the conceptual model continues to be an easy-to-understand system of expression even for nontechnical users, which is fundamental in conceptual modeling.

Some cases where the FuzzyEER Model has been applied and the results have been published include the following: for the quality control of the paper (Urrutia, 2002, 2003), for the management of a real estate agency (Urrutia & Galindo, 2002; Urrutia, 2003) and for museum exhibitions using the Internet (Aranda, Galindo, & Urrutia, 2002).

Chapter V studies how all the information that FuzzyEER can model can be represented in a relational database, and Chapter VI discusses the mapping of FuzzyEER Model concepts to relations, giving a FuzzyEER-to-relational mapping algorithm.

Some of the FuzzyEER notations may be used in the FSQL (Fuzzy SQL) server, which is an extension of SQL for permitting fuzzy queries and operations

(Galindo, Medina, Pons, & Cubero, 1998; Galindo, Medina, Cubero, & Pons, 1999; Galindo, 1999, 2005). In Chapter VII, we extend the first definition of FSQL in order to use all capabilities of the FuzzyEER Model.

In the near future we will try to study new useful components for FuzzyEER, to study how all the information that FuzzyEER can model can be represented in object-oriented databases, and to include fuzzy capabilities into deductive database modeling (Di Battista & Lenzerini, 1993; Blanco, Cubero, Pons, & Vila, 2000; Blanco, 2001) and into temporal database modeling (Gregersen & Jensen, 1999).

With regard to fuzzy constraints, an interesting research line is to achieve notational constructs to allow a greater selection of other fuzzy integrity constraints. For example, relaxing the constraints proposed in Davis and Bonnell (1989). In order to facilitate the task of using fuzzy quantifiers on the part of designers, another interesting study would be to classify the quantifiers that can be used in natural language and the relationships among them. As previously indicated, one constraint can be established with various fuzzy quantifiers, and in this case, the use of certain quantifiers conditions and limits the possibility of using others in the same constraint.

Other important future lines of works are (1) studying the repercussions of a fuzzy relationship between two entities with fuzzy constraints, (2) studying the repercussions of the inheritance characteristic with fuzzy entities and constraints, and (3) relaxing the universal quantifier (quantifier), which refers to *all* instances of any entity (for example, see Equations 4.10, 4.12, 4.14, 4.19, 4.23, and 4.26). The next step will be to define an automatic transformation of these fuzzy constraints into a fuzzy DBMS to create the necessary elements (e.g., triggers and assertions) in order to assure the fulfillment of the fuzzy constraints.

Object-modeling methodologies, such as OMT (Object-Modeling Technique) and UML (Universal Modeling Language), are becoming increasingly popular in software design and engineering, but an important part of these methodologies is similar in many ways to EER diagrams. The FuzzyEER concepts are easily mapped to these object-modeling methodologies.

Chapter V

Representation of Fuzzy Knowledge in Relational Databases: FIRST-2

The Relational Model was developed by E.F. Codd of IBM and published in 1970. It is currently the most used and has been a milestone in the history of databases, revolutionizing the market. In fact, relational databases have been the most widespread of all databases. On a theoretical level, many Fuzzy Relational Database models (Chapter II), which are based on the relational model, extend this so that vague and uncertain information can be stored and/or treated with or without fuzzy logic (see Chapter I).

The FuzzyEER Model (see Chapter IV) is an extension of the EER Model for creating conceptual schemas with fuzzy semantics and notations. This extension is a good eclectic synthesis between different models (see Chapter III) and provides new and useful definitions: fuzzy attributes, fuzzy entities, fuzzy relationships, fuzzy specializations, and so forth.

In this chapter, we propose the incorporation of FuzzyEER concepts into a relational DBMS. Our aim is to present this extension as simply and usefully as possible. We then extend the FIRST definitions (Medina, 1994; Medina, Pons, & Vila, 1995; Galindo, 1999), which have been used in some applications

(Blanco, Cubero, Pons, & Vila, 2000; Carrasco, 2003). FIRST-2 is the extension of FIRST in order to incorporate these new definitions.

Chapter VI describes the steps of an algorithm for FuzzyEER-to-FIRST-2 mapping. Chapter VII defines the FSQL language (Fuzzy SQL), which facilitates fuzzy database access and creation (queries, updates, etc.). Although FSQL is independent of FIRST-2, FSQL needs a fuzzy database as powerful as the one represented by using FIRST-2. Chapter VIII shows some applications of fuzzy databases, FIRST-2, and the FSQL language.

The section "Fuzzy Values: Fuzzy Attributes and Fuzzy Degrees" in Chapter IV showed the fuzzy attributes included in the FuzzyEER Model. We then define how to represent fuzzy data and fuzzy metaknowledge data.

For each fuzzy attribute type, it is necessary to clarify two aspects:

1. How to represent the values (which the attribute can store). This question is examined in the first two main sections in this chapter.
2. What information needs to be stored in the Fuzzy Metaknowledge Base (FMB) to process it, and how this information should be organized. This question is explained in the third main section of this chapter.

The FMB will be responsible for organizing all the information related to the inexact nature or context of these attributes. The FMB is contemplated as an extension of the catalogue of the system (Data Dictionary), and it organizes the information by using tables or relations.

In this chapter, we focus on these two aspects. Firstly, for each fuzzy type of information, we define its representation in the database of data, and then we detail the structure of the FMB, clarifying the second point. Certain approaches sharing this objective, such as Bosc and Galibourg (1989), have focused on queries rather than on representation issues.

In this representation, the following aspects have predominated (Medina, 1994):

- Execution speed against storage economy: For some of the types that this attribute can collect, it might be possible to use a more compact representation. However, this would result in a slower execution of the operations that involve fuzzy attributes.
- Uniformity in the presentation: Classical attributes are used to represent fuzzy attributes.

- Use of the elements of the RDBMS host to represent the information in a relational format. This criterion enables the reduction of any fuzzy operation to the terms of the relational classic model.

Representation of Fuzzy Values in Fuzzy Attributes

In this section, we study those fuzzy attributes that admit (either in their storage or processing) fuzzy values, which are more complex than a simple fuzzy degree. These attributes are the four fuzzy types that we list in Chapter IV.

Fuzzy Attributes Type 1

These attributes are represented as usual attributes because they do not allow fuzzy values. Nevertheless, information is stored in the FMB about the nature or context of them. They are classical attributes that admit fuzzy processing, and we will be able to perform fuzzy (flexible) queries by using the labels, for example.

Fuzzy Attributes Type 2

This type of attribute enables the storage of inexact information on ordered underlying domains. In Table 5.1, we show the system used to represent the fuzzy attributes Type 2. There, we can see that a fuzzy attribute Type 2, called **F**, for example, does in fact comprise five classical attributes:

- **FT**: This attribute stores the *type of value* corresponding to the data that we want to store, indicating its representation. This is one of the following types: UNKNOWN (0), UNDEFINED (1), NULL (2), CRISP (3), LABEL (4), INTERVAL (5), APPROXIMATE VALUE #*d* with implicit *margin* (6), TRAPEZOIDAL (7), APPROXIMATE VALUE $d \pm m$ with explicit *margin m* (8), POSSIBILITY DISTRIBUTION-2 with 2 values (9), and POSSIBILITY DISTRIBUTION-4 with 4 values (10).

Table 5.1. Internal representation for fuzzy attributes Type 2 (for an attribute **F***)*

Type of values	**Attributes in the DB for each fuzzy attribute Type 2**				
	FT	**F1**	**F2**	**F3**	**F4**
UNKNOWN	0	NULL	NULL	NULL	NULL
UNDEFINED	1	NULL	NULL	NULL	NULL
NULL	2	NULL	NULL	NULL	NULL
CRISP *d*	3	*d*	NULL	NULL	NULL
LABEL	4	FUZZY_ID	NULL	NULL	NULL
INTERVAL [*n*,*m*]	5	*n*	NULL	NULL	*m*
APPROXIMATE VALUE #*d*	6	*d*	*d – margin*	*d + margin*	*margin*
TRAPEZOIDAL	7	α	β	γ	δ
APPROXIMATE VALUE *d* ± m	8	*d*	*d – m*	*d + m*	*m*
POSSIBILITY DISTRIBUTION-2	9	p_1	d_1	p_2	d_2
POSSIBILITY DISTRIBUTION-4	10	d_1	d_2	d_3	d_4

- **F1, F2, F3, and F4**: The names of these attributes are formed by adding the numbers 1, 2, 3, and 4 to the attribute name. They store the description of the parameters that define the data and depend on the type of value (attribute **FT**).

For each possible type of value of the attribute **FT**, we have the following meaning of the other four attributes:

- **UNKNOWN, UNDEFINED, NULL**: These three values do not need any parameter for which, as Table 5.1 shows, the other 4 attributes remain Null (this value is the Null of the SGBD host, not the NULL of the fuzzy value).
- **CRISP**: A crisp value *d* needs one parameter, **F1,** in which the crisp value will be stored: *d*.
- **LABEL**: A label type value needs only one parameter to store the identifier associated to the label (FUZZY_ID). This indicator is useful to be able to access the FMB and to obtain the associated description to the label. Note that the labels are stored in the FMB, and when an attribute takes the value of a label, only the identifier of the label is stored, not its definition. If we change the definition of a label in the FMB, then we are also changing all the values that store that label.
- **INTERVAL**: This attribute needs the two extreme values of the interval [*n*, *m*], which are stored in **F1** and **F4**, respectively.

- **APPROXIMATE VALUE *d***: This value needs only a value that is stored in **F1** and is the central value of the triangular possibility distribution (refer to Figure 1.2b). Nevertheless, in order to reduce operations (mathematics and access to data), FIRST-2 takes advantage of the attributes **F2, F3,** and **F4** to store the values $d - margin$, $d + margin$, and *margin*, respectively. The *margin* value is a value stored in the FMB for each fuzzy attribute Type 2 (or 1), and its value depends on the meaning of this attribute. This allows us to store approximate values without indicating the *margin* by using the default *margin* stored in the FMB. If the *margin* of the FMB changes, these values must also change.
- **TRAPEZOIDAL**: This attribute must necessarily store the four values that identify the trapezoidal function $[\alpha, \beta, \gamma, \delta]$ (refer to Figure 7.1). Although the labels are also trapezoidal, this type allows us to store a trapezoidal function without having a defined label for that function.
- **APPROXIMATE VALUE $d \pm m$**: This value and the following ones are new in FIRST-2 and need the central value of the triangular possibility distribution (refer to Figure 1.2b), which is stored in **F1.** The attributes **F2, F3,** and **F4** store the values $d - m$, $d + m$, and m, respectively. The m value is now the *margin*, but in this type of value, the m value is not associated to the FMB. Thus, if the FMB changes, these values do not change. The user, of course, can change this type of value by using DML statements.
- **POSSIBILITY DISTRIBUTION-2 (2 values)**: This type stores one or two possible values (d_1 and d_2). Each one has a possibility degree (p_1 and p_2, respectively). We can represent this value with the possibility distribution $\{p_1/d_1, p_2/d_2\}$.
- **POSSIBILITY DISTRIBUTION-4 (4 values)**: This type stores one, two, three, or four possible values (d_1, d_2, d_3, and d_4), with possibility degree 1 for all of them. Thus, we can represent this value with the possibility distribution $\{1/d_1, 1/d_2, 1/d_3, 1/d_4\}$.

FIRST-2 does not allow any type of possibility distribution on a numeric underlying domain because these values are unusual. The values POSSIBILITY DISTRIBUTION-2 and -4 represent the types 6 and 4, respectively, of the GEFRED Model (see Table 2.4), limiting the number of possible values. However, it is easy to extend the representation of fuzzy attributes Type 2 in order to extend this limit. In fact, FIRST-2 does not allow many types to be

represented (compare this representation with the types of membership functions in Chapter I).

To summarize, for a fuzzy attribute Type 2, the representation uses five attributes to store the code of the value type (**FT**), and the attributes **F1, F2, F3,** and **F4** to store the parameters of each datum. FIRST represents the trapezoidal value storing $\beta - \alpha$ in **F2** and $\gamma - \delta$ in **F3**, but it does not imply any advantage — these values are useful for obtaining the linear functions, but in order to decide what function to use, we need the values β and γ. Although FIRST-2 does not allow us to store the extended trapezoid function (refer to Chapter I), the extension for including this type of membership function is easy. In such a case each fuzzy attribute would need more attributes in order to store a list of points `P1/N1`, ... `Pn/Nn`, where all the `Pi` belong to [0, 1] and all the `Ni` are between α and β or between γ and δ. The maximum value of n should be stored in the FMB (in the attribute LEN of the FUZZY_COL_LIST table; see the corresponding section on this table later in this chapter).

Fuzzy time attributes are treated as fuzzy attributes Type 2, where the underlying domain is the domain of date/time data types. Thus, in these fuzzy attributes, **F1**, **F2**, **F3**, and **F4** are not numeric. The values in the FMB (such as *margin*, labels, etc.) may be stored in seconds, minutes, hours, and so forth (see attribute UM in the FUZZY_COL_LIST table, later in this chapter). If the context needs date/time data types, then the FMB must copy its tables, changing the required data types.

Fuzzy Attributes Type 3

Type 3 attributes collect simple scalar data (SIMPLE) or possibility distributions (POSSIBILITY DISTRIBUTION) on this scalar domain. They also accept data of the types UNKNOWN, UNDEFINED, and NULL.

As in fuzzy attributes Type 2, attributes of this type need to store in the database the type of the value stored and the data of this value. The FMB also contains the attributes that are of this type, along with the "similarity relations" ("proximity relations" or "relations of resemblance") defined on the underlying domain (the scalars).

In Table 5.2, we show the system used to represent the fuzzy attributes Type 3. We can see that a fuzzy attribute Type 3, called **F**, for example, comprises a variable number of classical attributes:

Table 5.2. Internal representation for fuzzy attributes Type 3 or 4 (for an attribute **F***)*

Type of values	**Attributes in the DB for each fuzzy attribute Type 3**					
	FT	**FP1**	**F1**	**...**	**FPn**	**Fn**
UNKNOWN	0	NULL	NULL	...	NULL	NULL
UNDEFINED	1	NULL	NULL	...	NULL	NULL
NULL	2	NULL	NULL	...	NULL	NULL
SIMPLE	3	P	d	...	NULL	NULL
POSSIBILITY DISTRIBUTION	4	p_1	d_1	...	p_n	d_n

- **FT**: The *type* of value that corresponds to the data that we want to store. This could be UNKNOWN (0), UNDEFINED (1), NULL (2), SIMPLE (3), and POSSIBILITY DISTRIBUTION (4).
- **FT = 3**: In this case, the representation needs only two more attributes, called **FP1** and **F1,** where **F1** stores the label identifier, and **FP1** stores the possibility degree of this label. In a SIMPLE value, only the first **FP1** is used, which should be 1 in order to be normalized (Definition 1.11).
- **FT = 4**: In this case, a list of *n* couples is needed, with n > 1, with the format (possibility value/label identifier). These attributes store the next possibility distribution (using the Equation 1.1 format): {**FP1**/**F1**, ..., **FPn**/**Fn**}.

All the possibility values **FP1**, …, **FPn**, should belong to interval [0, 1]. In a value of type POSSIBILITY DISTRIBUTION (**FT** = 4), it is able to store to *n* couples. It can use fewer than *n* couples leaving the remainder attributes to NULL, but it cannot store more than *n* couples. That is to say, the length of a possibility distribution is delimited to the length *n*, which is adopted in the initial definition of each fuzzy attribute Type 3 or 4.

As you will see, the FMB stores the labels, their similarity relation (symmetrical or nonsymmetrical), and the value of *n*.

Fuzzy Attributes Type 4

The representation of fuzzy attributes Type 4 is exactly the same as the representation of the fuzzy attributes Type 3 (refer to Table 5.2); the only difference is in the FMB. Apart from indicating that these are of Type 4 and the value of its particular *n* value (the maximum length of its possibility distributions), these attributes have no similarity relation between the labels. This

characteristic simplifies this type of attribute with respect to Type 3. On the other hand, Type 4 can also be considered as equal to Type 3, having the precaution of storing a similarity relation in which the value is 0 for any different couple of labels, and 1 if the labels are equal. That is to say, for any couple of different labels of a fuzzy attribute Type 4, its similarity degree must be 0.

Fuzzy attributes Type 3 can also be used to define attributes in which the similarity relation has certain special characteristics (for example, not complying with the symmetrical property).

Representation of Fuzzy Degrees

We also consider here different types of attributes, but in this section we consider only those that are represented simply by means of a fuzzy degree. The domain of this degree should be the interval [0, 1]. For the majority of applications, it is sufficient to admit two decimals of precision. Thus, in Oracle, for example, the type of this attribute could be `NUMBER(3,2)`.

In all these cases, the representation is then as simple as adding one attribute to the table. The following aspects must be controlled: names assigned to these attributes, the meaning (or significance) of those degrees, and whether they are associated to values of other attributes in the same table.

So that they may be processed uniformly, minimizing the number of tables in the FMB, the types of degrees that we include in this section are called fuzzy attributes Type 5, 6, 7, and 8, respectively.

Fuzzy Degree Associated to Each Value of an Attribute: Type 5

In fuzzy attributes Type 5, this degree simply adds one attribute to the table. This attribute can be assigned any name, but FIRST-2 proposes that it be called **DegreeF,** where **F** is the name of the associated attribute. FSQL language has the function `FDEGREE` in order to access to this degree (see Chapter VII). In addition, the associated attribute of each degree and the meaning of each degree must be stored in the FMB.

Fuzzy Degree Associated to Values of Some Attributes: Type 6

The Type 6 degree adds one attribute to the table. This attribute can be assigned any name, but FIRST-2 proposes that it be called **DegreeF,** where **F** is the acronym of all the associated attributes. FSQL language has the function `FDEGREE` in order to access this degree (see Chapter VII). If there is a conflict with the name, it can be changed. As you will see, the associated attributes and the meaning of each degree must be stored in the FMB.

Fuzzy Degree Associated to the Whole Tuple: Type 7

The Type 7 degree also adds one attribute to the table. This attribute can be assigned any name, but FIRST-2 proposes that it be called **DegreeF,** where **F** is the table name. Nevertheless, FSQL language has the function `FDEGROW` in order to access this degree (see Chapter VII). The FMB stores the attributes of this type that exist, along with the meaning of each of them.

Fuzzy Degree With Its Own Meaning: Type 8

The Type 8 degree also adds one attribute to the table. This attribute can be given any name, and the user must write the name. The user should choose a good name that indicates something about its meaning. Moreover, this degree can have a standard meaning (or significance) stored in the FMB (see the section "Table FUZZY_DEGREE_SIG (FDS)," later in this chapter). This kind of degree does not have a default meaning because the meaning should be expressed in the name. The FMB stores the attributes of this type that exist, along with their associated standard meaning (if they have one).

FMB (Fuzzy Metaknowledge Base): Definition of Tables

As you have seen in the previous sections, certain types of information about the attributes need to be stored in an accessible form by the system or by the users. The Fuzzy Metaknowledge Base organizes all the information related to the imprecise or vague nature of these attributes.

The elements of the fuzzy processing stored in the FMB are as follows:

1. **Attributes with fuzzy processing**: Attributes of the database that receive fuzzy processing and the type of these attributes (from Type 1 to 8).
2. **Information about these attributes**: Depending on its type, different information is stored for each attribute:
   - Types 1 and 2:
     - Linguistic labels: name and definition (trapezoidal fuzzy set) for each one.
     - The *margin* value for approximate values.
     - The *much* value *M*: minimum distance *M* to consider two values as "*very*" separated (called "much"). This last value is used in comparisons such as "much greater than" (`MGT/NMGT`) and "much less than" (`MLT/NMLT`).
   - Types 3 and 4:
     - Maximum length for possibility distributions in values of these types (value *n* used in the fuzzy attributes Type 3 and Type 4 sections).
     - Linguistic labels: name.
     - Similarity relations between these labels (only for fuzzy attributes Type 3).
   - Types 5 and 6:
     - Degree significance (or meaning).
     - Attribute or attributes to which the degree is associated (in the Type 5 case, there is only one attribute).

- Types 7 and 8:
  - Degree significance (or meaning), which is optional in the Type 8 (fuzzy degree with its own meaning).

3. **Other objects or information**:
   - Fuzzy qualifiers (associated to any fuzzy attribute) are used to set a linguistic threshold in queries or to make comparisons with degree attributes.
   - Fuzzy quantifiers (associated to an attribute, to a table, or to the system). Fuzzy quantifiers (refer to Chapter I) are used in
     - Fuzzy queries (see Chapter VII): For example, "Give me employees who belong to *most* projects."
     - Fuzzy constraints (see "Fuzzy Constraints" section in Chapter IV): For example, "An employee must work in *many* projects."
   - Compatible columns: Those columns or attributes that may be compared are compatible. Thus, the definition of all values in the FMB must be done for one of them. For a Type 3 attribute, the user defines its labels and the similarity relation in the FMB. The other attributes that use these labels are marked as compatible to the first one.

It is important to emphasize that in order to compare two attributes Type 3 or two attributes Type 4, they should have the same domain, and therefore they must be marked as compatible. For the Type 1 or 2, they do not need to be compatible, because each attribute has its own labels, and the possibility distribution of these labels is what is compared (the definition that both labels have in the FMB). Thus, for example, we will be able to compare two fuzzy attributes Type 2, such as "height" and "weight," although this comparison does not make sense and is meaningless. Marking two attributes Type 1 or 2 as compatible avoids the creation of labels in one of them. In the Type 3 and 4, marking the attributes as compatible is obligatory, because (for the comparison) the attributes to be compared must have the same labels and the same characteristics.

Subsequently, we detail how FIRST-2 implements the FMB, its basic nucleus. This is done by means of the tables defined in the next section. FIRST-2 also defines some views that facilitate access to certain data (see the "Useful Views on the FMB" section later in this chapter). A summary of these objects is shown in Table 5.3.

Table 5.3. Tables and views of FIRST-2 and its synonyms

N.	Table/View	Utility	Synonym
1.	T. FUZZY_COL_LIST	List of fuzzy columns (or attributes).	FCL
2.	T. FUZZY_DEGREE_SIG	Fuzzy degrees significances (or meanings).	FDS
3.	T. FUZZY_OBJECT_LIST	List of fuzzy objects of columns.	FOL
4.	T. FUZZY_LABEL_DEF	Trapezoidal definitions (for labels).	FLD
5.	T. FUZZY_APPROX_MUCH	*Margin* and *M* values (Types 1 and 2).	FAM
6.	T. FUZZY_NEARNESS_DEF	Similarity relations (Type 3).	FND
7.	T. FUZZY_COMPATIBLE_COL	Compatible fuzzy attributes.	FCC
8.	T. FUZZY_QUALIFIERS_DEF	Qualifiers definition.	FQD
9.	T. FUZZY_DEGREE_COLS	Columns with an associated fuzzy degree.	FDC
10.	T. FUZZY_DEGREE_TABLE	Information about fuzzy degrees of tables.	FDT
11.	T. FUZZY_TABLE_QUANTIFIERS	Fuzzy quantifiers associated to tables.	FTQ
12.	T. FUZZY_SYSTEM_QUANTIFIERS	Fuzzy quantifiers associated to the system.	FSQ
13.	V. LABELS_FOR_OBJCOL	Trapezoidal labels for each attribute.	LFOC
14.	V. LABELS _OBJCOL_T3	Labels for Type 3 and 4.	LOCT3
15.	V. ALL_COMPATIBLES_T34	Compatible attributes Type 3 and 4.	ACT34

These tables and views are a general basic structure. A good structure should consist of sets of views with different information about all the accessible objects. For example, Oracle organizes its data dictionary into three kinds of views containing similar information and distinguished from each other by their prefixes: Prefix USER_ is adopted in views containing user objects (what is in the user's schema), views with prefix ALL_ contain accessible objects (what the user can access), and views with prefix DBA_ are database administrators' views (what is in all the users' schemas).

Relations in the FMB

Figure 5.1 shows the FMB relations (or tables), their attributes, their primary keys (underlined), and their foreign keys (with arrows).

We use OBJ# as the relation identifier, and COL# as the column or attribute identifier (as does Oracle). In a relational database, each attribute is assigned univocally to a couple of data (OBJ#, COL#), where OBJ# is the indicator of a table and COL# is the indicator of the column or concrete attribute in that table. From now on, we shall refer to OBJ# as the indicator of a table, although this can also be the indicator of a view (if we have one or several fuzzy tables, we will be able to define a fuzzy view on them).

For each table/view of the FMB, a public synonymous with the acronyms of the table/view is created in order to facilitate access, because the names of these

objects are long. Table 5.3 shows the set of tables and views of the FMB, its utility, and its synonym.

Subsequent sections describe how each table is created in FIRST-2, its structure, its meaning, and the meaning of each of its attributes. The primary key of each table is underlined in Figure 5.1.

Table FUZZY_COL_LIST (FCL)

The FCL table contains a description of those attributes in the database for fuzzy processing or treatment. The columns of this table and their meanings are as follows:

- OBJ#: This column stores the object number of the table that has a fuzzy attribute.
- COL#: This column stores the column number that admits a fuzzy processing in the table OBJ#. In fuzzy attributes Type 2, 3, or 4, this attribute stores the column number of the type attribute (**FT** in Tables 5.1 and 5.2).
- F_TYPE: This column stores the type of fuzzy attribute of the column identified by (OBJ#, COL#). This type can be an integer value between 1 and 8.
- LEN: This column stores information when the type of the fuzzy attribute (column F_TYPE) is Type 3 and 4. For these attributes, the column stores the maximum length *n* of a possibility distribution, that is, the maximum number of couples (possibility value / label) that admits a possibility distribution in this fuzzy attribute Type 3 or 4.
- CODE_SIG: This column stores information when the type of the attribute (column F_TYPE) is Type 5, 6, and 7. This column is a numerical value and it expresses the meaning of this degree, which must exist in table FUZZY_DEGREE_SIG. This value is optional for Type 8.
- COLUMN_NAME: For the sake of convenience, this field is useful for putting the alphanumeric name (without codes) of the attribute (OBJ#, COL#), using, for example, the format <Owner>.<Table>.<Column>.
- COM: This column stores an optional comment about each attribute.
- UM: This column stores the unit of measurement of the attribute optionally.

Table FUZZY_DEGREE_SIG (FDS)

This table contains a list with the meanings of the degrees used in the previous table:

- CODE_SIG: This stores the numerical code of the meaning.
- SIGNIFICANCE: This stores the alphanumeric meaning or significance in the database's own language. As you see in Chapter IV, these meanings can vary (fulfillment degree, uncertainty degree, possibility degree, importance degree, etc.).

Table FUZZY_OBJECT_LIST (FOL)

This table contains a list of the fuzzy objects that are defined in the columns of the database. The attributes of this table have the following meanings:

- (OBJ#, COL#): This stores the attribute identifier to which the object belongs.
- FUZZY_ID: The identifier of the fuzzy object.
- FUZZY_NAME: The name of the object without spaces.
- FUZZY_TYPE: The type of the object. It may be one of the following codes, and each code has an associated object. Fuzzy quantifiers codes begin at 10:

  0 For *scalars* of fuzzy attributes Type 3, subject to processing by means of a symmetrical similarity relation defined in table FUZZY_NEARNESS_DEF (with a nonordered underlying domain).

  1 For *scalars* of fuzzy attributes Type 3, subject to processing by means of a nonsymmetrical similarity relation (proximity relation) defined in table FUZZY_NEARNESS_DEF (with a nonordered underlying domain).

  2 For *scalars* of fuzzy attributes Type 4, subject to processing without similarity relation (with a nonordered underlying domain).

3 For *qualifiers* defined over the fulfillment degree in the query (threshold) and stored in table FUZZY_QUALIFIERS_DEF.

4 For linguistic *labels* of fuzzy attributes Type 1 or 2, which are of the trapezoidal type defined in table FUZZY_LABEL_DEF (with an ordered underlying domain).

10 Linguistic labels defined on *absolute quantifiers* (without arguments) and stored in the table FUZZY_LABEL_DEF with $\alpha, \beta, \gamma, \delta \geq 0$. See Example 1.6.

11 Linguistic labels defined on *relative quantifiers* (without arguments) and stored in the table FUZZY_LABEL_DEF with $\alpha, \beta, \gamma, \delta \in [0, 1]$. See Example 1.6.

12 Linguistic labels defined on *absolute quantifiers* with one *argument x*. They are also stored in the table FUZZY_LABEL_DEF with the values α, β, γ, and δ (which can be negative), so that the final quantifier is understood to be defined by adding (or reducing) the argument x to each value: $[\alpha + x, \beta + x, \gamma + x, \delta + x]$. See Example 1.7.

13 Linguistic labels defined on *absolute quantifiers* with one *argument x*. They are also stored in the table FUZZY_LABEL_DEF with α, β, γ, and δ usually in the interval [0, 1], so that the final quantifier is understood to be defined by multiplying each value by the argument x: $[\alpha * x, \beta * x, \gamma * x, \delta * x]$. See Example 1.7.

14 Linguistic labels defined on *relative quantifiers* with one *argument x*. They are also stored in the table FUZZY_LABEL_DEF with the values α, β, γ, and δ (which can be negative), so that the final quantifier is understood to be defined by adding (or reducing) the argument x to each value: $[\alpha + x, \beta + x, \gamma + x, \delta + x]$. See Example 1.7.

15 Linguistic labels defined on *relative quantifiers* with one *argument x*. They are also stored in the table FUZZY_LABEL_DEF with α, β, γ, and δ usually in the interval [0, 1], so that the final quantifier is understood to be defined by multiplying each value by the argument x: $[\alpha * x, \beta * x, \gamma * x, \delta * x]$. See Example 1.7.

16 Linguistic labels defined on *absolute quantifiers* with two *arguments x* and *y*. They are also stored in the table FUZZY_LABEL_DEF with the values α, β, γ, and δ (which can be negative), so that the final

quantifier is understood to be defined by $[\alpha+x, \beta+x, \gamma+x, \delta+x]$. See Example 1.7.

17 Linguistic labels defined on *absolute quantifiers* with two *arguments* x and y. They are also stored in the table FUZZY_LABEL_DEF with α, β, γ, and δ usually in the interval [0, 1], so that the final quantifier is understood to be defined by $[\alpha * x, \beta * x, \gamma * x, \delta * x]$. See Example 1.7.

18 Linguistic labels defined on *relative quantifiers* with two *arguments* x and y. They are also stored in the table FUZZY_LABEL_DEF with the values α, β, γ, and δ (which can be negative), so that the final quantifier is understood to be defined by $[\alpha+x, \beta+x, \gamma+x, \delta+x]$. See Example 1.7.

19 Linguistic labels defined on *relative quantifiers* with two *arguments* x and y. They are also stored in the table FUZZY_LABEL_DEF with α, β, γ, and δ usually in the interval [0, 1], so that the final quantifier is understood to be defined by $[\alpha * x, \beta * x, \gamma * x, \delta * x]$. See Example 1.7.

For more information about absolute and relative quantifiers (with or without arguments), refer to Chapter I. The FUZZY_TYPE for fuzzy quantifiers is shown in Table 5.4. Some examples are shown in Tables 5.5 and 5.6.

Table 5.4. Fuzzy quantifiers types

FUZZY_TYPE	Num. of Arguments	Absolute Relative	Building Type*	Final Quantifier
10	0	A	-	$[\alpha, \beta, \gamma, \delta]$
11	0	R	-	$[\alpha, \beta, \gamma, \delta]$
12	1	A	Sum	$[\alpha+x, \beta+x, \gamma+x, \delta+x]$
13	1	A	Product	$[\alpha*x, \beta*x, \gamma*x, \delta*x]$
14	1	R	Sum	$[\alpha+x, \beta+x, \gamma+x, \delta+x]$
15	1	R	Product	$[\alpha*x, \beta*x, \gamma*x, \delta*x]$
16	2	A	Sum	$[\alpha+x, \beta+x, \gamma+y, \delta+y]$
17	2	A	Product	$[\alpha*x, \beta*x, \gamma*y, \delta*y]$
18	2	R	Sum	$[\alpha+x, \beta+x, \gamma+y, \delta+y]$
19	2	R	Product	$[\alpha*x, \beta*x, \gamma*y, \delta*y]$

** Building Type sets how to obtain the final quantifiers starting from the values* $[\alpha, \beta, \gamma, \delta]$ *and the arguments x and y.*

Table FUZZY_LABEL_DEF (FLD)

This table contains the points that define the trapezoidal possibility distribution associated to the types of objects from 4 to 19 (trapezoidal labels and fuzzy quantifiers) of the attribute FUZZY_TYPE of the table FUZZY_OBJECT_LIST. The fields of this table are as follows:

- (OBJ#, COL#, FUZZY_ID): These three fields are the primary key of this table and are the foreign key to the table FUZZY_OBJECT_LIST.
- ALFA, BETA, GAMMA, and DELTA: These fields define a trapezoidal possibility distribution (see Figure 7.1).

Although FIRST-2 does not allow us to set labels such as "extended trapezoid function" (refer to Chapter I), the extension for including this type of membership function is easy. Furthermore, this table would need more attributes in order to store a list of points `P1/N1`, ... `Pn/Nn`, where all the `Pi` belong to [0, 1], and all the `Ni` are between α and β or between γ and δ.

Table FUZZY_APPROX_MUCH (FAM)

This table stores data that are useful for fuzzy attributes Type 1 or 2. Its columns have the following meaning:

- (OBJ#, COL#): These columns store the attribute identifier.
- MARGEN (*margin*, in Spanish): This is the *margin* used in the triangular labels of the approximate values.
- MUCH: This is the value *M* that indicates the minimum distance to consider two values of this attribute to be very separated.

This table should have a constraint that forces the value MARGEN to be less than the MUCH value for obvious and semantic reasons.

Table FUZZY_NEARNESS_DEF (FND)

This table presents the measures of resemblance, similarity, or proximity between the different values in the underlying domain of fuzzy attributes Type 3. Its columns are as follows:

- (OBJ#, COL#): These columns store the identifier of the attribute Type 3, which possesses the similarity relation.
- FUZZY_ID1: The identifier of an object label. The three attributes (OBJ#, COL#, FUZZY_ID1) form a foreign key that should exist in the table FUZZY_OBJECT_LIST, and in this table, column FUZZY_TYPE should have the value 0 or 1.
- FUZZY_ID2: The identifier of another object label. The three attributes (OBJ#, COL#, FUZZY_ID2) form a foreign key that should exist in the table FUZZY_OBJECT_LIST, and in this table, column FUZZY_TYPE should have the value 0 or 1.
- DEGREE: The similarity, proximity, or resemblance degree between the labels indicated by the two previous attributes (FUZZY_ID1 and FUZZY_ID2). This value should belong to the interval [0, 1].

Table FUZZY_COMPATIBLE_COL (FCC)

The FCC table indicates the fuzzy attributes that are compatible with others (mainly Type 3 or 4). In this way, it is unnecessary to define the labels (for Type 1, 2, 3, or 4) and the similarity relations (if they are Type 3) for each one of them. Its columns have the following meanings:

- (OBJ#1, COL#1): These columns store the identifier of the fuzzy attribute that is compatible with another. It indicates that this attribute does not own labels and will take the labels defined by the following attribute.
- (OBJ#2, COL#2): These columns store the identifier of an attribute that has labels (and a similarity relations if it is Type 3) defined on it, and they will be adopted for the previous attribute.

Table FUZZY_QUALIFIERS_DEF (FQD)

Definition of qualifiers defined over the fulfillment degree in a query (threshold) and that they will be able to be used in simple comparisons:

- (OBJ#, COL#, FUZZY_ID): These three fields are the primary key and foreign key to the table FUZZY_OBJECT_LIST and have identical meanings.
- QUALIFIER: The qualifier is a number in the interval [0, 1].

Table FUZZY_DEGREE_COLS (FDC)

Declaration of the columns associated to a fuzzy degree. The number of associated columns to a degree of Type 5 is one. In addition, a degree of Type 6 can be associated to any quantity of columns. The columns of this table are as follows:

- (OBJ#1, COL#1): These columns store the identifier of the attribute Type 5 or 6. These attributes are the foreign key of the table FUZZY_COL_LIST.
- (OBJ#2, COL#2): These two fields specify a column or attribute to which the fuzzy degree, defined by the two previous attributes, is associated.

It should be noted that the primary key of this table includes the four attributes, because a degree can be associated to many attributes (the case of the Type 6) and, additionally, an attribute can have many associated degrees to it (of Type 5 or 6 indistinctly). Of course, the degrees of Type 7 and 8 do not use this table.

Table FUZZY_DEGREE_TABLE (FDT)

This table stores information about the fuzzy tables, that is, tables with one or more degree attributes (i.e., Type 7 degrees). Fuzzy tables come from fuzzy entities or relationships (Definitions 4.8, 4.9, and 4.10). The columns of this table are as follows:

- OBJ#: This column stores the identifier of the table that represents the fuzzy entity or relationship.
- COL#: This column stores the identifier of the degree column, which must be represented in table FUZZY_COL_LIST with F_TYPE = 7. The meaning of the degree is expressed in that table.
- DEGREE_TYPE: The type of degree that can only be used at an *informative level* because it does not have an influence on the system. It can be of the following types:
  - "C" for degrees of fuzzy entities calculated automatically
  - "M" for degrees of fuzzy entities introduced manually
  - "E" for fuzzy weak entities due to dependency on existence
  - "I" for fuzzy weak entities due to dependency on identification
  - "R" when table OBJ# represents a fuzzy relationship and the degrees are calculated automatically
  - "S" when table OBJ# represents a fuzzy relationship and the degrees are introduced manually

As the definitions of fuzzy entities (Definition 4.9) and fuzzy relationships (Definition 4.11) express, the degree can be calculated automatically with a predefined function. The fuzzy weak entities due to dependency on identification are treated as the normal weak entities, bearing in mind that the primary key of the owner entity must be included.

Table FUZZY_TABLE_QUANTIFIERS (FTQ)

Definition of quantifiers associated to a relation or table (not to an attribute). These quantifiers are used in fuzzy constraints (refer to Chapter IV) and fuzzy queries (see Chapter VII). The columns of this table are as follows:

- OBJ#: This column stores the identifier of the table to which the quantifier is associated.
- FUZZY_NAME: The name of the quantifier without spaces.
- FUZZY_TYPE: The type of quantifier. This attribute uses the same codes as the table FUZZY_OBJECT_LIST for quantifiers (see Table 5.4).

- ALFA, BETA, GAMMA, and DELTA: These columns define the trapezoidal fuzzy quantifier.

Note that the primary key of this table is (OBJ#, FUZZY_NAME), which indicates that one table cannot have two quantifiers with the same name, but the same name can be used in different tables, and, of course, with possibly different definitions.

Table FUZZY_SYSTEM_QUANTIFIERS (FSQ)

Definition of quantifiers associated to the system (neither an attribute nor a table). These quantifiers are used in fuzzy constraints (refer to Chapter IV) and fuzzy queries (see Chapter VII). The columns of this table are as follows:

- FUZZY_NAME: The name of the quantifier without spaces.
- FUZZY_TYPE: The type of quantifier. This attribute uses the same codes as the table FUZZY_TABLE_QUANTIFIERS: See the preceding section and Table 5.4.
- ALFA, BETA, GAMMA, and DELTA: These columns define the trapezoidal fuzzy quantifier.

Note that the primary key of this table is (FUZZY_NAME), which indicates that each system quantifier has a unique name. Table 5.5 shows an example of FUZZY_SYSTEM_QUANTIFIERS with one quantifier of each type.

Some quantifiers are very dependent on the context, so they are not good system quantifiers. System quantifiers should either be relative or absolute with one or two arguments and type product (types 11, 13, 14, 15, 17, 18, and 19), because these types are not very dependent on the context (particularly the relative quantifiers).

This table should store some default values, and according to the previous information, Table 5.6 proposes some interesting system quantifiers. Furthermore, two quantifiers ($\exists$ and $\forall$) must be implemented directly in the system: **`Exists`** and **`For_all`** (or **`All`**), with the Equations 1.65 and 1.66, respectively.

Table 5.5. An example of table FSQ with one quantifier of each type

FUZZY_NAME	FUZZY_TYPE	ALFA	BETA	GAMMA	DELTA	Final Quantifier
Approx_8	10	6	8	8	10	[6, 8, 8, 10]
Almost_All	11	0.4	0.9	1	1	[0.4, 0.9, 1, 1]
Much_Greater_Than_x	12	1	9	MAX*	MAX*	[1+x, 9+x, MAX+x, MAX+x]
About_Half_of_x	13	0.25	0.5	0.5	0.75	[0.25x, 0.5x, 0.5x, 0.75x]
Approx_xth_part	14	-0.2	0	0	0.2	[x-0.2, x, x, x+0.2]
Less_Than_xth_part	15	0	0	1	1.25	[0, 0, x, 1.25x]
Approx_Between_x_and_y	16	-5	0	0	5	[x-5, x, y, y+5]
Approx_Between_ Half_x_and_Half_y	17	0.25	0.5	0.5	0.75	[0.25x, 0.5x, 0.5y, 0.75y]
Approx_Between_ xth_and_yth_part	18	-0.1	0	0	0.1	[x-0.1, x, y, y+0.1]
Approx_Between_Half_ xth_and_Half_yth_part	19	0.4	0.5	0.5	0.6	[0.4x, 0.5x, 0.5y, 0.6y]

* *Maximum value in the underlying domain (*MAX≡∞*).*

Table 5.6. An example of table FSQ with interesting system quantifiers (this table may be used as default table FSQ)

FUZZY_NAME	FUZZY_TYPE	ALFA	BETA	GAMMA	DELTA	Final Quantifier
Fuzzy_Exists	10	0	1	MAX	MAX	[0, 1, ∞, ∞]
Most	11	0.4	0.9	1	1	[0.4, 0.9, 1, 1]
Almost_All	11	0.4	0.9	1	1	[0.4, 0.9, 1, 1]
About_Half	11	0.25	0.5	0.5	0.75	[0.25, 0.5, 0.5, 0.75]
Minority	11	0	0	0.1	0.6	[0, 0, 0.1, 0.6]
About_Half_x	13	0.25	0.5	0.5	0.75	[0.25x, 0.5x, 0.5x, 0.75x]
Approx_x	13	0.9	1	1	1.1	[0.9x, x, x, 1.1x]
Twice_x (or Double_of_x)	13	1.75	2	2	2.25	[1.75x, 2x, 2x, 2.25x]
Approx_xth_part	14	-0.2	0	0	0.2	[x-0.2, x, x, x+0.2]
Less_Than_xth_part	15	0	0	1	1.25	[0, 0, x, 1.25x]
More_Than_xth_part	15	0.75	1	100	100	[0.75x, x, 100x, 100x]
Approx_Between_ Half_x_and_Half_y	17	0.25	0.5	0.5	0.75	[0.25x, 0.5x, 0.5y, 0.75y]
Approx_Between_x_and_y	17	0.75	1	1	1.25	[0.75x, x, y, 1.25y]
Approx_Between_ Twice_x_and_Twice_y	17	1.75	2	2	2.25	[1.75x, 2x, 2y, 2.25y]
Approx_Between_ xth_and_yth_part	18	-0.1	0	0	0.1	[x-0.1, x, y, y+0.1]

Useful Views on the FMB

These views simplify access to certain data. We present some views here, although more, of course, can be defined.

View LABELS_FOR_OBJCOL (LFOC)

The view LABELS_FOR_OBJCOL (LFOC) serves to easily obtain or query the defined labels on fuzzy attributes Type 1 or 2 and the parameters of the associated trapezoidal fuzzy set. The statement that creates this view is the following:

```
CREATE or replace view LABELS_FOR_OBJCOL AS
   SELECT  FOL.OBJ# OBJ#, FOL.COL# COL#, COLUMN_NAME,
           Fuzzy_Name LABEL, ALFA, BETA, GAMMA, DELTA
   FROM   FOL, FLD, FCL
   WHERE FOL.OBJ# = FLD.OBJ# AND FOL.COL# = FLD.COL#
     AND FCL.OBJ# = FOL.OBJ# AND FCL.COL# = FOL.COL#
     AND FLD.FUZZY_ID = FOL.FUZZY_ID
     AND FOL.FUZZY_TYPE = 4;
```

The columns of this view have the following meanings:

- (OBJ#, COL#): These columns store the identifier of the attribute Type 1 or 2 that owns the label.
- COLUMN_NAME: The name of the attribute.
- LABEL: The name of the linguistic label.
- (ALFA, BETA, GAMMA, DELTA): These columns define the trapezoidal label.

View LABELS_OBJCOL_T3 (LOCT3)

The view LABELS_OBJCOL_T3 (LOCT3) serves to obtain or to easily query the defined labels on fuzzy attributes Type 3 as well as the similarity relation among them.

```
CREATE or replace view LABELS_OBJCOL_T3 AS
     SELECT FND.OBJ# OBJ#, FND.COL# COL#, FUZZY_TYPE,
            FOL1.FUZZY_NAME  LABEL_1,
            FOL2.FUZZY_NAME  LABEL_2,
            DEGREE
     FROM   FOL FOL1, FOL FOL2, FND
     WHERE  FOL1.OBJ# = FOL2.OBJ# AND FOL1.COL# = FOL2.COL#
        AND FOL1.OBJ# = FND.OBJ#  AND FOL1.COL# = FND.COL#
        AND FOL1.FUZZY_TYPE = FOL2.FUZZY_TYPE
        AND FOL1.FUZZY_TYPE IN {0,1}
        AND FND.FUZZY_ID1 = FOL1.FUZZY_ID
        AND FND.FUZZY_ID2 = FOL2.FUZZY_ID;
```

The columns of this view have the following meanings:

- (OBJ#, COL#): These columns store the identifier of the attribute Type 3 that owns the labels and the similarity relation between them.
- FUZZY_TYPE: This column takes 0 for symmetrical similarity relations and 1 for nonsymmetrical similarity relations (proximity relations).
- LABEL_1: The name of a linguistic label defined in the previous fuzzy attribute.
- LABEL_2: The name of another linguistic label.
- DEGREE: The similarity degree between LABEL_1 and LABEL_2.

View ALL_COMPATIBLES_T34 (ACT34)

The ALL_COMPATIBLES_T34 (ACT34) view serves to easily query the fuzzy attributes Type 3 or 4, which have been defined as being compatible to

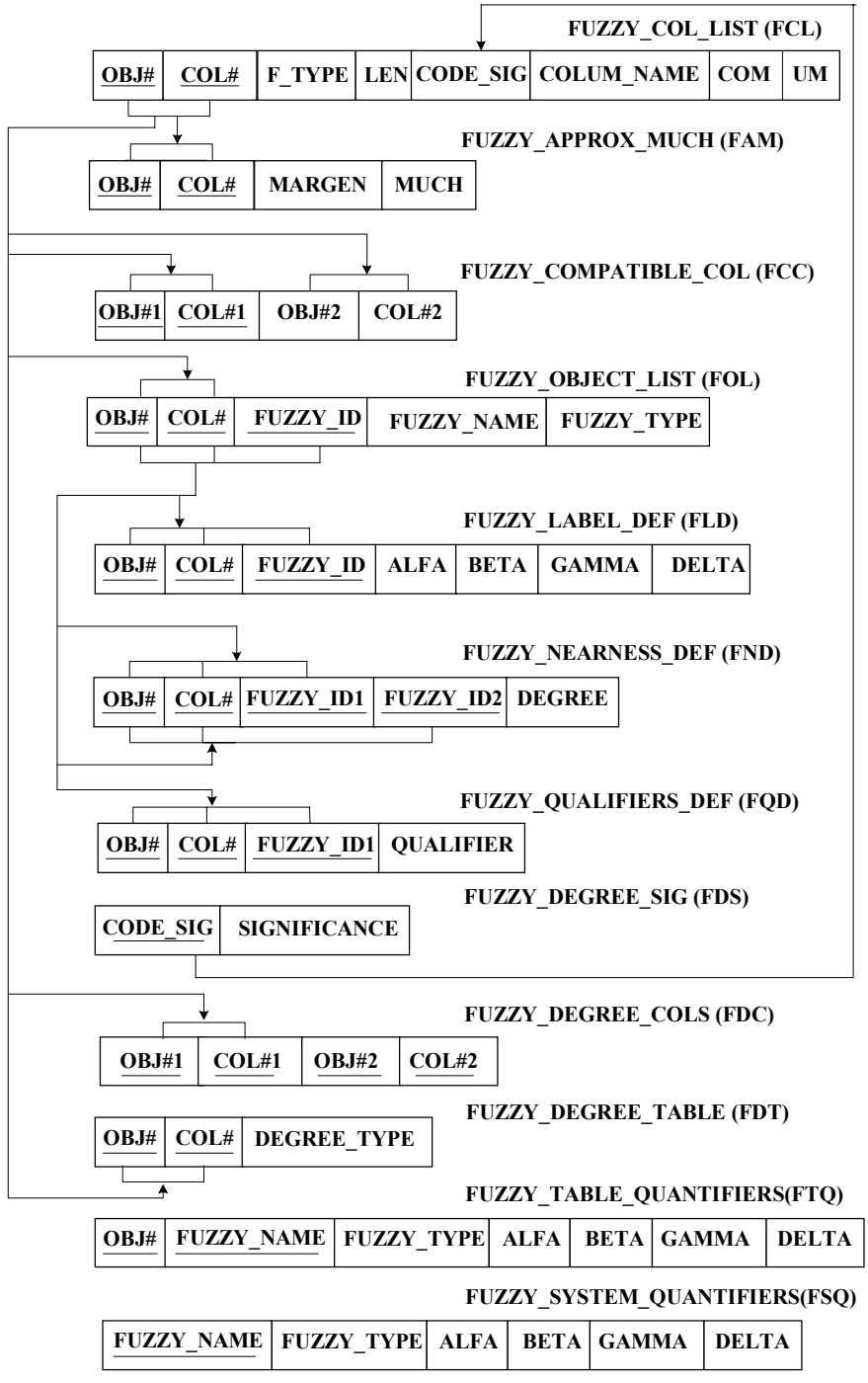
one another. In this way, we can discover which fuzzy attributes (Type 3 or 4) do not have defined labels on them and which attributes own the labels. In addition, the maximum length for the possibility distributions is shown.

```
CREATE or replace view ALL_COMPATIBLES_T34 AS
     SELECT distinct
            FCL1.COLUMN_NAME COLUMN_1,
            FCL1.LEN         LENGTH_1,
             FCL2.COLUMN_NAME COMPATIBLE_WITH,
            FCL2.LEN         LENGTH_2,
            FCL1.F_TYPE
     FROM  FCL FCL1, FCL FCL2, FCC
     WHERE FCL1.OBJ# = FCC.OBJ#1 AND FCL2.OBJ# = FCC.OBJ#2
       AND FCL1.COL# = FCC.COL#1 AND FCL2.COL# = FCC.COL#2
       AND FCL1.F_TYPE IN {3,4};
```

The columns of this view have the following meanings:

- COLUMN_1: The name of the column that is compatible with another. This indicates that this attribute does not own labels and will take the labels defined by the attribute COMPATIBLE_WITH.
- LENGTH_1: The maximum number of couples (possibility value, label) that admits a possibility distribution in fuzzy attribute COLUMN_1.
- COMPATIBLE_WITH: The name of the column that has objects defined on it and that will be adopted for the previous attribute COLUMN_1.
- LENGTH_2: The maximum number of elements in possibility distributions of the previous fuzzy attribute.
- F_TYPE: This column stores the fuzzy data type of the previous two attributes. In this view, this column can be an integer value 3 or 4.

Figure 5.1. FMB scheme of FIRST-2



Chapter VI

Mapping FuzzyEER Model Concepts to Relations

This chapter shows the transformation of the FuzzyEER model to a logical design by using relational databases. The FuzzyEER-to-Relational mapping algorithm is based on the "classical" EER-to-Relational mapping algorithm, published in Elmasri and Navathe (2000) and summarized in the first section of this chapter, but other versions are very similar (De Miguel, Piattini, & Marcos, 1999; Silverschatz, Korth, & Sudarshan, 2002). The FuzzyEER-to-Relational mapping algorithm includes additional rules for mapping fuzzy concepts.

The following sections translate the FuzzyEER concepts, that is, the definitions in Chapter IV, to the FIRST-2 schema, which was exposed in Chapter V.

Thus, this chapter relates Chapter IV with Chapter V, obtaining a fuzzy relational database. In addition, we need a comprehensive fuzzy database language with statements for data definition, query, and update. This language is **FSQL** (Fuzzy SQL), and we describe it in Chapter VII.

It should be noted that some definitions in Chapter IV define fuzzy degrees to the model (see the "Zvieli and Chen Approach" section in Chapter III and the "Fuzzy Degree to the Model" section in Chapter IV). Of course, these degrees are not mapped to the relational database. As such, Definitions 4.7, 4.8, and 4.12 are not treated in this chapter.

EER-to-Relational Mapping Algorithm

Basically an EER schema may be mapped into the corresponding relational database schema in 10 steps. The first seven steps are related to ER models, and the last steps are related to the superclass/subclass relationships in EER models.

STEP 1: For each regular (strong) entity type E, create a relation R that includes all the simple attributes or the simple component attributes of composite attributes of E. One or some attributes must be the primary key of E.

STEP 2: For each weak entity type W with owner entity type E, create a relation R, including the attributes as in Step 1. Furthermore, R must include a foreign key to the relation of E. The primary key of R is the combination of the primary key of the owner (relation of E) and the partial key of W, if any. It is common to choose the propagate option for the referential triggered action, that is, the SQL clauses `ON UPDATE CASCADE` and `ON DELETE CASCADE` in the foreign key, because a weak entity has an existence dependency on its owner entity.

STEP 3: For each binary 1:1 relationship type, choose one of the participating entities and include the primary key of the other one as foreign key of one of the entities' relations. It is better to choose an entity type with total participation in the relationship. Include all the simple or composite attributes in the relation of the chosen entity. An alternative mapping of a 1:1 relationship type is possible by merging the two entity types and the relationships into a single relation. This is better when both participations are total.

STEP 4: For each regular binary 1:N relationship, identify the relation S that represents the participating entity type at the N-side of the relationship type. Include as foreign key in the S relation the primary key of the other relation. Include all the simple or composite attributes in the relation of S.

STEP 5: For each binary M:N relationship type, create a new relation R. Include as foreign key in R the primary keys of the relations that represent the participating entity types; their combination will form the primary key of R. Include in R all the simple or composite attributes in the relationship. The

propagate (CASCADE) option for the referential triggered action should be specified on the foreign keys in the relation corresponding to the relationship, because each relationship instance has an existence dependency on each of the entities it relates.

It should be noted that we can always map 1:1 or 1:N relationships in a manner similar to M:N relationships, and this quality is particularly useful when few relationships instances exist, in order to avoid null values in foreign keys. In this case, the primary key will be only one of the foreign keys that reference the participating relations.

STEP 6: For each multivalued attribute A, create a new relation. This relation includes the attribute A, plus a foreign key K, to the primary key of the relation that represents the entity type or relationship type that has A as an attribute. The primary key is A and K. The propagate (CASCADE) option for the referential triggered action should be specified on the foreign key in the relation corresponding to the multivalued attribute for both ON UPDATE and ON DELETE.

STEP 7: For each n-ary relationship type, where n > 2, create a new relation. Include as foreign key the primary keys of the relations that represent the participating entity types; their combination will usually form the primary key. If the cardinality constraints of any of the participating entity types is 1, then the primary key should not include the corresponding foreign keys. Include all the simple or composite attributes in the relationship. The propagate (CASCADE) option for the referential triggered action should be specified on the foreign keys.

Derived attributes and the (min, max) notation are translated into triggers because they do not have standard constraints. The trigger of derived attributes computes the value when some other values are modified or when we insert an instance with the derived attribute. Similarly, the trigger of a constraint with the (min, max) notation controls that the constraint be satisfied.

STEP 8: Convert each specialization with *m* subclasses $\{S_1, S_2, ..., S_m\}$ and superclass C with attributes $\{k, a_1, a_2, ..., a_n\}$, where k is the primary key, into relation schemas using one of the four options:

1. **Option 8A**: Create a relation for C with attributes $\{k, a_1, a_2, ..., a_n\}$ and create *m* relations with the attributes of $\{S_1, S_2, ..., S_m\}$, respectively. The primary key of all these relations is k, which must be added to the *m* relations.
2. **Option 8B**: Create *m* relations with the attributes of $\{S_1, S_2, ..., S_m\}$, respectively. In addition, these *m* relations must include the attributes $\{k, a_1, a_2, ..., a_n\}$, where k is the primary key for all of them.
3. **Option 8C**: Create a single relation with all the attributes and t: $\{k, a_1, a_2, ..., a_n\} \cup \{\text{attributes of } S_1\} \cup ... \cup \{\text{attributes of } S_m\} \cup \{t\}$. The primary key is k. This option is for a specialization whose subclasses are disjoint, and t is a type attribute that indicates the subclass to which each tuple belongs, if any.
4. **Option 8D**: Create a single relation with the attributes $\{k, a_1, a_2, ..., a_n\} \cup \{\text{attributes of } S_1\} \cup ... \cup \{\text{attributes of } S_m\} \cup \{t_1, t_2, ..., t_m\}$. The primary key is k. This option is for a specialization whose subclasses are (mainly) overlapping, and t_i with $i = 1, 2, ..., m$ are Boolean attributes indicating whether a tuple belongs to S_i.

STEP 9: A shared subclass (or intersection type) is a subclass of several superclasses, and all must have the same key. We can apply any of the options discussed in Step 8, although option 8A is usually used.

STEP 10: A category is a subclass of the union of two or more superclasses that can have different keys. To map a category, it is customary to specify a new key attribute, called a *surrogate key*, when creating a relation to correspond to the category. The category is mapped into a relation with its attributes and the surrogate key as the primary key. We also add the surrogate key as a foreign key to each relation corresponding to a superclass of the category. For a category whose superclasses have the same key, a surrogate key is unnecessary.

Fuzzy Values in Fuzzy Attributes

Definitions 4.1 and **4.2** (in Chapter IV) define fuzzy attributes in the FuzzyEER Model:

1. Fuzzy attributes Type 1 are represented as usual attributes (see the corresponding section in Chapter V), storing in the FMB the values expressed in the "FMB (Fuzzy Metaknowledge Base): Definition of Tables" section in Chapter V (linguistic labels, the *margin* value and the *much* value). This attribute type implies to update the following FMB tables: FCL, FOL, FLD, and FAM.
2. Fuzzy attributes Type 2 are represented as the corresponding section in Chapter V shows, storing in the FMB the values expressed in the "FMB (Fuzzy Metaknowledge Base): Definition of Tables" section in Chapter V (linguistic labels, the *margin* value and the *much* value). See Table 5.1. This attribute type implies to update the following FMB tables: FCL, FOL, FLD, and FAM.
3. Fuzzy attributes Type 3 are represented as the corresponding section in Chapter V shows, storing in the FMB the values expressed in the "FMB (Fuzzy Metaknowledge Base): Definition of Tables" section in Chapter V (length, linguistic labels, and the similarity relation). See Table 5.2. This attribute type implies to update the following FMB tables: FCL, FOL, and FND.
4. Fuzzy attributes Type 4 are represented as the corresponding section in Chapter V shows, storing in the FMB the values expressed in the "FMB (Fuzzy Metaknowledge Base): Definition of Tables" section in Chapter V (length and linguistic labels). See Table 5.2. This attribute type implies to update the following FMB tables: FCL and FOL.

According to this, primary key attributes, simple attributes, composite attributes, derived attributes, and multivalued attributes (fuzzy or crisp) are represented by using the classic techniques shown in the preceding section.

Instead of updating the tables FCL, FOL, FLD, AM, and FND, it is also possible to update the table FCC, but only if the attribute is compatible with other already existing attributes.

Fuzzy Degrees

Definition 4.3 defines fuzzy degrees associated with each value of an attribute. This degree is represented as a numeric attribute. This attribute type implies to

update the following FMB tables: FCL (inserting a Type 5 attribute) and FDC (inserting only one row with the associated attribute identifier). Optionally, if the degree has a special meaning this must be inserted in the FDS table. Derived fuzzy degrees (**Definition 4.4**) are implemented by using triggers.

Definition 4.5 defines fuzzy degrees associated with values of some attributes. This degree is also represented as a numeric attribute. This attribute type implies to update the following FMB tables: FCL (inserting a Type 6 attribute) and FDC (inserting as many rows as associated attributes). Optionally, if the degree has a special meaning, this must be inserted in the FDS table. Derived fuzzy degrees are implemented by using triggers.

Definition 4.6 defines nonassociated fuzzy degrees, that is, fuzzy degrees with their own meaning. This degree is also represented as a numeric attribute. This attribute type implies to update the FCL table (inserting a Type 8 attribute). Optionally, if the degree has a special meaning, this must be inserted in the FDS table. Derived fuzzy degrees are implemented by using triggers.

Definitions 4.9, **4.10**, and **4.11** define fuzzy degrees associated with the whole tuple: fuzzy entities, fuzzy weak entities, and fuzzy relationships. These degrees are also represented as numeric attributes. This attribute type implies to update the FCL table (inserting a Type 7 attribute). Optionally, if the degree has a special meaning, this must be inserted in the FDS table. Derived fuzzy degrees are implemented by using triggers. In addition, this degree inserts one row in the FDT table, with the suitable value in the column DEGREE_TYPE (see the "Table FUZZY_DEGREE_TABLE (FDT)" section in Chapter V).

On the other hand, any fuzzy attribute or fuzzy degree may have fuzzy qualifiers. In this case, fuzzy qualifiers are inserted in the FOL and FQD tables.

Fuzzy Constraints

Fuzzy constraints use fuzzy quantifiers (see Definition 1.22) with zero, one, or two thresholds γ and δ (see the "Thresholds and Fuzzy Quantifiers for Relaxing Constraints" section in Chapter IV). These quantifiers must be inserted in the FMB. Quantifiers associated with an attribute are inserted in the FLD and FOL tables, quantifiers associated with a table (called *table quantifiers*) are defined in the FTQ table, and quantifiers associated with the system (called *system quantifiers*) are defined in the FSQ table. In the FOL, FTQ, and FSQ tables,

the quantifier must be inserted by using the suitable value of attribute FUZZY_TYPE (see Table 5.4). We suggest inserting in the FSQ table at least those quantifiers expressed in Table 5.6, because those quantifiers are general and basic enough and can be used in fuzzy constraints and in fuzzy queries (see Chapter VII).

All constraints are implemented by using triggers, which check whether the constraints are satisfied. Each constraint is represented by using a condition with the quantifier function and the thresholds. If the quantifier is absolute, then it uses only the value *a* of Equation 4.9, and if the quantifier is relative, then it uses the *a* and *b* values of that equation.

These values have different meanings according to the constraint type, and these meanings are indicated in the definition of each constraint: **Definitions 4.13**, **4.14**, **4.15**, **4.16**, **4.17**, **4.21**, **4.22**, **4.23**, and **4.24.**

Definitions 4.18 and **4.19** define fuzzy disjoint and overlapping specializations, respectively. There, the constraint sets that at least one of the subclasses is a fuzzy entity (Definition 4.9). Two options exist:

1. From the point of view of subclasses: Each fuzzy entity in the subclasses is considered like any other fuzzy entity (see the preceding section). This is the solution if you use **Options 8A** or **8B** (especially the second one) in order to map the specialization (see STEP 8 in the "EER-to-Relational Mapping Algorithm" section, earlier in this chapter).
2. From the point of view of the superclass: Each subclass is considered like a crisp entity, and we add a new fuzzy attribute in the superclass. If a similarity relation between the subclasses exists, then the fuzzy attribute is either a Type 3 or a Type 4. This attribute type was treated in the "Fuzzy Values in Fuzzy Attributes" section, and now, the length (attribute LEN in the FCL table) and the linguistic labels of this attribute depend on the chosen option when converting the specialization (STEP 8):
   - **Option 8A**: The length is the number of fuzzy subclasses, and the linguistic labels are the names of those subclasses.
   - **Option 8C**: The fuzzy attribute is the attribute *t* with length equal to 1 (disjoint specialization), and the linguistic labels are the names of all subclasses (fuzzy or not).
   - **Option 8D**: The fuzzy attribute is the attribute *t* with length equal to the number of subclasses *m* (overlapping specialization), and the linguistic labels are the names of all subclasses (fuzzy or not).

It should be noted that the length of this attribute in overlapping specialization controls the number of subclasses to which each member of the superclass can belong in a flexible manner. Thus, this length must be less than or equal to m.

Finally, **Definition 4.20** defines the fuzzy-attribute-defined specializations. Obviously, this definition implies a new fuzzy attribute in the superclass. In this case, the classification of each instance in the superclass is an automatic process, according to this fuzzy attribute and the specialization type (fd, fo, d, or o). In general, disjoint specializations (d) should be treated as fuzzy disjoint specializations (fd), and overlapping specializations (o) should be treated as fuzzy overlapping specializations (fo), because fuzzy versions are very expressive.

Chapter VII

FSQL: A Fuzzy SQL for Fuzzy Databases

The SQL language was essentially developed by Chamberlin and Boyce (1974) and Chamberlin et al. (1976). In 1986, the American National Standard Institute (ANSI) and the International Standards Organization (ISO) published the standard SQL-86 or SQL1 (ANSI, 1986). In 1989, an extension of the SQL standard, called SQL-89, was published, and SQL2 or SQL-92 was published in 1992 (ANSI, 1992).

SQL2 basically provided new types, constraints (such as checks or unique predicates), it supported subqueries in `UPDATE` and `DELETE` operations, and in the `FROM` clause, operator `IN`, `ANY` and `ALL`, `CASE` constructor, `JOIN`, `UNION`, `INTERSECT` and `EXCEPT` operators and the modification of base table through views.

In the latest version of SQL standard, SQL 2003, major improvements have been made in a number of key areas. Firstly, it has additional object-relational features, which were first introduced in SQL-1999. Secondly, SQL 2003 standard revolutionizes SQL with comprehensive OLAP features and data-mining applications. Thirdly, SQL 2003 integrates popular XML standards into SQL (SQL/XML). Finally, numerous improvements have been made throughout the SQL 2003 standard to refine existing features.

Nowadays, SQL may be considered one of the major reasons for the success or relational databases in the commercial world, and the bibliography about SQL language is very extensive (Date & Darwen, 1997; Elmasri & Navathe, 2000; Patrick, 2002). In all these sources, the basic commands of this language are explained in its two dimensions:

- **DML** (Data Manipulation Language): The DML statements (or sentences) enable the query (consultation) and the modification of the data stored in the database. Examples of this kind of sentences are `SELECT`, `INSERT`, `DELETE`, and `UPDATE`.
- **DDL** (Data Definition Language)[1]: The statements of this language enable the creation and modification of the structures in which the data will be stored. Examples of DDL statements are as follows: `CREATE` (to create objects of the database, such as tables, views, etc.), `DROP` (to remove objects), `ALTER` (to modify objects), and statements for security controls and indexes and for the control of the physical storage of the data.

In this chapter we focus on revising the syntax of the most useful and important commands, explaining the news that these commands incorporate into FSQL to enable us to handle fuzzy information. We do not explain the detailed syntax of each command but only the part that FSQL adds to SQL.

FSQL allows three types of comments to be incorporated into the statements that will not be used when the statements are analyzed or executed. The first of these comments (C style) starts with the character sequence `/*` and ends with the sequence `*/`. The second one uses a double hyphen -- and comments from this point to the end of the line. These two comment types are also used in SQL and PL/SQL. In the last type, the start of a comment is marked by the character sequence `/*`, and the comment is terminated by the end of the file (end of statement, end of string, etc.).

The FSQL user is able to prevent a statement by using the FSQL Server. In order to do so, we can use symbol `!` (admiration) as the first character of the statement. This is useful if the FSQL Client program does not admit the possibility of sending an SQL standard statement. With this system, we accelerate the process for this type of statement, because the FSQL Server is not executed completely. In other words, if the admiration symbol is the first character, then the statement will be considered as an SQL statement.

DML of FSQL: SELECT, INSERT, DELETE, and UPDATE

Inside the DML, the most complex and usual statement is the data query sentence SELECT, although INSERT, DELETE, and UPDATE are also useful statements. In this section we define the format for these statements in FSQL.

Novelties in the Fuzzy SELECT of FSQL

The SELECT statement is a very powerful, complex, and flexible sentence. Although very easy to use in simple queries, it is not so easy to use in complex queries due to its power and versatility. This statement is so powerful that rarely is all its expressive power used to carry out a consultation. The normal thing is to perform queries that are a great deal simpler than SELECT permits.

In order to simplify the writing and understanding of complex queries, intermediate views are sometimes used that are created as subqueries in the database (with the CREATE VIEW statement).

The SELECT statement has the following form, where only the two first clauses are obligatory:

```
SELECT <select list>
FROM   <table list>
[WHERE  <condition>]
[GROUP BY <grouping attributes>]
[HAVING   <condition on groups>]
[ORDER BY <columns for order>];
```

The FSQL language is an authentic extension of SQL. This 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. Basically, the following extensions are performed to this statement.

Linguistic Labels

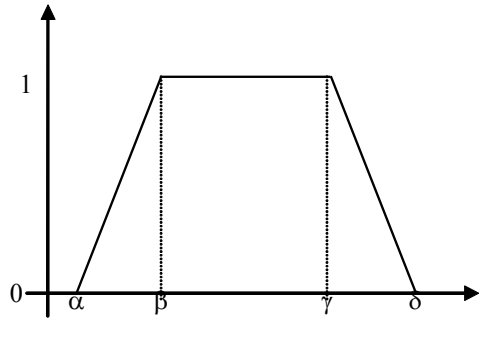
If an attribute is capable of fuzzy treatment, then linguistic labels can be defined on it. These labels will be preceded by the symbol `$` to distinguish them easily. There are two types of labels, which will be used in different fuzzy attribute types:

1. Labels for attributes with an *ordered underlined domain*: Every label of this type has associated a trapezoidal possibility distribution in the FMB[2]. This possibility distribution is generally trapezoidal, linear, and normalized, as Figure 7.1 shows. Example 4.1 uses attributes Height and Age with this kind of label: `$Short`, `$Tall`, `$Young`... (refer to Figure 4.2). These labels are used in fuzzy attributes Type 1 and 2 (see Chapter IV).
2. Labels for attributes with a *nonordered domain* (scalars of Types 1, 3, and 5 in Table 2.4). Here, a similarity relation may exist defined between each two labels in the domain and stored in the FMB. The similarity degree is in the interval [0, 1]. Consider, for example, attribute Color_hair in Example 4.1, which has defined the labels `$Blond`, `$Dark`, and `$Ginger`. These labels are used in fuzzy attributes Type 3 and 4 (refer to Chapter IV).

Fuzzy Comparators

In addition to the typical comparators (=, >, >=...), FSQL includes the fuzzy comparators in Table 7.1. As in SQL, fuzzy comparators compare one column with one constant or two columns of the same (or compatible) type.

Figure 7.1. Trapezoidal, lineal, and normalized possibility distribution



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). In the "Fuzzy Comparisons" section, later in this chapter, we provide a definition for all the comparators, fuzzy or nonfuzzy (see the "Fuzzy Comparators Restrictivity" section for more information about the restrictiveness of the comparators): `FEQ` (or `F=`) uses the possibility measure (Equation 1.40), and `NFEQ` (or `NF=`) uses the necessity measure (Equation 1.41).

In attributes with a nonordered underlying domain (Fuzzy Type 3 or 4), only the fuzzy comparators `FEQ`, `FDIF`, `INCL`, and `FINCL` can be used, because they lack order. These comparators are defined for these types of attributes in the "Fuzzy Comparisons" section, later in this chapter.

Comparators `INC` and `FINCL` do not use possibility and necessity measures, and `INCL` is more restrictive than `FINCL` (`INCL` retrieves fewer rows than `FINCL`).

The fuzzy comparator of "inequality" or "possibly different" may be modeled denying (with `NOT`) a comparison with `FEQ` or `NFEQ`, using the following format: `NOT <F_attribute> FEQ <attrib_or_const>`. However, as you see in the "Fuzzy Comparators" section, this method obtains different results when `FDIF` and `NFDIF` are used, and in addition, the behavior of the `NOT` operator may be changed (see the section on modifying FSQL options, later in this chapter).

Crisp comparators may be used in comparisons with fuzzy attributes, and these are defined later in this chapter.

Table 7.1. The 18 fuzzy comparators for FSQL (Fuzzy SQL): 16 in the possibility/necessity family, and two in the inclusion family

Possibility	Necessity	Significance
FEQ or **F=**	**NFEQ** or **NF=**	Possibly/Necessarily Fuzzy Equal than…
FDIF, **F!=** or **F<>**	**NFDIF**, **NF!=** or **NF<>**	Possibly/Necessarily Fuzzy Different to…
FGT or **F>**	**NFGT** or **NF>**	Possibly/Necessarily Fuzzy Greater Than…
FGEQ or **F>=**	**NFGEQ** or **NF>=**	Possibly/Necessarily Fuzzy Greater or Equal than…
FLT or **F<**	**NFLT** or **NF<**	Possibly/Necessarily Fuzzy Less Than…
FLEQ or **F<=**	**NFLEQ** or **NF<=**	Possibly/Necessarily Fuzzy Less or Equal than…
MGT or **F>>**	**NMGT** or **NF>>**	Possibly/Necessarily Much Greater Than…
MLT or **F<<**	**NMLT** or **NF<<**	Possibly/Necessarily Much Less Than…
FINCL	**INCL**	Fuzzy Included in… / Included in…

Sometimes, it is useful to change the default *margin* and *much* values. The margin is used in approximate values, and the other value is used with the comparators of type `MGT/MLT` in order to consider two time values as being very separated. Either both or one of the values can be specified after one fuzzy comparator in this order. For example, the following simple conditions change one or both values:

```
Distance FEQ(35) #9
Distance MGT(35,88) #9
Distance MGT(,88) #9
```

This syntax permits us to use fuzzy comparators with nonfuzzy columns. In that case, it is obligatory to specify the margin. The other value is only obligatory if we use fuzzy comparators that need it. This option is very important, because FSQL can be used in totally crisp systems. Of course, if we want to use labels, we must declare them.

Fulfillment Thresholds and Qualifiers

For each simple condition, a fulfillment threshold τ may be established (default is 1) with the following format:

```
<condition> THOLD τ
```

indicating that the condition must be satisfied with minimum degree $\tau \in [0, 1]$ to be considered. The reserved word `THOLD` (threshold) is optional and may be substituted by a traditional crisp comparator (=, <, >=, ...), modifying the query meaning. The word `THOLD` is equivalent to using the crisp comparator >=.

Rather than a number, τ may be a qualifier (see the section on qualifiers later in this chapter), that is, an identifier or label that should be defined in the FMB. Qualifiers are also preceded by the symbol `$`.

Example 7.1: "Give me all persons with fair hair (in minimum degree 0.5) that are possibly taller than label `$Tall` (with a high degree)":

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

If we are looking for persons that are *necessarily* taller than label `$Tall`, then we must use the fuzzy comparator `NFGT` instead of `FGT`. The expression `$High` is a qualifier of `Height` that must be defined in the FMB.

*

On a practical level, a qualifier is a constant inside the context of the degrees of an attribute. In addition to making the queries more understandable, another useful feature is that certain queries can be tuned up by simply changing the definition of the qualifier.

FSQL admits thresholds in compound conditions (with logical operators). In general, it is preferable to use parentheses to clarify the influence of the threshold. For example, we can set

```
(<condition1> AND <condition2>) THOLD τ
(<condition1> OR <condition2>) THOLD τ
       (NOT <condition>) THOLD τ
```

Instead of specifying the fulfillment threshold, we can establish the number of items that we desire to recover. This characteristic is also defined by the language SQLf (see the "Other SQL-Based Fuzzy Languages" section, later in this chapter), and here FSQL also establishes this number in square brackets after the word `SELECT`. For example, if we want to select the best 11 rows, we can use `SELECT[11] . . .`

Fuzzy Constants and Fuzzy Expressions

In FSQL, we can use the fuzzy constants as detailed and explained in Table 7.2. It should be noted that the noncontinuous possibility distributions (the last four constant types) are also disjunctive values (see Types 3, 4, 5, and 6 of the GEFRED Model, Table 2.4).

Table 7.3 shows these fuzzy constants and the fuzzy datatypes that can store and use them (in queries, fuzzy conditions, etc.). FIRST-2 limits the number of elements in the stored noncontinuous possibility distributions and in the extended trapezoid.

Fuzzy expressions can be built starting with the syntax of fuzzy constants. For example, if one table has a column `PERCENTAGE`, then some fuzzy expressions are as follows:

```
#PERCENTAGE +- 10
```

Approximately the value of that column with a margin of 10.

```
#5 +- (PERCENTAGE *2)
```

Approximately 5 with a margin specified in column `PERCENTAGE` multiplied by 2.

In other words, values of fuzzy constants can be substituted for expressions, in which they can use constants, names of columns, and arithmetic operators.

Function `CDEG()` and Logic Operators

The function `CDEG` (compatibility degree) may be used with an attribute in the argument. Thus, it computes the fulfillment degree of the condition of the query for the specific attribute, which is expressed between brackets as the argument. If logic operators appear in the condition, the calculation of this compatibility degree is carried out following Table 7.4. By default, the minimum t-norm and the maximum s-norm (or t-conorm) are used (see Chapter I), but the user may change these values with the `ALTER FSQL` statement (see the "Modifying FSQL Options: ALTER FSQL and ALTER SESSION" section, later in this chapter). The user can set the function to be used for every logic operator (`NOT`, `AND`, `OR`). Obviously, this function must be implemented in the FSQL Server or may be implemented by users themselves; the function for the `NOT` must have only a numerical argument, while the functions for the `AND` and for the `OR` must have two numerical arguments. The logical operator precedence is the habitual one, that is, from greater to smaller precedence are `NOT`, `AND`, and `OR`.

In order to change these functions dynamically for a specific logic operation, the FSQL user may use the following:

Table 7.2. Fuzzy constants that may be used in FSQL statements

Fuzzy Constant	Significance
UNKNOWN	Unknown value but the attribute is applicable.
UNDEFINED	The attribute is not applicable or it is meaningless.
NULL	Total ignorance: we know nothing about it.
$[α, β, γ, δ]	Fuzzy trapezoid (with $\alpha \le \beta \le \gamma \le \delta$): example in Figure 7.1.
$[α, β, γ, δ , P1/N1, …, Pn/Nn]	Extended fuzzy trapezoid (with some points Pi/Ni where all the Ni are between α and β or between γ and δ): example in Figure 1.10. Values β and γ are both optional. If they do not exist, then this constant should be a fuzzy value without kernel.
[n, m]	Interval "Between n and m".
n+-m	Fuzzy value "Approximately n": triangle n ± m.
#n	Fuzzy value "Approximately n": triangle n ± *margin* (Figure 1.2), where *margin* is stored in the FMB for each attribute.
$label	Linguistic Label: it may be a trapezoid or a scalar (defined in FMB).
{P1/L1, P2/L2, …, Pn/Ln}	Non-continuous possibility distribution on labels, where P1, P2, …, Pn are the possibility values and L1, L2, …, Ln are the labels.
{L1, L2, …, Ln}	Non-continuous possibility distribution on labels, where L1, L2, …, Ln are the labels, with possibility degree 1 for all of them: {1/L1, …, 1/Ln}.
{P1/N1, P2/N2, …, Pn/Nn}	Non-continuous possibility distribution on numbers, where P1, P2, …, Pn are the possibility values and N1, N2, …, Nn are the numbers.
{N1, N2, …, Nn}	Non-continuous possibility distribution on numbers, where N1, N2, …, Nn are the numbers, with possibility 1 for all of them: {1/N1, …, 1/Nn}.

Table 7.3. Fuzzy constants and the fuzzy datatypes that can store and use them

Fuzzy Constant	Datatypes for storing	Datatypes for using
UNKNOWN	2, 3 and 4	2, 3 and 4
UNDEFINED	2, 3 and 4	2, 3 and 4
NULL	All	All
$[α, β, γ, δ]	2	1 and 2
$[α, β, γ, δ , P1/N1, …, Pn/Nn]	2	1 and 2
[n, m]	2	1 and 2
n+-m	2	1 and 2
#n	2	1 and 2
$label	2, 3 and 4	1, 2, 3 and 4
{P1/L1, P2/L2, …, Pn/Ln}	3 and 4	3 and 4
{L1, L2, …, Ln}	3 and 4	3 and 4
{P1/N1, P2/N2, …, Pn/Nn}	2	1 and 2
{N1, N2, …, Nn}	2	1 and 2

Table 7.4. Default computation for function CDEG with logic operators in FSQL

<Condition> (with logic operators)	CDEG(<Condition>)
<cond1> AND <cond2>	min(CDEG(<cond1>),CDEG(<cond2>))
<cond1> OR <cond2>	max(CDEG(<cond1>),CDEG(<cond2>))
NOT <cond1>	1 CDEG(<cond1>)

1. `NOT (negation)`
2. `AND (t-norm)`
3. `OR (s-norm)`

where `negation`, `s-norm`, and `t-norm` are alphanumeric values indicating the function. Negation must be a negation function and the others use Tables 1.1 and 1.2: "minimum," "product," "drastic product," "bounded product *p*," "Einstein product," "Hamacher product *p*," "maximum," "sum-product," "drastic sum," "bounded sum *p*," "Einstein sum," and so forth. It should be noted that for the sake of simplicity, if the norm needs some argument *p*, it is included after the name.

If the argument of the `CDEG` function is an attribute, then it uses only the conditions that include that attribute. If the attribute indicated as the argument of `CDEG` does not appear in the condition, then this function is not applicable, but instead of giving an error, it proceeds to return degree 1 for all the rows.

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.

Function `CDEG` may be used in the select list (expressions after the reserved word `SELECT`). This is useful in order to show, in a column, the fulfillment degree for each row (or tuple).

Character `%`

Character % is similar to the character `*` of SQL, but it also includes the columns for the fulfillment degrees of the attributes in which they are relevant. In the result, you will also find the function `CDEG` applied to each and every one of the fuzzy attributes that appear in the condition. This character may also, of course, be used with the format `[[scheme.]table.]%`, as for the following example: `Person.%`.

If a fuzzy attribute does not appear in the `WHERE` clause, then its `CDEG` is not applicable, and its `CDEG` will not appear if the wild card `%` is used.

Condition With IS

The format of another kind of condition that can be used is as follows:

```
<Fuzzy_Attribute> IS [NOT] { UNKNOWN
                           { UNDEFINED
                           { NULL
```

Remarks concerning the condition with IS:

- This condition (without NOT) will be true if the left fuzzy attribute value (<Fuzzy_Attribute>) is the fuzzy constant placed on the right.
- If the attribute is not fuzzy and the constant is NULL, then this constant will be understood in the way given by the DBMS.
- If FEQ is used instead of IS, the compatibility degree between attribute and constant is compared (Equation 1.40), and not only when the attribute is *equal* to the constant.

Example 7.2: One query, which shows a compatibility degree, uses a trapezoidal constant, and avoids the UNKNOWN values, could possibly be as follows:

```
SELECT City, Inhabitants, CDEG(*)
FROM   Population
WHERE  Country = 'Spain'
  AND  Inhabitants FGEQ $[200,300,650,800] .75
  AND  Inhabitants IS NOT UNKNOWN
ORDER BY 3 DESC;
```

where Inhabitants is a fuzzy attribute Type 2 with *margin* = 100. For example, Table 7.5 shows some possible results for this query. Some remarks concerning the example include the following:

Table 7.5. Example 7.2: Some results

City	Inhabitants	CDEG(*)
Madrid	#3000	1
Córdoba	#314	1
Alicante	#275	0.88
La Coruña	$[225,242,275,300]	0.8
Granada	[225,275]	0.75

- The minimum threshold is set at 0.75. The word `THOLD` does not appear because it is optional. In the resulting table, the `CDEG` column will have values in [0.75, 1].
- As we are using the comparator `FGEQ`, the last two values of the trapezoid will not be used, that is, if the number of inhabitants equals or exceeds 300, then the degree will be 1. Naturally, if the number of inhabitants is equal to or less than 200, then the degree will be 0.
- If a Spanish city has the possibility distribution `$[50,150,200,300]` in the `Inhabitants` attribute, then its fulfillment degree of the condition is 0.5 and it does not appear in the final result, because the minimum has been set at 0.75.

*

Fuzzy Quantifiers in Queries

Absolute and relative fuzzy quantifiers (refer to Chapter I) can be used in queries. A survey of methods for evaluating quantified statements is shown in Delgado, Sánchez, and Vila (1999, 2000), Sánchez (1999), and Yager (1983).

In FSQL, each quantifier must be preceded by the symbol &. FSQL associates each quantifier to a column, to a table, or to the system. In one statement, FSQL decides which quantifier to use: column quantifier, table quantifier, or system quantifier. The user can however specify which quantifier to use, preceding the quantifier with the owner object, using the dot notation. Take a look at some examples:

1. `Players.Height.&Most`: This is the column quantifier `&Most` of the attribute `Players.Height`.

2. `Players.&Most`: This is the table quantifier `&Most` of the table `Players`.
3. `SYSTEM.&Most`: This is the system quantifier `&Most`.

If a statement needs a column quantifier and that quantifier is not defined, then the statement uses the table quantifier. Similarly, if a table quantifier is not defined, then the statement uses the system quantifier. Finally, if a statement needs a quantifier and that quantifier is not defined, then the statements give an error.

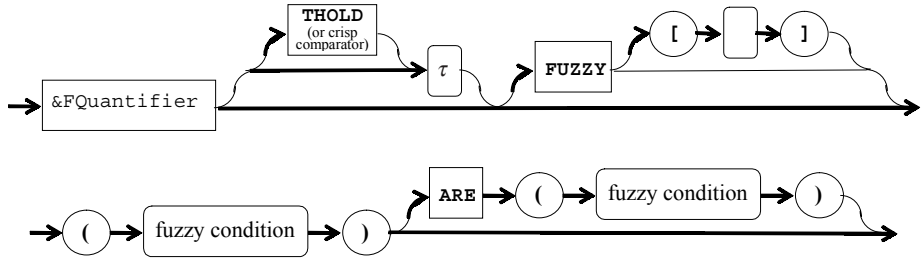
In addition, we can write a quantifier directly in FSQL by giving the four trapezoidal arguments: `&[`α, β, γ, δ`]`. In this case, we must decide if the quantifier is absolute or relative, preceding the definition with `&A` or `&R`. We can use the fuzzy quantifier types expressed in Table 5.4, using the `FUZZY_TYPE` value instead of `A/R`. Examples:

1. `&A&[10,20,30,40]`: This is the absolute quantifier "approximately between 20 and 30" (with margin 10).
2. `&R&[0.1,0.35,0.49,0.5]`: This is the relative quantifier "almost half or more."
3. `&12&[10,20,9999,9999](x)`: This is the Type 12 absolute quantifier "much greater than x," where x is an argument. The final quantifier is `[10+x,20+x,9999+x,9999+x]`.
4. `&16&[-100,0,0,100](x,y)`: This is the Type 16 absolute quantifier "approximately between x and y" (with margin 100). The final quantifier is `[x-100,x,y,y+100]`.

FSQL allows fuzzy quantifiers in queries in the clauses `HAVING` and `WHERE`. In the `HAVING` clause, FSQL distinguishes between statements with or without group functions. In the `WHERE` clause, FSQL distinguishes between explicit and implicit queries: Explicit queries are more flexible, but implicit queries easily solve many typical queries. FSQL then distinguishes the following four forms to use fuzzy quantifiers:

1. Fuzzy quantifiers in the `HAVING` clause.
2. Fuzzy absolute quantifiers in the `HAVING` clause with group functions.

Figure 7.2. Structure for fuzzy quantifiers in FSQL in the `HAVING` *clause*



3. Fuzzy quantifiers in the `WHERE` clause using explicit queries.
4. Fuzzy quantifiers in the `WHERE` clause using implicit queries.

1. Fuzzy Quantifiers in the `HAVING` Clause

The `HAVING` clause restricts the groups of returned rows to those groups formed by the `GROUP BY` clause. This format sets fuzzy conditions over these groups and is expressed in Figure 7.2. This format basically includes the following two forms, which are an extension of those of Bosc and Pivert (1995):

1. `&FQuantifier THOLD τ FUZZY[ρ] (<fuzzy_condition>)`
2. `&FQuantifier THOLD τ FUZZY[ρ] (<fuzzy_condition1>)`
   `ARE (<fuzzy_condition2>)`

where `&FQuantifier` is a fuzzy quantifier preceded by the symbol *&*. Some quantifiers may have arguments (Refer to Chapters I and V), such as `&About_half_of(`*x*`)` and `&Much_Greater_Than(`*x*`)`. The clauses `FUZZY` and `THOLD` are optional.

The usefulness of both forms is as follows:

- The first form restricts the groups to those groups for which the specified fuzzy condition is satisfied by a number of rows, such that this number satisfies the fuzzy quantifier `&FQuantifie` (with minimum degree τ). If the quantifier is relative, then the value of reference (value *b* in Equation 1.64) is the number of rows in the group.

- The second form restricts the groups to those groups for which the quantifier is satisfied by the number of rows (or elements of each group), which satisfy both conditions (1 and 2). If the quantifier is relative, then the value of reference (value *b* in Equation 1.64) is the number of rows in the group satisfying the condition 1. Of course, this second form should only be used with relative quantifiers (most, about half of, etc.).

As in simple fuzzy comparisons, τ is an optional quantifier threshold (by defect 1), normally in [0, 1], which should comply the quantifier so that the condition is evaluated as certain. Here, as well, the word `THOLD` is optional and can also be substituted for any crisp or traditional comparator, modifying the meaning of the query. The word `THOLD` is equivalent to using the crisp comparator >=.

This format uses the column quantifier of the first column in the `GROUP BY` clause. If this quantifier does not exist, then it uses the table quantifier of the first table in the `FROM` clause. If this table quantifier does not exist, then it uses the system quantifier. However, with the dot notation, the user can always decide which quantifier to use.

If the `GROUP BY` clause does not exist, then this statement uses the table quantifier applied to all rows in only one group.

By means of some examples, we will show the power of the fuzzy quantifiers in FSQL:

Example 7.3: Following Example 4.2, we can "select the basketball teams that have *many* (with minimum degree 0.5) very good and tall players (with thresholds 0.75), showing the degree with which each team complies the quantifier":

```
SELECT  Team,  CDEG(*)
FROM    Players
GROUP   BY Team
HAVING  &Many  THOLD  0.5
                (Height   FEQ $Tall       0.75  AND
                 Quality  FEQ $Very_Good  0.75);
```

It should be observed that the condition (which the quantifier requires) can be multiple, and the quantifier `&Many` should be defined in the FMB. The definition of `&Many` is firstly associated to the attribute `Team`. If this attribute does not have the quantifier `&Many`, then the statement uses the table quantifier `Players.&Many`, and if this quantifier does not exist, then it uses the system quantifier `System.&Many`. Of course, if the system quantifier is not defined either, then the statement gives an error.

*

When there are group statements (with `GROUP BY/HAVING`), function `CDEG(*)` references the condition on the groups (`HAVING`).

Example 7.4: "Select the basketball teams in which *most* (with minimum degree 0.5) of their tall players *are* also very good players (with thresholds 0.75)":

```
SELECT  Team,  CDEG(*)
FROM    Players
GROUP   BY Team
HAVING  &Most  0.5
               (Height   FEQ $Tall       0.75)
          ARE  (Quality  FEQ $Very_Good  0.75);
```

*

The `FUZZY` clause is optional, and its argument ρ is also optional. If this clause is used, then the evaluation of the quantifier is computed by adding the fulfillment degree of these elements, rather than the elements complying with the condition. We clarify this concept in the following examples.

Example 7.5: "Select the basketball teams with *many more than* 3 (with minimum degree 0.5) tall players (with minimum degree 0.75)":

```
SELECT  Team,  CDEG(*)
FROM    Players
```

```
GROUP   BY Team
HAVING  &Much_Greater_Than(3)  0.5
        (Height FEQ $Tall 0.75);
```

Suppose the quantifier `&Much_Greater_Than(x)` defined in Figure 1.27a. With $x = 3$, if $\phi \in [4, 12]$, then $Q(\phi) = (\phi - 4) / 8$. Thus, if a team has, for example, four players complying with the condition, that is to say four players that are tall with minimum degree 0.75, then this team complies with the quantifier in degree $Q(4) = 0$. Therefore, that team will not appear in the result, because a 0.5 minimum degree is required. Now suppose another team has eight players that comply with the condition. This team then complies with the quantifier in degree $Q(8) = 0.5$, so this team will appear in the result. Finally, if a team has 12 players that comply with the condition, the team satisfies the quantifier in degree $Q(12) = 1$.

*

In the previous example, if a team has *n* players that comply with the condition, its fulfillment degree does not depend on the degree to which each of those *n* players satisfies the condition. That is to say, a team with *n* tall players, all in degree 0.75, has the same fulfillment degree as a team with *n* tall players, all in degree 1.

In short, in the previous example, the set of players complying with the condition is considered to be crisp (a player either does or does not satisfy the condition of being tall with a minimum degree of 0.75).

Nevertheless, there is another method (Zadeh, 1983) that considers the set of the elements that comply with the fuzzy condition. FSQL computes by using this method if we utilize the reserved word `FUZZY` after the quantifier. This method does not count the number of elements that comply with the condition but adds the fulfillment degrees of those elements.

Example 7.6: "Select the basketball teams that have *many more than* three (with minimum degree 0.5) high players (with minimum degree 0.75), *considering the set of the players that comply with the condition as fuzzy*":

```
SELECT  Team
FROM    Players
```

```
GROUP   BY Team
HAVING  &Much_Greater_Than(3)  0.5  FUZZY
         (Height FEQ $Tall 0.75);
```

Consequently, for example, a team `T1` with eight players, all of whom satisfy the condition in degree 1, complies with the quantifier in degree $Q(8) = 0.5$. Suppose another team `T2` with eight players complying with the condition, but two of them comply with it in degree 1, and four of them in degree 0.75. In this last case, adding the fulfillment degrees, we obtain $(2 * 1 + 4 * 0.75) = 5$. The team then complies with the quantifier in degree $Q(5) = 0.125$, and this team will not be selected in the final result.

If the set of players that comply with the condition is considered as fuzzy (`FUZZY` clause), then `T1` is selected, and `T2` is not selected. Otherwise, both teams are selected with the same degree (0.5).

*

An argument $\rho \in [0, 1]$ can be used so that `FUZZY [ρ]` indicates that in order to evaluate the quantifier, both values are used (with and without the word `FUZZY`), weighting the first of these (with `FUZZY`) with the importance that $\rho - 1$ indicates and the second (without `FUZZY`) with the importance ρ. Then, using `FUZZY [0]` is equivalent to using `FUZZY` (without ρ), and using `FUZZY [1]` is equivalent to not using the `FUZZY` clause. If ρ has a small value, then the query is more restrictive (it retrieves fewer tuples), and if ρ has a large value, then the query is less restrictive (it retrieves more tuples).

2. Fuzzy Absolute Quantifiers in the `HAVING` Clause with Group Functions

The `HAVING` clause may contain conditions with group or aggregate functions such as `MAX`, `MIN`, `COUNT`, `SUM`, `AVG`, and so forth. The values returned by these functions may be compared with absolute fuzzy quantifiers by using fuzzy comparators.

Example 7.7: Suppose that each employee must belong to one department, and so a 1:N relationship exists between departments and employees. This constraint is implemented by using a foreign key attribute `Department_Code` in the table `Employee`, which references the table `Department`. In this

context, we can ask for the following query: "Give me the departments with approximately 5 employees":

```
SELECT  Department_Code,  Department_Name,  CDEG(*)
FROM    Department, Employee
GROUP  BY  Department_Code,  Department_Name
HAVING    COUNT(*)  FEQ  Employee&Approx_5;
                                                    *
```

Of course, we can use fuzzy conditions without fuzzy quantifiers in the HAVING clause. For example, consider the following clause in the previous example: `HAVING AVG(Salary) FEQ $Big_Salary`, where `Salary` may be a fuzzy attribute Type 1, with the `$Big_Salary` label defined in the FMB.

3. Fuzzy Quantifiers in the WHERE Clause Using Explicit Queries

In the WHERE clause, we can use a quantifier with the following format:

1. Absolute quantifiers:

```
&FQuantifier THOLD τ FUZZY[ρ] (<subquery>)
```

2. Relative quantifiers:

```
&FQuantifier THOLD τ FUZZY[ρ] (<subquery>):(<subquery2>)
```

where `<subquery>` and `<subquery2>` are SELECT statements (fuzzy or not). With absolute quantifiers, these are applied to the number of rows of `<subquery>`. With relative quantifiers, the value of reference (value *b* in Equation 1.64) is the number of rows of `<subquery2>`.

It is important to know that `<subquery2>` is optional. If `<subquery2>` does not appear, then the value of reference in relative quantifiers is the number

of rows in the tables of `<subquery>`. In this case, this subquery should use only one table. Furthermore, if the quantifier is a table quantifier, then the value of reference is the number of rows in this table (the table of the quantifier). These details simplify many queries, as shown in the following examples or the fuzzy division (see the section on fuzzy division queries later in this chapter).

These quantifiers are especially useful with correlated nested subqueries, in which the subquery is evaluated once per each row of the outer query.

By default, this format uses the system quantifiers. In any other case, it must be indicated in the statement.

The `FUZZY` clause is optional, and its argument ρ is also optional. If the `FUZZY` clause is used, then the evaluation of the quantifier is computed by adding the `CDEG(*)` values of the subquery. This clause would only be used with fuzzy queries. If the optional argument $\rho \in [0, 1]$ appears, then in order to evaluate the quantifier, both values (with and without the word `FUZZY`) will be used, weighting the first of these (with `FUZZY`) with $\rho - 1$ and the second value (without `FUZZY`) with ρ. Using `FUZZY[0]` is equivalent to using `FUZZY` (without ρ), and using `FUZZY[1]` is equivalent to not using the `FUZZY` clause. If ρ has a small value, then the query is more restrictive, and if ρ has a large value, then the query is less restrictive.

Example 7.8: Following the context of Examples 4.3 and 4.11 (see Figures 4.5 and 4.13), "select the skilled employees (with minimum degree 0.75) that work for *many* (with minimum degree 0.5) projects" (using the definition of `&Many` associated to table `Project`):

```
SELECT  Employee_Code,  Employee_Name,  CDEG(*)
FROM    Employee E
WHERE  Project.&Many  0.5
          (SELECT * FROM Works_For W
            WHERE W.Employee_Code = E.Employee_Code)
   AND Ability FEQ $Skilled 0.75;
```

This statement explicitly references the table quantifier over a correlated nested subquery. For each employee, the subquery searches for his or her projects.

*

Example 7.9: "Select the employees that work for *most* (with minimum degree 0.5) of the projects" (using the definition of `&Most` associated to the system):

```
SELECT  Employee_Code,  Employee_Name,  CDEG(*)
FROM    Employee E
WHERE  &Most  0.5
          (SELECT * FROM Works_For W
            WHERE W.Employee_Code = E.Employee_Code):
          (SELECT * FROM Project)
   AND Ability FEQ $Skilled 0.75;
```

If we use the table Project quantifier, then the second query is useless:

```
SELECT  Employee_Code,  Employee_Name,  CDEG(*)
FROM    Employee E
WHERE  Project.&Most  0.5
          (SELECT * FROM Works_For W
            WHERE W.Employee_Code = E.Employee_Code)
   AND Ability FEQ $Skilled 0.75;
                                                          *
```

Example 7.10: "Select the employees that work for *many* (with minimum degree 0.5) long-lasting projects (with long-lasting duration with degree 0.75), considering the set of long-lasting projects as a fuzzy set":

```
SELECT  Employee_Code,  Employee_Name,  CDEG(*)
FROM    Employee E
WHERE  Project.&Many  0.5  FUZZY
          (SELECT * FROM Works_For W,  Project P
            WHERE W.Project_Code = P.Project_Code
              AND W.Employee_Code = E.Employee_Code
              AND P.Duration FEQ $Long_Lasting 0.75);
                                                          *
```

Example 7.11: Let us suppose that the relation `MANAGERS(Employee_Code,Manager)` stores all the managers of one employee and all the employees of one manager (a many-to-many relationship). The query "Give me the employees with *approximately two* managers" is then solved as follows:

```
SELECT  Employee_Code,  CDEG(*)
FROM    MANAGERS M1
WHERE   &Approx_2 THOLD 0
        (SELECT *
         FROM   MANAGERS M2
         WHERE  M1.Employee_Code = M2.Employee_Code);
```

Note that we can use only fuzzy absolute quantifiers in this kind of query.

*

Example 7.12: "Give me the employees that earn less than the majority (that is to say, those employees for which most of the employees earn more)":

```
SELECT  Employee_Code,  Employee_Name,  CDEG(*)
FROM    Employee E1
WHERE   &Most 0.5
        (SELECT * FROM Employee E2
         WHERE E2.Salary > E1.Salary);
```

Quantifier `&Most` is relative, and a second subquery does not exist. The value of reference is then the number of rows in the subquery table, that is, `Employee (E2)`.

*

Example 7.13: With the scheme of Example 7.7, we can query the following: "Give me the employees that work for departments with less budget than the majority (that is to say, those employees for which most departments have a greater budget than the budget of his or her department)":

```
SELECT  Employee_Code,  Employee_Name,  CDEG(*)
FROM    Employee E, Department D1
WHERE   E1.Department_Code  =  D1.Department_Code
  AND   &Most 0.5
        (SELECT * FROM Department D2
         WHERE D2.Budget > D1.Budget);
```

The value of reference is the number of table `Department (D2)`.

*

4. Fuzzy Quantifiers in the `WHERE` Clause Using Implicit Queries

The implicit queries format for fuzzy quantifiers solves many typical queries simply. In the `WHERE` clause, we can use a quantifier using the following format for absolute and relative quantifiers:

```
<table1>.&FQuantifier  THOLD  τ  <table2>
```

where the threshold and the second table are optional.

With absolute quantifiers, these are applied to the number of rows in an intermediate table between the table of the quantifier `<table1>` and the table in the `FROM` clause. An intermediate table is a table with two foreign keys: One of them references the table `<table1>`, and the other references the table in the `FROM` clause. If there are two or more tables in the `FROM` clause with an intermediate table with `<table1>`, then the system chooses the first one.

With relative quantifiers, the value of reference (value *b* in Equation 1.64) is the number of rows of `<table1>` (table of the quantifier).

Example 7.14: Example 7.8 would be solved by

```
SELECT  Employee_Code,  Employee_Name,  CDEG(*)
FROM    Employee
WHERE   Project.&Many 0.5
  AND   Ability FEQ $Skilled 0.75;
```

Note that it is important for the quantifier to be the quantifier of `Project`. It is worth mentioning that although this query uses the table `Works_for`, it does not appear. A different command is "Select the projects in which *many* employees work":

```
SELECT  Project_Code,  Project_Name,  CDEG(*)
FROM    Project
WHERE   Employee.&Many  0.5;
```

The quantifier is now the quantifier of `Employee`, and table `Works_for` is also needed. Obviously, the concept of "many" is different in "many projects for one employee" and in "many employees for one project."

*

Example 7.15: Another command is "Select the projects in which *many* machines take part":

```
SELECT  Project_Code,  Project_Name,  CDEG(*)
FROM    Project
WHERE   Machine.&Many  0.5  Machines_for_Project;
```

The quantifier is now the quantifier of `Machine`, and table `Machines_for_Project` is also needed. If this table is the only table joining tables `Project` and `Machine`, then the specification is optional.

*

Example 7.16: Example 7.9 would be solved as follows:

```
SELECT  Employee_Code,  Employee_Name,  CDEG(*)
FROM    Employee E
WHERE   Project.&Most  0.75
  AND   Ability FEQ $Skilled 0.75;
```

*

If an intermediate table does not exist between the table of the quantifier and the table in the FROM clause, then FSQL searches a foreign key in the table of the quantifier, referencing the table in the FROM clause.

Example 7.17: Let us consider the tables of Example 7.7: Employee and Department. "Give me the departments with approximately five employees":

```
SELECT  Department_Code,  Department_Name,  CDEG(*)
FROM    Department
WHERE   Employee&Approx_5  0.5;
```

It should be noted that the following query is not valid:

```
SELECT  Employee_Code,  Employee_Name,  CDEG(*)
FROM    Employee
WHERE   Department&Approx_5  0.5;
```

*

If FSQL does not find that foreign key, then it raises an error.

Example 7.18: "Give me the departments with about half of the employees":

```
SELECT  Department_Code,  Department_Name,  CDEG(*)
FROM    Department
WHERE   Employee&About_Half_of  0.5;
```

In this relative quantifier, the value of reference is the number of tuples of Employee.

*

Fuzzy Division Queries

The relational division operation can also be performed with FSQL. This operation is not directly implemented in SQL, and its expression is not immediate. In this section, we give the FSQL syntax for fuzzy division, but for the sake of simplicity, we do not include how FSQL makes the division. This is explained in Galindo, Medina, Cubero, and García (2001).

Let R and R' be relations with headers (A, B, C) and (B, D), respectively, where A, B, C, and D are simple attributes or sets of attributes denoted by $A=\{A_1, A_2, ..., A_n\}$ and $B=\{B_1, B_2, ..., B_m\}$. The fuzzy division then has the following general format in FSQL language:

```
SELECT A [, CDEG(*)]
FROM   R [, <table_clause>]
WHERE  [ONCEPERGROUP]
       &FQuantifier THOLD τ
       ( <subquery> )
```

where the items in square brackets are optional items, the items in angled brackets are items to expand, and the meaning of each element is as follows:

- **Select list**: A is a list with the R attributes that we are looking for. `CDEG(*)` will show the compatibility or fulfillment degree of the A elements.
- **`FROM`**: This clause is used to indicate the relevant relation R and other possible relations in `<table_clause>`. These relations (as in SQL) may be names of tables, views, snapshots, or subqueries. The optional clause `<table_clause>` is useful to indicate a list of attributes $X \subseteq \{B \cup D\}$ by using a subquery such as (`SELECT` X `FROM` R') and whatever other relations. The normal division is defined with $X=\varnothing$. If X is not empty, then the command will show the Cartesian product with these attributes.
- **`&FQuantifier`**: The regular division uses the universal quantifier, but fuzzy division allows us to use any other quantifier. `&FQuantifier` is

a fuzzy quantifier. The fuzzy quantifier `&FQuantifier` must be defined in the FMB, with the exception of the quantifier `&ALL` (∀) and the quantifier `&EXISTS` (∃). The optional value τ indicates that the threshold (default is 1) applied to the fulfillment degrees in the resulting relation (column of `CDEG` function).

- **`<subquery>`**: This is a subquery with the following format:

```
SELECT  *
FROM  R'
WHERE  R.B1 <FCOMP1> R'.B1  [[THOLD]  γ1]
AND  ...
AND  R.Bm <FCOMPm> R'.Bm  [[THOLD]  γm]
```

where

- *R'* is the second relevant relation, the divisor, in the fuzzy division. It may be a subquery or the `DUAL` table. `DUAL` is a table that is automatically created by Oracle[3] together with the data dictionary. `DUAL` is in the schema of the user `SYS` but is accessible by the name `DUAL` to all users. We will use `DUAL` when the relation *R'* does not exist, but we want to use a virtual relation using a constants set in the `WHERE` clause. In some DBMSs, such as MySQL[4], the `FROM` clause is optional, and then the `DUAL` table is not useful.
- B_i with $i = 1, 2, ..., m$ are the set of attributes *B* of *R* and *R'*. These are qualified with the name of their relation (*R* or *R'*) because they would have the same name. If we use `DUAL` instead of *R'*, then we must use constants instead of *R'* attributes, distinguishing tuples with the OR operator (see Example 7.19).
- `<FCOMP`$_i$`>` with $i = 1, 2, ..., m$ are the fuzzy comparators (Table 7.1) used for each two attributes. In order to retrieve the standard division results, we must obviously use the fuzzy comparators `FEQ` or `NFEQ`. Fuzzy division selects tuples of the first relation that are related *in some way* to `&FQuantifier` of tuples in the second relation. This *way* is indicated by these fuzzy comparators.

  - γ_i with $i = 1, 2, ..., m$ are thresholds for each *B* attribute. All these thresholds must be 0 in the standard division.
- **`ONCEPERGROUP` option**: With classic relations, one single tuple in *R* connects with zero or one tuple in *R'*, but with fuzzy relations, one single tuple in *R* may connect with zero, one, some, or all of the tuples in *R'*. When the reserved word `ONCEPERGROUP` is used, then every tuple in *R* is used only *once in each group* (of one *A* element) according to where it obtains the greatest possibility degree. If there are some items with the same greatest value, then we must maximize all the degrees in that group of *A* values. It should be noted that when solving this problem we may prevent some possibly useful information being shown, and this substantially increases the number of operations. This option may be especially useful when fuzzy comparators that are different from `FEQ` or `NFEQ` are used. In summary, using `ONCEPERGROUP` is more restrictive than not using it, because it requires each *A* value of *R* to have at least as many tuples as relation *R'*.

Example 7.19: Suppose we have a fuzzy relational database about basketball players. A database relation `PLAYERS` (*R*) may have the attributes (`PLAYER`, `TEAM`, `HEIGHT`, `QUALITY`, `NUM_SHIRT`...), with `HEIGHT` and `QUALITY` being fuzzy attributes Type 2. In addition, TYPES (*R'*) is a relation with only two attributes (`HEIGHT`, `QUALITY`), storing some players' types. In this context, we are going to find those basketball teams whose player types (in `HEIGHT` and `QUALITY`) match or pair up with *most* of those in *R'*.

```
SELECT  TEAM,  CDEG(*)
FROM    PLAYERS
WHERE   &Most  THOLD  0
        (SELECT *
         FROM  TYPES
         WHERE  PLAYERS.HEIGHT  FEQ  TYPES.HEIGHT  THOLD  0
          AND PLAYERS.QUALITY FEQ TYPES.QUALITY THOLD 0);
```

If relation TYPES does not exist, then we can use a subquery instead of that relation. For example, in the following statement, relation TYPES is substituted by the player types of the Málaga team:

```
SELECT  TEAM,  CDEG(*)
FROM    PLAYERS
WHERE   &Most  THOLD  0
        (SELECT *
         FROM ( SELECT  HEIGHT,  QUALITY
                FROM  PLAYERS
                WHERE  TEAM  =  'Málaga')  TYPES
          WHERE PLAYERS.HEIGHT FEQ TYPES.HEIGHT THOLD 0
           AND PLAYERS.QUALITY FEQ TYPES.QUALITY THOLD 0);
```

Moreover, the same result may be obtained if we want to set the values of TYPES (*R'*) directly. For example, suppose only two tuples in the divisor relation:

```
SELECT  TEAM,  CDEG(*)
FROM    PLAYERS
WHERE   &Most  THOLD  0
        ( SELECT *
          FROM  DUAL
          WHERE  PLAYERS.HEIGHT  FEQ  $Short  THOLD  0
             AND  PLAYERS.QUALITY  FEQ  $Very_Good  THOLD  0
              OR  PLAYERS.HEIGHT   FEQ  $Very_Tall  THOLD  0
           AND  PLAYERS.QUALITY FEQ  $Bad        THOLD  0);
                                                          *
```

Table 7.6. Fuzzy set operators in FSQL, applied to queries `R` and `T`

Operator	Returns
`FUNION`	All rows selected by either query (`R` or `T`). If there are duplicated tuples (in `R` and `T`), it uses, by default, the maximum s-norm: max (`R.CDEGROW`, `T.CDEGROW`)
`FINTERSECT`	All rows selected by both queries (`R` and `T`). If there are duplicated tuples (in `R` and `T`), it uses, by default, the minimum t-norm: min (`R.CDEGROW`, `T.CDEGROW`)
`FMINUS`	All distinct rows selected by the first query (`R`) but not the second (`T`). If there are duplicated tuples (in `R` and `T`), it uses, by default, the function: max (`0`, `R.CDEGROW – T.CDEGROW`)

Fuzzy Set Operators

In SQL, you can combine multiple queries by using the set operators `UNION`, `UNION ALL`, `INTERSECT`, and `MINUS` (or `EXCEPT`). In FSQL, if these queries include some fuzzy degree associated to the whole tuple (or row), you can use an extended version of these set operators, which are listed and explained in Table 7.6 together with the default values. This degree is either the fuzzy degree of a fuzzy entity or a fuzzy relationship (refer to Chapter IV), or the compatibility degree, `CDEG(*)`, of a fuzzy subquery.

We can change the default functions of these fuzzy set operators by using the `ALTER FSQL` statement (see the "Modifying FSQL Options: ALTER FSQL and ALTER SESSION" section, later in this chapter). However, we can change these functions dynamically for a specific operation by using

1. `FUNION (s-norm)`
2. `FINTERSECT (t-norm)`
3. `FMINUS (s-norm)`

where s-norm and t-norm are alphanumeric values according to Tables 1.1 and 1.2: "minimum," "product," "drastic product," "bounded product *p,*" "Einstein product," "Hamacher product *p,*" "maximum," "sum-product," "drastic sum," "bounded sum *p,*" "Einstein sum," and so forth. Note that for the sake of simplicity, if the norm needs some argument *p*, it is included after the name.

Fuzzy Time

In this section, we present some new fuzzy data types based on time concepts. We extend the existing temporal data types in SQL2 and give a brief overview of basic definitions in fuzzy temporal databases. In particular, we explain how query operations in TSQL2 language need to be extended for fuzzy temporal querying. TSQL2 language extends SQL for querying valid time, transaction time, and bitemporal relational databases.

Many times databases require some aspect of time when their information is organized. Temporal databases, in the broadest sense, encompass all databases of this kind (Elmasri & Navathe, 2000; Jensen et al., 1994).

We extend the existing data types in SQL2. Time in databases is very useful, because it allows us to store the date and time in which the fact was considered to be true in the real world. However, sometimes this time is not exactly known or is a vague time period (gradual events). We study this scenario here and give a brief overview of basic definitions in fuzzy temporal RDB.

We also explain how query operations in TSQL2 language (Snodgrass, 1995) need to be extended for fuzzy temporal querying. TSQL2 language extends SQL for querying valid time, transaction time, and bitemporal RDB.

Dubois and Prade (1989) introduce a framework for modeling temporal knowledge pervaded with imprecision or uncertainty in terms of possibility distributions: ill-known dates, time intervals with fuzzy boundaries, fuzzy durations, and uncertain precedence relations between events. In Quian (1992), fuzzy time is used for problem solving in industrial dynamic systems. In Virant and Zimic (1996), the time-dependent fuzzy set $A(t)$ is defined as a fuzzy set whose membership function may change with time. An inference processing is also defined in order to obtain the fuzzy set $A(t)$ when t is a time fuzzy set, based on Dubois and Prade (1989). Some authors have applied these definitions in time-dependent fuzzy rules.

Many authors (Dubois & Prade, 1989; Malik & Binford, 1983) have pointed out that the problems of processing space or time information in reasoning are similar. Time may be considered as a fuzzy attribute Type 2, where its underlying domain (X axis in Figure 7.1) is the absolute time. This absolute time is an ordered sequence of points in time. It is necessary to set a *granularity* in this absolute time; that is, time is considered to be an ordered sequence of points with a minimum and fixed distance between every two consecutive points. Of course, granularity is determined by the application. This special

underlying domain makes the definition of new fuzzy types necessary.

There are two types of temporal information according to the event duration:

1. **Point events**: These are events or facts without duration and are typically associated with a single time point in some granularity. In SQL2, temporal data types including this information are as follows: `DATE` (specifying year, month, and day), `TIME` (specifying hour, minute, and second), and `TIMESTAMP` (specifying a `DATE` and `TIME` combination). Fuzzy point events allow fuzzy values to be used in this domain. We define new fuzzy time types named `FUZZY_DATE`, `FUZZY_TIME`, and `FUZZY_TIMESTAMP` (including their corresponding crisp values). Fuzzy point events are represented by using approximate values (triangular functions; refer to Figure 1.2b) with a predetermined *margin* (according to the context). For example, in healthcare applications, it may be interesting to store the fact that a patient recovered approximately on January 6, 1994. Additionally, by using fuzzy comparators (refer to Table 7.1), we can perform fuzzy queries. For example: "Retrieve all patients who recovered approximately on October 17, 2001."
2. **Duration events**: These events are associated with a specific time period in the database. In SQL2, temporal data types including this information are as follows: `INTERVAL` (a relative time duration, such as 2 days, 5 hours, or 30 seconds) and `PERIOD` (an anchored time duration with a fixed starting point, such as from August 3, 1970, to October 29, 1970).

   A new fuzzy `INTERVAL` type is unnecessary, because the underlying domain is the real numbers (just like fuzzy attributes Type 2). However, the measurement unit (years, hours, seconds, etc.) must be stored with each value or attribute according to the application. The measurement unit, of course, must be used in all computations.

   A new fuzzy `PERIOD` type is also unnecessary, because there are two alternative methods of representation. The first is to use fuzzy point events, that is, the fuzzy types defined previously but with other values, in particular the trapezoidal shape (refer to Figure 7.1). For example, a fuzzy period value may be represented by α = (May 11, 2003), β = (May 27, 2003), γ = (June 26, 2003), and δ = (June 29, 2003). The second method uses two fuzzy point events, representing the start and end time points. For example, we can store the fact that the most seriously ill stage of a patient's illness was from approximately April 14, 2005, until approximately July 25, 2005.

Fuzzy Temporal Databases

In this section, we study how to incorporate fuzzy time in RDB (tuple versioning). The extension for incorporating fuzzy time in object-oriented databases (attribute versioning) is easy (Elmasri & Navathe, 2000).

Basically, time may be interpreted to mean two different things (time dimensions):

- **Valid time** is the most natural interpretation and is the associated time when the event occurred, or the period during which the fact was considered to be true in the real world.
- **Transaction time** means that the associated time refers to the time when the information was actually stored in the database, that is, the value of the system time clock when the information was changed in the system.

Some applications need only one of the dimensions (valid time databases or transaction time databases), others need both time dimensions (bitemporal databases), and others need a user-defined interpretation (user-defined time databases).

We study only valid time databases, because transaction time databases need the exact system time. However, valid time databases need the time in which the fact was considered to be true in the real world. Sometimes, this time is not exactly known or is a vague time period.

In the same way as usual temporal RDB, valid time relations have two additional attributes whose data type is one of the previously defined fuzzy time types: `VST` (Valid Start Time) and `VET` (Valid End Time). These attributes in tuple t represent the fact that its information is only valid in the real world during the time period `[t.VST, t.VET]`.

A special value "`now`" is included in the domain of `VET` attribute. Value "`now`" in a tuple represents that this tuple is valid from `VST` till now; that is, this tuple represents the current values of the entity represented in the relation.

Example 7.20: Suppose a healthcare database storing the most seriously ill stage of certain illnesses. Some attributes of this relation would then be patient name, disease, treatment, and so forth, and attributes `VST` and `VET`. So, two valid tuples are

Dominique	Asthma	…	#(June 4)	$[July 6, July 6, July 6, July 15]
Dominique	Asthma	…	#(September 1)	now

The first tuple represents the fact that Dominique suffered an asthma attack on about June 4. The same symptoms continued until July 6. The patient then slowly began to get better until July 15. It should be noted that the VET attribute is a trapezoidal value with four values (like Figure 7.1). The second tuple represents the fact that Dominique suffered a new asthma attack on about September 1, and this is the current state.

*

It is worth mentioning that several tuples for the same entity may exist (like Dominique in the previous example). In temporal databases, this is solved by including VST (or VET) in the relation key. With fuzzy VST attributes, we must suppose that two tuples with different values in VST are different, even though they are very similar. The system must ensure that spurious or inconsistent tuples do not exist.

Fuzzy valid time relations keep track of the history of changes as they become effective in the real world, even if these changes occurred gradually or in an ill-known time. However, because updates, insertions, and deletions may be applied *proactively* (before the current time) or *retroactively* (after the current time), there is no record of the actual database state at any point in time. If it is needed, then we must use bitemporal relations. These include TST (Transaction Start Time) and TET (Transaction End Time) attributes, whose data type is typically crisp TIMESTAMP. In this case, we must include the TST attribute in the relation key, instead of VST.

FSQL Extends TSQL2 Language

So far, we have discussed how data models may be extended with fuzzy time by using temporal constructs. We now give a brief overview of how query operations need to be extended for fuzzy temporal querying. We briefly discuss an extension of TSQL2 language (Elmasri & Navathe, 2000; Snodgrass,

1995), which extends SQL for querying valid time, transaction time, and bitemporal relational databases.

Typical conditions for valid time databases are, for example, to select all tuple versions that were valid at a certain time point T or that were valid during a certain time period [T1,T2]. The values for the specified time point or time period may be fuzzy or crisp and are compared with the valid time period of each tuple version `t: [t.VST, t.VET]`. In Table 7.7 we extend the most common operations with the prefix **`F_`** (possibility versions) and **`NF_`** (necessity versions). Table 7.7 defines each one by using existing fuzzy comparators (Table 7.1). The last eight comparators are totally new. The comparators of type `MUCH_BEFORE/MUCH_AFTER` need a value *M* (according to the context) in order to consider two time values as being very separated.

Comparator `OVERLAPS` has also been called `INTERSECTS_WITH`. We can then use `F_INTERSECTS_WITH or NF_INTERSECTS_WITH`.

These new fuzzy temporal comparisons may be completed with the threshold clause in order to retrieve only values with a certain minimum threshold.

Example 7.21: "Select all patients (tuple versions) that were in a seriously ill stage (valid at any point) during around 1994 (in minimum degree 0.6)":

Table 7.7. The 18 fuzzy comparators for fuzzy time in FSQL, extending five TSQL2 comparators and eight new ones (possibility and necessity versions): `X` means "eXclusively"

Expression with Temporal Fuzzy Comparator	Equivalence
[t.VST,t.VET] F_INCLUDES [T1,T2]	T1 FGEQ t.VST AND T2 FLEQ t.VET
[t.VST,t.VET] F_INCLUDED_IN [T1,T2]	T1 FLEQ t.VST AND T2 FGEQ t.VET
[t.VST,t.VET] F_OVERLAPS [T1,T2]	T1 FLEQ t.VET AND T2 FGEQ t.VST
[t.VST,t.VET] F_BEFORE [T1,T2]	T1 FGEQ t.VET
[t.VST,t.VET] F_AFTER [T1,T2]	T2 FLEQ t.VST
[t.VST,t.VET] NF_INCLUDES [T1,T2]	T1 NFGEQ t.VST AND T2 NFLEQ t.VET
[t.VST,t.VET] NF_INCLUDED_IN [T1,T2]	T1 NFLEQ t.VST AND T2 NFGEQ t.VET
[t.VST,t.VET] NF_OVERLAPS [T1,T2]	T1 NFLEQ t.VET AND T2 NFGEQ t.VST
[t.VST,t.VET] NF_BEFORE [T1,T2]	T1 NFGEQ t.VET
[t.VST,t.VET] NF_AFTER [T1,T2]	T2 NFLEQ t.VST
[t.VST,t.VET] F_XBEFORE [T1,T2]	T1 FGT t.VET
[t.VST,t.VET] F_XAFTER [T1,T2]	T2 FLT t.VST
[t.VST,t.VET] F_MUCH_BEFORE [T1,T2]	T1 MGT t.VET
[t.VST,t.VET] F_MUCH_AFTER [T1,T2]	T2 MLT t.VST
[t.VST,t.VET] NF_XBEFORE [T1,T2]	T1 NFGT t.VET
[t.VST,t.VET] NF_XAFTER [T1,T2]	T2 NFLT t.VST
[t.VST,t.VET] NF_MUCH_BEFORE [T1,T2]	T1 NMGT t.VET
[t.VST,t.VET] NF_MUCH_AFTER [T1,T2]	T2 NMLT t.VST

```
SELECT Name, CDEG(*)
FROM Patients
WHERE [VST, VET] F_OVERLAPS [#(1994,1,1), #(1994,12,31)]
0.6;
                                                        *
```

In these definitions, we extend the TSQL2 language by using the FSQL comparators, in what may be called FTSQL2 language, for fuzzy temporal databases.

Other DML Statements: INSERT, DELETE, and UPDATE

The syntax of INSERT, DELETE, and UPDATE commands is very similar to SQL, modifying the expressions, the subqueries, and the conditions, by fuzzy expressions, fuzzy subqueries, and fuzzy conditions, respectively. In short, the FSQL modifications for each one are as follows:

1. **INSERT**: We will be able to use fuzzy expressions as values for the insertion as well as fuzzy subqueries, both in the values to insert and in the tables in which data are inserted.
2. **DELETE**: In the WHERE clause of this command, the same fuzzy conditions can be utilized as in the WHERE clause of the FSQL SELECT statement, explained previously.
3. **UPDATE**: The values to update can be fuzzy expressions or fuzzy subqueries. In addition, in the WHERE clause of this command, the same fuzzy conditions can be utilized as in the WHERE clause of the FSQL SELECT command, explained previously.

Example 7.22: According to Example 4.3 (Figure 4.5), the followings statements are valid:

```
INSERT INTO Employee (Employee_ID, Job, Experience, Ability,
   FDEGREE(Experience, Ability))
```

```
  VALUES (9999, "Chief", {1/Expert, 0.5/Normal}, $Skilled,
0.4);

INSERT INTO Employee SELECT * FROM Temporal
  WHERE Ability FEQ $[5,7,8,10] 0.75;

DELETE FROM Employee WHERE Ability FLEQ $Clumsy 0.5;

UPDATE Employee SET Salary = Salary + 10,
   Experience = {1/Expert, 0.7/Normal, 0.1/ Clumsy}
   WHERE Ability NFGT $Normal 0.75;
```

*

Some Useful Functions for Fuzzy Attributes

Sometimes we need to use a specific value of a fuzzy attribute, or we may want to change a fuzzy value. We can use the following functions, where **`fuzzy_attribute`** is the name of a fuzzy attribute, and **`attribute_list`** is a list of attributes separated by commas:

- **`FTYPE(fuzzy_attribute)`**: This function returns the type of the value in that fuzzy attribute, according to Tables 5.1 and 5.2. We can set this function as the default function in certain positions (see the "Modifying FSQL Options: ALTER FSQL and ALTER SESSION" section, later in this chapter). We can also, for example, discover how many approximate values are in the fuzzy attribute Type 2 `HEIGHT`, with the following query:

  ```
  SELECT COUNT(*) FROM PLAYERS
  WHERE FTYPE(HEIGHT) = 6;
  ```

- **`TO_CHAR(fuzzy_attribute)`**: This function returns the text that represents each value in the fuzzy attribute (for example, an approximate value is represented with the text "7 ± 2"). We can use this function in order to use this expression when the DBMS uses another value. We

can set this function as the default function in certain positions (see the "Modifying FSQL Options: ALTER FSQL and ALTER SESSION" section, later in this chapter).

- **FDEGREE(attribute_list)**: This function returns the fuzzy degree associated to the attribute or attributes given in its arguments. According to Example 4.2, the following query is possible:

```
SELECT * FROM PLAYERS
WHERE  QUALITY FEQ $Good AND FDEGREE(QUALITY) > .5;
```

- **FDEGROW(table)**: This function returns the fuzzy degree associated to the row (to the whole tuple). The argument of this function is the table name. However, this may be used as a pseudocolumn (like `ROWNUM` or `ROWID` in Oracle systems). This degree is the fuzzy degree of a fuzzy entity or a fuzzy relationship (refer to Chapter IV). According to Example 4.8, the following query is possible, where `$Very_High` is a qualifier defined in the FMB with the `CREATE QUALIFIER` statement (see the section on qualifiers later in this chapter):

```
SELECT  OWNER.Name,  PET.FDEGROW
FROM    PET, OWNER
WHERE   PET.Owner_ID  =  OWNER.Owner_ID
  AND   PET.FDEGROW   >  $Very_High;
```

- **MARGIN(fuzzy_attribute) / MUCH(fuzzy_attribute)**: These two functions return, respectively, the *margin* for approximate values and the *much* value *M* (refer to Chapter V) associated to the attribute given in its arguments (for fuzzy attributes Type 1 or 2). This may also be used with the dot notation: `fuzzy_attribute.MARGIN` and `fuzzy_attribute.MUCH`. These values may be used in expressions.

Some Useful Functions for Fuzzy Values

Sometimes we want to change a fuzzy value by modifying its membership function in order to tune the results. We can use the following functions, where **fuzzy_value** is a fuzzy value (including fuzzy attributes and fuzzy constants):

- **CARD(fuzzy_value)**: This function returns the cardinality of a fuzzy value, following Equation 1.25. For example, in order to know the rows with less fuzziness in an attribute than a fuzzy constant, a SELECT statement may include the next condition: CARD(Quality) < CARD(3+-2). It should be noted that CARD(3+-2) = 2 and generalizing CARD(3+- m) = m.
- **NORM(fuzzy_value)**: This function normalizes the fuzzy value, dividing the original membership function by the height of the fuzzy value (see Definitions 1.10 and 1.11). This function would be used with fuzzy attributes Type 3 or 4. Fuzzy attributes Type 2 are always normalized.
- **CONC_DILAT (fuzzy_value, p)**: If **p** > 1, then this function returns a *concentrated* version of the fuzzy value. The membership function of this version takes on relatively smaller values, being raised to power **p**. Usually, **p** = 2. If **p** ∈ (0, 1), then this function returns a *dilated* version of the fuzzy value. The membership function of this version takes on relatively greater values, being raised to power **p**. Usually, **p** = 0.5 (the square root).
- **MORE_CONTRAST(fuzzy_value, p)**: This is the contrast *intensification* function, and it returns the fuzzy value with the most contrast. The membership values lower than 0.5 are diminished, while the grades of membership above 0.5 are elevated. The operation is defined by (usually **p** = 2):

$$\text{MORE\_CONTRAST(A, p)}\ (x) = \begin{cases} 2^{p-1}\,\text{A}^{p}(x) & \text{if } \text{A}(x) \le 0.5 \\ 1 - 2^{p-1}(1-\text{A}(x))^{p} & \text{otherwise} \end{cases} \tag{7.1}$$

- **FUZZIFICATION(fuzzy_value, p)**: This function has a complementary effect to that of intensification. The operation is defined by (usually **p** = 2):

$$\texttt{FUZZIFICATION(A, p)}\ (x) = \begin{cases} \sqrt[p]{\texttt{A}(x)/2} & \text{if } \texttt{A}(x) \le 0.5 \\ 1-\sqrt[p]{(1-\texttt{A}(x))/2} & \text{otherwise} \end{cases} \tag{7.2}$$

- **UNION(fuzzy_values [, s_norm])**: This function returns the union (Definition 1.13) of the fuzzy values (separated with commas), with the s-norm (or t-conorm) indicated in the last argument. This last argument is optional, using by default the maximum s-norm. The s-norm is an alphanumeric value according to Table 1.2: "maximum," "sum-product," "drastic sum," "bounded sum *p*," "Einstein sum," and so forth. If the s-norm needs some argument *p*, then this is included after the name. Example: `UNION(Quality, $[4,5,6,7]`, "maximum").
- **INTERSECTION(fuzzy_values [, t_norm])**: This function returns the intersection (Definition 1.14) of the fuzzy values, with the t-norm indicated in the optional last argument. If this last argument does not appear, then the intersection uses the minimum t-norm. The t-norm is an alphanumeric value according to Table 1.1: "minimum," "product," "drastic product," "bounded product *p*," "Einstein product," "Hamacher product *p*," and so forth.

Functions **CONC_DILAT**, **MORE_CONTRAST**, and **FUZZIFICATION** are useful for implementing linguistic hedges such as *specially*, *very*, *slightly,* and *more or less*. Linguistic hedges may be regarded as operators that act on the fuzzy set representing the meaning of its operand. Perhaps it would be useful to express this just before a fuzzy expression. For example, the following fuzzy condition uses a linguistic hedge: `Salary FEQ $Very $Normal THOLD 0.75`.

Example 7.23: According to Example 4.2, the query "Select players who are *especially* Good" (with a 0.5 threshold) may be solved with the following FSQL statement:

```
SELECT * FROM PLAYERS
WHERE   QUALITY FEQ CON_DILAT($Good,2) 0.5;
                                                    *
```

Functions **UNION** and **INTERSECTION** are useful, for example, for shortening certain queries and for choosing the t-norm and the s-norm easily:

Example 7.24: "Select players who are Bad or Very Good" (using the sum-product s-norm, and a 0.5 threshold):

```
SELECT * FROM PLAYERS
WHERE  QUALITY FEQ UNION($Bad, $Very_Good, "sum-product") 0.5;
                                                    *
```

Remarks on Fuzzy Queries

Fuzzy queries reduce the risk of obtaining empty answers, because they use a finer scale of discrimination: the interval [0, 1] instead of the set {0, 1}. This means that tuples can be selected in queries where no answer would be obtained (or no tuple would be selected) in crisp mode. Sometimes, however, it might be that no element satisfies or complies with a fuzzy query. In order to solve this possible problem, the FSQL queries are especially flexible, and in every simple condition, we can modify the following four already defined important elements:

1. **Fuzzy comparators** (Table 7.1): Overcoat changing between possibility and necessity.
2. **Thresholds and qualifiers** (refer to the "Fulfillment Thresholds and Qualifiers" section, earlier in this chapter): In order to retrieve only the *most important* items. Other classic comparators may be used instead of the word THOLD to modify the query meaning, for example, to retrieve the *least important* items (using < or <=). We can also recover only the elements that comply with the condition to a specific degree using the comparator =.

3. **Fuzzy constants** (Table 7.2): If the right part of a simple condition is a fuzzy constant, this may be flexibly modified in order to better achieve the target.
4. **Functions** (refer to the preceding section): We can use functions to modify the fuzzy values of an attribute or to modify any fuzzy constant. For example, this allows us to represent linguistic hedges in FSQL queries.

Additionally, if the query uses fuzzy quantifiers (refer to the "Fuzzy Quantifiers in Queries" section), then we can use the `FUZZY` clause and the ρ argument to increase or decrease the restrictivity of the query. We can also use the `ONCEPERGROUP` option in fuzzy division queries (refer to the "Fuzzy Division Queries" section).

The weakening of queries has been studied in many publications (Andreasen & Pivert, 1994, 1995; Fuhr, 1990; Bosc, 1998). Another proposal in this sense (Ozawa & Yamada, 1994) is based on the fuzzy clustering of data. That is to say, when no data satisfy the query, the system shows the classes of data that are present and provides the class closest to the query. This system applies the algorithm of fuzzy clustering C-means (Bezdek, 1981) to classify data in various fuzzy groups, represented by linguistic labels defined by membership functions.

Fuzzy Comparisons

This section examines fuzzy comparisons and fuzzy comparators: definition, equivalences, restrictivity, types of fuzzy conditions, and the fuzzy comparison of crisp values.

Definition of Possibility and Necessity Comparators for Fuzzy Attributes Type 1 or 2

The fuzzy comparators (refer to Table 7.1) can be used to compare one fuzzy attribute with another attribute or with one constant (fuzzy or crisp).

In fuzzy attributes Type 2, we can store crisp values (as in the Type 1), but in addition, we can store fuzzy values such as the ones shown in Table 7.3. The most general case is the fuzzy trapezoid; therefore, this type of value includes all the others.

Here, we shall study the fuzzy comparators when two trapezoidal possibility distributions are compared, which is the most general case. These possibility distributions (refer to Figure 7.1) are denoted by `A = $[`α_A`, `β_A`, `γ_A`, `δ_A`]` and `B = $[`α_B`, `β_B`, `γ_B`, `δ_B`]`. We also use the `CDEG` function to express the compatibility degree of a fuzzy comparison, which is written as an argument.

It should be observed that in the necessity comparators, the possibility distribution in the left part of the comparison is denied (just as it is in Equation 7.4).

We define these fuzzy comparators by using the possibility and necessity measures (refer to Chapter I). Of course, fuzzy comparators may be defined by using other techniques, such as neural networks (Carrasco, Galindo, & Vila, 2001).

Possibly/Necessarily Fuzzy Equal and Fuzzy Different: FEQ/NFEQ and FDIF/NFDIF

The comparison of two possibility distributions A and B (trapezoidal) using FEQ, obtains a compatibility degree, denoted by `CDEG(A FEQ B)`:

$$\texttt{CDEG(A FEQ B)} = \sup_{x \in X} \left[\min \left(\texttt{A}(x), \texttt{B}(x) \right) \right] \tag{7.3}$$

where X is the underlying domain of `A` and `B`. It should be noted that `FEQ` uses the possibility measures (Equation 1.40). This fuzzy comparator is the only one that satisfies the commutative property (Equation 7.17).

The necessity comparator `NFEQ` describes the degree to which `A` is included in `B`. This degree is calculated for the following equation:

$$\texttt{CDEG(A NFEQ B)} = \inf_{x \in X} \left[\max \left(1 - \texttt{A}(x), \texttt{B}(x) \right) \right] \tag{7.4}$$

The possibly/necessarily fuzzy different comparators are defined by denying the previous definitions:

```
CDEG(A FDIF B)  =  1 - CDEG(A NFEQ B)   (7.5)
CDEG(A NFDIF B)  =  1 - CDEG(A FEQ B)   (7.6)
```

As the NFDIF comparator must be more restrictive than FDIF (see Table 7.8), NFDIF denies the comparison with FEQ, and FDIF denies the comparison with NFEQ. These equations are based on Equations 1.46 and 1.47.

Possibly/Necessarily Fuzzy Greater Than: FGT *and* NFGT

With other fuzzy comparators that are different from the fuzzy equality, the comparison is as if the possibility distribution of the right part of the comparison was modified and then Equation 7.3 or 7.4 was applied. With FGT/NFGT, the right part B is changed to a gamma function (Figure 1.5 with $a = \gamma_B$ and $b = \delta_B$). In other words, FGT/NFGT use the following equations:

$$\texttt{CDEG(A FGT B)} = \begin{cases} 1 & \text{if } \gamma_A \geq \delta_B \\ \dfrac{\delta_A - \gamma_B}{(\delta_B - \gamma_B) - (\gamma_A - \delta_A)} & \text{if } \gamma_A < \delta_B \text{ and } \delta_A > \gamma_B \\ 0 & \text{otherwise } (\delta_A \leq \gamma_B) \end{cases} \tag{7.7}$$

$$\texttt{CDEG(A NFGT B)} = \begin{cases} 1 & \text{if } \alpha_A \geq \delta_B \\ \dfrac{\beta_A - \gamma_B}{(\delta_B - \gamma_B) - (\alpha_A - \beta_A)} & \text{if } \alpha_A < \delta_B \text{ and } \beta_A > \gamma_B \\ 0 & \text{otherwise } (\beta_A \leq \gamma_B) \end{cases} \tag{7.8}$$

Possibly/Necessarily Fuzzy Greater or Equal: FGEQ *and* NFGEQ

With FGEQ/NFGEQ, the right part B is changed to a gamma function (Figure 1.5 with $a = \alpha_B$ and $b = \beta_B$) and then Equation 7.3 or 7.4 is applied. In other words, these fuzzy comparators use the following equations:

$$\text{CDEG(A FGEQ B)} = \begin{cases} 1 & \text{if } \gamma_A \geq \beta_B \\ \dfrac{\delta_A - \alpha_B}{(\beta_B - \alpha_B) - (\gamma_A - \delta_A)} & \text{if } \gamma_A < \beta_B \text{ and } \delta_A > \alpha_B \\ 0 & \text{otherwise } (\delta_A \leq \alpha_B) \end{cases} \tag{7.9}$$

$$\text{CDEG(A NFGEQ B)} = \begin{cases} 1 & \text{if } \alpha_A \geq \beta_B \\ \dfrac{\beta_A - \alpha_B}{(\beta_B - \alpha_B) - (\alpha_A - \beta_A)} & \text{if } \alpha_A < \beta_B \text{ and } \beta_A > \alpha_B \\ 0 & \text{otherwise } (\beta_A \leq \alpha_B) \end{cases} \tag{7.10}$$

Example 7.2 shows a query using FGEQ and its results.

Possibly/Necessarily Fuzzy Less Than: FLT *and* NFLT

With these fuzzy comparators, the right part B is changed to an L fuzzy set (Figure 1.4 with $a = \alpha_B$ and $b = \beta_B$) and then Equation 7.3 or 7.4 is applied. In other words, FLT/NFLT use the following equations:

$$\text{CDEG(A FLT B)} = \begin{cases} 1 & \text{if } \beta_A \leq \alpha_B \\ \dfrac{\alpha_A - \beta_B}{(\alpha_B - \beta_B) - (\beta_A - \alpha_A)} & \text{if } \beta_A > \alpha_B \text{ and } \alpha_A < \beta_B \\ 0 & \text{otherwise } (\alpha_A \geq \beta_B) \end{cases} \tag{7.11}$$

$$\texttt{CDEG(A NFLT B)} = \begin{cases} 1 & \text{if } \delta_A \le \alpha_B \\ \dfrac{\gamma_A - \beta_B}{(\alpha_B - \beta_B) - (\delta_A - \gamma_A)} & \text{if } \delta_A > \alpha_B \text{ and } \gamma_A < \beta_B \\ 0 & \text{otherwise } (\gamma_A \ge \beta_B) \end{cases} \tag{7.12}$$

Possibly/Necessarily Fuzzy Less or Equal: FLEQ *and* NFLEQ

With FLEQ/NFLEQ, the right part B is changed to an L fuzzy set (Figure 1.4 with $a = \gamma_B$ and $b = \delta_B$) and then Equation 7.3 or 7.4 is applied. In other words, these fuzzy comparators use the following equations:

$$\texttt{CDEG(A FLEQ B)} = \begin{cases} 1 & \text{if } \beta_A \le \gamma_B \\ \dfrac{\delta_B - \alpha_A}{(\beta_A - \alpha_A) - (\gamma_B - \delta_B)} & \text{if } \beta_A > \gamma_B \text{ and } \alpha_A < \delta_B \\ 0 & \text{otherwise } (\alpha_A \ge \delta_B) \end{cases} \tag{7.13}$$

$$\texttt{CDEG(A NFLEQ B)} = \begin{cases} 1 & \text{if } \beta_A \le \gamma_B \\ \dfrac{\gamma_A - \delta_B}{(\gamma_B - \delta_B) - (\delta_A - \gamma_A)} & \text{if } \delta_A > \gamma_B \text{ and } \gamma_A < \delta_B \\ 0 & \text{otherwise } (\gamma_A \ge \delta_B) \end{cases} \tag{7.14}$$

Possibly/Necessarily Much Greater Than: MGT *and* NMGT

In comparators that use the expression "Much" (much greater/less than), the distance *M* is used, which indicates the greatest minimum distance so that two values of that attribute are considered very separated. Additionally, with these comparators, the right part of the comparison is modified, adding *M* in MGT/NMGT and subtracting *M* in MLT/NMLT.

In order to obtain the compatibility degree with which A is possibly/necessarily much greater than B, FSQL uses the fuzzy set $ [α_B+M, β_B+M, γ_B+M, δ_B+M] and then applies Equation 7.7 or 7.8; that is, the following equations are used:

$$\text{CDEG(A MGT B)} = \begin{cases} 1 & \text{if } \gamma_A \geq \delta_B + M \\ \dfrac{\delta_A - \gamma_B - M}{(\delta_B - \gamma_B) - (\gamma_A - \delta_A)} & \text{if } \gamma_A < \delta_B + M \text{ and } \delta_A > \gamma_B + M \\ 0 & \text{otherwise } (\delta_A \leq \gamma_B + M) \end{cases} \tag{7.15}$$

$$\text{CDEG(A NMGT B)} = \begin{cases} 1 & \text{if } \alpha_A \geq \delta_B + M \\ \dfrac{\beta_A - \gamma_B - M}{(\delta_B - \gamma_B) - (\alpha_A - \beta_A)} & \text{if } \alpha_A < \delta_B + M \text{ and } \beta_A > \gamma_B + M \\ 0 & \text{otherwise } (\beta_A \leq \gamma_B + M) \end{cases} \tag{7.16}$$

Possibly/Necessarily Much Less Than: MLT *and* NMLT

With these fuzzy comparators, the right part B is changed to $ [α_B-M, β_B-M, γ_B-M, δ_B-M] and then Equation 7.11 or 7.12 is applied. In other words, MLT/NMLT use the following equations:

$$\text{CDEG(A MLT B)} = \begin{cases} 1 & \text{if } \beta_A \leq \alpha_B - M \\ \dfrac{\alpha_A - \beta_B + M}{(\alpha_B - \beta_B) - (\beta_A - \alpha_A)} & \text{if } \beta_A > \alpha_B - M \text{ and } \alpha_A < \beta_B - M \\ 0 & \text{otherwise } (\alpha_A \geq \beta_B - M) \end{cases} \tag{7.17}$$

$$\texttt{CDEG(A NMLT B)} = \begin{cases} 1 & \text{if } \delta_A \le \alpha_B - M \\ \dfrac{\gamma_A - \beta_B + M}{(\alpha_B - \beta_B) - (\delta_A - \gamma_A)} & \text{if } \delta_A > \alpha_B - M \text{ and } \gamma_A < \beta_B - M \\ 0 & \text{otherwise } (\gamma_A \ge \beta_B - M) \end{cases} \tag{7.18}$$

Equivalences Among Fuzzy Comparators and Exceptions to Its Definitions

Some fuzzy comparators are equivalent to others simply by exchanging the order of the compared values; that is, they obtain equal fulfillment degrees. For example, in crisp mode, the comparison `A > B` is equivalent to `B < A`.

The fuzzy comparators defined previously display the following **equivalences**:

```
A FEQ B ≡ B FEQ A                    (7.19)
A NFDIF B ≡ B NFDIF A                (7.20)
A FGT B ≡ B NFLEQ A                  (7.21)
A FLT B ≡ B NFGEQ A                  (7.22)
A FGEQ B ≡ B FLEQ A                  (7.23)
A NFGT B ≡ B NFLT A                  (7.24)
A NMGT B ≡ B NMLT A                  (7.25)
```

By definition, these equivalences are always satisfied. Nevertheless, some exceptions must be kept in mind for the sake of clarity. These **exceptions** are produced when a fuzzy constant `UNKNOWN`, `UNDEFINED`, or `NULL` is compared, using a necessity comparator. In these cases, a compatibility degree 0 is always returned, although in some specific comparisons the value is 1, following the comparator definition.

Example 7.25: In the following comparisons, the compatibility degree between both constants is 1, using the comparator definition:

```
UNDEFINED   NFEQ   #8
$[1,2,3,4]  NFEQ   UNKNOWN
[100,200]   NFLT   UNKNOWN
8           NFGT   NULL
```

Nevertheless, FSQL returns 0 in these comparisons, because it is the value that we hope to obtain. For example, in the second case, the necessity of any fuzzy value to be UNKNOWN is 1 (Equation 7.4), but intuitively the users hope that such tuples will not be obtained. It should be noted that all fuzzy values are included in the UNKNOWN fuzzy set, but if the query makes a join, then such matches are not interesting.

*

These exceptions mean that the returned value is the intuitive value. In addition, the necessity comparators clearly indicate that the query wants to recover *a few interesting* tuples. Thus, the selection of tuples with UNKNOWN values does not seem logical. On the other hand, if the query wishes to obtain all the tuples that possibly comply with the condition, then it should use a possibility comparator and not a necessity comparator.

Fuzzy Comparators Restrictivity

There are comparators whose results include others. For example, in crisp mode, the result of the comparator >= includes the result of >. We can then say that the comparator > is more restrictive than >=. This means that more restrictive comparators will recover a smaller or equal number of tuples, and

Table 7.8. Fuzzy comparators restrictivity by families

Family	Fuzzy Comparators, from greater to smaller restrictivity
Fuzzy Equal	NFEQ > FEQ
Fuzzy Different	NFDIF > FDIF
Fuzzy Greater	NFGT > FGT > NFGEQ > FGEQ
Fuzzy Less	NFLT > FLT > NFLEQ > FLEQ
Much Greater	NMGT > MGT
Much Less	NMLT > MLT

these recovered tuples will never have a greater fulfillment degree than with less restrictive comparators.

In Table 7.8, the restrictivity order can be seen for fuzzy comparators by families. It should be observed that any necessity comparator is more restrictive than its corresponding possibility comparator.

Fuzzy Comparators FEQ and FDIF for Fuzzy Attributes Type 3

Fuzzy attributes Type 3 and 4 use only FEQ and FDIF, because a relation of order does not exist (refer to Chapter IV). For example, we are not able to say whether the value "blond" is greater than the value "dark-haired," but we can measure their similarity.

When we compare two simple values of Type 3, the returned value is the value stored in their similarity relation, supposing that both values are normalized (Definition 1.11); that is, their possibility degree is 1.

The normal (and desirable) thing is that both the simple values and the possibility distributions on fuzzy attributes Type 3 are normalized; that is, their possibility degree is 1 in at least some of the components. Nevertheless, this is not obligatory.

Let us suppose that we want to compare two possibility distributions, F and X, on the linguistic labels of a fuzzy attribute Type 3:

$$\texttt{F FEQ X} \tag{7.26}$$

where

$$\texttt{F} = \{\texttt{FP}_i / \text{label}\texttt{F}_i\} \quad \text{with} \quad i = 1,2,...,\texttt{LEN}_\texttt{F} \tag{7.27}$$

$$\texttt{X} = \{\texttt{XP}_j / \text{label}\texttt{X}_j\} \quad \text{with} \quad j = 1,2,...,\texttt{LEN}_\texttt{X} \tag{7.28}$$

$\text{label}\texttt{F}_i$ and $\text{label}\texttt{X}_j$ being linguistic labels, which belong to the same attribute and therefore can be compared with its similarity relation. The values $\texttt{FP}_i$ and $\texttt{XP}_j$ are the possibility degrees, in [0, 1], associated to these labels, respectively.

$\texttt{LEN}_F$ and $\texttt{LEN}_X$ indicate the number of pairs {degree/label} of the possibility distributions F and X, respectively, with $\texttt{LEN}_F \geq 1$ and $\texttt{LEN}_X \geq 1$.

The compatibility degree of F and X (the fulfillment degree of the condition in 7.26) is subsequently computed by

$$\texttt{CDEG(F FEQ X)} = \max_{\substack{i=1,2,...,\texttt{LEN}_F \\ j=1,2,...,\texttt{LEN}_X}} \{\mu_R(\texttt{labelF}_i, \texttt{labelX}_j) * \texttt{FP}_i * \texttt{XP}_j\} \quad (7.29)$$

where $\mu_R(\texttt{labelF}_i, \texttt{labelX}_j)$ express the similarity degree between both labels (refer to Chapter V).

The previous equation is simplified when the comparison is performed directly on a label ($\texttt{label} = \texttt{labelX}_1$ with $\texttt{XP}_1 = 1$) :

$$\texttt{CDEG(F FEQ \$label)} = \max_{i=1,2,...,\texttt{LEN}_F} \{\mu_R(\texttt{labelF}_i, \texttt{label}) * \texttt{FP}_i\} \quad (7.30)$$

If for a couple of labels their similarity degree is not defined in the FMB, it is supposed to be 0.

The "fuzzy different" comparator, FDIF, is defined by denying the comparator FEQ in Equation 7.29:

$$\texttt{CDEG(A FDIF B)} = 1 - \texttt{CDEG(A FEQ B)} \quad (7.31)$$

Fuzzy Comparator FEQ for Fuzzy Attributes Type 4

Suppose that we want to compare two possibility distributions, F and X, on the linguistic labels of a fuzzy attribute Type 4, F FEQ X, where F and X are defined with Equations 7.27 and 7.28, respectively. The compatibility degree of F and X is then computed by Equation 7.29, where $\mu_R(\texttt{labelF}_i, \texttt{labelX}_j)$ is 1 if $\texttt{labelF}_i = \texttt{labelX}_j$, and 0 if $\texttt{labelF}_i \neq \texttt{labelX}_j$.

Similarly, FDIF uses Equation 7.31. In fuzzy attributes Type 3 or 4, FDIF has a definition different than the FDIF of fuzzy attributes Type 1 or 2 (Equation 7.5).

Types of Fuzzy Simple Conditions, With and Without Arithmetic Expressions

We can conclude that there are 13 basic types of simple comparisons or fuzzy simple conditions (without external arithmetic expressions). These conditions can be used with other conditions (fuzzy or not) by means of the logical operators (`NOT`, `AND`, and `OR`) to form more complex fuzzy conditions.

Let `<fcol>` and `<fcol2>` be two fuzzy columns (fuzzy attributes) in SQL format (with or without scheme and table) and `<FCOMP>` be a fuzzy comparator (refer to Table 7.1). The types of fuzzy simple conditions are then as follows:

1. Comparison of a fuzzy attribute Type 1 or 2 with the fuzzy constant "approximately n" using the *margin* of the FMB (Table 7.2):

```
<fcol> <FCOMP> #n
```

2. Comparison of a fuzzy attribute Type 1 or 2 with the fuzzy value "approximately n" with an explicit margin m (Table 7.2):

```
<fcol> <FCOMP> n+-m
```

3. Comparison of a fuzzy attribute Type 1 or 2 with an interval (Table 7.2):

```
<fcol> <FCOMP> [n,m]
```

4. Comparison of a fuzzy attribute Type 1 or 2 with a trapezoidal fuzzy constant (Table 7.2):

```
<fcol> <FCOMP> $[α, β, γ, δ]
```

5. Comparison of a fuzzy attribute Type 1, 2, 3, or 4 with a linguistic label, defined in the FMB (Table 7.2):

```
<fcol> <FCOMP> $label
```

6. Comparison of a fuzzy attribute Type 3 or 4 with a possibility distribution on labels (Table 7.2):

```
<fcol> <FCOMP> {P1/L1, P2/L2, …, Pn/Ln}
```

7. Comparison of a fuzzy attribute Type 3 or 4 with a possibility distribution on labels with possibility degree 1 for all of them (Table 7.2):

```
<fcol> <FCOMP> {L1, L2, …, Ln}
```

8. Comparison of a fuzzy attribute Type 1 or 2 with a possibility distribution on numbers (Table 7.2):

```
<fcol> <FCOMP> {P1/N1, P2/N2, …, Pn/Nn}
```

9. Comparison of a fuzzy attribute Type 1 or 2 with a possibility distribution on numbers with possibility degree 1 for all of them (Table 7.2):

```
<fcol> <FCOMP> {N1, N2, …, Nn}
```

10. Comparison of a fuzzy attribute Type 1, 2, 3, or 4 with an expression:

```
<fcol> <FCOMP> (<expression>)
```

If the expression includes fuzzy values (or fuzzy columns), then the included operations must be defined (refer to Chapter I). The expression should appear between parentheses. Otherwise, the operations will be considered as external to the fuzzy comparison; that is, the operations will use the compatibility degree of `<fcol> <FCOMP> F`, where `F` is the first value in the expression. In other words, fuzzy comparators have a greater precedence. The expression may use the functions in the

"Some Useful Functions for Fuzzy Values" section, earlier in this chapter.

11. Comparison of any type of fuzzy attribute with others, which must be compatible with each other:

```
<fcol>  <FCOMP>  <fcol2>
```

Fuzzy attributes Type 1 and Type 2 can be compared with each other, but none of them can be compared with a Type 3 or 4. Fuzzy attributes Type 3 and Type 4 can only be compared with other Type 3 and 4, respectively, and both fuzzy columns must be explicitly declared as compatible (refer to Chapter V).

12. Comparison of any type of fuzzy attribute with the fuzzy constant UNKNOWN, UNDEFINED, or NULL, using FEQ or NFDIF:

```
<fcol> { FEQ   } { UNKNOWN   }
       { NFDIF } { UNDEFINED }
                 { NULL      }
```

In this type of comparison, another fuzzy comparator different of FEQ cannot be employed, and the column <fcol> cannot be crisp or fuzzy Type 1.

13. Comparison of any type of fuzzy attribute with the fuzzy constant UNKNOWN, UNDEFINED, or NULL, using IS:

```
<fcol> IS [NOT] { UNKNOWN   }
                { UNDEFINED }
                { NULL      }
```

If <fcol> is Type 1, then only NULL can be used and will be considered and treated as the NULL of the DBMS.

At the end of all these fuzzy conditions, with the exception of the last one, a threshold may be added (`THOLD`), so that the condition established will only be true if its fulfillment degree surpasses this threshold.

The fuzzy comparisons (except `IS`) can be preceded or followed by arithmetic expressions that operate with the fuzzy comparison, so that these are considered as external to the fuzzy comparison (except when parentheses are used).

It should be noted that in a fuzzy comparison, the left part must be a fuzzy column, because the comparison operations need to know the context (*margin* in approximate values, value *M,* similarity relation, linguistic labels, etc.).

If a fuzzy attribute Type 2, 3, or 4 appears in an expression (with the exception of case 11), we will say that it is in an unusual or special position. In this case, the default processing for this attribute can be configured for the user, as explained in the "Modifying FSQL Options: ALTER FSQL and ALTER SESSION" section, later in this chapter.

Next, we present a series of examples that show the enormous variety of types of fuzzy comparisons with the arithmetic expressions that can be performed.

Example 7.26: Let `Salary` be a fuzzy attribute Type 1 or 2 and `Commission` a numeric crisp attribute or a fuzzy attribute Type 1. The following fuzzy comparisons and their meanings can then be observed:

1. `0.5 * Salary FGT (SQRT(Commission)+3/4) THOLD 0.25`

   This fuzzy comparison expresses that $(0.5 * \chi) \geq 0.25$, where χ is `CDEG(Salary FGT (SQRT(Commission)+3/4))`.

2. `0.5 * Salary FGT SQRT(Commission)+3/4 THOLD 0.25`

   This fuzzy comparison expresses that $(0.5 * \chi + 3/4) \geq 0.25$, where χ is `CDEG(Salary FGT SQRT(Commission))`.

3. `0.35 * SQRT(Commission) * Salary NFGEQ $[2,3,4,5] = 35`

This fuzzy comparison expresses that $(0.35 * \sqrt{\texttt{Commission}} * \chi) = 35$, where χ is `CDEG(Salary NFGEQ $[2,3,4,5])`.

4. `Salary NFEQ (#2 * Salary - $[2,3,4,5]) < .5`

This fuzzy comparison expresses that $\chi < 0.5$, where χ is `CDEG(Salary NFEQ ψ)`, and ψ is the fuzzy expression `(#2 * Salary - $[2,3,4,5])`. This, of course, implies the definition of fuzzy arithmetic operations (refer to Chapter I).

*

Comparison of Crisp Values Using Fuzzy Comparators

By using fuzzy comparators, FSQL enables two fuzzy attributes Type 1 (crisp) or a fuzzy attribute Type 1 with a crisp expression (including nonfuzzy attributes) to be compared. This could be considered erroneous, because from a certain point of view, it is pointless to diffusely compare two crisp values. In this case, the crisp (or classical) comparators should be used.

Nevertheless, FSQL permits the fuzzy comparison of crisp values, giving it a shade that can prove very useful and that crisp comparators cannot manage.

Hence, when we compare a fuzzy attribute Type 1 with a crisp value by means of a fuzzy comparator, this *blurs* (*fuzzifies*) the value situated in the right part of the comparison, converting it to an approximate value (refer to Figure 1.2b), with the *margin* belonging to the attribute situated to the left part of the comparison. Consequently, there must always be a fuzzy attribute in the left part of any fuzzy comparator. If the user does not want to blur that value, then a crisp comparator should be utilized.

Example 7.27: If `<fcol_T1>` is a fuzzy attribute Type 1, then the following two fuzzy comparisons are equivalent:

```
<fcol_T1> FGT 6
<fcol_T1> FGT #6
```

*

Similarly, when a fuzzy attribute Type 1 is compared with an interval `[n, m]`, utilizing a fuzzy comparator, this interval is also *blurred* (*fuzzified*), becoming the trapezoidal `$[n-`*`margin`*`, n, m, m+`*`margin`*`]` (refer to Figure 7.1). If the user does not desire it, then s/he must use a crisp comparator such as `BETWEEN n AND m`.

Comparators `MGT/NMGT` and `MLT/NMLT` ("Much Greater/Less Than") also blur the crisp value situated in the left part of the comparison (with the same *margin*). This is because these comparators are fuzzy *per se*, that is to say, the word "Much" includes an extra fuzzy shade that other comparators do not have. It should be noted that these comparators do not come from crisp comparators but are fuzzy by themselves.

Hence, by blurring (or fuzzifying) both values, these comparators are able to achieve a more gradual comparison; that is, values that would otherwise obtain a 0 degree now obtain a degree in the interval (0,0.5). The value 0.5 is due to the fact that both values are blurred (or fuzzified) with the same *margin*.

Crisp Comparators in Fuzzy Attributes

Crisp or classic comparators may be used in comparisons with fuzzy attributes. These comparisons return classic values: true, false, or unknown-maybe (see tri-valued logic in Table 2.1). The unknown or maybe value is returned only when values `UNKNOWN` or `NULL` appear. If the `UNDEFINED` value appears, then the comparison returns false. In any other case, FSQL uses the following equations, where `A` and `B` are two fuzzy values, and μ_A and μ_B are the membership functions of `A` and `B`, respectively, in the universe X:

- `A = B` returns
  - True if $\forall x \in X, \mu_A(x) = \mu_B(x)$ (Definition 1.2), and
  - False if $\exists x \in X, \mu_A(x) \neq \mu_B(x)$.
- `A > B` returns
  - True if $\max\{\text{Supp}(A)\} > \max\{\text{Supp}(B)\}$ (Supp is defined in Defini-

tion 1.4), and
  - False if $\max\{\text{Supp}(\texttt{A})\} \leq \max\{\text{Supp}(\texttt{B})\}$.
- `A >= B` returns
  - True if $\max\{\text{Supp}(\texttt{A})\} \geq \max\{\text{Supp}(\texttt{B})\}$, and
  - False if $\max\{\text{Supp}(\texttt{A})\} < \max\{\text{Supp}(\texttt{B})\}$.
- `A < B` returns
  - True if $\min\{\text{Supp}(\texttt{A})\} < \min\{\text{Supp}(\texttt{B})\}$, and
  - False if $\min\{\text{Supp}(\texttt{A})\} \geq \min\{\text{Supp}(\texttt{B})\}$.
- `A <= B` returns
  - True if $\min\{\text{Supp}(\texttt{A})\} \leq \min\{\text{Supp}(\texttt{B})\}$, and
  - False if $\min\{\text{Supp}(\texttt{A})\} > \min\{\text{Supp}(\texttt{B})\}$.

It is important to check that these definitions do not satisfy some basic properties. For example, if `A > B` is true, then this does not imply that `A <= B` is false.

`INCL` and `FINCL` Comparators for Fuzzy Attributes Types 1, 2, 3, and 4

The inclusion comparator `INCL` implements the $\subseteq$ operator. In this way, `A INCL B` examines if $\texttt{A} \subseteq \texttt{B}$ (Definition 1.3). This comparison returns a crisp value of the tri-valued logic:

$$\texttt{A INCL B} = \begin{cases} \texttt{NULL} & \text{if A or B are NULL} \\ \texttt{TRUE} & \text{if } \texttt{A}(x) \leq \texttt{B}(x), \forall x \\ \texttt{FALSE} & \text{otherwise} \end{cases} \tag{7.32}$$

The fuzzy inclusion comparator `FINCL` defines a degree of subsethood. If `CDEG(A FINCL B)` = 1, then `A` is totally included in `B`. In the other extreme, if `CDEG(A FINCL B)` = 0, then `A` is not included in `B` at all. This degree is computed by

$$\texttt{CDEG(A FINCL B)} = \frac{Card(\texttt{A}) - Card(\texttt{A} \cap \texttt{B})}{Card(\texttt{A})} \tag{7.33}$$

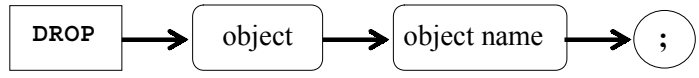
where *Card* is the cardinality function (Definition 1.12) of the membership function. It should be noted that in fuzzy attributes Type 1 or 2, the cardinality is the area of the membership function, and in fuzzy attributes Type 3 or 4, the cardinality is expressed in Equation 1.25. Similarly, we can use other expressions for this fuzzy inclusion, such as the approach presented in Kosko (1992). For the sake of simplicity, the intersection of A and B may use the minimum t-norm.

DDL of FSQL: CREATE, ALTER, and DROP

The **DDL** (data definition language) of FSQL includes the modification of certain statements and some new statements of three families: CREATE, DROP, and ALTER. These statements are applied to the following objects of an FRDB:

1. Object `TABLE`: Fuzzy relations or fuzzy tables (with fuzzy attributes). The group of statements formed by `CREATE TABLE`, `ALTER TABLE`, and `DROP TABLE` already exists in SQL standard. Here, we have expanded their syntax so that they enable the needs of an FRDB to be expressed.
2. Object `FDATATYPE`: This object allows the definition of specific fuzzy data types identified with one name. These names may be used wherever a fuzzy data type may be used.
3. Object `VIEW` or `MATERIALIZED VIEW (SNAPSHOP)`: The statements `CREATE`, `ALTER`, and `DROP` applied to these objects already exist in SQL, but in FSQL fuzzy queries are allowed with the `SELECT` of FSQL (refer to the "Novelties in the Fuzzy SELECT of FSQL" section, earlier in this chapter)
4. Object `LABEL` (refer to the "Novelties in the Fuzzy SELECT of FSQL" section, earlier in this chapter): This object includes fuzzy labels of

Figure 7.3. DROP statement syntax in SQL and FSQL



attributes Types 1, 2, and 4. If it belongs to Type 1 or 2, then it is associated to possibility distributions. This object is exclusive of FSQL.

5. Object NEARNESS (refer to the "Novelties in the Fuzzy SELECT of FSQL" section, earlier in this chapter): This object represents the similarity relationships of fuzzy attributes Type 3. The CREATE NEARNESS statement implies the definition of labels for fuzzy attributes Type 3 and the similarity relation between them. This object is exclusive of FSQL.
6. Object QUALIFIER (refer to the "Fulfillment Thresholds and Qualifiers" section, earlier in this chapter): This object represents a constant inside the context of the degrees of an attribute. This object is exclusive of FSQL.
7. Object QUANTIFIER (refer to the "Fuzzy Quantifiers in Queries" section, earlier in this chapter): This object represents fuzzy quantifiers inside the context of attributes, tables, or the system. This object is exclusive of FSQL.
8. Object MEANING (refer to Chapter IV): This object represents the meanings or significances for some of the degrees with which the FRDB works. This object is exclusive of FSQL.

FSQL includes statements on another two objects: FSQL and SESSION. These statements specify or modify any of the conditions or parameters that affect all their connections to the database (forever) or only the current session (or connection). These statements are shown in the "Modifying FSQL Options: ALTER FSQL and ALTER SESSION" section, later in this chapter.

Other clauses or statements for other objects have not been defined here, as these definitions are the same as that of SQL, or very similar to it or to clauses that have already been defined.

The CREATE and ALTER statements for the same object always have a similar syntax, and the DROP statement has a similar syntax for all objects (see Figure 7.3). In the following section, we focus on explaining the new clauses or

new incorporations to the CREATE statements.

TABLE and FDATATYPE

The CREATE and ALTER TABLE are the DDL statements that include more innovations. We discuss these new incorporations in the following subsections.

Fuzzy Data Types

New types of data have been added. For the sake of convenience in the learning, each new data type has two names (two reserved words). Some of them may have arguments. These are the new types and their format:

1. **FTYPE1 or CRISP**: This is used to declare a fuzzy attribute Type 1. It has two arguments, placed between parentheses and separated by a comma, which indicate the context values for the new fuzzy attribute: values MARGEN and MUCH (*M*) of the table FUZZY_APPROX_MUCH of the FMB (refer to Chapter V). Note that if we change the *margin* value, then the DBMS must change all the approximate values of type 6 (but not of type 8; refer to Table 5.1).

   Subsequently, the base type of the attribute can be expressed optionally, that is to say, the underlying domain, the real domain of the attribute Type 1. By default, this domain will be considered as the NUMBER data type of Oracle. Of course, the underlying domain cannot be a fuzzy data type.

   Optionally, the word DOMAIN followed by the name of another attribute (which can be from the same table) can be added to the definition of this type of attribute in order to indicate that the attribute being defined will be compatible with the attribute indicated after DOMAIN and will therefore be able to take its values, labels, and so forth and all the definitions of the FMB. The clause DOMAIN inserts a row into the table FUZZY_COMPATIBLE_COL (refer to Chapter V).
2. **FTYPE2 or POSSIBILISTIC**: This is used to declare a fuzzy attribute Type 2. It has the same two arguments and uses the same format as FTYPE1.
3. **FTYPE3 or SCALAR**: This is used to declare a fuzzy attribute Type 3.

It has an optional argument indicating the maximum number of data for the values of this attribute, that is, the maximum number of elements in the possibility distributions on scalars: value of LEN attribute in table FUZZY_COL_LIST (refer to Chapter V). The minimum value is the default value, 1.

Optionally, the word DOMAIN followed by the name of another attribute (which can be from the same table) can be added to the definition of this type of attribute in order to indicate that the attribute being defined will be compatible with the attribute indicated after DOMAIN and will therefore be able to take its values. The clause DOMAIN inserts a row into the table FUZZY_COMPATIBLE_COL (refer to Chapter V).

4. **FTYPE4 or NONSIMILAR**: This is used to declare a fuzzy attribute Type 4. The format of this fuzzy attribute is equal to the FTYPE3 attribute.
5. **FUZZY or FUZZY DEGREE**: This is used to declare a fuzzy degree with its own meaning (refer to Chapter IV). The user should choose a good name that indicates something about its meaning. Optionally, after this data type, the user can specify the meaning of this fuzzy degree with a number or some text (see the section on meaning later in this chapter). If this meaning is included, then it must exist in the FMB. This kind of degree does not have a default meaning, as the meaning should be expressed in the name.
6. **FUZZY_DATE, FUZZY_TIME, and FUZZY_TIMESTAMP**: These data types are used to store fuzzy time values (see the section on fuzzy time earlier in this chapter). They are treated as Type 2 attributes, where the underlying domain is the domain of DATE (specifying year, month, and day), TIME (specifying hour, minute, and second), and TIMESTAMP (specifying a DATE and TIME combination), respectively. Some DBMSs, like Oracle, use only the TIMESTAMP data type (called DATE here).

The UM Clause

In the definition of an attribute of the previous data types, the UM clause may be used to set the Unit of Measurement. This value is alphanumeric and should end with the abbreviation in parentheses.

Example 7.28: Let us see the declaration of a fuzzy attribute Type 2:

```
Height FTYPE2 (3,7) NUMBER(4,1)
   UM 'Centimeters (cm.)'
   DEFAULT UNKNOWN
```

*

In fuzzy time data types, the unit of measurement is the unit of measurement of the *margin* and MUCH (*M*) values. If this clause appears, then FSQL gives a numeric data type to these attributes in the table FUZZY_APPROX_MUCH (refer to Chapter V); otherwise, it gives a crisp date-and-time data type (in another dual table).

Specific Fuzzy Data Types: The Object FDATATYPE

The FSQL user may create new and specific fuzzy data types by using the following statement:

```
CREATE FDATATYPE FTYPE_Name AS <definition>;
```

where `FTYPE_Name` is the name of the new fuzzy data type and `<definition>` is its definition, following the syntax previously exposed. After the specific fuzzy data type has been created, other objects (such as labels, qualifiers, quantifiers, etc.) may be created and associated to the new specific fuzzy data type.

Example 7.29: Let us create a specific fuzzy datatype for a fuzzy area:

```
CREATE FDATATYPE Fuzzy_Area AS
   FTYPE2 (3,7) NUMBER(4,1) UM 'm^2'
   DEFAULT UNKNOWN;
```

Figure 7.4. Structure for defining a fuzzy degree associated to one attribute

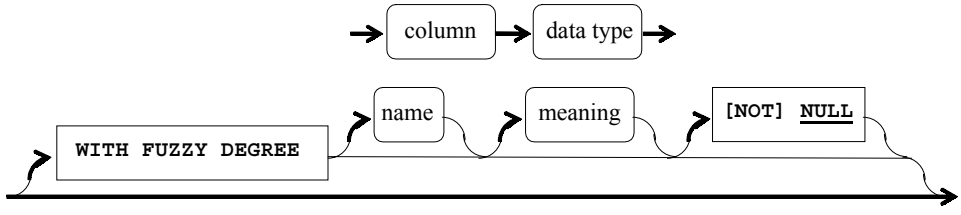
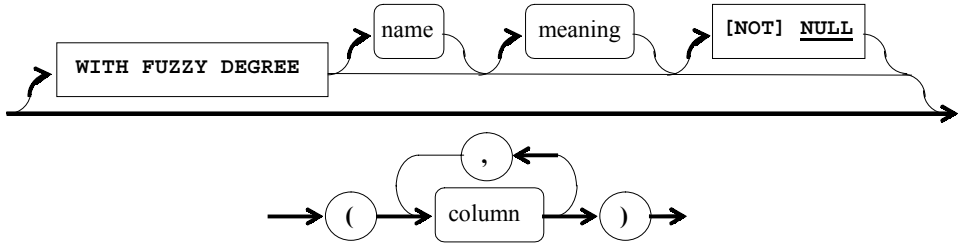


Figure 7.5. Structure for defining a fuzzy degree associated to one or some attributes



After this statement we can associate other objects to the new specific fuzzy data type. For example, in order to associate a label to this new specific fuzzy data type, one possible statement is as follows (see syntax in the "Label" section, later in this chapter):

```
CREATE LABEL Fuzzy_Area.$Very_Big VALUES 60, 70, 80, 90;
                                                        *
```

Associated Fuzzy Degrees

In Chapter V, you see four types of fuzzy degrees. These degrees have default names, but in the declaration of these attributes the user may call them anything.

FSQL language has functions in order to access these degrees (see the section "Some Useful Functions for Fuzzy Attributes," earlier in this chapter).

1. **Fuzzy degree associated to each value of an attribute**: In the declaration of an attribute, after its data type we can use the words **`WITH FUZZY DEGREE`**. This clause creates a fuzzy degree associated to the attribute of the declaration. If **F** is the name of the attribute, then the name of the degree is, by default, **DegreeF**, but this name can be set just after the words **`WITH FUZZY DEGREE`**. Optionally, the user can specify the meaning of this fuzzy degree with a number or some text (refer to Chapter V). If this meaning is included, it must exist in the FMB. The default meaning is "fulfillment." In addition, the optional clause `NOT NULL` prohibits this value in the fuzzy degree (default is the `NULL` clause). The format is expressed in Figure 7.4. However, this type of degree can also be declared by using the following format (see Figure 7.5).
2. **Fuzzy degree associated to values of some attributes**: As part of the table definition, out of the definition of an individual column or attribute, we can use the words **`WITH FUZZY DEGREE`**, followed by all the columns in parentheses. Optionally, we can include the name, meaning, and the clause `NOT NULL`, which prohibits this value in the fuzzy degree. If this meaning is included, then it must exist in the FMB. The default name is **DegreeF**, where **F** is the acronym of all the associated attributes. The default meaning is "fulfillment." The format is depicted in Figure 7.5.
3. **Fuzzy degree associated to the whole tuple (or row)**: As part of the table definition, out of the definition of an individual column or attribute, we can use the words **`TABLE WITH FUZZY DEGREE`** in order to define this degree. After this clause, we can optionally include the name, meaning, and the clause `NOT NULL`. If this meaning is included, then it must exist in the FMB. The default name is **DegreeF**, where **F** is the table name. The default meaning is "fulfillment."

Subqueries, Constants, Conditions, and Fuzzy Expressions

When these elements form part of a `CREATE TABLE` statement, the user can utilize their fuzzy versions (just as they were presented in the "Novelties in the

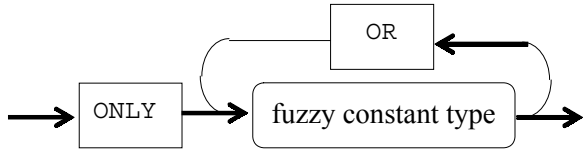
Fuzzy SELECT of FSQL" section, earlier in this chapter). For example, the DEFAULT clause lets you specify a value to be assigned to the column if a subsequent INSERT statement omits a value for the column. In the DEFAULT clause you can write a fuzzy expression (for example, using fuzzy constants from Table 7.2).

Column Constraints

You can define constraints as part of the definition of an individual column or attribute. The classic constraints are [NOT] NULL, UNIQUE, PRIMARY KEY, CHECK, and REFERENCES. In addition, other constraints in FSQL can be controlled:

1. **NOT clause**: This prohibits a database value from being a fuzzy value or type of fuzzy value. These fuzzy constant types are the crisp type and those expressed in Table 7.2. UNKNOWN, UNDEFINED, and NULL prohibit these values in fuzzy attribute Types 2, 3, and 4. Word LABEL prohibits values of linguistic label type. Similarly, CRISP, TRAPEZOID, INTERVAL, APPROX (with *margin* in the FMB), and APPROXM (with an explicit *margin*) prohibit this type of value. Finally, the word DISTRIBUTIONS prohibits noncontinuous possibility distributions in fuzzy attributes Types 2, 3, or 4. If the word NOT does not appear, then these values are allowed explicitly (this is the default option). The format is as follows, and the relevant data types are expressed on the right:

Figure 7.6. Structure for the ONLY clause



	UNKNOWN	(2, 3 and 4)
	UNDEFINED	(2, 3 and 4)
	NULL	(all datatypes)
	LABEL	(2, 3 and 4)
	CRISP	(2)
	TRAPEZOID	(2)
[NOT]	INTERVAL	(2)
	APPROX	(2)
	APPROXM	(2)
	DISTRIBUTIONS	(2, 3 and 4)

2. **`ONLY` clause**: This explicitly allows only a database value to be a certain value or type of value. This clause uses the same reserved words as the previous clause (the fuzzy constants types), but now these words may be linked with the word `OR`: `UNKNOWN`, `UNDEFINED`, `NULL`, `LABEL`, `CRISP`, `TRAPEZOID`, `INTERVAL`, `APPROX`, `APPROXM`, and `DISTRIBUTIONS`. The format of this clause is shown in Figure 7.6.

   Note that using `ONLY CRISP OR NULL` over a fuzzy attribute Type 2 is similar to using a fuzzy attribute Type 1.
3. **`CHECK` clause**: A **check constraint** in an FSQL statement lets you specify a fuzzy condition that each row in the table must satisfy.

Example 7.30: Let us create the `HOUSING` relation for an estate agency:

```
CREATE TABLE HOUSING (
  CODE#     NUMBER(9) NOT NULL PRIMARY KEY,
  OWNER#    NUMBER(9) NOT NULL,
  ADDRESS   VARCHAR2(60),
  AREA      FTYPE1 (15,25) NUMBER(4)
              UM 'Square metres (m2.)'
```

```
                WITH FUZZY DEGREE UNCERTAINTY
                CONSTRAINT NOT_NULL_AREA  NOT NULL,
  PRICE     FTYPE2 (500,3000) NUMBER(6) DEFAULT UNKNOWN
              CONSTRAINT NOT_NULL_PRICE       NOT NULL
            CONSTRAINT NOT_UNDEFINED_PRICE NOT UN-
DEFINED,
  DISTRICT FTYPE3 (3) DEFAULT UNKNOWN
      CONSTRAINT ONLY_DISTRIB_DISTRICT ONLY DISTRIBUTIONS
              CONSTRAINT NOT_NULL_DISTRICT      NOT NULL
             CONSTRAINT NOT_UNDEF_DISTRICT   NOT UNDEFINED,
  VALID     FUZZY_DATE(5,10)
              UM 'Days (d.)'
  QUALITY   FUZZY DEGREE);
```

Note that the attribute AREA has an associated degree with the meaning of uncertainty (refer to Chapter IV), and attribute QUALITY is a fuzzy degree with its own meaning.

*

VIEW

The CREATE and ALTER VIEW statements permit us to use views with fuzzy queries. The syntax is the same as in SQL but it admits fuzzy queries (see the "Novelties in the Fuzzy SELECT of FSQL" section, earlier in this chapter).

Example 7.31: The following statement creates a view with the data of those properties situated in the center (with a minimum degree of 0.8) and lower in price than Cheap (with a minimum degree of 0.5):

```
CREATE VIEW Center_Cheap AS
   SELECT CODE#, PRICE, DISTRICT, CDEG(DISTRICT)
   FROM HOUSING
```

```
WHERE DISTRICT FEQ   $CENTER 0.8 AND
      PRICE    FLEQ $CHEAP  0.5
READ ONLY;
```

The last attribute indicates the degree to which each property belongs to the central zone. In the view, that degree will be understood in the interval [0.8, 1].

*

LABEL

This object is new and is exclusive to FSQL. The `CREATE LABEL` statement supplies labels in the domain of a fuzzy attribute or a specific fuzzy data type. The label name should be preceded with the symbol `$`, although this is optional in DDL statements. The `CREATE LABEL` statement has different possible syntaxes. In the following definitions we use one attribute (with its optional schema and its table), but a specific fuzzy data type can also be used (refer to the section on specific fuzzy data types earlier in this chapter).

1. To create a label for a fuzzy attribute Type 1 or 2 with a trapezoidal possibility distribution:

```
CREATE LABEL [schema.]table.attribute.[$]Label_Name
        VALUES <fuzzy_constant>;
```

   After the reserved word `VALUES`, the fuzzy constant must follow the syntax exposed in Table 7.2. This statement will generate an error if the attribute is not fuzzy Type 1 or 2 or if the attribute has already defined a label with the same name. In the second case, the `ALTER LABEL` statement should be used with the same syntax. FIRST-2 limits the fuzzy constant type of these labels, but it is easy to change. Example 7.29 shows the creation of one trapezoidal label for a specific fuzzy data type.

2. To create a label for a fuzzy attribute Type 4:

```
CREATE LABEL [schema.]table.attribute.[$]Label_Name;
```

3. To copy one defined label for an attribute, `attribute2`, in another attribute, `attribute1`. Both attributes should be fuzzy attributes Types 1 or 2 indistinctly. Its syntax is as follows:

```
CREATE LABEL [schema.]table.attribute1.[$]Label_Name
        FROM  [schema.]table.attribute2;
```

4. To copy all labels, the character `*` should be used instead of the label name. Both attributes should be fuzzy attributes Type 1 or 2 indistinctly, both of Type 3, or both of Type 4.

```
CREATE  LABEL  [schema.]table.attribute1.*
        FROM   [schema.]table.attribute2;
```

   If the first attribute has some labels, then these labels are not deleted. FSQL gives an error and does not execute the `CREATE LABEL` statement if the second attribute does not have any label. In fuzzy attributes Type 3 or 4, it does not copy the labels from one to another but establishes that both attributes are compatible. The effect is the same as if we apply the `DOMAIN` clause in the `CREATE LABEL` statement (see the section on fuzzy data types earlier in this chapter).

The `DROP LABEL` sentence must include the table, the attribute, and the label using dot notation:

```
DROP  LABEL  <table>.<attribute>.[$]<Label_Name>;
```

Note that if the erased label belongs to a fuzzy attribute Type 3, then the similarity degrees between this label and the other labels will be also erased. In addition, we can drop all labels of an attribute using `*` instead of the label:

```
DROP  LABEL  <table>.<attribute>.*;
```

Example 7.32: To create a label called CHEAP on the attribute PRICE of the table HOUSING:

```
CREATE  LABEL  HOUSING.PRICE.$CHEAP
        VALUES $[50, 60, 70, 80];
```

To drop this label, the user must use the following statement (with or without the $):

```
DROP  LABEL  HOUSING.PRICE.$CHEAP;
```

*

NEARNESS

This object is for creating labels and a similarity relation between them for a fuzzy attribute Type 3 or for a specific fuzzy data type defined like Type 3. The syntax is as follows:

```
CREATE NEARNESS ON Owner
        LABEL  Label_list
        VALUES Similarities_list;
```

- Owner is the owner of this definition. It may be of two classes: [schema.]table.attribute for any fuzzy attribute Type 3 and the name of a specific fuzzy data type defined like Type 3.
- Label_list is the list of all labels for the fuzzy attribute separated by commas.
- Similarities_list is the list of all similarity degrees between each two labels. The number of elements in this list depends on the type of similarity relation: symmetrical or non-symmetrical. The order of the elements depends on the order of the Label_list. If $L_1, L_2, ..., L_n$, are the n labels, and $s(i,j)$ is the similarity degree between L_i and L_j, then the Similarities_list must be written in the following order:

- For non-symmetrical similarity relations:

$$\begin{array}{lllll} s(1,1), & s(1,2), & s(1,3), & \ldots & s(1,n), \\ s(2,1), & s(2,2), & s(2,3), & \ldots & s(2,n), \\ s(3,1), & s(3,2), & s(3,3), & \ldots & s(3,n), \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ s(n,1), & s(n,2), & s(n,3), & \ldots & s(n,n); \end{array}$$

- For symmetrical similarity relations:

$$\begin{array}{lllll} s(1,2), & s(1,3), & s(1,4), & \ldots & s(1,n), \\ & s(2,3), & s(2,4), & \ldots & s(2,n), \\ & & s(3,4), & \ldots & s(3,n), \\ & & & \ldots & \\ & & & & s(n-1,n); \end{array}$$

For n labels, after the word `VALUES`, we must then write n^2 numbers in [0, 1] for non-symmetrical similarity relations. Note that, in general, $s(i, i)$ should be 1, $\forall\, i = 1, 2, \ldots, n$. For symmetrical similarity relations, we must write $n^2/2 - n/2$ numbers in [0,1], where / is the integer division (truncating). For example, with 4 labels, the `VALUES` clause needs 6 numbers, and with 5 labels, it needs 10 numbers.

This statement gives an error in the following cases:

1. The attribute is not a fuzzy attribute Type 3.
2. The attribute already has a similarity relation defined on it. In order to modify the relationship, the `ALTER NEARNESS` statement should be used.
3. The attribute is compatible with another fuzzy attribute Type 3. In this case, we must modify the `NEARNESS` object in the owner attribute. We can delete the object with the `DROP NEARNESS` statement.
4. The `Similarities_list` does not have the precise number of elements, or some of them are not in [0, 1].

Example 7.33: To create labels for the DISTRICT attribute in table HOUSING (Example 7.30) with a symmetrical similarity relation, the syntax is as follows:

```
CREATE NEARNESS ON HOUSING.DISTRICT
          LABEL CENTER, NORTH, SOUTH, EAST, WEST
          VALUES 0.8, 0.8, 0.8, 0.8,
                      0  , 0.5, 0.5,
                           0.5, 0.5,
                                0;
                                                    *
```

The ALTER NEARNESS statement has the same syntax in order to modify all the labels and their similarity relation. It gives an error if the owner does not have a similarity relation. Additionally, we can modify only the similarity degree between two labels, Label_1 and Label_2, with the following syntax:

```
ALTER NEARNESS ON Owner
     CDEG BETWEEN Label_1 AND Label_2 IS <New_Degree>;
```

This statement gives an error in the following cases:

1. The attribute is not a fuzzy attribute Type 3.
2. Labels Label_1 or Label_2 do not exist.
3. The attribute does not have a similarity relation defined on it or is compatible with another fuzzy attribute Type 3. In this case, we must modify the degree in the owner attribute.
4. The new degree is not in [0, 1].

In order to erase labels in a fuzzy attribute Type 3 or in a specific fuzzy data type, we must use the DROP NEARNESS statement:

```
DROP NEARNESS ON Owner;
```

However, if this statement is applied over a compatible fuzzy attribute Type 3, then the statement does not drop the labels and the similarity relation but only removes the compatibility.

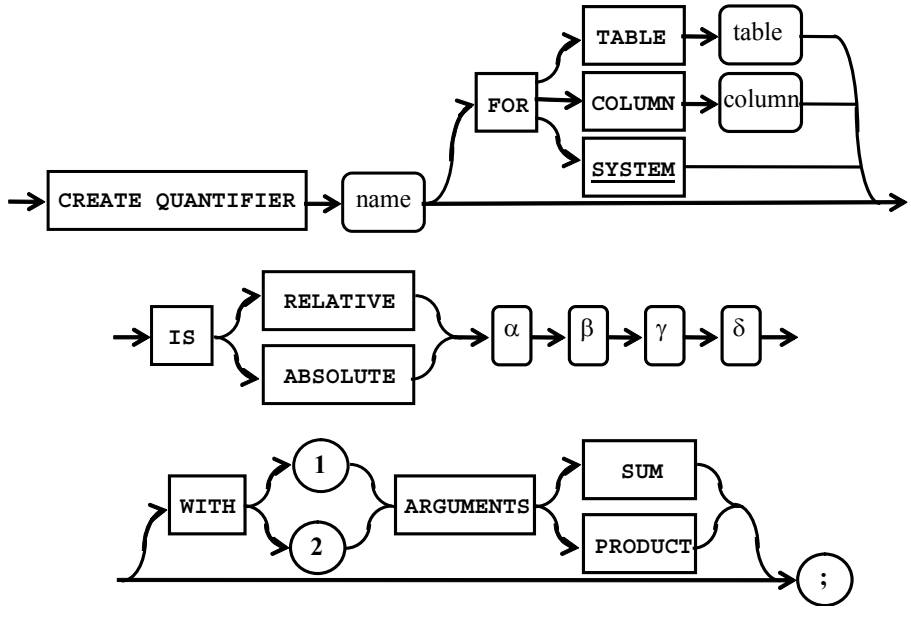
It is worth remembering that we can drop only a label using the `DROP LABEL` statement.

QUALIFIER

This object is new and is exclusive to FSQL. The `CREATE QUALIFIER` statement supplies qualifiers in the [0, 1] interval for a fuzzy attribute or for a specific fuzzy data type. It has the following syntax:

```
CREATE  QUALIFIER  Owner.[$]Qualifier_Name
        IS <value>;
```

Figure 7.7. Structure for `CREATE QUANTIFIER` statement



`Owner` is the owner of this definition, that is, `[schema.]table.attribute` for any fuzzy attribute and the name of a specific fuzzy data type. The qualifier name may be preceded by the optional symbol `$`. After the reserved word `IS`, the value must be in the [0, 1] interval. This statement will generate an error if the attribute is not fuzzy (from Type 1 to 8) or if it has already defined a qualifier with the same name. In the second case, the `ALTER QUALIFIER` statement should be used with the same syntax.

QUANTIFIER

This object is new and is exclusive to FSQL. The `CREATE QUANTIFIER` statement defines fuzzy quantifiers for attributes, tables, and the system. The name may be preceded by the optional symbol `&`. This statement has the syntax expressed in Figure 7.7 with the following clauses:

- `FOR` clause: This optional clause specifies the owner of the quantifier (a table, a column, or the system). By default, the owner is the system. If the owner is a column, then we can write the name of that column, that is, `[schema.]table.attribute`, or we can write the name of a specific fuzzy data type (see the section on specific fuzzy data types earlier in this chapter).
- `RELATIVE` or `ABSOLUTE` words set the type of the quantifier.
- Values α, β, γ, and δ define the trapezoidal function (refer to Figure 7.1).
- `WITH` clause: This optional clause specifies the number of arguments of the quantifier: 1 or 2. By default, the number of arguments is zero. If the number of arguments is one or two, then we must indicate the equation type for constructing the final quantifier according to its arguments:
  - Quantifiers with one argument *x*:
    - `SUM: [α + x, β + x, γ + x, δ + x].`
    - `PRODUCT: [α * x, β * x, γ * x, δ * x].`
  - Quantifiers with two arguments *x* and *y*:
    - `SUM: [α + x, β + x, γ + y, δ + y].`
    - `PRODUCT: [α * x, β * x, γ * y, δ * y].`

MEANING

This object is new and is exclusive to FSQL. The CREATE MEANING statement supplies a new meaning or significance, which can be used in the definition of degrees (see the "Associated Fuzzy Degrees" section, earlier in this chapter). The syntax is as follows:

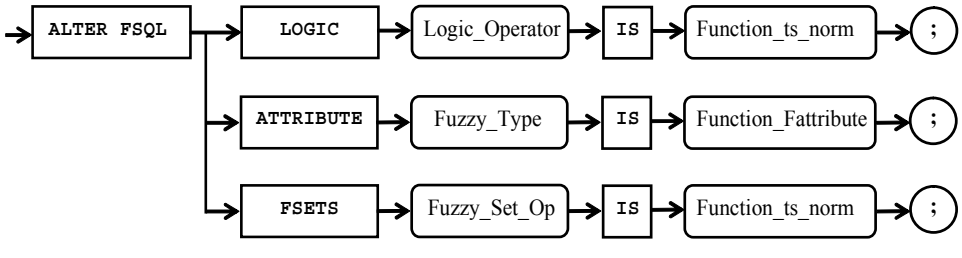
```
CREATE MEANING Meaning_Name IS <number>;
```

The Meaning_Name is the name (for example, importance, possibility, uncertainty, etc.) and the <number> is an associated number that can be used without distinction. This statement will generate an error if the meaning or number already exists in the FMB. In such cases, the ALTER MEANING or DROP MEANING statements should be used. The ALTER MEANING statement allows only the number to be modified, whereas the DROP MEANING statement allows any meaning to be dropped, giving the name or its number.

Modifying FSQL Options: ALTER FSQL and ALTER SESSION

These statements specify or modify certain parameters that affect the behavior of some aspects in FSQL statements. ALTER FSQL affects all personal connections to the database (definitively), whereas ALTER SESSION affects only the current session (or connection). Both statements have the same

Figure 7.8. ALTER FSQL statement



syntax, and the ALTER FSQL statement syntax is represented in Figure 7.8. This statement has three clauses:

1. **LOGIC clause**: This clause specifies the function to use (in the CDEG function) when logic operators are used.
   - **Logic_Operator** may be one of the following words: {**NOT**, **AND**, **OR**, **ALL**}. The word ALL refers to all the three basic logic operators (NOT, AND, and OR).
   - **Function_ts_norm** is the function to use in the previously specified logic operator. Of course, the function must be defined in the DBMS, and the user must be permitted to execute it. Besides, the NOT function must have only one argument, and the AND/OR functions must have two arguments, and they represent a particular t-norm and s-norm, respectively (refer to Tables 1.1 and 1.2). If we use the word **DEFAULT**, then the statement sets the default functions shown in Table 7.4 (GREATEST function for the OR operator, LEAST function for the AND operator, and negation 1-X function for the NOT operator).
2. **ATTRIBUTE clause**: This clause specifies what FSQL does by default when it finds fuzzy attributes in unusual or special positions (see the "Types of Fuzzy Simple Conditions, With and Without Arithmetic Expressions" section, earlier in this chapter). For example, imagine that a fuzzy attribute appears in an ORDER BY clause, or like an argument in a function (different than CDEG).
   - **Fuzzy_Type** may be one of the following words: {**FTYPE2**, **FTYPE3**, **FTYPE4**, **ALL**}. The word ALL refers to all three fuzzy attributes Types 2, 3, and 4.
   - **Function_Fattribute** is the function to use when the previously specified fuzzy attribute type appears in a special position. The function, of course, must be defined in the DBMS, and the user must be permitted to execute it. In addition, we can use the following predefined options:
     - **ERROR**: FSQL gives an error.
     - **FTYPE**: FSQL uses the type of the value in that fuzzy attribute, according to Tables 5.1 and 5.2.

- **TO_CHAR**: FSQL uses the text that represents each value in the fuzzy attribute (for example, an approximate value is represented with 7 ± 2).
- With fuzzy attributes Type 2 we can use the name of one function for the comparison of any fuzzy values in order to decide which of them is greater.

3. **FSET clause**: This clause is useful for specifying the function to use when fuzzy set operators are used (see the section on fuzzy set operators earlier in this chapter).
   - **Logic_Set_Op** may be one of the following words (Table 7.6): {**FUNION**, **FINTERSECT**, **FMINUS**, **ALL**}. The word ALL refers to all three fuzzy set operators.
   - **Function_ts_norm** is the function to use in the previously specified fuzzy set operator. The function, of course, must be defined in the DBMS, and the user must be permitted to execute it. In addition, this function must have two arguments. If we use the word **DEFAULT**, then the statement sets the default functions (see Table 7.6).

Other SQL-Based Fuzzy Languages

SQLf language (Bosc & Pivert, 1995) represents a synthesis of the characteristics and functionalities suggested in other previous proposals of flexible query in classical databases, such as Tahani (1977), Bosc, Galibourg, and Hamon (1988), Wong (1990), and Nakajima, Sogoh, and Arao (1993). These works study only the SELECT statement and ignore other implementation aspects that are very important (refer to Chapter V).

An SQLf query basically follows this format:

```
SELECT [N|T|N,T] <select list>
       FROM   <table list>
       WHERE  <fuzzy condition>
```

The statement applies the fuzzy condition to the Cartesian product of the `FROM` clause. This statement returns only the `N` best rows and/or those with a fulfillment degree greater than `T`. Values `N` and `T` are optional, and the user can write one or both of them. On the other hand, the FSQL presented in this chapter allows thresholds (the `T` value) to be used in any simple or compound condition, and the number of elements `N` may be limited by using the `ORDER BY` clause and the standard pseudocolumn `ROWNUM`, with the following condition: `ROWNUM <= N`.

In the fuzzy condition of SQLf, besides classical conditions, the only fuzzy conditions that can be employed are the comparison between crisp values (columns) with linguistic labels and the use of the "approximately equal" fuzzy comparator between crisp values. For example, if "`well-paid`" is a defined fuzzy predicate, then an SQLf fuzzy condition may be as follows: ... `WHERE Salary = well-paid`.

SQLf allows fuzzy quantifiers in the `HAVING` clause (see the "Fuzzy Quantifiers in Queries" section, earlier in this chapter) without using group functions. However, SQLf does not relate the definition of fuzzy quantifiers to any fuzzy database object. This question is important because, as you see in Example 7.14, the concept of some fuzzy quantifiers — for example, "many" — is different in different contexts. The number of projects in the expression "many projects for one employee" is different than the number of employees in "many employees for one project."

The "Remarks on Fuzzy Queries" section, earlier in this chapter, sets five important aspects in fuzzy queries: fuzzy comparators, thresholds, fuzzy constants, functions for fuzzy treatment, and fuzzy quantifiers. SQLf is more limited than FSQL in all these aspects.

There are some extensions to SQLf (Goncalves & Tineo, 2001a, 2001b), and a version called SQLfi has been implemented (http://www.bd.cesma.usb.ve/~sqlfi).

In addition, dmFSQL was proposed by Carrasco (2003) as an extension of FSQL for data-mining processes. This extension for data mining includes statements for

1. Clustering
2. Classification
3. Obtaining fuzzy global dependencies, or FGD (Carrasco, Vila, Galindo, & Cubero, 2000a, 2000b)

FSQL has also been used as a base for a fuzzy deductive language for fuzzy deductive relational databases (Blanco, Cubero, Cuenca, & Pons, 1999, 2000; Blanco, 2001).

Endnotes

1 Some authors extract other sublanguages from the **DDL**: the **DCL** (Data Control Language) for control purposes (security, etc.), the **SDL** (Storage Definition Language) to define the internal schema of the database and the physical storage, and the **VDL** (View Definition Language), to specify user views, but in most DBMSs the DDL is used to define both conceptual and external schemas (Elmasri & Navathe, 2000).

2 The Fuzzy Metaknowledge Base (FMB) is the catalog of the fuzzy system with information about the fuzzy attributes. The FMB is explained in Chapter V.

3 Oracle is a commercial object-relational database-management system. Web page of Oracle: http://www.oracle.com

4 The MySQL database server is the world's most popular open-source database. Web page of MySQL: http://www.mysql.com

Chapter VIII

Some Applications of Fuzzy Databases With FSQL

The applications of databases are immense. In almost all of them, the advantages of the fuzzy databases can be applied, exploiting their innovative features and possibilities without losing usefulness. Even the model presented here permits an easy use of those advantages in already existing traditional databases. The Type 1 fuzzy attributes are traditional attributes that admit fuzzy queries on them (by using labels, approximate values, fuzzy comparators, etc.).

Imprecise information is a common phenomenon in any context, so it is not unusual to receive information in an incomplete or inexact way. In traditional databases, if information other than precise information exists, the value NULL is stored, preventing the storage of any known information, because the facts are not precise.

Fuzzy databases and the FSQL language have many applications, and the deductive power is very important. For example, in a hospital one could make queries such as the following: "Give me a list of young patients suffering from hepatitis who were admitted approximately more than 5 weeks ago." In a supermarket, it would be useful to know the answer to a request such as the

following: "Give me a listing of the products that have sold very well, but on which we have spent little for publicity."

The list of management applications and useful queries that can be done in this way is endless. We have studied some applications such as the management of a travel agency (Galindo & Aranda, 1999) and the management of a rural accommodation (Galindo, Aranda, Guevara, Caro, & Aguayo, 2002). Another management application is summarized in this chapter: the management of a real estate agency (Galindo, Medina, Cubero, & Pons, 1999; Urrutia & Galindo, 2002; Barranco, Campaña, Cubero, & Medina, 2004; Barranco, Campaña, Medina, & Pons, 2005).

However, the applications of FSQL are not limited to management applications. FSQL can be used for deductive processes in the so-called Fuzzy Deductive Relational Databases (Blanco, Cubero, Cuenca, & Pons, 1999; Blanco, Cubero, Pons, & Vila, 2000; Blanco, 2001) and for data-mining applications (Carrasco, Vila, & Galindo, 2002; Carrasco, 2003).

FSQL is a good tool for data-mining applications because it is flexible and powerful and fulfills a series of requirements for data-mining systems (Chen, Han, & Yu, 1996; Frawley, Piatetsky-Shapiro, & Matheus, 1991):

- High-Level Language: For knowledge discovery and also for outputting the results of the user's request for information (i.e., queries).
- Efficiency: The process should be efficient, that is, the running time should be acceptable.
- Certainty: The discovered knowledge should accurately reflect the content of the database.
- Handling of Different Types of Data.
- Interactive Knowledge Mining: Allows the user to refine a data-mining request online.

In this way, the "Clustering and Fuzzy Classification With FSQL" section in this chapter includes one concrete and easy application of the data-mining sphere: the classification of elements after a clustering process. FSQL in real time makes this operation very easy and, in addition, we are able to treat the different clusters as fuzzy even if they have been obtained by a crisp algorithm (Carrasco, Galindo, Aranda, Medina, & Vila, 1998; Carrasco, Galindo, Vila, & Medina, 1999). Another application for data-mining purposes, briefly explained in the

"FSQL: A Tool for Obtaining Fuzzy Dependencies" section, is to use FSQL as a tool for obtaining fuzzy dependencies.

Another example in the classification area is the classification of images and the fuzzy retrieval of images by using fuzzy attributes of these images (Aranda & Galindo, 1998; Galindo, 1999; Aranda, Galindo, & Urrutia, 2002; Eloy-García, 2003). This application is summarized in the "Fuzzy Classification and Image Retrieval in a Fuzzy Database" section.

The advantages of FRDB are easily evaluated with FSQL because this language is powerful. It has a great quantity of fuzzy comparators (most of them already implemented), the flexibility to establish fulfillment thresholds, the possibility to be installed and used on traditional DBMSs, as well as other characteristics. Therefore, the transfer of research results to the business world has taken an important step.

Management of a Real Estate Agency

In this section we study the case of an estate agency devoted to the sale and rent of apartments, flats, houses, chalets, semidetached houses, building sites, coach houses, and industrial plants. In the management system for a real estate agency some attributes of the landed properties may be fuzzy, that is, we can store imprecise information about it. Besides, this system allows the user to make flexible or fuzzy queries in order to retrieve the most relevant properties of our database, starting with the customer information. The goal is to retrieve the most interesting landed properties according to the initial customer preferences. Of course, we can obtain a membership degree for each landed property in the fuzzy query result.

The database schema for an estate agency must include classic attributes (in addition to fuzzy ones), such as the customers' names, telephone numbers, and addresses. In general, to give a greater versatility to the system, you can define as many Type 2 fuzzy attributes as you consider, instead of Type 1. However, it is important to take into account that the Type 2 fuzzy attributes require in general more storage space and more processing time. So we must choose between *flexibility* (in the representation and fuzzy treatment) and *efficiency* (in storage space and CPU time). However, the storage space and the processing time for fuzzy attributes is not very significant.

Examples of Type 1 fuzzy attributes include the following: number of rooms, number of toilets, price of the community, altitude of the floor, and so forth. It can be observed that in general, the values of the previous attributes are usually well known and clear.

In this particular case, most of the attributes are Type 2. Thus, the database is as flexible as possible. Among these attributes are the sizes of the coach house, the garden or the land (where it proceeds), highlighting the following ones:

- **Price**: Many times, the price is not fixed, and the salesman (owner) establishes an approximate value.
- **Area (m^2)**: Sometimes, it is difficult to quickly access the title deed of the property or to do an exact measurement of its surface. The possibility to store approximate values was very interesting for the consulted estate agents.
- **Age**: Perhaps it is difficult and in general unnecessary to know the exact age of the property, although it is tremendously useful to know its approximate age. So, we can store that a house is new, nearly new, old, or approximately eight years old, for example.

The Type 3 fuzzy attributes include the following, each having its corresponding similarity relationship: lightness (sun), noise, sights, quality of the furniture (if a furnished apartment), and so on.

- **District**: This attribute has been implemented with length 3, indicating that a landed property may be situated among three areas, with different degree. For example, {0.5/Center, 1/North, 0.7/Northwest} indicates that the property is situated in the north district, nearer to the northwest district than to the town center. The similarity relation depends on the distance between the different districts and on its extension. This similarity relation may be seen as a fuzzy relationship. Refer to Example 4.9 for a FuzzyEER representation of this fuzzy relationship.
- **Kind of landed property**: This attribute distinguishes among apartments, flats, chalets, houses, semidetached houses, building sites, industrial plants, and so forth. In general, the similarity relation will be 0 between most of them, but between some of them it will be different. For example, we can establish that a chalet is similar to a semidetached house in degree

0.8. A customer that looks for a chalet is a potential customer of semidetached houses. This point is taken into account in order to show a customer all the relevant properties.

With a database schema like the preceding one, the type of different queries that can be carried out are immense, and the database highlights the comparison among the relation of available properties and the relation of demands for properties. The first relation stores the available properties we can operate with (to sell, to rent, and so forth), and the relation of demands stores the general characteristics of the properties that customers are looking for. Later, the relation of demands is matched with the other relation by using fuzzy necessity comparators (refer to Table 7.1) and thresholds strictly bigger than 0, ranking the results in decreasing order by the compatibility degree of every property. If the query retrieves too many properties, then we can increase the fulfillment threshold, and if the query retrieves too few properties, then we can use possibility comparators rather than those of necessity, because the latter is more restrictive.

Another possibility is to consult the FRDB online at the same time as the customer indicates his or her preferences.

Example 8.1: Suppose that a customer says, "I am looking for a big chalet with about seven rooms and in the northern area." The following FSQL query retrieves the properties that comply with those conditions, ranking by the first attribute, which is the compatibility degree:

```
SELECT CDEG(*), Sales.* FROM Sales
  WHERE Kind      FEQ  $Chalet .5
    AND Surface   FGEQ $Big    .5
    AND Rooms     FGEQ #7      .5
    AND District  FEQ  $North  .5
  ORDER BY 1 DESC;
```

In order to select a greater quantity of properties, we have chosen possibility comparators because several elemental conditions are included. In the previous query, the semidetached houses will also be retrieved if this `Kind` of

property has a similarity degree greater or equal to 0.5 with regard to chalet. If we look exclusively for chalets, then we must establish the threshold to 1.

*

It is easy to note that the number of possible queries and the usefulness of their answers is considerable. Thus, we will first show to each customer the property that has a greater compatibility degree. In the case that none of the retrieved properties satisfy the customer, we can make a more flexible query, putting down the thresholds (until reaching 0), changing the fuzzy constants on the right of the simple conditions, changing fuzzy comparators, eliminating some unimportant conditions, or exchanging some logical comparator AND for OR (using parentheses to establish the precedence).

Naturally, the flexible query system does not ensure the accomplishment of operations, but it does ensure that we find the property most accordant with the customer's needs and preferences. It is necessary to take into account that when somebody looks for any type of property, he rarely has a fixed idea but rather looks for something with some initial basic characteristics. Frequently, what the customer finally acquires is not very similar to what he or she initially looked for.

Besides, this system allows the real estate agency to maintain a large database without having to remember the characteristics of the properties. This situation makes it impossible to handle many properties effectively, and therefore has to be solved by our proposed system.

In some conditions, other characteristics may be used. For example, for a real estate agency, it is sometimes useful to set an urgency degree to each landed property. To do so we use a fuzzy degree for the whole instance (refer to the "Fuzzy Entity as a Fuzzy Degree in the Whole Instance of an Entity" section in Chapter IV), that is, a Type 7 degree (refer to the corresponding section in Chapter V).

Clustering and Fuzzy Classification With FSQL

Usually, real data-mining applications make use of several techniques (statistics, management of databases, artificial intelligence, etc.) to obtain their goals,

that is, discovering implicit and unknown knowledge from a database, which is potentially useful, in a nontrivial way. Fuzzy clustering techniques are useful for classifying without rules, and the FSQL language may be used as a data-mining language.

We propose the use of the FSQL language as a technique of data mining, which can be applied to classify and to obtain the clustering and classification results in real time. This enables us to evaluate the process of extraction of information (data mining) at both a practical and a theoretical level.

After the different clusters are defined, the process of classification can be made with FSQL in real time, and we are able to treat the different clusters as fuzzy, even if they have been obtained with a crisp algorithm. Hence, we can obtain a membership degree for each element to its cluster or clusters. Using FSQL we do not have to use classification algorithms, and the results of the clustering can be used directly in the FSQL queries. We can also treat a crisp cluster (obtained with a nonfuzzy algorithm) as fuzzy. With FSQL we retrieve tuples in real time, in the same way as we do with standard SQL.

The clustering process assigns each individual in the population to a certain cluster (Bezdek, 1981; Michalski and Stepp, 1984; Delgado, Gómez-Skarmeta, & Vila, 1996; Weber, 1996). The clustering is often carried out on a set of examples from the database and not on the entire database. After clustering, the process of computing a cluster, the central values, so-called centroids or centers, obtain the tuple that identifies each cluster or group, that is, the values of the attributes that represent each cluster:

- Numeric attributes (including fuzzy degrees): For each numeric attribute we obtain the average with the tuples that belong to the cluster. Note that the variance should be small, because all tuples belong to the same cluster. However, we can choose an interval instead of a single value.
- Scalar and binary attributes: Every possible value of the scalar attributes is identified with a linguistic label. Thus, a probability distribution is obtained for each scalar attribute in a certain cluster with all tuples in this cluster. A scalar value that is a kind of "average" of the probability distribution using a convex combination of the linguistic labels is obtained (Delgado, Verdegay, & Vila, 1993). Binary attributes are considered as a particular case of scalar attributes that have two possible values.
- Fuzzy attributes Type 1 or 2: In this case, we get a fuzzy value with the approximate average.

- Fuzzy attributes Type 3 or 4: The centered value is a probability distribution obtained with all the values of all tuples in the same cluster.

These centers describe the clusters. The problem is the assignment of the rest of the database and the newly inserted tuples to a particular cluster. Usually a solution is to use a classification algorithm to obtain the rules (e.g., decision trees), which describe each group. Our approach is to use the centurions in an FSQL query. In this manner, the descriptions of the clustering can be made with FSQL in a user-friendly way in *real time*.

Starting from the centers (C_1, ..., C_n) for n attributes (A_1, ..., A_n) of a specific cluster C, we create an FSQL query with the following format:

```
SELECT  table.*,  CDEG(*)
FROM    table
WHERE   A1  FEQ  #C1  THOLD  τ
  AND   A2  FEQ  #C2  THOLD  τ
  AND   …
  AND   An  FEQ  #Cn  THOLD  τ;
```

With this type of query, we retrieve the objects belonging to the cluster C with a minimum membership degree of τ. The value $\#C_i$ means "approximately C_i", represented by a triangular possibility distribution (refer to Table 7.2). Instead of those approximate values, we can also use previously defined linguistic labels or possibility trapezoids expressed with the format $\$[C_i - \kappa, C_i - \lambda, C_i + \lambda, C_i + \kappa]$, choosing some appropriate positive values for κ and λ with $\kappa > \lambda$.

In addition, we can also use another fuzzy comparator, such as NFEQ, to obtain more precise results. In order to avoid the unknown values in an attribute A_i, we can add conditions with the following format: A_i IS NOT UNKNOWN. In fuzzy and crisp attributes, we can also use conditions with the following format in order to avoid null values: A_i IS NOT NULL.

It is easy to see that in this way, we can use different clusters as fuzzy clusters even if they have been obtained with a crisp algorithm. As a result, we can make fuzzy queries to the database, for example: "Give me the objects that belong to cluster C with a minimum ownership degree of τ."

FSQL is a high-level language, and a final user can make a transparent use of it through an FSQL client program (see Appendix B).

Example 8.2: This system has been applied to the problem of the segmentation of bank customers in a real-life situation. The relevant attributes identified by the banking expert have been payroll, credit card use level, and average account balance. `Payroll` is a binary attribute that indicates whether the client receives payroll through the financial company (value Y) or not (value N). The `Credit_Card_Use` attribute measures the level of use of the credit card using the Low, Medium, and High labels, obtained through an analytic study in the company data warehouse system together with the similarity degree among them.

By means of a sample of 1,000 tuples, a clustering process obtained six clusters as the optimum number in the population. Central values of cluster 1 are (`'N'`, `$Medium`, `-307,667`). In order to retrieve, for instance, the customers (in the entire database) who belong to cluster 1 with a minimum degree of 0.7, the FSQL query will be as follows:

```
SELECT Customer#, CDEG(*)
FROM Customers
WHERE Payroll = 'N'
   AND Credit_Card_Use FEQ $Medium THOLD 0.7
   AND Balance FEQ #-307667 THOLD 0.7
ORDER BY 2 DESC;
```

where `Customer#` attribute is the code for every customer in table `Customers`.

*

Note that the user can easily weight the importance of the attributes. Consequently, it is possible to give more or less importance to the fulfillment of the condition for the `Balance` attribute, raising or lowering its corresponding threshold. This may be useful in order to improve the focus of the customer's selection according to the concrete application.

Summarizing, with the FSQL language we are able to extract the elements of a specific cluster or group with a certain minimum membership degree. The

particular membership degree for an element can also be extracted and shown (using the CDEG function of FSQL). Of course, the FSQL queries may be implemented in a program in such a way that the user does not need to know the FSQL syntax. Perhaps the main advantage is that the algorithm of classification (which is often applied later than the clustering algorithm) is unnecessary, because this "classification" is done in *real time* by using FSQL.

FSQL: A Tool for Obtaining Fuzzy Dependencies

Interest in Functional Dependencies (FDs) has been motivated by the fact that FDs can capture some forms of redundancy. The use of FDs has come about as a result of their usefulness in database design. Fuzzy Functional Dependencies (FFDs) arise in the framework of fuzzy relational databases. Various definitions of FFDs have been proposed, but they have not often been closely connected with database design. However, FFDs seem very appropriate to discover properties, which exist in the current manifestation of the data. This is another form of a data-mining process. We can make the same observation about Gradual Functional Dependencies (GFDs), which are a special type of fuzzy dependencies that reflect monotonicity in the data.

Fuzzy and Gradual Functional Dependencies: FFDs and GFDs

There have been several approaches to the problem of defining the concept of FFD, but a single approach has not dominated. We begin by briefly describing the concept of classical FD, then we give a general definition of FFD and GFD based on fuzzy functions, and lastly we introduce more relaxed definitions of FFD and GFD in order to manage exceptions.

Definition 8.1: The relation *R* with attribute sets $X = (x_1, \ldots, x_n)$, and $Y = (y_1, \ldots, y_m)$ in its scheme verifies the **Functional Dependency**, **FD** $\mathbf{X \rightarrow Y}$ if and only if for every instance *r* of *R* it is verified that

$$\forall t_1, t_2 \in r, t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y] \quad (8.1)$$

*

Definition 8.2: The concept of **Fuzzy Functional Dependency** (Cubero & Vila, 1994; Cubero, Medina, Pons, & Vila, 1998) consists in replacing the equality used in the FD definition by fuzzy resemblance relations. The relation *R* verifies an $\alpha - ß$ **FFD** $X \rightarrow_{FT} Y$ if and only if for every instance *r* of *R* it is verified that

$$\forall t_1, t_2 \in r, F(t_1[X], t_2[X]) \geq \alpha \Rightarrow T(t_1[Y], t_2[Y]) \geq ß \quad (8.2)$$

where F and T are fuzzy resemblance relations.

*

If F is a weak resemblance measure and T is a strong one, we get interesting properties for database design (decomposition of relations).

Often, just a few tuples in a database can prevent the FFD from being completed. To avoid this, we can relax the FFD definition in such a way that all the tuples of the relationship are not forced to fulfill the above condition, and therefore we define the confidence of a FFD:

Definition 8.3: The relation *R* verifies an $\alpha - ß$ **FFD** $X \rightarrow_{FT} Y$ in an instance *r* of *R* **with confidence** *c*, where *c* is defined as follows:

$$c = \begin{cases} 0 \\ \dfrac{Card\{(t_1, t_2) \;\; t_1, t_2 \in r / F(t_1[X], t_2[X]) \geq \alpha \;\wedge T(t_1[Y], t_2[Y]) \geq \beta\}}{Card\{(t_1, t_2) \;\; t_1, t_2 \in r / F(t_1[X], t_2[X]) \geq \alpha\}} \end{cases}$$

$$\text{if } Card\{(t_1, t_2) \;\; t_1, t_2 \in r / F(t_1[X], t_2[X]) \geq \alpha\} = 0$$

$$\text{Otherwise} \quad (8.3)$$

where $\wedge$ is the logical operator *and*. The basic idea consists in computing the percentage of tuples that fulfill the antecedent and consequent together with respect to those that fulfill only the consequent.

*

Another way of considering the connections among data in databases is to specify a relationship between objects in a data set and reflect monotonicity in the data by means that we have called GFDs. This is closely related to the basic idea of gradual rules introduced by Dubois and Prade (1992). An intuitive example of a GFD is "the bigger business are, the higher earnings they have," and we assume that the concept of GFD can be considered, in this way, as similar to the FFD one.

Definition 8.4: The relation *R* verifies an **α – ß Gradual Functional Dependency,** denoted by **α – ß GFD $X\int_{FT}Y$**, if and only if for every instance *r* of *R* it is verified that

$$\forall\, t_1, t_2 \in r,\ F'(t_1[X], t_2[X]) \geq \alpha \Rightarrow T'(t_1[Y], t_2[Y]) \geq ß \qquad (8.4)$$

where F' and T' are fuzzy relations different of the equality: *fuzzy greater than, fuzzy less than, fuzzy not equal,* and so forth.

*

We can define the **confidence *c* of an α – ß GFD $X\int_{F',T'}Y$** in the same way that we have made it for FFD (see Definition 8.3).

Applying FSQL to Obtain Global Dependencies

Now, it is necessary to relate the FSQL environment to the previous definitions. First, we introduce a general definition of fuzzy global dependencies (GDs) based on FSQL operators and the CDEG function, and later we show how GDs can be calculated with FSQL.

Definition 8.5: The relation *R* with attribute sets $X = (x_1, \ldots, x_n)$, and $Y = (y_1, \ldots, y_m)$ whose attributes may be fuzzy attributes, verifies an **α – ß Global**

Dependency, denoted by $\alpha - ß$ **GD** $X \blacktriangleright_{F^*T^*} Y$ with $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n) / \alpha_i \in [0, 1] \; \forall \; i = 1, \ldots, n$, and $ß = (ß_1, ß_2, \ldots, ß_m) / ß_j \in [0, 1] \; \forall \; j = 1, \ldots, m$, if and only if every instance *r* of *R* verifies that

$$\forall t_1, t_2 \in r, \wedge_{i=1,2\ldots,n}[\text{CDEG}(t_1[x_i] \; F^*_i \; t_2[x_i]) \geq \alpha_i] \Rightarrow \wedge_{j=1,2\ldots,m}[\text{CDEG}(t_1[y_j] \; T^*_j \; t_2[y_j]) \geq ß_j] \quad (8.5)$$

where

- $F^*_i \; \forall \; i = 1, \ldots, n$ are fuzzy comparators (Table 7.1) in FSQL, denoting fuzzy comparators in the antecedent of the Global Dependency.
- $T^*_j \; \forall \; j = 1, \ldots, m$, are fuzzy comparators (Table 7.1) in FSQL, denoting fuzzy comparators in the consequent of the Global Dependency.
- The CDEG function expresses the compatibility degree in the comparison.

*

Now, we can make a new definition of FFDs and GFDs as a particular case of GDs.

Definition 8.6: Let *r* be an instance of *R* with an $\alpha - ß$ **Global Dependency**, $\alpha - ß$ **GD** $X \blacktriangleright_{F^*T^*} Y$, with $F^*_i, T^*_j \in \{\text{FEQ}, \text{NFEQ}\} \; \forall \; j = 1, \ldots, m, \; \forall \; i = 1, \ldots, n$. Then we say that *R* verifies an $\alpha - ß$ **FFD** $X \rightarrow_{F^*T^*} Y$.

*

Definition 8.7: Let *r* be an instance of *R* with an $\alpha - ß$ **Global Dependency**, $\alpha - ß$ **GD** $X \blacktriangleright_{F^*T^*} Y$, with at least one i or one j with $F^*_i, T^*_j \notin \{\text{FEQ}, \text{NFEQ}\}$. Then we say that *R* verifies an $\alpha - ß$ **GFD** $X \int_{F^*T^*} Y$.

*

Of course, we can define the **confidence** *c* **of an** $\alpha - ß$ **GD** $X \blacktriangleright_{F^*T^*} Y$ in the same sense that we have made it for FFD (see Definition 8.3).

Let *R* be a relation with attribute sets $X = (x_1, \ldots, x_n)$, $Y = (y_1, \ldots, y_m)$ and $PK = (pk_1, \ldots, pk_S)$ included in its scheme, where PK is the primary key of *R*. To

determine whether *R* verifies an $\alpha - ß$ GD $X \blacktriangleright_{F^*T^*} Y$ for an instance *r*, we create an FSQL query with the following general format:

```
SELECT count(*) FROM r A1, r A2
WHERE A1.PK <> A2.PK
   AND A1.x1 F*1 A2.x1 THOLD α1
   AND ...
   AND A1.xn F*n A2.xn THOLD αn
   AND NOT (A1.y1 T*1 A2.y1 THOLD ß1
            AND …
             AND A1.ym T*m A2.ym THOLD ßm);
```

The basic idea consists in computing the tuples that fulfill the antecedent and do not fulfill the consequent in Equation 8.5. If the result of the query is 0, then we can say that *R* verifies GD for the instance *r*. If the result of previous counting is not 0, then we can determine whether *R* verifies an $\alpha - ß$ GD $X \blacktriangleright_{F^*T^*} Y$ with confidence *c* by means of a simple procedure as follows:

- **Step 1**: To obtain the value *a* as the number of tuples that fulfill both the antecedent and consequent:

```
SELECT count(*) FROM r A1, r A2
WHERE A1.PK <> A2.PK
   AND A1.x1 F*1 A2.x1 THOLD α1
   AND …
   AND A1.xn F*n A2.xn THOLD αn
   AND A1.y1 T*1 A2.y1 THOLD ß1
   AND …
   AND A1.ym T*m A2.ym THOLD ßm;
```

- **Step 2**: To obtain the value *b* as the number of tuples that fulfill only the antecedent:

```
SELECT count(*) FROM r A1, r A2
WHERE A1.PK <> A2.PK
   AND A1.x1 F*1 A2.x1 THOLD α1
   AND …
   AND A1.xn F*n A2.xn THOLD αn;
```

- **Step 3**: To obtain the degree of confidence ***c*** as ***c*** = *a*/*b*.
- **Step 4**: To determine whether the computed degree indicates that the GD is good enough, we can compare the value ***c*** with some fuzzy quantifier defined in the FMB (for example, *most*).

If the purpose is to search for FFDs in order to discover intentional properties (constraints that exist in every possible manifestation of the database frame) it seems more appropriate to use a weak resemblance measure in the antecedent (`FEQ`, based on possibility) and a strong one in the consequent (`NFEQ`, based on necessity). In this way, we get interesting properties that can help us with the decomposition of relations (Frawley, Piatetsky-Shapiro, & Matheus, 1991). Searching for FFDs or GFDs to discover extensional properties (those existing in the current manifestation of the data) is a task for data-mining purposes. In this case, the specific problem should indicate the choice of the fuzzy comparators and the parameters (α and ß).

Fuzzy comparators of FSQL have been applied to the definition of fuzzy global dependencies (GD), which are presented as the common framework for fuzzy functional dependencies and gradual functional dependencies. Note that the FSQL language is the natural way to obtain such GDs. We consider that this model satisfies the requirements of data-mining systems.

Fuzzy Classification and Image Retrieval in a Fuzzy Database

A common target in image processing is the image classification or retrieval according to the image's content. In various contexts, fuzzy databases may help by getting fuzzy characteristics of the image objects. Later on, these characteristics will be stored and consulted by using FSQL. We will assume

2-D images in which the object is clearly distinguishable on a homogeneous background.

The object will be represented by using a set of characteristics obtained from the curvature curve of the object's shape, or contour. These characteristics, which are stored in a database, may be treated as fuzzy or crisp.

In order to carry out this classification, we use an object representation based exclusively on the shape (contour) information of such an object, that is, on its external form, which is one of the main characteristics of every object. The final characterization is achieved by defining a set of attributes that can take fuzzy values and extracting the specific characteristics for each shape. By "fuzzy characteristics," we mean those that denote imprecision and that may have inaccurate meanings for humans — for example, the size of an object may be large, medium, or small. Also, should a flexible classification be allowed, the proposed representation reduces the dimensionality of the characterization, so the method may be useful in time-restricted applications.

We briefly explain the process in the following sections. First, we provide a general idea of the method to represent the shape of an object. Then, we obtain characteristics based on that shape, which may be represented as fuzzy attributes of the FRDB; thus, the final classification will be carried out by utilizing such attributes (Aranda & Galindo, 1998; Galindo, 1999; Eloy-García, 2003).

Representing the Shape of an Object

The first important step is to isolate the object to be characterized within a generic 2-D image (if the image contains only gray levels, meaning it is colorless, the process is simpler). The segmentation process, which allows us to subdivide the image into its constituting parts, becomes a complex task in image processing (Jain, 1989). In general, the method selection depends on the nature of the images to be worked on. We will assume that the entry image already has only one object to be represented. For example, images of objects moving along an assembly line (on a flat background) or images that are to be classified and are stored in a database .

In order to represent the object that is present in the image, we assume that all the necessary information is part of the curve defining its contour. Initially, the shape will be characterized by utilizing a function or a *curvature curve* calculated over the object digital contour. The digital contour is a sequence of consecutive adjacent points that do not intersect but define the object border.

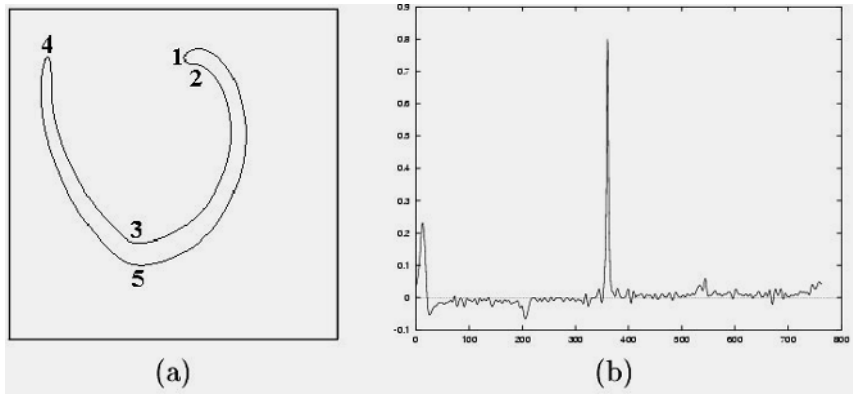
We calculate the contour by connecting the border points obtained with a border extracting algorithm: gradient-based methods (Jain, 1989) or methods based on local energy (Morrone & Owens, 1987).

An object contour is represented by $C = \{(x_i , y_i) \mid i = 1, ..., N\}$, where i indicates the point location within the contour. Thus, (x_i , y_i) is the i-th point along the curve from the initial point (1) to the final one (N).

Let $K = \{k_i \mid i = 1, ..., N\}$ be the graph of the curvature values computed from the digital contour C, and let k_i be the curvature value of (x_i , y_i). The values of the K curvature determine a one-dimensional representation of a flat curve. This one-dimensional signal constitutes a useful descriptor for the contour shape extraction, so the curvature is null in the points of a contour straight section, and on a curve section the smaller the curve radius, the greater the curvature (in absolute value). The curvature value sign indicates the curve direction (concave or convex). More details on the curvature estimation can be seen in Mokhtarian and Mackworth (1986), Aranda and Galindo (1998), and Eloy-García (2003).

Figure 8.1a shows the contour of an object, in this case, the contour of a nematode. Figure 8.1b shows the curvature curve *smoothed out* to the most significant scale. The most significant scale for a curvature is the *smoothing* "level" with which a lower level of quantization noise is obtained. This noise derives from the fact that a digital image contour cannot be continuous; it is a digital contour because the angle between neighboring pixels can only vary in 45-degree increments. That is why the curvature graph is so undulating, although the noise is greatly reduced with the significant scale.

Figure 8.1. a) 2-D curve representing the contour of a nematode worm and b) its curvature



Obtaining the Characteristics of the Shape

The curvature curve will be used to detect the shape characteristics in order to get the final representation. These characteristics represent the shape, which can later be either identified or classified within a given set.

In order to achieve this, some important contour points, called *characteristic points,* are selected; these characteristic points are those points having curvature values, both positive and negative, which stick out of the other ones. The sign tells whether the point belongs to a concave or convex part of the contour, and it depends on how this has been calculated. In this case, the contour is always calculated by starting from the highest point in the shape on the 2-D image and following the contour counterclockwise. The curvature is also calculated by following this order for each of the contour points.

We assume that the curvature values follow a normal distribution having an average of 0 and a σ^2 variance, N(0, σ^2). The sample variance is calculated like this:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(k_i - k_m)^2 \tag{8.6}$$

where k_m is the average of the curvature values, and N is the number of contour points.

The points to be considered important for the shape characterization, that is, the characteristic points, will be *outliers* from the previous distribution. In other words, a point p_i is considered a characteristic point of its contour if the value of its curvature k_i verifies that

$$k_i \notin [-\eta\sigma, +\eta\sigma] \tag{8.7}$$

where η will be used to define the *curvature degree* of a given point. That is to say, the important thing is not the precise curvature value of those points but its comparison with the rest of the points. Then, η measures how big the curvature is, and that information may be coded on different fuzzy linguistic labels. For example, High, Medium, or Low curvature degree. These labels are associated to possibility distributions, as shown in Figure 7.1. Only three possible labels have been considered, but this number may vary according to particular applications.

These labels are important for further treatment because they make classification flexibility possible, so they must be carefully defined.

Points being part of the scope of a characteristic point usually also have a high curvature value. Only the local maximum of each *outliers* points subgroup of the distribution is considered a characteristic point.

The distribution of these points will identify the shape type. The relative distance between every two consecutive characteristic points is also taken into account and is calculated as the number of contour points existing between two such characteristic points with respect to the total number *N* of contour points. For example, square shapes can be differentiated from rectangular shapes.

Summarizing, the object shape is represented as a set of characteristic points. Each one of them keeps information on the curvature sign, the curvature degree (relative to the curvature absolute value, as described earlier in this chapter), and the distance from the previous characteristic point. For the first characteristic point, the distance will be measured with respect to the last one (objects always have closed contours).

Classification and Image Retrieval

After the set of characteristic points has been extracted from the digital contour of an object image, we proceed to the classification process. To do that, as seen above, the following basic attributes of the contour are considered:

1. Number of characteristic points: Naturally, the number of vertexes (corners) of a contour is a basic characteristic of such a contour.
2. Curvature value sign on each characteristic point: This sign makes it possible to see whether the contour follows a concave or convex trajectory. As a matter of fact, the importance of this sign lies in the sign distribution among the found characteristic points.
3. Number of contour points: *N* value.
4. Distance between characteristic points: The distance is calculated as relative distance with respect to total distance (*N*) and allows the distinction between different shapes having an equal number of characteristic points and an equal sign distribution on its curvature, for example, between a square and a rectangle. These distances prevent us from distinguishing between identical objects of different size or at different

distances from the camera. If that were interesting in any given context, the global size *N* of the contour would have to be used.

5. Curvature degree of each characteristic point: The curvature degree is calculated by using the curvature absolute value on each characteristic point and allows the distinction between highly pointed (sharp) characteristic points and less pointed ones. Each point is associated with a linguistic label in order to facilitate classification, because, as stated previously, the exact value is of no interest to us, except with respect to the rest of the values.

Example 8.3: Let us assume we have an `Objects` table containing the characteristics of a multitude of objects extracted from various images of such objects. The query to select objects that are isosceles triangles or similar ones would have the following format:

```
SELECT  Image#,  CDEG(*)
FROM    Objects
WHERE   NumVertices = 3
   AND  Sign1='P' AND Dist1 FEQ $Big 0.8 AND DegreeK1 FEQ $Big 0.7
   AND  Sign2='P' AND Dist2 FEQ $Big 0.8 AND DegreeK2 FEQ $Normal 0.7
   AND  Sign3='P' AND Dist3 FEQ $Small 0.8 AND DegreeK3 FEQ $Normal 0.7;
                                                                    *
```

In certain environments, the model may be expanded in order to take into account other characteristics (crisp or fuzzy), such as color or size (size is not used here).

Even though the second type includes the ones in the first type, this classification may be done in two different ways, according to the type of figure (or shape) that needs to be classified: a) geometrical shapes classification and b) complex shapes classification. Eloy-García (2003) applies this method to compare plane figures, and the results obtained are very promising. Both the program and the results may be obtained on the Internet FSQL[1] Web page.

Depending on the context, the shape may be rotated some degrees. In order to explore this possibility, more comparisons will have to be made between two correlated characteristic points, making sure the same order is kept.

On the other hand, if the number of characteristic points is variable, the database scheme should be different. It should have a table for the general attributes of each object without indicating the values of its characteristic points and another table to store the attributes of the characteristic points of all the shapes.

Endnote

[1] http://www.lcc.uma.es/~ppgg/FSQL

Chapter IX

Brief Summary and Future Trends

Fuzzy logic (Chapter I) allows us to bring the operation of information systems closer to the working methods of humans. People frequently deal with fuzzy concepts (for example, terms such as "almost all," "the majority," "approximately 8," "high," or "low"), which include a certain vagueness or uncertainty and which traditional information systems do not understand and therefore cannot use.

Fuzzy databases (Chapter II) have also been widely studied with the following main objectives: firstly, to allow imprecise or fuzzy data to be stored, and secondly, to allow the possibility of imprecise or fuzzy queries by using existing data (whether imprecise or not). Traditionally, the application of fuzzy logic to databases has paid little attention to the problem of conceptual modeling.

The extension of the ER model for dealing with fuzzy data has been studied in various publications, as we describe in Chapter III, but the FuzzyEER Model in Chapter IV is the most exhaustive version (for more information, see the "Comparison of Some Fuzzy Models" section in Chapter IV). Besides the future lines shown in the concluding section of Chapter IV, it is important to note that fuzzy logic and fuzzy databases are fields of very scientific interest. Thus,

this book is obsolete before seeing the light, because every year thousands of papers and books addressing these themes are published.

The definitions of fuzzy databases and fuzzy database models are not useful if we do not have a DBMS in which we could create these databases. Chapter V describes how to represent fuzzy knowledge in relational databases (FIRST-2). Chapter VI gives the steps of an algorithm for FuzzyEER-to-FIRST-2 mapping. Of course, FIRST-2 has several limitations that can be improved. Nevertheless, we believe that FIRST-2 is sufficiently complete for the immense majority of the applications. Each application has its details, which can cause us to see deficiencies in FIRST-2. In the future, these deficiencies will have to be solved in the general model as well as in each specific application.

Perhaps the more interesting part of this book is the definition of a fuzzy query language, which we have named as Fuzzy SQL or FSQL. Chapter VII describes this language, accentuating its differences with the popular language SQL. We assume that the reader has an understanding of relational database theory and SQL. If the reader does not have this knowledge, he or she will still understand the basic target of FSQL, because Chapter VII includes many examples.

Of course, the FSQL characteristics shown may be improved and extended. However, we think that those definitions are sufficiently good for the majority of the applications. On the other hand, two lines of investigation and development are very important. One of them consists of developing efficient interface programs. These programs should permit the user to perform a multitude of different fuzzy queries, from comfortable and intuitive form. This depends a lot of the concrete application, but the development of generic interfaces could be very interesting and useful. In this line, some studied applications exist, and some of them are briefly shown in Chapter VIII. A DBMS should improve its natural language interfaces so that they incorporate fuzzy characteristics.

Another very important research line consists of studying the use of FSQL as data-mining language. In many cases, the data-mining operations or the data-mining targets are inherently fuzzy. In this field, the development of a complete and efficient interface for data mining is also important. Throughout this book, many research lines are opened, and we think that this book may be useful for postgraduate courses in computer science and for doctoral programs.

At the end of our commentary, we purport that fuzzy databases will be fundamental in the future of databases. In the future, all the DBMSs will permit the use of incomplete information and the possibility to perform fuzzy queries. We hope that this book contributes to that objective, at least a little.

References

American National Standard Institute. (1986). *American national standard for information systems: Database language SQL*. FDT, ANSI X3, 135-1986. New York: American National Standard Institute.

American National Standard Institute. (1992). *Database language SQL*. ANXI X3, 135-1992. New York: American National Standard Institute.

Andreasen, T., & Pivert, O. (1994). On the weakening of fuzzy relational queries. In *Proceedings of the Eighth International Symposium on Methodologies for Intelligent Systems* (pp. 144-153).

Andreasen, T., & Pivert, O. (1995). Improving answers to failing fuzzy relational queries. In *Proceedings of the Sixth International Fuzzy Systems Association Congress* (pp. 414-418).

Aranda, M. C., Galindo, J., & Urrutia, A. (2002). Museos digitales en Internet: Modelo EER difuso y recuperación de imágenes basada en contenido. In *IV Turismo y Tecnologías de la Información y las Comunicaciones (TuriTec'2002*), Málaga, Spain (pp. 411-425).

Aranda, M. C., & Galindo, J. (1998). Clasificación de imágenes de una base de datos utilizando información de su forma. *IV Jornadas Internacionales de Informática* (pp. 565-574). Spain: Las Palmas de Gran Canaria.

Atzeni, P., Ceri, S., Paraboschi, S., & Torlone, R. (1999). *Database systems concepts languages and architecture.* McGraw-Hill.

Baldwin, J. (1983). Knowledge engineering using a fuzzy relational inference language. In *Proceedings of the IFAC Symposium on Fuzzy Information Knowledge Representation and Decision Analysis* (pp. 15-21).

Barbara, D., Garcia-Molina, H., & Porter, D. (1992). The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering, 4,* 487-502.

Barranco, C. D., Campaña, J., Cubero, J. C., & Medina, J. M. (2005). A fuzzy object relational approach to flexible real estate trade. In *Proceedings of the Fourth WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Databases*.

Barranco, C. D., Campaña, J., Medina, J. M., & Pons, O. (2004). ImmoSoftWeb: A Web-based fuzzy application for real estate management. In *Lecture Notes in Computer Science (Vol. 3034): Advances in Web intelligence: Second international atlantic Web intelligence conference* (pp. 196-206). Heidelberg: Springer-Verlag.

Batini, C., Ceri, S., & Navathe, S. (1994). *Diseño conceptual de bases de datos*. Addison Wesley/Díaz de Santos.

Bezdek, J. (1981). *Pattern recognition with fuzzy objective function algorithms*. New York: Plenum Press.

Blanco, I. (2001). *Deducción en bases de datos relacionales difusas*. Unpublished doctoral dissertation, University of Granada, Spain. Retrieved from http://frontdb.ugr.es

Blanco, I., Cubero, J. C., Cuenca, F., & Pons, O. (1999). Implementation of an inference engine for fuzzy databases. In *Proceedings of the EUSFLAT-ESTYLF Joint Conference*, Palma de Malloraca, Spain (pp. 493-496).

Blanco, I., Cubero, J. C., Pons, O., & Vila, M. A. (2000). An implementation for fuzzy deductive relational databases. In G. Bordogna & G. Pasi (Eds.), *Recent issues on fuzzy databases* (pp. 183-207). Physica-Verlag.

Bordogna, G., Leporati, A., Lucarella, D., & Pasi G. (2000). The fuzzy object-oriented database management system. In G. Bordogna & G. Pasi (Eds.), *Recent issues on fuzzy databases* (pp. 209-236). Physica-Verlag.

Bordogna, G., Lucarella, D., & Pasi, G. (1999). A fuzzy object-oriented data model managing vague and uncertain information. *International Journal of Intelligent Systems, 14*(7), 623-651.

Bosc, P. (1998). On diverse answers issued from flexible queries. In T. Andreasen, H. Christiansen, & H. L. Larsen (Eds.), *Lecture Notes in Artificial Intelligence (Vol. 1495): Flexible query answering systems* (pp. 55-67). Springer.

Bosc, P., Dubois, D., Pivert, O., & Prade, H. (1997). Flexible queries in relational databases: The example of the division operador. *Theoretical Computer Science, 171*, 281-302.

Bosc, P., Galibourg, M., & Hamon, G. (1988). Fuzzy querying with SQL: Extensions and implementation aspects. *Fuzzy Sets and Systems, 28*, 333-349.

Bosc, P., & Galibourg, M. (1989). Indexing principles for a fuzzy data base. *Information Systems, 14*(6), 493-499.

Bosc, P., & Pivert, O. (1995). SQLf: A relational database language for fuzzy querying. *IEEE Transactions on Fuzzy Systems, 3*, 1-17.

Buckles, B. P., & Petry, F. E. (1982a). Fuzzy databases and their applications. In M. Gupta & E. Sanchez (Eds.), *Fuzzy information and decision processes* (Vol. 2, pp. 361-371). North-Holland, Amsterdam.

Buckles, B. P., & Petry, F. E. (1982b). A fuzzy representation of data for relational databases. *Fuzzy Sets Systems, 7*, 213-226.

Buckles, B. P., & Petry, F. E. (1984). Extending the fuzzy database with fuzzy numbers. *Information Sciences, 34*, 45-55.

Buckles, B. P., & Petry, F. E. (1985). Uncertainty models in information and database systems. *Information Sciences, 11*, 77-87.

Butnario, D., & Klement, E. P. (1993). *Triangular norm-based measures and games with fuzzy coalitions*. Dordrecht: Kluwer Academic.

Carrasco, R., Galindo, J., Aranda, M. C., Medina, J. M., & Vila, A. (1998). Classification in databases using a fuzzy query language. In C. S. R. Prabhu (Ed.), *Databases for the millennium 2000* (pp. 193-203). McGraw-Hill.

Carrasco, R., Galindo, J., & Vila, A. (2001). Using artificial neural network to define fuzzy comparators in FSQL with the criterion of some decision-maker. In J. Mira & A. Prieto (Eds.), *Lecture Notes in Computer Science (Vol. 2085, Part II): Bio-inspired Applications of Connectionism* (pp. 587-594). Springer.

Carrasco, R., Galindo, J., Vila, A., & Medina, J. M. (1999). Clustering and fuzzy classification in a financial data mining environment. In *Proceedings*

of the Third International ICSC Symposium on Soft Computing (SOCO'99), Italy (pp. 713-720).

Carrasco, R., Vila, A., Galindo, J., & Cubero, J. C. (2000a). FSQL: A tool for obtaining fuzzy dependencies. In *Proceedings of the Eighth International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Spain (pp. 1916-1919).

Carrasco, R., Vila, A., Galindo, J., & Cubero, J. C. (2000b). Fuzzy global dependencies in databases. In *X Congreso Español sobre Tecnologías y Lógica Fuzzy*, Spain (pp. 175-180).

Carrasco, R. A. (2003). *Lenguajes e interfaces de alto nivel para data mining con aplicación práctica a entornos financieros.* Unpublished doctoral dissertation, University of Granada, Spain.

Carrasco, R. A., Vila, M. A., & Galindo, J. (2002). FSQL: A flexible query language for data mining. In M. Piattini, J. Filipe, & J. Braz (Eds.), *Enterprise information systems IV* (pp. 68-74). Kluwer Academic.

Cavallo, R., & Pittarelli, M. (1987). The theory of probabilistic databases. In *Proceedings of the Third International Conference on Very Large Databases* (pp. 102-110).

Chamberlin, D. D., & Boyce, R. F. (1974). SEQUEL: A structured English query language. In *Proceedings of the ACM SIGMOD Workshop on Data Description* (pp. 249-264).

Chamberlin, D. D. et al. (1976). SEQUEL 2: A unified approach to data definition, manipulation and control. *IBM Journal of Research and Development, 20*(6), 560-575.

Chaudhry, N., Moyne, J., & Rundensteiner, E. (1994). A design methodology for databases with uncertain data. In *Proceedings of the Seventh International Working Conference on Scientific and Statistical Database Management*, Charlottesville, VA (pp. 32-41). Retrieved from www.mitexsolutions.com

Chaudhry, N., Moyne, J., & Rundensteiner, E. A. (1999). An extended database design methodology for uncertain data management. *Information Sciences, 121,* 83-112.

Chen, G., Kerre, E., & Vandenbulcke, J. (1994). A computational algorithm for the ffd transitivity closure and complete axiomatization of fuzzy functional dependence (ffd). *International Journal of Intelligent Systems, 9*, 421-440.

Chen, G. Q. (1998). Fuzzy logic in data modeling, semantics constraints, and databases design. In A. K. Elmagarmid (Ed.), *The Kluwer international series on advances in database systems*.

Chen, G. Q., & Kerre, E. E. (1998). Extending ER/EER concepts towards fuzzy conceptual data modeling. In *Proceedings of the IEEE International Conference on Fuzzy Systems, 2* (pp. 1320-1325).

Chen, M., Han, J., & Yu, P. S. (1996). Data mining: An overview from a data base perspective. *IEEE Transactions on Knowledge and Data Engineering, 8*(6), 866-883.

Chen, P. (1976). The entity-relationship model toward a unified view of data. *ACM Transactions on Database Systems, 1*(1), 9-36.

Codd, E. F. (1979). Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems, 4,* 262-296.

Codd, E. F. (1986). Missing information (applicable and inapplicable) in relational databases. *ACM SIGMOD Record, 15*(4).

Codd, E. F. (1987). More commentary on missing information in relational databases. *ACM SIGMOD Record, 16*(1).

Codd, E. F. (1990). *The relational model for database management, Version 2*. Reading, MA: Addison-Wesley.

Connolly, T., Begg, C., & Strachan, A. (1998). *Databases systems, a practical pproach to design, implementation and management* (2nd ed.). Addison-Wesley.

Cubero, J. C., Medina, J. M., Pons, O., & Vila, M. (1998). Fuzzy loss decompositions in databases. *Fuzzy Sets and Systems, 97*(2), 145-167.

Cubero, J. C., & Vila, M. A. (1994). A new definition of fuzzy functional dependency in fuzzy relational databases. *International Journal of Intelligent Systems, 9,* 441-449.

Date, C. J. (1986). Null values in database management. In C. J. Date (Ed.), *Relational Database: Selected Writings*. Reading, MA: Addison-Wesley.

Date, C. J., & Darwen, H. (1997). *A guide to SQL standard* (4th ed.). Addison-Wesley.

Davis, J. P., & Bonnell, R. D. (1989). Modeling semantic constraints with logic in the EARL data model. In *Proceedings of the Fifth International Conference on Data Engineering* (pp. 226-233).

De Caluwe, R. (Ed.) (1997). Fuzzy and uncertain object-oriented databases: Concepts and models. *Advances in Fuzzy Systems: Application and Theory, 13*.

De Luca, A., & Termini, S. (1974). Entropy of L-fuzzy sets. *Information and Control, 24,* 55-73.

De Miguel, A., Piattini, M., & Marcos, E. (1999). *Diseño de bases de datos relacionales*. Spain: Ra-Ma Editorial.

De Miguel, A., & Piattini, M. (1999). *Fundamentos y modelos de bases de datos relacionales* (2nd ed.). Spain: Ra-Ma Editorial.

Delgado, M., Gómez-Skarmeta, A. F., & Vila, A. (1996). On the use of hierarchical clustering in fuzzy modelling. *International Journal of Approximate Reasoning, 14*, 237-257.

Delgado, M., Sánchez, D., & Vila, M. A. (1999). A survey of methods for evaluating quantified sentences. In *Proceedings of the EUSFLAT-ESTYLF Joint Conference*, Spain (pp. 279-282).

Delgado, M., Sánchez, D., & Vila, M. A. (2000). Fuzzy cardinality based evaluation of quantified sentences. *International Journal of Approximate Reasoning, 23*, 23-66.

Delgado, M., Verdegay, J. L., & Vila, M. A. (1993). On aggregation operations of linguistic labels. *International Journal of Intelligent Systems, 8*, 351-370.

Di Battista, G., & Lenzerini, M. (1993). Deductive entity relationship modeling. *IEEE Transactions on Knowledge and Data Engineering*, pp. 439-450.

Dubois, D., & Prade, H. (1980). *Fuzzy sets and systems: Theory and applications*. New York: Academic Press.

Dubois, D., & Prade, H. (1985a). Fuzzy cardinality and the modeling of imprecise quantification. *Fuzzy Sets and Systems, 16*, 190-230.

Dubois, D., & Prade, H. (1985b). *Fuzzy number. An overview, the analysis of fuzzy information*. Boca Raton, FL: CRS Press.

Dubois, D., & Prade, H. (1988). *Possibility theory. An approach to computerized processing of uncertainty*. New York: Plenum Press.

Dubois, D., & Prade, H. (1989). Processing fuzzy temporal knowledge. *IEEE Transactions Systems Man Cybernetics, 19,* 729-744.

Dubois, D., & Prade, H. (1992). Gradual rules in approximate reasoning. *Information Sciences, 61*, 103-122.

Dubois, D., & Prade, H. (1998). *Théorie des possibilites applications á la représentation des connaissances en informatique* (2nd ed.).

Elmasri, R., & Navathe, S. B. (1997). *Sistemas de bases de datos conceptos fundamentales* (2nd ed.). Addison-Wesley.

Elmasri, R., & Navathe, S. B. (2000). *Fundamentals of database systems* (3rd ed.). Addison-Wesley.

Elmasri, R., Weeldreyer, J., & Hevner, A. (1985, May). The category concept: An extension to the entity-relationship model. *International Journal on Data and Knowledge Engineering, 1*(1).

Eloy-García, J. (2003). Recuperación de imágenes usando atributos difusos. Proyecto Fin de Carrera de Ingeniería Superior en Informática (Universidad de Málaga), directed by M.C. Aranda and J. Galindo. Retrieved from http://www.lcc.uma.es/~ppgg/PFC

Escobar, C. (2003). Software para control difuso de todo tipo de sistemas (SCD): Aplicación al Control de Invernaderos Industriales. Proyecto Fin de Carrera de Ingeniería Técnica Industrial en Electrónica (Universidad de Málaga), directed by J. Galindo. Retrieved from http://www.lcc.uma.es/~ppgg/PFC

Fowler, M., & Scott, K. (1999). *UML gota a gota*. Addison Wesley Longman, Pearson.

Frawley, W. J., Piatetsky-Shapiro, G., & Matheus, C. J. (1991). Knowledge discovery in databases: An overview. In G. Piatetsky-Shapiro & W. J. Frawley (Eds.), *Knowledge discovery in databases* (pp. 1-31). The AAAI Press.

Fuhr, N. (1990). A probabilistic framework for vague queries and imprecise information in databases. In *Proceedings of the Sixth International Conference on Very Large Data Bases* (pp. 77-85).

Fujishiro, I. et al. (1991). The design of a graph-oriented schema for the management of individualized fuzzy data. *Japanese Journal of Fuzzy Theory and Systems, 3*(1), 1-14.

Fukami, S., Umano, M., Muzimoto, M., & Tanaka, H. (1979). Fuzzy database retrieval and manipulation language (Tech. rep. no. AL-78-85). *IEICE Technical Reports, 78*(233), 65-72.

Galindo, J. (1999). *Tratamiento de la imprecisión en bases de datos relacionales: Extensión del modelo y adaptación de los SGBD actuales.* Unpublished doctoral dissertation, University of Granada, Spain.

Galindo, J. (2001). Curso sobre "Conjuntos y Sistemas Difusos (Lógica Difusa y Aplicaciones)." Informe Técnico de Docencia. LCC-ITI-2001-11, Universidad de Málaga, Departamento de Lenguajes y Ciencias de la Computación. Retrieved from http://www.lcc.uma.es

Galindo, J. (2005). New characteristics in FSQL, a fuzzy SQL for fuzzy databases. In *Proceedings of the Fourth WSEAS International Conference on Artificial Intelligence, Knowledge Engineering, and Databases*. Retrieved from www.lcc.uma.es/~ppgg/FSS

Galindo, J., & Aranda, M. C. (1999). Gestión de una agencia de viajes usando bases de datos difusas y FSQL. In *Proceedings of Turismo y Tecnologías de la Información y Las Comunicaciones: Nuevas Tecnologías y Calidad (TuriTec '99)*, Malaga, Spain (pp. 343-6).

Galindo, J., Aranda, M. C., Guevara, A., Caro, J. L., & Aguayo, A. (2002). Applying fuzzy databases and FSQL to the management of rural accommodation. *Tourist Management Journal, 23*(6), 623-629.

Galindo, J., Medina, J. M., Aranda, M. C. (1999). Querying fuzzy relational databases through fuzzy domain calculus. *International Journal of Intelligent Systems, 14*(4), 375-411.

Galindo, J., Medina, J. M., Cubero, J. C., & García, M. T. (2001). Relaxing the universal quantifier of the division in fuzzy relational databases. *International Journal of Intelligent Systems, 16*(6), 713-742.

Galindo, J., Medina, J. M., Cubero, J. C., & Pons, O. (1999). Management of an estate agency allowing fuzzy data and flexible queries. In *Proceedings of the EUSFLAT-ESTYLF Joint Conference*, Spain (pp. 485-488).

Galindo, J., Medina, J. M., & Rodríguez, J. M. (2000). Comparadores para bases de datos difusas: Definiciones, clases y relaciones. In *X Congreso Español sobre Tecnologías y Lógica Fuzzy*, Spain, (pp. 187-192). Retrieved from *www.lcc.uma.es*

Galindo, J., Medina, J. M., Vila, M. A., & Pons, O. (1998). Fuzzy comparators for flexible queries to databases. In *Proceedings of the Sixth Iberoamerican Conference on Artificial Intelligence*, (pp. 29-41).

Galindo, J., Medina, M., Pons, O., & Cubero, J. C. (1998). A server for fuzzy SQL queries. In T. Andreasen, H. Christiansen, & H. L. Larsen (Eds.), *Lecture Notes in Artificial Intelligence (Vol. 1495): Flexible query answering systems* (pp. 164-174). Springer.

Galindo, J., Oliva, R. F., & Carrasco, R. A. (2004). Acceso Web a bases de datos difusas: Un cliente visual de fuzzy SQL. In *XII Congreso Español sobre Tecnologías y Lógica Fuzzy (ESTYLF 2004)*, Jaen, Spain (pp. 83-88).

Galindo, J., Urrutia, A., & Carrasco, R. (2002). Cuatro tipos de restricciones difusas en especializaciones. In *Workshop de Bases de Datos*, Chile (pp. 79-88).

Galindo, J., Urrutia, A., Carrasco, R., & Piattini, M. (2001). Fuzzy constraints using the enhanced entity-relationship model. In *Proceedings of the 21st International Conference of the Chilean Computer Science Society*, Chile (pp. 86-94).Retrieved from http://computer.org/proceedings/sccc/1396/13960086abs.htm

Galindo, J., Urrutia, A., Carrasco, R., & Piattini, M. (2004). Relaxing constraints in enhanced entity-relationship models using fuzzy quantifiers. *IEEE Transactions on Fuzzy Systems, 12*(6), 780-796.

Galindo, J., Urrutia, A., & Piattini, M. (2004a). Fuzzy aggregations and fuzzy specializations in fuzzy EER model. In Keng Siau (Ed.), *Advanced Topics in Database Research* (Vol. III, pp. 105-126). Hershey, PA: Idea Group Publishing.

Galindo, J., Urrutia, A., & Piattini, M. (2004b). Storing fuzzy knowledge and fuzzy metaknowledge in relational systems. In *Proceedings of the Fifth WSEAS International Conference on Fuzzy Sets and Fuzzy Systems*, Italy.

Geneste, L., & Ruet, M. (2001). Experience based configuration. In *Proceedings of the 17th International Conference on Artificial Intelligence*, USA (pp. 51-60).

Geneste, L., & Ruet, M. (2002). Fuzzy case based configuration. In *Proceedings of the 15th European Conference on Artificial Intelligence*, France (pp. 130-142).

George, R., Srikanth, R., Petry, F. E., & Buckles, B. P. (1996). Uncertainty management issues in the object-oriented data model. *IEEE Transactions on Fuzzy Systems, 4*(2), 179-192.

Giardina, C. (1979). *Fuzzy databases and fuzzy relational associative processors*. Hoboken, NJ: Stevens Institute of Technology.

Goncalves, M., & Tineo, L. (2001a). SQLf flexible querying extension by means of the norm SQL2. In *Proceedings of the IEEE International Fuzzy Systems Conference* (pp. 473-476).

Goncalves, M., & Tineo, L. (2001b). SQLf3: An extension of SQLf with SQL3 features. In *Proceedings of the IEEE International Fuzzy Systems Conference*, (pp. 477-480).

Goswami, A., & Kumar, P. (2001). Conceptual modeling of fuzzy object-oriented database systems. In *ICEIS*, 287-291.

Grant, J. (1980). Incomplete information in a relational database. *Fundamenta Informaticae, 3*, 363-378.

Gregersen, H., & Jensen, C. S. (1999). Temporal entity-relationship models: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 464-497.

Hammer, M., & McLeod, D. (1981). Database description with SDM: A semantic data model. *ACM Transactions on Database Systems, 6*(3), 351-386.

Hernández, R., Fernández, C., & Baptista, P. (1998). *Metodología de la investigación.* McGraw Hill.

Jain, A. K. (1989). *Fundamentals of digital image processing.* Englewood Cliffs, NJ: Prentice-Hall.

Jensen, C. et al. (1994). A glossary of temporal database concepts. *ACM SIGMOD Record, 23*(1).

Kerre, E. E, & Chen, G. (1995). An overview of fuzzy data models. In P. Bosc & J. Kacprzyk (Eds.), *Studies in fuzziness: Fuzziness in database management systems* (pp. 23-41). Physica-Verlag.

Kerre, E. E., & Chen, G. (2000). Fuzzy data modeling at a conceptual level: Extending ER/EER concepts. In O. Pons, M. A. Vila, & J. Kacprzyk (Eds.), *Knowledge management in fuzzy databases* (pp. 3-11). Physica-Verlag.

Kosko, B. (1992). *Neural networks and fuzzy systems: A dynamical systems approach to machine intelligence*. Englewood Cliffs, NJ: Prentice-Hall.

Kruse, R., Gebhardt, J., & Klawonn, F. (1994). *Foundations of fuzzy systems*. Wiley.

Lipski, W. (1979). On semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems, 4*, 262-296).

Liu, Y., & Kerre, E. E. (1998a). An overview of fuzzy quantifiers. Part I: Interpretations. *Fuzzy Sets and Systems, 95*(1), 1-21.

Liu, Y., & Kerre, E. E. (1998b). An overview of fuzzy quantifiers. Part II: Reasoning and applications. *Fuzzy Sets and Systems, 95*(2), 135-146.

Luque, I., Gómez-Nieto, M. A., López, E., & Cerruela, G. (2001). *Bases de datos desde chen hasta codd con ORACE*. Editorial Rama.

Ma, Z. M., Zhang, W. J., & Ma, W. Y. (2004). Extending object-oriented databases for fuzzy information modeling. *Information Systems, 29*, 421-435.

Ma, Z. M., Zhang, W. J., Ma, W. Y., & Chen, Q. (2001). Conceptual design of fuzzy object-oriented databases using extended entity-relationship model. *International Journal of Intelligent Systems, 16*(6), 697-711.

Malik, J., & Binford, T. O. (1983). Reasoning in time and space. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 343-345).

Marín, N., Pons, O., & Vila, M. A. (2000). Fuzzy types: A new concept of type for managing vague structures. *International Journal of Intelligent Systems, 15,* 1061-1085.

Marín, N., Vila, A., Blanco, I., & Pons, O. (2000). Fuzzy types as a new layer on an object oriented database system. In *Proceedings of the Eighth International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'2000)*, Madrid, Spain (pp. 1099-1102).

McNeill, F. M., & Thro, E. (1994). *Fuzzy logic: A practical approach.* AP Professional.

Medina, J. M. (1994). *Bases de datos relacionales difusas: Modelo teórico y aspectos de su implementación.* Unpublished doctoral dissertation, Universidad de Granada, Spain. Retrieved from decsai.ugr.es

Medina, J. M., Pons, O., & Vila, M. A. (1994). GEFRED: A generalized model of fuzzy relational databases. *Information Sciences, 76*(1/2), 87-109.

Medina, J. M, Pons, O., & Vila, A. (1995). FIRST: A fuzzy interface for relational systems. *Proceedings of the Sixth International Fuzzy Systems Association World Congress*, Brazil (pp. 409-412).

Menger, K. (1942). Statistical metric spaces. In *Proceedings of the National Academy of Sciences, 37*, USA (pp. 535-537).

Michalski, R. S., & Stepp, R. E. (1984). Learning from observation: Conceptual clustering. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.),

Machine learning: An artificial intelligence approach (pp. 331-363). Berlin: Springer-Verlag.

Mohammad, J., Vadiee, N., & Ross, T. J. (Eds.) (1993). *Fuzzy logic and control: Software and hardware applications*. Englewood Cliffs, NJ: Prentice-Hall.

Mokhtarian, F., & Mackworth, A. (1986). Scale-based description and recognition of planar curves and two-dimensional shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 8*, 34-43.

Morrone, M. C., & Owens, R. A. (1987). Feature detection from local energy. *Pattern Recognition Letters, 6*, 303-313.

Motro, A. (1990). Accommodating imprecision in database systems: Issues and solutions. *SIGMOD Record, 19*(4), 69-74.

Motro, A. (1995). Management of uncertainty in database system. In W. Kim (Ed.), *Modern database system: The object model, interoperability and beyond.* Addison-Wesley.

Mouaddib, N. (1994). Fuzzy identification database: The nuanced relation division. *International Journal of Intelligent Systems, 9*, 461-473.

Nakajima, H., Sogoh, T., & Arao, M. (1993). Fuzzy database language and library: Fuzzy extension to SQL. In *Proceedings of the Second International Conference on Fuzzy Systems (FUZZ-IEEE'93)* (pp. 477-482).

Oliva, R. F. (2003). Visual FSQL: Gestión visual de bases de datos difusas en ORACLE a través de Internet usando FSQL. Proyecto Fin de Carrera, directed by J. Galindo, of Ingeniería Superior en Informática, in the University of Málaga. Retrieved from http://www.lcc.uma.es/~ppgg/PFC

Ozawa, J., & Yamada, K. (1994). Cooperative answering with macro expression of a database. In *Proceedings of the International Conference on Information Processing and Management in Knowledge-Based Systems* (pp. 17-22).

Patrick, J. J. (2002). *SQL fundamentals* (2nd ed.). Prentice-Hall.

Pawlak, Z. (1982). Rough sets. *International Journal of Computer and Information Sciences, 11*, 341-356.

Pawlak, Z. (1991). *Rough sets: Theoretical aspects of reasoning about data.* Norwell, MA: Kluwer Academic.

Pedrycz, W., & Gomide, F. (1998). *An introduction to fuzzy sets: Analysis and design*. Cambridge, MA: MIT Press.

Petry, F. E. (with Bose, P.). (1996). Fuzzy databases: Principles and applications. In H. J. Zimmermann (Ed.), *International series in intelligent technologies*. Kluwer Academic.

Piegat, A. (2001). *Fuzzy modeling and control*. Physica-Verlag.

Pons, O. (1996). *Representación lógica de bases de datos difusas: Fundamento teórico e implementación.* Unpublished doctoral dissertation, Universidad de Granada.

Prade, H. (1984). Lipski's approach to incomplete information databases restated and generalized in the setting of Zadeh's possibitity theory. *Information Systems, 9*, 27-42.

Prade, H., & Testemale, C. (1984). Generalizing database relational algebra for the treatment of incomplete/uncertain information and vague queries. *Information Sciences, 34*, 115-143.

Prade, H., & Testemale, C. (1987a). Fuzzy relational databases: Representational issues and reduction using similarity measures. *J. Am. Soc. Information Sciences, 38*(2), 118-126.

Prade, H., & Testemale, C. (1987b). Representation of soft constraints and fuzzy attribute values by means of possibility distributions in databases. In J. Bezdek (Ed.), *Analysis of fuzzy information (Vol. II): Artificial intelligence and decision systems* (pp. 213-229). CRC Press.

Quian Da-qun. (1992). Representation and use of imprecise temporal knowledge in dynamic systems. *Fuzzy Sets and Systems, 50*, 59-77.

Raju, K., & Majumdar, A. (1988). Fuzzy functional dependencies and lossless joint decomposition of fuzzy relational database system. *ACM Transactions on Database Systems, 13*(2), 129-166.

Reingruber, M., & Gregory, W. (1994). *The data modeling handbook: A best-practice approach to building quality data models*. Wiley.

Rumbaugh, J., Jacobson, J., & Booch, G. (1999). *The unified modeling language reference manual.* Addison-Wesley.

Rundensteiner, E., & Bic, L. (1989). Semantic database models and their potential for capturing imprecision. In *Proceedings of the Conference on Management of Data*, India, (pp. 43-57).

Ruspini, E. (1986). Imprecision and uncertainty in the entity-relationship model. In H. Prade & C. V. Negoita, *Fuzzy logic in knowledge engineering* (pp. 18-22). Verlag TUV Rheinland.

Sánchez, D. (1999). *Adquisición de relaciones entre atributos en bases de datos relacionales*. Unpublished doctoral dissertation, University of Granada, Spain.

Saaty, T. L. (1980). *The analytic hierarchy processes*. New York: McGraw-Hill.

Schweizer, B., & Sklar, A. (1983). *Probabilistic metric spaces*. North-Holland, Amsterdam.

Shoshani, A., & Wong, H. (1985). Statistical and scientific database issues. *IEEE Transactions on Software Engineering, 11*, 235-256.

Silverschatz, A., Korth, H. F., & Sudarshan, S. (2002). *Database system concepts*. New York: McGraw-Hill.

Snodgrass, R. (1995). *The TSQL2 temporal query language*. Kluwer.

Tahani, V. (1977). A conceptual framework for fuzzy query processing a set toward very intelligent database system. *Information Processing and Management, 13*, 289-303.

Tré, G., de Caluwe, R., & Van der Cruyssen, B. (2000). A generalised object-oriented database model. In G. Bordogna & G. Pasi (Eds.), *Recent issues on fuzzy databases* (pp. 155-182). Physica-Verlag.

Trillas, E. (1979). Sobre funciones de negación en la teoría de conjuntos difusos. *Stochastica, 3*(1), 47-59.

Umano, M. (1982). Freedom-O: A fuzzy database system. In M. Gupta & E. Sanchez (Eds.), *Fuzzy information and decision processes* (pp. 339-347). North-Holland, Amsterdam.

Umano, M. (1983). Retrieval from fuzzy database by fuzzy relational algebra. In M. Gupta & E. Sanchez (Eds.), *Fuzzy information, knowledge representation and decision analysis* (pp. 1-6). New York: Pergamon Press.

Umano, M., & Fukami, S. (1994). Fuzzy relational algebra for possibility-distribution-fuzzy-relation model of fuzzy data. *Journal of Intelligent Information Systems, 3*, 7-28.

Urrutia, A. (2002). Implementación de bases de datos difusas: Un caso de control de la calidad del papel. Revista electrónica Gerencia Tecnología Informática AEDO, Volumen 1, número 1. Colombia. Retrieved from http://cidlisuis.org

Urrutia, A. (2003). *Definición de un modelo conceptual para bases de datos difusas*. Unpublished doctoral dissertation, University of Castilla-

La Mancha, Spain. Retrieved from http://www.ganimides.ucm.cl/aurrutia

Urrutia, A., & Galindo, J. (2001, December). Notación para datos con imprecisión en un modelo conceptual difuso. *Revista Académica de la Universidad Católica del Maule* (number 27, pp. 39-48). Chile.

Urrutia, A., & Galindo, J. (2002). Algunos aspectos del modelo conceptual EER difuso: Aplicación al caso de una agencia inmobiliaria. In *XI Congreso Español sobre Tecnologías y Lógica Fuzzy*, Spain (pp. 359-364).

Urrutia, A., Galindo, J., & Jiménez, L. (2001). Representación de información imprecisa en un modelo conceptual EER difuso. In *VIII Congreso Internacional de Investigación en Ciencias Computacionales*, México (pp. 15-27).

Urrutia, A., Galindo, J., & Jiménez, L. (2002). Extensión del modelo conceptual EER para representar tipos de datos difusos. *I+D Computación, 1*(2). Retrieved from http://www.sd-cenidet.com.mx/Revista

Urrutia, A., Galindo, J., & Piattini, M. (2002). Modeling data using fuzzy attributes. In *Proceedings of the 22nd International Conference of the Chilean Computer Science Society*, Chile (pp. 117-123).

Urrutia, A., & Piattini, M. (2001). Transformation of imprecise data to linguistic labels for ERModels. In *Proceedings of the Seventh International Conference on Information Systems Analysis and Synthesis*, USA (pp. 351-355).

Van Gyseghem, N., & de Caluwe, R. (1997). The UFO model: Dealing with imperfect information. In *Fuzzy and uncertain object-oriented databases, concepts and models* (pp. 123-185). World Scientific.

Van Gyseghem, N., de Caluwe, R., & Vandenberghe, R. (1993, March). UFO: Uncertainty and fuzziness in an object-oriented model. In *Proceedings of the Second IEEE International Conference Fuzzy Systems*, San Francisco (pp. 773-778).

Vandenberghe, R. M. (1991). An extended entity-relationship model for fuzzy databases based on fuzzy truth values. In *Proceedings of the Fourth International Fuzzy Systems Association World Congress*, Brussels (pp. 280-283).

Varas, M., Contreras, R., & Campos, D. (1998). Constraints in generalization structures in conceptual database schemes. In *Conferencia Internacional de la Sociedad Chilena de Ciencia de la Computación*, Chile.

Vert, G., Morris, A., Stock, M., & Jankowski, P. (2000). Extending entity-relationship modelling notation to manage fuzzy datasets. In *Proceedings of the Eighth International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Spain (pp. 1131-1138).

Vila, M., Cubero, J. C., Pons, O., & Medina, J. M. (n.d.). *A fuzzy object-oriented data model represented by means of a semantic data model*. Granada, Spain: Universidad de Granada, Department of Computer Science.

Virant, J., & Zimic, N. (1996). Attention to time in fuzzy logic. *Fuzzy Sets and Systems, 82*(1), 39-49.

Weber, R. (1996). Customer segmentation for banks and insurance groups with fuzzy clustering techniques. *Fuzzy Logic*.

Wong, E. A. (1982). Statistical approach to incomplete information in database systems. *ACM Transactions on Database Systems, 7*, 479-488.

Wong, M., & Leung, K. (1990). A fuzzy database-query language. *Information Systems, 15*, 583-590.

Yager, R. R. (1980). On a general class of fuzzy connectives. *Fuzzy Sets and Systems*, 235-242.

Yager, R. R. (1983). Quantified propositions of a linguistic logic. *International Journal of Man-Machine Studies, 19,* 195-227.

Yager, R. R. et al. (1987). *Fuzzy sets and applications: Selected papers by L. A. Zadeh.* Wiley Interscience.

Yazici, A., & Akkaya, K. (2000). Conceptual modeling of geographic information system applications. In G. Bordogna & G. Pasi (Eds.), *Recent issues on fuzzy databases* (pp. 129-151). Physica-Verlag.

Yazici, A., Buckles, B. P., & Petry, F. E. (1999). Handling complex and uncertain information in the ExIFO and NF2 data models. *IEEE Transactions on Fuzzy Systems, 7*(6), 659-675.

Yazici, A., & Cinar, A. (2000). Conceptual modeling for the design of fuzzy object oriented database. In O. Pons, M. A. Vila, & J. Kacprzyk (Eds.), *Knowledge management in fuzzy databases* (pp. 12-35). Physica-Verlag.

Yazici, A., & George, R. (1999). *Fuzzy database modeling.* New York: Physica-Verlag.

Yazici, A., & Koyuncu, M. (1997). Fuzzy object-oriented database modeling coupled with fuzzy. *Fuzzy Sets and Systems, 89*, 1-26.

Yazici, A., & Merdan, O. (1996). Extending IFO data model for uncertain information. In *Proceedings of the Fourth International Conference on Information Processing and Management of Uncertainty (IPMU'96, Vol. III)*, Spain (pp. 1283-1282).

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control, 8*, 338-353.

Zadeh, L. A. (1971). Similarity relations and fuzzy orderings. *Information Sciences, 3*, 177-200.

Zadeh, L. A. (1975a). The concept of linguistic variable and its application to approximate reasoning: Part 1. *Information Sciences, 8*, 199-249.

Zadeh, L. A. (1975b). The concept of linguistic variable and its application to approximate reasoning: Part 2. *Information Sciences, 8*, 301-367.

Zadeh, L. A. (1975c). The concept of linguistic variable and its application to approximate reasoning: Part 3. *Information Sciences, 9*, 43-60.

Zadeh, L. A. (1978). Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems, 1,* 3-28.

Zadeh, L. A. (1983). A computational approach to fuzzy quantifiers in natural languages. *Computer Mathematics with Applications, 9,* 149-183.

Zadeh, L. A. (1992). Knowledge representation in fuzzy logic. In *An introduction to fuzzy logic applications in intelligent systems*. Kluwer Academic.

Zemankova-Leech, M., & Kandel, A. (1984). *Fuzzy relational databases: A key to expert systems*. Köln, Germany: Verlag TUV Rheinland.

Zemankova-Leech, M., & Kandel, A. (1985). Implementing imprecision in information systems. *Information Sciences, 37*, 107-141.

Zimmermann, H.-J. (1991). *Fuzzy set theory and its applications* (2nd ed.). Kluwer Academic.

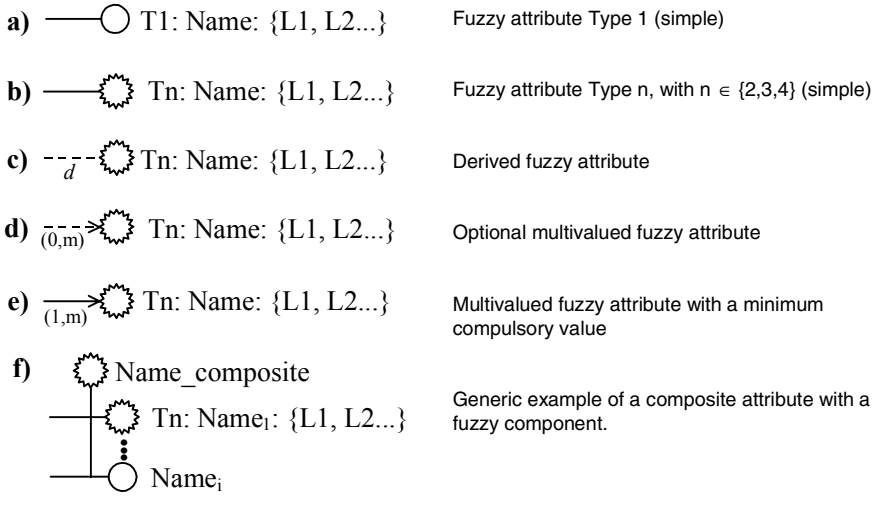
Zvieli, A., & Chen, P. (1986). ER modeling and fuzzy databases. In *Proceedings of the Second International Conference on Data Engineering* (pp. 320-327).

Appendices

Appendix A: Summary of FuzzyEER Model

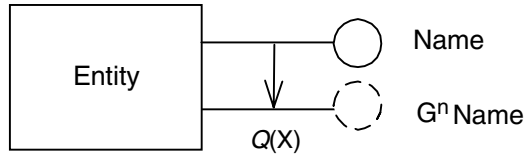
The FuzzyEER model can be summarized in a total of 18 graphic representations. Chapter IV includes 24 definitions, 4 types of fuzzy attributes, 4 types of fuzzy degrees, 22 formal examples, and the comparison of FuzzyEER with some other fuzzy models (Table 4.3).

1. **Fuzzy values in fuzzy attributes** (Definitions 4.1 and 4.2):

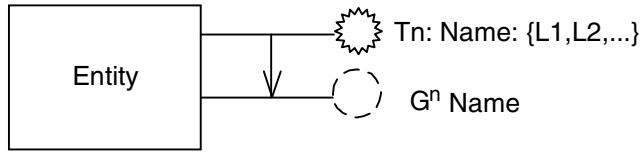


2. **Fuzzy degree associated to each value of an attribute** (Definitions 4.3 and 4.4): Fuzzy degree Name with meaning n, in G^n

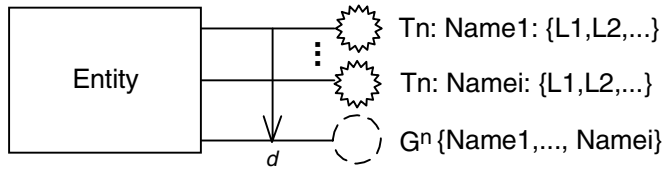
   a. Derived fuzzy degree with function *Q*:



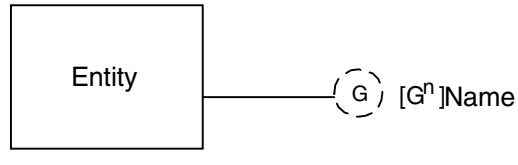
   b. Nonderived fuzzy degree:



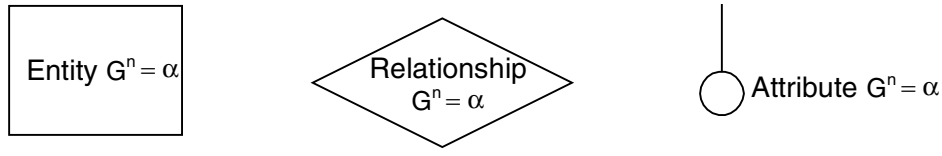
3. **Fuzzy degree associated to some attributes** (Definition 4.5): Fuzzy degree Name with meaning n, in G^n, associated to i attributes with $i \geq 2$:



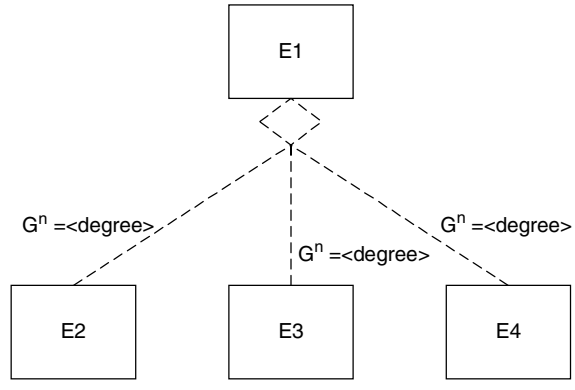
4. **Fuzzy degree with its own meaning** (Definition 4.6): Fuzzy degree Name (with optional meaning n, in G^n):



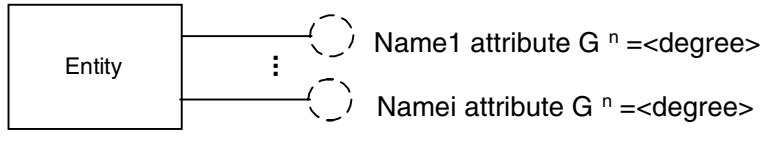
5. **Fuzzy degree to the model of an entity, relationship, or attribute** (Definition 4.7): Fuzzy degree to the model, with meaning n, in G^n, and degree α:



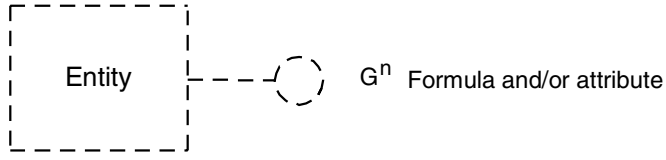
6. **Fuzzy aggregations** (Definition 4.8):
   a. Fuzzy aggregation of entities:



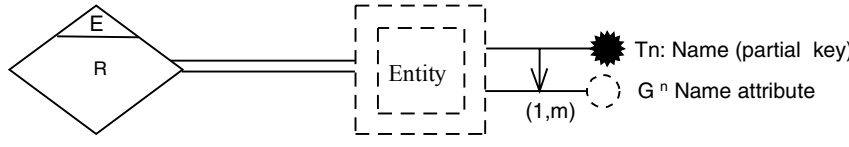
   b. Fuzzy aggregation of attributes:



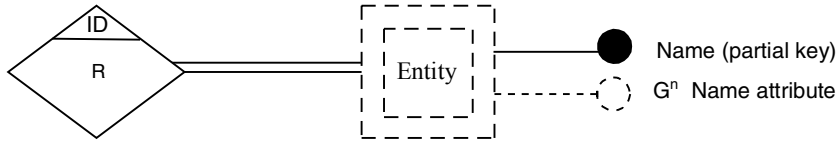
7. **Fuzzy entity as a fuzzy degree in the whole instance of an entity**, with meaning n (Definition 4.9):



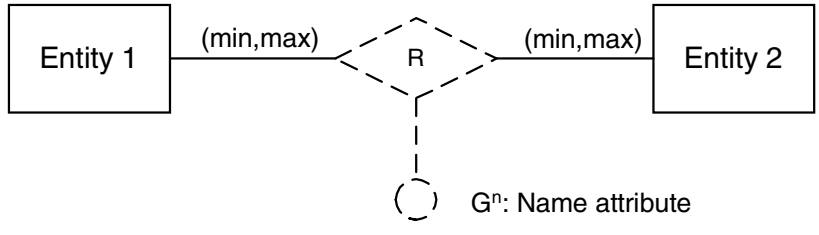
8. **Fuzzy weak entities** (Definition 4.10):
   a. Fuzzy weak entity due to dependency on existence:



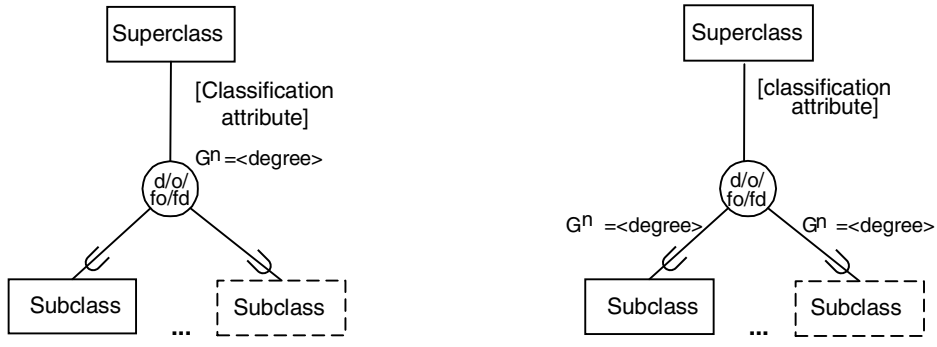
   b. Fuzzy weak entity due to dependency on identification:



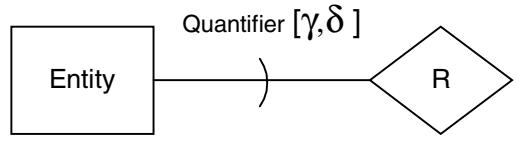
9. **Fuzzy relationships** (Definition 4.11): Degree in the relationships:



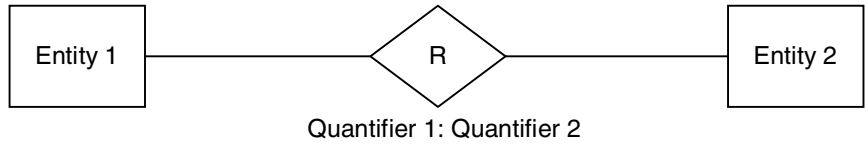
10. **Fuzzy degrees in specializations** (Definition 4.12): a) Degree in the specialization (left) and b) Degree in some subclasses (right):



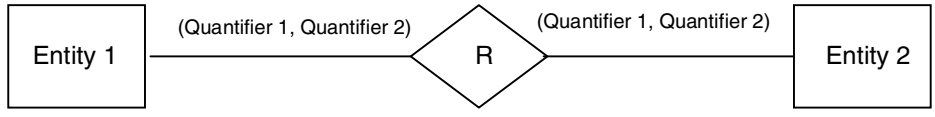
11. **Fuzzy participation constraint using one fuzzy quantifier with two thresholds** (Definition 4.13):



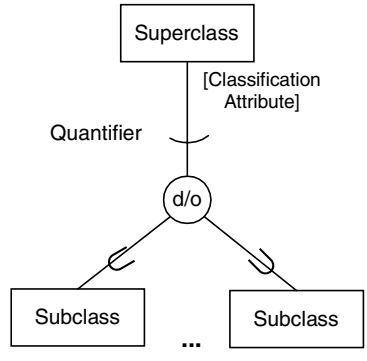
12. **Fuzzy cardinality constraint using two fuzzy quantifiers** (Definition 4.14):



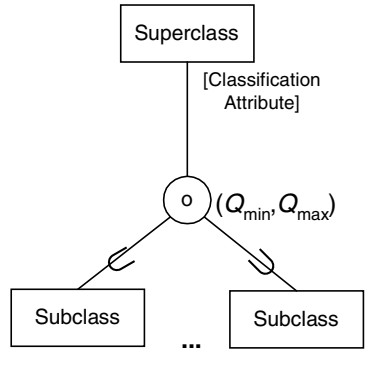
13. **Fuzzy (min, max) notation on relationships using fuzzy quantifiers** (Definition 4.15):



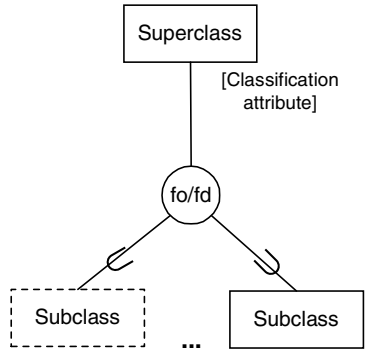
14. **Fuzzy completeness constraint on specializations with one quantifier** (Definition 4.16):



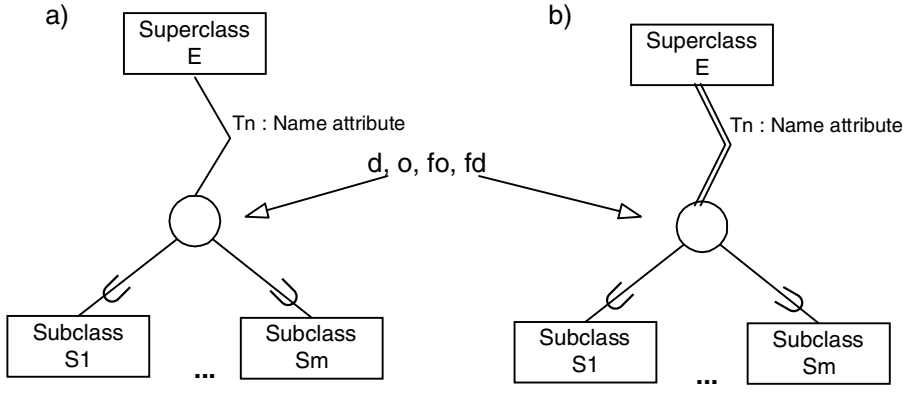
15. **Fuzzy cardinality constraint using the fuzzy (min, max) notation on overlapping specializations** (Definition 4.17):



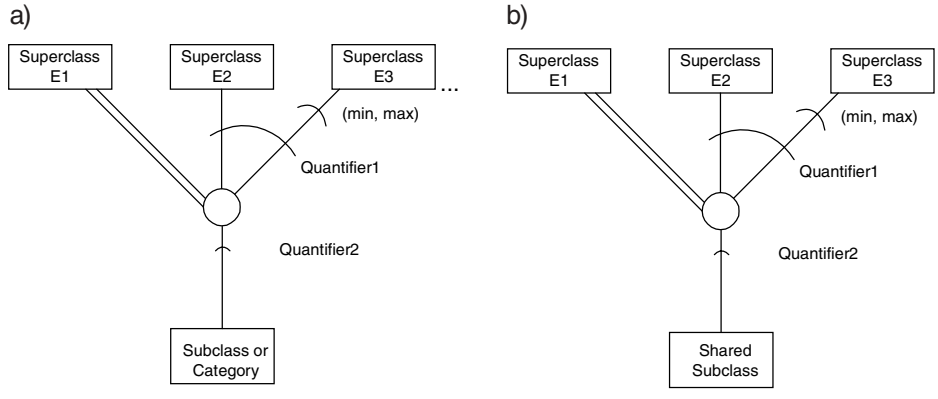
16. **Fuzzy disjoint and fuzzy overlapping constraint on specializations, with fuzzy subclasses** (Definitions 4.18 and 4.19):



17. **Fuzzy attribute-defined specializations** (Definition 4.20): a) with partial participation constraint and b) with total participation constraint:



18. **Fuzzy participation and completeness constraints in** a) Union types or categories (Definitions 4.21 and 4.22) (left) and b) Intersection types or shared subclasses (Definitions 4.23 and 4.24) (right):



Appendix B: FRDB Architecture: The FSQL Server

The FSQL Server is a program that allows users to employ the FSQL language (see Chapter VII). The FRDB and the FSQL Server have been implemented partially by using an already existing DBMS[1].

Actually, the FSQL Server does not include all the statements and clauses defined in the FSQL language. However, it is useful for the most usual operations and queries. Basically, using an existing DBMS involves three consequences:

1. The system will be slower than if it were programmed at a lower level.
2. The task is made much simpler (we do not have to build the DBMS).
3. We obtain all the advantages of the host DBMS (security, efficiency, etc.) without the server having to take them into account.

The DBMS chosen was Oracle because of its adaptability, its large extension, and its ability to program packages (with functions and procedures) internal to the system in its own language, PL/SQL, which turns out to be quite efficient. Of course, this architecture can be implemented in other DBMSs, and we are now working in an implementation using PostgreSQL[2].

The tests that we carried out prove that the FSQL Server is very fast due to its ability to function as a real-time server. Obviously, if the query is very complicated and the database is very large, even if the translation is fast, the information recovery may be a little slow; if the condition were long, then the DBMS would carry out many operations.

The usefulness of this server is clear (see Chapter VIII), because it is even useful in traditional databases, which use only fuzzy attributes Type 1. However, we would like to underline the possibilities the server could offer when used in combination with data-mining techniques (such as classification).

Data, FSQL Server, and FSQL Client

Basically, the architecture of the FRDB with the FSQL Server is as follows:

1. **Data**: Traditional database and fuzzy metaknowledge base (see Chapter V).
2. **FSQL Server**.
3. **FSQL Client**.

FSQL Server

The Oracle version of the FSQL Server has been programmed entirely in PL/SQL and includes three kinds of functions:

1. Translation Function (`FSQL2SQL`): This function carries out a lexical, syntactic, and semantic analysis of the FSQL query. If errors of any kind are found, it will generate a table with all those errors. If the query has no errors, it is translated into a standard SQL sentence, which includes reference to the following two kinds of functions.
2. Representation Function: This function is used to show the fuzzy attributes in a comprehensible way for the user and not in the internal format.
3. Fuzzy Comparison Function: This function is utilized to compare the fuzzy values and to calculate the compatibility degrees (`CDEG` function).

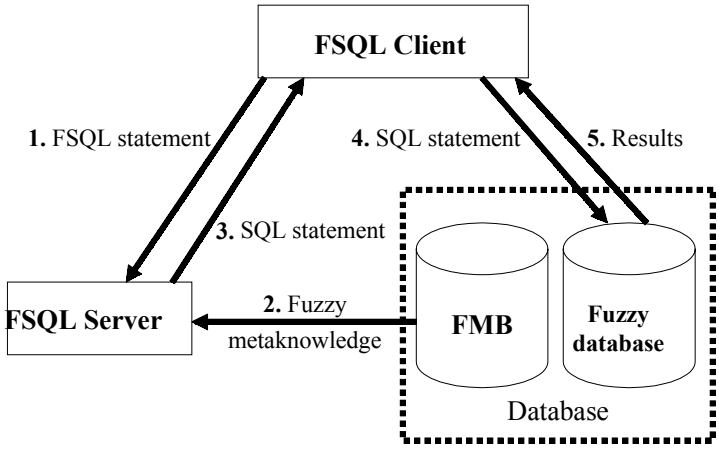
Summarizing, the translation function replaces the fuzzy attributes of the `SELECT` by calls to representation functions, the fuzzy conditions by calls to the fuzzy comparison functions, and the `CDEG` functions by calls to the fuzzy comparison functions and other functions if some logic operators exist (default functions are shown in Table 7.4).

The current version of the FSQL Server stores and controls some aspects about it and its processing: version, installation date, last utilization date, number of error for the last utilization and for all of them, number of access without errors, time employed for the last utilization and for all of them, and so forth.

FSQL Client

The FSQL Client is an independent program that serves as an interface between the user and the FSQL Server. The user introduces the FSQL query, and the client program communicates with the server and with the database in

Figure B.1. Basic architecture for the FRDB with the FSQL Server



order to obtain the final results. The translation function of the FSQL Server is the only function that the client directly executes. We have developed a FSQL Client for Windows, called FQ, Fuzzy Queries (Galindo, 1999). On the other hand, there exists another FSQL Client, called Visual FSQL, which allows the user to construct an FSQL query through mouse clicks, without the need to write much (Oliva, 2003; Galindo et al., 2004d).

Calling to the FSQL Server

We summarize the process of using the FSQL Server for queries in Figure B.1. In short, an FSQL query involves the following steps:

1. The FSQL Client program sends the FSQL query to the FSQL Server.
2. The FSQL Server analyzes the query and, if it is correct, generates an SQL sentence starting from the original query. In this step, the FSQL Server uses the information of the FMB.
3. After the query has been generated in SQL, the client program reads it.
4. The client program sends the SQL query to any database that is coherent with the FMB. In the execution of this query, functions of the FSQL Server are used (representation and fuzzy comparison functions).
5. Finally, the client receives the resulting data, which shows them.

Steps 3 and 4 could have been eliminated to increase the efficiency, but the method we have presented achieves independence between the translation phase (Steps 1, 2, and 3) and the consultation phase (Steps 4 and 5). As such, if we make use of a local database with FSQL Server and FMB, we will be able to translate our sentences locally and send the translated queries to a remote database, avoiding network overload with error messages, translated queries, and so forth. This way, the remote database would not have to have the translation function installed.

If the statement is not a query, the process is similar. That statement is analyzed, and the FSQL Server generates modifications in the FMB and/or in the storage structures (including the Data Dictionary of the DBMS).

The presented architecture is not ideal for a final product, but it allows us to evaluate the possibilities of an FRDB on a practical level rather than only on a theoretical one. Let us hope that the DBMS soon incorporates new types of internal data (see Chapter IV) that allow the storing of fuzzy values and the fuzzy processing of these values.

Endnotes

1 The actual version of the FSQL Server can be downloaded free from http://www.lcc.uma.es/~ppgg/FSQL

2 PostgreSQL is Open Source database software: http://www.postgresql.org

Appendix C: Acronyms and the Greek Alphabet

DBMS	DataBase Management System
DCL	Data Control Language
DDL	Data Definition Language
DM	Data Mining
DML	Data Manipulation Language
EER	Enhanced (or Extended) Entity-Relationship (model)
ER	Entity-Relationship (model)
ERD	Entity-Relationship Diagram
ExIFO	Extended IFO
FDNER	Fuzzy Diagrams Nested Entity Relationship
FEER	Fuzzy EER
FIRST	Fuzzy Interface for Relational SysTems
FMB	Fuzzy Metaknowledge Base
FOOD	Fuzzy Object-Oriented Diagrams Fuzzy Object-Oriented Data (model)
FOODB	Fuzzy Object-Oriented DataBase
FQ	Fuzzy Queries
FRDB	Fuzzy Relational DataBase
FRDBMS	Fuzzy Relational DataBase Management System
FRSL	Fuzzy Requirement Specification Language
FSQL	Fuzzy SQL
FT	Fuzzy Type
FTSQL2	Fuzzy Temporal SQL2
FUPME	Fuzzy Update Protocol Model Expressions
GEFRED	GEneralized model for Fuzzy RElational Databases
OMT	Object Modeling Technique
OSQL	Object SQL
RDB	Relational DataBase
RDBMS	Relational DataBase Management System
SDL	Storage Definition Language
SQL	Structured Query Language
UML	Unified Modeling Language

Appendix C: (cont.)

Letter (uppercase/lowercase)	Name
Α α	Alpha
Β β	Beta
Γ γ	Gamma
Δ δ	Delta
Ε ε, ∈	Epsilon
Ζ ζ	Zeta
Η η	Eta
Θ θ	Theta
Ι ι	Iota
Κ κ	Kappa
Λ λ	Lambda
Μ μ	Mu
Ν ν	Nu
Ξ ξ	Xi
Ο ο	Omicron
Π π	Pi
Ρ ρ	Rho
Σ σ, ς	Sigma
Τ τ	Tau
Υ υ	Upsilon
Φ φ, ϕ	Phi
Χ χ	Chi
Ψ ψ	Psi
Ω ω	Omega

About the Authors

José Galindo earned his PhD in computer science at the University of Granada, Spain, and is a professor of computer science in the School of Engineering at the University of Málaga, Spain. Galindo is also the author of several books and papers on computer science, databases, information systems, and fuzzy logic. His research interests include fuzzy logic, fuzzy databases, and ethical issues in the technological age. He is a member of the IDBIS research group and the RITOS-2 Ibero-American research net.

Angélica Urrutia Sepúlveda is an associate professor at the Catholic University of Maule, Chile, in the Computer Science Department. She is a member of the Chilean Computer Science Society and RITOS-2 (Red iberoamericana de tecnologías del software para la década del 2000) working group of CYTED. Urrutia is the founder and president of the Chilean Workshop on Databases. In 2003 she graduated sobresaliente cum laude with a PhD in computer science at the Castilla-La Mancha University, Spain. She earned her master's and engineer's degree in computer science from the Concepcion University and Santiago University, Chile, respectively. Her major research topics include fuzzy databases and information systems, about which she has authored several scientific papers.

Mario Piattini earned an MSc and a PhD in computer science from the Politechnical University of Madrid and an MSc in Psychology by the UNED, Spain. He is also a certified information system auditor and a certified information system manager by the Information System Audit and Control Association (ISACA). Piattini is a professor in the Department of Computer Science at the University of Castilla-La Mancha, in Ciudad Real, Spain, and is the author of several books and papers on databases, software engineering, and information systems. He also leads the ALARCOS research group of the Department of Computer Science at the University of Castilla-La Mancha. His research interests include advanced databases, database quality, software metrics, security and audit, and software maintenance.

* * *

Leoncio Jiménez received his bachelor's degree in IT and human science from the Free University of Brussels (ULB-Belgium). In 2000, he earned his master's degree in industrial systems from the National Polytechnic Institute of Toulouse (INPT-France) and in 2005, his doctorate in the same institution. Since 1997, he has been an assistant professor with the Computer Science and IT Department, Maule Catholic University (Chile). He is a permanent member of the Spanish & Latin American Software Technology Network (RITOS2, CYTED). His research areas are systems, complex systems analysis, and information management and knowledge management systems. He was the winner of the Fernand Gallais Trophy, awarded by the Ecole Nationale Supérieure des Ingénieurs en Arts Chimiques et Technologiques (ENSIACET - INPT) for obtaining the qualification "Very Honorable" when defending his doctor's thesis "Gestion des connaissances imparfaites dans les organisations industrielles. Cas d'une industrie manufacturière en Amérique Latine" (Fuzzy Knowledge Management. A Case Study in the Manufacturing Industry).

Juan M. Medina is an associate professor in the School of Computer Engineering, University of Granada, Spain. He is a member of the IDBIS (Intelligent DataBases and Information Systems) research group. Medina has authored many books and papers on databases and its applications. He is also the main researcher of several projects. His research interests include databases architecture and applications, fuzzy, logic and object-oriented databases, flexible content based retrieval, knowledge extraction

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z