UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Sistemas de sugerencias basadas en semejanza de entidades
Autores: Fernando Boucquez
Director: Dra. Claudia Pons
Codirector: Dra. Clara Smith
Carrera: Licenciatura en Informática (Plan 90)

Resumen

En una dinámica simplificada de mercado, entiendo por búsqueda a la acción que comienza un cliente al pedir un producto con determinadas características, y por sugerencias a la acción que comienza el vendedor al entender las características que el cliente está buscando y al ofrecer productos en los que éste pudiera estar interesado. Estas acciones también son replicadas por los sistemas de compras; un usuario puede buscar productos mientras los sistemas sugieren productos que pueden ser interesantes.

Por otro lado, los sistemas de información tradicionales almacenan y recuperan información precisa en bases de datos. Esta información es de algún modo una representación del mundo real. El problema es que para representar el mundo real muchas veces se utilizan términos del lenguaje natural donde su semántica depende del contexto o del observador. Investigadores y universidades han propuesto e implementado diferentes maneras de extender los sistemas de información y los lenguajes de consultas para representar esta naturaleza vaga de la información.

El objetivo de esta tesis es entender la dinámica de mercado, estudiar cómo almacenar y consultar información difusa usando las principales herramientas como FSQL y SQLf, e implementar búsquedas y sugerencias en un sistema ejemplo utilizando conjuntos difusos permitiendo términos del lenguaje natural.

Palabras Claves

Lógica Difusa, Bases de Datos Difusas, Consultas Difusas, SQLf, FSQL, Aplicaciones Web, Sugerencias, ORM

Trabajos Realizados

En esta tesis he realizado un estudio sobre cómo la lógica difusa puede solucionar el manejo de información subjetiva; en especial sobre bases de datos relacionales. He mostrado el estado del arte de bases de datos y lenguajes de consultas difusas, en especial FSQL y SQLf. Además he descrito la dinámica de consultas y sugerencias en aplicaciones, presentando una solución que utiliza semejanza de entidades y consultas difusas

Conclusiones

El desarrollo de esta tesis me permitió descubrir una línea importante de investigación que comenzó con la teoría de conjuntos difusos de Zadeh y que luego fue adaptada a las base de datos.
He reflejado en esta tesis cómo la información imprecisa puede ser encontrada en ciertos dominios y debe ser tenida en cuenta durante la definición de modelo de datos.
Considero que este trabajo puede servir como referencia al momento de diseñar e implementar sistemas con datos subjetivos y a la elección de las herramientas.

Trabajos Futuros

Como trabajo a futuro es posible extender la librería SQLf para que soporte consultas agrupadas, de modificación de datos y esquemas. Otro posible trabajo es estudiar base de datos OO puras con información subjetiva y su integración con sugerencias.

Fecha de la presentación: Junio 2012

Universidad Nacional de La Plata
Facultad de Informática

Tesis de Licenciatura en Informática

Sistemas de sugerencias basadas en semejanza de entidades

por

Fernando Leonel Boucquez

Directora: Dra. Claudia Pons

Co-directora: Dra. Clara Smith

La Plata, 2012

Índice general

Capítulo 1

Introducción

En un modelo simplificado del concepto tradicional de mercado, coexisten dos tipos de actores: **oferentes** y **demandantes**. Los oferentes han sido quienes tradicionalmente motivan y dan comienzo a la dinámica que tiene como resultante el mercado. Ellos son los que ponen a disposición de los demandantes diferentes productos, que estos a su vez eligen y adquieren. Esta interacción es entonces lo que llamamos **mercado**.

Haciendo un primer análisis de los sistemas de subastas o compras más exitosos de la web (Amazon, Mercado Libre, Ebay), nos encontramos con una dinámica similar. Los oferentes publican artículos que luego son buscados por los demandantes, que a su vez se van interiorizando de detalles y características específicas de los productos; redescubriendo y entendiendo así *lo que ofrece el mercado.*

Cuando se produce una negociación, un vendedor trata de entender lo que quiere el comprador; y a partir de ahí comienza una exploración para ver cómo se adecuan las diferentes alternativas que él le puede ofrecer (sus productos en stock, por caso). Esta tarea la realiza basándose en su conocimiento empírico. Trata de acercar las entidades reales a las entidades ideales.

Si un vendedor experimentado atiende a un **nuevo cliente** que llega a un comercio y solicita algún producto, le ofrecerá lo que se le está solicitando en la forma más cercana posible; pero seguramente también le ofrezca *lo que él sabe que realmente le están pidiendo.* Yendo a otro ejemplo, supongamos que en una tienda de artículos tecnológicos un cliente solicita un producto reconocido como de última generación de cierta marca. Podría ocurrir que ya exista un artículo de similares características, pero de un modelo superior. El vendedor hábil seguramente le sugerirá ese otro producto. Lo que está haciendo el comerciante no es más que decodificar el *verdadero pedido* del cliente, que en este caso no esta pidiendo en realidad el artículo X sino **"lo último"**.

Entiendo por **búsqueda** a la acción que comienza un cliente al pedir un

producto con determinadas características, y **por sugerencias** a la acción que comienza el vendedor al entender las características que el cliente está buscando y al ofrecer productos que éste pudiera estar interesado.

Estas acciones también son replicadas por los sistemas de compras. Un usuario puede buscar productos mientras los sistemas sugieren productos que pueden ser interesantes.

Por otro lado, los sistemas de información tradicionales almacenan y recuperan información precisa en base de datos. Esta información es de algún modo una representación del mundo real. El problema es que para representar el mundo real muchas veces se utilizan términos del lenguaje natural donde su semántica depende del contexto o del observador. Los deseos del comprador y lo que infiere el vendedor generalmente están representados en lenguaje natural.

Durante los últimos años diferentes investigadores y universidades han propuesto e implementado diferentes maneras de extender los sistemas de información y lenguajes de consultas para poder representar esta naturaleza vaga del lenguaje. Una de las principales corrientes de investigación es la utilización de elementos de la lógica difusa de Zadeh [Zad65] en motores de base de datos relacionales, en particular extendiendo el lenguaje SQL. Dentro de estos trabajos se encuentran SQLf presentado por Bosc-Pivert [Bos95], FSQL presentado por Galindo-Medina-Pons-Cubero ([GMPC98]), Soft-SQL de Bordogna-Psaila [BP05] y SummarySQL de Rasmussen-Yager [RY96]. Cada uno de estos trabajos da puntapié a una basta serie de artículos y extensiones.

En general, cuando se busca un producto se definen algunas características a través de determinados parámetros precio, marca, antigüedad, etc; pero al establecer un valor definido, por ejemplo que un producto valga hasta 1000 pesos, probablemente no se esté tomando una decisión taxonómica. En estos casos seguramente se está pidiendo algo *aproximadamente* de hasta ese valor

¿Pero qué ocurre con un producto de 1001 pesos? Si fuésemos estrictos (como suelen serlo las consultas clásicas), este producto quedará **excluido** de los resultados y posiblemente estemos dejando fuera artículos de interés para el comprador. Esto es así porque la búsqueda se basa en consultas que utilizan lógica proposicional clásica que encierra el concepto del *tercero excluido*. En este entorno no podemos expresar que hay artículos que, a pesar de no coincidir estrictamente con todos los parámetros, bien podrían ser interesantes. También es importante contar con manera de extraer las características del artículo para poder sugerir productos similares y tener la posibilidad de medir que tan interesantes o similares son los productos. Creo que tanto en las consultas como en la extracción de las características importantes la lógica difusa puede ser una herramienta efectiva.

El objetivo de esta tesis es implementar búsquedas y sugerencias utilizando la lógica difusa como herramienta. Para ello analizaré qué facilidades ofrecen las soluciones de estos autores. Me concentraré en dos de las soluciones más impor-

tantes: SQLf y FSQL. Mostraré con ejemplos sencillos el poder expresivo extra que las consultas difusas proporcionan a SQL. Este análisis me ayudará a elegir la herramienta y la técnica correcta que luego será integrada a una aplicación web ejemplo de inmobiliarias.

1.1. Estructura del resto del documento

- El Capítulo 2 es una recolección de conceptos y definiciones básicos necesarios para la compresión de la tesis. Se exponen las nociones de lógica y conjuntos difusos.
- En el Capítulo 3 se exponen los conceptos del manejo de información impresa, de consultas difusas y se explica en profundidad lo que SQLf brinda. Se extenderá un sistema sencillo permitiendo realizar consultas difusas y mediante ejemplos claros y fáciles de entender se relacionarán los conceptos de la lógica difusa estudiados en el Capítulo 2 con las consultas difusas.
- El Capítulo 4 analiza el concepto de persistir información de naturaleza imprecisa para luego ser consultada. Se presentará un resumen del modelo GEFRED ([MPM94]) y se estudiará FSQL mostrando sus diferencias con SQLf tanto en especificación como en implementación.
- El Capítulo 5 integra las consultas difusas a la aplicación web Inmobiliaria. Se estudiará cómo integrar y ejecutar consultas difusas en ambientes orientados a objectos. Este capítulo también explica el concepto de sugerencias y como pueden ser implementadas con lógicas difusas.
- El Capítulo 6 presenta las conclusiones de esta tesis y el trabajo a futuro.

Capítulo 2

Lógica difusa

2.1. Introducción

La lógica difusa (*fuzzy logic*) fue abordada por primera vez por el Ingeniero Lofty A. Zadeh [Zad65]. La gran diferencia con la lógica *clásica* es que la lógica difusa es multivaluada, permitiendo valores intermedios definidos entre los valores convencionales verdadero y falso. Expresiones como *más o menos alto* y *muy veloz* pueden ser formuladas matemáticamente y así, interpretadas por una computadora. Esta nueva manera de expresar *estructuras lógicas* permite un acercamiento al mecanismo de *pensamiento humano* a la hora de programar (codificar) un dominio con una raíz lógica subyacente.

La lógica difusa introdujo el concepto de conjunto difuso (*fuzzy set*) basado en la idea de que los elementos sobre los que se construye el pensamiento humano son *etiquetas lingüísticas*. La lógica difusa permite representar el conocimiento común en un lenguaje matemático a través de la teoría de conjuntos difusos y funciones características asociadas a ellos. Claro está que las formas lingüísticas son inherentemente menos precisas que las numéricas, pero en muchas ocasiones aportan información útil para el razonamiento humano. Como usualmente se suele decir 'las cosas no son siempre blancas o negras, afortunadamente o no muchas veces hay grises'.

El campo de aplicación de la lógica difusa ha sido fundamentalmente el desarrollo de controles industriales y el manejo de información y consultas imprecisas como la que aquí tratamos.

2.2. Teoría de conjuntos difusos

Uno de los conceptos más difundidos de la matemática moderna es el de **conjunto** (*crisp set*). El primer ejemplo que utilizó Zadeh para ilustrar la noción

de conjunto difuso fue el conjunto de *hombre altos*.

Si consideramos los conjuntos de la lógica clásica, al conjunto de *hombre altos* pertenecerían por ejemplo solo aquellas personas cuya estatura sea mayor a un cierto valor, por caso 180 centímetros. Así es que todos los hombres de esa altura o más altos pertenecerían a él; mientras que los hombres con una estatura menor quedarían excluidos. Una persona de 150 centímetros no pertenecería al conjunto, pero tampoco una de 179 centímetros, mientras que otra de 180 sí. En este sentido puede parecer estricto o poco *natural* pensar que una persona es denominada *alta* y otra *no alta* cuando su diferencia es de solo un centímetro. Este enfoque puede ser aceptable para algunos tipos de aplicaciones o dominios pero para otros es claramente demasiado rígido.

Podemos ver a los conjuntos difusos como una *extensión* de los conjuntos tradicionales en donde se relaja la condición de tener una frontera clara o taxonómica sobre cuándo un elemento pertenece o no a él. Los conjuntos difusos contemplan la idea de pertenencia parcial de un elemento debido a la *imprecisión*, *ambigüedad* o *subjetividad* subyacente a determinada información; de manera que su grado de pertenencia está dado por un número real entre 0 y 1 (*fuzzy degree*) relacionado directamente con dicho valor. Un elemento pertenece completamente a un conjunto cuando el grado de pertenencia es 1 y no pertenece cuando es 0. Nótese que en base a esta última definición podemos pensar la lógica difusa como una extensión a la lógica clásica. Utilizando conjuntos difusos ahora podemos decir que por ejemplo una persona de 1,79 centímetros de altura pertenece con un grado 0,9 al conjunto *hombre altos* o que una persona de 173 centímetros pertenece con un grado 0,3.

En este capítulo me dedicaré a introducir algunas nociones elementales sobre la teoría de conjuntos difusos. Estas nociones son una recopilación de la gran cantidad de trabajos dedicados a este tema. Varios de estos conceptos son obtenidos de [Zad65], [Zad75] y [Zim85] y adaptados al marco de esta tesis. En la tesis de grado [CMC05] también se puede encontrar un excelente capítulo introductorio a la lógica difusa pero orientado a los controles difusos que difiere de los objetivos de esta tesis.

Como ya se mencionó, trabajaré sobre una teoría que se apoya a su vez en la teoría de conjuntos clásica; por lo que muchas de las definiciones que usaré las tomaré de allí. Desde luego necesitaré redefinir o extender algunos conceptos.

Ejemplo 1. Si trabajamos con la *temperatura del cuerpo humano*, los posibles valores oscilarán seguramente en el rango comprendido entre 35 y 45 grados centígrados. Esto es lo que llamamos universo de discurso U y que podemos definir simbólicamente de la siguiente manera:

$$U = \{u \in \mathbb{R} \mid 35 \text{ centígrados} \leq u \leq 45 \text{ centígrados}\}$$

En cambio, si trabajamos con la variable *sabor de la comida*, el universo de discurso U podría definirse como:

$$U = \{dulce, amargo, salado, agrio, ...\}$$

Nótese que en el primer caso al tomar como dominio un rango cerrado y continuo del universo de los números reales estamos trabajando sobre un conjunto infinito; mientras que en el segundo caso al poder definir el dominio por extensión, estamos considerando un conjunto finito y discreto. Formalizando lo antedicho:

Definición 1 (Universo de discurso)**.** *El* ***universo de discurso*** *U es el dominio sobre el que cada variable, tanto de entrada como salida, toma valores. El universo de discurso puede ser un dominio continuo o un conjunto discreto.*

2.2.1. Conjuntos tradicionales contra conjuntos difusos

En la lógica clásica, un elemento u pertenece o no a un conjunto A, subconjunto del universo de discurso U. La pertenencia (o no pertenencia) es típicamente definida con valores 0 y 1.

$$A \Rightarrow \mu_a(u) = \begin{cases} 1 & \text{if } u \in A \\ 0 & \text{if } u \notin A \end{cases} \tag{2.1}$$

En la lógica difusa, un elemento pertenece a un conjunto difuso A del universo de discurso U con un grado de pertenencia $\mu_a(u) \in [0, 1]$ y ese valor es proporcional a la relevancia del elemento en el conjunto A. La relación entre los valores del universo en discurso y el grado de pertenencia son caracterizados por un conjunto de pares ordenados:

Definición 2 (Conjunto difuso)**.** *Un conjunto difuso (predicado o relación) A sobre el universo de discurso U es un conjunto de pares*

$$A = \{(x, \mu_A(x)) | x \in U, \mu_A(x) \in [0, 1]\} \tag{2.2}$$

donde $\mu_A(x)$ es el grado de pertenencia del elemento x en el conjunto difuso A.

$\mu_A(x) = 0$ indica que x no pertenece en absoluto al conjunto difuso A
$\mu_A(x) = 1$ indica que x pertenece totalmente al conjunto difuso A
$\mu_A(x) = 0{,}6$ indica que x pertenece con un grado 0,6 al conjunto difuso A

Si la función de pertenencia solo produce valores $\{0, 1\}$ entonces el conjunto que genera es un conjunto tradicional. Se pueden definir conjuntos tradicionales usando conjuntos difusos.

Se puede definir un conjunto difuso exhaustivamente definiendo todos sus pares, o podemos definir una función $\mu_A(x)$ llamada función de pertenencia o característica.

En general definimos los conjuntos exhaustivamente cuando el dominio toma valores discretos.

Ejemplo 2. Sobre el dominio Color:

$$Color = \{Amarillo, Blanco, Negro, Rojo, Azul, Verde, Celeste\}$$

podemos definir el conjunto difuso *Brillante* como:

$$Brillante = \{(0{,}8; Amarillo), (1; Blanco), (0{,}4; Rojo), (0{,}1; Celeste)\}$$

Utilizamos funciones de pertenencia para definir conjuntos cuando el dominio toma valores continuos.

Ejemplo 3. El conjunto difuso *alto* esta definido sobre el conjunto de enteros y utiliza la siguiente función de pertenencia.

$$\mu_{alto}(x) = \begin{cases} 0 & x \leq 170 \\ x - 180/10 & 170 < x \leq 180 \\ 1 & 180 < x \end{cases}$$

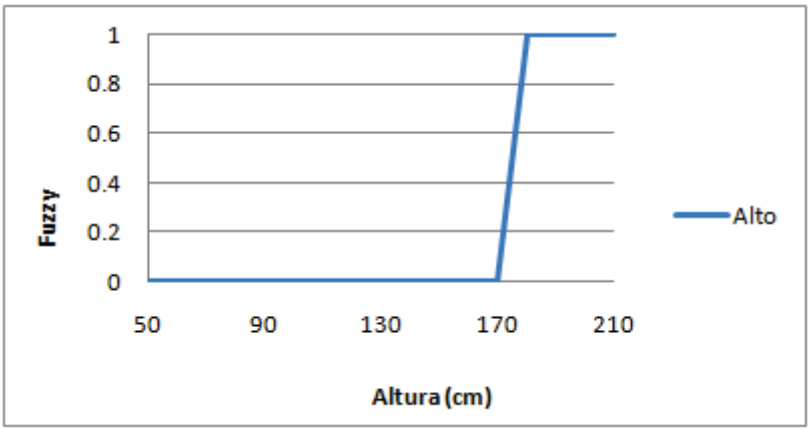

Figura 2.1: Función de pertenencia del conjunto difuso *alto*

Tanto el identificador *brillante* como *alto* con la connotación que lleva asociado su conjunto difuso reciben el nombre de *etiqueta lingüística*. Este concepto será explicado con mayor detalle en la sección 2.5.

2.3. Conceptos elementales sobre Conjuntos difusos

Sobre los conjuntos difusos, como con todo elemento algebraico, se pueden aplicar funciones y operaciones. Haremos una primer aproximación intuitiva a algunos de estos conceptos, para luego formalizar las definiciones. Estas definiciones formales fueron recopiladas de la tesis de doctorado [Góm99]. La idea general

será *extender* la relación de pertenencia entre elementos y conjuntos incluyendo el grado de pertenencia entre ambos.

En la lógica clásica decimos que dos conjuntos A y B son iguales si todos los elementos pertenecientes a A también pertenecen a B y viceversa. Análogamente, decimos que en los conjuntos difusos la igualdad se da cuando el grado de pertenencia de cada elemento respecto a ambos conjuntos es el mismo.

Definición 3 (Igualdad de conjuntos difusos)**.** *Dados los conjuntos difusos A y B sobre el dominio U se dicen iguales si cumplen:*

$$A = B \iff \forall x \in U, \mu_A(x) = \mu_B(x)$$

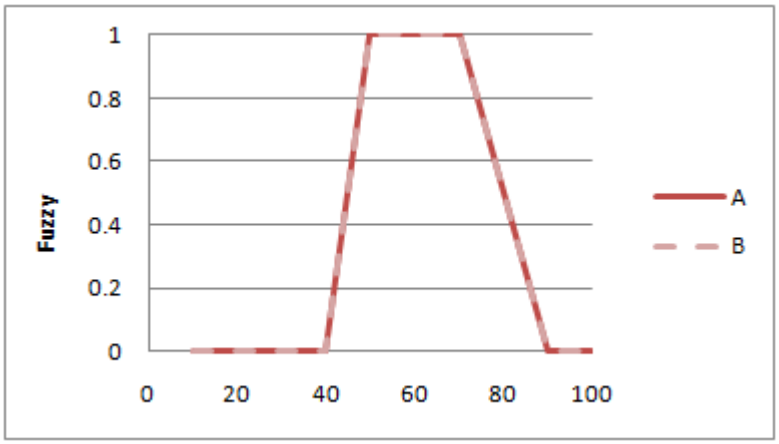

Figura 2.2: Igualdad de conjuntos difusos

Notemos cómo se mantiene la idea inicial de que la lógica difusa extiende los elementos de las teorías clásicas; la definición es valida para conjuntos clásicos donde solo consideramos grados de pertenencia 0 y 1.

Siguiendo esta noción intuitiva, definimos la inclusión de un conjunto A en uno B en la lógica difusa cuando el primero conjunto *cubre* el segundo.

Definición 4 (Inclusión de conjuntos difusos)**.** *Dados 2 conjuntos difusos A y B sobre el dominio U decimos que:*

$$A \subseteq B \iff \forall x \in U, \mu_A(x) \leq \mu_B(x)$$

Una idea interesante puede ser la de agrupar aquellos elementos que tienen aunque sea mínimamente un grado de relevancia en una relación separando aquellos que no tienen ninguna relación.

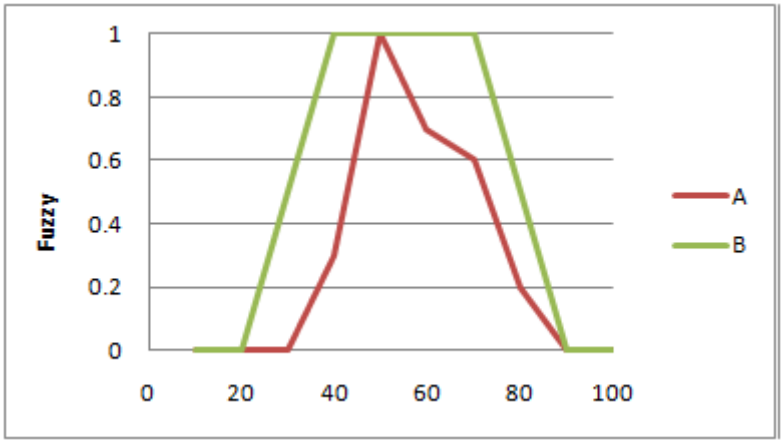

Figura 2.3: Inclusión de conjuntos difusos

Podemos imaginar algo así en la aceptación a un partido político, podemos pensar en diferentes grados de aceptación por un lado y en aquellos que se declaran completamente opositores. Los primeros conjunto define el *soporte del conjunto.*

Definición 5 (Soporte de conjuntos difusos)**.** *El soporte de un conjunto difuso A definido sobre U es un subconjunto de dicho universo que satisface:*

$$support(A) = \{x \in U, \mu_A(x) > 0\}$$

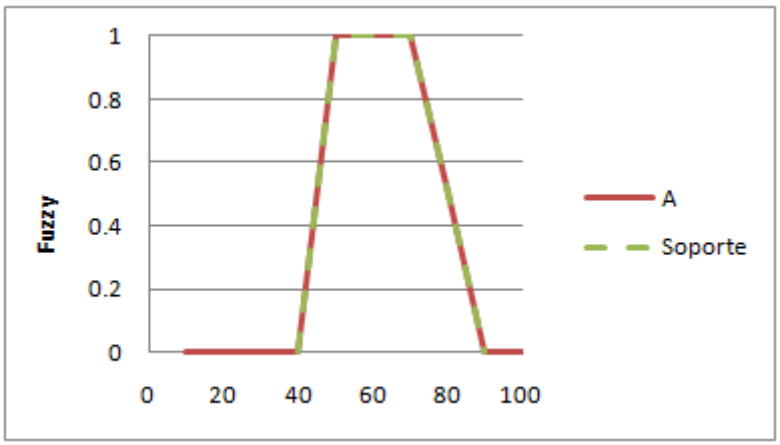

Figura 2.4: Soporte de un conjunto difuso

Siguiendo con la idea anterior podríamos agrupar a aquellas personas con intencionalidad de voto a ese partido político. Para eso podemos definir un valor determinado de grado de relevancia (por ejemplo 0,7) y presumir que aquellos

elementos que se encuentren por encima de ese valor se inclinan a votarlo. Ese conjunto se denomina α-corte.

Definición 6 (α-corte de un conjunto difuso)**.** *Un α-corte de un conjunto difuso A sobre U, denotado como A_α es un subconjunto no difuso de elementos de U cuya función de pertenencia toma un valor mayor o igual a algún valor concreto α de dicho universo que satisface:*

$$A_\alpha = \{x : x \in U, \mu_A(x) \geq \alpha\}$$

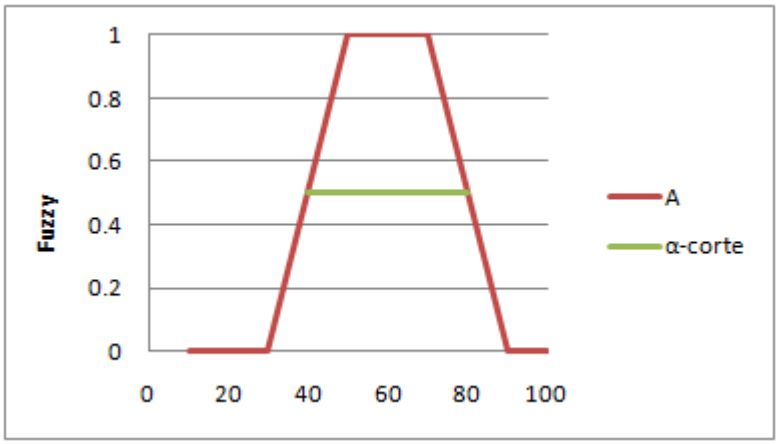

Figura 2.5: α-corte de un conjunto difuso

Continuando con el ejemplo anterior podemos ver el conjunto de aquellos electores seguros siendo estrictos y considerando solo aquellos que tengan un grado de relevancia de 1. En términos de la lógica difusa estos son el núcleo del conjunto.

Definición 7 (Núcleo de un conjunto difuso)**.** *El núcleo (core) de un conjunto difuso A sobre U es un subconjunto de dicho universo que satisface:*

$$core(A) = \{x \in U, \mu_A(x) = 1\}$$

Cabe destacar que los conjuntos suporte, α-corte y núcleo son conjuntos clásicos.

Si tomamos la realidad acerca de las ideas políticas de un partido, tal vez nadie está completo acuerdo. Es en este caso que podemos medir cuál es el máximo grado de afinidad que se podría tener. Surge así el concepto de altura:

Definición 8 (Altura de un conjunto difuso)**.** *La altura (height) de un conjunto difuso A sobre U se define como:*

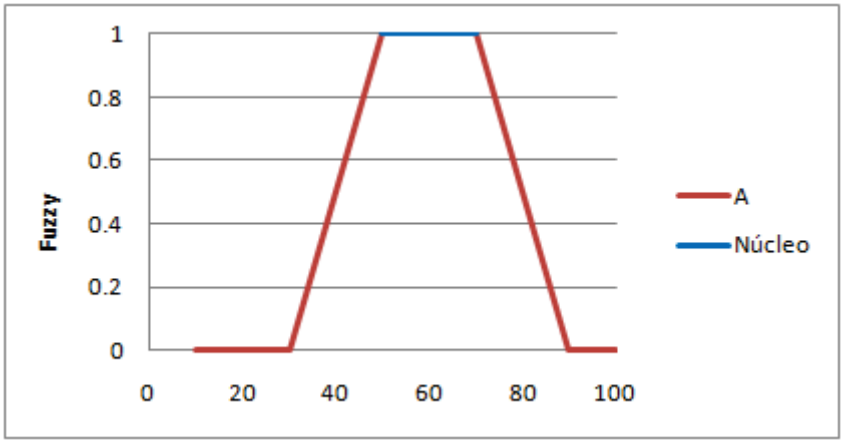

Figura 2.6: Núcleo de un conjunto difuso

$$height(A) = \sup_{x \in U} \mu_A(x)$$

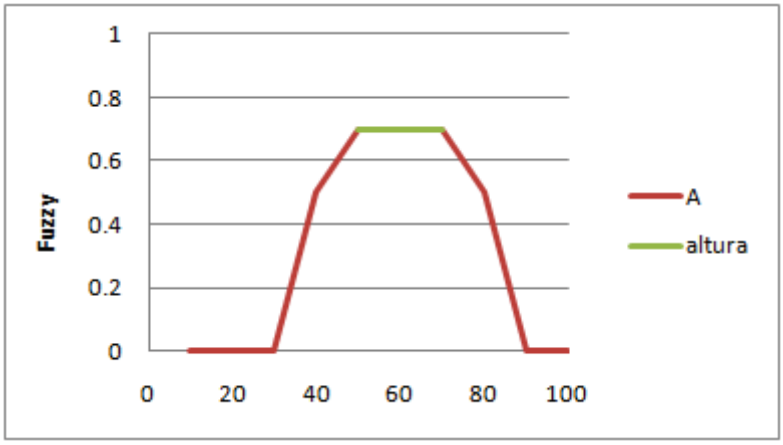

Figura 2.7: Altura de un conjunto difuso

Definición 9 (Conjunto difuso normalizado)**.** *Un conjunto difuso A sobre U se dice normalizado si y sólo si*

$$\exists x \in U, \mu_A(x) = height(A) = 1$$

En la figura 2.6 se puede ver que el conjunto difuso A es normalizado.

2.4. Conjuntos difusos sobre dominios continuos

Como se vio, las gráficas *describen* de alguna manera la relación de los elementos con los conjuntos difusos de acuerdo a la función que determina el grado de pertenencia. A continuación mostraremos las funciones continuas más utilizadas por su simplicidad matemática y manejabilidad. También mostraremos una forma de definir conjuntos difusos usando estas funciones que serán utilizadas durante la presente tesis, y en particular, al momento de realizar consultas. La primera de ellas define formalmente un trapezoide en este contexto, que como se pudo apreciar en el apartado anterior es una figura usual en este dominio:

$$\mu_{Trapezoid(h,\alpha,\beta,\gamma,\epsilon)}(x) = \begin{cases} 0 & x \leq \alpha \\ (x-\beta)h/\beta - \alpha & \alpha < x \leq \beta \\ h & \beta < x \leq \gamma \\ (\epsilon - x)h/\epsilon - \gamma & \gamma < x \leq \epsilon \\ 0 & \epsilon < x \end{cases} \tag{2.3}$$

Donde h es la altura del trapezoide y se cumple que $\alpha \leq \beta \leq \gamma \leq \epsilon$. Los argumentos (10) $h, \alpha, \beta, \gamma, \epsilon$ pueden ser *configurados* modificando el comportamiento de la función.

Definición 10 (Argumentos de una función de pertenencia)**.** *Defino argumentos de una función de pertenencia a las constantes de la función que pueden ser configuradas y modifican su curva.*

En la función clásica lineal $f(x) = mx+b$ podemos definir m (la pendiente) y b (el punto de corte del eje x e y) como argumentos. La etiqueta lingüística *promedio* sobre la altura de las personas está definida como $\mu_{Trapezoid(1,165,170,180,185)}$ donde 1, 165, 170, 180 y 185 son los argumentos de la función $Trapezoid$ que definen le pendiente del trapezoide.

Es posible definir otras funciones de pertenencia que son especializaciones o *instancias* de $Trapezoid$ u otras:

$$\mu_{NormalTrapezoid(\alpha,\beta,\gamma,\epsilon)} = \mu_{Trapezoid(1,\alpha,\beta,\gamma,\epsilon)} \tag{2.4}$$

$$\mu_{Triangular(h,\alpha,\beta,\gamma)} = \mu_{Trapezoid(h,\alpha,\beta,\beta,\gamma)} \tag{2.5}$$

$$\mu_{NormalTriangular(\alpha,\beta,\gamma)} = \mu_{Triangular(1,\alpha,\beta,\gamma)} \tag{2.6}$$

$$\mu_{RightTrapezoid(h,\alpha,\beta)} = \mu_{Trapezoid(h,\alpha,\beta,\infty,\infty)} \tag{2.7}$$

$$\mu_{NormalRightTrapezoid(\alpha,\beta)} = \mu_{RightTrapezoid(1,\alpha,\beta)} \tag{2.8}$$

$$\mu_{LeftTrapezoid(h,\alpha,\beta)} = \mu_{Trapezoid(h,-\infty,-\infty,\alpha,\beta)} \tag{2.9}$$

$$\mu_{NormalLeftTrapezoid(\alpha,\beta)} = \mu_{LeftTrapezoid(1,\alpha,\beta)} \tag{2.10}$$

$$\mu_{Like(c,e1)} = \mu_{Triangular(1,c-e1,c,c+e1)} \tag{2.11}$$

$$\mu_{Around(c,e1,e2)} = \mu_{Trapezoid(1,c-e2,c-e1,c+e1,c+e2)} \tag{2.12}$$

Como lo son trapezoide normal (2.4), triangular (2.5), triangular normal (2.6), trapezoide a derecha (2.7), trapezoide normal a derecha (2.8), trapezoide a izquierda (2.9), trapezoide normal a izquierda (2.10), parecido (2.11) y alrededor (2.12).

Otras funciones comunes son la Gausiana (2.13) y la igualdad difusa (2.14)

$$\mu_{Gaussian(\mu,k)}(x) = e^{-(x-\mu)^2/k^2} \tag{2.13}$$

$$\mu_{Equal(c)}(x) = \frac{\text{mín}(x,c)}{\text{máx}(x,c)} \tag{2.14}$$

2.4.1. Conjuntos difusos sobre dominios discretos

Como ya se ha tratado en la definición 2, los conjuntos difusos sobre dominios discretos son generalmente definidos por extensión sobre elementos de ese dominio. Dentro de este tipo de conjuntos difusos nos detendremos en uno en particular: *similitud*.

Definición 11 (Similitud)**.** *Definimos similitud (similarity) al grado de semejanza que hay entre dos elementos de un dominio discreto. Sobre un dominio de discurso U con n elementos, podemos definir n conjuntos difusos de similitud:*

$$Similarity(i) = \{(\mu_{Similarity(i)}(j), j) \mid i, j \in U\} \tag{2.15}$$

Donde $\mu_{similarity(i)}(j)$ es el grado de semejanza $[0,1]$ que hay entre los elementos i y j.

Formando una relación entre los elementos del dominio que en general es simétrica (2.16) y es reflexiva (2.17)

$$\mu_{Similarity(i)}(j) = \mu_{Similarity(j)}(i) \tag{2.16}$$

$$\mu_{Similarity(i)}(i) = 1 \tag{2.17}$$

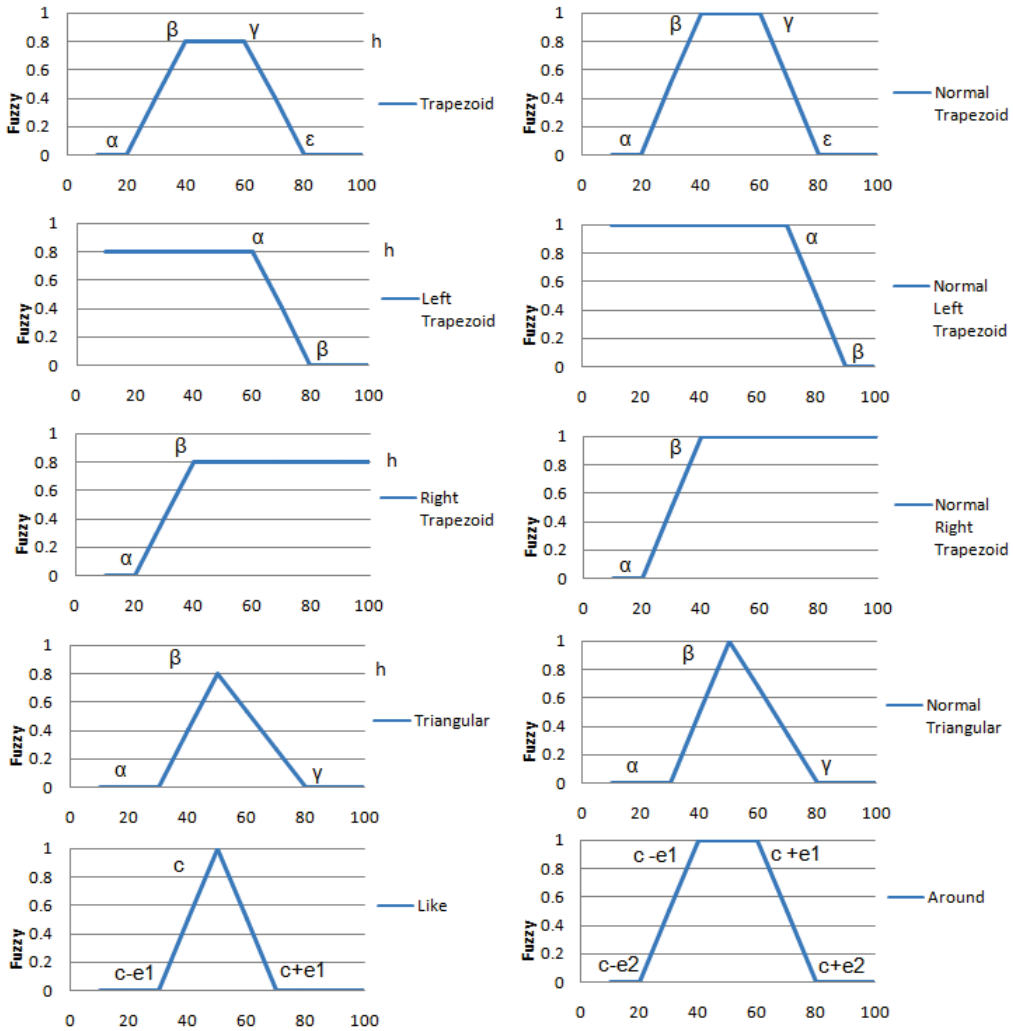

Figura 2.8: Funciones de pertenencias trapezoidales

Ejemplo 4. Usando el dominio $Color = \{Amarillo, Blanco, Negro, Rojo, Azul, Verde, Celeste\}$ definido en el ejemplo 2 podemos definir 7 conjuntos discretos. Donde uno de ellos es por ejemplo la similitud que tienen los colores con el color blanco

$$Similarity(Blanco) = \{(Blanco, 1), (Amarillo, 0{,}5), (Celeste, 0{,}2), ..\}$$

2.5. Variable y etiquetas lingüísticas

A menudo queremos describir el estado de un objeto o fenómeno usando conceptos abstractos como por ejemplo *temperatura*, *altura*, *claridad*, etc. En la lógica difusa esos conceptos pueden ser expresados utilizando las llamadas

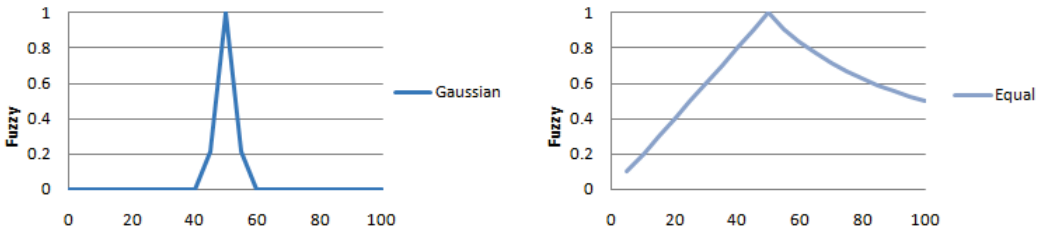

Figura 2.9: Otras funciones difusas comunes

variables lingüísticas: Los valores que pueden tomar estas variables se denominan etiquetas lingüísticas.

Definición 12 (Etiqueta lingüística)**.** *Es aquella palabra en lenguaje natural que expresa un estado de un objeto.*

Con esta definición, pretendemos afirmar que en nuestra vida cotidiana utilizamos multitud de etiquetas lingüísticas para expresar conceptos abstractos: *joven*, *viejo*, *frío*, *caliente*, *barato*, *caro*, *limpio*, *sucio*, etc. La definición intuitiva de esas etiquetas no sólo puede variar de un individuo a otro y del momento particular, sino que también varía de acuerdo al contexto en el que se aplique. Por ejemplo, seguramente no tendrán la misma altura una persona *alta* y un edificio *alto*. La representación de conjuntos difusos puede ser variada y depende, fundamentalmente de la naturaleza del universo de discurso (establece el contexto) sobre el que definamos el conjunto difuso.

Definición 13 (Variable lingüística)**.** *Una variable lingüística es un conjunto que agrupa etiquetas lingüísticas que están lógicamente relacionadas.*

Por ejemplo, la variable lingüística *estatura de personas* agruparía las etiquetas lingüísticas *bajo*, *mediano*, *alto* pero no agruparía la etiqueta *frío*. Esto es posible ya que las primeras tres etiquetas están definidas sobre conjuntos que operan sobre la altura (en centímetros) de las personas, mientras que la última sobre la temperatura (en centígrados) del ambiente.

En la lógica difusa una etiqueta lingüística esta definida con un conjunto difuso, y una variable lingüística esta definida con una partición difusa.

Hay variables cuya definición es más compleja porque se mueven en dominios subyacentes poco claros y no es natural trasladarlos a valores numéricos: *sabiduría*, *honestidad*, etc.

La siguiente definición es una definición más formal de variable lingüística también encontrada en [Zad75]. Esta definición ayuda a relacionar una variable lingüística con las etiquetas lingüísticas básicas y cómo la gramática ayuda a generar nuevas etiquetas. También cómo cada etiqueta esta relacionada con un conjunto difuso.

Definición 14 (Variable lingüística, definición formal)**.** *Una variable lingüística es una 5-tupla:* $< N, U, T(N), G, M >$

- N *es el nombre de la variable y* U *dominio subyacente.*
- $T(N)$ *es el conjunto de términos o etiquetas que puede tomar* N*.*
- G *es una gramática para generar las etiquetas de* $T(N)$*: muy alto, no muy bajo, extremadamente normal, bajo y normal, etc.*
- M *es una regla semántica que asocia cada elemento de* $T(N)$ *con un conjunto difuso en* U *de entre todos los posibles:* $M : T(N) \to F(U)$

Para la definición de la gramática G utilizaremos los modificadores lingüísticos. Estos están desarrollados en la sección 2.7.

Definición 15 (Partición difusa)**.** *Una partición difusa es simplemente un conjunto de funciones de pertenencia cuyas etiquetas lingüísticas están lógicamente relacionadas. Una variable lingüística expresa una partición difusa. Definimos la partición* S *como*

$$S = \{\mu_{A_i}(x) | i = 1..s\}$$

Donde S *es la cantidad de conjuntos difusos que forman la partición. En lógica difusa, una partición expresa una variable lingüística.*

El tamaño de una partición y la definición de sus funciones de pertenencia pueden ser diferentes entre las personas.

Ejemplo 5. Definimos la variable lingüística *Temperatura* y cuatro funciones de pertenencia con sus etiquetas lingüísticas *helado*, *fresco*, *templado* y *caluroso*. En este ejemplo, la variable lingüística agruparía las etiquetas lingüísticas.

Las funciones de pertenencia de cada etiqueta son:

$$\mu_{helado}(x) = \mu_{NormalLeftTrapezoid(5,10)}(x)$$

$$\mu_{fresco}(x) = \mu_{NormalTrapezoid(0,10,15,25)}(x)$$

$$\mu_{templado}(x) = \mu_{NormalLike(25,10)}(x)$$

$$\mu_{caluroso}(x) = \mu_{NormalRightTrapezoid(25,35)}(x)$$

La partición difusa se define como:

$$Temperatura = \{\mu_{helado}(x), \mu_{fresco}(x), \mu_{templado}(x), \mu_{caluroso}(x)\}$$

La figura 2.10 muestra como son esas funciones y su relación son el universo en discurso.

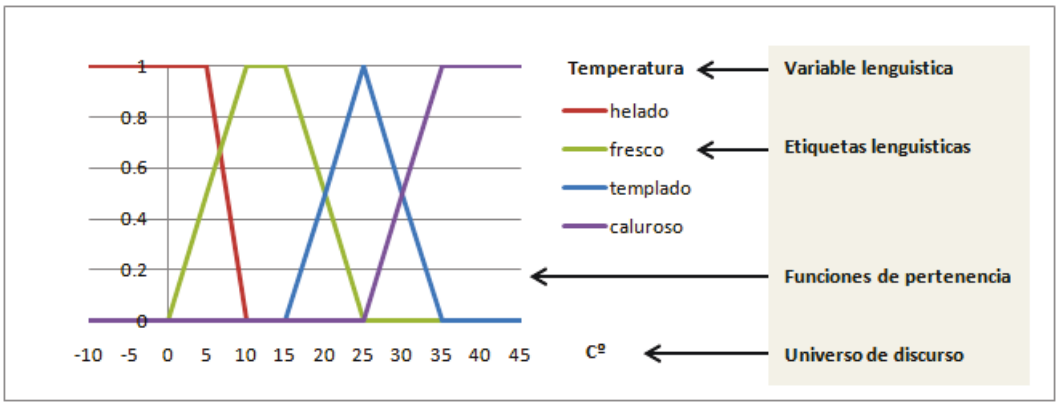

Figura 2.10: Variable lingüística *Temperatura* y sus conjuntos difusos

Ejemplo 6. Podemos extender el ejemplo 4 definiendo la variable lingüística *Color similar* como la suma de todos los conjuntos difusos de similitud sobre el conjunto *Color*:

Sea el conjunto $Color = \{Amarillo, Blanco, Negro, Rojo, Azul, Verde, Celeste\}$, definimos las etiquetas lingüística:

$similiarAmarillo = similarity(Amarillo) = \{(Amarillo; 1), (Blanco; 0{,}5)\}$

$similiarBlanco = similarity(Blanco) = \{(Blanco; 1), (Amarillo; 0{,}5), (Celeste; 0{,}2)\}$

$similiarNegro = similarity(Negro) = \{(Negro; 1), (Azul; 0{,}2), (Verde; 0{,}1)\}$

$similiarRojo = similarity(Rojo) = \{(Rojo; 1)\}$

$similiarAzul = similarity(Azul) = \{(Azul; 1), (Celeste; 0{,}7), (Negro; 0{,}2)\}$

$similiarVerde = similarity(Verde) = \{(Verde; 1), (Negro; 0{,}1)\}$

$similiarCeleste = similarity(Celeste) = \{(Celeste; 1), (Azul; 0{,}7), (Blanco; 0{,}2)\}$

Cada conjunto difuso define qué *parecido* es un color con respecto al otro. También definimos la partición difusa:

$ColorSimilar = \{similiarAmarillo, similiarBlanco...similiarCeleste\}$

Definición 16 (Propiedad de cobertura)**.** *Se dice que una partición S cumple con la propiedad de cobertura (coverage) si*

$$\exists i \in 1..s, \forall x \in U, \mu_{A_i}(x) > 0$$

En el ejemplo 5, *Temperatura* cumple la propiedad de cobertura: cada posible valor del dominio pertenece a uno o más de los cuatro conjuntos difusos. El ejemplo 6, *Color* también cumple la propiedad de cobertura, cada color por lo menos pertenece al conjunto similitud del mismo color.

2.6. Operaciones con conjuntos difusos

Las operaciones de la lógica clásica como la unión, intersección, diferencia y el complemento son extendidas en la lógica difusa, Es posible realizar operaciones análogas con los conjuntos difusos, pero a diferencia de la lógica clásica, existe más de una definición para estas operaciones.

Sean A y B son dos subconjuntos tradicionales de U. La unión, denotada $A \cup B$, contiene todos los elementos de A o de B (o de ambos, desde luego). La intersección, denotada $A \cap B$, contiene todos los elementos que están simultáneamente en A y B. El complemento $\bar{A}$ contiene todos los elementos que no están en A. En cambio en la lógica difusa, la unión e intersección de dos conjuntos difusos definen otro conjunto difuso el cual tiene también una función de pertenencia asociada.

La unión e intersección en la lógica difusa están definidas con funciones de *t-norma* (Norma triangular) y *s-norma* (Conorma triangular) respectivamente [Pet96]. La noción de complemento se puede construir de la negación fuerte.

Definición 17 (T-norma)**.** *Una t-norma (o norma triangular) es una función* $T : [0,1]X[0,1] \longrightarrow [0,1]$ *que satisface las siguientes propiedades:*

- *El 1 es el elemento neutro:* $T(x,1) = x$
- *Monótona creciente:* $T(x1,x2) \leq T(x3,x4)$ *cuando* $x1 < x3, x2 < x4$
- *Conmutativa:* $T(x2,x1) = T(x2,x1)$
- *Asociativa:* $T(T(x1,x2),x3) = T(x1,T(x2,x3))$

Las *t-norma* más comunes son el mínimo y el producto. Entonces podemos definir la intersección como:

$$\mu_{A\cap B}(x) = \text{mín}(\mu_A(x), \mu_B(x)) \tag{2.18}$$

$$\mu_{A\cap B}(x) = \mu_A(x)\mu_B(x) \tag{2.19}$$

Definición 18 (S-norma)**.** *Una s-norma (o conorma triangular) es una función* $S : [0,1]X[0,1] \longrightarrow [0,1]$ *que satisface las siguientes propiedades:*

- *El 0 es el elemento neutro:* $S(x,0) = x$
- *Monótona creciente:* $S(x1,x2) \leq S(x3,x4)$ *cuando* $x1 < x2, x3 < x4$
- *Conmutativa:* $S(x2,x1) = S(x2,x1)$
- *Asociativa:* $S(S(x1,x2),x3) = S(x1,S(x2,x3))$

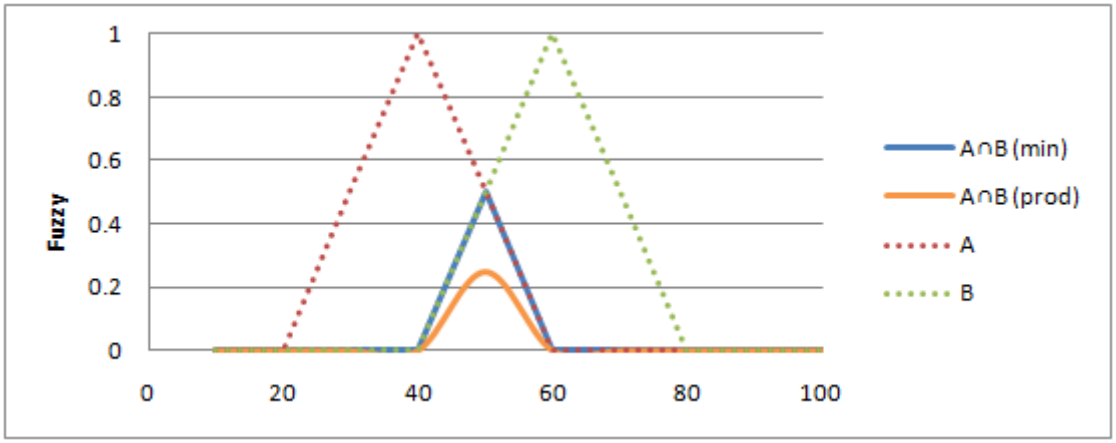

Figura 2.11: Operaciones de *t-norma* entre conjuntos difusos

Las *s-norma* más comunes son el máximo, la suma probabilística y la suma acotada. Entonces podemos definir la unión como:

$$\mu_{A\cup B}(x) = \text{máx}(\mu_A(x), \mu_B(x)) \tag{2.20}$$

$$\mu_{A\cup B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x) \tag{2.21}$$

$$\mu_{A\cup B}(x) = \text{mín}(1, \mu_A(x) + \mu_B(x)) \tag{2.22}$$

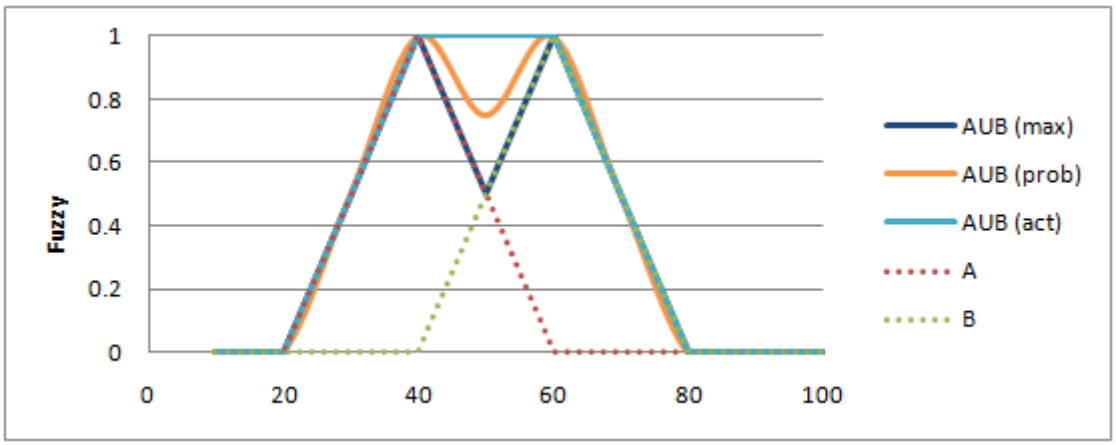

Figura 2.12: Operaciones de *s-norma* entre conjuntos difusos

Hay que destacar que existen dos leyes fundamentales de la teoría clásica de conjuntos que no se cumplen en la teoría de conjuntos difusos: El principio de contradicción $A \cup \bar{A} = U$ y el principio de exclusión $A \cap \bar{A} = \varnothing$

Definición 19 (Negación fuerte)**.** *Una función* $C : [0,1] \longrightarrow [0,1]$ *es una negación fuerte si satisface las siguientes condiciones:*

- $C(0) = 1$
- $C(C(x)) = x$ *(Involución)*
- C *es estrictamente decreciente*
- C *es continua*

Existen varias funciones que son negaciones fuertes, pero la más comúnmente usada es la definida por Zadeh en [Zad65].

$$C(x) = 1 - x; \tag{2.23}$$

Entonces la función de pertenencia sobre el complemento de A

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \tag{2.24}$$

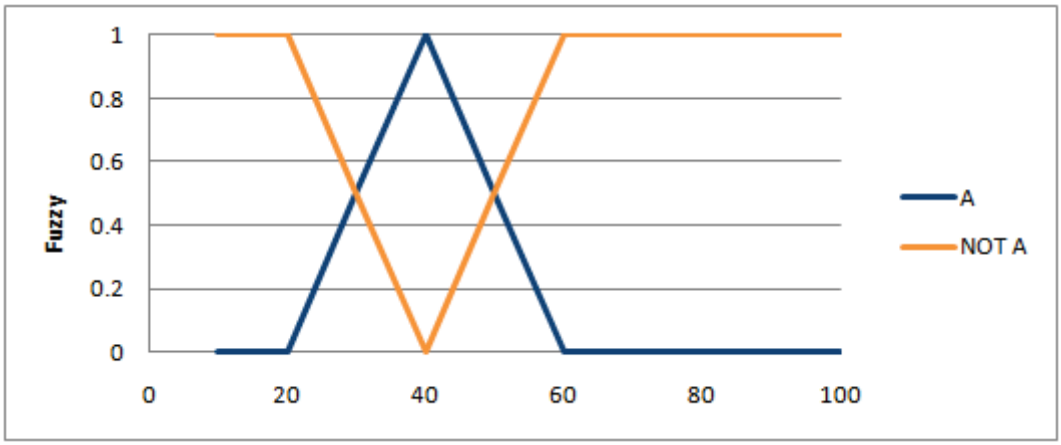

Figura 2.13: Operación de negación entre conjuntos difusos

Ejemplo 7. Si definimos los conjuntos *bajo*, *medio* y *alto* como:

- $\mu_{bajo}(x) = \mu_{NormalLeftTraprapezoid(150,160)}(x)$
- $\mu_{medio}(x) = \mu_{NormalTraprapezoid(155,160,170,175)}(x)$
- $\mu_{alto}(x) = \mu_{NormalRightTraprapezoid(170,180)}(x)$

Entonces podemos decir que una persona de una altura de 172 cm. pertenece, con un cierto grado, a los siguientes conjuntos:

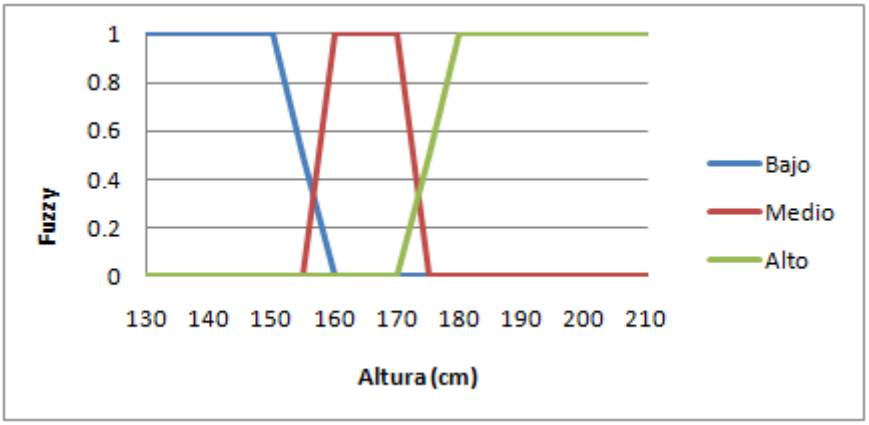

Figura 2.14: Conjuntos difusos bajo, medio y alto

- $\mu_{bajo}(172) = 0$
- $\mu_{\bar{bajo}}(172) = 1$
- $\mu_{medio}(172) = 0{,}66$
- $\mu_{\bar{medio}}(172) = 0{,}33$
- $\mu_{alto}(172) = 0{,}2$
- $\mu_{\bar{alto}}(172) = 0{,}8$
- $\mu_{bajo \cup medio}(172) = 0{,}66$
- $\mu_{bajo \cap medio}(172) = 0$
- $\mu_{bajo \cup alto}(172) = 0{,}2$
- $\mu_{bajo \cap alto}(172) = 0$
- $\mu_{medio \cup alto}(172) = 0{,}66$
- $\mu_{medio \cap alto}(172) = 0{,}2$

Donde la intersección utiliza la función mínimo y la unión utiliza la función máximo.

2.7. Modificadores lingüísticos

Los modificadores se definen en la lógica difusa para poder mantenerse cercanos al lenguaje natural. Tradicionalmente son utilizados como modificadores de los conjuntos difusos los que en el lenguaje natural llamamos adverbios. La interpretación en el modelo difuso de estos enunciados consiste en la composición de la función de pertenencia con una operación aritmética simple.

Un modificador lingüístico altera de cierto modo el significado de un término. Por ejemplo, en la oración "muy cerca de 0", la palabra "muy" modifica "cerca de 0" que podemos caracterizar por medio de un conjunto difuso. Otros ejemplos de modificadores son "un poco", "más o menos", etc.

Definición 20 (Modificadores lingüísticos)**.** *Los modificadores lingüísticos son operadores unarios* $h : [0, 1] \to [0, 1]$ *que permiten extender los conjuntos difusos dando de algún modo más poder expresivo.*

Ejemplo 8. La negación difusa $\mu_{notA}(x) = 1 - \mu_A(x)$ es un ejemplo de modificador

Si bien es difícil decir con precisión el efecto que tiene el modificador "muy", podemos asegurar que en nuestro lenguaje coloquial tiene un efecto intensificador. El modificador "más o menos" tiene un efecto opuesto al modificador "muy". Generalmente se los aproxima con las funciones:

- muy: $\mu_{\text{very } A}(x) = \mu_A(x)^2$
- más o menos: $\mu_{\text{moreo\_or\_less } A}(x) = \sqrt{\mu_A(x)}$

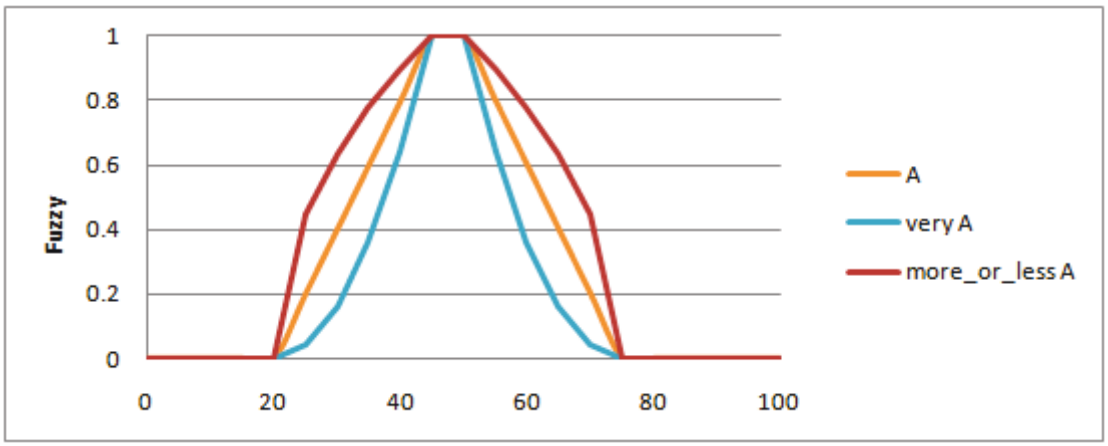

Figura 2.15: Funciones de pertenencia aplicando distintos adverbios al conjunto A

Cualquier función potencia podría ser utilizada. Una familia entera de modificadores es generada por a^p donde p es alguna potencia.

Definición 21 (Modificadores potencia)**.** *Los modificadores potencia son operaciones potencia de la forma* $\mu_{A^n}(x) = \mu_A(x)^n$.

Podemos dividir estos modificadores en dos grupos:

- Concentración: La función es modificada por una función potencia donde $p > 1$. El efecto es que la función de pertenencia toma valores más pequeños centrándose en los valores mayores. El efecto de aplicar la concentración lo podemos ver en la función *very* A de la figura 2.15.
- Dilatación: La función es modificada por una función potencia donde $0 < p < 1$. El efecto es el contrario a la concentración. El efecto de aplicar la dilatación lo podemos ver en la función *moreo_or_less* A de la figura 2.15

Otros modificadores potencia predefinidos son:

- Extremadamente: $\mu_{\text{extremely } A}(x) = \mu_A(x)^4$
- Algo: $\mu_{\text{some } A}(x) = \sqrt[3]{\mu_A(x)}$

Propiedades de los modificadores

Sea $h, g : [0,1] \to [0,1]$ modificadores, se cumplen las siguientes propiedades:

- $h(0) = 0$, si un elemento no pertenece al conjunto tampoco pertenece al conjunto modificado.
- $h(1) = 1$, si un elemento pertenece completamente al conjunto también pertenece completamente al conjunto modificado.
- h es una función continua
- si h es un concentrador h^{-1} es un dilatador y viceversa.
- g compuesto con h es un modificador.

2.8. Traslaciones lingüísticas

Las traslaciones lingüísticas cumplen el mismo objetivo general que los modificadores lingüísticos: ampliar el poder expresivo de las funciones difusas. Trabajan sobre los valores de entrada de las funciones de pertenencia y no alteran la salida de la función.

Definición 22 (Traslación lingüística)**.** *Sea U el dominio en discurso de un conjunto difuso A, una traslación lingüística es un operador unario $t : U \to U$ que modifican la entrada a la función de pertenencia μ_A.*

$$\mu_{tA}(x) = \mu_A(t(x)) \tag{2.25}$$

Ejemplo 9. De acuerdo a B. Bouchon-Meunier y Yao [BBM92], se pueden definir adverbios como traslaciones. Por ejemplo, el adverbio *realmente* se define como como $really(x) = x + 10$, entonces:

$$\mu_{\text{really A}}(x) = \mu_A(x + 10)$$

trasladando la función diez lugares hacia la derecha en el eje x. Generalizamos estas traslaciones cómo $S_\alpha(x) = x + \alpha$.

Ejemplo 10. Antónimos como *chico* - *grande* y *joven* - *viejo* pueden ser usados como traslaciones.

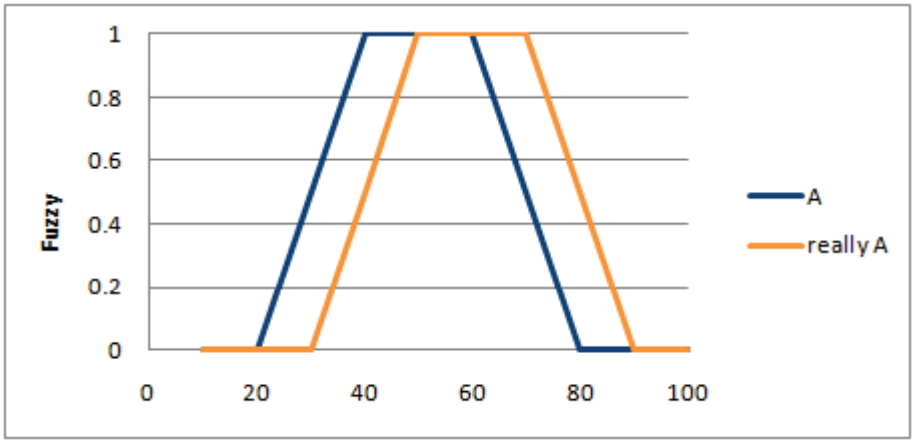

Figura 2.16: Funciones de pertenencia del conjunto A y *really* A.

Supongamos que el antónimo al conjunto difuso A es el conjunto difuso B, entonces

$$\mu_{ant\ A}(x) = \mu_B(x)$$

Ejemplo 11. Si el antónimo del conjunto difuso *bajo* (del ejemplo 7) es *alto* entonces:

$$\mu_{antbajo}(172) = \mu_{alto}(172) = 0{,}2$$

Donde 172 es la altura de la persona.

2.9. Cuantificadores lingüísticos

Se usan para medir (o cuantificar) la cantidad o la proporción de objetos o elementos que cumplen o satisfacen cierta condición.

En lógica clásica existen dos muy importantes:

- $\forall$ (todo): Se refiere a todos los elementos u objetos.
- $\exists$ (existe): Se refiere al menos a uno de los elementos u objetos.

La utilización de los cuantificadores en la lógica clásica está bien definida. Utilizamos $\forall$ (todo) para *preguntar* si todos los elementos del *universo en discurso* pertenecen a un conjunto A. Utilizamos $\exists$ (existe) para *preguntar* si al menos un elemento del *universo en discurso* pertenece al conjunto A. Como ya sabemos, la frontera entre pertenecer y no pertenecer es clara.

Pero en la lógica difusa esta interpretación no es trivial ¿Cómo decimos que todos los elementos pertenecen a un conjunto difuso? Si todos los elementos pertenecen parcialmente al conjunto, ¿Está bien decir que $\forall$ es verdad? Si es verdad,

¿Con qué grado de verdad? Si casi todos los elementos pertenecen totalmente al conjunto, ¿Es posible definir un $\forall$ donde sea parcialmente verdad? Si un elemento pertenece parcialmente, ¿Existe ($\exists$) es verdad o no? ¿Con qué grado de verdad?

En realidad, la lógica difusa no utiliza los cuantificadores clásicos $\forall$ y $\exists$. Zadah [Zad83] propone dos tipos más generales: los *absolutos* y los *relativos*. La forma de definir los cuantificadores es con funciones similares a las funciones de pertenencia. Un cuantificador difuso puede ser verdad con un cierto grado, donde ese grado es un valor entre 0 y 1 inclusive. Al igual que en la lógica clásica, el valor de verdad de un cuantificador depende del universo en discurso. En la sección 2.9.1 se desarrollará cómo decir que los elementos del *universo en discurso* hacen verdad al cuantificador.

Definición 23 (Cuantificador absoluto)**.** *Un cuantificador absoluto es una función*

$$Q_{abs} : \mathbb{R}^+ \to [0, 1] \tag{2.26}$$

Para evaluar la verdad de un cuantificador absoluto necesitamos una única cantidad: los elementos que cumplen la condición.

Ejemplo 12. Como ejemplos cuantificadores absolutos tenemos:

- una docena: $a\_dozen = Like(12, 2)$
- alrededor de cinco: $around\_five = Around(5, 1, 1)$
- al menos tres: $at\_least\_three = NormalRightTraprapezoid(2, 3)$

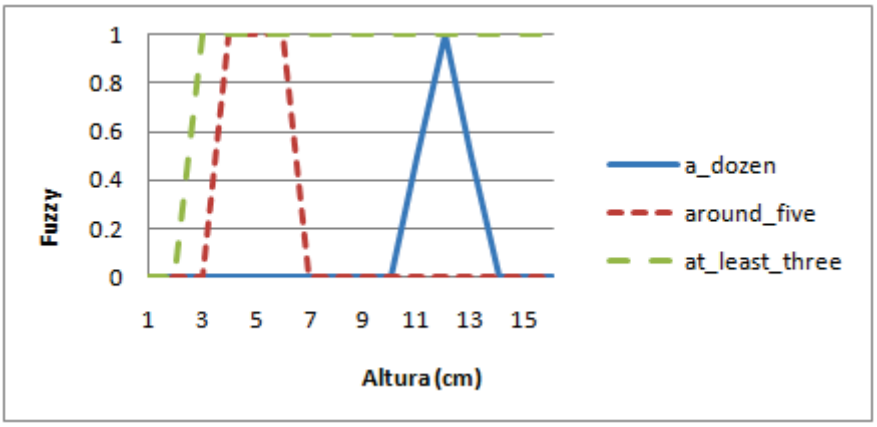

Figura 2.17: Ejemplos de cuantificadores absolutos.

Definición 24 (Cuantificador relativo)**.** *Un cuantificador relativo es una función de la forma*

$$Q_{rel} : [0, 1] \to [0, 1] \tag{2.27}$$

Un cuantificador relativo se aplica a la división del número de elementos que cumplen la condición y el número de elementos totales.

Ejemplo 13. Como ejemplos cuantificadores relativos tenemos:

- la mayoría: $the\_most = NormalRightTraprapezoid(0{,}5; 0{,}6)$
- un tercio: $third = NormalTraprapezoid(0{,}2; 0{,}25; 0{,}35; 0{,}4)$
- casi todos: $almost\_all = NormalRightTraprapezoid(0{,}7; 0{,}9)$

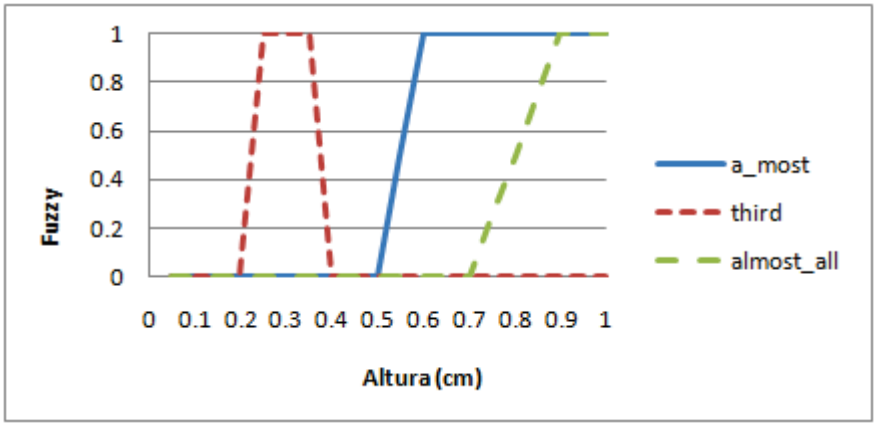

Figura 2.18: Ejemplos de cuantificadores relativos.

2.9.1. Agregación de los elementos en los cuantificadores difusos

Diferentes interpretaciones han sido propuestas para dar valor de verdad a sentencias cuantificadas. En [Bos95] podemos encontrar un resumen de estas. Por simplicidad utilizaremos la propuesta por Zadah en [Zad83]:

Definición 25 (Aproximación computacional de cuantificadores)**.** *El valor de verdad de un cuantificador absoluto Q_a sobre el conjunto A es:*

$$V_{Q_a}(A) = \mu_{Q_a}(\sum_{i=1}^{n} \mu_A(x_i))$$

El valor de verdad de un cuantificador relativo Q_r sobre el conjunto A es:

$$V_{Q_r}(A) = \mu_{Q_r}(1/n * \sum_{i=1}^{n} \mu_A(x_i))$$

Donde A es un conjunto difuso sobre el universo en discurso U, μ_A su función de pertenencia y n la cardinalidad de U. La primera función la llamaremos suma difusa y la segunda función la llamaremos promedio difuso.

Ejemplo 14. Definimos el universo de todas las personas con su respectiva altura como:

$$U = \{(Juan, 182), (Pedro, 176), (Maria, 162), (Jorge, 172), (Luciano, 152), (Clara, 174)\}$$

La cardinalidad n del universo en discurso es 6. Reutilizaremos el conjunto difuso *alto* donde el grado de pertenencia de cada persona es:

- $\mu_{alto}(Juan) = 1$
- $\mu_{alto}(Pedro) = 0{,}6$
- $\mu_{alto}(Maria) = 0$
- $\mu_{alto}(Jorge) = 0{,}2$
- $\mu_{alto}(Luciano) = 0$
- $\mu_{alto}(Clara) = 0{,}4$

Si realizamos la suma difusa y el promedio difuso tenemos que:

$$\sum_{i=1}^{n} \mu_{alto}(x_i) = 2{,}2$$

$$1/n * \sum_{i=1}^{n} \mu_{alto}(x_i) = 0{,}366$$

Con estos resultados podemos calcular el grado de verdad de los cuantificadores definidos en los ejemplos 12 y 13.

$$V_{a\_dozen}(alto) = \mu_{a\_dozen}(2{,}2) = 0$$

$$V_{around\_five}(alto) = \mu_{around\_five}(2{,}2) = 0$$

$$V_{at\_least\_three}(alto) = \mu_{at\_least\_three}(2{,}2) = 0{,}2$$

$$V_{the\_most}(alto) = \mu_{the\_most}(0{,}366) = 0$$

$$V_{third}(alto) = \mu_{third}(0{,}366) = 0{,}66$$

$$V_{almost\_all}(alto) = \mu_{almost\_all}(0{,}366) = 0$$

Los únicos cuantificadores con un grado de verdad son los cuantificadores *al menos tres* y *un tercio*.

Capítulo 3

Consultas difusas y SQLf

3.1. Introducción

La utilización de información imprecisa (o vaga) en sistemas de computadoras ha sido tema de estudio de muchos investigadores durante varios años. La idea principal es relajar los modelos relacionales para permitir imprecisión y la utilización de terminología similar al lenguaje natural. La información vaga en los sistemas puede encontrarse en dos niveles: El primer nivel considera la posibilidad de realizar consultas imprecisas a base de datos clásicas. El segundo nivel involucra almacenar información imprecisa directamente en el sistema.

El primer nivel nos permitirá realizar consultas de la forma *dame los trabajadores con salario alto*. La ventaja de este nivel es que permite trabajar sobre base de datos clásicas de manera más directa. En este ejemplo, la base de datos deberá conocer a priori los trabajadores y el valor numérico de los salarios. Estas consultas podrán ser ejecutadas sobre un modelo relacional clásico.

Los sistemas de segundo nivel, por ejemplo, permitirán almacenar empleados donde su salario es desconocido, su valor podría ser un valor inconcreto-subjetivo como *salario alto* o su veracidad no es absoluta. Para este tipo de sistemas, el modelo relacional clásico y por lo tanto las base de datos relacionales no son suficientes.

En cualquiera de los dos niveles, la lógica difusa es una buena herramienta para representar la naturaleza imprecisa de la información. Varios investigadores han utilizado la lógica difusa y propuesto soluciones a estos dos niveles. La mayoría de las propuestas intentan reutilizar el modelo relacional y por ende los motores de base de datos relaciones existentes. De la misma manera, extienden el lenguaje SQL para poder expresas la características imprecisas de los datos y las consultas. Entre las dos soluciones mas importantes están SQLf y FSQL. En este capítulo repasaremos brevemente el modelo relacional y el lenguaje SQL

para luego poder mostrar en detalle SQLf. En el siguiente capítulo evaluaremos FSQL mostrando sus diferencias con SQLf.

3.1.1. El modelo relacional

El modelo relacional es el modelo más utilizado en la actualidad. Este modelo nos permite almacenar y relacionar la información en un sistema de computadoras. El modelo relacional fue introducido por Codd ([Cod70]) y almacena la información en tuplas o registros agrupados en relaciones o tablas. Cada registro está conformado por columnas o campos. Este modelo es independiente de la implementación y permite a los SGBDs (Sistema Gestor de Base de Datos) realizar sus propias optimizaciones. Es el utilizado por los SGBDs por su gran versatilidad, potencia y formalismos matemáticos en los que se basa. El modelo relacional utiliza dos lenguajes formales de consultas, el álgebra relacional y el cálculo relacional y solo permite almacenar datos precisos o *crisp*.

3.1.2. El lenguaje SQL

El lenguaje de consulta estructurado o SQL (por sus siglas en inglés Structured Query Language) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en éstas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo efectuar consultas con el fin de recuperar *de una forma sencilla* información de interés de una base de datos, así como también hacer cambios sobre ella.

SQL se basa el sub-lenguage de consultas que Edgar Frank Codd propone en 1970 para manipular su modelo relacional. En 1977 IBM define SEQUEL que es utilizado en su SGBD experimental System R. En 1979 Oracle presenta el primer SGBD comercial que incluye SQL, la evolución de SEQUEL. En 1986 ANSI estandariza el lenguaje en su primera versión y es también adoptado en la mayoría de los SGBDR.

SQL está dividido en dos categorías principales: el Lenguaje de Definición de Datos (LDD) y el Lenguaje de Manipulación de Datos (LMD).

- **LDD - DDL** (Lenguaje de Definición de Datos, Data Definition Language): Se utiliza para la creación, modificación y borrado de las estructuras y los objetos de una base de dato relacional. Existen tres operaciones básicas: CREATE para el creado de objetos, ALTER para la modificación de objetos y DROP para el borrado de los objetos. En el contexto de las bases de datos relacionales, los objetos pueden ser tablas, vistas, índices, etc.

- **LMD - DML** (Lenguaje de Manipulación de Datos, Data Manipulation Language): Se utiliza para la inserción, alteración, borrado y consulta de

datos. Sus operaciones básicas son: INSERTE, UPDATE, DELETE y SELECT.

3.1.3. La sentencia SELECT

La operación más común en SQL es la consulta que se realiza utilizando la sentencia SELECT. La consulta se define de una manera declarativa y da cómo resultado un conjunto de tuplas. Los detalles de su ejecución son abstraídos y delegados a la implementación de SGBDR.

Una consulta está formada por la palabra SELECT seguida por la lista de columnas que van a formar parte del resultado. El asterisco (*) puede ser usado para incluir todas las columnas de la tabla. La declaración de las columnas es equivalente a la *proyección* del álgebra relacional.

La operación también puede incluir las siguientes palabras claves y cláusulas:

- **FROM**: Esta cláusula indica las tablas de donde los datos son obtenidos. Es equivalente al *Producto Cartesiano* o *Reunión* del álgebra relacional. La palabra clave JOIN se puede utilizar para especificar las reglas de la reunión.
- **WHERE**: Especifica la filas que tendrá el resultado. Utiliza un predicado booleano. Si la evaluación de la fila da verdadero, la fila será parte de los resultados. Es equivalente a la *selección* del álgebra relacional.
- **GROUP BY**: Agrupa filas que comparten una propiedades formando grupos. Funciones de agregación pueden ser aplicadas a cada grupo.
- **HAVING**: Selecciona los grupos usando un predicado booleano similar a la utilizada en la cláusula WHERE. Funciones de agregación pueden ser utilizadas en la condición.

Sin lugar a duda SQL, es el lenguaje estándar utilizado para manipular bases de datos. Existe una cantidad importante de bibliografía y los detalles del lenguaje escapan al alcance de esta tesis. Igualmente, cada una de sus operaciones será *repensada* en el marco de las consultas difusas.

3.2. SQLf

El lenguaje SQLf [Bos95] es un lenguaje estructurado de consultas difusas que trabaja sobre base de datos relacionales normales. Es una extensión de SQL que permite realizar consultas donde los resultados obtenidos son presentados con un grado de precisión. El usuario puede definir términos lingüísticos y utilizarlos en consultas utilizando operadores de la lógica difusa. El resultado de una consulta SQLf es un conjunto difuso, cada tupla del resultado representa un elemento del

conjunto y tiene asociado un grado pertenencia. Este grado es el mismo explicado en el capítulo 1. Si el grado es 0, la tupla estará completamente excluida de los resultados; si el grado es 1 la tupla cumplirá completamente con la consulta. SQLf extiende las expresiones booleanas clásicas de SQL permitiendo expresiones de lógica difusa. El objetivo ahora no solo consiste en encontrar los elementos que satisfacen la consulta, sino también en saber el grado en que lo satisface; es decir el grado de pertenencia. SQLf es una solución a los sistemas difusos de primer nivel; no permite almacenar datos imprecisos pero sí permite realizar consultas imprecisas.

La ventaja principal de SQLf es su adaptabilidad. Es posible reutilizar las bases de datos relacionales existentes junto con la base de conocimiento para realizar consultas imprecisas sobre los datos. Su sencillez es también su debilidad, no permite almacenar datos imprecisos ya que no es una base de datos difusa.

Para poder explicar SQLf y su funcionamiento, he programado una pequeña aplicación en Java que define todos los casos de pruebas que veremos. Esta aplicación utiliza una implementación de SQLF llamada SQLf_j ([MSKD]) la cual he revisado y adaptado simplificándola y agregándole nueva funcionalidad. En el capítulo 5 explicaré los detalles de esta librería y los cambios realizados. En este capítulo me concentraré más en lo que SQLf nos permite realizar. Los resultados que veremos en las consultas (clásicas y difusas) son resultados obtenidos luego de ejecutar las pruebas. También he basado los ejemplos en SQLfi ([GT08], [GT07]), otra implementación de SQLf que permite sentencias que no están en SQLf_j. A continuación desarrollaré las consultas SQLf mostrando una cantidad de ejemplos donde los elementos de la lógica difusa son fusionados con SQL. Me basaré en la literatura y en los resultados obtenidos de los casos de prueba de la aplicación.

3.3. La sentencia SELECT de SQLf

Las consultas SQLf, al ser una extensión, son muy similares a las consultas SQL. Estas consultas extienden la sentencia SELECT y sus palabras claves.

Una consulta SQLf tiene la forma:

```
SELECT [TOP N] [THRESHOLD T] <atributos>
  FROM <relaciones> WHERE <condicion_fuzzy>
```

Donde N son las n mejores tuplas que cumplan con el umbral (`threshold`) mayor o igual que `T`. `<relaciones>` es una lista de relaciones usuales (`crisp`) y `<condicion_fuzzy>` son condiciones tanto de lógica booleana como difusa unidas por conectores.

El resultado de la ejecución no solo retorna las columnas indicadas en la proyección sino también una columna extra con el umbral de pertenencia o *fuzzy degree* entre $[T, 1]$.

En lenguaje natural una consulta SQLf se expresaría como: dame los primeros N elementos de <relaciones> con un grado de pertenencia mayor a T que cumplen la condición difusa <condicion_fuzzy>.

Para poder mostrar ejemplos de consultas difusas definiré un pequeño sistema hipotético que debe ser extendido para poder satisfacer consultas imprecisas.

3.4. Ejemplo: Estación de policía

Existe en producción un sistema que almacena un registro de todos los criminales conocidos. De cada criminal se conoce su nombre, fecha de nacimiento, características físicas y las zonas por donde actúan. Cuando un crimen ocurre, un oficial de policial toma declaración a un testigo y trata de encontrar los posibles sospechosos dentro del sistema. El objetivo de este sistema es poder encontrar los criminales eficientemente; es decir, dar una lista de posibles sospechosos

El sistema actual tiene un motor de búsquedas tradicionales utilizando SQL sobre la base de datos relacional. Este motor puede retornar todas las personas que cumple con cierta características físicas, que están entre un rango de edad,que atacan en ciertas zonas, etc.

El problema con el sistema actual es que la entrada de las consultas es tomada de una fuente no 100 % confiable; los datos que un testigo provee son datos imprecisos. Por ejemplo, después de un crimen durante la noche, un testigo del hecho afirmó que el sospechoso tiene cabello negro. Podemos ver que este dato no es siempre confiable, tal vez el criminal tenía el pelo castaño oscuro y la oscuridad de la noche confundió al testigo. En el motor de búsqueda actual, el sistema simplemente ignoraría los criminales con cabello castaño oscuro.

Del mismo modo, el testigo podría no dar información concreta y utilizaría palabras del lenguaje natural. Por ejemplo, el criminal era delgado y alto. En términos del sistema actual, delgado y alto no tiene ninguna representación; el sistema entiende el peso en kilogramos y la altura en centímetros.

3.4.1. Modelo de datos

A continuación definiré el modelo de datos del ejemplo. Este modelo sencillo, conformado por una tabla, nos servirá como base para definir ejemplos de consultas tanto SQL como SQLf.

```
Criminal={id:long,
        nombre:string,
        sexo:boolean,
        fechaNacimiento:date,
        altura:int,
        peso:float,
```

```
        colorPelo:string,
        colorPiel:string,
        zona:string
}
```

3.4.2. Ejemplo de datos

La siguiente tabla muestra un ejemplo de datos posibles del sistema. Estos datos serán útiles para poder mostrar resultados luego de *ejecutar* los ejemplos.

id	nombre	sexo	fechaNacimiento	altura	peso	colorPelo	colorPiel	zona
1	Juan	M	22/07/1985	182	90	Castaño	Caucásico	La Plata
2	María	F	09/10/1987	152	65	Negro	Negro	Quilmes
3	Pedro	M	21/02/1975	169	75	Gris	Negro	La Plata
4	Julio	M	03/02/1990	172	120	Albino	Amarilla	Beriso
5	Paula	F	04/01/1973	162	62	Pelirrojo	Caucásico	Caballito
6	Rodrigo	N	30/01/1965	190	85	Negro	Amarilla	Recoleta
7	Natalia	F	21/08/1944	154	72	Rubio	Caucásico	La Plata
8	Fabián	M	15/07/1955	192	100	Castaño	Hispana	Berisso
9	Daniela	F	12/08/1962	157	50	Castaño	Caucásico	La Plata
10	Ana	F	12/08/1970	170	67	Castaño	Caucásico	La Plata
11	Celeste	F	10/10/1970	178	90	Negro	Hispana	Quilmes

3.4.3. Conjuntos difusos y etiquetas lingüísticas

En este sistema es posible definir fácilmente la consulta *Personas con altura mayor a 175*. Esta consulta se traduciría a SQL como:

Consulta 3.1: Criminales con altura mayor a 175 cm.

```
SELECT * FROM Criminal WHERE altura > 175
```

El resultado son las personas que pertenecen al conjunto tradicional *altura mayor a 175cm*.

La consulta anterior es muy restrictiva para nuestro sistema. El testigo posiblemente sea incapaz de dar una altura concreta del delincuente o tal vez expresar la altura en términos del lenguaje natural: *El delincuente era alto* o *La altura del delincuente era alrededor de 175cm*.

Si queremos realizar este tipo de consultas primero tenemos qué definir que significa que una persona sea *alta* y qué significa que la altura sea *alrededor de 175*. Para ello utilizaremos los conjuntos difusos: una persona es *alta* si pertenece al conjunto difuso *alto* (con su grado de pertenencia). Del mismo modo debemos definir el conjunto difuso *alrededor_de_175*

A continuación describiré conceptualmente los elementos de la lógica difusa que el sistema utilizará durante las consultas SQLf. Estos elementos son los conjuntos difusos, relaciones de similitud, etiquetas lingüísticas, modificadores, traslaciones y cuantificadores difusos.

3.4.4. Etiqueta lingüística Altura

Sobre el dominio de los enteros positivos que representan las alturas de las personas en centímetros, podemos definir la variable lingüística *Altura*:

$$Altura = \{bajo, medio, alto\}$$

Donde las funciones de pertenencia de conjuntos difusos *bajo*, *medio* y *alto* están definidas como:

- $\mu_{bajo}(x) = \mu_{NormalLeftTrapezoid(150,160)}(x)$
- $\mu_{medio}(x) = \mu_{NormalTrapezoid(155,160,170,175)}(x)$
- $\mu_{alto}(x) = \mu_{NormalRightTrapezoid(170,180)}(x)$

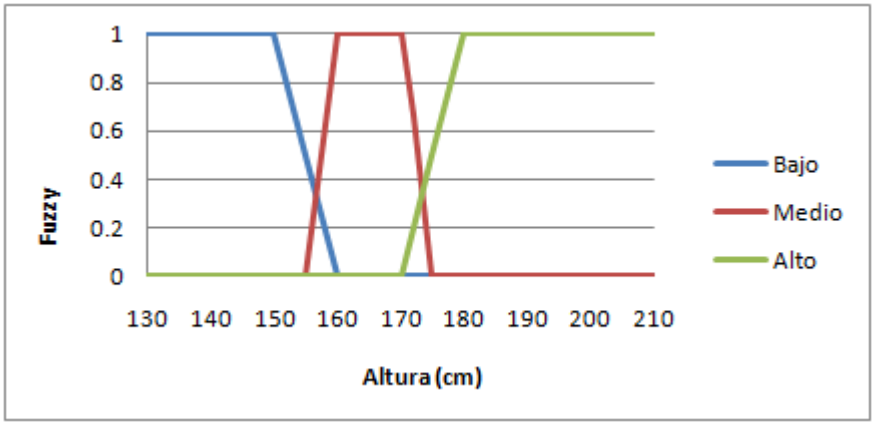

Figura 3.1: Etiqueta *Altura* y sus conjunto difusos

3.4.5. Etiqueta lingüística Contextura Física

Sobre el dominio de los reales positivos que representan los pesos de las personas en kilogramos, podemos definir la etiqueta lingüística *Contextura Física*.

$$ContexturaFisica = \{flaco, normal, gordo, obeso\}$$

Donde los conjuntos difusos *flaco*, *normal*, *gordo* y *obeso* están definidos como:

- $\mu_{flaco}(x) = \mu_{NormalLeftTrapezoid(50,60)}(x)$
- $\mu_{normal}(x) = \mu_{Gaussian(80,20)}(x)$
- $\mu_{gordo}(x) = \mu_{Gaussian(100,20)}(x)$
- $\mu_{obeso}(x) = \mu_{NormalRightTrapezoid(120,140)}(x)$

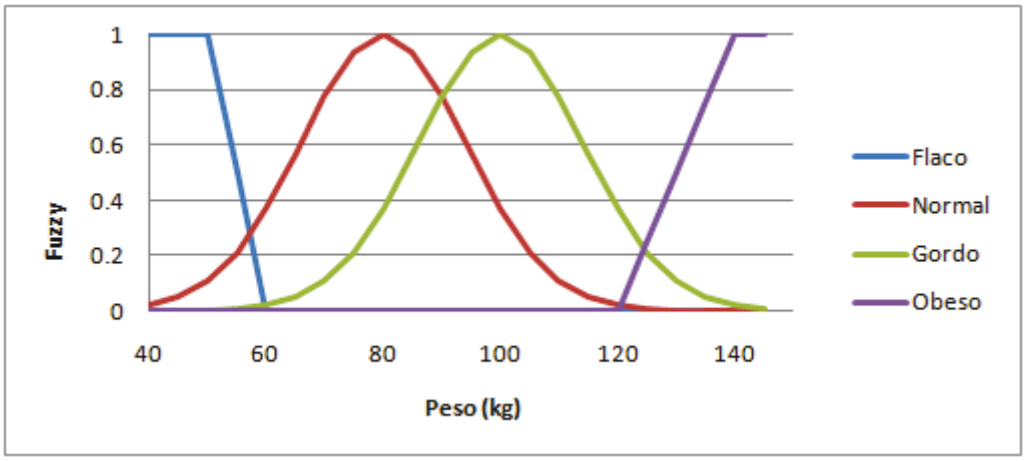

Figura 3.2: Etiqueta *Contextura Física* y sus conjunto difusos

3.4.6. Etiquetas lingüísticas Similitud de color pelo

Un dominio discreto que podemos definir en este sistema es $ColorPelo$. Como hemos visto en la sección 2.4.1, de cada elemento del dominio podemos definir un conjunto difuso de similitud. El dominio $ColorPelo$ está definido como:

$$ColorPelo = \{Negro, Rubio, Castaño, Gris, Pelirrojo, Albino\}$$

$ColorPelo$	Negro	Rubio	Castaño	Gris	Pelirrojo	Albino
$\mu_{similarity(Negro)}$	1.0	0.1	0.7	0.7	0.5	0.0
$\mu_{similarity(Rubio)}$	0.1	1.0	0.2	0.5	0.2	0.7
$\mu_{similarity(Castaño)}$	0.7	0.2	1.0	0.6	0.5	0.0
$\mu_{similarity(Gris)}$	0.7	0.5	0.6	1.0	0.5	0.6
$\mu_{similarity(Pelirrojo)}$	0.5	0.2	0.5	0.5	1.0	0.2
$\mu_{similarity(Albino)}$	0.0	0.7	0.0	0.6	0.2	1.0

La variable lingüística *Similitud de color pelo* está definida como:

$$Similarty(ColorPelo) = \{similarity(Negro), similarity(Rubio), similarity(Castaño),$$
$$similarity(Gris), similarity(Pelirrojo), similarity(Albino)\}$$

3.4.7. Etiquetas lingüística Similitud de color piel

Otro dominio continuo que podemos definir es $ColorPiel$. De la misma manera, este dominio nos permite crear conjuntos difusos de similitud:

$$ColorPiel = \{Negra, Caucásico, Amarilla, Hispana\}$$

$ColorPiel$	Negra	Caucásico	Amarilla	Hispana
$\mu_{similarity(Negra)}$	1.0	0.1	0.0	0.3
$\mu_{similarity(Caucásico)}$	0.1	1.0	0.3	0.3
$\mu_{similarity(Amarilla)}$	0.0	0.3	1.0	0.2
$\mu_{similarity(Hispana)}$	0.3	0.3	0.2	1.0

La variable lingüística *Similitud de color piel* está definida como:

$$Similarty(ColorPiel) = \{similarity(Negra), similarity(Caucásico),$$

$$similarity(Amarilla), similarity(Hispana)\}$$

Modificadores y Traslaciones

Como modificadores vamos a utilizar los más clásicos e independientes del dominio de la aplicación.

- Muy: $\mu_{\text{muy } A}(x) = \mu_A(x)^2$
- Más o menos: $\mu_{\text{mas\_o\_menos } A}(x) = \sqrt{\mu_A(x)}$
- Extremadamente: $\mu_{\text{extremadamente } A}(x) = \mu_A(x)^4$

Con estos modificadores extenderé el lenguaje de consulta; se podrá consultar por las personas *extremadamente flacas*. Como traslaciones utilizaremos las siguientes:

- Realmente: $\mu_{\text{realmente } A}(x) = \mu_A(x + 10)$
- Más que: $\mu_{\text{mas\_que } A}(x) = \mu_A(x + 20)$
- Menos que: $\mu_{\text{menos\_que } A}(x) = \mu_A(x - 20)$

Esto nos permitirá realizar consultas como: personas *más que flacas*

3.4.8. Cuantificadores

Para estas consultas, reutilizaré los cuantificadores relativos que definidos en la sección 13:

- la mayoría: $la\_mayoria = NormalRightTraprapezoid(0{,}5; 0{,}6)$
- un tercio: $un\_tercio = NormalTraprapezoid(0{,}2; 0{,}25; 0{,}35; 0{,}4)$
- casi todos: $casi\_todos = NormalRightTraprapezoid(0{,}7; 0{,}9)$
- la mitad: $mitad = NormalTraprapezoid(0{,}4; 0{,}45; 0{,}55; 0{,}6)$

También reutilizaré los cuantificadores absolutos definidos previamente en 12:

- una docena: $una\_docena = Like(12, 2)$
- alrededor de cinco: $alrededor\_de\_cinco = Around(5, 1, 1)$
- al menos tres: $al\_smenos\_tres = NormalRightTraprapezoid(2, 3)$
- al menos diez: $al\_menos\_diez = NormalRightTraprapezoid(8, 10)$

Estos cuantificadores permitirán realizar consultas cuantificadas del tipo I y II ([Bos95]) que serán abarcadas más adelante en este capítulo.

3.5. Consultas difusas sobre Estación de Policía

En esta sección mostraré la forma de realizar diferentes consultas difusas sobre una base de datos. Me basaré en el sistema Estación de Policía, su modelo de datos y los conjuntos difusos definidos en las secciones anteriores. El objetivo de cada consulta es mostrar un tipo de consulta difusa y como se expresa en SQLf. Estas consultas serán ejecutadas en el motor de búsqueda SQLf utilizando los datos de ejemplo definidos anteriormente. Cada consulta no solo retornará los datos solicitados sino también la columna `fuzzy_degree` que es el grado de pertenencia de la fila al resultado.

3.5.1. Consulta crisp

La primera consulta que mostraré no es una consulta difusa, es una consulta SQL estándar utilizando el motor de búsqueda SQLf.

Consulta 3.2: Id nombre y altura de todas los criminales con altura mayor que 175 cm

```
SELECT id, nombre, altura FROM criminal WHERE altura > 175
```

id	nombre	altura	fuzzy_degree
1	Juan	182	1
6	Rodrigo	190	1
8	Fabián	192	1
11	Celeste	178	1

Se puede observar que el resultado obtenido es el esperado de la consulta SQL. La diferencia es que esta consulta retorna la nueva columna `fuzzy_degree`. Debido a que es una consulta clásica, el valor de pertenencia de cada elemento es uno; es decir, el elemento pertenece totalmente al resultado o no pertenece.

3.5.2. Consulta utilizando predicados difusos sobre dominios continuos

Finalmente, luego de haber definido nuestro modelo de datos y los elementos de la lógica difusa que vamos a utilizar en el sistema, estamos en condiciones de expresar la primera consulta difusa en SQLf.

Consulta 3.3: Id nombre y altura de todos los criminales altos

```
SELECT id, nombre, altura FROM criminal WHERE ~altura IS 'alto'
```

id	nombre	altura	fuzzy_degree
1	Juan	182	1.0
6	Rodrigo	190	1.0
8	Fabián	192	1.0
11	Celeste	178	0.8
4	Julio	172	0.2

Esta consulta utiliza el predicado difuso `~altura IS 'alto'` el cual se basa en el conjunto difuso *alto* definido en 3.4.4. La función de pertenencia del conjunto es una función trapezoide normalizado a derecha. Las personas con altura menor a 170 no pertenecen al resultado. Las personas entre 170 y 180 pertenecen parcialmente como por ejemplo *Julio* con un grado de pertenencia de 0.2. Los criminales con altura mayor a 180 pertenecen completamente al resultado y tienen un grado de pertenencia de 1.

Las condiciones difusas sobre dominios continuos se definen utilizando el carácter tilde (`~`) seguido por el nombre de la columna (`altura`), el comparador difuso (`IS`) y un conjunto difuso continuo (`alto`) relacionado con la columna.

3.5.3. La palabra clave TOP

La palabra clave `TOP` nos permite especificar la cantidad máxima de resultados que la consulta retornará priorizando aquellos que tienen un grado de pertenencia

más alta. Por ejemplo, podemos limitar la cantidad de resultados de la consulta 3.3 anterior a solo dos resultados en la consulta 3.4.

Consulta 3.4: Id nombre y altura de los primeros dos criminales altos

```
SELECT TOP 2 id,nombre,altura FROM criminal WHERE ~altura IS
    'alto'
```

id	nombre	altura	fuzzy_degree
1	Juan	182	1.0
6	Rodrigo	190	1.0

Se puede notar que algunos elementos con mismo nivel de pertenencia fueron ignorados debido a que están posicionados después del máximo resultado.

3.5.4. La palabra clave THRESHOLD

La palabra clave `THRESHOLD` permite limitar los resultados a solo aquellos cuyo grado de pertenencia es mayor o igual a un umbral especificado. Si no se define el `THRESHOLD` de la consulta, el resultado incluirá todos los elementos con un grado de pertenencia mayor que cero. Por ejemplo, podemos limitar la cantidad de resultados de la consulta 3.5 a todos los elementos con un grado de pertenencia mayor o igual que 0,5 en la consulta 3.6.

Consulta 3.5: Id nombre y altura de todos los criminales bajos

```
SELECT id,nombre,altura FROM criminal WHERE ~altura IS 'bajo'
```

id	nombre	altura	fuzzy_degree
2	María	152	0.8
7	Natalia	154	0.6
9	Daniela	157	0.3

Consulta 3.6: Id nombre y altura de los criminales bajos con un grado de pertenencia mayor que 0.5

```
SELECT THRESHOLD 0.5 id,nombre,altura FROM criminal WHERE ~
    altura IS 'bajo'
```

id	nombre	altura	fuzzy_degree
2	María	152	0.8
7	Natalia	154	0.6

Se puede observar que *Daniela* pertenece al resultado de la consulta 3.5 pero no al resultado de la consulta 3.6 debido a que su grado de pertenencia 0.3 es menor que el umbral 0.5.

Los predicados difusos de estas consultas utilizan el conjunto difuso *bajo* definido en 3.4.4.

3.5.5. Orden de los resultados

Por defecto, los resultados de una consulta SQLf están ordenados de mayor a menor pertenencia; es decir, en forma descendente por la columna `fuzzy_degree`. El orden de los elementos con el mismo grado de pertenencia es indeterminado.

Es posible utilizar las palabras claves `ORDER BY`, `ASC` y `DESC` para ordenar los resultados de la misma manera que en las consultas SQL. La consulta 3.8 es la consulta 3.7 ordenada por nombre.

Consulta 3.7: Id nombre y peso de las criminales mujeres bajas

```
SELECT id,nombre,peso,sexo FROM criminal WHERE ~peso IS '
    flaco' and sexo = 'F'
```

id	nombre	peso	sexo	fuzzy_degree
9	Daniela	50	F	1.0
5	Paula	62	F	0.4
2	María	65	F	0.25
10	Ana	67	F	0.15

Consulta 3.8: Id nombre y peso de las criminales mujeres bajas ordenadas lexicogr?ficamente por nombre en forma descendente

```
SELECT id,nombre,peso,sexo FROM criminal WHERE ~peso IS '
    flaco' and sexo = 'F' ORDER BY nombre DESC
```

id	nombre	peso	sexo	fuzzy_degree
5	Paula	62	F	0.4
2	María	65	F	0.25
9	Daniela	50	F	1.0
10	Ana	67	F	0.15

SQLf permite ordenar utilizando la columna `fuzzy_degree`. La consulta 3.9 ordena los resultados de menor a mayor pertenencia.

Consulta 3.9: Id nombre y peso de las criminales mujeres bajas ordenadas por grado de pertenencia en forma

```
SELECT id,nombre,peso,sexo FROM criminal WHERE ~peso IS '
    flaco' and sexo = 'F' ORDER BY fuzzy_degree
```

id	nombre	peso	sexo	fuzzy_degree
10	Ana	67	F	0.15
2	María	65	F	0.25
5	Paula	62	F	0.4
9	Daniela	50	F	1.0

Estas consultas son las primeras que utilizan el conector `and` para unir dos predicados, `~peso IS 'flaco'` y `sexo = 'F'`. El primer predicado es un predicado de la lógica difusa explicado previamente y el segundo es un predicado de la lógica clásica. Cómo son *conectados* y evaluados los predicados en este motor SQLf será explicado en la sección 3.5.8.

3.5.6. Consultas utilizando operadores difusos sobre dominios discretos

De manera similar, en SQLf podemos realizar consultas utilizando conjuntos difusos definidos sobre conjuntos discretos. En las secciones 3.4.6 y 3.4.7 definimos los conjuntos discretos $ColorPelo$ y $ColorPiel$. Para cada elemento de esos conjuntos definimos un conjunto difuso de similitud que expresa qué tan *parecido* es ese elemento a los otros del mismo conjunto.

Podemos utilizar estos conjuntos de similitud en SQLf:

Consulta 3.10: Id nombre y color pelo de los criminales con color de pelo similar a negro

```
SELECT id,nombre,colorPelo FROM criminal WHERE #colorPelo IS
    'Negro'
```

id	nombre	colorpelo	fuzzy_degree
11	Celeste	Negro	1.0
2	María	Negro	1.0
6	Rodrigo	Negro	1.0
10	Ana	Castaño	0.7
9	Daniela	Castaño	0.7
8	Fabián	Castaño	0.7
1	Juan	Castaño	0.7
3	Pedro	Gris	0.7
5	Paula	Pelirrojo	0.5
7	Natalia	Rubio	0.1

La consulta 3.10 utiliza el conjunto difuso $similarity(negro)$. Ana, de cabello castaño, es parte de los resultados con un grado de pertenencia de $\mu_{similarity(negro)}(castaño) =$ 0,7.

Las condiciones difusas sobre dominios discretos se definen utilizando el carácter numeral (#) seguido por el nombre de la columna (`altura`), el comparador difuso (`IS`) y un conjunto difuso de similitud (`Negro`) relacionado con la columna.

La consulta 3.11 es similar a la consulta 3.10 pero utiliza un predicado clásico. Esta consulta clásica solo devuelve los criminales de cabello negro.

Consulta 3.11: Id nombre y color pelo de los criminales con color de pelo negro

```
SELECT id,nombre,colorPelo FROM criminal WHERE colorPelo = '
    Negro'
```

id	nombre	colorpelo	fuzzy_degree
2	María	Negro	1
6	Rodrigo	Negro	1
11	Celeste	Negro	1

3.5.7. Negación de predicados

En SQLf se pueden negar tanto predicados difusos como clásicos; es posible consultar por los elementos que *NO* cumplen con una condición. El predicado difuso de la consulta 3.12 es el predicado negado de la consulta 3.13

Consulta 3.12: Id nombre y color piel de los criminales con color de piel similar a Caucásico

```
SELECT id,nombre,colorPiel FROM criminal WHERE #colorPiel IS
    'Caucásico'
```

id	nombre	colorpiel	fuzzy_degree
1	Juan	Caucásico	1.0
5	Paula	Caucásico	1.0
7	Natalia	Caucásico	1.0
9	Daniela	Caucásico	1.0
10	Ana	Caucásico	1.0
4	Julio	Amarilla	0.3
6	Rodrigo	Amarilla	0.3
8	Fabián	Hispana	0.3
11	Celeste	Hispana	0.3

Consulta 3.13: Id nombre y color piel de los criminales con color de piel distinto a Caucásico

```
SELECT id,nombre,colorPiel FROM criminal WHERE NOT #colorPiel
    IS 'Caucásico'
```

id	nombre	colorpiel	fuzzy_degree
2	María	Negro	1.0
3	Pedro	Negro	1.0
4	Julio	Amarilla	0.7
6	Rodrigo	Amarilla	0.7
8	Fabián	Hispana	0.7
11	Celeste	Hispana	0.7

Podemos observar que Juan pertenece completamente al resultado de la consulta 3.12 pero no al de la consulta 3.13. Julio pertenece parcialmente a ambos resultados.

La negación de predicados en SQLf es equivalente a la negación de conjuntos difusos. Esta operación es una de las operaciones discutidas en la sección 2.6.

El grado de pertenencia de los criminales que cumplen la condición *NOT #colorPiel IS Caucásico* es el grado de pertenencia de los criminales en el conjunto *Caucásico*.

El motor SQLf utiliza la función $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$ (2.23) para calcular los valores de pertenencia. El grado de pertenencia de una persona en un resultado es 1 menos el grado de pertenencia de la misma persona en el otro resultado. Recordemos que las personas con grado de pertenencia 0 son ignoradas. A diferencia de la lógica clásica, es posible que un elemento pertenezca tanto a un conjunto difuso como a su opuesto.

3.5.8. Composición de predicados

De manera similar a SQL, SQLf permite componer predicados utilizando los operandos `and` (conjunción) y `or` (disyunción). La diferencia principal es que SQLf utiliza la lógica difusa en vez de la clásica como SQL.

Las consultas 3.14 y 3.15 son consultas con predicados difusos simples los cuales conectaremos.

Consulta 3.14: Id nombre y altura de los criminales bajos

```
SELECT id,nombre,altura FROM criminal WHERE ~altura IS 'bajo'
```

id	nombre	altura	fuzzy_degree
2	María	152	0.8
7	Natalia	154	0.6
9	Daniela	157	0.3

Consulta 3.15: Id nombre y color pelo de los criminales con color de pelo gris

```
SELECT id,nombre,colorPelo FROM criminal WHERE #colorPelo IS
    'Gris'
```

id	nombre	colorpelo	fuzzy_degree
3	Pedro	Gris	1.0
11	Celeste	Negro	0.7
6	Rodrigo	Negro	0.7
2	María	Negro	0.7
4	Julio	Albino	0.6
8	Fabián	Castaño	0.6
9	Daniela	Castaño	0.6
10	Ana	Castaño	0.6
1	Juan	Castaño	0.6
5	Paula	Pelirrojo	0.5
7	Natalia	Rubio	0.5

Conjunción de predicados

Es posible consultar por los elementos que cumplan dos o más predicados utilizando la palabra clave `AND`. La consulta 3.16 muestra el resultado de la conjunción de los predicados de las consultas 3.14 y 3.15.

Consulta 3.16: Id nombre altura y color pelo de los criminales bajos y con color de pelo gris

```
SELECT id,nombre,altura,colorPelo FROM criminal WHERE ~altura
    IS 'bajo' AND #colorPelo IS 'Gris'
```

id	nombre	altura	colorpelo	fuzzy_degree
2	María	152	Negro	0.7
7	Natalia	154	Rubio	0.5
9	Daniela	157	Castaño	0.3

La conjunción de predicados es equivalente a la operación de intersección de conjuntos difusos (sección 2.6). El grado de pertenencia de los criminales que cumplen con la condición `~altura IS bajo AND #colorPelo IS Gris` es el grado de pertenencia de los criminales en el conjunto difuso $bajo \cap similarity(Gris)$.

El motor de búsqueda presentado utiliza la función t-norma *min* (2.18) para calcular el grado de pertenencia. Daniela, que pertenecía con un grado de pertenencia 0,3 al resultado de la consulta 3.14 y con un grado de pertenencia 0,6 al resultado de la consulta 3.15, pertenece con un grado de pertenencia de 0,3 (mínimo) al resultado de la conjunción en la consulta 3.16.

Disyunción de predicados

Análogamente, es posible consultar por los elementos que cumplan con al menos uno de los predicados utilizando la palabra clave `OR`. La consulta 3.17

muestra el resultado de la disyunción de los predicados de las consultas 3.14 y 3.15.

Consulta 3.17: Id nombre altura y color pelo de los criminales bajos o los criminales color de pelo gris

```
SELECT id,nombre,altura,colorPelo FROM criminal WHERE ~altura
    IS 'bajo' OR #colorPelo IS 'Gris'
```

id	nombre	altura	colorpelo	fuzzy_degree
3	Pedro	169	Gris	1.0
2	María	152	Negro	0.8
11	Celeste	178	Negro	0.7
6	Rodrigo	190	Negro	0.7
4	Julio	172	Albino	0.6
7	Natalia	154	Rubio	0.6
8	Fabián	192	Castaño	0.6
9	Daniela	157	Castaño	0.6
10	Ana	170	Castaño	0.6
1	Juan	182	Castaño	0.6
5	Paula	162	Pelirrojo	0.5

La disyunción de predicados es equivalente a la operación de unión de conjuntos difusos (sección 2.6). El grado de pertenencia de los criminales que cumplen con la condición `~altura IS bajo OR #colorPelo IS Gris` es el grado de pertenencia de los criminales en el conjunto difuso $bajo \cup similarity(Gris)$.

El motor de búsqueda utiliza la función s-norma max (2.20) para calcular el grado de pertenencia. Daniela pertenece con un grado de 0,6 al resultado de la la consulta 3.17. Este grado es el máximo entre 0,3 (consulta 3.14) y 0,6 (consulta 3.15).

Los predicados utilizados en los ejemplos de composición son predicados difusos. En SQLf es posible utilizar también uno o más predicados clásicos; el comportamiento es el mismo solo que el grado de pertenencia para los predicados clásicos es restricto a 0 ó 1.

3.5.9. Agrupación y funciones de agregación

En SQL es posible agrupar filas de una consulta en grupos para luego operar sobre ellos. La forma de agrupar en SQL es utilizando la cláusula `GROUP BY`. El ejemplo 3.18 es una consulta SQL estándar que agrupa los criminales por color de pelo. Este ejemplo también utiliza la función de agregación `COUNT` para contar la cantidad de criminales que pertenecen a cada grupo.

Consulta 3.18: Criminales agrupados por color de pelo

```
SELECT colorPelo, COUNT(*) FROM criminal GROUP BY colorPelo
```

colorPelo	COUNT(*)
Albino	1
Castaño	4
Gris	1
Negro	3
Pelirrojo	1
Rubio	1

En esta consulta clásica cada criminal pertenece a un solo grupo. No es posible que una persona tenga el cabello *castaño* y *negro* al mismo tiempo. La suma de todas las cantidades es igual a la cantidad total de criminales en el sistema. Podríamos definir cada valor posible de color de pelo como un conjunto clásico donde cada criminal pertenece a uno solo. También podemos notar que el dominio de la propiedad color de pelo de un criminal es discreto y la cantidad de valores posibles es relativamente pequeña. Agrupar sobre un dominio continuo posiblemente no sería muy útil: la cantidad de los grupos podría ser muy grande y la cardinalidad de los mismos muy pequeña. No es útil agrupar los criminales por peso o altura ya que estos toman valores enteros.

Agrupación utilizando conjuntos difusos discretos

En SQLf, es posible agrupar las filas por conjunto difuso. Una fila pertenecerá al grupo si pertenece al conjunto con algún grado de pertenencia. Esta agrupación difusa se basa en la proyección difusa tratada en [KD]. Los grupos y los conjuntos difusos subyacentes están definidos por la variable lingüística utilizada. La consulta 3.19 no agrupa pero nos servirá para entender un poco más este tipo de consultas.

Consulta 3.19: Nombre color de pelo y similitud de color de pelo de todos los criminales

```
SELECT THRESHOLD 0.7 nombre, #colorPelo as colorPelo FROM
    criminal ORDER BY colorPelo
```

nombre	colorPelo	fuzzy_degree
Julio	Albino	1.0
Natalia	Albino	0.7
Fabián	Castaño	1.0
Daniela	Castaño	1.0
Ana	Castaño	1.0
Juan	Castaño	1.0
Celeste	Castaño	0.7
María	Castaño	0.7
Rodrigo	Castaño	0.7
Pedro	Gris	1.0
María	Gris	0.7
Celeste	Gris	0.7
Rodrigo	Gris	0.7
Rodrigo	Negro	1.0
María	Negro	1.0
Celeste	Negro	1.0
Fabián	Negro	0.7
Pedro	Negro	0.7
Ana	Negro	0.7
Juan	Negro	0.7
Daniela	Negro	0.7
Paula	Pelirrojo	1.0
Natalia	Rubio	1.0
Julio	Rubio	0.7

Podemos observar en el resultado de la consulta anterior que un criminal puede aparecer en más de una fila. Esto indica que un criminal pertenece a más de un conjunto de similitud. Un criminal puede tener color de pelo similar a más de un color de pelo. Julio que tiene color de pelo *albino* también tiene color de pelo similar al *rubio*. Esta consulta también retorna el grado de pertenencia del criminal al conjunto difuso. Agrupar por conjuntos difusos significa crear una fila por cada grupo de filas que estén relacionadas con un conjunto difuso. Ahora que entendemos esta característica mostraré un ejemplo de agrupación difusa en la consulta 3.20.

Consulta 3.20: Cantidad y cantidad difusa de los criminales agrupados por similitud de color de pelo

```
SELECT #colorPelo as colorPelo, COUNT(*), FUZZY_COUNT(*) as
    FUZZY_COUNT FROM criminal GROUP BY #colorPelo
```

colorPelo	COUNT($*$)	FUZZY_COUNT
Albino	4	2.5
Castaño	10	7.4
Gris	11	7.1
Negro	10	7.1
Pelirrojo	11	5.4
Rubio	11	3.5

La consulta anterior crea grupos por similitud de color de pelo. La variable lingüística utilizada es *Similitud de color pelo* (sección 3.4.4) y los posibles grupos son los conjuntos de similitud de esta variable lingüista. Un criminal pertenecerá a los grupos donde su color de pelo sea similar al del grupo. Esta consulta también muestra dos funciones agregación: `COUNT` y `FUZZY_COUNT`.

- La función `COUNT` cuenta la cantidad de criminales que pertenecen al grupo sin importar el grado de pertenencia que tiene cada criminal al conjunto difuso subyacente. Podemos decir que esta columna es la cardinalidad clásica del conjunto difuso.

- La función `FUZZY_COUNT` suma el grado de pertenencia de cada criminal en un grupo. La diferencia con `COUNT` es que se tiene en cuenta el grado de pertenencia. Si un criminal pertenece totalmente *pesará* 1 en la suma, si pertenece parcialmente *pesará* proporcionalmente al grado difuso de la fila al grupo.

Agrupación utilizando conjuntos continuos

En SQLf también es posible agrupar utilizando una variable lingüística que tiene conjuntos difusos sobre dominios continuos. La consulta 3.21 no es una agrupación pero nos ayudará a entender el comportamiento de cada uno de los criminales con respecto a los conjuntos difusos de la variable lingüística altura y de esta forma comprenderemos mejor el comportamiento de los grupos.

Consulta 3.21: Nombre y altura de los criminales

```
SELECT nombre, ~altura as altura FROM criminal ORDER BY
    altura
```

nombre	altura	fuzzy_degree
Juan	alto	1.0
Fabián	alto	1.0
Rodrigo	alto	1.0
Celeste	alto	0.8
Julio	alto	0.2
María	bajo	0.8
Natalia	bajo	0.6
Daniela	bajo	0.3
Paula	medio	1.0
Pedro	medio	1.0
Ana	medio	1.0
Julio	medio	0.6
Daniela	medio	0.4

Como podemos observar en la consulta, los valores posibles de altura no son números enteros sino las diferentes etiquetas lingüísticas que forman parte de la variable lingüística *altura* (subsección 3.4.4). Un criminal puede aparecer en más de una fila si pertenece a más de uno de los conjuntos difusos. Podemos ver que *Julio* es tanto *alto* con un grado de 1 como *medio* con un grado de 0,6 y es por ello que *Julio* pertenece a ambos grupos. La consulta 3.22 muestra un ejemplo de agrupación por la *altura* de los criminales.

Consulta 3.22: Cantidad y cantidad difusa de criminales agrupadas por altura

```
SELECT ~altura as altura, COUNT(*), FUZZY_COUNT(*) as
    FUZZY_COUNT FROM criminal GROUP BY ~altura
```

altura	COUNT($*$)	FUZZY_COUNT
alto	5	4.0
bajo	3	1.7
medio	5	4.0

Esta consulta muestra cuántos criminales pertenecen a cada conjunto. La primera suma es la cardinalidad clásica (5 criminales son altos o *algo* altos) y la segunda suma es la suma de los grados de pertenencia (4,0 es la suma difusa de los criminales altos).

Funciones de agregación

En las consultas anteriores se desarrolló informalmente cómo aplicar funciones de agregación sobre los grupos. A continuación mostraré las funciones de agregación que acepta esta implementación de SQLf y cómo son evaluadas. De cada función se describe la fórmula utilizada para evaluarla donde *column* es el

nombre de la columna en la cual se aplicará la función de agregación, n la cantidad de filas resultado que tiene el grupo, $column(i)$ es el valor de la columna en la fila i y $\mu_G(i)$ es el grado de pertenencia (`fuzzy_degree`) de la fila resultado del conjunto difuso formado por el grupo.

- $COUNT(*) = \sum_{i=1}^{n} 1$: Cuenta la cantidad de elementos que tiene el grupo sin importar el grado de pertenencia.
- $SUM(column) = \sum_{i=1}^{n} column(i)$: Suma la columna de todas las filas sin importar el grado de pertenencia.
- $AVG(column) = (\sum_{i=1}^{n} column(i))/n$: Promedio de la columna de todas las filas sin importar el grado de pertenencia.
- $MAX(column) = \text{máx}_{i=1}^{n} column(i)$: Máximo valor de la columna en el grupo sin importar el grado de pertenencia.
- $MIN(column) = \text{mín}_{i=1}^{n} column(i)$: Mínimo valor de la columna en el grupo sin importar el grado de pertenencia.
- $FUZZY\_COUNT(column) = \sum_{i=1}^{n} \mu_g(i)$: Cuenta la cantidad de elementos que tiene el grupo utilizando el grado de pertenencia para determinar el *peso* de la fila. Es la cardinalidad difusa del grupo.
- $FUZZY\_SUM(column) = \sum_{i=1}^{n} column(i) * \mu_g(i)$: Suma de la columna de todas las filas del grupo, tomando de cada fila solo la proporción de la pertenencia de la fila al grupo.
- $FUZZY\_AVG(column) = \sum_{i=1}^{n} column(i) * \mu_g(i) / \sum_{i=1}^{n} \mu_g(i)$: Es la división de la suma difusa sobre la cardinalidad difusa del grupo.

Todas las filas que pertenecen a agrupaciones clásicas (e.g: consulta 3.18) tienen un grado de pertenencia 1. En este caso `COUNT` será igual a `FUZZY_COUNT`, `SUM` será igual a `FUZZY_SUM` y `AVG` será igual a `FUZZY_AVG`.

3.5.10. Modificadores en en consultas difusas

Cómo hemos visto en la sección 2.7, un modificador difuso es un término que permite alterar la salida de una función de pertenencia creando nuevos conjuntos difusos. Los modificadores en SQLf se utilizan con el término seguido del carácter ! antes de la etiqueta lingüística en un predicado difusa. Las siguientes consultas son consultas donde los predicados son *alterados* con los modificadores difusos del sistema descritos en la sección 3.4.7.

Consulta 3.23: Id nombre y altura de todos los criminales altos

```
SELECT id, nombre, altura FROM criminal WHERE ~altura IS 'alto'
```

id	nombre	altura	fuzzy_degree
1	Juan	182	1.0
6	Rodrigo	190	1.0
8	Fabián	192	1.0
11	Celeste	178	0.8
4	Julio	172	0.2

Consulta 3.24: Id nombre y altura de todos los criminales MUY altos

```
SELECT id,nombre,altura FROM criminal WHERE ~altura IS very!
    'alto'
```

id	nombre	altura	fuzzy_degree
1	Juan	182	1.0
6	Rodrigo	190	1.0
8	Fabián	192	1.0
11	Celeste	178	0.64
4	Julio	172	0.04

La consulta 3.24 modifica la consulta 3.23 utilizando el modificador difuso *very* o *muy* donde $\mu_{\text{muy } A}(x) = \mu_A(x)^2$.

Consulta 3.25: Id nombre y altura de todos los criminales MÁS O MENOS altos

```
SELECT id,nombre,altura FROM criminal WHERE ~altura IS
    moreOrLess! 'alto'
```

id	nombre	altura	fuzzy_degree
1	Juan	182	1.0
6	Rodrigo	190	1.0
8	Fabián	192	1.0
11	Celeste	178	0.894427191
4	Julio	172	0.447213595

La consulta 3.25 también modifica la consulta 3.23 utilizando el modificador difuso *más_o_menos* donde $\mu_{\text{más\_o\_menos } A}(x) = \sqrt{\mu_A(x)}$.

3.5.11. Sentencias difusas cuantificadas

Los cuantificadores en general son utilizados para describir la cantidad de elementos que cumplen un predicado. Los cuantificadores difusos (tratados en 2.9) pueden ser utilizados en consultas difusas. Las sentencias cuantificadas en SQLf fueron introducidas por Bosc y Pivert en [Bos95] y están divididas en dos tipos: tipo I y tipo II. Las consultas cuantificadas también retornarán un valor de

verdad entre 0 y 1 indicando en qué grado *cumplen* los predicados con respecto al cuantificador.

Definición 26 (Proposición cuantificada del tipo I)**.** *Las proposiciones cuantificadas del tipo I son sentencias cuantificadas que utilizan un conjunto clásico como base para la cuantificación. Estas son de la forma:*

$$Q'X \text{ are } A$$

Donde X es un conjunto clásico, A un predicado difuso sobre X y Q es un cuantificador difuso. El cuantificador puede ser absoluto o relativo.

En lenguaje natural, estas consultas se expresarían como: *Q de los elementos de X cumplen A*. El conjunto X es la base de esta consulta. En nuestro sistema podríamos definir consultas como:

- *Al menos tres criminales son bajos.*
- *Alrededor de cinco criminales tienen pelo castaño.*
- *Casi todos los criminales son viejos.*

En el primer ejemplo, la base para la cuantificación es el conjunto clásico de los criminales (X), el cuantificador es el cuantificador absoluto *al menos tres* (Q) y el predicado difuso es *son bajos* (A).

Definición 27 (Proposición cuantificada del tipo II)**.** *Las proposiciones cuantificadas del tipo II son sentencias cuantificadas que utilizan un conjunto difuso como base para la cuantificación. Estas son de la forma:*

$$Q\ B'X \text{ are } A$$

Donde X es un conjunto difuso, B y A dos predicados difusos sobre X y Q es un cuantificador difuso.

En lenguaje natural, estas consultas se expresarían como: *Q de los elementos que cumplen B de X cumplen A*. En nuestro sistema podríamos definir consultas como:

- *La mayoría de los criminales viejos operan en Quilmes.*
- *Al menos diez criminales jóvenes son altos*
- *Casi todos los criminales bajos son flacos*

En el primer ejemplo, la base para la cuantificación es el conjunto difuso criminales viejos (B) sobre el conjunto clásico de los criminales (X), el cuantificador es cuantificador relativo *la mayoría* (Q) y el predicado es *operan en quilmes* (A).

Las proposiciones de tipo I y II se dicen que son proposiciones cuantificadas verticalmente. Cuando una consulta *evalúa* la cantidad de elementos que satisfacen un predicado se dice que la consulta es cuantificada verticalmente. Cuando una consulta *evalúa* la cantidad de predicados que un elemento satisface se dice que la consulta es cuantificada horizontalmente ([Tinb], [Tina]). Existen distintas interpretaciones matemáticas para las sentencias cuantificadas. Una interpretación definirá cómo el grado de pertenencia de la sentencia cuantificada será calculada. Estas propuestas pueden encontrarse en [Bos95] y [Tinb].

3.5.12. Consultas difusas cuantificadas

En SQLf es posible utilizar proposiciones cuantificadas dentro de las sentencias `SELECT`. Esto es posible mediante consultas anidadas similares a las consultas clásicas que utilizan `EXIST` o `ANY`. También es posible utilizar proposiciones cuantificadas cuando se desea filtrar grupos que no cumplen con una condición usando la cláusula `HAVING`. Este tipo de consultas no han sido implementadas en la librería SQLf_j que he utilizado para probar los ejemplos anteriores. Extender SQLf_j para que soporte estas consultas cuantificadas sería un buen punto para un futuro trabajo.

Los siguientes consultas son recolectadas y analizadas de la bibligrafía. Mostraré posibles consultas que se podrían definir en nuestro sistema ejemplo utilizando los cuantificadores descritos en 3.4.8. A continuación nombraré las diferentes alternativas para realizar consultas SQLf cuantificadas ([Tinb]).

Consultas anidados del tipo I

Las consultas anidadas del tipo I utilizan una sub-consulta en la condición. La se condición cumplirá cuando la cantidad de elementos de la sub-consulta cumpla el cuantificado difuso. Su sintaxis es la siguiente:

```
SELECT [THRESHOLD T] <atributos> FROM A WHERE
  Q(SELECT <atributos> FROM B WHERE <condiciones>)
```

Donde `A` y `B` son dos relaciones y `Q` un cuantificador. El umbral `T` es opcional. Las condiciones de la consulta anidada pueden hacer referencia a atributos de las relaciones A y B. El ejemplo 3.26 es una consulta anidada del tipo I que podría ser realizada en nuestro sistema.

Consulta 3.26: Zonas que tengan al menos tres criminales flacos

```
SELECT DISTINCT zona FROM criminal a WHERE
  al_menos_tres(SELECT * FROM criminal b
  WHERE b.~peso is 'flaco' AND a.zona = b.zona)
```

Consultas anidadas del tipo II

Las consultas del tipo II comparan un atributo de la relación consultada contra una sub-consulta. Si ese comparador es *verdadero* para la cantidad de elementos definidos por la sub-consulta y el cuantificador la fila aparecerá en el resultado. Su sintaxis es la siguiente:

```
SELECT [THRESHOLD T] <atributos> FROM A WHERE <atributo>
  <comparador> Q(SELECT <atributos> FROM B WHERE <condiciones
      >)
```

Donde `A` y `B` son dos relaciones y `Q` un cuantificador. El umbral `T` es opcional. Las condiciones de la consulta anidada pueden hacer referencia a atributos de las relaciones A y B. El atributo en la cláusula `WHERE` es un atributo de la relación `A` y está siendo evaluado comparándolo con la consulta anidada. Es posible que el comparador sea tanto un comparador clásico como difuso. Los comparadores difusos serán abarcados más adelante en esta tesis.

El ejemplo 3.27 es una consulta anidada del tipo II que podría ser realizada en nuestro sistema.

Consulta 3.27: Criminales con peso mayor a la mayoría de los criminales de la misma zona

```
SELECT * FROM criminal a WHERE peso > la_mayoria
  (SELECT * criminal b WHERE a.zona = b.zona)
```

Consultas agrupadas del tipo I

Las consultas agrupadas evalúan la cantidad de elementos del grupo que cumplen una condición y comparan esta cantidad con el cuantificador. Si el cuantificador es *verdadero*, la fila definida por el grupo aparecerá en el resultado. Su sintaxis es la siguiente:

```
SELECT [THRESHOLD T] <atributos> FROM A
  GROUP BY <atributos> HAVING Q ARE <condiciones>
```

Donde `A` es una relación y `Q` un cuantificador. El umbral `T` es opcional. Esta consulta agrupa las filas utilizando la lista de `<atributos>` de manera similar a una consulta SQL clásica. Luego suma la cantidad de los elementos del grupo que satisfacen las condiciones y compara esta cantidad con el cuantificador. Si esa cantidad satisface el cuantificador en un grado mayor al grado `<threshold>`, el grupo pertenecerá al resultado final. Existen varias interpretaciones matemáticas para calcular la suma de los elementos del grupo, la más trivial es utilizar la suma difusa (`FUZZY_SUM`) cuando el cuantificado es absoluto, y el promedio difuso (`FUZZY_AVG`) cuando el cuantificador es relativo. Estas funciones fueron vistas en 3.5.9. Las condiciones pueden hacer referencia tanto a atributos de la relación como a funciones de agregación. El ejemplo 3.28 es una consulta de este tipo.

Consulta 3.28: Zonas donde un tercio de los criminales tiene color de pelo similar al gris

```
SELECT zona FROM criminal a GROUP BY
  zona HAVING un_tercio ARE (#colorPelo is 'Gris')
```

En la consulta anterior, SQLf creará grupos de criminales por zonas. Luego contará la cantidad de criminales con pelo similar al gris y en base a ese número sobre el tamaño del grupo calculará el promedio difuso. El promedio difuso será parámetro de la función de pertenencia definida en el cuantificador *un_tercio* y el resultado será el grado de pertenencia final del grupo.

Consultas agrupadas del tipo II

Las consultas agrupadas del tipo II utilizan dos condiciones en la cláusula HAVING y su sintaxis es la siguiente:

```
SELECT [THRESHOLD T] <atributos> FROM A
  GROUP BY <atributos> HAVING
  Q <condicionesA> ARE <condicionesB>
```

Como se puede observar, estas consultas son muy similares a las agrupadas del tipo I. La diferencia es que utilizan dos predicados; las condiciones `A` limitan los elementos del grupo (formando un sub-grupo) que será evaluado con las condiciones `B`. El promedio difuso entre el tamaño del sub-grupo y los elementos del sub-grupo que cumplen `B` será el argumento de la función de pertenencia del cuantificador `Q`. El resultado de la función será el grado de pertenencia final del grupo que tiene que ser superior al umbral `T` para que el grupo aparezca en los resultados.

Si el cuantificador `Q` es absoluto o las condiciones `A` son condiciones booleanas clásicas, las consultas agrupadas del tipo II pueden ser reducidas a consultas agrupadas del tipo I ([Tinb]). La consulta anterior será equivalente a:

```
SELECT [THRESHOLD T] <atributos> FROM A
  GROUP BY <atributos> HAVING
  Q (<condicionesA> AND <condicionesB>)
```

El ejemplo 3.29 es una consulta del tipo II que nos puede servir en nuestro sistema ejemplo.

Consulta 3.29: Zonas donde la mitad de los criminales que son altos tiene color de pelo similar al gris

```
SELECT zona FROM criminal a GROUP BY
  zona HAVING la_mitad (~altura is 'Alto')
  ARE (#colorPelo is 'Gris')
```

3.6. Consultas DDL de SQLf

Hasta el momento hemos visto cómo utilizar los elementos de la lógica difusa en la sentencia SELECT para aumentar su poder expresivo. SQLf tiene sentencias especiales para poder definir cada uno de estos elementos de la lógica difusa. A continuación mostraré cómo SQLf nos permite crear las etiquetas, predicados, cuantificadores y modificadores de nuestro sistema ejemplo. Estas sentencias son parte del Lenguaje de Definición de Datos (DDL) de SQLf. Lamentablemente SQLf_j tampoco tiene implementadas las sentencias DDL de SQLf pero es posible agregar la meta-información difusa por medio de inserciones clásicas SQL.

3.6.1. Creación de predicados

En la sección 3.4.3 hemos definido las etiquetas y conjuntos difusos que luego hemos utilizado durante las consultas del capítulo. SQLf provee de sentencias para crear estas etiquetas y sus funciones de pertenencia asociadas. La sintaxis para crear un predicado difuso sobre dominio discreto es la siguiente:

```
CREATE FUZZY PREDICATE <nombre>
  ON n..m AS (a, b, c, d)
```

Donde `<nombre>` es nombre de la etiqueta del predicado. El predicado está definido en base al trapezoide normal $\mu_{NormalTrapezoid(a,b,c,d)}$ donde el valor de entrada debe estar en el rango $[n, m]$. Los valores reales a, b, c y d pueden ser reemplazados por la palabra [-] INIFINIT para representar trapezoides a derecha o izquierda.

Para asociar el predicado a una función Gausiana se utiliza la siguiente sintaxis:

```
CREATE FUZZY PREDICATE <nombre>
  ON n..m AS GAUSSIAN(u,k)
```

El predicado de nombre `<nombre>` estará definido por la función Gausiana $\mu_{Gaussian(u,k)}$. La consulta 3.30 crea algunos de los predicados usados en el capítulo.

Consulta 3.30: Crear los predicados *bajo* *alto* *flaco* y *gordo* del ejemplo

```
CREATE FUZZY PREDICATE bajo
  ON 0..250 AS (-INFINIT, -INFINIT, 150, 160)

CREATE FUZZY PREDICATE alto
  ON 0..250 AS (170, 180, INFINIT, INFINIT)

CREATE FUZZY PREDICATE flaco
  ON 0..300 AS (-INFINIT, -INFINIT, 50, 60)
```

```
CREATE FUZZY PREDICATE obeso
  ON 0..300 AS GAUSSIAN(100,20)
```

Un predicado puede ser eliminado utilizando DROP PREDICATE (ejemplo 3.31).

Consulta 3.31: Eliminar el predicado *flaco*

```
DROP PREDICATE flaco
```

3.6.2. Creación de cuantificadores

La creación de cuantificadores difusos en SQLf es similar a la creación de predicados. Su sintaxis es la siguiente:

```
CREATE [ABSOLUTE|RELATIVE]
  QUANTIFIER <nombre> AS (a, b, c, d)
```

Donde <nombre> es nombre del cuantificador definido por el trapezoide normal $\mu_{NormalTrapezoid(a,b,c,d)}$. Los valores reales a, b, c y d pueden ser remplazados por la palabra [-] INIFINIT para representar trapezoides a derecha o izquierda. La palabra reservada ABSOLUTE o RELATIVE define el tipo del cuantificador. Los cuantificadores están asociados a funciones de pertenencia trapezoidales. Una extensión interesante sería poder definir cuantificadores en base a otro tipo de funciones.

Consulta 3.32: Crear los cuantificadores *un_tercio* y *al_menos_tres* de la aplicación ejemplo

```
CREATE RELATIVE QUANTIFIER un_tercio
  AS (0.2, 0.25, 0.35, 0.4)

CREATE ABSOLUTE QUANTIFIER \textit{al_menos_tres}
  AS (2, 3, INIFINIT, INIFINIT)
```

La eliminación de un cuantificador es utilizando DROP QUANTIFIER(ejemplo 3.33).

Consulta 3.33: Eliminar el cuantificador *al_menos_tres*

```
DROP RELATIVE ABSOLUTE al_menos_tres
```

3.6.3. Creación de modificadores y traslaciones

En SQLf existen dos alternativas para crear modificadores:

- Por medio de una función potencia.
- Por medio de una función triangular.

Función potencia:

CREATE MODIFIER <nombre> **AS** POWER <n>

Donde el modificador <nombre> estará definido en base a la función potencia de <n>.

Función triangular:

CREATE MODIFIER <nombre> **AS** <norma> POWER <n>

Donde <norma> es una norma o una conorma triangular que se especifica mediante una expresión con las variables predefinidas x, y que denotan al primer y al segundo argumento respectivamente ([Tinb]).

También es posible definir una traslación:

CREATE MODIFIER <nombre> **AS TRANSLATION** <n>

Donde la traslación <nombre> estará definida por la función $\mu_A(x + n)$.

Consulta 3.34: Crear los modificadores *muy* *mas_o_menos* *extremadamente* y la traslación *realmente* de la aplicación ejemplo

CREATE MODIFIER muy **AS** POWER 2

CREATE MODIFIER mas_o_menos **AS** POWER −2

CREATE MODIFIER extremadamente **AS MAX**(x+y−1,0) POWER 4

CREATE MODIFIER realmente **AS TRANSLATION** −10

La forma de eliminar un modificador es mediante DROP MODIFIER.

Capítulo 4

Base de datos difusas y FSQL

4.1. Introducción

Como hemos visto, SQLf es un lenguaje que permite realizar consultas difusas a base de datos. Es una de las línea más importantes de investigación que da solución a sistemas de primer nivel. Paralelamente, existe otra linea de investigación también muy importante que da solución a sistemas de segundo nivel, es decir, permite no solo poder realizar consultas difusas sino también almacenar información difusa. Esta línea de investigación es la definida teóricamente por Medina-Pons-Miranda ([MPM94]) en su modelo GEFRED e implementada por Galindo-Medina-Pons-Cubero ([GMPC98]) en su modulo FIRST y lenguaje FSQL.

Es posible decir que los problemas que resuelve un sistema de primer nivel, también los podría resolver un sistema de segundo nivel. Un sistema de segundo nivel podría realizar consultas difusas sobre una base de datos clásica comportándose como un sistema de primer nivel. Aunque SQLf y FSQL son similares, existen diferencias. SQLf tiene características que FSQL no presenta. También las librerías que implementan estos lenguajes tienen características diferentes. En este capítulo mostraré las características principales del lenguaje FSQL y las compararé con SQLfñ pero previamente mostraré de forma breve el modelo teórico GEFRED y el módulo FIRST.

4.2. El modelo GEFRED

El modelo GEFRED (GEneralized model for Fuzzy RElational Databases, Modelo Generalizado para Base de Datos Difusas) introducido por Medina-Pons-Miranda [MPM94] es un modelo teórico que permite representar información

precisa e imprecisa en base de datos relacionales. Este modelo es una extensión del modelo relacional.

GEFRED extiende el modelo relacional basándose en los conceptos de *dominios difusos generalizados* y de *relaciones difusas generalizadas*. El trabajo de Medina-Pons-Miranda también describe un *álgebra relacional difusa generalizada* sobre el álgebra relacional clásica y extiende todas las operaciones clásicas como selección, unión, intersección, producto cartesiano, join y proyección permitiendo manipular relaciones difusas. Este modelo permite utilizar no solo comparadores clásico como *mayor*, *menor*, *igual*, etc; si no que también permite utilizar comparadores difusos. Estos comparadores son funciones que toma dos elementos del dominio y retorna un valor real [0,1] y no un valor booleano como los comparadores clásicos. Como ejemplos de comparadores difusos tenemos *mucho mayor*, *aproximadamente igual* o *más eficiente que*. En [MPM94] se pueden encontrar todas la definiciones formales de estas extensiones.

Al mismo tiempo GEFRED adopta conceptos de modelos anteriores que intentan almacenar información imprecisa; entre ellos se encuentran las relaciones de similitud entre valores escalares, numéricos y números difusos del modelo de Buckles-Petry ([BP84]) y los conceptos de *desconocido* (Unknown), *indefinido* (Undefined) y *nulo* (Null) de Umano-Fukami ([UF94]).

Una relación de similitud sirve para medir la similitud o el parecido entre dos elementos. El valor de similitud toma valores en el intervalo [0,1] siendo 0 *completamente diferentes* y 1 *completamente similares o iguales*. En la sección 2.4.1 hemos visto cómo la *similitud* nos permite definir conjuntos difusos.

GEFRED utiliza *distribuciones de posibilidad* para representar el grado de posibilidad, de pertenencia o veracidad de los atributos. Estos valores son representados p/v, donde p es la posibilidad [0,1] de que el valor v sea el *real*. La siguiente lista muestra los posible tipos de los valores que forman parte del *dominio difuso generalizado*:

1. Un escalar simple (Ej. $Aptitud = buena$ con distribución de posibilidad de $1/buena$).

2. Un número simple (Ej. $Edad = 28$ con distribución de posibilidad de $1/28$).

3. Un conjunto de posibles asignaciones excluyentes de escalares (Ej. $Aptitud = \{mala, buena\}$ con posibilidad $\{1/mala, 1/buena\}$).

4. Un conjunto de posibles asignaciones excluyentes de número (Ej. $Edad = \{20, 21\}$ con posibilidad $\{1/20; 1/21\}$).

5. Una distribución de posibilidad en el dominio de los escalares (Ej. $Aptitud = \{0{,}6/mala; 0{,}7/buena\}$).

6. Una distribución de posibilidad en el dominio de los números (Ej. $Edad = \{0{,}4/23; 1{,}0/24; 0{,}8/25\}$).

7. Un número Real en [0,1] representando grados de cumplimiento (Ej. $Calidad = 0{,}9$).

8. Un valor desconocido (Unknown) dado por la distribución de posibilidad $Unknown = \{1/u : u \in U\}$.

9. Un valor indefinido (Undefined) dado por la distribución de posibilidad $Undefined = \{0/u : u \in U\}$.

10. Un valor nulo (Null) dado por $Null = \{1/Unknown; 1/Undefined\}$.

Para entender mejor cómo GEFRED almacena la información, mostraremos un ejemplo de una relación o tabla donde algunos atributos son valores imprecisos o difusos. La tabla 4.1 representa una relación difusa. Las columnas nombre y rendimiento son columnas precisas o *crips* con valores escalares simples, la columna salario es otra columna precisa de tipo numérico.

La columna edad es una columna con valores imprecisos o difusos. Esto significa que no sabemos la edad exacta de las personas pero tenemos cierta información que nos será útil al momento de consultar y tomar decisiones. El símbolo & indica distribución de posibilidad, el símbolo # significa *aproximadamente* y el símbolo * es un intervalo. En esta relación podemos ver que *Luis* tendría una edad de 30 o 31 con una posibilidad de 0,8 y 1 respectivamente. *Antonio* es *joven* con distribución de posibilidad definida por la función de pertenencia de la etiqueta *joven* (figura 4.1). *Julia* tiene una edad de aproximadamente 28 y la edad de *Ana* se encuentra entre 30 y 35.

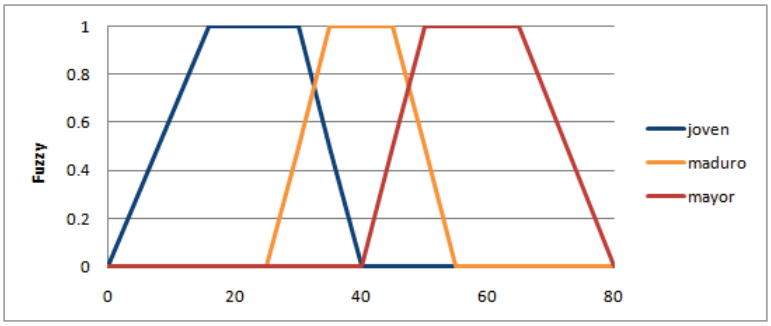

Figura 4.1: Etiquetas lingüísticas definidas sobre el dominio edad del ejemplo GEFRED

Los valores posibles de la columna rendimiento son *malo*, *regular*, *bueno* y *excelente*. Aún asumiendo que la columna rendimiento es precisa, ya que sabemos con una posibilidad 1 que el rendimiento de *Antonio* es *regular*, GEFRED nos

Nombre	Edad	Rendimiento	Salario
Luis	&.8/30,1/31	Bueno	110000
Antonio	Joven	Regular	100000
Juan	Mayor	Malo	90000
Francisco	30	Excelente	150000
Julia	#28	Bueno	130000
Ana	*30,35	Bueno	105000

Cuadro 4.1: Ejemplo de una relación difusa en GEFRED

	Malo	Regular	Bueno	Excelente
Malo	1	0.8	0.5	0.1
Regular	0.8	1	0.7	0.5
Bueno	0.5	0.7	1	0.8
Excelente	0.1	0.5	0.8	1

Cuadro 4.2: Ejemplo de relaciones de similitud en GEFRED

permite definir la relación de similitud que existe entre cada tupla de valores. La tabla 4.2 muestra un ejemplo de relaciones de similitud.

4.3. FIRST y FSQL

Una de las investigaciones más importantes sobre base de datos difusas es la realizada por Galindo en su tesis de doctorado [Góm99]. Galindo crea un módulo que extiende el gestor de base de datos Oracle permitiendo soportar el modelo teórico GEFRED y de esta forma representar y manipular información imprecisa. A este módulo lo denomina FIRST (Fuzzy Interface for Relational SysTems, Interface Difuso para Sistemas Relacionales).

Para poder operar sobre FIRST Galindo utiliza el lenguaje de consultas *FSQL* (SQL difuso o Fuzzy SQL). FSQL es una extensión de SQL que no solo permite las operaciones clásicas, sino que también manipula los elementos de la lógica difusa del modelo GEFRED. FIRST junto con FSQL forman la implementación más importante hasta el momento del modelo de GEFRED.

4.3.1. FIRST

La arquitectura de FIRST está formada por distintos módulos. La figura 4.2 muestra los módulos de FIRST que son explicados brevemente a continuación:

- SGBDR (Sistema de gestión de bases de datos relacionales, Relational database management system, RDBMS): es el sistema de base de datos que

proporciona la plataforma para FIRST. El servidor FSQL procesará todas las consultas FSQL y las traducirá al lenguaje anfitrión ejecutable en el SGBDR. En su implementación el lenguaje anfitrión es SQL y PL/SQL de Oracle.

- BD (Base de datos): La base de datos propiamente dicha que almacenará en forma relacional la información de la aplicación. Esta base de datos permitirá almacenar datos difusos. Aunque las tablas son tablas relacionales comunes, el formato de las columnas para almacenar atributos difusos es diferente. Los atributos difusos pueden ser de tres tipo:
  - Tipo 1: Son atributos con datos precisos o clásicos y pueden tener etiquetas lingüísticas asociados a ellos. Es posible realizar consultas difusas sobre ellos aunque no soporten valores difusos. Por ejemplo, el atributo salario almacena el valor en pesos de empleado y sería posible consultar por empleados con salario *alto*.
  - Tipo 2: Son atributos que pueden almacenar datos imprecisos o difusos sobre un dominio ordenado. La tabla 4.3 muestra las posibles constantes que se pueden almacenar en estos atributos. Por ejemplo, el atributo salario podría almacenar valores subjetivos cómo *alto*, alrededor de 4000, intervalo $[4000, 5000]$ o trapezoide $(5000, 5200, 5500, 5700)$.
  - Tipo 3: Son atributos que pueden almacenar datos imprecisos o difusos sobre un dominio discreto. Se deberán definir las relaciones de similitud entre los valores del dominio. Por ejemplo, el atributo actitud podría almacenar el valor *bueno* que caracteriza al empleado.
- FMB (Fuzzy Metaknowledge Base, Base de meta-conocimiento difuso): Son las tablas de las base de datos que almacenan el metaconocimiento de la información difusa. En estas tablas se definen los conjuntos difusos, variables y etiquetas lingüísticas, conjuntos de similitud, etc. Esta información es independiente de los datos propios de la aplicación, pero estos podrán utilizar la FMB para almacenar el conocimiento difuso. Por ejemplo, la FMB almacenará el significado de las etiqueta lingüísticas *salario alto* y *salario bajo* y su relación mientras que la base de dato difusa almacenará los empleados donde sus salarios podrían ser representados con estas etiquetas. La FMB almacenará la meta información acerca de los tres tipos posibles de atributos difusos.
- Servidor FSQL: Son los módulos implementados en PL/SQL que convierten las consultas FSQL a consultas SQL que el RDBMS ejecutará. Tanto el proceso de traducción como la ejecución de las consultas SQL utilizan la FMB. Cualquier sistema que tiene como base de datos Oracle podría instalar este servidor.

Constante	Significado
UNKNOWN	El valor es desconocido pero es aplicable.
UNDEFINED	El atributo no aplicable o no tiene sentido.
NULL	Ignorancia total.
$[a, b, c, d]$	El valor es definido con una función trapezoidal.
$\$label$	Etiqueta lingüística, puede ser una función trapezoidal o un valor escalar.
$[m, n]$	Intervalo *entre* m *y* n. Es una función trapezoidal donde $a = b = m$ y $c = d = n$.
$\#n$	Valor *aproximado a* n. Es una función trapezoidal donde $b = c = n$, $a = n - m$ y $d = n + m$ donde m es el margen.

Cuadro 4.3: Constantes difusas que pueden ser almacenadas o utilizadas en consultas FSQL

- Cliente FSQL: Es un cliente que permite al usuario ejecutar consultas SQLF sobre el servidor. En particular, el cliente que la tesis presenta es una aplicación Windows que permite al usuario editar consultas FSQL para luego traducirlas y/o ejecutarlas. El único requisito para que una aplicación programada en cualquier lenguaje pudiera ejecutar consultas FSQL sobre el servidor es poder invocar procedimientos almacenados (store procedures, [Wikb]) de Oracle, como por ejemplo una aplicación Java utilizando JDBC (Java DataBase Connectivity, [Wika]).

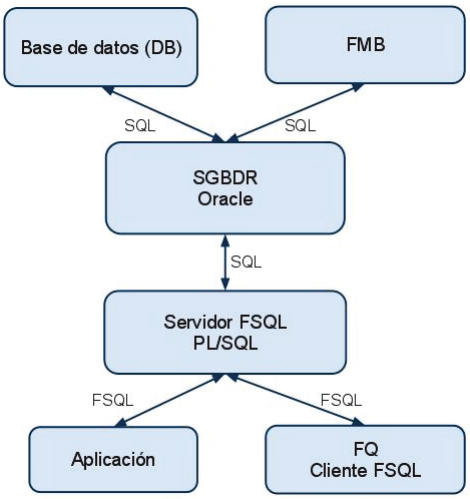

Figura 4.2: Módulos de FIRST

El trabajo de Galindo [Góm99] explica en detalle cómo son los esquema de base de datos de la FMB y cómo son los esquemas de las base de datos propias de

una aplicación que desea almacenar atributos de los tres tipos difusos. También muestra ejemplos de cómo los datos serán almacenados en estas tablas.

4.3.2. El lenguaje FSQL

El lenguaje FSQL es el lenguaje elegido por Galindo que las aplicaciones utilizarán para comunicarse con FIRST. Al igual que SQLf, FSQL es una extensión de SQL. FSQL está divida en sentencias DML para la manipulación de datos y sentencias DDL para la modificación del esquema de una base de datos difusas. A continuación resumiré brevemente la sintaxis y los aportes que realiza cada extensión de las sentencias FSQL para el tratamiento de información imprecisa.

4.3.3. Sentencias DML de FSQL

Las sentencias DML trabajan sobre los datos de una base de datos, Las más utilizadas son las operaciones INSERT, DELETE, UPDATE y particularmente SELECT. Según Galindo, FSQL es una extensión auténtica de SQL, todas las sentencias válidas en SQL lo son también en FSQL.

La sentencia SELECT de FSQL

La sentencia SELECT es utilizada para recuperar información de una base de datosl; en particular de tablas y vistas. En el caso de FSQL, es posible recuperar información imprecisa. Las extensiones efectuadas son las siguientes:

Etiquetas lingüísticas

Si un atributo es susceptible de tratamiento difuso, entonces es posible definir etiquetas sobre él. Las etiquetas lingüísticas son precedidas por el símbolo $. Las etiquetas lingüísticas en FSQL pueden estar definidas sobre un dominio subyacente *ordenado* o a un dominio subyacente *no ordenado*. En el caso de dominio *ordenado*, una etiqueta estará asociada a una distribución de posibilidad trapezoidal. Por ejemplo, podemos definir las etiquetas $BAJO, $NORMAL y $ALTO sobre el atributo *altura* de una persona. En el caso de un dominio *no ordenado*, una etiqueta estará asociada a una relación de similitud entre los elementos del dominio. Por ejemplo, podemos describir el atributo que almacenará el color de pelo de una persona como $RUBIO, $CASTAÑO y $PELIRROJO y definir el grado de similitud entre estas etiquetas.

Las etiquetas lingüísticas en FSQL se utilizan de manera similar a las SQLf. SQLf delega a la implementación las funciones que son posibles asociar a las etiquetas. En dominios continuos, la implementación SQLf_j no solo permite funciones trapezoidales sino también funciones normales. Como extensión de FSQL sería interesante poder asociar una etiqueta con otras funciones como las funciones normal, max-min, trapezoides no normales (2.4), etc; o poder relacionarla a

Comparador de Posibilidad	Comparador de Necesidad	Significado (Posiblemente y Necesariamente)
FEQ	NFEQ	Fuzzy igual (Fuzzy Equal)
FGT	NFGT	Fuzzy mayor que (Fuzzy Greater Than)
FGEQ	NFGEQ	Fuzzy mayor igual que (Fuzzy Greater or Equal)
FLT	NFLT	Fuzzy menor que (Fuzzy Less Than)
FLEQ	NFLEQ	Fuzzy menor o igual que (Fuzzy Less or Equal)
MGT	NMGT	Mucho mayor que (Much Greater Than)
MLT	NMLT	Mucho menor que (Much Less Than)

Cuadro 4.4: Compradores difusos de FSQL

intervalos y valores aproximados. En dominios discretos, tanto SQLf como FSQL utilizan relaciones de similitud de la misma manera. La principal ventaja de FSQL es que una etiqueta puede ser *guardada* como valor de columna cuando se desconoce su valor concreto.

Comparadores Difusos

Además de los comparadores clásicos (=, >, etc.), FSQL incluye comparadores difusos de la tabla 4.4. Estos comparadores están divididos en comparadores de *posibilidad* y de *necesidad*. Los comparadores de *posibilidad* son menos restrictivos que los de *necesidad*; es decir, recuperarán más resultados. Sobre atributos de dominio no ordenados solo puede usarse el comparador FEQ, ya que no hay orden. Es posible componer la negación (NOT) con los comparadores *invirtiendo* el resultado. Por ejemplo `NOT <a> FEQ <b>`. Como hemos visto, estos comparadores difusos retornan un valor de pertenencia [0,1] y no un valor de verdad. En [Góm99] se pueden ver las definiciones matemáticas de los comparadores y ejemplos de uso.

El lenguaje SQLf también permite la utilización de comparadores difusos. Como ya hemos visto, SQLfi tienen sentencias especificas para que el usuario cree sus propios comparadores. La librería SQLf_j no tiene implementado este tipo de comparadores. Aunque FSQL utiliza comparadores difusos, estos vienen con el lenguaje y no tienen alguna sentencia que permita al usuario agregar nuevas.

Conectores difusos

En FSQL, dos condiciones pueden ser conectadas con los operados lógicos AND, OR y NOT. Del mismo modo en que los conectores AND, OR y NOT de SQL estándar están relacionados a las operaciones de intersección, unión y opuesto en los conjuntos clásicos; los operadores AND, OR y NOT de FSQL están relacionadas a las mismas operaciones pero en el mundo de los conjuntos difusos (2.6). En FSQL, AND y OR son dos funciones $[0,1]X[0,1] \longrightarrow [0,1]$ y por defecto son las funciones triangulares mínimo y máximo respectivamente. En FSQL, el operador NOT es una función $[0,1] \longrightarrow [0,1]$ y por defecto es la función $1-\mu$.

Los conectores en FSQL y SQLf son tratados de la misma manera. El usuario tiene la posibilidad de cambiar las funciones que FSQL usará para AND, OR y NOT cambiando la configuración del servidor FSQL PL/SQL.

Umbral de cumplimiento

El SELECT de FSQL utiliza la palabra reservada `THOLD [comparador] T` para indicar que una condición simple debe ser satisfecha con un grado mínimo de $T \in [0,1]$ para que la tupla evaluada aparezca en el resultado. El `comparador` es opcional y por defecto es $\geq$. El comparador $<$ devolverá todas tuplas con un grado menor al umbral; es decir, las menos importantes. El comparador $=$ devolverá todas las tuplas que cumplan exactamente con el umbral.

La ventaja principal de FSQL es que permite cambiar el comparador. SQLf solo permite el comparador $\geq$ aunque la implementación específica podría agregar esta característica. La librería SQLf_j utiliza la palabra THRESHOLD T seguida de la palabra SELECT. En cambio SQLfi utiliza las palabras WITH CALIBRATION T al final de la consulta. El significado en ambas es el mismo.

Grado de compatibilidad o pertenencia

La función `CDEG(atributo)` (Compatibiliy DEGree) puede ser utilizada dentro en la lista de selección para retornar el grado de pertenencia de una tupla al resultado. Si su argumento es una columna, mostrará el grado de cumplimiento de las condiciones que contienen dicha columna. Si su argumento es asterisco (*), esta función de selección mostrará el grado de cumplimiento de toda la condición. $CDEG$ utilizará las funciones triangulares para *juntar* los grados de pertenencias de las condiciones compuestas. El carácter % es utilizado como comodín para mostrar el grado de pertenencia de todas las columnas que forman parte de las condiciones de manera independiente.

La ventaja principal de FSQL sobre SQLf es que permite retornar el grado de pertenencia de las condiciones relacionadas con un solo atributo además del grado de pertenencia de la condición completa. Esta columna es explícita siendo

opcional en el resultado. En la librería SQLf_j el grado de pertenencia, llamado `fuzzy_degree`, siempre aparece en la última columna. En SQLfi siempre aparece como primera columna de consultas difusas y se llama `mu`.

Constantes difusas

De la misma manera en que el usuario puede utilizar valores constantes en consultas SQL, FSQL también acepta constantes difusas. Estas constantes difusas son las vistas en la tabla 4.3 y pueden ser utilizadas en las condiciones difusas.

La sintaxis por defecto de SQLf no permite estas constantes y tampoco están implementadas en SQLf_j y SQLfi. Una de las extensiones que desarrollé en el capítulo 5 es cómo describir estas constantes por medio de funciones difusas anónimas.

Comparador IS

FSQL extiende el comparador *IS* de SQL permitiendo preguntar si un atributo es `NULL`, `UNDEFINED` o `UNKNOWN`. La forma de este comparador es:

```
<atributo> IS [NOT] (NULL|UNDEFINED|UNKNOWN)
```

Esta extensión no aplica a SQLf debido a que este lenguaje no permite almacenar valores como `UNDEFINED` o `UNKNON` en la base de datos. Como ya sabemos, SQLf se basa en el modelo relacional clásico que solo permite valores nulos.

Agrupación usando cuantificadores

FSQL también nos permite utilizar cuantificadores tanto relativos como absolutos (tratados en 2.9) en las consultas. Estas consultas pueden ser tanto del tipo 1 como del tipo 2 (3.5.11) y la sintaxis es la siguiente:

- `$Cuantificador FUZZY[r] (condicion_difusa) THOLD T` (Tipo 1)
- `$Cuantificador FUZZY[r] (condicion_difusa_1) ARE (condicion_difusa_2) THOLD T` (Tipo 2)

Donde `Cuantificador` es un cuantificador (absoluto o relativo) precedido por el símbolo $. Algunos cuantificadores pueden recibir argumentos entre corchetes, e. g. `$AL_MENOS(3)` donde `AL_MENOS` es un cuantificador absoluto *parametrizable* y `3` es su argumento.

La palabra `FUZZY[r]` es opcional e indica si la *proporción* a tomar en cuenta de la fila resultante es completa o relativa. Si no se encuentra la palabra `FUZZY`, se tomará completamente la fila (proporción 1,0) aún cuando la fila no cumpla completamente con la condición. Si se encuentra la palabra `FUZZY`, la proporción de la fila al resultado será el grado de pertenencia de la condición. En ambos

casos, el grado de pertenencia de la fila debe ser mayor al umbral definido con `THOLD` o sino no será tomada en cuenta.

Los cuantificadores son especialmente útiles en consultas con agrupación. Recordemos que la cláusula GROUP BY de SQL agrupa los resultados de una consulta utilizando alguna columna, y la cláusula HAVING filtra los grupos que no cumplan cierta condición. FSQL permite definir la condición de la cláusula HAVING utilizando cuantificadores. Estos cuantificadores definirán el grado de pertenencia de cada grupo filtrándolos en caso de que no superen el umbral. Estas consultas son muy similares a las de SQLf explicadas en 3.5.12.

Aunque la sintaxis y la evaluación de consultas cuantificadas en FSQL ha sido defina, el servidor actual de FSQL no tiene estas consultas implementadas. La posibilidad de personalizar la evaluación matemática mediante la palabra `FUZZY[r]` es una ventaja con respecto a SQLf. Aun así, las consultas cuantificadas en SQLf han sido más desarrolladas; como por ejemplo las consultas cuantificadas anidadas explicadas previamente en 3.5.12 y las consultas cuantificadas horizontalmente ([Tina])

El lenguaje FSQL también carece de una sintaxis que permita crear los cuantificadores. Esta sintaxis no debería diferir en mucho de la sintaxis utiliza para definir etiquetas sobre dominios continuos. Las funciones asociadas a los cuantificadores son muy similares a las funciones de pertenencia, por ejemplo una función trapezoidal $\mathbb{R} -> [0,1]$ podría estar asociada a un cuantificador absoluto y una función trapezoidal $[0,1] -> [0,1]$ a un cuantificador relativo.

4.3.4. Sentencias para modificar datos de FSQL

Como hemos visto en 3.1.2, las sentencias de SQL para modificar los valores guardados en una base da datos son INSERT, DELETE y UPDATE. FSQL extiende estas sentencias para permitir modificar los datos difusos que la SGBRD guardará. La sintaxis utilizada es muy similar a la original de SQL. Las modificaciones son las siguientes:

- **INSERT:** Permite insertar valores difusos como los de la tabla 4.3.
- **DELETE:** Permite utilizar condiciones difusas en la cláusula WHERE de manera similar a la cláusula SELECT.
- **UPDATE:** Permite actualizar columnas con valores clásicos o difusos y las condiciones de la cláusula WHERE también pueden ser difusas.

La posibilidad de guardar valores difusos no aplica a SQLf, pero sí la posibilidad de agregar condiciones difusas a las sentencias DELETE y UPDATE. Aunque estas condiciones pueden ser difusas, una fila no puede ser actualizada o borrada parcialmente. El umbral de cumplimiento es particularmente importante

en estos casos. Si el grado de pertenencia de la fila a la condición es superior al umbral, la fila será completamente actualizada o borrada; en caso contrario la fila no se modificará. La librería SQLfi tiene implementada las extensiones de DELETE y UPDATE permitiendo condiciones difusas. La libraría SQLf_j carece de esta característica.

4.3.5. Sentencias DDL de FSQL

Las sentencias DDL modifican la estructura de la base de datos agregando tablas, columnas, claves, índices, etc. En el caso de FSQL, esto también implica la capacidad de agregar y modificar columnas que guardarán valores difusos. FSQL también tiene sentencias para modificar FML, permitiendo almacenar y modificar los elementos de la lógica difusa que usaremos durante las consultas; como son las etiquetas lingüísticas y las relaciones de similitud. A continuación resumiré las novedades más importantes de FSQL.

4.3.6. Creación/alteración de una tabla

Las operaciones de CREATE/ALTER TABLE fueron extendidas para soportar los siguiente:

- **Tipos de datos:** Se pueden definir columnas con los tres tipos de datos nuevos (4.3.1). Para los atributos difusos de tipo 1 se utilizan la palabras reservadas `CRISP` o `FTYPE1`, para los atributos de tipo 2 las palabras reservadas `POSSIBILISTIC` o `FTYPE2`; y para los atributos de tipo 3 las palabras `SCALAR` o `FTYPE3`.

- **Valores por defecto:** Se pueden utilizar valores difusos cuando se define el valor por defecto de una columna. Por ejemplo, si al insertar una fila no se define el valor de la columna *altura*, su valor será la etiqueta `$ALTO`.

- **Restricciones de columna:** FSQL permite agregar restricciones o *constraints* cuando se define una columna. Esto evitará que se inserten valores inválidos. Estas restricciones son similares a las clásicas de SQL. La tabla 4.5 muestra las distintas posibilidades de restricciones y a qué tipos FSQL son aplicables.

FSQL también extiende CREATE VIEW permitiendo basar la vista en una consulta difusa.

La posibilidad de permitir tipos difusos a columnas no aplica a SQLf. Lo que sí es posible es definir una restricción del tipo `CHECK (<condición>)` donde la condición puede ser difusa. También es posible definir vistas en base a consultas difusas. La librería SQLfi posee estas dos características.

Restricción de columna	Descripción	Aplicable a tipos
`NOT NULL`	Prohíbe el valor `NULL`	Tipos 1, 2, 3 y clásicos
`NOT UNDEFINED`	Prohíbe el valor `UNDEFINED`	Tipos 2 y 3
`NOT UNKNOWN`	Prohíbe el valor `UNKNOWN`	Tipos 2 y 3
`NOT LABEL`	Prohíbe valores etiquetas de la forma `$label`	Tipos 2 y 3
`NOT CRISP`	Prohíbe valores crips	Tipo 2
`NOT TRAPEZOID`	Prohíbe funciones trapezoidales de la forma `$[a,b,c,d]`	Tipo 2
`NOT INTERVAL`	Prohíbe intervalos de la forma `$[n,m]`	Tipo 2
`NOT APPROX`	Prohíbe valores aproximados de la forma `#n`	Tipo 2
`ONLY LABEL`	La columna solo puede tomar valores etiquetas de la forma `$label`. Prohíbe los demás tipos de valores	Tipos 2 y 3
`ONLY LABEL OR UNKNOWN`	La columna solo puede tomar valores etiquetas de la forma `$label` o el valor Unknown. Prohíbe los demás tipos de valores. Es útil porque Unknown es un valor usual en la práctica	Tipos 2 y 3
`CHECK (<condición>)`	Verifica que los valores de la columna cumplan la condición	Tipos 1, 2, 3 y clásicos solo si la condición no es difusa.

Cuadro 4.5: Restricciones de columna de FSQL

4.3.7. Creación/modificación de elementos de la lógica difusa

Una vez definidas las tablas que almacenarán tanto los datos difusos como los clásicos, también es necesario definir los elementos de la lógica difusa que darán significado a los valores almacenados y a las condiciones difusas. Estos elementos son las etiquetas lingüísticas asociadas a los dominios continuos y discretos de los atributos. Cuando el atributo tiene un dominio continuo, las etiquetas tendrán una función de trapezoidal asociada. Cuando el atributo tiene un dominio discreto, las etiquetas (o valores del dominio) tendrán asociadas grados de similitud entre ellas.

Creación de etiquetas lingüísticas

La sentencia CREATE LABEL es una sentencia nueva que incorpora FSQL. Permite crear etiquetas con su respectiva función de pertenencia sobre dominios

continuos. Estas etiquetas solo se pueden asociar a tributos difusos del tipo 1 o 2. La sintaxis para crear una etiqueta lingüística es la siguiente:

```
CREATE LABEL nombre_etiqueta ON
  tabla.atributo VALUES alfa, beta, gamma, delta
```

La sentencia anterior creará una etiqueta de nombre `nombre_etiqueta` y la asociará al atributo de la tabla definida luego de la palabra `ON`. La función de pertenencia de esa etiqueta será el trapezoide $\mu_{NormalTrapezoid(alfa,beta,gamma,delta)}(x)$. Por ejemplo, FSQL asociará la etiqueta ALTO al atributo ALTURA de la tabla PERSONA. Cada vez que se utilice la palabra `nombre_etiqueta` en el contexto del atributo especificado, FSQL reemplazará la etiqueta con esa función trapezoidal.

El atributo especificado deberá ser del tipo difuso 1 y 2. La palabra ALTER puede ser utilizada para modificar una etiqueta que ya está asociada a un atributo. La palabra DROP sirve para borrar una o todas las etiquetas asociadas a un atributo. Su sintaxis es la siguiente:

```
DROP LABEL [*|nombre_etiqueta]
  ON tabla.atributo
```

Donde `*` eliminará todas las etiquetas y `nombre_etiqueta` eliminará solo la etiqueta con ese nombre.

También es posible *copiar* las etiquetas de un atributo a otro. Por ejemplo, es posible copiar las etiquetas ALTO, NORMAL y BAJO desde el atributo ALTURA de la tabla CLIENTE, al atributo ALTURA pero de la tabla EMPLEADO. La etiqueta significará lo mismo en ambos contextos. La sintaxis es la siguiente:

```
CREATE LABEL * ON tabla1.atributo1
  FROM tabla2.atributo2
```

Las etiquetas de `atributo1` se copiarán y asociarán al `atributo2`. Estas etiquetas se agregarán a las que ya tiene definido el `atributo2`. La sentencia generará un error si `atributo1` no tiene etiquetas asociadas.

SQLf nombra a las etiquetas difusas como predicados difusos y estos predicados no están relacionados a columnas ni atributos específicos como en FSQL. SQLf permite definir el predicado *alto* independientemente de una tabla o atributo. La ventaja es que la misma etiqueta podrá ser reutiliza en consultas sobre diferentes tablas sin necesitada de redefinirla. La desventaja es que no podríamos reutiliza la misma etiqueta con distintos significados: un edificio *alto* no debería valer lo mismo que una persona *alta*.

Creación de relaciones de similitud

FSQL permite comparar por similitud atributos difusos de dominio discreto, es decir del tipo 3. Pero primero hay que difinir los posibles valores o etiquetas del dominio y el grado de similitud que existe entre ellas. Estos tipos de relaciones

se han visto en 2.4.1. FSQL incorpora la sentencia CREATE NEARNESS que permite definir las posibles etiquetas de un atributo y sus relaciones de similitud. La sintaxis es la siguiente:

```
CREATE NEARNESS ON tabla.atributo
  LABEL lista_de_etiquetas
  VALUES lista_de_similitudes
```

Las `lista_de_etiquetas` son las posibles etiquetas que puede tomar el atributo separadas por comas. La `lista_de_similitudes` es una lista de grados de similitud entre 2 etiquetas. Si n es la cantidad posibles de etiquetas, la lista de similitudes tendrá $n^2/2 - n/2$ valores. Esta lista está conformada por todas las combinaciones de etiquetas sin importar el orden e ignorando la identidad. Se asume que el grado de similitud entre la misma etiqueta es 1. También se asume que la relación de similitud es simétrica, el grado de similitud entre las etiquetas a y b es igual al grado de similitud entre b y a.

Si el atributo ya tiene una relación de similitud, este puede ser modificado con la sentencia ALTER NEARNESS. Este comando borrará todas las etiquetas y las relaciones recreándolas nuevamente. La sintaxis es igual a CREATE NEARNESS. También es posible modificar solo un grado de similitud utilizando la sintaxis:

```
ALTER NEARNESS ON tabla.atributo
  CDEG(etiqueta1,etiqueta2)=grado
```

Donde `etiqueta1` y `etiqueta2` son dos etiquetas del dominio subyacente al atributo y `grado` es el nuevo grado de similitud entre las etiquetas

Es posible borrar las posibles etiquetas y las relaciones de similitud asociadas a un atributo utilizando las sentencia DROP NEARNESS. Su sintaxis es la siguiente:

```
DROP NEARNESS [etiqueta] ON tabla.atributo
```

La `etiqueta` es opcional. Si no se define, el atributo quedará sin ninguna etiqueta ni relación de similitud. Si se define, se eliminará la etiqueta de entre los posibles que puede tomar el atributo y los grados de similitud de esta etiqueta y el resto de las etiquetas.

SQLf utiliza comparadores difusos especiales para definir la comparación de similitud entre dos valores. De manera similar a los predicados, estos comparadores no están relacionados a una tabla o columna específica permitiendo ser reutilizados en consultas sobre diferentes tablas.

4.4. Selección e implementación de SQLf y FSQL

La posibilidad de almacenar datos difusos de FSQL claramente da una ventaja sobre SQLf. Esta característica permite almacenar información cuando esta no es del todo precisa u objetiva para poder consultarla y tomar decisiones en

el futuro. Aun así, el lenguaje de consultas propiamente dicho para recuperar información no está tan desarrollado como el de SQLf. Entre las características faltantes se encuentran los modificadores difusos (3.5.10), funciones de agregación difusa (3.5.9), proyección difusa (3.5.9) y las consultas agrupadas horizontalmente. La elección del lenguaje a utilizar se basará entre la posibilidad de almacenar información difusa de FSQL contra la mayor expresividad de SQLf.

La elección del lenguaje también se basará en las características de la implementación. Esto significa la elección del lenguaje del sistema cliente, el lenguaje de la librería de consultas difusas y el SGBD.

En general, las distintas implementaciones utilizan la misma solución. La arquitectura de esta solución está formada por los siguientes componentes:

- Base de datos de aplicación: La base de datos en la que almacenará la información propia de la aplicación. Esta puede ser clásica para sistemas del tipo 1 o incluir información difusa para sistemas del tipo 2.
- Meta-conocimiento difuso: Es la información que no es parte de la aplicación propia pero que la librería usará para procesar la consulta difusa y ejecutarla en la base de datos. Este conocimiento incluye las etiquetas lingüísticas, funciones de pertenencia, relaciones de similitud, comparadores, modificadores, cuantificadores difusos, etc.; y puede estar almacenado en la base de datos, ser parte de la librería y/o ser configurado en archivos.
- Gramática del lenguaje: Es la gramática que define la sintaxis del lenguaje de consultas que la librería implementará. Debido a que tanto SQLf y FSQL son extensiones de SQL, la gramática de ambas debería incluir todas las sentencias que permite SQL.
- Parser y Compilador: Es el componente encargado de procesar las sentencias del lenguaje y convertirlas al lenguaje que el SGBD ejecutará validando que estas sean sintácticamente y semánticamente correctas. Este componente utilizará el meta-conocimiento difuso para generar la sentencia SQL equivalente.
- Ejecutador: Es el encargado de tomar la sentencia SQL ya convertida y ejecutarla en SGBD.

Las características de cada uno de estos componentes se deberá tomar en cuanta al momento de la elección de la solución/librería a utilizar. La implementación del servidor FIRST-FSQL presentado por Galindo es completamente dependiente de la base de datos Oracle. Su base de datos difusa y FMB se basan en Oracle. El parser fue realizado completamente en PL/SQL. El SQL resultante de la compilación de una sentencia FSQL es también dependiente de la sintaxis especial de Oracle y utiliza procedimientos almacenados PL/SQL. Esto significa

que si deseamos utilizar FIRST y FSQL necesitaremos una base de datos Oracle. La ventaja es que la aplicación podría estar implementada en cualquier lenguaje que permita llamar procedimientos almacenados. También existe otra versión de FIRST-FSQL ([AU]) que utiliza la base de datos PostgreSql ([Pos]).

Tanto SQLf_j como SQLfi son dos librerías Java. Las tablas que utilizan para la base de datos y para almacenar el meta-conocimiento difuso son tablas relacionales clásicas. El parser y el compilar están realizados completamente en Java y el SQL resultante es SQL estándar. Ambas librerías utilizan JDBC para conectarse a las bases de datos. En teoría, sería posible utilizar cualquier base de datos que respete el estándar SQL. El lenguaje de la aplicación debería estar realizado en Java o debería ser capaz de comunicarse con una librería Java como por ejemplo JRuby ([JRu]), PHP-Java bridge ([php]), etc; o mediante alguna forma de comunicación cliente-servidor entre un servidor Java SQLf y una aplicación cliente en cualquier lenguaje. Otra implementación de SQLf es FuzzyQuery ([Fuz]). Esta librería transforma SQLf en consultas PostgreSql y se encuentra en versión beta.

Capítulo 5

Integración de consultas y sugerencias utilizando semejanza de entidades

5.1. Introducción

La programación orientada a objetos usa objetos y sus interacciones para implementar una aplicación. En la mayoría de los casos, estas aplicaciones también van a tener la necesidad de guardar y obtener información persistente. En este paradigma lo ideal es que la información guardada sean objetos teniendo la capacidad de trabajar objetos persistentes. Las dos soluciones principales para persistir objetos es la utilización una base de datos orientada a objetos o la utilización de una base de datos relacional junto con algún mapeador objeto-relacional (ORM, Object-Relational Mapping). La aplicación manipula objetos persistentes (también llamados entidades) en vez de filas y columnas de una tabla.

En los capítulos anteriores he mostrado cómo las consultas difusas son integradas a las bases de datos relacionales, pero no hemos visto cómo estas consultas pueden ser ejecutadas en un entorno orientado a objetos. En este capítulo integraré una pequeña aplicación Java con las consultas difusas de SQLf previamente analizadas. También utilizaré SQLf para extraer las características importantes de los objetos consultados y de esta manera implementar sugerencias. Integraré estas características manteniendo la orientación a objetos definida en la aplicación.

Para poder demostrar esta integración, he desarrollado una pequeña aplicación web que se dedica a la compra-venta de propiedades inmobiliarias. Permitirle al usuario llegar a ver y comprar su propiedad ideal sería uno de los objetivos fundamentales de la aplicación.

Esta aplicación la he creado con Spring ROO ([Spra]). Spring ROO es una

herramienta que permite realizar aplicaciones Web Java rápidamente generando código. Es una herramienta de línea de comando donde el programador *configura* el modelo y las características de la aplicación que será generada. La arquitectura de la aplicación cumple el estándar de una aplicación típica Spring ([Sprb]). Este estándar es una aplicación web multi-capa que utiliza Spring como contenedor y una librería ORM sobre una base de datos relacional. Aunque esta arquitectura no es un estándar de Sun (el creador de Java) para el desarrollo de aplicaciones empresariales, sí es un estándar de facto en la comunidad Java. Spring ROO puede ser integrado con muchas tecnologías permitiendo al programador personalizar la aplicación al seleccionar el motor de base de datos, la librería ORM, WEB, de servicios, seguridad, etc. Spring ROO también genera casos de pruebas unitarias y de integración con la base de datos y web.

La razón por la que utilicé ROO para generar la aplicación de inmobiliarias es que quería tener una aplicación OO estándar a la cual pudiera integrar las consultas difusas. Otro motivo secundario fue estudiar y aprender Spring ROO analizando la solución generada y evaluando su posible mejora en la productividad.

5.2. Modelo Inmobiliaria

La figura 5.1 es un modelo de clases UML (Unified Modeling Language, Lenguaje Unificado de Modelado) que define el dominio de la aplicación. He decido utilizar este lenguaje de modelado y no uno orientado a bases de datos relacionales (e.g. modelo DER, Diagrama de Entidad Relación), ya que deseo que la aplicación esté pensada e implementada en un ambiente orientados a objetos. La creación del esquema de la base de datos relacional es delegada a librería ORM. En este caso JPA([JPA]) e Hibernate([Hib]).

En este modelo simplista las propiedades serán las entidades más importantes. Una propiedad tiene un tipo (casa, departamento, etc.), un precio de venta, su estado de mantenimiento y su superficie en metros cuadrados. Una propiedad está ubicada en una dirección de un barrio, tiene un dueño que es un cliente y es administrada por un empleado. Un mismo cliente puede ser dueño de más de una propiedad y un empleado puede administrar más de una propiedad. Los objetos del modelo son entidades persistentes y cada uno de ellos tiene la capacidad de realizar las operaciones relacionadas como guardar (`persit` y `flush`), actualizar (`update`), eliminar (`remove`), etc.

Uno de los objetivos de esta aplicación es la búsqueda de propiedades utilizando lógica difusa. Cómo hemos visto en el capítulo 3, antes de poder realizar búsquedas difusas, hay que definir los elementos de lógica difusa en los que se regirá el sistema. He definido sobre el objeto `Propiedad` los siguientes elementos difusos:

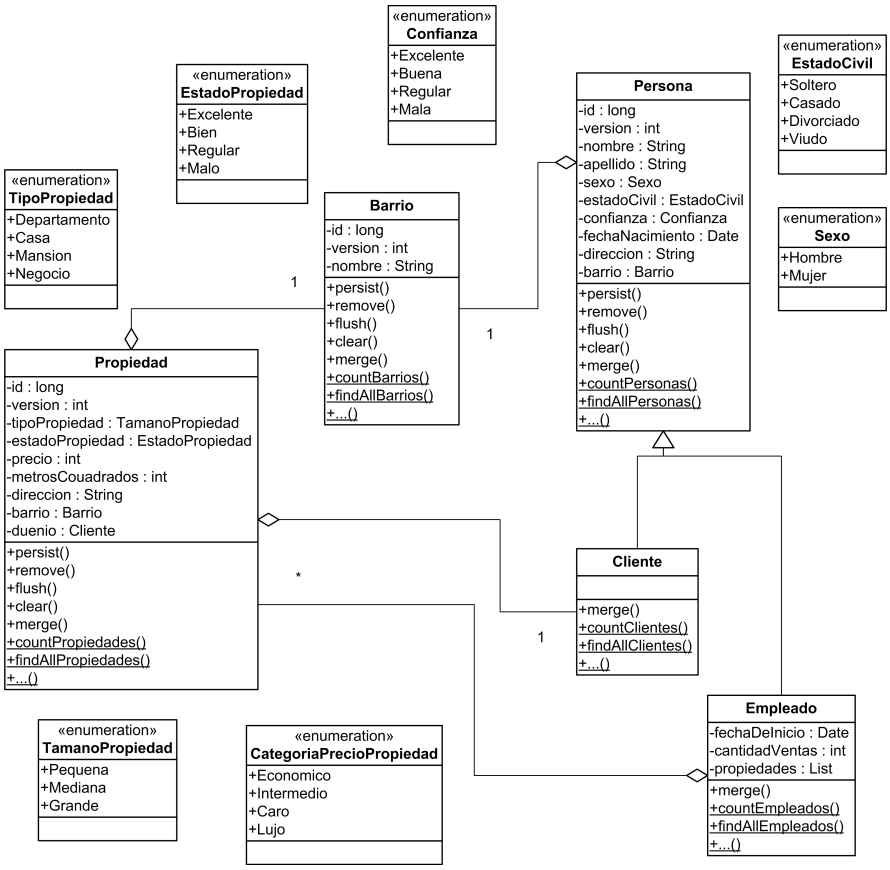

Figura 5.1: Diagrama de clases de Inmobiliaria

- Los conjuntos de similitud relacionados con los valores `departamento`, `casa`, `mansión` y `negocio` del tipo enumerativo `TipoPropiedad` del dominio discreto definido por el atributo `tipoPropiedad`.
- Los conjuntos de similitud relacionados con los valores `excelente`, `bien`, `regular` y `malo` del tipo enumerativo `EstadoPropiedad` del dominio discreto definido por el atributo `estadoPropiedad`.
- Los conjuntos difusos `económico`, `intermedio`, `caro` y `lujo` definidos por los valores del tipo enumerativo `CategoríaPrecioPropiedad` sobre el dominio continuo del atributo `precio`.
- Los conjuntos difusos `pequeña`, `mediana` y `grande` definidos por los valores del tipo enumerativo `TamañoPropiedad` sobre el dominio continuo del atributo metros cuadrados.

Más adelante en este capítulo mostraré cómo son las funciones difusas de estos conjuntos. De esta forma de definición de conjuntos difusos puedo destacar dos aspectos:

1. Los conjuntos discretos han sido implementados utilizando tipos enumerativos de Java. Los valores del enumerativo son las opciones posibles de conjuntos difusos de similitud. Estos conjuntos nos permitirán evaluar qué tan parecidos son dos valores en el contexto de un atributo de un objeto (por ejemplo los valores de `TipoPropiedad` en el contexto del atributo `tipoPropiedad` del objeto `Propiedad`).

2. Las variables y etiquetas difusas han sido descritas también utilizando tipos enumerativos de Java. Los valores del enumerativo son las etiquetas de los conjuntos difusos definidos sobre el dominio continuo en el contexto de un atributo y un objeto en particular (por ejemplo los valores de `CategoríaPrecioPropiedad` en el contexto del atributo `precio` del objeto `Propiedad`).

Una alternativa a utilizar tipos enumerativos para los casos anteriores son las cadenas de caracteres (`Strings`). La ventaja de los enumerativos es que los posibles valores están formalmente definidos permitiendo seguridad de tipos en tiempo de compilación. La ventaja de las cadenas de caracteres es que es más simple agregar posibles nuevos valores sin necesidad de recopilar la aplicación. En nuestro contexto permitiría agregar nuevos valores a los dominios discretos en el caso 1 o nuevas etiquetas lingüísticas en el caso 2. No sería dificil cambiar la implementación y utilizar cadenas de caracteres o una combinación de ambas.

5.3. Librería SQLf

La aplicación está dividida en dos: una implementación SQLf en una librería Java y la propia aplicación web que utiliza la interfaz de la librería para realizar las consultas difusas. La librería SQLf es la que utilicé para realizar las pruebas del capítulo 3. He basado la librería en SQLf_j ([MSKD]), la cual reutilicé y extendí modificando la gramática del lenguaje SQLf, el parser y el traductor de sentencias SQLf a secuencias SQL. A continuación mostraré los cambios más importantes que he realizado. Estos me servirán para lograr los objetivos de este capítulo.

5.3.1. Extensiones a la librería

Funciones difusas anónimas sobre dominio continuos

Una de las extensiones que he agregado a la gramática y el traductor de la librería es la posibilidad de utilizar predicados difusos anónimos (también llamados constantes difusas). Podemos declarar una función de pertenencia embebida en la consulta sin necesidad de definir la etiqueta lingüística (consulta 5.1).

Consulta 5.1: Casas con precio en el conjunto difuso definido con la función trapezoidal Trapezoid(75000 80000 90000 95000)

```
SELECT FROM Propiedad
  WHERE ~precio IN trapezoid(75000,80000,90000,95000)
  AND tipo = 'Casa'
```

La ventaja principal de las funciones anónimas es su versatilidad, ya que los argumentos (definición 10) de la función pueden ser ingresados en la consulta y ser cambiados en diferentes ejecuciones. Su desventaja es que la consulta se aleja del lenguaje natural. Describir un trapezoide es menos natural que utilizar una etiqueta lingüística como `caro`.

He utilizado funciones anónimas para simular los comparadores difusos. Los comparadores implementados son *menor*, *mayor*, *igual* y *alrededor*. Estos comparadores se definen en base al valor a comparar o pivote P y dos errores *e1* y *e2*. La imagen 5.2 muestra los gráficos de las funciones que los comparadores utilizan y su relación con los argumentos P, *e1* y *e2*. Los errores determinan la amplitud y la forma de los funciones trapezoidales. Mientra más grandes sean los errores, más elementos serán retornados en la comparación. En el caso extremo de que tanto *e1* y *e2* valgan 0, la comparación difusa se comportará de la misma manera que la comparación clásica $\leq$, $\geq$ e $=$.

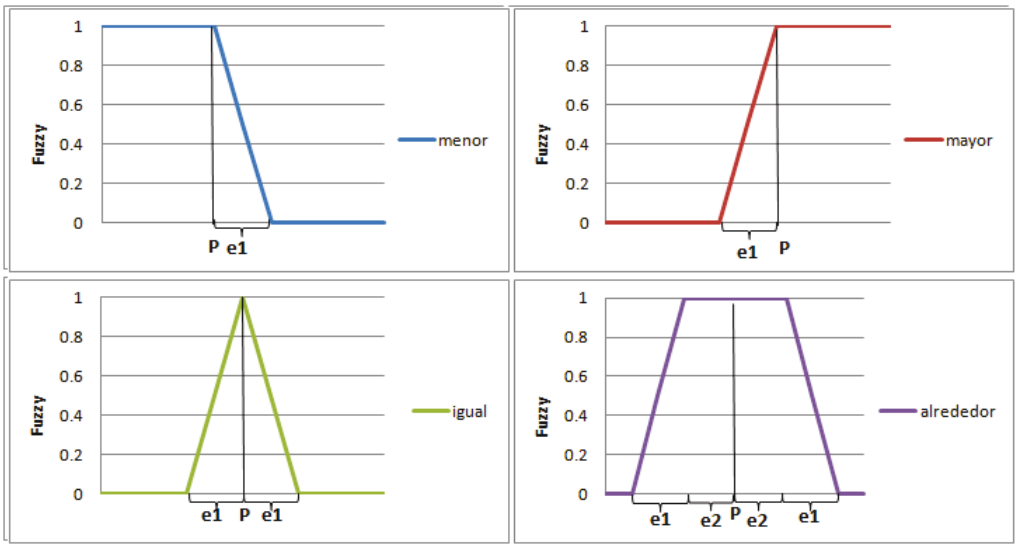

Figura 5.2: Comparadores difusos con funciones anónimas

En esta aplicación el usuario puede buscar propiedades seleccionando un precio y un comparador difuso. El resultado de esta consulta mostrará todas las propiedades (junto con el grado de pertenencia) comparando su precio de manera difusa al valor seleccionado. En este contexto los errores *e1* y *e2* fueron previamente configurados con valores 10000 y 5000 respectivamente. Si el usua-

rio desea las propiedades con precio *alrededor* de 80000 pesos, obtendrá no solo las propiedades con ese valor si no también las propiedades con precios similares basándose en la función $\mu_{NormalTrapezoid(65000,75000,85000,95000)}(x)$. Esto permitirá mostrar al usuario propiedades que están cerca del precio deseado y que podrían serles interesantes. Estas propiedades estarían excluidas si utilizáramos comparadores clásicos.

De manera similar el usuario puede buscar propiedades por tamaño especificando un valor en metros cuadrados y un comparador. En este caso los errores *e1* y *e2* valen 10 y 5 respectivamente.

Parámetros por nombre

También he extendido la librería para permitir parámetros por nombre. Los parámetros `:top`, `:media` y `:varianza` de la consulta 5.2 pueden ser reemplazos en la aplicación por valores concretos, como por ejemplo los ingresados por un usuario de la aplicación.

Consulta 5.2: Propiedades con precio en el conjunto difuso definido con la función Gaussiana Gaussian(:media :varianza)

```
SELECT TOP :top FROM Propiedad
  WHERE ~precio IN Gaussian(:media,:varianza)
```

Generadores de funciones de pertenencia

La forma de definir los tipos de funciones de pertenencia que la librería puede utilizar es por medio de objetos generadores. Estos generadores son utilizados por el parseador SQLf para generar la porción de la consulta SQL que calculará el grado de pertenencia. Por defecto, la librería viene cargada con los generadores de las funciones trapezoidal, trapezoidal a derecha, trapezoidal a izquierda y Gaussiana. La librería soporta generadores con hasta 6 argumentos. Por ejemplo el generador trapezoidal utiliza 4 argumentos (α, β, γ y ϵ) y el generador Gaussiana utiliza 2 argumentos (media y varianza). Los argumentos de los generadores definen la curva de la función de pertenencia. Estos son asignados al momento de definir una etiqueta lingüística (La etiqueta *grande* utiliza el generador trapezoidal a la derecha con $\alpha = 180$ y $\beta = 400$) o cuando se ejecuta una consulta con funciones anónimas.

No es difícil implementar otros generadores permitiendo utilizar todas las funciones de pertenencias descritas en 2.4 o nuevas funciones de pertenencias. La manera de agregar nuevos generadores es implementando la interfaz `FuzzyExpresionGenerator` o extendiendo la clase `AbstractFuzzyExpresionGenerator`.

La clase `SigmoidFuzzyExpresionGenerator` (5.3) es el generador de la función difusa $\mu_{sigmoid(a,b)}(x) = 1/(1+exp(-(x-a)*1/b))$. Sigmoide es también

llamada función logística o función S ([Wikc]), donde el argumento a define el centro de la función y el argumento b la curvatura de la S.

Consulta 5.3: Implementación del generador de la función difusa $\mu_{sigmoid(a,b)}$

```
package edu.unlp.info.inmobiliaria.sqlf;

import edu.unlp.info.sqlf.function.AbstractFuzzyExpresionGenerator;

public class SigmoidFuzzyExpresionGenerator extends
    AbstractFuzzyExpresionGenerator {

  @Override
  public String getName() {
    return "Sigmoid";
  }

  @Override
  public String getFuzzyDegree(String parameter, String... arguments) {
    String a = arguments[0];
    String b = arguments[1];
    return "1/(1+exp(-(" + parameter + "-" + a + ")*1/" + b + "))";
  }

  @Override
  public String getFuzzyDegree(String parameter, String alias) {
    String prefix = alias != "" ? alias + '.' : "";
    String a = prefix + "a";
    String b = prefix + "b";
    return getFuzzyDegree(parameter, a, b);
  }

  @Override
  protected int getArgumentCount() {
    return 2;
  }

}
```

Este generador nos permitirá utilizar la función Sigmoide al momento de definir una etiqueta lingüística o usarla como función anónima. La consulta 5.4 es un ejemplo consulta donde la etiqueta *enorme* está definida con la función Sigmoide $\mu_{enorme} = \mu_{sigmoid(250,25)}$ (figura 5.3).

Consulta 5.4: Id y metros cuadrados de propiedades enormes

```
SELECT id, metrosCuadrados FROM Propiedad
  WHERE ~metrosCuadrados is 'enorme'
```

Modificadores

Una de las extensiones es la incorporación de modificadores lingüísticos como los vistos en 3.5.10. Los modificadores pre-cargados son *muy* (`very` $\mu_A(x)^2$)), *más o menos* (`moreOrLess` $\sqrt{\mu_A(x)}$) y *extremadamente* (`extremely` $\mu_A(x)^4$)). El programador puede crear nuevos modificadores implementando la interfaz `FuzzyModifier`.

Consulta 5.5: Propiedades con estado muy similar a excelente

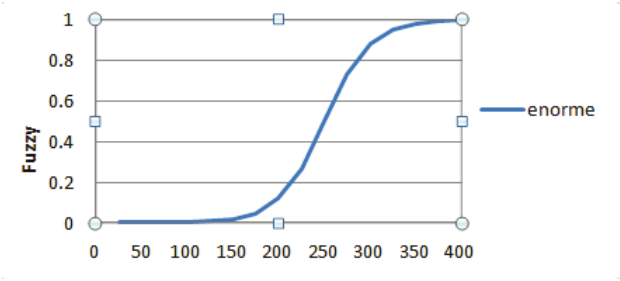

Figura 5.3: Función de pertenencia de la etiqueta *enorme*

```
SELECT TOP :top FROM Propiedad
  WHERE #estado_propiedad IS very! 'Excelente'
```

Motor de gráficos

He integrado el motor de búsqueda a la librería de gráficos JFreeChart ([jfr]) para permitir mostrar gráficamente las funciones de pertenencia y de similitud de los conjuntos difusos definidos.

API de la librería

He definido una API (Application Programming Interface, Interfaz de Programación de Aplicaciones) que le permite a las aplicaciones realizar consultas difusas de una manera simple. La API convierte de manera manual o automática las filas resultantes de consultas difusas a objetos del dominio de la aplicación. También permite crear, borrar y actualizar los predicados difusos y las relaciones de similitud en los conjuntos. Esta API tiene métodos para registrar nuevos generadores de funciones difusas y modificadores. Las operaciones principales que la librería provee son los definidos en la interfaz `SqlfOperations` que define la fachada de la librería. Para simplificar tanto la interfaz como la implementación he utilizado clases utilitarias de Spring.

Fuzzy Criteria

Por encima de `SqlfOperations` he creado un conjunto de clases *Fuzzy Criteria*. Estas clases permiten realizar consultas difusas de una manera más orientada a objetos, similar a Criteria de Hibernate [Hib]. Con estas dos variantes el programador tiene la opción de definir las consultas SQLf con sentencias y parámetros nombrados o componiendo objetos con *Fuzzy Criteria*. La ventaja tanto de Fuzzy Criteria como de Hiberante Criteria con respecto a sentencias SQLf o SQL es que permite construir consultas dinámicas simplemente construyendo y asignando objetos, en vez de concatenando cadenas caracteres. La función `buscarPropiedades` del ejemplo 5.6 muestra cómo una búsqueda puede ser

implementada con Fuzzy Criteria. Los parámetros de la función son todos opcionales, y la consulta SQLf a ejecutar dependerá de ellos. Esta función regresa una lista de objetos `Propiedad` junto con su grado de pertenencia. El ejemplo 5.7 muestra diferentes maneras de llamar a esta función y cómo serán las sentencias SQLf generadas en cada caso. La función `buscarPropiedades` podría ser implementada concatenando cadenas de caracteres y creando la sentencia SQLf correspondiente en cada caso, pero sería menos elegante.

Consulta 5.6: Buscar propiedades por tamaño y estado seleccionando umbral y cantidad máxima.

```
public List<FuzzyResult<Propiedad>> buscarPropiedades(TamanoPropiedad
    tamanoPropiedad, EstadoPropiedad estadoPropiedad, Double threshold,
    Integer top) {
  FuzzyCriteria<Propiedad> criteria = fuzzyQueryFactory.createFuzzyCriteria(
      Propiedad.class);
  if (tamanoPropiedad != null) {
    criteria.add(FuzzyRestriction.possibility("metrosCuadrados",
        tamanoPropiedad));
  }
  if (estadoPropiedad != null) {
    criteria.add(FuzzyRestriction.similarity("estadoPropiedad",
        estadoPropiedad));
  }
  if (threshold != null){
    criteria.setThreshold(threshold);
  }
  if (top != null){
    criteria.setTop(top);
  }
  return criteria.toEntityList();
}
```

Consulta 5.7: Ejemplos de invocación a la función `buscarPropiedades` y las sentencias SQLf generadas en cada caso.

```
List<FuzzyResult<Propiedad>> resultado1 = buscarPropiedades(TamanoPropiedad.
    Grande, EstadoPropiedad.Excelente, null, null);
// Select * from propiedad where ~metros_cuadrados is 'Grande' and #
    estado_propiedad is 'Excelente'

List<FuzzyResult<Propiedad>> resultado2 = buscarPropiedades(null,
    EstadoPropiedad.Malo, null, 15);
// Select top 15 * from propiedad where #estado_propiedad is 'Malo'

List<FuzzyResult<Propiedad>> resultado3 = buscarPropiedades(TamanoPropiedad.
    Mediana, EstadoPropiedad.Regular, 0.5, 10);
// Select threshold 0.5 top 10 * from propiedad where ~metros_cuadrados is '
    Mediana' and #estado_propiedad is 'Regular'
```

5.3.2. Detalles de implementación de SQLf

Los diagramas 5.4 y 5.5 muestran las clases e interfaces principales de la API y la implementación SQLf. La siguiente lista explica brevemente las responsabilidades de las mismas.

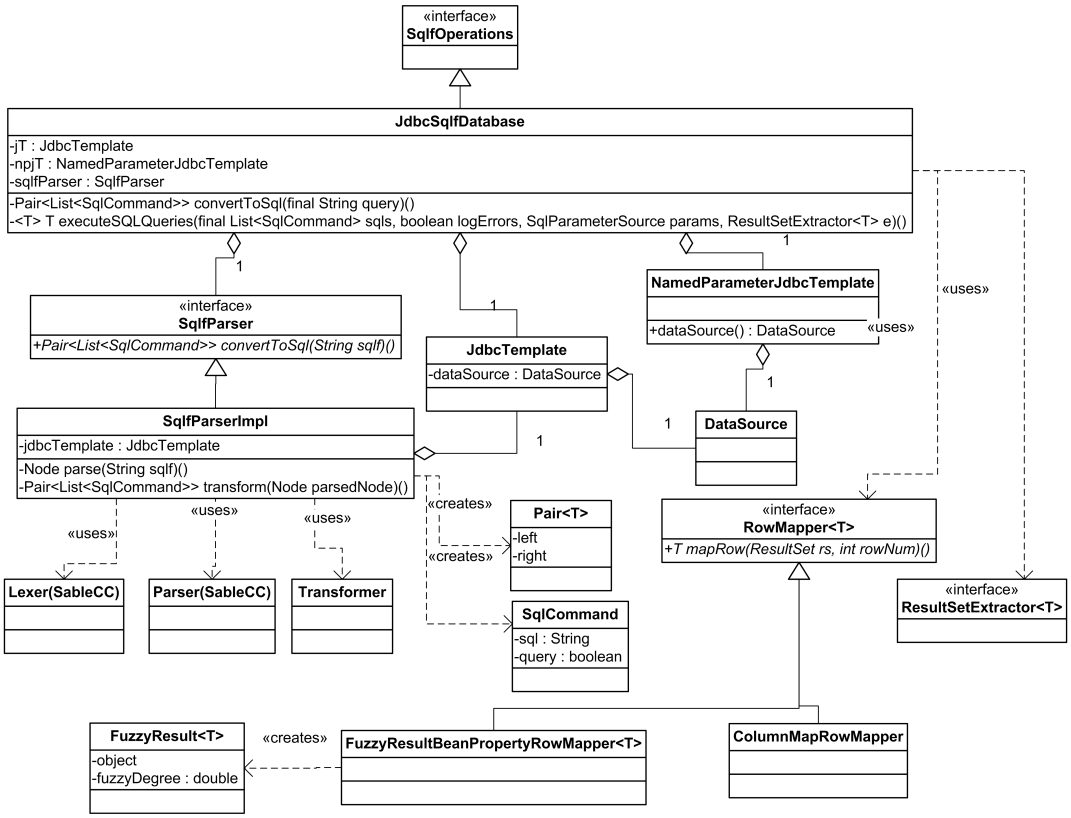

Figura 5.4: Diagrama de clases de SQLf

- `SqlfOperations`: Interfaz principal de la librería. La aplicación utilizará esta Interfaz para ejecutar consultas SQLf, configurar las funciones de pertenencias, registrar nuevos generadores y modificadores. Permite realizar consultas con parámetros y devolver tanto filas resultados cómo objetos resultados.

- `FuzzyResult`: Asocia un resultado de una consulta a un grado de pertenencia. En general, el objeto asociado es un objeto o entidad del dominio de la aplicación. Es uno de los puntos de integración orientado a objetos de la librería. La aplicación utilizará objetos del dominio y grados de pertenencia en vez de filas y columnas de tablas relacionales.

- `JdbcSqlfDatabase`: Implementación básica de `SqlfOperations`. La ejecución de una consulta SQLf está dividida en tres etapas: La conversión de la sentencia SQLf a sentencias SQL, la ejecución de las sentencias SQL y la conversión de los resultados a los objetos de la aplicación. La conversión es delegada al colaborador `SqlfParser`, la ejecución a la clase utilitaria `JdbcTemplate` y la conversión de los resultados a los mapeadores `RowMapper` y `ResultSetExtractor`.

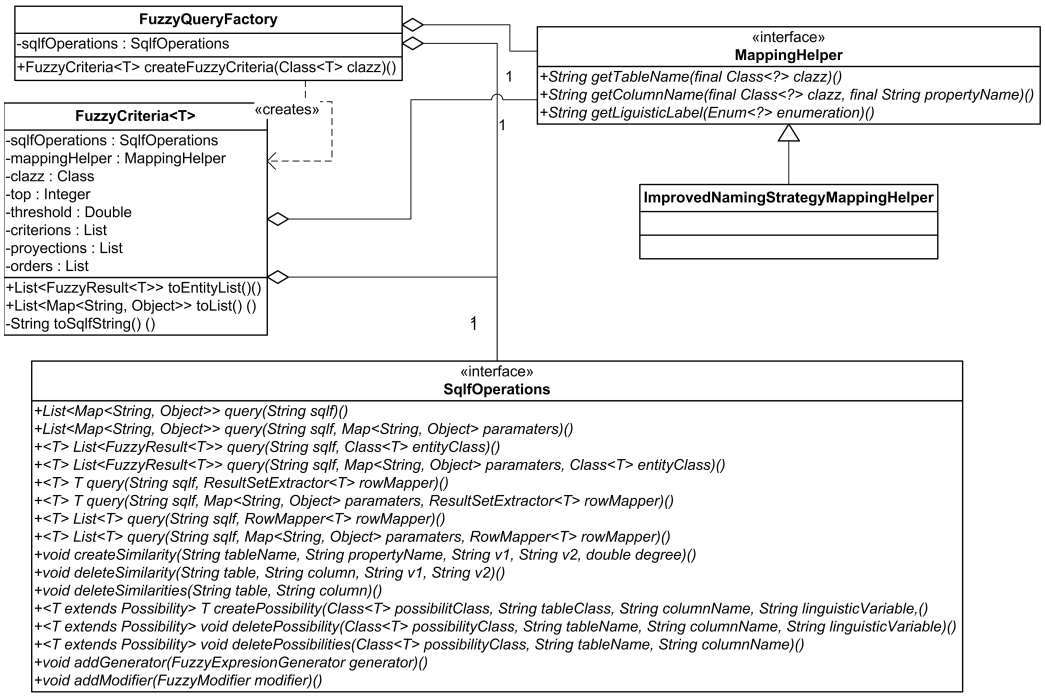

Figura 5.5: Diagrama de clases de SQLf (cont)

- `SqlfParser`: Interfaz encargada de convertir una sentencia SQLf a sentencias SQL clásicas. En esta implementación no es posible ejecutar una consulta SQLf con solamente una sentencia SQL. Por simplicidad, se crean tablas temporales en memoria que luego son borradas. Las tablas temporales almacenan los resultados parciales de las sub-consultas SQL requeridas.
- `SqlfParserImpl`: Implementación de `SqlfParser` basada en la librería SQLf_j ([MSKD]) la cual he revisado, extendido y adaptado a los objetivos de esta tesis. La conversión que realiza esta clase involucra analizar léxicamente y parsear la sentencia SQLf para luego transformar los nodos en las sentencias SQL. El parser utiliza la librería SableCC ([sab]) y obtiene de la base de datos la meta-información de las tablas involucradas en las consultas para poder crear las sentencias SQL.
- `JdbcTemplate`: Clase utilitaria de Spring que facilita la ejecución de consultas SQL.
- `NamedParameterJdbcTemplate`: Clase utilizaría de Spring que permite ejecutar consultas SQL con parámetros por nombre.
- `DataSource`: Clase JDBC que conecta Java con la base de datos.
- `RowMapper`: Interfaz genérica de Spring que convierte una fila resultante de consulta SQL a un objeto.

- `ResultSetExtractor`: Interfaz genérica de Spring que convierte resultados completos de consultas SQL a un objeto.
- `FuzzyResultBeanPropertyRowMapper`: Implementación de `RowMapper` que convierte una fila resultado a la asociación `FuzzyResult`.
- `ColumnMapRowMapper`: Implementación de `RowMapper` que convierte una fila resultado a un mapa Java [String-Objeto] donde la clave es el nombre de la columna y el valor es el valor de la columna a la fila en cuestión. Este formato de resultado es el que decidí utilizar cuando los resultados no están asociados a objetos del dominio y por ende no es posible utilizar `FuzzyResult`.
- `FuzzyCriteria`: Clase principal del conjunto de clases *Fuzzy Criteria* (5.3.1). Este objeto funciona como *builder* y ejecutador de consultas SQLf. El programador configura las consultas para luego ejecutarlas. Estas configuraciones son las proyecciones, filtros, agrupaciones, ordenes, top, threshold, etc. Un objeto `FuzzyCriteria` esta asociado a una clase de entidad del domino permitiendo consultar en su tabla en la base de datos y retornar objetos de esa clase. Es una alternativa al manejo de cadena de caracteres cuando se construyen sentencias SQL o SQLf.
- `FuzzyQueryFactory`: Clase que funciona como fábrica de objetos `FuzzyCriteria`.
- `MappingHelper`: Interfaz utilitaria encargada de convertir una clase de una entidad en un nombre de tabla y un atributo de un objeto a un nombre de columna. También convierte valores enumerativos en cadena de caracteres.
- `ImprovedNamingStrategyMappingHelper`: Implementación de `MappingHelper` que utiliza una de las estrategias de Hibernate.

5.4. Aplicación Web

La aplicación prototipo es una aplicación web que permite a los usuarios buscar y navegar las propiedades registradas. Fue inicialmente generada con Spring ROO y luego extendida para soportar la funcionalidad difusa que describiré en esta sección. Se ha agregado esta funcionalidad continuando los lineamientos de la arquitectura definida por ROO, Spring y Spring MVC.

En esta aplicación existen dos roles: los usuarios normales y los administradores. Los administradores tienen permiso de realizar todas las operaciones de la aplicación. Realizan las altas bajas y modificaciones de las entidades del sistema y entienden los elementos de la lógica difusa; tales como las funciones difusas, las relaciones de similitud, los grados de similitud, etc. Los usuarios normales solo

pueden buscar y navegar propiedades y todos los elementos de la lógica difusa están ocultos ellos. Continuando con la arquitectura, la seguridad fue implementada con Spring Security. Por simplicidad solo existen dos usuarios, uno para cada rol. Se podría agregar fácilmente usuarios y roles para los empleados y clientes pero esto escapa a la tesis. El objetivo de los roles es estudiar qué características difusas mostrar dependiendo del tipo de usuario.

El código fuente y binarios serán entregados como parte de esta tesis. Igualmente he publicado la aplicación en Internet para ser probada y evaluada fácilmente. Creé un servidor utilizando los servicios de Amazon EC2 ([EC2]) donde se puede acceder a la aplicación inmobiliaria. En caso de querer probar la aplicación instalada en EC2, por favor solicitarme la dirección URL y las credenciales. La imagen 5.6 es una captura de pantalla de la página de inicio de sesión.

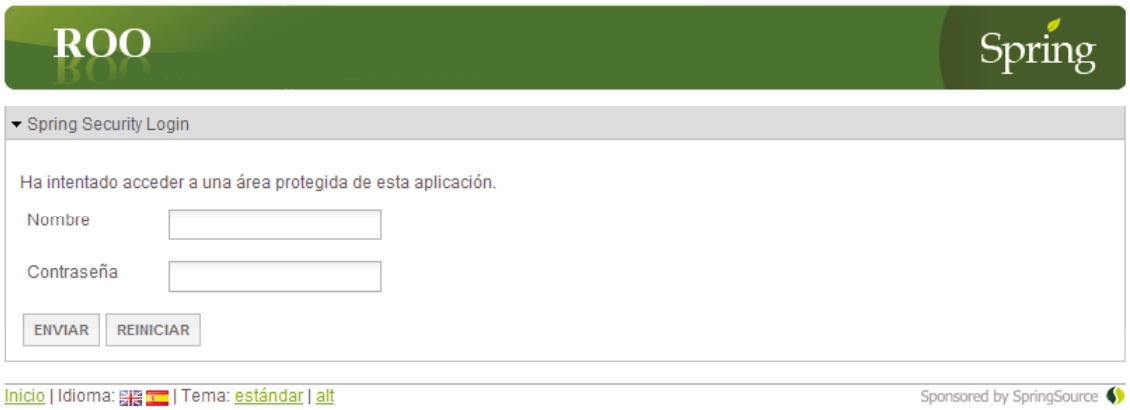

Figura 5.6: Página de inicio de sesión de la propiedad

Recomiendo entrar como administrador ya que se puede utilizar toda la funcionalidad. Los datos en el sistema fueron ingresados de manera aleatoria.

5.4.1. Consultas difusas en la aplicación Inmobiliaria

La búsqueda de propiedades es una de las funciones más importantes de la aplicación y simula la acción de un cliente al pedir un determinado producto (en este caso un inmueble) al vendedor. Tanto los usuarios normales como administradores tienen la posibilidad de buscar propiedades. La imagen 5.7 es una captura de la página de búsqueda. El usuario ingresará las características de la propiedad para luego ser buscada utilizando el motor de búsqueda difusa. Las condiciones formadas por las características están unidas por la conjunción vista en la sección 3.5.8. La propiedad resultante deberá cumplir con un cierto grado todas las condiciones. Las opciones que tiene el usuario son las siguientes:

- *Precio*: Incluye las propiedades cuyo precio esté en la función trapezoidal

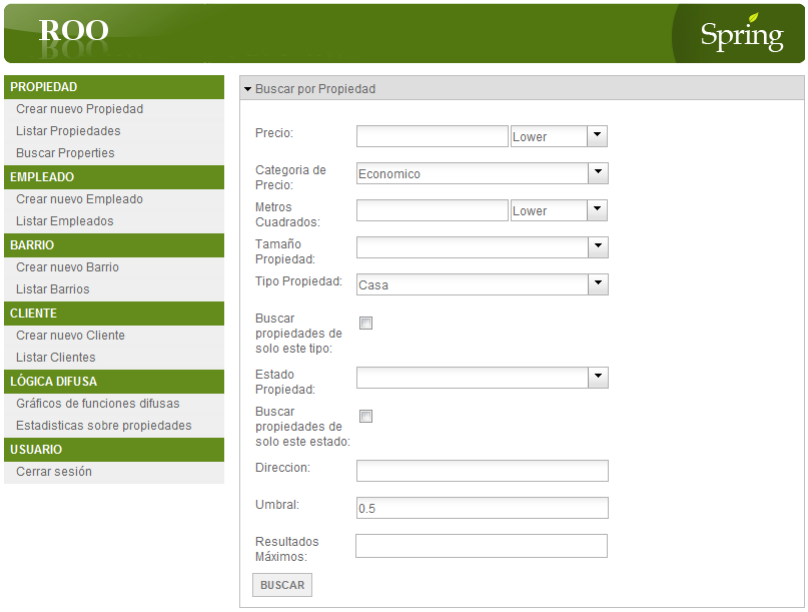

Figura 5.7: Página de búsqueda de propiedades

formada por el comparador seleccionado, el pivote ingresado y los errores $e1 = 10000$ y $e2 = 5000$.

- *Categoría de Precio*: Las propiedades que pertenezcan a la categoría de precio seleccionada (Económico, Intermedio, Medio y Caro). Cada una de estas categorías tiene su conjunto difuso y función de pertenencia asociados.

- *Metros Cuadrados*: Incluye las propiedades donde los metros cuadrados están en la función trapezoidal formada por el comparador seleccionado, el pivote ingresado y los errores $e1 = 10$ y $e2 = 5$.

- *Tamaño Propiedad*: De manera similar a *Categoría de Precio*, las propiedades que pertenezcan al tamaño seleccionado (Pequeña, Mediana y Grande). Cada uno de estos posibles tamaños tiene su conjunto difuso y función de pertenencia.

- *Tipo Propiedad*: Filtrará las propiedades por tipo (Departamento, Casa, Mansión y Negocio). Si *Buscar propiedades de solo este tipo* está seleccionado, la búsqueda retornará solo las propiedades del tipo elegido. De lo contrario, la búsqueda retornará las propiedades con tipo similar utilizando las relaciones de similitud difusa.

- *Estado Propiedad*: De manera similar a *Tipo Propiedad*, esta propiedad filtrará las propiedades por estado (Excelente, Bien, Regular y Malo). Si

Buscar propiedades de solo este estado está seleccionado, la búsqueda retornará solo las propiedades del estado elegido. De lo contrario, la búsqueda retornará las propiedades con estado similar utilizando las relaciones de similitud difusa.

- *Dirección*: Cadena de caracteres incluida en la dirección. Es una condición clásica con el comparador SQL `like`.
- *Umbral*: Número real entre 0 y 1 inclusive que es el grado de pertenencia mínimo para que una propiedad sea incluida en el resultado (*Threshold*).
- *Resultados máximos*: Número entero mayor a 0 que define la cantidad máxima de resultados (*Top*).

La implementación de las búsquedas (5.8) está realizada con *Fuzzy Criteria*. Este es un ejemplo donde construir consultas con objetos es notablemente más sencillo que la concatenación de cadena de caracteres. Notar que el resultado es una lista de objetos `FuzzyResult` donde cada uno incluye el objeto propiedad resultante más el grado de pertenencia.

Consulta 5.8: Búsquedas realizadas con Fuzzy Criteria

```
public class InmobiliariaQueryService {

  @Autowired
  private FuzzyQueryFactory fuzzyQueryFactory;

  public List<FuzzyResult<Propiedad>> buscarPropiedades(
      BuscarPropiedadesCriteria params) {
    FuzzyCriteria<Propiedad> criteria = fuzzyQueryFactory.
        createFuzzyCriteria(Propiedad.class);
    if (params.getTipoPropiedad() != null) {
      if (params.isTipoPropiedadExacto()) {
        criteria.add(FuzzyRestriction.eq("tipoPropiedad", params.
            getTipoPropiedad()));
      } else {
        criteria.add(FuzzyRestriction.similarity("tipoPropiedad", params.
            getTipoPropiedad()));
      }
    }
    if (params.getEstadoPropiedad() != null) {
      if (params.isEstadoPropiedadExacto()) {
        criteria.add(FuzzyRestriction.eq("estadoPropiedad", params.
            getEstadoPropiedad()));
      } else {
        criteria.add(FuzzyRestriction.similarity("estadoPropiedad", params.
            getEstadoPropiedad()));
      }
    }
    if (params.getPrecio() != null) {
      criteria.add(FuzzyRestriction.functionTrapezoid("precio", params.
          getPrecioFuzzyComparator(), params.getPrecio(), 10000, 5000));
    }
    if (params.getCategoriaPrecioPropiedad() != null) {
      criteria.add(FuzzyRestriction.possibility("precio", params.
          getCategoriaPrecioPropiedad()));
    }
    if (params.getMetrosCuadrados() != null) {
      criteria.add(FuzzyRestriction.functionTrapezoid("metrosCuadrados",
          params.getMetrosCuadradosFuzzyComparator(),
```

```
            params.getMetrosCuadrados(), 10, 5));
        }
        if (params.getTamanoPropiedad() != null) {
          criteria.add(FuzzyRestriction.possibility("metrosCuadrados", params.
              getTamanoPropiedad()));
        }
        if (params.getDireccionLike() != null)
          criteria.add(FuzzyRestriction.like("direccion", "%" + params.
              getDireccionLike() + "%"));

        if (params.getThreshold() != null) {
          criteria.setThreshold(params.getThreshold());
        }
        if (params.getTop() != null) {
          criteria.setTop(params.getTop());
        }
        return criteria.toEntityList();
      }

      ......
}
```

El resultado de la búsqueda es una lista de objetos propiedades más el grado de satisfacción o pertenencia a las condiciones de la consulta. La imagen 5.8 muestra una captura de pantalla del resultado de la consulta. A la derecha se encuentra el grado de pertenencia de cada resultado. Esta columna es invisible para los usuarios simples.

Figura 5.8: Resultado de una búsqueda de propiedades

Una vez elegida una propiedad, el usuario podrá navegar a la misma y ver su descripción completa. Es en este momento donde la aplicación recomendará propiedades similares.

5.4.2. Sugerencias de propiedades en la aplicación Inmobiliaria

Cuando un usuario observa una propiedad, la aplicación interpretará que es de su interés y le mostrará propiedades similares simulando un vendedor. El resul-

tado final es una lista con sugerencias en la página de una propiedad. La imagen 5.9 es un ejemplo de esta página. Además de la propiedad principal se muestran las características difusas de la propiedad y las 4 propiedades más *similares* con sus grados de pertenencia. Las características y los grados de pertenencia son visibles solamente al usuario administrador. La solución que propongo en esta tesis se divide en dos pasos: *detección de características* y *búsqueda de propiedades similares*.

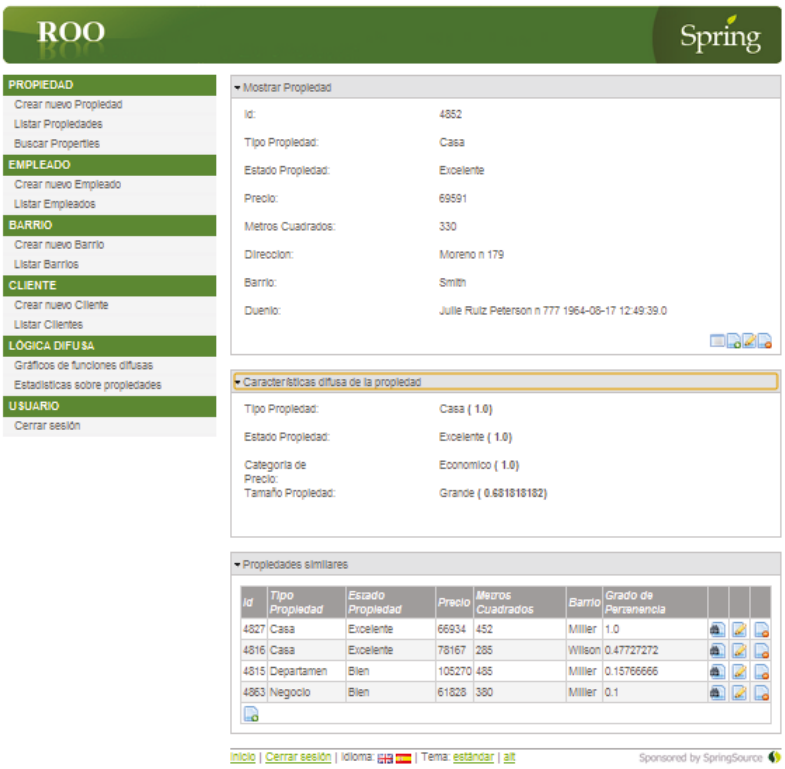

Figura 5.9: Página de una propiedad con sus características difusas y sugerencias.

Detección de características

El proceso de detección es la acción de inferir las características de un objeto. En el contexto de lógicas difusas un objeto cumple una característica con un cierto grado de pertenencia. La idea es conseguir las etiquetas lingüísticas de los conjuntos con mayor grado de pertenencia en cada una de las variables lingüísticas que aplican al objeto en cuestión. En este ejemplo las variables lingüísticas son: *tipo de propiedad* (similitud), *estado de propiedad* (similitud), *categoría de precio* y *tamaño de propiedad*. Una propiedad puede pertenecer a más de un conjunto difuso definido por cada variable. El sistema se quedará con el más representativo.

Debido a la naturaleza reflexiva de los conjuntos difusos de similitud y de los atributos con dominio discreto,es trivial conseguir la etiqueta lingüística con mayor grado de pertenencia. Sabemos que el tipo de propiedad más similar a un casa es *casa*; el grado de pertenencia del valor *casa* al conjunto difuso *similar casa* es 1 y dos tipos diferentes pueden ser similares pero con un grado de pertenencia menor a 1 a menos que se configuren estos conjuntos difusos de una manera no típica. Esta característica también aplica la similitud de valores de *estado de propiedad*.

La naturaleza descrita anteriormente no sucede en los conjuntos difusos definidos sobre dominio continuos. Los atributos con dominio continuos pueden (o no) pertenecer a conjuntos difusos. Intentaremos obtener la etiqueta lingüística donde el resultado de la función difusa aplicada al atributo continuo sea el más alto. En esta aplicación, dado el atributo entero *precio* de una propiedad, se recuperará la etiqueta (*económico*, *intermedio*, *caro* o *lujo*) y el grado de pertenencia más representativo de la variable *categoría de precio*. Este proceso se replica para la variable *tamaño de propiedad* y el atributo metros cuadrados.

Una alternativa a las etiquetas difusas sobre dominio discreto es utilizar funciones difusas anónimas. Por ejemplo, la característica precio de la propiedad es una función *alrededor* con el precio real como pivote y ciertos márgenes de error.

He implementado las consultas que retornan las características con proyecciones difusas (vistas en 3.5.9). La consulta 5.9 muestra cómo se obtiene la mejor fila en la proyección difusa sobre el dominio discreto *tipo de propiedad*. Si la relación de similaridad es reflexiva, esta consulta puede ser reemplazada por la consulta clásica 5.10. Ambas consultas retornarán el tipo de propiedad y el grado de pertenencia de 1.

Consulta 5.9: Tipo de propiedad más similar a la de la propiedad con identificador 100

```
SELECT TOP 1 #tipo_propiedad AS tipoPropiedad FROM propiedad
    WHERE id = 100
```

Consulta 5.10: Tipo de propiedad de la propiedad con identificador 100

```
SELECT tipo_propiedad AS tipoPropiedad FROM propiedad WHERE
    id = 100
```

En el caso de los conjuntos difusos sobre dominios continuos, también es posible obtener la mejor etiqueta utilizando proyecciones difusas. La consulta 5.11 retornará la categoría de precio a la que pertenece la propiedad junto con el grado de pertenencia.

Consulta 5.11: Categoría de precio de la propiedad con identificador 100

```
SELECT TOP 1 ~precio AS categoriaPrecioPropiedad FROM
    propiedad WHERE id = 100
```

Búsqueda de propiedades con características similares

Una vez obtenidas las características de una propiedad, buscar las propiedades similares es simplemente realizar una consulta difusa con esas características como parámetros. Por ejemplo, se ha detectado que la propiedad con id 100 tiene estado *regular*, es de tipo *negocio*, precio *caro* y tamaño *mediano*. La consulta 5.12 muestra la consulta que realizará la aplicación. El resultado de la consulta son las 4 mejores entidades propiedad que *cumplen* con todas las condiciones.

Consulta 5.12: Propiedades similares a la propiedad con identificador 100

```
SELECT TOP 4 * FROM propiedad WHERE #estado_propiedad IS '
    Regular' AND #tipo_propiedad IS 'Negocio' AND ~precio is
    'Caro' AND ~metros_cuadrados is 'Mediana' AND id <> 100;
```

5.4.3. Funcionalidad adicional de la aplicación

Gráficos de funciones difusas

El administrador tiene la posibilidad de ver los gráficos de las funciones de pertenencia y relaciones de similitud de los conjuntos difusos que la aplicación utiliza. En la imagen 5.10 se pueden observar los gráficos de las funciones difusas de la variable *categoría de precio* y las relaciones de similitud de la variable *tipo de propiedad*.

Consultas estadísticas difusas

Como ejemplo adicional he creado una página (imagen 5.11) que muestra el resultado de una consulta difusa agrupada. Esta consulta crea un grupo por cada combinación posible de las 4 variables difusas que están relacionadas con la entidad propiedad. De cada combinación se muestra la cantidad y la cantidad difusa (3.5.9) de las propiedades que pertenecen a ese grupo.

Figura 5.10: Gráficos de los elementos difusos

Tipo Propiedad	Estado Propiedad	Precio	Metros Cuadrados	Cantidad	Cantidad difusa
Casa	Bien	Caro	Grande	43	11.1850636
Mansion	Bien	Caro	Grande	43	10.8305181
Mansion	Regular	Caro	Grande	43	9.95
Casa	Regular	Caro	Grande	43	8.88181818
Casa	Bien	Caro	Mediana	64	8.69648907
Casa	Excelente	Caro	Grande	34	8.68506363
Mansion	Bien	Caro	Mediana	64	8.25489983
Casa	Regular	Caro	Mediana	64	8.19394891
Negocio	Bien	Caro	Grande	43	8.05
Departamen	Bien	Caro	Mediana	64	8.03983367
Negocio	Bien	Caro	Mediana	64	7.75738441
Departamen	Regular	Caro	Mediana	64	7.66960201
Departamen	Bien	Caro	Grande	43	7.4487
Mansion	Excelente	Caro	Grande	34	7.43051818
Negocio	Regular	Caro	Grande	43	7.25991818
Mansion	Regular	Caro	Mediana	64	7.06825521
Casa	Malo	Caro	Grande	26	6.74179090
Negocio	Regular	Caro	Mediana	64	6.70237198

Figura 5.11: Estadísticas de propiedades

Capítulo 6

Conclusión

Este capítulo concluye el presente informe de trabajo de grado mencionando los resultados obtenidos en el mismo y trabajos futuros.

6.1. Resultados obtenidos

Este documento y la aplicación son el resultado de mi estudio sobre cómo la lógica difusa puede solucionar el manejo de información subjetiva, en especial sobre bases de datos relacionales. El desarrollo de esta tesis me permitió ir descubriendo una línea importante de investigación que comenzó con la teoría de conjuntos difusos de Zadeh y que luego fue adaptada a los modelos de datos de los sistemas informáticos. He podido entender y reflejar en este documento cómo la información imprecisa o subjetiva puede ser encontrada en ciertos dominios y debe ser tenida en cuenta al momento de definir el modelo de datos de un sistema. Estudié y probé las soluciones que los autores han propuesto tanto a los modelos de datos teóricos e implentaciones a nivel de base de datos y de aplicación. En particular, he utilizado la librería SQLf_j como base de un sistema ejemplo. Revisé y re-implementé completamente esta librería, proporcionándole un marco teórico, agregándole funcionalidad e integrándola de un forma más OO a una arquitectura actual. El sistema ejemplo de inmobiliarias ejemplifica los conceptos de consultas y sugerencias que un sistema puede proveer utilizando información subjetiva y lenguaje natural.

El informe final del trabajo de grado comienza realizando una introducción al concepto de consultas y sugerencias que se encuentran tanto en una dinámica de mercado *real* como *online*. Analiza cómo la información almacenada en un sistema de datos informáticos es una representación del mundo real, el cual tiene una característica subjetiva y depende de la interpretación del observador. Explica

brevemente cómo la lógica difusa y las bases de datos difusas permiten solventar esta característica de la información.

El segundo capítulo explica en detalle los conceptos de la lógica difusa extendiendo las definiciones y operaciones de la lógica clásica para soportar permanencia parcial de elementos a los conjuntos mediante la utilización de grados de pertenencia.

El siguiente capítulo de este documento realiza una introducción al concepto de información subjetiva expresada en lenguaje natural. Clasifiqué los sistemas que utilizan información subjetiva entre: los que permiten realizar consultas difusas sobre modelo de datos clásicos (sistemas de tipo 1) y los que permiten almacenar información difusa (sistemas de tipo 2). Luego, el capítulo se centra en el primero de estos sistemas. He mostrado cómo la posibilidad de realizar consultas difusas sobre bases de datos relacionales clásicas es más restrictiva pero puede ser adoptada más rápidamente en sistemas existentes. Como ejemplo de lenguaje de consultas que permite estos tipos de sistemas he utilizado el lenguaje SQLf. He explicado detalladamente cada una de sus características. Estas extensiones fueron ampliamente ejemplificadas siempre haciendo referencia al marco teórico de la lógica clásica y difusa explicado previamente.

El cuarto capítulo se centró en sistemas que permiten no solo realizar consultas difusas sino almacenar información en lenguaje natural. Para poder explicar cómo los autores pudieron almacenar este tipo de información, he tenido que subir un nivel de abstracción volviendo al modelo relacional. El ejemplo principal que extiende el modelo relacional es el modelo GEFRED. Una vez explicado este modelo conceptual, he continuado con la implementación principal del mismo: FIRST y su lenguaje FSQL. Resumí la arquitectura FIRST y comparé los dos lenguajes FSQL y SQLf, y cada una de sus operaciones DDL y DML. Aunque estos lenguajes están basados en SQL, tienen características y beneficios diferentes. Al final del capítulo, generalicé las soluciones a consultas difusas vistas presentando una arquitectura común y describiendo cada uno de los componentes necesarios.

El quinto capítulo es el resultado de la integración concreta de SQLf a una aplicación web clásica. La posibilidad difusa que brinda SQLf a la aplicación permite realizar tanto consultas como sugerencias. Este capítulo muestra las extensiones de SQLf que he realizado, cómo pueden ser utilizadas las consultas y sugerencias en un entorno OO y la nueva arquitectura de la aplicación. Como conclusión final considero que este trabajo puede servir como referencia al momento de diseñar e implementar sistemas con datos subjetivos. Definitivamente todas las herramientas vistas van a ser útiles en mi carrera profesional informática.

6.2. Trabajos futuros

Como trabajo futuro se puede extender SQLf_j para que soporte todas las operaciones que están definidas en el lenguaje SQLf, como lo son las consultas agrupadas del tipo I y II, las sentencias para modificar datos (DELETE, UPDATE e INSERT) y las sentencias DDL.

También es posible ejercitar la integración de una base de datos difusa como FIRST y FSQL (sistemas tipo 2) de manera similar a la integración realizada en el quinto capítulo con SQLf.

Otra línea de trabajo futuro sería analizar en profundidad la derivación de consultas SQLF o FSQL a SQL, estudiar el rendimiento de la ejecución de consultas difusas e intentar mejorarlo.

En el quinto capítulo, la integración fue realizada entre un aplicación programada en un lenguaje orientado a objetos como Java con una base de datos relacional usando la técnica de mapeo objeto-relacional (ORM). Como variante se podrían investigar bases de datos orientadas a objetos con información difusa ([Ma04]) y de esta manera soportar la persistencia de objetos de una manera nativa.

Bibliografía

[AU] Alejandro Sepúlveda Angélica Urrutia, José Galindo. Implementación de una base de datos difusas con first-2 y postgresql. [cited at p. 81]

[BBM92] J. Yao B. Bouchon-Meunier. Linguistic modifiers and imprecise categories. 1992. [cited at p. 28]

[Bos95] O. Bosc, P. Pivert. Sqlf: A relational database language for fuzzy querying. *IEEE Transactions on Fuzzy Systems*, 1995. [cited at p. 6, 31, 35, 42, 56, 58]

[BP84] Bill P. Buckles and Frederick E. Petry. Extending the fuzzy database with fuzzy numbers. *Inf. Sci.*, 34(2):145–155, 1984. [cited at p. 66]

[BP05] Gloria Bordogna and Giuseppe Psaila. Extending sql with customizable soft selection conditions. In *SAC*, pages 1107–1111, 2005. [cited at p. 6]

[CMC05] Vidaurreta C., Donajger M, and Smith C. *Diseño de Agentes Inteligentes Difusos para Marketing Web*. Mayo 2005. [cited at p. 10]

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970. [cited at p. 34]

[EC2] Amazon ec2: `http://aws.amazon.com/ec2`. [cited at p. 95]

[Fuz] Fuzzyquery: `http://code.google.com/p/fuzzyquery/`. [cited at p. 81]

[Góm99] José Galindo Gómez. *Tratamiento de la Imprecisión en Base de Datos Relacionales: Extensión del modelo y adaptación de los SGBD actuales*. PhD thesis, Dpto. de Ciencias de la Computación e Iteligencia Artificial. Universidad de Granada, 1999. [cited at p. 12, 68, 70, 72]

[GMPC98] José Galindo, Juan Miguel Medina, Olga Pons, and Juan C. Cubero. A server for fuzzy sql queries. In *FQAS*, pages 164–174, 1998. [cited at p. 6, 65]

[GT07] Marlene Goncalves and Leonid Tineo. A web tool for web document and data source selection with sqlfi. In *ICEIS (1)*, pages 449–452, 2007. [cited at p. 36]

[GT08] Marlene Goncalves and Leonid Tineo. Sqlfi y sus aplicaciones. *RASI*, 5(2):33–40, 2008. [cited at p. 36]

[Hib] Hibernate: `http://www.hibernate.org`. [cited at p. 84, 90]

[jfr] Jfreechart: http://www.jfree.org/jfreechart. [cited at p. 90]

[JPA] Jpa: http://en.wikipedia.org/wiki/Java_Persistence_API. [cited at p. 84]

[JRu] Jruby: http://jruby.org. [cited at p. 81]

[KD] Maciej Hapke Krzysztof Dembczynski, Dominik Przybyl. Flexible query with fuzzy projection. [cited at p. 51]

[Ma04] Zongmin Ma. *Advances in Fuzzy Object-Oriented Databases: Modeling and Applications*. Idea Group Inc (IGI), 2004. [cited at p. 105]

[MPM94] Juan Miguel Medina, Olga Pons, and María Amparo Vila Miranda. Gefred: A generalized model of fuzzy relational databases. *Inf. Sci.*, 76(1-2):87–109, 1994. [cited at p. 7, 65, 66]

[MSKD] M. Sc. Piotr Kalinowski M. Sc. Krzysztof Dembczynski, M. Sc. Dominik Przybyl. Sqlf_j - fuzzy querying system in java. [cited at p. 36, 86, 93]

[Pet96] P. E. Petry. *Fuzzy Databases: Principles and Applications*. Kluwer Academic Publishers, 1996. [cited at p. 23]

[php] Php-java bridge: http://php-java-bridge.sourceforge.net. [cited at p. 81]

[Pos] Postgresql: http://www.postgresql.org. [cited at p. 81]

[RY96] Dan Rasmussen and Ronald R. Yager. Summarysql - a flexible fuzzy query language. In *FQAS*, pages 1–18, 1996. [cited at p. 6]

[sab] Sablecc: http://www.sablecc.org. [cited at p. 93]

[Spra] Spring roo: http://http://www.springsource.org/spring-roo. [cited at p. 83]

[Sprb] Spring: http://http://www.springsource.org. [cited at p. 84]

[Tina] Leonid Tineo. Sqlf horizontal fuzzy quantified query processing. [cited at p. 58, 75]

[Tinb] Leonid Tineo. Una contribución a la interrogación flexible de base de datos difusas: Evaluación de consultas difusas cuantificadas. [cited at p. 58, 60, 63]

[UF94] Motohide Umano and Satoru Fukami. Fuzzy relational algebra for possibility-distribution-fuzzy-relational model of fuzzy data. *J. Intell. Inf. Syst.*, 3(1):7–27, 1994. [cited at p. 66]

[Wika] Wikipedia. Wikipedia jdbc: http://en.wikipedia.org/wiki/Java_Database_Connectivity. [cited at p. 70]

[Wikb] Wikipedia. Wikipedia procedimiento almacenado: http://es.wikipedia.org/wiki/Procedimiento_almacenado. [cited at p. 70]

[Wikc] Wikipedia. Wikipedia sigmoid function: http://en.wikipedia.org/wiki/Sigmoid_function. [cited at p. 89]

[Zad65] L.A. Zadeh. Fuzzy sets, information and control. 1965. [cited at p. 6, 9, 10, 25]

[Zad75] L.A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning. 1975. [cited at p. 10, 20]

[Zad83] L.A. Zadeh. A computational approach to fuzzy quantifiers in natural languages. 1983. [cited at p. 30, 31]

[Zim85] H. J. Zimmermann. *Fuzzy Set Theory and its Applications*. Boston, Kluwer, 1985. [cited at p. 10]