MODELO DIFUSO DE BASES DE DATOS OBJETO-RELACIONAL: PROPUESTA DE IMPLEMENTACIÓN EN SOFTWARE LIBRE

TESIS DOCTORAL

Luis Cuevas Rodríguez

DIRECTORES: Nicolás Marín Ruiz María Amparo Vila Miranda

Dpto. de Ciencias de la Computación e Inteligencia Artificial.
Universidad de Granada

17 de febrero de 2009

Editor: Editorial de la Universidad de Granada
Autor: Luis Cuevas Rodríguez
D.L.: GR. 2049-2009
ISBN: 978-84-692-2251-5

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

La sociedad del siglo XXI ha incorporado a su andar habitual el ordenador como medio de trabajo o entretenimiento. Hoy se puede encontrar un ordenador en cualquier hogar o cualquier negocio cumpliendo las más disímiles tareas. No es difícil contactar negocios que implementan soluciones informáticas para sus gestiones diarias. La implementación de estas soluciones se ha convertido en una necesidad para mantenerse en un mercado cada vez más exigente y en constante evolución.

Las bases de datos: un componente fundamental de las aplicaciones

Se encuentran aplicaciones informáticas en todas las ramas de las ciencias, en las telecomunicación, la gestión, el entretenimiento, hasta las que soportan el trabajo sobre uno de los medios más utilizados en la actualidad, la Internet. Todas estas aplicaciones, en su gran mayoría, incorporan un elemento fundamental, las bases de datos. Las bases de datos son utilizadas en la actualidad para almacenar y administrar datos de todo tipo, relacionados con una actividad o el problema en particular.

Las bases de datos no han estado ajenas a la evolución de la ciencia y la tecnología en los últimos años. El surgimiento de nuevas tecnologías de hardware junto con la necesidad de almacenar nuevos tipos de datos ha incrementado notablemente las investigaciones para el desarrollo de nuevos sistemas de bases de datos.

En la década de los 70 surge la *primera generación de bases de datos*, implementadas por medio de los sistemas de bases de datos jerárquicos y de redes. Estos fueron los primeros sistemas en ofrecer funciones para un *SGBD* (*Sistema Gestor de Bases de Datos*) en un único sistema con definición de datos y lenguajes de manipulación de colecciones de registros.

Durante los 80 se desarrolla lo que se conoce como *segunda generación de sistemas de bases de datos*. Esta segunda generación empieza a consolidarse a partir de los trabajos de E. F. Cood [Cod90b] que plantean un modelo muy sólido para la representación y manipulación de bases de datos, el Modelo Relacional. Este modelo garantiza un sustancial grado de independencia en los datos e incorpora formalismos tanto procedurales como no procedurales para la manipulación de dichos datos. En nuestros días el modelo relacional es suficiente para soportar un gran porcentaje de las aplicaciones de bases de datos, en su gran mayoría relacionadas con procesos de gestión empresariales. Sin embargo, han surgido nuevas clases de problemas para los que el Modelo Relacional. encuentra dificultades en su representación. Entre estos tipos de problemas están: los sistemas *CAD (Computer Aided Design)*, herramientas *CASE (Computer Aided Software Engineering)*, aplicaciones de hipertexto, aplicaciones multimedia, etc.

Además de la necesidad de desarrollar aplicaciones de bases de datos para estos nuevos tipos de problemas, el modelo relacional no es el más adecuado para la representación de un modelo conceptual, debido a sus limitaciones al tratar de capturar parte de la semántica asociada a los datos. Estas dos razones son el motivo que llevó a los investigadores en el área a pensar en una nueva generación de *DBMS*.

La *tercera generación de bases de datos* viene a garantizar un marco de desarrollo para aplicaciones tratando de responder a los inconvenientes del Modelo Relacional para enfrentar determinados tipos de problemas. Las principales características de esta nueva generación se pueden resumir en: soporte de estructuras de datos no convencionales, especificación de reglas para datos, registros y colecciones, soporte avanzado para un lenguaje de consulta no-procedural.

Dos corrientes fundamentales se han desarrollado en esta tercera generación. La primera corriente con los *sistemas de bases de datos orientadas a objetos, SGBDOO*[Atk90] y una segunda corriente en los *sistemas de bases de datos objeto-relacionales, SGBDOR* [Sto89, Sto99].

Las dos corrientes citadas anteriormente nacen fruto de considerar la orientación a objetos como el paradigma a seguir a la hora de estructurar los datos. Nadie duda hoy día de la eficacia y la potencia de modelado que posee este paradigma que lo ha hecho triunfar de forma innegable en el mundo de la programación. Entre los lenguajes más utilizados que soportan este paradigma están *Java, C++, C#*. Acercando el modelo de datos de la base de datos a este paradigma se obtiene también un beneficio adicional: los datos en la base de datos tienen una estructura muy similar a la que utilizarán las aplicaciones de gestión para

manipularlos (limitando por tanto el problema de falta de correspondencia – *impedance mismatch* – que tantas líneas de código genera a la hora de desarrollar tales aplicaciones).

Los *SGBDOO* se basan en el paradigma de la orientación a objetos aplicado al modelado y programación de sistemas de bases de datos. Por consiguiente logran un soporte para objetos complejos y tienen la capacidad de poder ser extensibles mediante reutilización, adaptándose mucho más al modelo semántico de los datos. Entre las principales características de estos tipos de sistemas podemos señalar [Cod90b]: soporte para objetos complejos, identidad de objetos, encapsulación de objetos, soporte para tipos y clases, herencia de tipos y clases, polimorfismo de funciones, extensibilidad de tipos, persistencia, soporte de concurrencia y recuperación de información, y facilidad de consulta.

Estos sistemas, si bien han tenido un desarrollo en los últimos años y han resuelto los problemas planteados para la tercera generación, han encontrado dificultades desde el punto de vista del rendimiento, lo que les ha llevado a no poder representar más que una pequeña parte del mercado comercial, teniendo un uso relativamente reducido.

Según los investigadores más experimentados en el área, la tercera generación debía soportar muchas de las características de los *SGBDOO* pero sin renunciar a las ventajas del *Modelo Relacional* [Dat01]. De forma que se produzca un acercamiento del modelo relacional al modelado semántico de los datos, tratando de no perder la esencia y la eficacia que ha demostrado este modelo.

Así surgen los *sistemas de bases de datos objeto-relacionales* que combinan algunas de las características de la orientación a objetos y las características del modelo relacional, soportando un dialecto de SQL (el *SQL3* [EM99]) para sus consultas.

El mundo de las bases de datos actualmente puede dividirse en cuatro categorías [Sto99] como se muestra en la figura 1.1.

En el cuadrante inferior izquierdo de la figura (1.1) se encuentran aquellas aplicaciones que procesan datos simples y no tienen requerimientos de consulta de cara al usuario. Ejemplos de este tipo de aplicaciones son los procesadores de textos, que pueden utilizar el sistema operativo subyacente para adquirir algunas cualidades de persistencia correspondientes a un *SGBD*.

En el cuadrante inferior derecho se encuentran aquellas aplicaciones que procesan datos complejos, pero tampoco tienen grandes requerimientos de consulta, ejemplo de estas son los sistemas de ayuda al diseño y para este tipo de aplicaciones es ideal un sistema de bases de datos orientado a objetos *clásico*, donde se combinen buenas facilidades de representación y tratamiento con alguna capacidad de procesamiento.

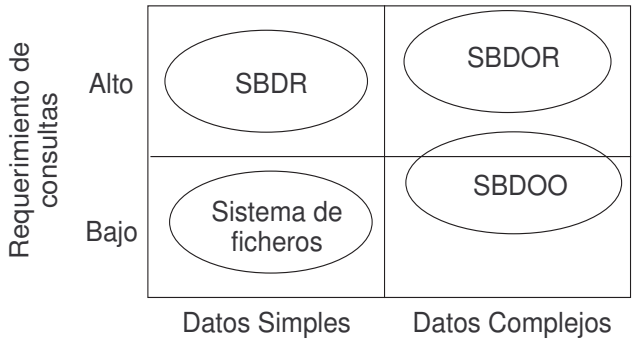


Figura 1.1: Matriz de clasificación de los *SGBD*, [Sto99]

En el cuadrante superior izquierdo se encuentran aquellas aplicaciones que procesan datos sencillos pero que tienen requerimientos de consulta complejos, y para este tipo de aplicaciones que son las clásicas de bases de datos, lo ideal son los sistemas relacionales.

Por último en el cuadrante superior derecho se encuentran las aplicaciones que procesan datos complejos e implican consultas complejas. Estas aplicaciones son las que corresponden a tratamientos avanzados de bases datos y que deben ser soportadas por medio de sistemas de bases de datos objeto-relacionales (ver Figura 1.1).

Estos últimos sistemas están teniendo gran desarrollo en la actualidad, y el hecho de que algunas de las casas comerciales líderes en el mercado estén derivando hacia ellos, hace pensar que el futuro de las bases de datos está ahí.

Los sistemas de bases de datos relacionales y objeto – relacionales están formados por una arquitectura de tres niveles: interno, conceptual y externo. De estos tres, el nivel conceptual es el más relevante para el desarrollador de un sistema de bases de datos. En el nivel conceptual se organizan y describen las entidades, los atributos y las relaciones que formarán el sistema. Los niveles conceptuales de los *SGBDR* y los *SGBDOR* tienen muchas diferencias, producto de las características que cada uno soporta [Cod90b, Sto99]. Sin embargo estos modelos están pensados para almacenar tipos de datos precisos.

Los datos que tengan asociado algún tipo de imperfección como puede ser incertidumbre al no poder evaluar la certeza o falsedad de un dato, imprecisión al no poderse especificar un valor exacto para un atributo o vaguedad cuando los conjuntos a los que pertenece un valor no están claramente definidos, no pueden representarse en los gestores de bases de datos antes expuestos. Ninguno de los dos sistemas está diseñado y pensado para soportar el almacenamiento y manipulación de datos e información con algún tipo de imperfección.

Manejo de datos imperfectos

Las recientes aplicaciones en sistemas de información requieren de capacidades para manipular datos e información imperfecta. Se considera como datos imperfectos aquellos que encierran alguna imprecisión o incertidumbre en la definición de su valor. Por ejemplo, el tiempo que demora un atleta en recorrer una distancia de 400 m lisos. Inicialmente, se podría pensar que el atributo que almacena ese valor es de tipo numérico, e.j. 45 segundos. Sin embargo, en algunas ocasiones, puede que no se disponga de un cronómetro para obtener el valor exacto de la velocidad. Sin embargo, por observación se puede definir un criterio que en este caso pudiera ser *rápido, promedio, lento.* Si el atributo está definido como un número: ¿cómo se almacena este nuevo valor?, ¿cómo se puede operar con los dos tipos de valores a la vez?

El problema de soportar el almacenamiento y consulta de datos imperfectos en una base de datos viene siendo estudiado desde hace bastante tiempo.

Los primeros trabajos en este sentido partieron de la utilización del valor *nulo* para representar la existencia de imperfección en un atributo. La semántica de este valor nulo es que cualquier valor del dominio correspondiente al dominio del atributo es posible, representando que el valor existe pero es desconocido. A partir de esto fue necesario realizar trasformaciones en el modelo relacional para representar estos valores nulos y se extendió el calculo relacional sobre una lógica basada en tres valores: verdadero, falso y quizás [AM97]. En este tipo de acercamiento se enmarcan los trabajos presentados por Codd [Cod79, Cod86, Cod87, Cod90a]. En trabajos posteriores el valor desconocido, nulo, fue limitado a un subconjunto de posibles valores, valores disyuntivos. El trabajo con estos tipos de valores puede consultarse en [AM97]. El uso de valores nulos para tratar la imperfección es retomado por lo trabajos de Guy de Tré y otros [TCP08].

Otras propuestas para tratar la imperfección de la información en la bases de datos abarca la utilización de la teoría de las probabilidades. En este caso al valor de un atributo se asocia una distribución de probabilidad [AM97]. Algunos de los trabajos en este sentido pueden encontrarse en [CP87, BGMP92].

A pesar de estos acercamientos para tratar la imperfección en las bases de datos, buena parte de las principales propuestas para enfrentar este problema se han basado en la Teoría de Subconjuntos Difusos y en la Lógica Difusa aplicada a bases de datos.

Los principales esfuerzos se han realizado en el Modelo Relacional de bases de datos, tanto en la definición de modelos relacionales difusos, como la posibilidad de escribir consultas flexibles [PT84, MPM94, ZK96, Gal99, CV95, BP82, **?**]. En el modelo de base de

datos orientado a objetos existen muchos trabajos de investigadores para extender estos modelos de forma que soporten trabajar con la imperfección [BLP94, RDP98, GBP91, TKS91, DPR91, Mar01, Cal97, TC03, GCV93] .

Como puede notarse el trabajo con datos difusos ha sido implementado en las dos últimas generaciones de bases de datos. Se pueden encontrar en la literatura varios trabajos de este tipo, tanto para el modelo relacional como para el orientado a objeto. Sin embargo, en los sistemas objeto - relacionales es ahora cuando se está empezando a trabajar en este sentido.

Software libre: una alternativa para el desarrollo

Por otra parte, nadie duda de la importancia y el interés que suscita en el mundo de las tecnologías de la información lo que se denomina el *Software Libre*. *Software Libre* se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. Cuatro son las libertades fundamentales que se asocian a los usuarios de este tipo de software:

1. La libertad de usar el programa con cualquier propósito legal.

2. La libertad de estudiar cómo funciona el programa, y adaptarlo a las necesidades propias.

3. La libertad de distribuir copias.

4. La libertad de mejorar el programa y hacer públicas las mejoras a los demás.

Como se ve, este tipo de software está a disposición de cualquier usuario para ser utilizado. Esto conlleva un conjunto de ventajas económicas, fundamentalmente las relacionadas con la ausencia de pago por licencias, aunque al mismo tiempo conlleva desventajas con respecto a la asistencia técnica.

Desde el punto de vista tecnológico, la utilización de software libre trae varias ventajas. Algunas de estas ventajas son: software muy eficiente, muchos programadores trabajan diariamente en ellos mejorándolos y optimizándolos; la robustez de los sistemas es alta pues los desarrolladores los pueden ir mejorando independientemente del creador o la compañía que lo produjo; tienden a ser muy diversos. Estos sistemas son desarrollados por personas con distintos tipos de necesidades, por lo que el software está adaptado a una mayor cantidad de problemas.

Además de estas ventajas se puede señalar el control que tiene el usuario sobre su sistema, elemento este muy importante en aplicaciones de misión critica. La reutilización del conocimiento es otro punto positivo, permitiendo que el desarrollo de un software no tenga necesariamente que comenzar de cero. Finalmente la posibilidad de adaptar el software a las propias necesidades, el aprendizaje de técnicas de programación y el reconocimiento de colegas que trabajan una misma área son factores que también hablan a favor del software libre.

El uso o no de este tipo de software responde a particularidades de los desarrolladores y usuarios, aunque en ciertos ambientes donde la solvencia económica no permite acceder al software comercial, no existe otra alternativa legal. El desarrollo científico y tecnológico de ambientes con limitaciones económicas puede encontrar un apoyo fundamental en el uso de este tipo de programas.

Sin lugar a dudas, el ejemplo de software libre que mayor desarrollo ha alcanzado en los últimos tiempos es el sistema operativo *Linux*. *Linux*, en sus numerosas distribuciones, es un sistema operativo tipo *Unix,* con características de multi-usuario, multi-tarea, memoria virtual, librerías compartidas, protocolo *TCP/IP*, etc. Su distribución es gratuita para todo el mundo sobre una licencia *LGPL (Lesser General Public License)*. Sobre este sistema operativo, investigadores del departamento de *Ciencias de la Computación de la Universidad de Berkeley* han desarrollado un sistema de gestión de bases de datos objeto - relacional conocido como *PostgreSQL* (www.postgresql.org) [DD03]. A partir de la versión 7.4 este gestor de bases de datos fue desarrollado también para el sistema operativo Windows.

PostgreSQL actualmente se distribuye de forma gratuita. Este gestor fue pionero en muchos de los conceptos que hoy se manejan para las bases de datos objeto-relacional. Como sistema de bases de datos relacional soporta un modelo de datos que consiste de una colección de relaciones cada una representada por atributos definidos sobre tipos específicos. Por la parte de objetos ofrece un poder adicional a lo que una base de datos relacional común puede ofrecer. *PostgreSQL* incorpora, entre otros, los siguientes conceptos:

1. Definición de tipos de datos creados por el usuario.
2. Declaración de funciones definidas por el usuario.
3. Soporte para trabajar con objetos complejos.
4. Herencia entre tablas.

5. Definición de reglas.

Por lo tanto, *PostgreSQL* es un sistema de bases de datos relacional que incorpora algunas de las funcionalidades de la orientación a objetos. Sin embargo, este gestor de base de datos con buena capacidad de modelado no da soporte para la representación y manipulación de datos imperfectos. Se debe señalar que trabajos publicados recientemente [**?**] mencionan la adaptación del lenguaje de consultas difusas *FSQL* a *PostgreSQL*. Sin embargo, es conveniente continuar investigando para potenciar en gestores de bases de datos distribuidos de forma gratuita de un soporte para el trabajo con información imperfecta.

1.1. Objetivos del trabajo de investigación

En la literatura pueden encontrarse varias propuestas para la representación de la información difusa en los sistemas de bases de datos, tanto en sistemas relacionales como orientados a objetos (ver referencias en este mismo capítulo). Sin embargo, de una implementación de bases de datos difusas utilizando gestores de bases de datos con características objeto - relacional sólo existe la referencia publicada por Cubero y otros, [CJ04]. En este caso la implementación se propone sobre un Oracle, un gestor de bases de datos propietario.

Por otro lado las implementaciones desarrolladas de sistemas de bases de datos difusas orientadas a objetos no están disponibles para su distribución y por lo tanto para la utilización por parte de usuarios de otras disciplinas. En ninguno de los casos existe una distribución para el soporte de bases de datos difusas en un ambiente de software libre.

A partir de los elementos expuestos, se define como objetivo de esta investigación el siguiente:

Desarrollo de un modelo de base de datos difusa orientada a objetos, y su implementación en un gestor de bases de datos distribuido como software libre y con características objeto-relacionales.

Para el cumplimiento de este objetivo es necesario tener en cuenta los siguiente objetivos específicos:

1. Realizar un estudio de los antecedentes de los modelos de bases de datos, la lógica difusa y las bases de datos difusas. Este objetivo permite evaluar y conocer el estado actual del problema a resolver.

2. Desarrollar un modelo teórico de bases de datos difusas orientadas a objetos a partir de incorporar nuevas extensiones a modelos existentes.

3. Diseño e implementación de una extensión para *PostgreSQL* que permita el trabajo con objetos descritos por atributos difusos.

4. Diseño e implementación de una interfaz de usuario que permita el acceso a las extensiones difusas incorporadas a *PostgreSQL*.

La culminación de esta investigación es un aporte a los modelos de bases de datos existentes y a *PostgreSQL*. *PostgreSQL* dispondrá de una contribución que da soporte para el trabajo con información difusa, con lo cual aumentan las posibilidades de este gestor de bases de datos. Los resultados serán distribuidos y publicados en Internet para el libre uso de programadores y personal interesado en modelar y utilizar bases de datos difusas. Esta contribución a *PostgreSQL* será distribuida y presentada en los foros de discusión pretendiendo, en un futuro, la distribución junto con el paquete de instalación de *PostgreSQL*.

1.2. Contenidos de la memoria.

Esta memoria pretende reflejar los principales resultados obtenidos en esta investigación y en tal sentido se estructura de la siguiente forma:

- Segundo capítulo: Se describen los antecedentes teóricos de los modelos de bases de datos existentes y de los principales conceptos de lógica difusa que serán utilizados durante la investigación.

- Tercer capítulo: Se enmarcan los antecedentes existente en la representación de información imperfecta por medio de la lógica difusa en bases de datos, haciendo énfasis en las bases de datos orientadas a objetos.

- Cuarto capítulo: Refleja el marco teórico del modelo de base de datos difusa orientado a objetos propuesto en esta investigación. Los elementos tomados de otras propuestas y las extensiones realizadas durante la investigación.

- Quinto capítulo: Se describe la implementación realizada en *PostgreSQL* del modelo teórico, definiendo las extensiones realizadas al SQL para poder definir y manipular

objetos descritos por atributos difusos. Se incluye la descripción de los principales elementos de la interfaz de usuario desarrollada.

- Sexto capítulo: Muestra un ejemplo de utilización de los resultados de esta investigación.
- Séptimo capítulo: Conclusiones de la realización de este trabajo y los trabajos futuros que se derivan de la investigación realizada.

Capítulo 2

Lógica difusa en bases de datos.

En este capítulo se realiza un estudio de los principales temas relacionados con la lógica difusa y su aplicación en bases de datos. Se realiza un recorrido por los principales elementos de la teoría de conjuntos difusos, haciendo énfasis sólo en aquellos elementos que serán empleados en la investigación que esta memoria refleja. se realiza un recorrido por las propuestas existentes en la literatura para la representación de la borrosidad en bases de datos, con énfasis en las realizadas para bases de datos orientadas a objetos. De cada propuesta se ven sus características fundamentales y la forma en que fue tratado cada nivel de vaguedad. Al final del capítulo se describen un grupo de aplicaciones de las bases de datos difusas.

2.1. Modelos de Bases de Datos.

Con el objetivo de soportar la estructura de una base de datos y permitir la manipulación de sus datos se definen varios modelos de datos. Estos modelos pueden clasificarse en dos categorías: lo modelos basados en registros y los modelos basados en objetos. Entre los modelos basados en registros encontramos [KS98] el Modelo Jerárquico de Datos, el Modelo de Red, y el Modelo Relacional. El representante de los modelos basados en objetos es sin duda el Modelo Orientado a Objetos [Cat00]. Sin embargo en los últimos años se ha propuesto un modelo que combina las dos categorías, el modelo Objeto-Relacional [Sto99].

En el apéndice ([**?**]) se describe el Modelo Relacional de bases de datos partiendo de su definición y de los elementos que lo caracterizan. Se realiza una breve descripción del lenguajes para la definición y el modelado de los datos, haciendo énfasis en los comandos

principales. Finalmente se listan los principales gestores de bases de datos relacionales y se describen las principales limitaciones del modelo.

El Modelo Orientado a Objetos se basa en toda la tecnología de orientación a objetos. Por esta razón, como elemento del modelo, se estudian los diferentes conceptos de la *Modelado Orientado a Objetos*: tipos, clases, instancia, herencia y polimorfismo. Luego se realiza un estudio de sus principales características, ventajas e inconvenientes. También se muestra un listado de los productos comerciales que hay en el mercado, todo esto puede consultarse en el apéndice ([**?**]).

El Modelo Objeto - Relacional, se describe en el apéndice ([**?**]). Se ha sido más extenso en su explicación, por ser este el modelo base de la investigación que se presenta en esta memoria. Se realiza un recorrido por las principales características y se ejemplifican utilizando la propuesta presentada por Stonebraker [Sto99] que responde al estándar aceptado SQL:1999. Finalmente se lista un grupo de gestores que hoy pueden encontrarse en el mercado, reflejando en particular algunas de las características de *PostgreSQL* en el apéndice ([**?**]).

Los elementos reflejados en los apéndices antes mencionados permiten ubicar la investigación desde el punto de vista de las bases de datos. El otro elemento dentro de la investigación es la teoría de subconjuntos difusos y la lógica difusa a las cuales se le dedica el siguiente epígrafe.

2.2. Teoría de Conjuntos Difusos.

El desarrollo de soluciones informáticas está dirigido a la solución de problemas del mundo real. Cuando se trabaja con este tipo de problemas puede encontrase información afectada por ciertas imperfecciones que son difíciles de modelar por los modelos clásicos existentes. Se encuentran informaciones afectadas por imprecisiones al momento de definirlas o por la incertidumbre que pueda existir en el valor obtenido de determinada fuente.

Un ejemplo clásico para mostrar la imperfección que puede tener una información, es la representación de la estatura de una persona. Si se cuenta con un metro es posible medir con exactitud la altura, pero si no, se puede estimar el valor. Para los humanos siempre será más fácil decir que una persona es *alta, media o baja* que tratar de estimar su estatura en metros. La pregunta a responder es: ¿Cómo representar los valores *alto, media, bajo* para la estatura? Ahora, si el valor de la estatura de una persona se estima a una determinada distancia del observador; mientras más alejado de la persona este mucho

más difícil será estimar el valor de la altura y entonces la información también tendrá un incertidumbre intrínseca que puede hacer dudar del valor de la información.

Para la representación de este tipo de información se han desarrollado varias teorías, entre ellas la teoría de la probabilidad [Fel71], la teoría de la evidencia [Sch76], el uso de factores de certeza [SB75] o la teoría de subconjuntos difusos [Zad75].

Todas estas propuestas dan una solución al problema desde una óptica diferente. En los últimos 20 años ha habido un despunte en el desarrollo de aplicaciones en todo el mundo que incorporan en sus modelos la teoría de conjuntos difusos. En la Ciencia de la Computación esta teoría se ha ido utilizando en cada una de sus áreas. A continuación se presentan algunos de los principales elementos de esta teoría. Un compendio completo de esta teoría puede ser consultada en [KY95].

2.2.1. Conjuntos clásicos.

La base de la teoría de subconjuntos difusos está en la teoría clásica de conjuntos, por tal razón se debe tener en cuenta la definición de conjunto clásico. Los conjuntos clásicos pueden estar representados de dos formas: por medio de una definición extensiva y utilizando una proposición [KY95].

1. Dando una definición extensiva: se indica una lista con los miembros que forman el conjunto $A = \{a_1, a_2, \ldots, a_n\}$. Cualquier otro elemento que no esté en la lista no pertenecerá automáticamente al conjunto.

2. Proporcionando una proposición P: el conjunto estará formado por aquellos elementos del universo que satisfacen la proposición, $A = \{x|P(x)\}$, la proposición puede ser *verdadera* o *falsa*.

Como se aprecia, la teoría de conjuntos clásicos no proporciona la posibilidad de representar conjuntos de elementos en los cuales exista incertidumbre respecto a su pertenencia al conjunto.

2.2.2. Conjuntos difusos.

En la matemática clásica de conjuntos la función característica de un conjunto asigna un valor en {0,1} para cada uno de los individuos del universo. Ahora se trata de buscar una función característica que asigne valores en el rango [0,1] a cada uno de los elementos

del conjunto, especificando con qué grado pertenece el elemento a dicho conjunto. En tal sentido un conjunto difuso queda definido de la siguiente forma.

Definicion 2.1. ***Conjunto difuso:****Conjunto formado por individuos que tienen asociado un nivel de pertenencia en el intervalo [0,1], descrito por su función de pertenencia. La función de pertenencia del conjunto A sobre el universo X será:*

$$\mu_A : X \rightarrow [0, 1] \tag{2.1}$$

De forma que para cada $x \in X$, $\mu_A(x)$ representa el grado de pertenencia del elemento x al conjunto A. El conjunto difuso A estará representado por los pares de valores x, $\mu_A(x)$.

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

$$A = \mu_A(x_1)/x1 + \mu_A(x_2)/x2 + \cdots + \mu_A(x_n)/xn \tag{2.3}$$

2.2.2.1. Conceptos básicos sobre conjuntos difusos.

Uno de los más importantes conceptos en la teoría de conjuntos difusos es el de $\alpha-corte$ y su variante de $\alpha - corte$ sólido.

Definicion 2.2. *($\alpha - corte$): Tenemos un conjunto difuso A definido sobre el universo X y un número cualquiera $\alpha \in$ [0,1]. Se definen $\alpha - corte$ de A (A_α) y $\alpha - corte$ sólido (A_α^+) como los conjuntos precisos*

$$A_\alpha = \{x | \mu_A(x) \geq \alpha\} \tag{2.4}$$

$$A_\alpha^+ = \{x | \mu_A(x) > \alpha\} \tag{2.5}$$

El subconjunto difuso formado por todos aquellos individuos del universo con un grado de pertenencia mayor o igual que un valor α será al $\alpha - corte$ de A y los que su grado de pertenencia sea estricatamente mayor, serán el $\alpha - corte$ sólido de A.

Propiedad 1-2-1 (Inclusión entre α-*cortes)*: Entre dos α-*cortes* siempre se cumplirá que:

$$\text{Si } \alpha_1 > \alpha_2 \text{ entonces } {}^{\alpha_1}A \subseteq^{\alpha_2} A \tag{2.6}$$

Definicion 2.3. ***(Conjunto de Niveles)****:es el conjunto de valores $\alpha \in$ [0,1] para los que existe al menos un elemento del universo de referencia que pertenece al conjunto difuso A con ese grado. Formalmente:*

$$\wedge(A) = \{\alpha : \mu_A(x) = \alpha \text{ para algún } x \in X\} \tag{2.7}$$

Definicion 2.4. ***(Soporte de un conjunto difuso):*** *es el conjunto clásico formado por todos los elementos del universo X que tienen un grado de pertenencia mayor que 0.*

$$S(A) =^{0+} A = \{x \in X : \mu_A(x) > 0\} \tag{2.8}$$

Definicion 2.5. ***(Núcleo de un conjunto difuso):*** *es el conjunto preciso formado por todos los elementos del universo X que tienen un grado de pertenencia igual a 1.*

$$N(A) = \{x \in X : \mu_A(x) = 1\} \tag{2.9}$$

Definicion 2.6. ***(Altura de un conjunto difuso):*** *es el mayor grado de pertenencia obtenido por un elemento en el conjunto.*

$$h(A) = sup_{x \in X} \mu_A(x) \tag{2.10}$$

Definicion 2.7. ***(Conjunto difuso normalizado):*** *es aquel conjunto difuso con una altura igual a 1.*

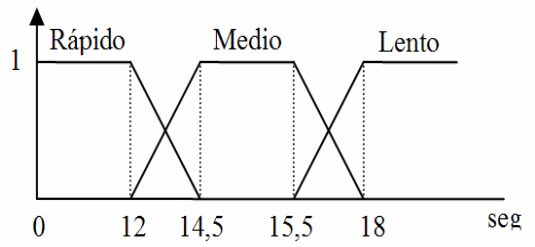
$$h(A) = 1 \tag{2.11}$$

Ejemplo 2.1. *Sobre el universo definido por la medida en segundos del tiempo que tarda un atleta en recorrer 100 m se definen los conceptos de rápido, medio y lento por medio de las siguientes funciones:*

$$Rapido = \begin{cases} 1 & x \leq 12 \\ \left(\frac{14,5-x}{2,5}\right) & 12 < x < 14,5 \\ 0 & x \geq 14,5 \end{cases}$$

$$Medio = \begin{cases} 0 & x \leq 12 \vee x \geq 18 \\ \left(\frac{x-14,5}{2,5}\right) & 12 < x < 14,5 \\ 1 & 14,5 \leq x \leq 15,5 \\ \left(\frac{18-x}{2,5}\right) & 15,5 < x < 18 \end{cases}$$

$$Lento = \begin{cases} 0 & x \leq 15,5 \\ \left(\frac{x-18}{2,5}\right) & 15 < x < 18 \\ 1 & x \geq 18 \end{cases}$$



2.2.2.2. Representación de conjuntos difusos

El principal papel de los $\alpha - cortes$ es su capacidad de representar conjuntos difusos. Cada conjunto difuso puede ser representado por la familia de todos sus $\alpha - cortes$. Esta representación permite extender varias de las propiedades y operaciones de los conjuntos clásicos a los conjuntos difusos. La representación de un conjunto difuso A utilizando conjuntos difusos especiales definidos en términos de $\alpha-cortes$ se denomina descomposición de A.

Definicion 2.8. ***(Teoremas de descomposición):*** *Para cualquier conjunto difuso A se cumple que:*

Primer teorema de descomposición: $A = \bigcup_{\alpha \in [0,1]} \alpha.^{\alpha}A(x)$
Segundo teorema de descomposición: $A = \bigcup_{\alpha \in [0,1]} \alpha.^{\alpha+}A(x)$
Tercer teorema de descomposición: $A = \bigcup_{\alpha \in \wedge(A)} \alpha.^{\alpha}A(x)$

2.2.2.3. Operaciones sobre conjuntos difusos.

Las operaciones que se realizan sobre los conjuntos clásicos pueden ser extendidas a los conjuntos difusos mediante el uso de los $\alpha - cortes$. De esta forma quedan definidas las operaciones sobre conjuntos difusos.

Definicion 2.9. ***(Complementos difusos):*** *Dado un conjunto difuso A definido sobre el*

universo X, $\mu_A(x)$ es interpretado como el grado de pertenencia de x a A. El complemento será $c(\mu_A(x))$y representa el grado en el cual x no pertenece al conjunto A. La función c es independiente del valor de x. Esta función debe verificar los siguientes requerimientos:

1. *c(0) = 1 y c(1)=0 (condición de frontera)*
2. *Para todo a, b ∈ [0,1], si a =b entonces c(a) = c (b) (monotonía)*
3. *c es una función continua*
4. *c(c(a)) = a para todo a ∈ [0,1](involutiva)*

Definicion 2.10. ***(Intersección de conjuntos difusos):*** *La intersección de dos conjuntos difusos A y B está especificada por una función de la forma* $t : [0,1]x[0,1] \rightarrow [0,1]$

Para cada elemento x del conjunto universo, esta función toma como argumentos el par formado por los grados de pertenencia del elemento a los conjunto A y B y produce el grado de pertenencia del elemento al conjunto formado por la intersección de A y B.

(A ∩ B)(x) = t [A(x), B(x)]

Estas funciones son conocidas como t-normas y deben cumplir las siguientes propiedades para todo *a, b, d ∈[0,1]*:

1. *t(a, 1) = a (condición de frontera)*
2. *b = d entonces t(a, b) = t(a, d) (monótona)*
3. *t (a, b) = t (b, a) (conmutativa)*
4. *t (a, t (b, d)) = t (t(a, b), d) (asociatividad)*
5. *t es una función continua*
6. *t(a, a) <= a*
7. *si a1<a2 y b1<b2 entonces t (a1, b1) < t (a2, b2)*

Algunas t-normas de uso frecuente para el cálculo de intersección son:
Intersección estándar: *t(a, b) = min (a, b)*
Producto algebraico: *t(a, b) = a. b*

Diferencia limitada: *t(a, b) = max (0, a + b-1)*

Intersección drástica: $t(a,b) = \begin{cases} a & \text{cuando b} = 1 \\ b & \text{cuando a} = 1 \\ 0 & \text{otras} \end{cases}$

Definicion 2.11. ***(Unión de conjuntos difusos):*** *La unión de dos conjuntos difusos A y B está especificada por una función de la forma* $u : [0,1]x[0,1] \rightarrow [0,1]$

Para cada elemento x del conjunto universo, esta función toma como argumentos el par formado por los grados de pertenencia del elemento a los conjunto A y B y produce el grado de pertenencia del elemento al conjunto formado por la unión de A y B.

(A ∪ B)(x) = u [A(x), B(x)]

Estas funciones son conocidas como t-conormas y debe cumplir las siguientes propiedades para todo *a, b, d ∈[0,1]*:

1. *u (a, 0) = a (condición de frontera)*
2. *b = d entonces u(a, b) = u(a, d) (monótona)*
3. *u (a, b) = u (b, a) (conmutativa)*
4. *u (a, u (b, d)) = u (u(a, b), d) (asociatividad)*
5. *u es una función continua*
6. *u(a, a) >= a*
7. *si a1<a2 y b1<b2 entonces u (a1, b1) < u (a2, b2)*

Algunas t-conormas de uso frecuente para el cálculo de intersección son:
Unión estándar: *u (a, b) = max (a, b)*
Suma algebraica: *u (a, b) = a + b – a b*
Suma limitada: *u(a, b) = min (1, a + b)*

Unión drástica: $U(a,b) = \begin{cases} a & \text{cuando b} = 0 \\ b & \text{cuando a} = 0 \\ 1 & \text{otras} \end{cases}$

Un ejemplo de estas dos operaciones lo tenemos en la Figura 2.1.

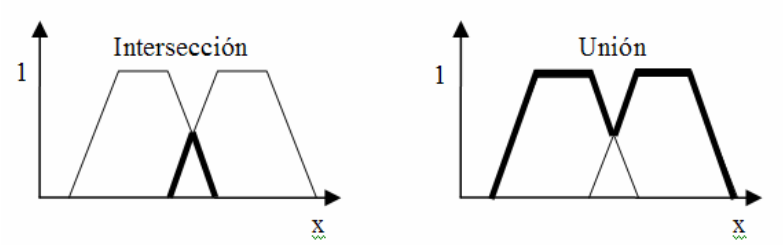


Figura 2.1: Intersección y Unión estándar.

2.2.2.4. Implicación.

La operación lógica de implicación es fundamental en la lógica difusa pues mediante ella podemos realizar inferencias dando la base para lo que se conoce como razonamiento aproximado.

Definicion 2.12. ***(Implicación difusa):*** *En general, una implicación difusa es una función de la forma I: [0,1] x [0,1] → [0,1]. Para cualquier par de valores verdadero a y b de las proposiciones difusas p y q respectivamente, el valor de certeza de la proposición condicional "si q, entonces p"será la implicación del a en b, I(a, b). Esta función es una extensión de la implicación clásica, p⇒ q definida para el dominio [0,1].*

I(a, b): es el grado de verdad de que B implica A.
Es necesario que la función de implicación satisfaga las siguientes propiedades:

1. $a \leq b$ *implica que* $I\ (a,\ c) \geq I\ (b,\ c)\ \forall a,\ b,\ c \in [0,1]$
2. $a \leq b$ *implica que* $I\ (c,\ a) \leq I\ (c,\ b)\ \forall a,\ b,\ c \in [0,1]$
3. $I\ (0,\ a) = 1\ \forall a \in [0,\ 1]$
4. $I\ (1,\ a) = a\ \forall a \in [0,\ 1]$
5. $I\ (a,\ a) = 1\ \forall a \in [0,\ 1]$
6. $I\ (a,\ I\ (b,\ c)) = I\ (b,\ I\ (a,\ c))\ \forall a,\ b,\ c \in [0,\ 1]$
7. $I\ (a,\ b) = 1$ *si* $a \leq b\ \forall a,\ b \in [0,\ 1]$
8. $I\ (a,\ b) = I(c\ (b),\ c\ (a))$

9. *I es una función continua.*

La definición general de estos operadores se puede obtener a partir de la extensión de definiciones de implicación en la lógica clásica:

Usando el complemento y t-conormas:

$$I(a,b) = u(c(a),b) \tag{2.12}$$

Usando t-norma:

$$I(a,b) = sup\{x \in [0,1] | t(a,x) \leq b\} \tag{2.13}$$

Existen diferentes enfoques para especificar un operador de implicación. En todos los casos se obtienen a partir de seleccionar una t-norma, una t-conorma o un complemento difuso como base del operador.

En primer lugar están los operadores a partir de t-conormas, llamados S-implicaciones, y que se basan además en operaciones de complemento difuso. Se destacan en la literatura las siguientes implicaciones:

$$\begin{array}{ll}
-I(a,b) = max(1-a,b) & \text{Kleene - Dienes} \\
-I(a,b) = 1-a+ab & \text{Reichen} \\
-I(a,b) = min(1,1-a+b) & \text{Lukasiewicz} \\
-I(a,b) = \begin{cases} b & \text{cuando } a = 1 \\ 1-a & \text{cuando } b = o \\ 1 & \text{otro caso} \end{cases} & \text{Unión drástica. S-implicación larga.}
\end{array} \tag{2.14}$$

Los operadores que utilizan t-normas para determinar el grado de implicación son conocidos como R-implicaciones y la literatura destaca los siguientes:

$$-I(a,b) = sup\{x|min(a,x) \leq b\} = \begin{cases} 1 & a \leq b \\ b & a > b \end{cases} \text{ Gödel}$$
$$-I(a,b) = sup\{x|ax) \leq b\} = \begin{cases} 1 & a \leq b \\ \frac{b}{a} & a > b \end{cases} \text{ Goguen}$$

(2.15)

$$-I(a,b) = sup\{x|max(0,a+x-1) \leq b\} min(1,1-a+b) \quad \text{Lukasiewicz}$$
$$-I(a,b) = \begin{cases} b & a = 1 \\ 1 & \text{otros} \end{cases} \text{ Limite de toda R-implicación}$$

Relación entre conjuntos difusos

Al igual que en la teoría clásica de conjuntos, en la teoría de conjuntos difusos es posible definir relaciones entre conjuntos. Estas relaciones representan la presencia o ausencia de asociación, interacción e interconexión entre los elementos de dos o más conjuntos. En la teoría clásica los conjuntos pueden estar relacionados entre sí en todo momento, esta relación podrá tener valores en el conjunto {0,1}. Zadeh [Zad75] generalizó el concepto de relación clásica de forma que la función característica que representa una relación entre dos conjuntos pudiera tener como resultado grados de pertenencia definidos entre [0, 1]. Así se definen los conceptos de relación difusa, relación de semejanza y relación de similitud.

Definicion 2.13. ***(Relación difusa):*** *La relación difusa entre dos conjuntos difusos está definida por el producto cartesiano de los dos universos, con una función característica asociada que asigna valores en el rango [0,1] para cada tupla de la relación.*

Definicion 2.14. ***(Relaciones de semejanza o Relaciones de compatibilidad):*** *Es una relación difusa binaria $R(X,Y)$ reflexiva y simétrica.*

1. $\mu_R(x,x) = 1$, para toda $x \in X$ (reflexiva)
2. $\mu_R(x,y) = \mu_R(y,x)$, para toda $x,y \in X$ (simétrica)

Definicion 2.15. ***(Relación de similitud o Relación de equivalencia):*** *Es una relación difusa binaria $R(X,Y)$ reflexiva, simétrica y max-min transitiva.*

1. $\mu_R(x,y) = max_{w \in U}\{min(\mu_R(x,w), \mu_R(w,y))\}$ max-min transitividad

Principio de extensión.

El principio de extensión propuesto por Zadeh y reflejado en [KY95], permite extender operaciones sobre conjuntos clásicos al caso de conjuntos difusos.

Definicion 2.16. ***(Principio de Extensión):*** *Sea x e Y referenciales sobre los que existe una correspondencia f: X→Y. A y B subconjuntos difusos definidos sobre X e Y respectivamente. Tenemos que el principio de extensión plantea que:*

$$\mu_B(y) = sup_x\{\mu_A(x)|y = f(x)\} \tag{2.16}$$

Si la función está definida por medio de un producto cartesiano, por ejemplo $f : X_1 \times X_2 \times \cdots \times X_3 \rightarrow Y$, donde $X_i, i = 1 \ldots n$ e Y son universos de referencia y $A_1, A_2, \ldots, A_n$ subconjuntos difusos definidos sobre los universos de referencia X_i, entonces la extensión de la función se define como

$$\mu_B(y) = sup_{(x_1,x_2,\ldots,x_n)} inf\{\mu_{A1}(x1), \ldots, \mu_{An}(xn)|y = f(x_1, ..., x_n)\}\} \tag{2.17}$$

Variables lingüísticas

Los números difusos juegan un importante papel en la formulación de variables difusas cuantitativas. En un modelo pueden existir variables cuyos valores son números difusos. Cuando los números difusos representan conceptos lingüísticos la estructura resultante se denomina *variable lingüística* [KY95].

Esta definición nos permite representar y manipular datos más sofisticados, más cercanos a la forma humana de razonar y percibir la realidad. Los humanos tendemos a utilizar palabras para expresar nuestro razonamiento evitando la especificación que conlleva el uso de números.

Por ejemplo acostumbramos a decir *"Jorge es un atleta lento"* y no *"Jorge recorre los 100 m a una velocidad de 19,65"*. La segunda sentencia contiene más información pero en ocasiones resulta difícil poder expresarnos con esos niveles de exactitud. Y es aquí donde se definen las variables lingüísticas para representar conceptos imprecisos.

Definicion 2.17. ***(Variable lingüística):*** *Cada variable lingüística está totalmente caracterizada por un quíntuplo (v, T, X, g, m) en el cual:*

1. v es el nombre de la variable,
2. T el conjunto de valores lingüísticos de v,
3. x es el universo de discurso de la variable,
4. g una regla sintáctica para la generación de los términos lingüísticos y
5. m es una regla sintáctica que asigna para cada término lingüístico $t \in T$ su significado.

El ejemplo de la variable lingüística definida en Ejemplo 1-22 quedaría caracterizada de la siguiente forma:

v = "velocidad"

T = {Rápido, Media, Lenta}

X = velocidad medida en segundos

g = las reglas para generar los valores de T

m = las reglas que asocian a cada elemento de T su significado. Para cada valor $L \in T$, m (L) será un subconjunto de X

Definicion 2.18. ***(Variable lingüística estructurada):*** *una variable lingüística es estructurada si T y m pueden ser caracterizados algorítmicamente.*

La variable lingüística se puede construir a partir de un conjunto de términos atómicos, permitiendo la construcción de otros nuevos compuestos mediante el uso de modificadores.

Definicion 2.19. ***(Modificador lingüístico):*** *termino lingüístico que se puede aplicar sobre otro ya existente para construir uno nuevo, actuando como un operador sobre el conjunto difuso que representa el significado de su operando.*

Los modificadores {*muy, bastante, poco, más o menos*} se pueden aplicar a un conjunto de términos {*rápido, medio, lento*} para construir nuevos términos {*muy rápido, rápido, medio, lento más o menos lento, muy lento*}.

Roles de las variables lingüísticas en la lógica difusa.

Las variables además de ser utilizadas para expresar predicados vagos pueden ser utilizadas en otros contextos. Por ejemplo se puede emplear una variable lingüística para expresar la veracidad que se tenga sobre una proposición o para denotar probabilidades. En estos casos se utiliza como referencial el intervalo real definido en [0,1] y sobre el se definen un conjunto de etiquetas que expresen valores en una gama definida.

Ejemplo 1-23: Veracidad expresada como variable lingüística.

Para expresar la veracidad podemos utilizar el siguiente conjunto de etiquetas {*falso, bastante falso, más bien falso, tan falso como verdadero, más bien verdadero, bastante verdadero, verdadero*}. Cada una de estas etiquetas estará definida por un número difuso sobre el referencia [0,1]. De forma similar se puede expresar la probabilidad que se tenga sobre un valor.

2.3. Bases de datos difusas.

Mucha de la información existente en el mundo real está afectada por imperfección. Esto puede estar motivado por la incertidumbre que puede existir al momento de obtener la información o por la imprecisión de los términos con que esta información se define. En algunos casos se observan ambos tipos de imperfecciones en la información.

Los sistemas más comunes para el almacenamiento y tratamiento de la información, representada por grandes volúmenes de datos, sin lugar a dudas, son los sistemas de gestión de bases de datos. Estos tipos de sistemas, han evolucionado en diferentes modelos: modelo jerárquico, modelo en red y, el más aceptado por todos, el modelo relacional [KS98]. En una etapa posterior, y respondiendo a las necesidades del mercado, surgen las bases de datos orientadas a objetos y más adelante las bases de datos objeto -relacional, que vinieron a dar soluciones a problemas que no podían ser representados por otros modelos. Pero hasta aquí toda la información que se almacenaba y manipulaba tenía una característica común: era precisa.

Durante los últimos años las investigaciones en el campo de las bases de datos se han dirigido a la incorporación de semántica adicional a los modelos de datos que permitan la representación y manipulación de información imprecisa. El primero en proponer una forma de representar información imprecisa en las bases de datos fue el propio creador del modelo relacional, Cood, con la introducción del uso de valores nulos, incorporando un nuevo resultado a la comparación entre dos valores el "quizás". Luego esta propuesta fue ampliada por el propio Cood en trabajos posteriores [AM97]. Esta estrategia para trabajar la imperfección en la información fue retomado por Guy de Tré y otros [TCP08].

La lógica difusa y la teoría de subconjuntos difusos propuesta en los años 80 por Zadeh se convirtieron en una de las bases para hacer frente a los distintos tipos de imprecisión e incertidumbre que pueden aparecer en los datos del mundo real. Inicialmente el modelo relacional fue el primer modelo al que se le comenzaron a realizar extensiones para soportar

el uso de la teoría de conjuntos difusos y la lógica difusa en la representación de datos. Los trabajos se dirigieron a extender el modelo de datos básico y los lenguajes de consulta de forma que permitieran la representación y recuperación de datos imprecisos.

En la literatura se pueden encontrar varios tipos de bases de datos relacionales difusas. Los primeros intentos fueron realizados por Baldwin [Bal79] y Umano [Uma82]. Luego surgieron otros modelos basados en relaciones de similaridad [BP82] o relaciones de proximidad [SM89] o semejanza [RHB89]. Otros modelos utilizan un enfoque posibilístico utilizando distribuciones de posibilidad [PT84, RM88], los cuales a su vez, pueden ser clasificados en dos categorías: tuplas relacionadas con posibilidades y valores de los atributos representados por distribuciones de posibilidades [RDP98]. Otra de las extensiones hechas a este modelo son las que combinan distribuciones de posibilidad y relaciones de similaridad [CKV92, CKV94, CKV96, MZM00, MZMC00, RHB89].

Muchas han sido las investigaciones en este sentido, pudiéndose dividir en tres áreas significativas: permitir realizar consultas difusas sobre una bases de datos relacional con datos crisp, realizar consultas difusas sobre bases de datos relacionales con datos imprecisos y aproximaciones orientadas a objetos [DP94, Mou99, VCMP94a, Yag91, Gal05, CVK91, CV94, CV95, RM88, VCMP94a, VCMP94b, **?**]

En 1994 se propone el modelo GEFRED, [MPM94], que constituye una síntesis de todos los modelos hasta esa fecha publicados. Luego otros investigadores han continuando expendiéndolo aumentando considerablemente sus funcionalidades. Este modelo se basa en los Dominios Difusos Generalizados y las Relaciones Difusas Generalizadas, definiciones que incluyen los dominios clásicos y referencias clásicas, respectivamente. Este modelo da soporte a varios tipos de datos. Basado en sus tipos de datos se definen operadores difusos de posibilidad y de necesidad [Gal05], **??** capaces de comparar una columna con una constante o dos columnas del tipos compatibles.

En extensiones realizadas a GEFRED, [Gal05] se propone un lenguaje que modifica el lenguaje SQL con la finalidad de expresar valores difusos, condiciones difusas, atributos difusos, etc. Este lenguaje se le dio el nombre de Fuzzy SQL o FSQL y requiere del Servidor FSQL para interpretar las sentencias difusas y traducirlas al SQL soportado por la Bases de Datos. La implementación inicial de este modelo se realizó en el Sistema Gestor de Bases de Datos Oracle y actualmente existe una versión para *PostgreSQL* [**?**]. Desde el punto de vista práctico esta propuesta es la más acertada.

2.4. Imprecisión en bases de datos orientas a objetos.

Trabajos posteriores se han dirigido a la representación y manipulación de datos difusos en bases de datos orientadas a objetos. Varios han sido los modelos difusos de base de datos orientadas a objetos propuestos en la literatura.

Puesto que el objetivo de esta investigación es trasladar algunas de las características de esos modelos al modelo objeto-relacional de *PostgreSQL*, se hará un recorrido algo más detallado por algunas de las propuestas más relevantes que se encuentran en la literatura.

De forma general, la vaguedad en los modelos de bases de datos orientados a objetos se ha incorporado en los siguientes niveles [Cal97]:

1. *Nivel de atributos:* se permite la entrada de datos difusos en la base de datos, los atributos se definen con dominios difusos.

2. *Nivel de relaciones de instancia*: vaguedad en la pertenencia de un objeto a una determinada clase. Un objeto puede pertenecer a una clase en un grado definido en el intervalo [0, 1] en vez de su pertenencia o no en los sistemas clásicos.

3. *Nivel de relaciones de herencia*: en este caso existe cierta vaguedad en la relación entre una clase y sus superclases. Se define un grado para las relaciones *IS_A* entre la clase y la superclase, difuminando la herencia del conjunto de variables de instancia. Otra perspectiva en este caso es considerar cómo se heredan los rangos de valores asociados a una instancia.

4. *Nivel de definición*: se centra en la definición del tipo de la clase y de los mismos objetos, considerando la presencia de vaguedad en el mismo nivel de la estructura.

5. *Nivel de conducta*: presencia de vaguedad en la conducta de los objetos.

Los trabajos que se reflejan en la literatura, en su gran mayoría, abordan los tres primeros niveles de vaguedad y sólo algunos modelos tratan los niveles de definición y conducta. Estos trabajos están dirigidos a tres campos principales: la manipulación de sistemas de gestión de bases de datos, el modelado de software y la representación de conocimiento en inteligencia artificial.

Durante muchos años de investigación varios autores han trabajado en la incorporación de la teoría de conjuntos difusos y la lógica difusa al tratamiento de la vaguedad en los datos, aplicado a *SGBDOO*.

Los primeros trabajos estuvieron dirigidos a la utilización de modelos semánticos avanzados para la representación de la imperfección en los datos. Estos trabajos comenzaron por crear un marco de reconocimiento y definiciones de los distintos tipos de información imperfecta que se deben consideran en un modelo de datos. Se comienza a hablar de representar, mediante términos lingüísticos y conjuntos difusos, la imperfección que pueda tener un dato. Y comienzan a plantearse los distintos niveles en los que la información puede estar afectada por imperfección [Rus86, ZC86].

En 1988 Granger [Gra88] presenta una investigación donde se aplica la teoría de la posibilidad al problema de reconocimiento de objetos. Este problema lo plantea como un problema de clasificación, caracterizado por una representación orientada a objetos del conocimiento y estrategias de control basadas en procedimientos difusos de reconocimiento de patrones. La taxonomía de las clases es representada por jerarquías y el cálculo de la coincidencia se basa en la teoría de la posibilidad. Las diferencias entre objetos y el cálculo de esa diferencia dependen del tipo y de las propiedades del objeto. Ya aquí se tienen en cuenta pesos para los campos.

Estas ideas iniciales, son presentadas por Vandenberghe et al. [VC91] mucho más elaboradas, definiendo conceptos como los de conjunto difuso de entidades, subclases difusas y categorías difusas.

Estos trabajos motivaron la investigación en las *SGBDOO* difusos. Surgió una nueva generación de investigadores en este campo que comienzan a desarrollar trabajos tanto desde el punto de vista teórico como práctico, para introducir la representación de vaguedad en los *SGBDOO*. Algunos de estos trabajos han sido.

1. Rossazza, Dubois y Prade [RDP98] presentan en 1991 una investigación donde utilizan la teoría de la posibilidad para representar jerarquías de clases difusas, hablando por primera vez de un modelo jerárquico de clases difusas. En su propuesta, las clases se describen en término de sus atributos, distinguiendo para cada uno entre rangos permitidos y rangos típicos. Se estudia el concepto de inclusión, aplicado a la relación entre clase y objetos y clases. También se definen tres tipos de herencia: herencia típica, herencia normal y herencia atípica. Estos trabajos iniciales fueron enriquecidos por los mismos autores en [Rus86].

2. Un tiempo después, George, Buckles y Petry [GBP91, GBP93], basados en lo trabajos de Rossazza et al., consideran la utilización en el modelado orientado a objetos de jerarquías de clase difusas, de forma tal que una clase puede heredar de una superclase

en cierto grado. En los valores de atributos se empiezan a considerar conectores lógicos que además se utilizan para desarrollar el cálculo del grado de inclusión entre clases y pertenencia entre objetos y clases respectivamente. Se comienzan a utilizar las relaciones de similitud para modelar la imperfección en los valores de los atributos.

3. Tanaka et al. [TKS91] en su trabajo especifica una extensión de Modelo de Datos Orientado a Objetos que involucra la manipulación de imprecisión e incertidumbre en el nivel de los atributos y de las relaciones entre los objetos. La propuesta aborda la extensión difusa del *MDOO* y se utiliza sobre aplicaciones reales basadas en estructuras de objetos complejos.

4. Los trabajos presentados por George et al. le sirvieron a Yazici et al. para desarrollar su propio modelo [YAG96, YC98, YGA98, YK97]. En sus trabajos desarrollan bases de datos difusas orientadas a objeto para la representación del conocimiento y la extensión de capacidades de deducción. En la propuesta se maneja la vaguedad a nivel de atributo, nivel objeto/clase y nivel clase/superclase, basado siempre en relaciones de similitud.

5. Bordogna y Pasi et al. [BLP94] proponen un modelo orientado a objeto difuso para manejar datos precisos y datos difusos. Ellas desarrollaron un modelo de datos basado en una notación gráfica donde la imprecisión y la incertidumbre pueden ser manejadas en el nivel de los atributos de los objetos y en las relaciones entre los objetos. Para formular preguntas imprecisas y para recuperar objetos exactos o imprecisos con un cierto grado, la incertidumbre se maneja en dos niveles: imprecisión en los datos y en el conocimiento de los datos. En [BP01] se aboga por un conjunto de operaciones basadas en grafos para visualizar y recuperar la información a partir de un modelo orientado a objeto difuso representado por grafos. En 1999 Pasi [PY99] modelan la incompletud a nivel de las relaciones de herencia.

6. En [**?**] Nepal et al. proponen un lenguaje de consulta sobre objetos difusos (*FOQL*) que soporta borrosidad para una base de datos de imágenes. El lenguaje puede ser usado para definir esquemas y conceptos de alto nivel y para consultar bases de datos de imágenes.

7. En 1994 Van Gyseghem y Rita Caluwe [GCV93] proponen el modelo *UFO* (*Uncertainty and Fuzziness in Object-orientation*) para bases de datos, una propuesta muy

completa donde la representación de la vaguedad en las base de datos orientadas a objeto se expresa por medio de distribuciones de posibilidad y se introduce el concepto de objetos-rol. En esta propuesta además de tratar el nivel de atributo, de relaciones de instancia y nivel de herencia, se trata también el nivel de conducta.

8. Vila M.A. et al. [VCMP95, VCMP96a, VCMP96b, VCMP97, VCMP98]proponen un modelo semántico de datos para la representación de información imprecisa en una base de datos orientada a objetos. La representación del modelo se basa en el modelo IFO [AH87] e incorpora nuevos elementos gráficos que representan los diferentes niveles de vaguedad que se pueden encontrar en la base de datos. Se definen las diferentes formas en que la imprecisión puede afectar a los valores de los atributos y sus formas de representarla. Además de estudiar la imprecisión en las relaciones entre clases y subclase y entre objeto y clase.

9. Valerie Cross et al. [CF00] describen la incertidumbre en los valores de los atributos para objetos y clase, los diferentes significados de esa incertidumbre y cómo afectan a la herencia clase/superclase. En [Cro01] se examinan varias semánticas de incertidumbre y la interacción para relaciones de herencia en modelos de objetos.

10. N. Marín et al. [MBPV03, MMP$^+$03, MPV01, MVP00], basándose inicialmente en los trabajos de Vila [VCMP98] proponen un modelo orientado a objetos que soporta la vaguedad en los cinco niveles previamente definidos. Presentando una perspectiva para trabajar con el nivel de definición basada en el uso de tipos difusos asociados a una clase. En el cálculo del parecido entre objetos se utilizan relaciones de semejanza.

11. En el 2003 Guy de Tré y R. Caluwe [TC03], presentan una investigación donde se aplica el concepto de *conjuntos difusos de nivel-2* en el modelado de bases de datos orientadas a objetos.

Sin duda, estos son los trabajos más relevantes en el área. Dentro de ellos destacan los modelos de Rossaza, Gyseghem, George, Yazici, Marín y Bordogna. Algunos de los principales detalles de estas propuestas se describen en los próximos subtemas de este capítulo.

2.4.1. Modelo FOOD e IFOOD.

A partir de las propuestas de R. George en [GBP91], A. Yazici [YGA98] plantea el modelo *FOOD* con el objetivo de representar situaciones del mundo real dentro de un sistema de gestión de bases de datos. El modelo es capaz de soportar vaguedad a nivel de atributo, nivel de relaciones de instancia y nivel de relaciones de herencia. La base de este modelo es la sustitución de la relación de igualdad entre atributos por relaciones de similitud. La Tabla 1-8 resume las principales aportaciones de este modelo.

M. Koyuncu junto a el propio Yazisi et al. proponen una extensión al modelo *FOOD* [YK97]. Ellos proponen una arquitectura para *bases de datos orientas a objetos difusas inteligentes* basado en la unión de un sistema de bases de datos orientadas a objetos difusos con un sistema basado en el conocimiento. Se define y extiende un lenguaje con definiciones declarativas y gran capacidad de consultas sobre bases de datos orientadas a objeto difusas. Presenta el concepto de *clases virtuales* con reglas difusas, el usuario puede definir sus clases virtuales especificando reglas de derivación a partir de clases existentes en la base de datos.

El modelo *FOOD* forma parte de la arquitectura *IFOOD* junto con una base de conocimiento difuso (*FKB*). El tratamiento de la imperfección del modelo se basa en los elementos definidos para *FOOD*.

2.4.2. Modelo UFO.

Van Gyseghem N. y De Caluwe R. en [GC96, GC98, GCV93] hacen una propuesta de un modelo para manipular la imprecisión y la incertidumbre en bases de datos orientadas a objetos. Esta es una de las propuestas más completas para el modelado de bases de datos orientadas a objetos difusas. La Tabla 2.2 muestra los principales elemento de esta propuesta.

El modelo de datos UFO logra una representación de la información imperfecta de forma que su manipulación es transparente para el usuario. Se modela la imprecisión y la incertidumbre por medio de distribuciones de posibilidad y se introduce el concepto de rol (*role*). El concepto de rol expresa las relaciones de imprecisión e incertidumbre que puedan existir entre objetos o entre objetos y clases de objetos, pudiendo además modelar la imprecisión que pueda existir en la aplicabilidad de una propiedad a un objeto.

Nivel	Descripción
Atributos	A cada atributo de una clase se le asocia un dominio y un rango que lo caracteriza, definiendo una relación de similitud para el conjunto de valores que pueden tomar el atributo. Los rangos se definen como subconjuntos difusos de los valores del dominio especificando una semántica asociada que puede ser <AND>, {OR} o [XOR] en dependencia del significado que se le quiera dar a los valores de los atributos. Al especificar <AND> el atributo puede tomar como valor un subconjunto del rango. Si es {OR} el valor puede ser uno u otro de los especificados en el rango. Y si se especifica [XOR] solo un valor del rango es verdadero para el atributo.
Relaciones de Instancia	Para calcular el grado de pertenencia de un objeto a su clase se utiliza la similaridad entre los valores de los atributos del objeto y el rango de valores definido en la clase, esto se calcula por medio de la inclusión. Para este cálculo se tiene en cuenta el grado de relevancia de cada atributo. Cuanta mayor similaridad exista entre los valores de los atributos de un objeto y el rango definido en la clase mayor será el grado de pertenencia de objeto a la clase
Relaciones de Herencia	El cálculo de la relación de pertenencia de una clase a una superclase, respondiendo a una relación de herencia, se realiza teniendo en cuenta la relación existente entre los rangos de los atributos y la relevancia de los atributos para la relación. Se realiza de forma similar a las relaciones de instancia pero se trabaja con los rangos de ambas clases.

Cuadro 2.1: Aportaciones del modelo FOOD.

Nivel	Descripción
Atributos	Se puede especificar le imperfección que existe en los valores de los atributos por medio de clases, sobre las que se pueden construir distribuciones de posibilidad. Los atributos pueden tener valores con un sentido disyuntivo o con un sentido conjuntivo. Para cada uno de los casos existen definidas clases que permiten modelar los valores. Se facilitan métodos que permiten manipular los conjuntos difusos creados.
Relaciones de Instancia	Los objetos definidos en el modelo tienen asociado un grado de pertenencia a la clase.
Relaciones de Herencia	Existe una relación de herencia difusa entre las clases. La herencia puede ser parcial o condicionada. En la herencia parcial se hereda el núcleo de las propiedades requeridas de la clase como propiedades requeridas de la subclase. El resto de las propiedades se heredan de forma condicional. En la herencia condicionada todas las propiedades se heredan de forma condicional.
Definición	Las clases en el modelo pueden modelarse de forma difusa especificando *propiedades requeridas* y *propiedades opcionales.* Al crear un objeto se especifica qué propiedades opcionales se incluirán y se define un grado de aplicabilidad asociado a cada propiedad difusa, indicando el grado que esa propiedad es aplicable al objeto. Es posible utilizar un modelo hipotético donde una clase puede tener distintos aspectos. Al crearse una instancia de la clase se deben definir ROLES para modelar el posible aspecto del objeto en dependencia de las posibilidades que permite la definición de la clase.
Conducta	La borrosidad presente en los valores de los atributos se propaga hacia los métodos. Al aplicar un método sobre un objeto difuso se obtendrá como resultado un conjunto difuso definido sobre una clase resultado.

Cuadro 2.2: Características del modelo UFO

2.4.3. Modelo de Vila et al.

En [VCMP95, VCMP96a, VCMP96b, VCMP97, VCMP98] se propone un modelo semántico para representar la vaguedad en un sistema de bases de datos orientado a objetos. Usando el modelo semántico de datos propuesto se logra representar diferentes tipos de información imprecisa. Se trabaja con diferentes clases de datos imprecisos y se especifica la forma en que pueden describirse. La propuesta, representa la imprecisión de un dato y su incertidumbre en una única representación.

Se realiza un estudio detallado de la imprecisión e incertidumbre que puede parecer en el nivel estructural de la base de datos, concentrándose en las relaciones objeto/clase. Un análisis muy detallado se realiza de las relaciones de especificación/generalización entre clase, definiendo formas para representar la vaguedad que pueda existir en estas relaciones y como influye la naturaleza imprecisa de una clase en la naturaleza de sus atributos. De la misma forma se enfoca el problema de la relación entre clases difusas, especificando las diferentes posibilidades y dando propuestas para solucionarlo. En la Tabla 2.3 se resumen las aportaciones de esta propuesta en la representación de vaguedad en sistemas de bases de datos orientadas a objeto.

En estos trabajos se utiliza un modelo semántico clásico, IFO, de está forma se garantiza que el modelado de datos imprecisos pueda ser representado en un modelo de bases de datos orientado a objetos por medio de una traslación de un modelo a otro. Las primeras pautas para hacer esta traslación son presentadas también en estos trabajos.

2.4.4. Modelo de G. Bordogna et al.

Gloria Bordogna y Gabriela Pasi presentan una propuesta para la representación de información *crisp* e información imperfecta en un modelo de bases de datos difusos orientado a objetos [BLP94], [BP01]. El modelo propuesto se define como una extensión al modelo de objeto basado en gráficos definido por Lucarella en 1993 [LSZ93] y está formalizado en las teorías de conjuntos difusos y de posibilidades.

El trabajo se basa en el paradigma de que los esquemas conceptual y de instancias de una base de datos pueden ser representados directamente por etiquetas en un gráfico. El modelo solo se centra en la representación de las distintas formas que puede tomar una información imperfecta presente en una base de datos. Elementos como el cálculo de relaciones entre los elementos del esquema no son tratados. Veremos como estos autores tratan los diferentes niveles de imprecisión en el modelo (Tabla 1-11).

Nivel	Descripción
Atributos	Se describen tres formas de representar la vaguedad en este nivel: atributos definidos sobre valores sin representación semántica asociada, valores con representación semántica asociada con un significado disyuntivo y valores con representación semántica asociada pero con un significado conjuntivo. Para cada caso se especifica la forma de representarlo en el modelo semántico. De la misma forma se especifica la forma de representar la incertidumbre que se tenga de un valor. Esta especificación es posible hacerla también a relaciones entre objetos. Las relaciones entre objetos admiten la definición de un grado de importancia.
Relaciones de Instancia	Se considera que un objeto puede pertenecer a una clase con un grado. Para esto se define un atributo en la clase que representa la pertenencia del objeto a la clase. Se define una forma para representar la vaguedad que pueda existir en jerarquías de clases, tanto construidas a partir de una especialización como de una generalización. En todos los casos el modelo se auxilia de atributos valuados en el intervalo [0,1] definidos sobre las clases para representar la pertenencia de un objeto a una clase.
Relaciones de Herencia	No se consideran grados para las relaciones superclase – subclase.

Cuadro 2.3: Modelo Vila M.A. et. al.

Nivel	**Descripción**
Atributos	Se pueden definir dominio difusos para lo atributos. Un atributo puede ser simple, definido sobre un dominio o complejo cuando su valor puede ser un objeto. El valor puede tener significado disyuntivo o conjuntivo. Las relaciones de los objetos con un atributo pueden tener definido un grado de certeza que se tenga sobre el valor.
Relaciones de Instancias	Se definen clases precisas y clases difusas. Las instancias de una clase tendrán asociado un grado de pertenencia que tomará valor entre las siguientes etiquetas {*none, very low, low, médium, high, very high, full*}. Si la instancia es de una clase precisa el grado será *full*. En cualquier otro caso se especificará al crear la instancia.
Relaciones de Herencia	Existen relaciones de herencia entre las clases que están especificadas por medio de un modificar de la relación que puede tomar los siguientes valores {*not completely, more or less, very, definitely*}. Este modificador representa el grado en el cual una clase puede ser considerada como subclase de otra clase.

Cuadro 2.4: Resumen modelo Bordogna – Pasi.

Como mencionamos el modelo se basa en una representación gráfica. Se representan dos esquemas, el esquema conceptual y el esquema de instancias. En el esquema conceptual se define por medio de un quintuple de valores {*C, T, A, P, H*} donde: *C* es el conjunto de nombre de clases, *T* el conjunto de nombre de tipos, *A* el conjunto de nombre de los atributos, *P* las relaciones entre los atributos y los tipos y *H* las relaciones de herencia entre clases. Este esquema refleja en nodos en forma rectangular las clases precisas, nodos rectangulares rellenados para las clases difusas, óvalos para lo tipos precisos y óvalos rellenados en el caso de tipos difusos. Un ejemplo en la Figura 2.2.

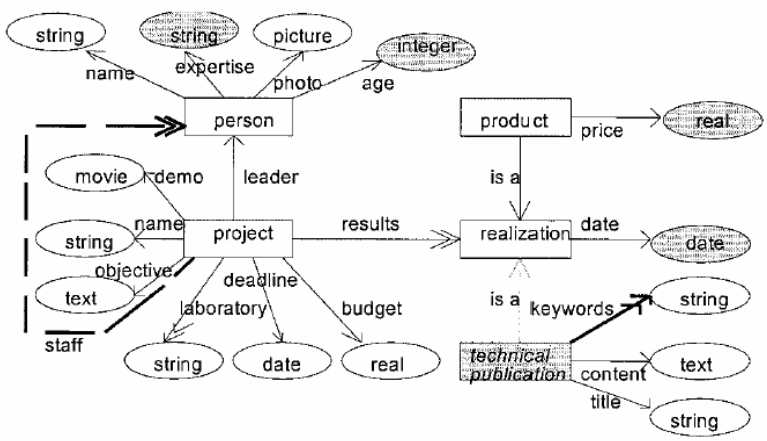


Figura 2.2: Esquema conceptual de una BD [BP01]

El esquema de instancias se define de igual forma por un quintuple de elementos *M = (, O, V (P), L, I)* donde: es un esquema conceptual previamente definido, *O* es el conjunto de objetos de la base de datos instancias de las clases previamente definidas, *V (P)* es el conjunto de valores reales cargados en la base de datos. *L,* son las relaciones de tipo link que existen entre los objetos y sus atributos ya sean simples o complejos. *I,* son las relaciones de instancia entre un objeto y una clase. Un ejemplo de un esquema de instancia lo tenemos en la Figura 1-8.

El modelo presentado esta orientado al razonamiento, logrando una representación de las distintas formas de información imperfecta que puede aparecer en una base de datos.

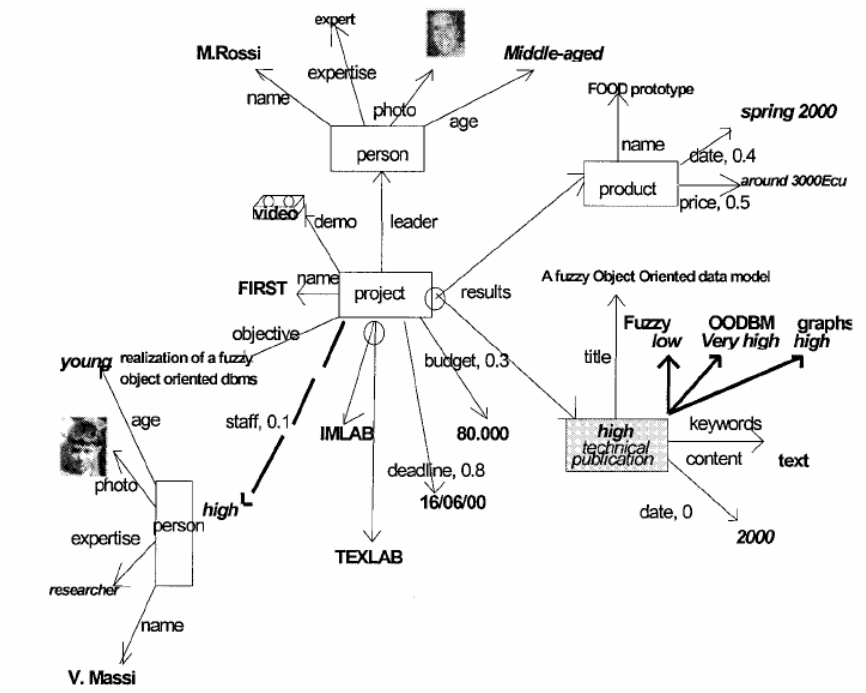


Figura 2.3: Esquema de instancia de una BD [BP01]

2.4.5. Modelo de J-P Rossaza at al.

Esto autores proponen en [RDP98] un modelo bastante completo para la representación de información imperfecta en una bases de datos orientada a objetos. Ellos abarcan casi todos los niveles donde se puede encontrar vaguedad en el modelo, especificando en cada uno la forma de representarla. Además dan los elementos necesarios para el cálculo del grado de pertenencia de una clase a la superclase y de un objeto a la clase. En la Tabla 2.5 se resumen los principales aportes del modelo.

Nivel	**Descripción**
Atributos	Permite especificar rangos difusos para cada atributo, dividiéndolos en *rango de valores permitidos* y *rango de valores típicos.* El primero es un conjunto difuso formado por los *valores permitidos* en el atributo y el grado de posibilidad del valor. El segundo es también un conjunto difuso de los valores más o menos *típicos*, en la mayoría de los casos este conjunto esta formado por el núcleo del *rango de valores permitidos.* Se definen las *clases necesarias* y las *clases típicas.* Las primeras formadas por el producto cartesiano de de los *valores permitidos* y la segunda por los *rangos típicos.* Al trabajar con una instancia aparecen los conceptos de *rango posible* y *rango creíble* para los atributos del objeto.
Relaciones de Instancia	Una instancia tiene un grado de pertenencia con respecto a su clase, este grado se determina midiendo el grado de inclusión del *rango de valores posibles* de cada atributo del objeto con el *rango permitido* definido en la clase.

Relaciones de Herencia	Es posible determinar el grado de pertenencia de una clase a su superclase, calculando el grado de inclusión del *rango permitido* de los atributos de la clase con respecto al *rango permitido* de los atributos de la superclase.
Definición	En el modelo se da la posibilidad de definir herencias parciales entre las clases definiéndose tres tipos de herencia: típica, atípica y normal. C hereda de forma típica de D: los *rangos posibles* de los atributos de C se heredan de los *rangos típicos* de D C hereda de forma atípica de D: los *rangos posibles* de los atributos de C se heredan de la intersección de los *rangos posibles* de D con el complemento de los *rangos típicos* de D. Esta herencia es la que afecta el grado de inclusión entre clases. C hereda de forma normal de D: los rangos de los atributos de C se heredan de los rangos de D. No soporta la herencia ponderada por el usuario.

Conducta	En el modelo se da la posibilidad de definir herencias parciales entre las clases definiéndose tres tipos de herencia: típica, atípica y normal. C hereda de forma típica de D: los *rangos posibles* de los atributos de C se heredan de los *rangos típicos* de D C hereda de forma atípica de D: los *rangos posibles* de los atributos de C se heredan de la intersección de los *rangos posibles* de D con el complemento de los *rangos típicos* de D. Esta herencia es la que afecta el grado de inclusión entre clases. C hereda de forma normal de D: los rangos de los atributos de C se heredan de los rangos de D. No soporta la herencia ponderada por el usuario.

Cuadro 2.5: Modelo J-P Rossaza

Es importante remarcar que para el cálculo de la pertenencia de una clase a la superclase y de un objeto a la clase se determina el grado de inclusión entre ellos por medio de la siguiente ecuación.

$$N\left(B|A\right) = Inf_{u \in U}\left\{I\left(\mu_A\left(u\right), \mu_B\left(u\right)\right)\right\}$$

Donde A es la clase y B la super clase.

Para cálculo de la pertenencia de una clase a la superclase se utilizaran los *rangos permitidos* definidos en cada clase. Y para obtener la pertenencia de un objeto a una clase se mide el grado de inclusión del *rango de valores posibles* de cada atributo del objeto con el *rango permitido* definido en la clase.

2.4.6. El modelo de N. Marín et al.

En [Mar01] N. Marín, se basa inicialmente en el modelo de Vila et. al. [VCMP95, VCMP96a, VCMP96b, VCMP97, VCMP98] y propone un modelo que analiza la presencia de la vaguedad en los cinco niveles, convirtiéndose en uno de los modelos más completos que se pueden encontrar en la literatura. Inicialmente el modelo se centra en el nivel de

atributos, estudiando las diferentes formas en que los valores de un atributo pueden tener imprecisión y la forma de afrontar la falta de certeza sobre la veracidad de los mismos. La igualdad de estado entre objetos se realiza utilizando relaciones de semejanza definidas sobre los dominios básicos. El modelo es capaz de operar sobre objetos complejos.

Se utiliza una extensión difusa para las clases en el *nivel de relaciones de instancia* y se interpreta al mismo tiempo el significado que puede tener esto desde un punto de vista semántico. En el *nivel de relaciones superclase-clase* el estudio se enfoca hacia los procesos de generalización y de especialización. También se aporta la forma de representar la incertidumbre cuando afecta a la propia definición de la estructura de los objetos y de las clases, analizando el sentido que puede tener la falta de certeza cuando aparece en estos niveles.

En la Tabla 2.6 se resumen los principales aportes del modelo.

Una de las principales aportaciones de este modelo es la definición del concepto de *tipo difuso* con el objetivo de superar algunas limitaciones del concepto clásico de tipo. Este concepto es utilizado desde el punto de vista estructural y de comportamiento, adaptándose los mecanismos de instanciación y herencia característicos de la orientación a objetos.

2.4.7. Modelo de Guy de Tré et al. Con conjuntos difusos de nivel-2.

Con el objetivo de modelar imperfecciones en los datos, que no pueden ser tratadas con conjuntos difusos de primer nivel Guy de Tré y R. Caluwe proponen una técnica para el modelado de bases de datos orientadas a objetos difusas, basado en el concepto de çonjuntos difusos de nivel 2"[TC03]. En este trabajo se define un concepto genérico de tipo para realizar una descripción de las características comunes de una colección de datos en bases de datos orientadas a objeto difusas. El tipo propuesto se ajusta a los principios reflejados en el *ODMG* [GCV93] y en las llamadas *reglas de oro* de Atkinson [Atk90] para el modelado orientado a objeto no difuso. Como son tratados los diferentes niveles de imprecisión los tenemos en la Tabla 2.7.

2.5. Bases de Datos Objeto-Relacional Difusa.

Basado en los trabajo de Vila[VCMP98], Marín [Mar01] y Medina [MPM94] se presenta la implementación de un framework para manipular objetos difusos en un ambiente de bases de datos objeto - relacional [CJ04], para la implementación se propone el Sistema

Nivel	Descripción
Atributos	Se pueden definir dominios con compartimiento difuso y luego una propiedad del objeto puede definirse sobre el dominio creado. Da soporte para varios tipos de dominios: atómicos con y sin representación semántica asociada, y dominios con sentido conjuntivo. Es posible asignar grado de incertidumbre al valor de un atributo difuso y a la relación existente entre objetos se suaviza
Relaciones de Instancia	Implementa la posibilidad de que un objeto puede pertenecer a una clase con un grado, por medio de un atributo en la clase que representa la pertenencia del objeto a la clase. De esta forma queda representada la vaguedad que pueda existir en jerarquías de clases, tanto construidas a partir de una especialización como de una generalización. En todos los casos el modelo se auxilia de atributos valuados en el intervalo [0,1] definidos sobre las clases para representar la pertenencia de un objeto a una clase.
Relaciones de Herencia	Es posible definir el grado de pertenencia de una clase a una superclase, teniendo en cuenta la generalización y especialización de clases.
Definición	Este nivel se garantiza por la incorporación del concepto de Tipo Difuso que permite definir incertidumbre a nivel de definición de una clase.
Conducta	La posibilidad de definir un nivel para una instancia de un objeto con lo cual incorporan solo aquellos atributos del nivel especificado, provoca que no todos los métodos definidos en el tipo sean necesarios para el objeto y por lo tanto se garantiza incertidumbre también en la definición de las clases.

Cuadro 2.6: Modelo N. Marín

Nivel	**Descripción**
Atributos	Se pueden definir dominios difusos para los datos. Estos dominios se define usando dos capas, una capa interna que son los todos los valores validos para un tipo y una capa externa representada por conjuntos difusos de nivel-2 definidos sobre la capa interna. El grado de pertenencia de la capa exterior es interpretado como grado de incertidumbre y es usado para modelar la incertidumbre sobre los datos El grado de pertenencia de la capa interior tiene varias interpretaciones: grado de incertidumbre, cuando es usado para representar la imprecisión y la vaguedad, grado de preferencia, cuando se usa para denotar una intensidad de preferencia en favor de sus elementos asociados. La validez de la interpretación depende del contexto en el cual el tipo generalizado es usado. Se representan valores para los atributos tanto con significado disyuntivo como conjuntivo.

Cuadro 2.7: Características de modelo Guy de Tré

Gestor de Bases de Datos Oracle, utilizando sus características de orientación a objetos. Este modelo define un grupo de operadores para almacenar, manipular y consultar datos difusos implementando un FORDBMS. Algunas extensiones a este modelo se reflejan en el tópico aplicaciones de este capítulo.

2.5.1. Aplicaciones

Varias aplicaciones sobre los modelos de bases de datos difusas se han publicado en el ámbito científico, en muchos de los casos las aplicaciones tienen que hacer extensiones a los modelos base. En una búsqueda en las principales revistas y eventos de los últimos años se encontraron los siguientes trabajos.

1. **Manipulación de objetos difusos en XML:** En este trabajo [LFKL03] se presenta una aproximación a la manipulación de objetos difusos en *XML*, brindando un conjunto de reglas para representar modelos y especificaciones de objetos difusos en esquemas y documentos *XML* respectivamente

2. **Sistemas de información geográfica:** En [CF00] Cross utiliza la tecnología de bases de datos orientadas a objetos y la teoría de conjuntos difusos para representar el conocimiento en sistema de información geográfica (GIS). En el trabajo se establece un modelo de objetos difusos estándar a partir de dos niveles. Un primer nivel define cuál es la vaguedad que puede aparecer en los objetos que describen la información, definiéndose objetos difusos. El segundo nivel tiene que ver con cuáles son las vaguedades o ambigüedades en la definición de tipos. Además se trabaja con las relaciones clase/subclase. Luego se especifica la implementación de este tipo de bases de datos en dos prototipos *FuzzyCOOL* como sistema experto y en *FuzzyVersant* como interfaz comercial con un sistema de bases de datos orientado a objetos. Toda la parte de modelado se basa en ODMG.

3. **Representación de información espacial:** En el trabajo [BC02], Bordogna et al. propone una representación consistente de la información espacial por medio de la teoría de las posibilidades y la teoría de conjuntos difusos. Se sugiere una aproximación para representar y manipular información en una base de datos difusa orientada a objetos. Se parte del concepto de variable lingüística y la teoría de las posibilidades para manipular información vaga y con incertidumbre en un contexto de bases de datos. Se especifican los atributos vagos y con incertidumbre como subconjuntos difusos

del dominio de los atributos. La utilización de variables lingüísticas para describir las propiedades de entidades espaciales es el mayor aporte de este trabajo.

4. **Información multimedia:** El uso de bases de datos orientadas a objeto difusas aplicadas en areas de multimedia puede encontrase en [MBS02]. En esta investigación utilizan subconjuntos difusos para trabajar la incertidumbre en imágenes particularmente en las relaciones topológicas y espaciales existentes entre los objetos de la imagen.

5. **FoodBi** [Mar01, MBPV03]: es una interfaz gráfica para la creación y gestión de esquemas de bases de datos orientadas a objetos difusas. Con esta aplicación es posible construir una jerarquía de clases con tipos difusos que utilicen dominios para el manejo de la vaguedad.

6. **FOODB en POET**: en este trabajo se presenta el diseño e implementación de una base de datos orientada a objetos difusa utilizando el sistema de bases de datos orientadas a objeto *POET*. El sistema propuesto permite consultas precisas y difusas especificadas por medio de un lenguaje natural. Es capaz de almacenar datos difusos y datos precisos. Se presenta una interfaz gráfica para ejecutar consultas difusas, entrar datos difusos y examinar el resultado de las consultas difusas [SLK01].

7. **Extracción de tipos**: En [Mar01] N. Marín propone formas para identificar tipos difusos a partir de una estructura que representa un conjunto de datos. Se proponen dos herramientas una en forma de grafo, *grafo de inclusión*, para la representación de la información estructural y un algoritmo para la extracción de la definición de un tipo difuso.

8. **Sistema para recuperar el color de una imagen**: La propuesta presentada en [BMCMSH06] utilizan un gestor de bases de datos objeto relacional difusas y un método de descripción de imágenes para consultar de forma rápida información referente al color de una imagen como lo es el color predominante. En este trabajo se modifica la forma de calcular el Grado de Inclusión Guiado por semejanza proponiendo una que tiene en cuenta el peso del color menos dominante con el más dominante, Grado de Inclusión de Semejanza Modificado.

9. **ImmoSoftWeb**: Una aplicación Web construida sobre un modelo de bases de datos relacional difuso que permite expresar los atributos usando datos difusos y consul-

tarlos por medio de consultas difusas adaptadas a los requerimientos del usuario [BCMP04].

10. **Trabajo con datos temporales**: En [CGMP07] se utiliza un modelo de base de datos relacional difusa para representar y consultar datos de tipo tiempo por medio de intervalos difusos de fechas.

11. **Lenguaje FSQL**: Una aplicación del lenguaje FSQL al sector turístico y otra a una empresa de fabricación da cartulinas se presenta en [Gal02] y [JUGZ05] , en estos ejemplos se trabaja con información difusa la cual es almacenada en los atributos de sus productos o materias primas y puede ser consultada por medio de consultas difusas flexibles.

En estos tiempos se evidencia un grupo de areas de investigación donde el uso de bases de datos para almacenar y manipular información con imperfección es muy importante. Dentro de estas áreas esta la minería de datos, los sistemas de información geográficos de los próximos años de la lógica difusa y datos espaciales, y recuperación de información difusa [BKP05].

2.6. Conclusiones

Los modelos de bases de datos han evolucionado con los años, adaptándose a los nuevos requerimientos de esa forma fueron desde las bases de datos jerárquicas hasta las hoy bases de datos objeto - relacional. Estas últimas por las características que tiene de usar tanto el modelo relacional como la orientación a objeto, son el modelo del presente y del futuro en el desarrollo de sistemas con soporte sobre bases de datos. Uno de los inconvenientes de los primeros modelos de bases de datos fue la de no permitir representar información imperfecta, para dar solución a esta problemática varios estudioso hicieron sus propuestas, un grupo de ellas se reflejaran en el capítulo 2, una de las técnicas utilizadas para este fin es la teoría de subconjuntos difusos y la lógica difusa que con sus posibilidades de permitir el modelado de problemas con datos afectados por imperfección brinda un marco perfecto para solucionar este tipo de problemas.

En este capítulo se han presentado brevemente el estado del arte en materia de bases de datos difuso, haciéndose especial énfasis en la orientación a objeto. De las propuesta sobre el modelo relacional se presento un poco más de detalle del modelo GEFRED y

la existencia del lenguaje FSQL asociado a ese modelo, estas dos propuestas son una de las bases para algunas de las funcionalidades incluidas en el modelo presentado en esta tesis. Se mostraron los principales modelos de las bases de datos orientadas a objeto difusa publicados, describiendo en cada caso como tratan los diferentes niveles donde puede encontrarse vaguedad en este tipo de sistemas. Se evidencia que el modelo de N. Marín [Mar01] es uno de los más completos y con mayores posibilidades de implementación y aplicación a problemas reales. El listado de varias aplicaciones realizadas sobre estos tipos de bases de datos muestran su total aplicación en diferentes ramas de la investigación científica.

Capítulo 3

Modelo Teórico

Este capítulo describe los principales fundamentos teóricos del modelo desarrollado en esta investigación. Este modelo viene a completar propuestas anteriores de modelos de bases de datos difusas orientadas a objetos, en particular la propuesta realizada por Marín y otros [Mar01, VCMP98].

Como fue reflejado en el tópico anterior, en una base de datos difusa se puede encontrar vaguedad en cinco niveles: nivel de atributos, nivel de relaciones de instancia, nivel de relaciones de herencia, nivel de definición y nivel de conducta. Los trabajos realizados en esta investigación se centran en dos niveles, el nivel de atributos y el nivel de de definición. Estos dos niveles, según el autor de esta investigación, son considerados los más importantes desde el punto de vista práctico.

Se comienza por el nivel de atributo, en el cual se permite representar y manipular información con naturaleza imperfecta. El modelo incorpora el uso de dominios difusos con diferentes naturalezas que pueden ser usados para representar los atributos de una clase. Se completan propuestas recientes al incorporar el uso de diferentes operadores difusos para cada tipo de dominio.

El uso de atributos definidos sobre estos dominios permite trabajar con objetos difusos complejos sobre los cuales se define el estado de igualdad entre objetos. Se incorporan además operadores para comparar dos objetos difusos complejos, tratando la igualdad en base a distintos criterios.

Los modelos de bases de datos difusas orientados a objetos anteriores, utilizan para comparar las propiedades de cada objeto la igualdad difusa. Como veremos, a partir de las extensiones realizadas en esta investigación, es posible definir perspectivas de compa-

ración para cada objeto. Cada perspectiva estará definida por un nombre, el peso y el operador difusos que se utilizará al comparar cada atributo del objeto. Las perspectivas de comparación se generalizan con un enfoque recursivo que permite operar sobre objetos complejos.

Un tipo especial de objeto con atributos difusos son los tipos difusos, concepto fundamental para trabajar con el nivel de definición. Para el modelo, los tipos difusos se comportan como objetos difusos complejos pero con la particularidad que definen alfacortes al crear una instancia. Se proponen extensiones de la igualdad difusa para comparar instancias de tipos difusos. Se define el marco teórico de la comparación entre objetos de un Tipo Difuso y se incorporan tres estrategias en la comparación: una estrategia optimista, otra pesimista y la tercera donde se ignoran los atributos no existentes. El elemento esencial de estas estrategias es cómo interpretar la comparación entre objetos pertenecientes a niveles diferentes de un Tipo Difuso. Los tipos difusos además pueden ser utilizados como dominio para un atributo de un objeto y definir sobre ellos perspectivas de comparación.

Finalmente el modelo incorpora el uso de atributos inferidos difusos. Partiendo de la existencia de atributos derivados en una base de datos y teniendo en cuenta que una de las principales aplicaciones de la teoría de los subconjuntos difusos y la lógica difusa es su utilización en sistemas difusos, se incorpora al modelo la posibilidad de definir atributos cuyo valor se obtiene a partir de un sistema difuso formado por reglas difusas con antecedentes definidos utilizando los propios atributos del objeto.

Con las extensiones propuestas en esta investigación para el modelo de bases de datos difusas orientadas a objetos de Marín y otros, se ha logrado un modelo teórico con un mayor número de prestaciones en un ambiente de bases de datos difusas. El modelo se enriquece desde el punto de vista conceptual al poder representar atributos inferidos difusos, dando soporte al uso de sistemas de reglas difusas en bases de datos. Es de gran valor para el usuario la incorporación de las características que permiten definir y trabajar con perspectivas de comparación para los objetos difusos complejos y el uso de diferentes estrategias al comparar tipos difusos.

3.1. Nivel de atributos

El primer nivel donde puede encontrarse vaguedad en una base de datos orientada a objetos es en el nivel de atributos. Cuando se analiza la naturaleza del valor de un atributo es posible encontrar valores precisos o valores afectados por algún tipo de imperfección.

La imperfección del valor de un atributo puede estar dada por una mala definición del atributo, con lo que se dice que el valor es impreciso, una afectación en la veracidad de la información por existir incertidumbre o la combinación de ambos casos [Rus86]. El modelo que se presenta en esta investigación centra su estudio en el primer tipo de información imperfecta que puede tener un atributo en una base de datos orientada a objetos.

La imprecisión inherente al valor de un atributo está dada, fundamentalmente, por la naturaleza del dominio del atributo. Por lo tanto el modelo de datos que se utilice debe permitir trabajar con las diferentes interpretaciones que puedan tener los dominios. En modelos anteriores de bases de datos orientadas a objetos difusas se permite definir valores por medio de distribuciones de posibilidad mediante el uso de variables lingüísticas [BLP94] y se trabaja con semánticas conjuntivas y disyuntivas para los valores que puede tener un atributo [GBP91, YK97].

El modelo propuesto en esta investigación permite manipular dominios definidos sobre un referencial de valores o sobre un referencial de objetos (ver Figura 3.1). De esta forma, un atributo puede tener un referencial de valores o un referencial de objetos. Si el dominio de un atributo es difuso entonces la naturaleza del referencial básico será difusa.

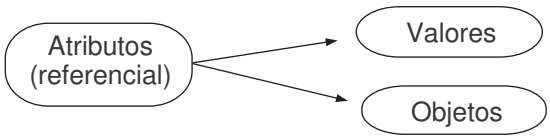


Figura 3.1: Referencial de un atributo.

Los valores imprecisos en el modelo pueden estar representados por una etiqueta lingüística, que dispondrá de diferentes interpretaciones en dependencia de la semántica del dominio donde se definen, por un valor perteneciente al universo sobre el que se define el dominio o por un subconjunto difuso. El modelo define y utiliza funciones de semejanza para determinar el parecido entre dos valores imprecisos. Con estas características se incorpora la posibilidad de definir la estructura de un atributo, más la capacidad de cada atributo de compararse [Mar01].

Viendo en más detalle los dominios soportados por el modelo, estos se pueden clasificar según la cardinalidad del valor (ver Figura 3.2), según la existencia o no de vaguedad (véase Figura 3.3) y según la forma en que se introduce el valor (ver Figura 3.4).

De esa forma a partir de la cardinalidad del dominio se pueden distinguir dos tipos de dominio: los dominios atómicos y los dominios conjuntivos (ver Figura 3.2). Los dominios

atómicos son aquellos en los cuales el atributo sólo puede tomar un valor y los conjuntivos permiten que el atributo tome como valor un subconjunto de valores.

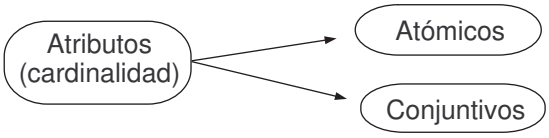


Figura 3.2: Cardinalidad de los atributos.

En el modelo se pueden representar tanto atributos con valores precisos como con valores difusos (véase Figura 3.3). Si el valor es difuso se dispondrá de distintos dominios para su representación, cada uno con una naturaleza diferente. Atendiendo a esto último se tienen dominios sin representación semántica asociada y dominios con representación semántica asociada. Dentro del segundo grupo de dominios es posible, en dependencia del referencial sobre el que se define, tener dos tipos: los de referencial finito y los de referencial continuo. En los dominios con referencial continuo se definen funciones de pertenencia y solo en este tipo de dominio no es posible definir el referencial sobre objetos.

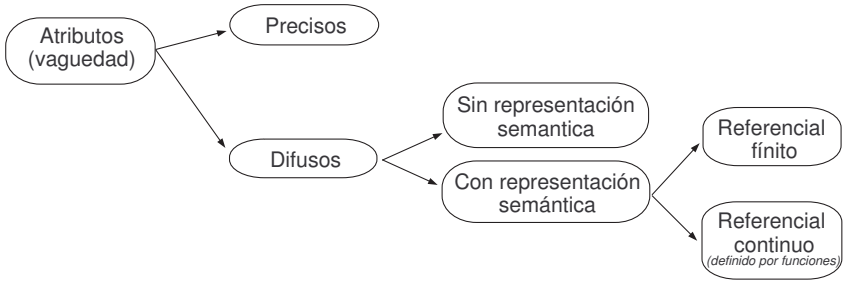


Figura 3.3: Vaguedad en los atributos.

Finalmente el valor de un atributo puede ser introducido, en el modelo, de forma implícita o explícita. De forma implícita el usuario introduce el valor directamente. Si es de forma explícita el valor será inferido utilizando un sistema de reglas difusas (véase Figura 3.4).

Un resumen de los tipos de dominios difusos soportados por el modelo se muestra en la figura 3.5. La definición de estos dominios parte de trabajos anteriores [Mar01, VCMP98]. Estos dominios, como se mencionó inicialmente, podrán tener un referencial sobre valores o sobre objetos. En la mayoría de estos dominios se define el proceso de comparación y pueden ser utilizados como atributos definidos de forma implícita.

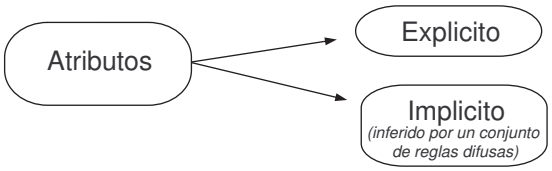


Figura 3.4: Valor de un atributo.

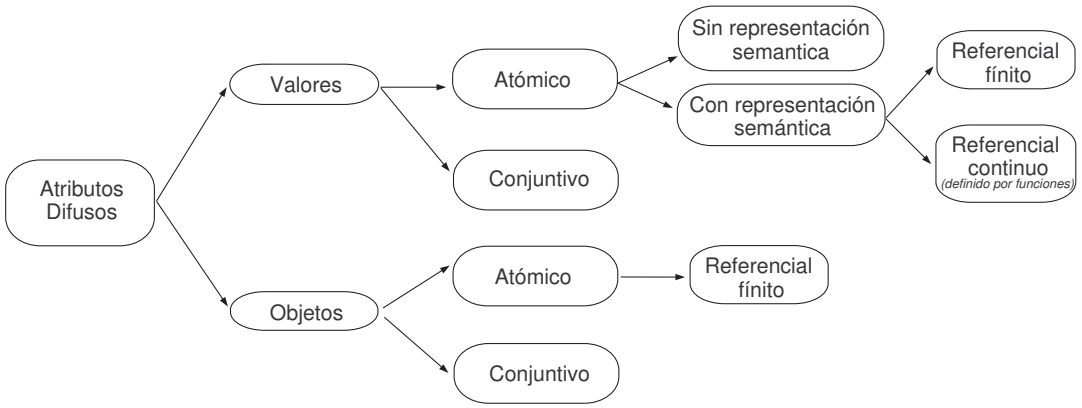


Figura 3.5: Resumen de dominios soportados por el modelo.

Una vez presentados cómo pueden clasificarse los dominios difusos soportados por el modelo se realizará un recorrido por los diferentes tipos de dominios. Se presenta un modelo teórico resultante de añadir a propuestas anteriores algunas mejoras que lo hacen más completo en los niveles de atributo y de definición.

El análisis comienza con los dominios definidos sobre un referencial de valores(ver Figura 3.5). Algunos ejemplos reales de valores para un atributo con estas características se presentan en la siguiente lista:

1. La calidad de un producto: en la mayoría de los casos es difícil de buscar una representación precisa de la calidad por medio de algún valor numérico. Es mucho más natural representarla por medio de palabras como *alta*, *media* o *baja*.

2. La definición de la estatura de una persona: En ocasiones se necesita almacenar la estatura de un grupo de personas. Dependiendo de la naturaleza del problema, se puede conoce la estatura de una persona con exactitud, pero existirán otros casos donde, por varias razones, sólo se dispondrá de una representación lingüística de esa estatura, por ejemplo la etiqueta *alto*. La etiqueta *alto* estará definida sobre un

dominio finito de edades. En este caso, el atributo podrá tomar valores dentro de los números reales o simplemente ser una etiqueta lingüística.

3. La dureza de un mineral: la dureza de una piedra es medida por medio de una escala finita en el rango de [1,10]. Sin embargo se suele hablar de una piedra *muy dura, dura, media, blanda* o *muy blanda*. Estas últimas etiquetas están representadas por distribuciones de posibilidad sobre un dominio finito que son los posibles valores de dureza (1, 1.5, 2, 2.5, 3, 3.5,. . . , 9.5, 10).

4. Se desea representar los idiomas que domina un investigador. Un investigador puede dominar varios idiomas y cada uno con un determinado nivel que varía de idioma en idioma. En este caso el dominio básico es el conjunto de idiomas {*inglés, francés, ruso, italiano, español*}. Si se define un atributo en una tabla *Investigador* para representar los idiomas que domina un estudiante, pudiera tener los siguientes valores:

```
investigador1.idioma = {1.0/español/ + 0.6/inglés + 0.4/frances}
investigador2.idioma = {1.0/español + 0.7/inglés + 0.3/frances}
```

En estos ejemplos se muestran las diferentes naturalezas de imprecisión en la definición del dominio de un atributo. En el primer ejemplo es difícil encontrar un dominio subyacente sobre el cual expresar las etiquetas definidas para el atributo *calidad*, por lo que se está en presencia de un *dominio atómico sin representación semántica asociada*.

En el segundo y tercer caso, se está en presencia de dominios atómicos con representación semántica asociada, cuyos valores pueden ser, en el caso de la estatura, la estatura precisa de una persona, o una etiqueta lingüística de un conjunto previamente definido para este dominio. En el caso de la dureza el posible valor de un atributo será una de las etiquetas definidas o alguno de los valores de dureza existentes.

El cuarto ejemplo el valor que puede tomar el atributo *idioma* de un investigador está dado por un subconjunto difuso definido sobre un referencial de valores. El referencial en este caso será el conjunto de nombre de idiomas existentes. Por lo tanto este atributo debe estar definido sobre un dominio con cardinalidad conjuntiva y referencial formado por valores, un *dominio conjuntivo de valores*.

A partir de estos ejemplo, se puede precisar que en los dominios difusos definidos sobre un referencial de valores se pueden encontrar dominios con una cardinalidad atómica y dominios con cardinalidad conjuntiva, resumiéndose de la siguiente forma:

- Cardinalidad atómica.
  - Dominio sin representación semántica asociada.
  - Dominios con representación semántica asociada.
    - Con referencial continuo.
    - Con referencial finito.
- Cardinalidad conjuntiva.
  - Conjuntivo de valores.

Cada uno de estos dominios será presentado a continuación.

3.1.1. Dominios atómicos sin representación semántica

Para este tipo de dominio (ver Figura 3.6) se define un dominio básico B formado por un conjunto de etiquetas lingüísticas. Al no existir ninguna representación semántica subyacente asociada al dominio, no se especifica distribución de posibilidad para las etiquetas. Por lo tanto, para poder gestionar la imprecisión representada por el dominio, se utiliza una relación de semejanza definida sobre el dominio básico que permitirá realizar comparaciones entre dos valores del dominio.

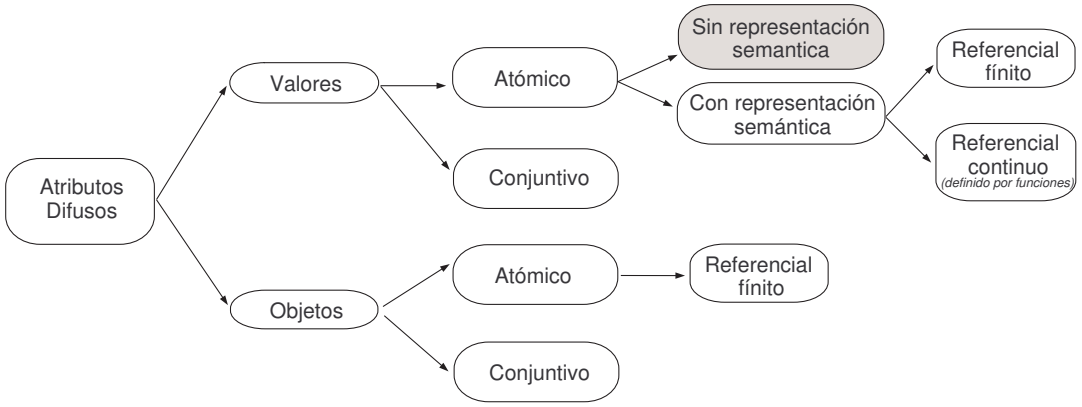


Figura 3.6: Dominio atómico sin representación semántica.

Sobre el dominio básico B se define una relación de semejanza. Por ejemplo si el conjunto R puede tomar un sólo valor definido entre las etiquetas *(A, B, C, D)*, se define para estas

etiquetas una relación de semejanza de forma explícita representada de la siguiente forma (ver Tabla 3.1).

Cuadro 3.1: Relación de semejanza entre etiquetas sin representación semántica.

	A	*B*	*C*	*D*
A	1	0.7	0.4	0.2
B		1	0.6	0.3
C			1	0.4
D				1

Si se desea calcular la semejanza entre dos de los valores definidos, sólo será necesario observar la semejanza definida entre ellos. Esta relación de semejanza daría el soporte al operador para determinar el parecido entre dos valores.

Este tipo de dominio se utiliza al representar casos como el de la *calidad de un producto*, señalado en el apartado anterior. En esta situación no es posible definir un dominio básico para representar la calidad de un producto, por lo que en todo momento el valor del atributo estará dado por una de las etiquetas previamente definidas. Se define entonces una relación de semejanza entre las etiquetas para gestionar la imprecisión inherente al conjunto de etiquetas, permitiendo establecer comparaciones entre las etiquetas. Esta relación de semejanza la define el usuario que va a utilizar el dominio (ver Ejemplo 3.1).

Ejemplo 3.1. *Medir la calidad de un producto. La calidad de un producto se puede reflejar por medio de las etiquetas lingüísticas alta, media, baja. Estas etiquetas son el dominio del atributo calidad sobre el que se define una relación de semejanza (ver Tabla 3.2).*

Cuadro 3.2: Relación de semejanza del atributo *calidad*

	alta	*media*	*Baja*
Alta	1	0.8	0.2
media		1	0.4
Baja			1

Para determinar la semejanza entre dos etiquetas se tiene en cuenta la matriz de semejanza definida (como se muestra en el Ejemplo 3.1 y la Tabla 3.2). Si se quiere tener la semejanza que existe entre una calidad *alta* y una calidad *media* se busca la celda donde se interceptan ambas etiquetas y esa será la semejanza entre ambas, en este caso *0,8*.

Es posible aumentar el dominio con nuevas etiquetas lingüísticas. En este caso el nuevo valor debe estar representado por un subconjunto difuso de las etiquetas previamente definidas para el dominio, a partir de esto se modifica la relación de semejanza mediante la ecuación 3.1,[Mar01].

$$\forall x \epsilon D, \mu_S(l, x) = max_{y \epsilon D}(\mu_S(y, x) \otimes \mu_L(y)) \tag{3.1}$$

Donde: D es el conjunto de etiquetas;

S la relación de semejanza previamente definida;

L un subconjunto difuso sobre D que define la nueva etiqueta;

Otros dominios, sí tienen una representación semántica asociada a cada valor, es decir, existe un dominio subyacente perfectamente definido. En este caso se mezclan las etiquetas lingüísticas definidas y los valores del dominio básico para dar el dominio real del dominio difuso definido. En el siguiente apartado se describen las características de este tipo de dominio.

3.1.2. Dominios atómicos con representación semántica asociada

Estos dominios, como su nombre indica, tienen un dominio básico asociado sobre el cual se define un conjunto de etiquetas lingüísticas representadas por distribuciones de posibilidad que permiten representar valores con imprecisión. La interpretación del valor es mono valuado, es decir el objeto podrá tomar un valor perteneciente al dominio básico o podrá ser una de las etiquetas lingüísticas definidas. Nunca podrá ser una combinación de estos valores. Ejemplos de este tipo de dominio son los ejemplos de la dureza de un material (ver Ejemplo 3.2) y la estatura de una persona (véase Ejemplo 3.3).

Ejemplo 3.2. *La dureza de cualquier mineral se mide por medio de una escala, la escala de dureza de Mohs, desarrollada por Friedrich Mohs hace aproximadamente 200 años. La escala tiene un rango entre 1 y 10, siendo una dureza 10 para los minerales más duros y una dureza 1 para minerales más blandos, logrando una clasificación de los minerales por sus características físicas. Si se analiza la siguiente expresión "la kunsita es una piedra dura". se puede notar que se está en presencia de un dominio con imprecisión en su definición. El dominio básico de la dureza es el conjunto finito* $B = \{1, 1.5, 2, 2.5, 3, \ldots, 9.5, 10\}$ *y se pueden añadir los valores imprecisos* {*"Muy duro", "duro", "medio", "blando", "muy blando"*}*. Se tiene para ello las siguientes distribuciones de posibilidad:*

Muy Duro = {*0.3/7, 0.5/7.5, 0.7/8, 0.8/8.5, 1.0/9, 1.0/9.5, 1.0/10*}
Duro = {*0.3/6, 0.6/6.5, 1.0/7, 1.0/7.5, 1.0/8, 1.0/8.5,0.7/9,0.5/9.5,0.2/10*}
Medio = {*0.2/2.5, 0.4/3, 0.6/3.5, 0.8/4, 1.0/4.5, 1.0/5, 1.0/5.5, 1.0/6, 1.0/6.5, 0.8/7, 0.6/7.5, 0.4/8, 0.2/8.5*}
Blando = {*0.4/1.5,0.8/2, 1.0/2.5, 1.0/3, 1.0/3.5, 1.0/4, 0.7/4.5, 0.4/5*}
Muy Blando = {*1.0/1, 1.0/1.5, 1.0/2, 0.8/2.5, 0.6/3, 0.5/3.5, 0.3/4*}

Ejemplo 3.3. *Este tercer ejemplo representa la estatura de una persona, el dominio básico es el conjunto de números enteros, pero además se pueden definir etiquetas lingüísticas que permitan expresar la estatura de una persona con imprecisión. En este caso las etiquetas estarán expresadas por medio de funciones trapezoidales, definiéndose las siguientes etiquetas: Bajo = (0, 0, 150, 160), Medio = (150, 160, 170, 180), Alto = (170, 180, 300, 300) (véase la Figura 3.7).*

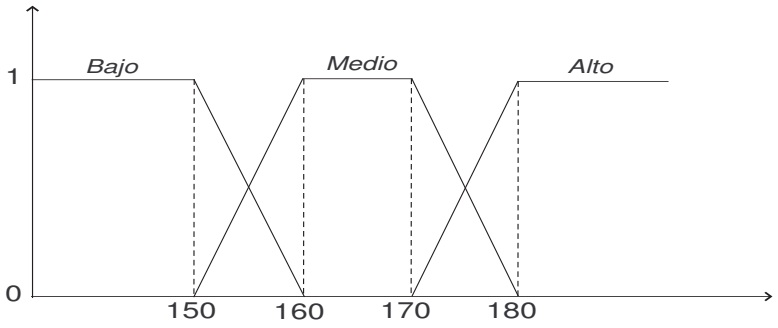


Figura 3.7: Etiquetas lingüísticas para el atributo *estatura*.

Analizando estos ejemplos se pueden dividir los dominios atómicos con representación semántica asociada, en dependencia de cómo se representan las distribuciones de posibilidad, en dos tipos:

- Dominios atómicos con representación semántica asociada, definidos sobre un referencial continuo (ver Ejemplo 3.3).
- Dominios atómicos con representación semántica asociada definidos sobre un referencial finito (ver Ejemplo 3.2).

Estos dominios necesitan de una relación difusa como relación de semejanza entre los posibles valores que puede tomar. Cómo representar las relaciones difusas en cada tipo

de dominios atómicos con representación semántica es lo que se muestra en los siguientes apartados.

3.1.2.1. Dominios atómicos con referencial continuo.

En este dominio (ver Figura 3.8) la distribución de posibilidad de las etiquetas lingüísticas se definen por medio de funciones. Esta forma de representación puede ser utilizada cuando no sea adecuado utilizar una definición extensiva de la distribución de posibilidad. Comúnmente se puede utilizar cualquier tipo de función para representar la distribución de posibilidad. Las funciones lineales son las más utilizadas en este sentido[Gal99]. La razón está dada porque estas funciones son más simples de trabajar desde el punto de vista computacional y si se tiene en cuenta que se está actuando en un ambiente difuso, donde la información tiene una naturaleza imprecisa, es muy recomendable trabajar con funciones sencillas. El uso de funciones trapezoidales como una forma de generalizar el resto de las funciones es aceptado como suficiente por la comunidad de investigadores. Por tales razones, este dominio basa la representación de las etiquetas lingüísticas en funciones trapezoidales.

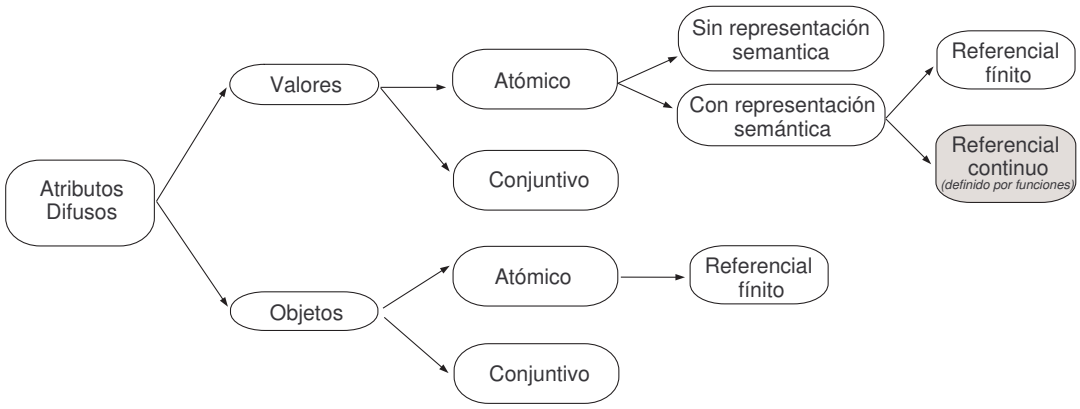


Figura 3.8: Dominios atómicos con referencial continuo.

Las funciones trapezoidales son representadas por medio de cuatro atributos (a, b, c, d), (ver Figura 3.9). En dependencia de los valores asignados a cada atributo es posible trabajar con funciones trapezoidales y triangulares, simétricas o no.

Desde el punto de vista funcional, la distribución de posibilidad de una etiqueta A representada por medio de una función trapezoidal, se expresa mediante la ecuación (3.2)

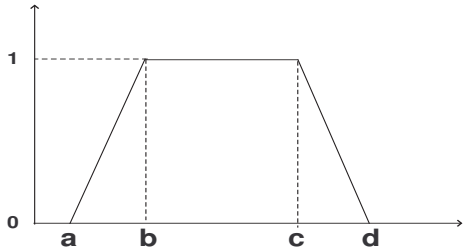


Figura 3.9: Función Trapezoidal.

definida por rangos. En la ecuación los argumentos *a, b, c* y *d* son los parámetros de la función trapezoidal mientras que x es un valor del dominio subyacente.

$$\mu_A(x) = \begin{cases} 0 & x \leq a \vee x \geq d \\ \left(\frac{x-a}{b-a}\right) & a < x < b \\ 1 & b \leq x \leq c \\ \left(\frac{d-x}{d-c}\right) & c < x < d \end{cases} \tag{3.2}$$

El ejemplo de la *estatura* de una persona (ver Ejemplo 3.3) es una información que cuando se analiza es conveniente representar sus valores imprecisos utilizando etiquetas lingüísticas con distribuciones de posibilidad representadas por funciones. En este caso se define el dominio básico B como el conjunto de números reales en el rango [0,300] y además se utiliza un conjunto L de etiquetas lingüísticas para representar los valores imprecisos de la estatura: $Bajo = (0, 0, 150, 160), Medio = (150, 160, 170, 180), Alto = (170, 180, 300, 300)$ (ver Figura 3.7).

La semejanza entre dos valores del dominio no está representada de forma explícita y debe ser determinada por medio de expresiones. De esta forma, si se desea obtener la semejanza entre un valor x del dominio básico B y una de las etiquetas $l_i \in L$, una de las formas es recurrir a la distribución de posibilidad presentada en la ecuación 3.2.

$$\mu_S(x, l_1) = \mu_{l_1}(x) \tag{3.3}$$

Si se desea determinar la semejanza entre dos etiquetas lingüísticas l_1 y l_2, tal que $l_1, l_2 \in L$ se utilizan también, las distribuciones de posibilidad de las etiquetas mediante la

siguiente ecuación (3.4).

$$\mu_S(l_1, l_2) = \sup_{x \in U} min(\mu_{l_1}(x), \mu_{l_2}(x)) \tag{3.4}$$

donde U es el dominio subyacente y $\mu_{l_1}(x), \mu_{l_2}(x)$ son los grados de pertenencia del valor x a la etiquetas lingüísticas l_1 y l_2 representadas por sus distribuciones de posibilidad. Por medio de estas ecuaciones se puede dar soporte a un operador *FEQ (posiblemente igual difuso)* para este tipo de dominio.

Al existir definida una relación de orden en este tipo de dominio es posible definir otros operadores difusos, además del operador *FEQ*. Como será mostrado, la definición de estos operadores incrementa la potencia semántica en las consultas sobre el modelo.

3.1.2.1.1. Operadores difusos en dominios atómicos con referencial continuo. Debido a las características de este tipo de dominio en el cual existe una relación de orden entre sus valores es posible definir un conjunto de operadores difusos que permiten comparar dos valores de éste tipo o comparar un valor con una constante. Los operadores con los que se trabaja fueron propuestos por Galindo en [Gal05, Gal99]. Las ecuaciones para los cálculos que se utilizan son las propuestas en las bibliografías antes citadas.

Las ecuaciones aplicadas a cada operador parten de comparar dos distribuciones de posibilidad con los siguientes parámetros $A = a_A, b_A, c_A, d_A$ y $B = a_B, b_B, c_B, d_B$ (ver Figura 3.9). Teniendo en cuenta esto, los operadores se definen por medio de las funciones por intervalo mostradas en las tablas 3.3 y 3.4 .

Los operadores MGT y MLT utilizan en su cálculo el parámetro $\mathcal{M}$ que representa la distancia mínima para que dos valores del dominio sean considerados muy separados. Este parámetro debe ser suministrado por el usuario.

Además de poder representar las distribuciones de posibilidad por medio de funciones, el modelo permite utilizar una definición extensiva del conjunto difuso sobre un referencial discreto, este tipo de dominio es el que se presenta en el siguiente apartado.

3.1.2.2. Dominios atómicos con referencial finito.

En este tipo de dominio (ver Figura 3.10) las distribuciones de posibilidad de las etiquetas lingüísticas que representan valores imprecisos se expresan por medio de una definición extensiva del conjunto difuso. Se considera que B es el dominio básico y L el conjunto de etiquetas, de forma que l_i representa una etiqueta lingüística, $l_i \in L$. La distribución de

Cuadro 3.3: Operadores difusos sobre dominios atómicos con referencial continuo.

Operador	**Descripción**	**Ecuación**
FGT	Grado en el que una distribución de posibilidad A es posiblemente mayor difuso que otra B	$FGT(A,B) = \begin{cases} 1 & c_A \geq d_B \\ \frac{d_A - c_B}{(d_B - c_B) - (c_A - d_A)} & c_A < d_B \text{ y } d_A > c_B \\ 0 & d_A \leq c_B \end{cases}$
FGEQ	Obtiene el grado en el que una distribución de posibilidad A es posiblemente mayor o igual difuso que otra B.	$FGEQ(A,B) = \begin{cases} 1 & c_A \geq b_B \\ \frac{d_A - a_B}{(b_B - a_B) - (c_A - d_A)} & c_A < b_B \text{ y } d_A > a_B \\ 0 & d_A \leq a_B \end{cases}$
FLT	Obtiene el grado en el que una distribución de posibilidad A es posiblemente menor difuso que otra B.	$FLT(A,B) = \begin{cases} 1 & b_A \leq a_B \\ \frac{a_A - b_B}{(a_B - b_B) - (b_A - a_A)} & b_A > a_B \text{ y } a_A < b_B \\ 0 & a_A \geq b_B \end{cases}$

Cuadro 3.4: Operadores difusos sobre dominios atómicos con referencial continuo (continuación)

Operador	**Descripción**	**Ecuación**
FLEQ	Obtiene el grado en el que una distribución de posibilidad A es posiblemente menor o igual difuso que otra B.	$FLEQ(A,B) = \begin{cases} 1 & b_A \leq c_B \\ \frac{d_B - a_A}{(b_A - a_A)-(c_B - d_B)} & b_A > c_B \text{ y } a_A < d_B \\ 0 & a_A \geq d_B \end{cases}$
MGT	Obtiene el grado en el que una distribución de posibilidad A es posiblemente mucho mayor difuso que otra B	$MGT(A,B) = \begin{cases} 1 & c_A \geq d_B + \mathcal{M} \\ \frac{c_B + \mathcal{M} - d_A}{(c_A - d_A)-(d_B - c_B)} & c_A < d_B + \mathcal{M} \text{ y} \\ & d_A > c_B + \mathcal{M} \\ 0 & d_A \leq c_B + \mathcal{M} \end{cases}$
MLT	Obtiene el grado en el que una distribución de posibilidad A es posiblemente mucho menor difuso que otra B	$MLT(A,B) = \begin{cases} 1 & b_A \leq a_B - \mathcal{M} \\ \frac{b_B - \mathcal{M} - a_A}{(b_A - a_A)-(a_B - b_B)} & b_A > a_B - \mathcal{M} \text{ y} \\ & a_A < b_B - \mathcal{M} \\ 0 & a_A \geq b_B - \mathcal{M} \end{cases}$

posibilidad de l_i tendrá un soporte finito. El dominio real D queda definido por medio de la unión entre el dominio básico B y el conjunto de etiquetas lingüísticas definido sobre ese dominio L.

$$D = B \bigcup L \tag{3.5}$$

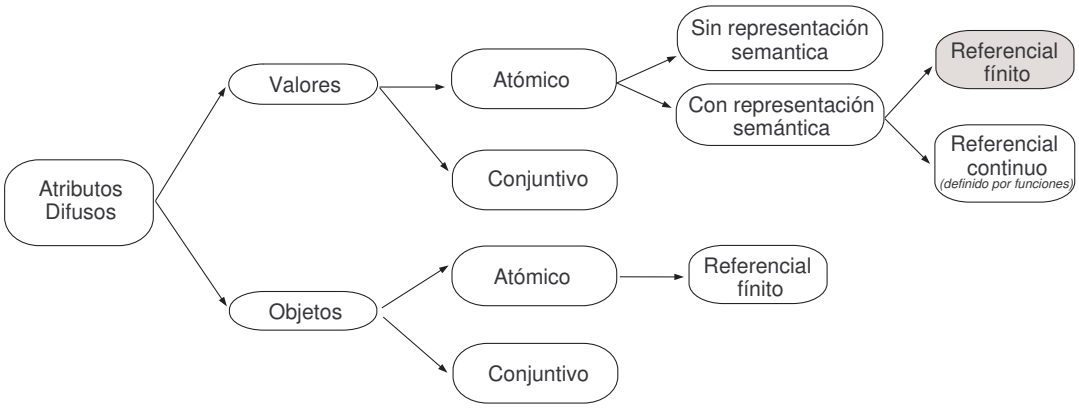


Figura 3.10: Dominios atómicos con referencial continuo.

En el ejemplo de la dureza de un material (véase Ejemplo 3.2), se puede apreciar cómo se definen las distribuciones de posibilidad de las etiquetas lingüísticas. Se utiliza un conjunto de pares de valores donde el primero pertenece al dominio básico y el segundo al intervalo real [0,1]. El dominio real en ese ejemplo es *1, 1.5, 2, 2.5, 3,. . . , 9.5, 10, muy_duro, duro, medio, blando, muy_blando.*

En el momento de determinar el grado de semejanza entre dos valores cualesquiera del dominio, se utiliza una relación de semejanza determinada a partir de la ecuación (3.6) [Mar01], donde $x, y, z \in D$. Esta relación de semejanza da el valor de un operador *FEQ*, definida sobre un dominio con estas características.

$$\mu_{\mathrm{S}}(\mathrm{x},\,\mathrm{y}) = \begin{cases} 1 & (x = y) \wedge (x, y \in B) \\ 0 & (x \neq y) \wedge (x, y \in B) \\ \mu_l(z) & ((x = l \in L) \wedge (y = z \in B)) \vee \\ & (((y = l \in L) \wedge (x = z \in B)) \\ \text{máx}_{z \in B} \otimes(\mu_x(z), \mu_y(z)) & \text{otro caso} \end{cases} \tag{3.6}$$

Aplicando la ecuación 3.6 es posible obtener una relación de semejanza entre todos los

elementos del dominio. El resultado para el ejemplo de la dureza (ver Ejemplo 3.2), se muestra en la tabla 3.5 (se ha utilizado el mínimo como t-norma).

Se han presentado los dominios atómicos con referencial definido por valores. Sin embargo, en ocasiones el valor de un atributo no es atómico y esta dado por un subconjunto de valores. Esta situación se evidencia en el caso de un atributo para almacenar los idiomas que domina un investigador (ver Ejemplo en 3.1). En este caso el investigador no domina un solo idioma, sino un subconjunto de idiomas y cada uno en un grado. Con el objetivo de permitir este tipo de valor el modelo soporta un dominio que permite una cardinalidad conjuntiva. Este tipo de dominio se presenta a continuación.

3.1.3. Dominio conjuntivo sobre valores.

En el dominio con significado conjuntivo (ver Figura 3.11), a diferencia del dominio atómico con referencial finito mostrado en el apartado anterior, sus valores son una combinación de todos los posibles valores del referencial. En este caso el valor del atributo no es una distribución de posibilidad.

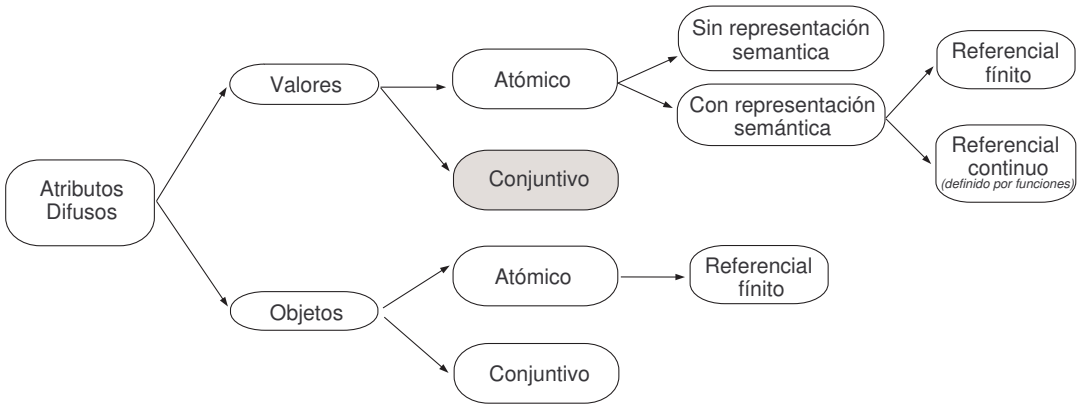


Figura 3.11: Dominio conjuntivo sobre valores.

Estos dominios toman como valor un subconjunto difuso de un dominio básico específico B. El dominio básico B estará definido sobre un referencial de valores en el cual los elementos utilizan la igualdad clásica en el momento de determinar el parecido entre ellos.

Es necesario disponer de una representación adecuada para este tipo de dominio. Partiendo de la existencia de un dominio básico B, los valores que puede tomar un atributo definido sobre este dominio será un subconjunto A con una función de pertenencia repre-

	1	*1.5*	*2*	*2.5*	*3*	*3.5*	*4*	*4.5*	*5*	*5.5*	...	*8*	*8.5*	*9*	*9.5*	*10*	t_{MD}	t_D	t_M	t_B	t_{MB}
1	1	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	1
1.5		1	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0.4	1
2			1	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0.8	1
2.5				1	0	0	0	0	0	0		0	0	0	0	0	0	0	0.2	1	0.8
3					1	0	0	0	0	0		0	0	0	0	0	0	0	0.4	1	0.6
3.5						1	0	0	0	0		0	0	0	0	0	0	0	0.6	1	0.5
4							1	0	0	0		0	0	0	0	0	0	0	0.8	1	0.3
4.5								1	0	0		0	0	0	0	0	0	0	1	0.7	0
5									1	0		0	0	0	0	0	0	0	1	0.4	0
5.5										1		0	0	0	0	0	0	0	1	0	0
6												0	0	0	0	0	0	0.3	1	0	0
6.5												0	0	0	0	0	0	0.6	1	0	0
7												0	0	0	0	0	0.3	1	0.8	0	0
7.5												0	0	0	0	0	0.5	1	0.6	0	0
8												1	0	0	0	0	0.7	1	0.4	0	0
8.5													1	0	0	0	0.8	1	0.2	0	0
9														1	0	0	1	0.7	0	0	0
9.5															1	0	1	0.5	0	0	0
10																1	1	0.2	0	0	0
t_{MD}																	1	0.8	0.5	0	0
t_D																		1	0.8	0	0
t_M																			1	0.8	0.5
t_B																				1	0.8
t_{MB}																					1

Cuadro 3.5: Relación de semejanza para el atributo *dureza*.

sentada de la siguiente forma:

$$\forall A \text{ subconjunto difuso de } B | Soporte(\text{A}) \text{ es finito}, A \subseteq \mathcal{P}(B \times [0,1]) \tag{3.7}$$

Es decir cualquier valor para este dominio será un subconjunto difuso que tendrá una función de pertenencia de soporte finito representada por un conjunto de pares de valores, en los que el primer valor pertenece al dominio básico y el segundo al intervalo real [0,1].

Si se tiene D como el conjunto de todos los posibles valores de un atributo, $D = x_1, x_2, ..., x_n$, el valor del atributo att_1 definido sobre un dominio conjuntivo con referencial de valores, se representa de la siguiente forma:

$$att_1 = \mu_{att_1}(x_1)/x_1 + \mu_{att_1}(x_2)/x_2 + ... + \mu_{att_1}(x_n)/x_n$$

En este caso el tipo de D será preciso y corresponderá a un tipo definido por el sistema. Si se desea determinar el valor de la igualdad entre dos valores de este dominio se debe recurrir al cálculo del parecido entre dos subconjuntos difusos. La comparación entre subconjuntos difusos se suele hacer en términos de inclusión.

$$A = B \textit{ si y sólo si } (A \subseteq B) \wedge (B \subseteq A) \tag{3.8}$$

Para el cálculo del grado de inclusión del subconjunto A en el subconjunto B se tomó la propuesta de Rossazza y otros en [RDP98]. Esta propuesta propone la ecuación 3.9 para el cálculo de la inclusión [Mar01].

$$N(B|A) = \min_{u \in U} \{I(\mu_A(u), \mu_B(u)\} \tag{3.9}$$

Como se muestra en la ecuación 3.8, es necesario calcular la inclusión en los dos sentidos para poder determinar el grado de semejanza entre los dos conjuntos. Los dos grados de inclusión previamente determinados deben ser combinados para obtener el grado de semejanza entre los dos conjuntos mediante la ecuación 3.10.

$$\mu_S(A, B) = OC(N(B|A), N(A|B)) \tag{3.10}$$

Donde OC es un operador de combinación. Las propuestas para combinar los grados de inclusión son las siguientes.

- Utilizar un t-norma para combinar los grados de inclusión.

- Utilizar un t-conorma para combinar los grados de inclusión.
- Utilizar el promedio para combinar los grados de inclusión.

Si se utiliza la t-norma se obtendrá un valor para el grado de semejanza entre los dos conjuntos más restrictivo que si se empleara una t-conorma, mientras que el uso del promedio supondrá un valor intermedio. El uso de una t-conorma para combinar los grados de inclusión es un libertad que se le da al usuario al comparar subconjuntos. A continuación se vuelve el ejemplo de los idiomas presentado al inicio del apartado.

El cálculo de la semejanza entre el valor del atributo *idioma* del investigador1 y del investigador2 puede realizarse teniendo en cuenta las características lingüísticas de la información que se desea obtener. Es posible obtener entre los dos investigadores *si hablan los mismos idiomas* o *hasta qué punto los idiomas que dominan se parecen en un promedio.*

Para este ejemplo cada operación queda como se muestra en la tabla 3.6.

Cuadro 3.6: Ejemplo idiomas que domina un investigador.

Objeto de la comparación	**Cálculo**
¿Hablan los mismos idiomas?	$\mu_S(inv1.idioma, inv2.idioma) =$ $\otimes((\min_{u\in U} I(\mu_{inv2}(u), \mu_{inv1}(u))), (\min_{u\in U} I(\mu_{inv1}(u), \mu_{inv2}(u))))$ $= \otimes(min(1,1,0.75), min(1,0.85,1)) = \mathbf{0.75}$
¿Hasta qué punto los idiomas que dominan se parecen en un promedio?	$\mu_S(inv1.idioma, inv2.idioma) =$ $AVG((\min_{u\in U} I(\mu_{inv2}(u), \mu_{inv1}(u))), (\min_{u\in U} I(\mu_{inv1}(u), \mu_{inv2}(u))))$ $= AVG(min(1,1,0.75), min(1,0.85,1)) = \mathbf{0.8}$

En el ejemplo se utilizó como t-norma (ver epígrafe 2.2.2.3), el mínimo, como t-conorma, el máximo y la implicación propuesta por Goguen (ver Epígrafe 2.2.2.4).

Hasta este momento se han presentado los diferentes tipos de dominios difusos soportados por el modelo y que permiten ser utilizados como dominio de un atributo. Los atributos describen características de objetos, encapsulados en clases, principal elemento de un modelado orientado a objeto. A continuación se muestra como el modelo permite el trabajo

con objetos y las operaciones que con ellos se pueden realizar.

3.2. Representación de Objetos.

Al momento de representar un problema, en el modelo de la base de datos pueden existir varias clases que representen las entidades de una solución. Estas clases tendrán definido un conjunto de atributos con tipos que van desde los más básicos a los tipos asociados a dominios difusos. Cada uno de los objetos de esas clases se considera como objetos que en dependencia de su definición podrán incorporar atributos precisos, atributos difusos u objetos.

Si el objeto incorpora solamente atributos precisos, se tiene un objeto preciso. Si incorpora atributos precisos y al menos uno definido sobre un dominio difusos, estamos en presencia de objetos con atributos difusos; y si el objeto tiene en sus atributos otro objeto entonces se está en presencia de objetos complejos con atributos difusos. La propuesta permite manipular estos tres tipos de objetos.

En los objetos complejos al menos uno de los atributos esta definido sobre un referencial de objetos. Por lo tanto para el modelo los objetos con atributos difusos se convierten a su vez en dominios difusos para el modelo.

3.2.1. La comparación entre objetos con atributos difusos

Los objetos con atributos difusos, como se presentó anteriormente, son objetos que tienen más de un atributo, de los cuales al menos uno de esos atributo es difuso. El modelo debe permitir obtener qué parecido existe entre dos objetos pues este parecido puede ser un elemento de consulta por parte del usuario.

Es adecuado pensar que para obtener el parecido entre dos objetos se deben tener en cuenta los valores que tienen cada uno de sus atributos. Por tal razón, el cálculo del parecido entre dos objetos parte inicialmente de determinar el parecido existente entre cada par de atributos.

Una vez determinado el parecido existente entre los valores de los atributos de dos objetos se deben combinar estos resultados para obtener un valor general de semejanza entre los dos objetos (ver Figura 3.12).

La figura 3.12 muestra en qué consiste este proceso de comparación, inicialmente se determina la semejanza entre cada par de atributo. Denominando como $S_{a_i}(o_1, o_2)$ a la

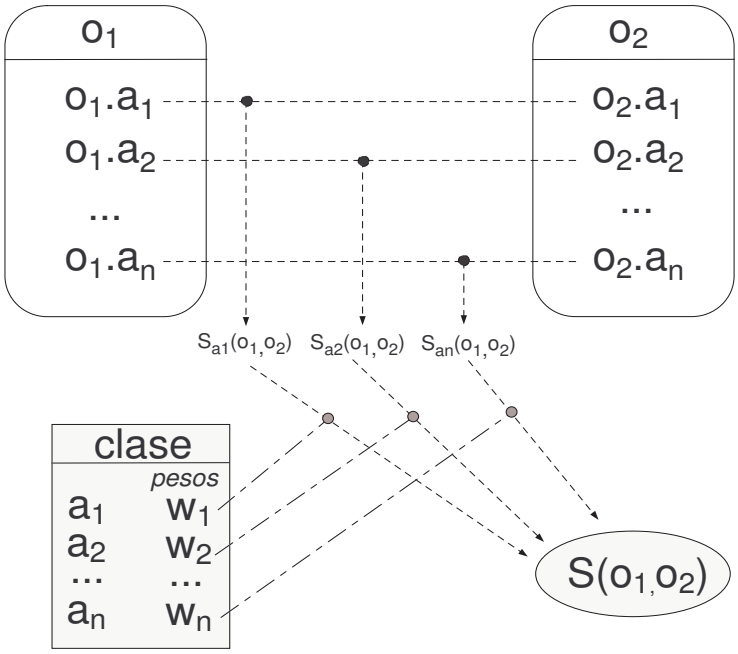


Figura 3.12: Comparación entre objetos.

semejanza entre el valor del atributo a_i en el objeto uno (o_1) y el valor del mismo atributo en el objeto dos (o_2). Luego se determina una agregación de todos los $S_{a_i}(o_1, o_2)$, dando como resultado el $S(o_1, o_2)$ que representa la semejanza entre los dos objetos. Este proceso es similar al propuesto por Marín y otros ([Mar01, MMP+03]).

Importancia de los atributos.

En el proceso de comparar dos objetos puede ser que interese dar un nivel de importancia a un atributo por encima de otros de forma tal que su influencia en el resultado final de la comparación sea mayor. O en los casos en que se comparan objetos que tienen definidos atributos sobre dominios precisos, si esta propiedad tiene la misma importancia que el resto de las propiedades, la semejanza se verá influenciada por la igualdad clásica que se establece entre los valores de esa propiedad en los dos objetos.

El modelo permite definir un nivel de importancia o peso para cada propiedad. Se le asocia un peso w_i a cada atributo a_i (véase Figura 3.12). El valor del peso estará definido en el intervalo [0,1], $p_i \in [0, 1]$.

A partir de esto se plantea el problema de cómo agregar los distintos valores de semejanza entre los atributos, teniendo en cuenta el grado de importancia asignado a cada

uno, de forma que pueda obtenerse el valor final de la semejanza entre los dos objetos. A continuación se presenta cómo realizar esta agregación.

Agregación de semejanza

El cálculo del grado de parecido entre dos objetos de una misma clase se reduce a calcular el grado de cumplimiento de la siguiente sentencia ([Mar01, MMP+03]):

"La mayoría de los atributos importantes tienen valores semejantes en los dos objetos"

Esta sentencia se puede analizar como una sentencia de tipo II de acuerdo con [Yag92] [Zad83] que sería "Q de D son A"donde:

Q: cuantificador que expresa mayoría

X: conjunto de los atributos que caracterizan la clase

D: conjunto de atributos relevantes para el cálculo del parecido expresado como:

$$\mu_D(a_i) = p_i, \forall ai \in X \tag{3.11}$$

A:conjunto de atributos que presentan valores semejantes en los dos objetos expresados como:

$$\mu_A(a_i) = Sai, (o1, o2), \forall ai \in X \tag{3.12}$$

pi: peso asociado a cada atributo reflejando la importancia del atributo.

Para la agregación se utiliza el enfoque de Vila y otros [VCMP94a]. Este enfoque está basado en el concepto de familia coherente de cuantificadores, cuyos extremos son $\exists$ y $\forall$. La idea básica es considerar que el grado de cumplimiento de la sentencia tipo II estará en un punto intermedio entre los grados de cumplimiento de las sentencias *Existe un D que es A* y *Todos los Ds son A*.

El grado de cumplimiento de la sentencia *."Existe un D que es A"* viene dado por:

$$\exists x \in X, (\mu_D(x) \wedge \mu_A(x)) = \max_{x \in X}(\mu_D(x) \wedge \mu_A(x)) \tag{3.13}$$

El de la sentencia *"Todos los Ds son A"*

$$\forall x \in X, (\mu_D(x) \rightarrow \mu_A(x)) = \min_{x \in X}(\mu_A(x) \vee (1 - \mu_D(x))) \tag{3.14}$$

Teniendo en cuenta estas ecuaciones, Vila y otros [VCMP94a] definen una familia de cuantificadores para Q de forma que pueda calcularse el grado de cumplimiento de la sentencia *"Q de D son A"* en un rango entre *."Existe un D que es A"* y *"Todos los Ds son*

A". A partir de esto se define una familia coherente de cuantificadores Q_i que tienen como elemento máximo $Q_1 = \exists$ y mínimo a $Q_n = \forall$ con valores γ_i en el intervalo [0,1], siendo $\gamma_\exists = 1$ y $\gamma_\forall = 0$. El cálculo del grado de cumplimiento de la sentencia *"Q de D son A"* queda definido mediante la siguiente expresión [VCMP94a]:

$$V_Q(A|D) = \gamma_Q \underset{x \in X}{max}(\mu_D(x) \wedge \mu_A(x)) + (1 - \gamma_Q) \underset{x \in X}{min}(\mu_A(x) \vee (1 - \mu_D(x))) \tag{3.15}$$

El usuario podrá especificar que cuantificador quiere utilizar para el cálculo de la semejanza entre objetos dando el valor de γ_Q en el rango [0,1]. En dependencia de la necesidad de la aplicación el valor de γ_Q se podrá acercar a $\gamma_\exists$ o a $\gamma_\forall$.

La utilización de este procedimiento para determinar la semejanza entre dos objetos permite definir el operador *FEQ* para comparar todos los objetos existentes en el modelo. En la función de agregación que se utiliza para el operador *FEQ*, propuesto por Vila y otros [VCMP94a], ecuación 3.15, juega un papel importante el número de orness. Dependiendo del valor que se asigne a este número el cumplimiento de la sentencia de Tipo II se moverá hacia lo existencial o lo universal.

Se ha presentado el proceso para obtener el parecido entre dos objetos, partiendo de la semejanza que existe entre los valores de los atributos. En apartados anteriores se mostró como determinar la semejanza entre dos valores de un dominio difuso cuando el referencial son valores. En el modelo propuesto se hizo una adaptación de los dominios para el caso de que el referencial sean objetos.

A continuación se mostrará las extensiones realizadas a los dominios para soportar referenciales definidos sobre objetos. Los dominios que fueron extendidos son: atómico con referencial finito y los dominios conjuntivos. Cada uno de estos dominios será mostrado en los siguientes apartados.

3.2.2. Dominio atómico con referencial finito sobre objetos.

Cuando el referencial de los dominios atómicos son objetos se tiene un dominio de valores atómicos con un significado disyuntivo (ver Figura 3.13). El referencial de este tipo de dominio puede ser cualquier objeto soportado por el modelo, incluyendo objetos con una sola propiedad definida sobre uno de los dominios atómicos anteriormente estudiados, objetos con propiedades definidas sobre dominios conjuntivos u objetos difusos complejos.

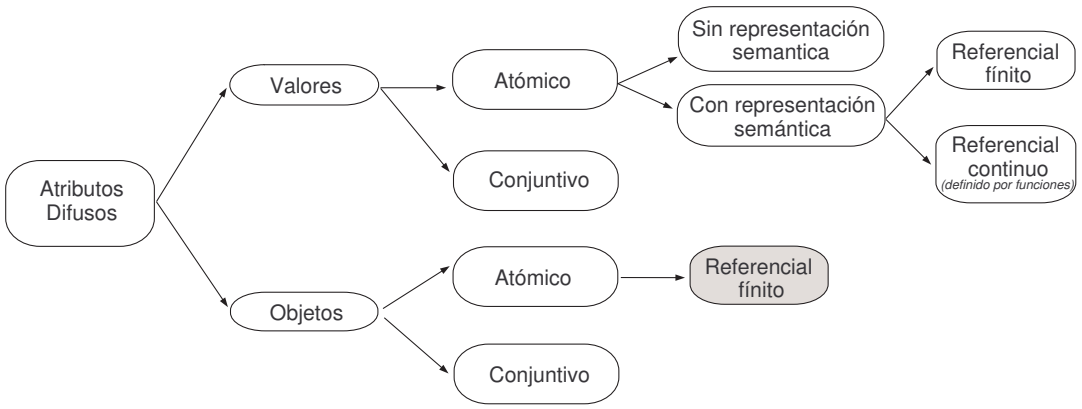


Figura 3.13: Dominio atómico con referencial finito sobre objetos.

El cálculo del parecido entre dos valores de este dominio se realizará teniendo en cuenta su naturaleza disyuntiva y para el mismo se propone utilizar la ecuación (3.16).

$$DIS(A|B)_S = max_{x\epsilon U} max_{y\epsilon U} \otimes (\mu_S(x,y), \otimes(\mu_A(x), \mu_B(y))) \tag{3.16}$$

Con la aplicación de la ecuación (3.16) se está buscando en qué medida un elemento está presente en los dos subconjuntos, teniendo en cuenta la semejanza existente entre los elementos. Inicialmente se calcula la semejanza entre los elemento y luego se restringe por la pertenencia de los elementos a ambos subconjuntos. Un ejemplo de este tipo de dominio se muestra a continuación.

Ejemplo 3.4. *Se tienen de tres artículos cuales son sus autores y el grado de participación de cada uno en la autoría. De esta forma tenemos los siguientes valores:*

$$articulo1.autor = 0{,}6/David + 0{,}7/Fernando$$

$$articulo2.autor = 0{,}75/David + 0{,}3/Isabel + 0{,}63/Esther$$

$$articulo2.autor = 0{,}85/Isabel + 0{,}4/Esther$$

Si se quiere determinar en que grado en los artículos han participado un mismo autor. En este caso se deben analizar la comparación entre los dos valores con una perspectiva disyuntiva. Para esto se aplica la ecuación (3.16), dando el siguiente resultado:

$$DIS(articulo1.autor, articulo2.autor) = 0{,}6$$

$$DIS(articulo1.autor, articulo3.autor) = 0{,}2$$

$$DIS(articulo2.autor, articulo3.autor) = 0{,}4$$

Además de este dominio, los dominios con un significado conjuntivo, previamente definidos sobre un referencial de valores, fueron extendidos a un referencial de objetos.

3.2.3. Dominio conjuntivos sobre objetos.

Como fue mencionado, no sólo es posible definir un dominio conjuntivo para un referencial de valores, también se puede trabajar con dominios conjuntivos con referencial de objetos (ver Figura 3.14). En este caso los elementos que forman el dominio básico B son objetos que tienen definida o permiten calcular una relación de semejanza entre ellos.

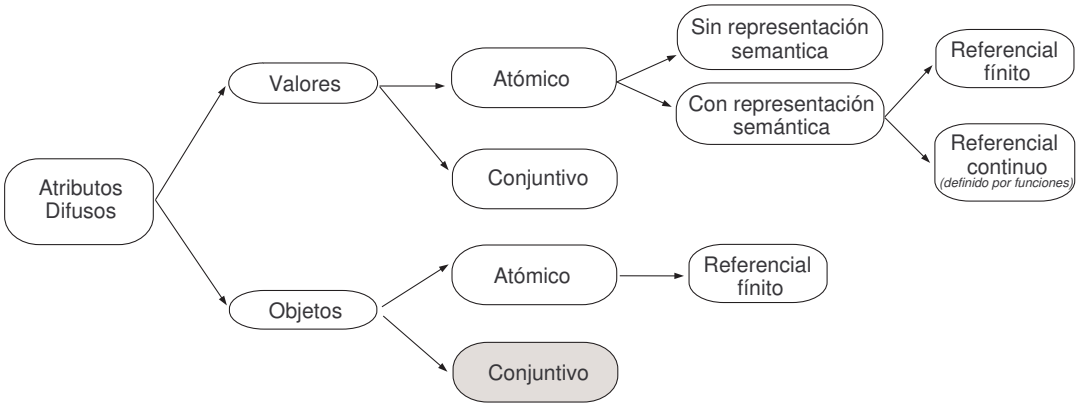


Figura 3.14: Dominio conjuntivos sobre objetos.

En el siguiente ejemplo se muestra este tipo de dominio, en el se utiliza como elementos, en el subconjunto difusos, objetos.

Ejemplo 3.5. *Definiendo con más detalle el ejemplo de los investigadores vinculados a un proyecto. El dominio básico serán todos los investigadores de la organización, por ejemplo {'David', 'Isabel', 'Fernando', 'Esther'}, cada uno de estos investigadores está caracterizado por varios atributos: identificación, nombre, edad, sexo, idiomas que domina, características personales, etc. En este caso si se define un atributo en una tabla Proyectos para reflejar la participación de los investigadores en los proyectos, dos posibles valores serían:*

$$proyecto1.participan = 0{,}8/David + 0{,}6/Esther + 0{,}6/Isabel$$
$$proyecto2.participan = 1{,}0/Fernando + 0{,}3/David + 0{,}3/Esther + 0{,}6/Isabel$$

El ejemplo define el referencial sobre objetos, sin embargo también puede definirse el dominio conjuntivo sobre valores entre los que existe definida una relación de semejanza. Para el modelo estos dominios son tratados como objetos. Este caso se refiere a dominios difusos soportados por el modelo. El siguiente ejemplo muestra esta situación (ver Ejemplo 3.6).

Ejemplo 3.6. *Se quiere almacenar en un atributo de la tabla Investigador una valoración de las características personales de cada uno de los investigadores. Para esto se definió un conjunto de etiquetas que representan las características de una persona. El domino básico entonces estaría formado por {'honesto', 'trabajador', 'independiente', 'creativo', 'líder'}. Sobre este dominio quedó definida una relación de semejanza según la Tabla 3.7.*

Cuadro 3.7: Relación de semejanza para el atributo *personalidad*.

	Honesto	*Trabaj*	*Indep*	*Creativo*	*Líder*
Honesto	1.0	0.8	0.4	0.3	0.3
Trabajador	0.8	1.0	0.6	0.4	0.3
Independiente			1.0	0.8	0.9
Creativo				1.0	0.9
Líder					1.0

Luego la personalidad, atributo *caract_perso* del objeto *Investigador* puede tomar alguno de estos valores:

$$investigador1.caract\_perso = \{0{,}8/honesto/ + 0{,}6/trabajador + 0{,}4/independiente + 0{,}9/lider\}$$
$$investigador2.caract\_perso = \{1{,}0/trabajador + 0{,}7/honesto + 0{,}2/lider\}$$

En este ejemplo el valor del atributo *caract_perso* está formado por elementos definidos sobre un dominio difuso sin representación semántica asociada, estos elementos son considerados objetos que poseen un solo atributo.

A diferencia de los dominios conjuntivos con referencial sobre valores, en estos dominios conjuntivos los elementos del subconjunto son objetos. En tal sentido el cálculo de la

semejanza entre dos valores se hará siguiendo el siguiente procedimiento. Se parte de que se tienen dos subconjuntos difusos A y B.

1. Se acepta que la comparación entre subconjuntos difusos se realiza en términos de inclusión como se planteó en la ecuación (3.8).

2. Se determina el Grado de Inclusión de los dos subconjuntos en las dos direcciones. Es decir, el grado de inclusión del conjunto A en el conjunto B y luego el grado de inclusión de B en A. Se debe tener en cuenta la semejanza que existe entre los elementos de los subconjuntos por lo que se determina el Grado de Inclusión Guiado por Semejanza.

3. Se combinan los dos grados de inclusión calculados. La combinación puede realizarse utilizando una t-norma entre las dos inclusiones determinando el Grado de Semejanza Generalizada, utilizar una t-conorma para la combinación entonces se obtiene el Grado de consistencia entre los dos subconjuntos o utilizar una media entre una actitud pesimista y una optimista.

Los principales elementos del proceso para determinar la semejanza entre dos atributos definidos sobre dominios conjuntivos se describen a continuación.

Como los valores que se quieren comparar son subconjuntos difusos y sus elementos tienen una naturaleza imprecisa, la igualdad clásica no puede ser aplicada y se sustituye por una relación de semejanza. Para esto se definen los siguientes conceptos.

Grado de Inclusión Guiado por Semejanza: Este grado determina para todo elemento de A un elemento en B que se le parezca y que su pertenencia a B sea mayor que el grado de pertenencia del elemento de A al subconjunto A. El cálculo se realiza por medio de la ecuación 3.17, [Mar01].

$$\begin{aligned} &\Theta_S(B|A) = \min_{x\in U} \max_{y\in U} \theta_{A,B,S}(x,y) \\ &donde \theta_{A,B,S}(x,y) = \otimes(I(\mu_A(x), \mu_B(y)), \mu_S(x,y)) \end{aligned} \tag{3.17}$$

A y B son dos subconjuntos difusos definidos sobre un universo de referencia U sobre el que hay definida una relación de semejanza S, mientras que x, $y \in U$

Se determina entonces la implicación en los dos sentidos calculando el grado de inclusión por semejanza en ambos sentidos. Para obtener el grado de semejanza entre los

dos conjuntos se deben combinar los dos grados de inclusión, obteniéndose el Grado de Semejanza Generalizado.

Grado de Semejanza Generalizado: Este grado de semejanza tiene en cuenta las dos direcciones de la inclusión y se define según la ecuación 3.18, [Mar01]:

$$\beth_{S,\otimes}(A,B) = \otimes(\Theta_S(B|A), \Theta_S(A|B)) \tag{3.18}$$

Como se puede ver se emplea una t-norma *($\otimes$)* dándole un carácter restrictivo al cálculo de la semejanza. Si en vez de utilizar una t-norma se utiliza una t-conorma se determinaría el grado de consistencia entre los dos subconjuntos difusos, ecuación 3.19, [Mar01].

$$\beth_{S,\oplus}(A,B) = \oplus(\Theta_S(B|A), \Theta_S(A|B)) \tag{3.19}$$

Por lo tanto de forma similar a los dominios conjuntivos con referencial de valores, se definen tres posibilidades para combinar los grados de inclusión entre dos subconjuntos difusos:

- Utilizando una t-norma con lo cual se determina el Grado de Semejanza Generalizado, en este caso se define el operador *FEQ* respondiendo a la ecuación 3.18, [Mar01].
- Utilizando una t-conorma con lo cual se determinan el grado de consistencia entre los dos subconjuntos difusos y se utiliza el operador *COG*, según la ecuación 3.19, [Mar01].
- Y con el objetivo de obtener una media entre un actitud pesimista y una optimista se define la media como operador, en este caso *AVGC*.

Aplicando las tres formas de combinar los grados de inclusión al ejemplo de las características de la personalidad de un investigador (ver Ejemplo 3.6) y comparando la personalidad del investigador1 con el investigador2, obtenemos el resultado que se muestra en la tabla (3.8).

Queda a selección del usuario el uso de uno u otro operador. En el ejemplo se utilizó el mínimo como t-norma, máximo como t-conorma y la implicación de Goguen (ver Epígrafe 2.2.2.4).

Cuadro 3.8: Ejemplo personalidad de un investigador.

Objeto de la comparación	**Cálculo**
Tienen las mismas características en su personalidad	$\sqsupset_{S,\otimes}(invest1.caract\_perso, invest2.caract\_perso) = \otimes(\Theta_S(invest2.caract\_perso \mid invest1.caract\_perso), \Theta_S(invest1.caract\_perso \mid invest2.caract\_perso)) = \otimes(0.8, 0.5) = \mathbf{0.5}$
Son compatibles sus carácter	$\sqcap_{S,\oplus}(invest1.caract\_perso, invest2.caract\_perso) = \oplus(\Theta_S(invest2.caract\_perso \mid invest1.caract\_perso), \Theta_S(invest1.caract\_perso \mid invest2.caract\_perso)) = \otimes(0.8, 0.5) = \mathbf{0.8}$
Hasta qué punto las personalidades se parecen en un promedio	$AVG_{S,AVG}(invest1.caract\_perso, invest2.caract\_perso) = AVG(\Theta_S(invest2.caract\_perso \mid invest1.caract\_perso), \Theta_S(invest1.caract\_perso \mid invest2.caract\_perso)) = AVG(0.8, 0.5) = \mathbf{0.65}$

En estos casos no se considera la cardinalidad de los subconjuntos que se están comparando. Al no considerar la cardinalidad de los subconjuntos puede ocurrir que dos subconjuntos con cardinalidades muy distantes, al compararlos den como resultado una semejanza mayor que 0. En ocasiones puede interesar que la cardinalidad se refleje en el resultado del cálculo de la semejanza. Para tener en cuenta la cardinalidad, se debe ponderar el Grado de Semejanza Generalizado con un factor que dependa de la distancia entre los cardinales de los dos conjuntos. El factor de cardinalidad entre dos conjuntos se define según la ecuación 3.20, [Mar01].

$$\Phi(A,B) = \begin{cases} 1 & \text{si } A = \emptyset \wedge B = \emptyset \\ 0 & \text{si } (A = \emptyset \wedge B = \emptyset) \vee (A \neq \emptyset \wedge B \neq \emptyset) \\ \frac{\text{mín}(|A|,|B|)}{\text{máx}(|A|,|B|)} & \text{otro caso} \end{cases} \tag{3.20}$$

En el ejemplo de las características de la personalidad del *investigador1* y el *investigador2*, el factor de cardinalidad se define como: $\Phi(A,B) = \frac{\text{mín}(4,3)}{\text{máx}(4,3)} = 0.6$. Este factor afecta el valor de semejanza obtenido.

Finalmente con el objetivo de dar mayores posibilidades al trabajo con dominio conjuntivos tanto con referencial sobre valores como sobre objeto se definen operadores para comparar dos valores del dominio por medio de la inclusión y teniendo en cuenta la semejanza existente entre los elementos del referencial.

3.2.3.1. Comparación entre dominios conjuntivos por medio de la inclusión.

En ocasiones es necesario solamente conocer en qué grado un subconjunto difuso incluye a otro, este tipo de operación permite responder preguntas del tipo: en qué grado el estudiante Juan domina determinado idioma?, o cualquier otro tipo de pregunta que necesite comparar un subconjunto con respecto a otro pero en un solo sentido. En estos casos la comparación también se realiza en términos de inclusión pero ahora sólo se tendrá en cuenta una sola dirección. Para estos casos se proponen dos operadores, el operador *GIS* que determina el Grado de Inclusión Guiado por Semejanza de un subconjunto en otro y el *GIN* que determina el Grado de inclusión teniendo en cuenta la semejanza.

El operador *GIS* se calcula utilizando el Grado de Inclusión Guiado por Semejanza, reflejado en la ecuación 3.17 [Mar01]. El resultado final de este operador estará restringido por el grado de semejanza entre los elementos que componen los subconjuntos. Este operador cumple las propiedades de ser válido cuando la relación de semejanza no es otra que

la igualdad clásica y la de ser monótona con respecto a las relaciones de inclusión.

De la misma forma se propone el operador *GIN* un poco menos restrictivo y definido mediante la ecuación 3.21.

$$GIN_S(B|A) = \min_{x\in U} \max_{y\in U} I(\mu_A(x), \otimes(\mu_B(y), \mu_S(x,y))) \tag{3.21}$$

Este operador determina el grado de implicación entre el nivel de pertenencia del elemento en A y el nivel de pertenencia del elemento en B restringido por el grado de semejanza entre los elementos. Se busca el elemento de B que mayor pertenencia tiene a B y que más se parece al elemento de A. En este caso el grado de la inclusión lo da la implicación, reflejando un resultado para la inclusión válido para situaciones que requieren menos restricciones. En este mismo apartado se mostrará un ejemplo de cálculo con este operador.

Los operadores *GIS* y *GIN* deben cumplir la propiedad de ser válidos cuando la relación de semejanza no es otra que la igualdad clásica y que sea monótona con respecto a las relaciones de inclusión. En el caso del operador *GIS* estas propiedades fueron demostradas por Marín y otros [Mar01]. Queda por demostrar si el operador *GIN* cumple estas propiedades.

El operador GIN debe cumplir inicialmente la propiedad de ser válido cuando la relación de semejanza no es otra que la igualdad clásica.

Propiedad 3.1. *Sean A y B dos subconjuntos difusos definidos sobre un universo de referencia U sobre el que hay definida una relación de semejanza S que es la igualdad clásica. Entonces $\Upsilon_S(B|A) = N(B|A)$.*

Demostración. Si $x \neq y$ entonces $N(B|A) = 0$. Como $\mu_S(x,y) = 0$ queda $\Upsilon_S(B|A) = \min_{x\in U} \max_{y\in U} I(\mu_A(x), \otimes(\mu_B(y), 0))$, $\otimes(\mu_B(y), 0) = 0$, $\min_{x\in U} \max_{y\in U} I(\mu_A(x), 0) = 0$, como se quiere demostrar. Por otra parte si $x = y$ entonces $N(B|A) = \min_{x\in U} I(\mu_A(x), \mu_B(x))$. Como $\mu_S(x,x) = 1$ queda $\Upsilon_S(B|A) = \min_{x\in U} I(\mu_A(x), \otimes(\mu_B(x), 1))$, $\otimes(\mu_B(y), 1) = \mu_B(y)$, $\Upsilon_S(B|A) = \min_{x\in U} I(\mu_A(x), \mu_B(x))$, como se quiere demostrar. $\Upsilon_S(B|A) = N(B|A)$.

La otra propiedad deseada en una ecuación para el cálculo del grado de inclusión es que sea monótona con respecto a las relaciones de inclusión.

Propiedad 3.2. *Sean A, B y C tres subconjuntos difusos definidos sobre un universo de*

referencia U sobre el que hay definida una relación de semejanza S. Entonces $A \subseteq B \Rightarrow \Upsilon_S(C|A) \geq \Upsilon_S(C|B)$.

Demostración. Si $A \subseteq B$ entonces $\mu_A(x) \leq \mu_B(x)\ \forall x \in U$, teniendo en cuenta las propiedades de la implicación, por lo tanto $\mu_A(x) = \mu_B(x) - \varepsilon$ con $\varepsilon \geq 0$. Si se busca la inclusión $\Upsilon_S(C|A)$ y $\Upsilon_S(C|B)$, poniendo todo en función de $\mu_B(x)$ y teniendo en cuenta nuevamente las propiedades de la inclusión se obtiene que $\Upsilon_S(C|A) = \min\limits_{x\in U} \max\limits_{y\in U} I(\mu_B(x) - \varepsilon, \otimes(\mu_B(y), \mu_S(x,y))) \geq \Upsilon_S(C|B) = \min\limits_{x\in U} \max\limits_{y\in U} I(\mu_B(x), \otimes(\mu_B(y), \mu_S(x,y)))$

Como es de esperar no existe simetría en la ecuación (3.21)pues se está buscando un grado de inclusión de un subconjunto difuso en otro.

Los operadores *GIS* y *GIN* permiten calcular la inclusión de un subconjunto difuso en otro. Corresponde al usuario decidir, en dependencia del problemas y de la características lingüísticas de la información que se desea obtener, qué operador utilizar. ¿Cuándo es conveniente utilizar un operador u otro? el ejemplo (3.7) pretende dar elementos para responder esta pregunta.

Ejemplo 3.7. *Se tiene un listado de lenguajes de programación, sobre los cuales se define una relación de semejanza a partir de la filosofía y de las características sintácticas de cada uno. La relación de semejanza está representada en la tabla 3.9.*

Cuadro 3.9: Ejemplo de semejanza entre lenguajes de programación

	Java	JScript	JSP	C	C++	C#
Java	1	0,45	0,9	0,6	0,7	0,7
JScript	0,45	1	0,5	0,4	0,4	0,4
JSP	0,9	0,5	1	0,55	0,6	0,6
C	0,6	0,4	0,55	1	0,8	0,7
C++	0,7	0,4	0,6	0,8	1	0,8
C#	0,7	0,4	0,6	0,7	0,8	1

Un atributo, leng_prog*, de un objeto denominado* Estudiante*, tomará valores sobre un dominio conjuntivo con un referencial de objetos que son los lenguajes de programación definidos en el dominio {Java, JScript, JSP, C, C++, C#}. De esa forma algunos posibles valores que puede tener este atributo para tres estudiantes son:*

$$estudiante1.leng\_prog = 0{,}1/Java/ + 1{,}0/C++$$
$$estudiante2.leng\_prog = 0{,}5/Java/ + 0{,}6/C++$$
$$estudiante3.leng\_prog = 0{,}45/Java/ + 0{,}5/C++$$

Y por ejemplo, se desea seleccionar un estudiante para un proyecto que será realizado con Java, con este objetivo se define el siguiente patrón:

$$patron.leng\_prog = 0{,}6/Java$$

Como puede notarse, el *estudiante1* domina con un grado 1.0 el lenguaje C++, sin embargo conoce muy poco del lenguaje Java (0.1). Por otro lado el *estudiante2* ha trabajado en C++ y tiene un nivel de dominio de 0.6 pero domina algo más el lenguaje Java (0.5).

Se quiere responder estas dos preguntas ¿en qué grado dominan los estudiante el lenguaje que se exige en el contrato? ¿en qué grado podrán los estudiantes programar en el lenguaje que se exige para el contrato?. Estas dos preguntas pueden ser respondidas usando los operadores GIN y GIS, previamente definidos. Los resultados de usar estos operadores son los siguientes, tabla (3.10).

Cuadro 3.10: Utilización de los operadores GIN y GIS

Pregunta	**Operador**	**Est1**	**Est2**	**Est3**
Dominan los lenguajes exigidos	GIS	0.6	0.83	0.75
Pueden programar en los lenguajes exigidos	GIN	1	1	0.83

En los resultados mostrados en la tabla (3.10) se utilizó la inclusión propuesta por Goguen (ver Epígrafe 2.2.2.4). y el mínimo como t-norma. Como se evidencia, la inclusión propuesta por el operador *GIN* es menos restrictiva pues el parecido entre los objetos del subconjunto compensa un poco la pertenencia de un objeto al subconjunto. Por ejemplo, en el caso del *estudiante1* la pertenencia del lenguaje *Java* es baja (0,1) con respecto a lo que se desea (0.6), sin embargo, esto es compensado por el grado de parecido que existe entre los lenguajes *Java* y *C++* (0.7). Como el *estudiante1* domina el lenguaje C++ en 0.6 pues el operador da como resultado 1.0.

Se han presentado las extensiones realizadas a los dominios difusos para que soporten referenciales definidos sobre objetos. Pero el modelo también define operaciones directamente sobre los objetos, en particular la comparación entre objetos difusos.

3.2.4. Ejemplo investigadores-línea de investigación.

A continuación se presenta un ejemplo (véase Ejemplo 3.8) para mostrar lo propuesto hasta este momento. Este ejemplo será retomado en los próximos apartados para demostrar otros elementos del modelo. En este momento se definirá la estructura general del ejemplo.

Ejemplo 3.8. *Se tiene el ejemplo de los* investigadores*. Un investigador esta descrito por los atributos que se muestran en la tabla (3.11). Los objetos investigador están descritos por medio del siguiente conjunto de atributos:* nombre, edad, altura, idioma, personalidad, experiencia, salario, retenciones, idoneidad, evaluación*, definidos de la siguiente forma:*

Cuadro 3.11: Atributos de la clase Investigador

Atributo	**Descripción**	**Tipo**
nombre	nombre del empleado	preciso
edad	edad del empleado	difuso DT
altura	altura del empleado	difuso DT
idioma	idiomas que domina el empleado	difuso Conjuntivo
personalidad	personalidad del empleado	difuso Conjuntivo
experiencia	experiencia del empleado	difuso DT
salario	salario del empleado	preciso
retenciones	por ciento de retención del empleado	preciso
idoneidad	idoneidad del empleado	difuso DT
evaluación	evaluación del empleado	difuso DT

- El *nombre* del empleado es un tipo de dato precisos definido sobre un tipo de dato especificado por el sistema.
- La *edad* del empleado se representa en este ejemplo por medio de un dominio atómico con referencial continuo (ver Figura 3.15).

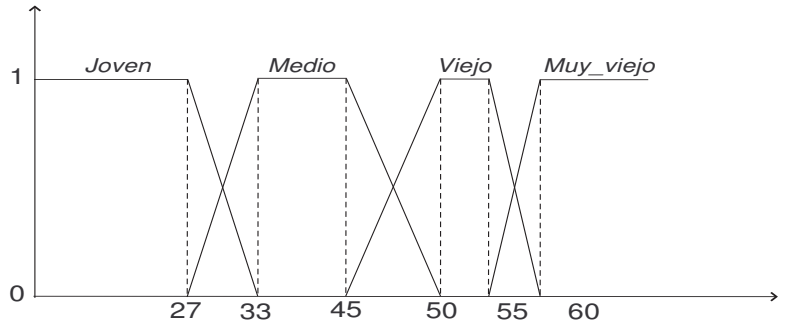


Figura 3.15: Etiquetas lingüísticas para el atributo *edad*.

- El atributo altura está definido sobre un dominio atómico con referencial continuo (véase Figura 3.16).

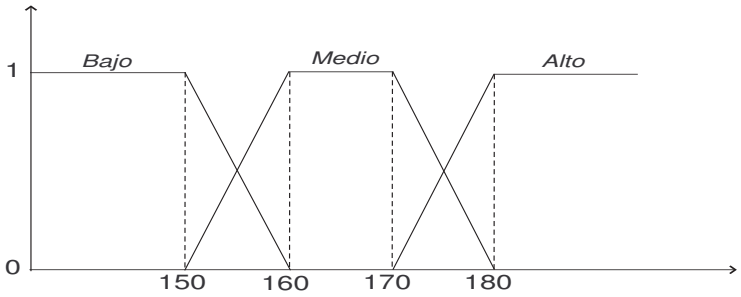


Figura 3.16: Etiquetas lingüísticas para el atributo *altura*.

- Los *idiomas* que domina un empleado son un subconjunto de idiomas cada, uno con un grado de pertenencia que representa el grado de dominio de un idioma por un empleado. Como entre los idiomas no existen relaciones de semejanza definida, se representan los idiomas con un dominio conjuntivo sobre un referencial de valores.

- La *personalidad* de un empleado está compuesta por varios rasgos de la personalidad cada uno con un grado que representa el nivel de presencia de ese rasgo en un empleado. Entre los rasgos de la personalidad existe una relación de semejanza definida por lo que los rasgos son representados como objetos desusos definidos sobre un dominio sin representación semántica asociada y con la relación se semejanza que se presenta en la tabla (3.13).

Cuadro 3.12: Relación de semejanza para los rasgos de la personalidad.

	Honesto	*Trabaj*	*Independ*	*Creativo*	*Líder*
Honesto	1.0	0.8	0.4	0.3	0.3
Trabajador		1.0	0.6	0.4	0.3
Independiente			1.0	0.8	0.9
Creativo				1.0	0.9
Líder					1.0

Para representar la personalidad se utilizará un dominio conjuntivo sobre un referencial de objeto que son los rasgos de la personalidad.

- La experiencia del empleado se mide en números de meses pero se trabaja por medio de expresiones lingüísticas, por ello es necesario definir una variable lingüística

que estará representada por un dominio atómico con referencial continuo (como se muestra en la Figura 3.17).

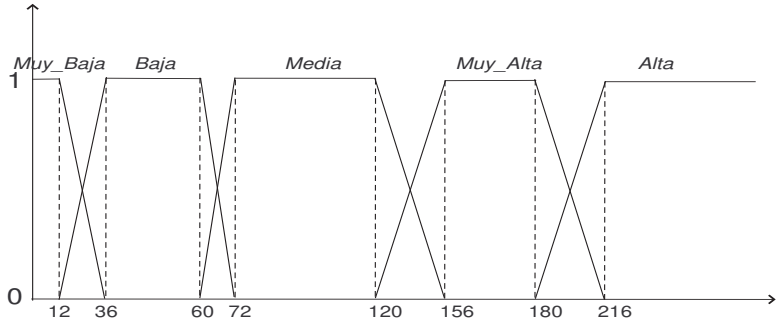


Figura 3.17: Etiquetas lingüísticas para el *experiencia altura*.

- Salario es un atributo que especifica el salario que recibe un empleado en un mes de trabajo, es un atributo preciso.

- El atributo retención almacena el por ciento del salario que se le retendrá al empleado en cada mes de trabajo por lo que su dominio es preciso. Este valor depende el monto total del salario. Por lo tanto se define como atributo inferido con un consecuente preciso y un tipo final preciso también (la definición y manipulación de este tipo de atributo será presentado en próximos apartados). El sistema de reglas que permitirán inferir el valor es el siguiente:

```
IF salario <= 500 THEN retencion IS 0,
IF salario > 500 AND salario < 1500 THEN retencion IS 1.5,
IF salario >= 1500 THEN retencion IS 3
```

- La idoneidad es la característica del empleado se ser adecuado y apropiado para el trabajo que realiza. Se evalúa por puntos en el rango [0,10] siendo 0 la menor idoneidad deseada y 10 la mayor. Al manipularla es conveniente utilizar expresiones lingüísticas por lo que se define un dominio atómico con referencial continuo para este atributo. Las diferentes etiquetas lingüísticas se definen según la figura 3.18.

  El valor de este atributo depende del valor que tomen otros atributos en el objeto, por lo tanto se define el atributo como inferido con un tipo en el consecuente definido

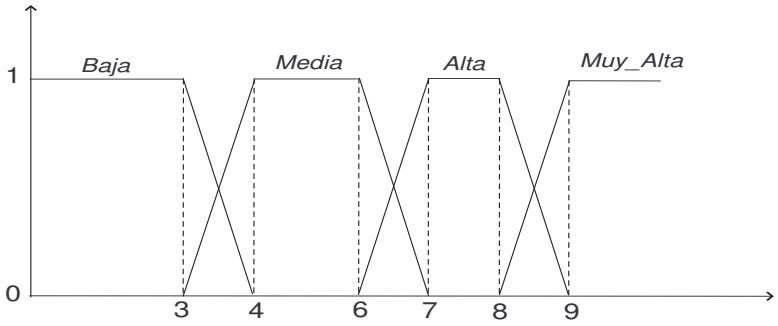


Figura 3.18: Etiquetas lingüísticas para el atributo *idoneidad*.

sobre un dominios dominio atómico con referencial continuo (ver Figura 3.18) y un tipo final para el atributo preciso que representa el valor de la idoneidad para el empleado. El sistema de regla definido con este fin es el siguiente.

```
R1: IF edad FGT medio AND altura NFGEQ media AND personalidad GIN
(0.9/honesto+0.8/trabajador+0.4/lider+0.6/creativo+0.4/independiente)
 AND experiencia FGEQ alta AND evaluacion IS excelente THEN
idoneidad IS muy_alta, R2: IF edad FGT medio AND altura NFGEQ media
AND personalidad GIN
(0.9/honesto+0.8/trabajador+0.4/lider+0.6/creativo+0.4/independiente)
AND experiencia FGEQ alta AND evaluacion IS muy_buena THEN idoneidad
IS muy_alta, R3: IF edad IS joven AND altura NFGEQ media AND
personalidad GIN
(0.9/honesto+0.8/trabajador+0.4/lider+0.6/creativo+0.4/independiente)
THEN idoneidad IS alta, R4: IF edad IS muy_viejo THEN idoneidad IS
baja
```

- Evaluación es un atributo que refleja la última evaluación obtenida por el empleado y se representa por medio de un dominio atómico sin representación semántica asociada donde se definen las etiquetas lingüísticas y relación de semejanza entre ella que se muestran en la tabla (3.13).

Se tienen tres objetos de la clase *Empleados*, caracterizados de la siguiente manera:
empleado1: nombre=Fernando, edad=viejo, altura=177, idiomas=1.0/español, 0.6/inglés, 0.8/frances, personalidad=0.8/honesto, 0.5/líder, 0.7/independiente, experiencia=alta, sa-

Cuadro 3.13: Relación de semejanza para los rasgos de la personalidad.

	Excelent	*M_buena*	*Buena*	*Regular*	*Mala*
Excelente	1.0	0.85	0.5	0.2	0.0
Muy_buena		1.0	0.7	0.6	0.2
Buena			1.0	0.85	0.4
Regular				1.0	0.9
Mala					1.0

lario=430, evaluación=excelente

Al insertar el objeto se infieren los valores: retenciones=0.0, idoneidad=muy_alta

empleado2: nombre=David, edad=47, altura=media, idiomas=1.0/español, 0.9/inglés, 0.7/frances, personalidad=0.8/honesto, 0.5/líder, 0.7/independiente, experiencia=muy_alta, salario=520, evaluación=excelente

Al insertar el objeto se infieren los valores: retenciones=1.5, idoneidad=muy_alta

empleado3: nombre=Esther, edad=viejo, altura=178, idiomas=1.0/español, 0.6/inglés, 0.8/frances, personalidad=0.4/honesto, 0.5/trabajador, 0.5/independiente, experiencia=alta, salario=1530, evaluación=muy_buena Al insertar el objeto se infieren los valores: retenciones=3.0, idoneidad=baja

Si se desea comparar a los investigadores *Esther* y *David* es necesario ir realizando la comparación de cada uno de los atributos, el resultado de dicha comparación es el siguiente (ver Tabla 3.14).

Cuadro 3.14: Comparación entre investigadores

Atributo	Investigador	Investigador	Parec.
nombre	Esther	David	0
edad	viejo	47	0,4
altura	media	178	0,4
idioma	(1.0/Espanol +0.6/Ingles +0.8/Frances)	(1.0/Espanol +0.9/Ingles +0.7/Frances)	0,67
personalidad	(0.4/honesto +0.5/trabajador +0.5/independiente)	(0.8/honesto +0.5/lider +0.7/independiente)	0.63
experiencia	alta	muy_alta	0,5
salario	1530	520	0
retención	3	1.5	0
idoneidad	baja	muy_alta	0
evaluación	muy_buena	excelente	0.85

Aplicando la ecuación 3.15 para determinar el parecido entre los dos investigadores, con un número de orness de 0.2, se obtiene el siguiente resultado:

$\mu_S(Esther, David) = 0.2 * 0{,}85 + (1 - 0.2) * 0{,}0 = 0{,}17$

Los investigadores Esther y David se parecen en 0.17.

Se han presentado los elementos teóricos en la definición de los objetos y cómo se realiza la comparación entre dos objetos. Sin embargo, uno de los principales aportes de este modelo es el de permitir trabajar con objetos complejos. Los objetos complejos se caracterizan por estar definidos por varios atributos pero uno de esos atributos es a la vez un objeto. En estos casos al comparar dos objetos complejos se requiere de recursividad en la comparación.

3.2.5. Recursividad en objetos complejos

La determinación de la semejanza entre dos objetos está resuelto, según lo descrito hasta este momento. Sin embargo, producto al poder de interrelación entre los datos que permite el modelo orientado a objetos se introduce un nivel de complejidad en el momento de determinar el parecido entre dos objetos. Teniendo en cuenta que los objetos complejos con atributos difusos pueden tener atributos definidos sobre otros objetos y estos a la vez tener definidos atributos sobre otros objetos y así hasta tener varios niveles de profundidad, se está ante un problema que sólo puede ser resuelto haciendo uso de la recursividad.

Por ejemplo si se está comparando dos objetos complejos o_1 y o_2 y los valores de un atributo para los anteriores objetos son a la vez dos objetos o_3 y o_4, para poder calcular la semejanza entre los dos primeros es necesario calcular primeramente la semejanza entre los objetos o_3 y o_4 si estos dos últimos objetos tuvieran a la vez atributos definidos sobre otros objetos ocurriría lo mismo (ver Figura 3.19). De la misma forma, si se define un atributo en un dominio conjuntivo sobre un referencial de objetos difusos complejos para un objeto o_1, al comparar el valor del atributo con el valor del mismo atributo en otro objeto será necesario tener en cuenta el parecido entre cada uno de los objetos involucrados en los subconjuntos que dan valor a los atributos. El ejemplo (3.9) muestra cómo se realiza este proceso.

Ejemplo 3.9. *Se quiere almacenar la información de las* líneas de investigación *en un grupo de investigación. Cada línea de investigación esta caracterizada por el siguiente conjunto de atributos* nombre, jefe, integrantes*, cada uno de estos atributos tiene las siguientes características.*

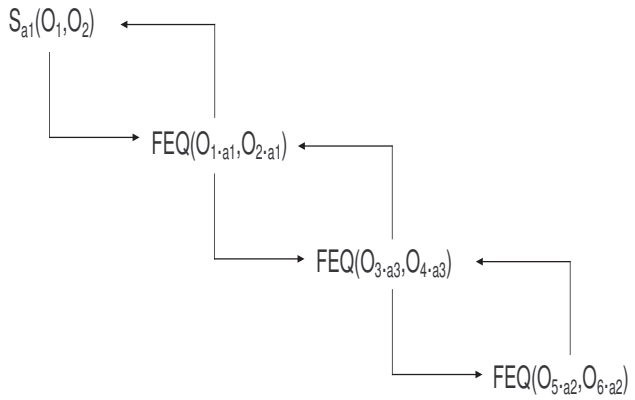


Figura 3.19: Recursividad en la comparación entre objetos complejos.

- *Nombre del grupo es un atributo preciso que representa el nombre por el que se conoce el grupo de investigación.*
- *Jefe del grupo de investigación, es un objeto del tipo* empleado *presentado en el ejemplo (3.8).*
- *Integrantes, representa los empleados que forman parte del grupo de investigación, cada integrante tiene un grado de participación en el grupo. Teniendo en cuenta estas características se utilizará un dominio conjuntivo con referencial formado por objetos de la clase* empleado.

Se tiene dos objetos de la clase línea de investigación *caracterizados de la siguiente manera:*

línea1:nombre=001, jefe=David, integrantes=1.0/David+0.3/Fernando+0.8/Esther

línea2:nombre=002, jefe=Esther, integrantes=1.0/Esther+0.6/David+0.4/Fernando

Se desea comparar las dos líneas de investigación por lo que es necesario realizar los siguientes cálculos.

Comparar los atributos de linea1 *y* linea2 *y luego hacer la agregación para determinar el parecido. Como el atributo* Jefe *es un objeto se requiere retomar la comparación entre objetos para determinar el parecido entre los jefes de las líneas. De forma similar sucede al comparar los integrantes de las dos líneas. El atributo integrantes está definido por un dominio conjuntivo sobre un referencial de objetos complejos. En este caso se requiere invocar en varias ocasiones la comparación entre objetos.*

El resultado del proceso de comparación entre las dos líneas se muestra a continuación.

1. $FEQ(linea1.nombre, linea2.nombre) = \mu_S(001, 002) = 0.0$

2. $FEQ(linea1.jefe, linea2.jefe) = \mu_S(David, Esther)$

   Para obtener $\mu_S(David, Esther)$ *se requiere comparar los atributos de David y Fernando y luego hacer la agregación. Los resultados de este proceso se muestra en la tabla 3.14.*

   $\mu_S(Dadid, Fernando) = 0.17$

3. $FEQ(linea1.integrantes, linea2.integrantes) = \mu_S(\{1.0/David + 0.3/Fernando$

   $+0.8/Esther\}, \{1.0/Esther + 0.6/David + 0.4/Fernando\})$

   Para obtener el parecido de los integrantes de las dos líneas se utiliza las ecuaciones (3.17) y (3.18). El resultado se obtiene calculando el parecido entre los objetos que componen los valores de los atributos. Como estos objetos son del tipo investigador *se utiliza un procedimiento similar al de comparar los* jefes *de las líneas. El resultado de esta comparación es:*

   $\mu_S(\{1.0/David + 0.3/Fernando + 0.8/Esther\},$

   $\{1.0/Esther + 0.6/David + 0.4/Fernando\}) = 0.6$

Finalmente el parecido entre las líneas aplicando la agregación (orness=0.2) es:

$$\mu_S(linea1, linea2) = 0.2max(0, 0.15, 0.6) + (1 - 0.2)min(0, 0.17, 0.6) = 0{,}12$$

En el ejemplo, (3.9), se muestra como se requiere de recursividad para obtener el parecido entre objetos complejos cuando alguno de sus atributos se define sobre otro objeto (ver Figura 3.20).

Se ha presentado como comparar dos objetos, pero qué sucede si al comparar dos objetos, alguno de sus atributos es nulo. Cuando ocurre esto el modelo está preparado para realizar la comparación y retornar un valor. La forma que tiene el modelo de gestionar los valores nulos se presenta en el siguiente apartado.

3.2.6. Gestión de valores nulos.

En los trabajos de Marín y otros, [Mar01, VCMP98], se propone una solución para el problema que se origina al comparar atributos de dos objetos en el cual uno de ellos tiene

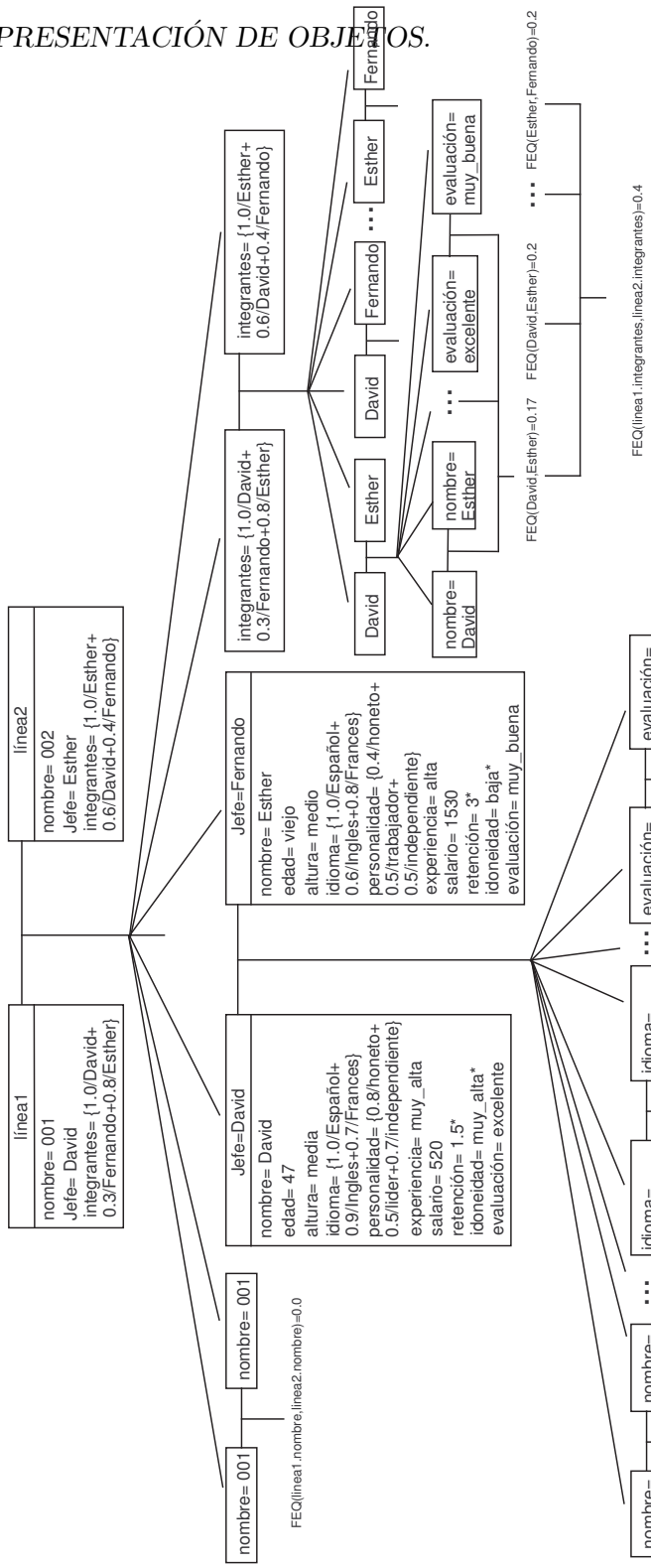


Figura 3.20: Recursividad en el ejemplo (3.9).

un valor nulo. La solución propuesta parte de analizar la naturaleza del valor y a partir de esto define varias aproximaciones:

- Si el nulo es no aplicable la semejanza debe ser 0.0.
- Si el nulo es desconocido, se puede tomar una postura optimista y considerar que la semejanza es 1. El valor de la semejanza se puede bajar con lo cual se toman actitudes menos optimistas
- Si se desconoce cuál de las anteriores interpretaciones es la correcta se está en un caso similar al anterior.

El modelo propuesto en esta investigación utiliza las aproximaciones señaladas anteriormente para gestionar los valores nulos. En este sentido se define cuatro formas distintas de comparar nulos:

- Al comparar dos valores nulos el resultado será siempre 1.0.
- Al comparar un valor nulo con otro valor el resultado dependerá de la estrategia definida por el usuario. En este sentido se definen tres estrategias:
  - Pesimista: supone que la semejanza entre dos atributos si uno de ellos es nulo es igual a 0,0.
  - Optimista: el parecido entre dos atributos del cual se desconoce el valor de uno de ellos es 1,0.
  - Indiferente: en la comparación se descartan los atributos en los que el valor de uno es nulo.

Como se aprecia no existen valores intermedios en la comparación, aproximación que se tendrá en cuenta en próximas versiones del modelo. Además, los valores nulos son utilizados como respuesta al proceso de inferencia del valor de un atributo inferido, en el cual ninguna de sus reglas se dispara por encima del umbral especificado para el sistema de reglas (los atributos inferidos serán presentados en próximo apartados).

Con los ejemplos mostrados en el anterior apartado y la gestión de valores nulos queda resuelto el problema de la comparación entre objetos complejos. Sin embargo, en los ejemplo mostrados hasta este momento se ha considerado que todos los atributos tienen la misma importancia. En el siguiente apartado se introducirá la utilización de perspectivas de comparación para objetos.

3.2.7. Perspectiva de comparación para los objetos difusos.

Hasta aquí se ha presentado cómo el modelo es capaz de permitir el trabajo con objetos complejos y cómo estos objetos pueden ser comparados para determinar su parecido. En los modelos de bases de datos orientadas a objetos consultados, la comparación entre objetos complejos sólo utiliza la igualdad difusa como operador de comparación entre los atributos de dos objetos. Si embargo, las posibilidades actuales de este modelo, incorporando varios operadores difusos para los diferentes dominios más la posibilidad de definir pesos que indiquen la importancia de cada atributo en el proceso de comparación, permiten generalizar la propuesta de comparación, de forma tal que se adapte su semántica a un determinado contexto.

De esta forma quedan definidas para una clase las perspectivas de comparación que incluyen:

- Un identificador o nombre para la perspectiva.
- Un conjunto de pesos referidos a la importancia de cada atributo de la clase en el proceso de comparación.
- Un conjunto de perspectivas u operadores difusos que indica que operador se utilizará para comparar cada par de atributo.
- La especificación de un valor para el número de *orness* que se utilizara al hacer la agregación.
- Una valor que indica el tratamiento de los valores nulos en el proceso de comparación.

Analizando los elementos necesarios para definir una perspectiva puede notarse que las perspectivas en sí incluyen un proceso de recursividad en dependencia de la naturaleza del atributo.

Para atributos definidos sobre un referencial de objetos complejos se debe especificar con qué perspectiva se desea comparar y en caso contrario se especifica el operador que se quiere utilizar (ver Figura 3.21).

La figura 3.21 muestra las características de recursividad presentes en la perspectivas de comparación. En el ejemplo de la figura se tienen cuatro clases. La clase c_1 se describe por n atributos. El atributo a_1 y a_n esta definido sobre un referencial de valores y los atributos a_2 y a_3 sobre un referencial de objetos. Para esta clase se define una perspectiva $c_1.pers$,

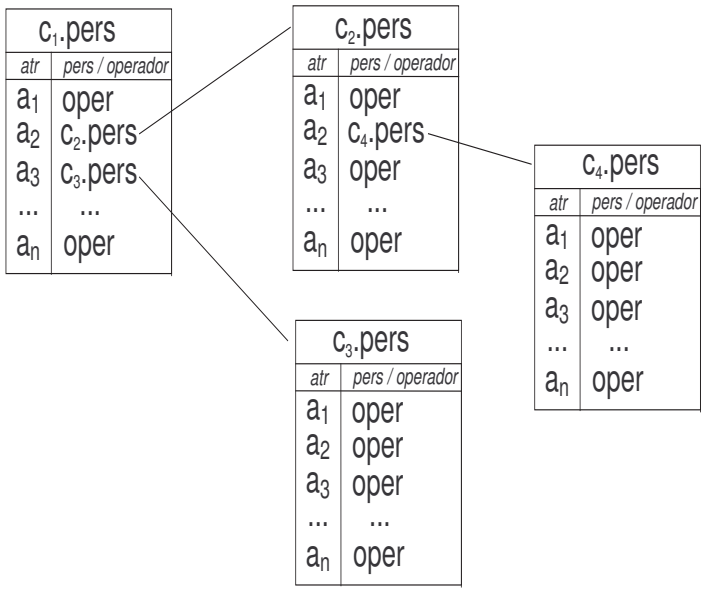


Figura 3.21: Perspectiva de comparación, recursividad.

en la cual se especifica un operador para los atributos a_1 y a_n y una perspectiva para a_2 y a_3. Las perspectivas asignadas a estos atributos, estarán de la misma forma definidas para la clase de los objetos.

De esa forma el usuario podrá definir cuantas perspectivas estime conveniente para una clase, en dependencia de los numerosos criterios que puedan existir para hacer la comparación entre objetos (ver Figura 3.22). En esta figura se describe el proceso de comparación entre objetos teniendo en cuenta perspectivas previamente definidas.

Además de estas perspectivas el modelo permite definir perspectivas para los dominios conjuntivos definidos sobre un referencial de objetos. Las características de estas perspectivas se mostrarán en el siguiente apartado.

3.2.8. Perspectiva de comparación para dominios conjuntivos.

Cuando es necesario determinar el parecido entre dos atributos definidos por medio de un dominio conjuntivo con un referencial de objetos (ver Epígrafe 3.2.3) se requiere comparar cada uno de los objetos que componen el subconjunto valor del atributo. Para estos objetos, como se presentó en el anterior apartado, es posible definir perspectivas

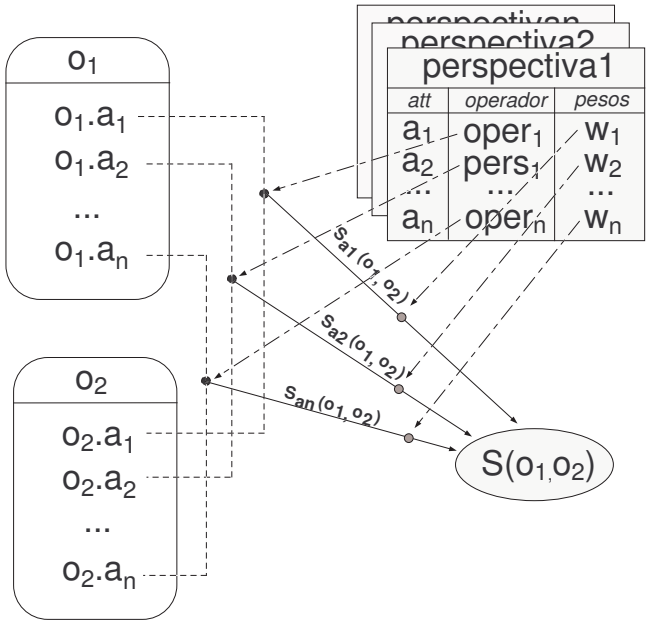


Figura 3.22: Comparación usando perspectiva de comparación.

de comparación. Es conveniente que estas perspectivas puedan ser utilizadas también al momento de comparar objetos incluidos en un subconjunto.

Para permitir esto en el modelo es posible definir perspectivas en los dominios conjuntivos. Estas perspectivas tendrán las siguientes características:

- Un identificador o nombre para la perspectiva.
- El operador que se utilizará para comparar los subconjuntos. Estos operadores puede ser FEQ (ver Epígrafe 3.2.3), GIS (ver Ecuación 3.17) o GIN (ver Ecuación 3.21).
- La perspectiva de comparación que se utilizará para comparar los objetos que forman el referencial.
- Método de comparación. Este parámetro solo se define cuando el operador es FEQ. Sus posibles valores son: FEQ, COG, AVG (ver Epígrafe 3.2.3)

Esta perspectiva a diferencia de la anterior, es una perspectiva asociada a un dominio. Las perspectivas definidas para el dominio podrán ser utilizadas al momento de definir perspectivas en los objetos, con lo que esto significa.

A continuación se muestra un ejemplo del uso de las perspectivas en la comparación entre objetos difusos.

Ejemplo 3.10. *Analizando el ejemplo de las líneas de investigación (ver Ejemplo 3.9) se decía que todos los atributos de la clase* línea *y de la clase* investigador *tenían la misma importancia. Si se quiere utilizar importancias diferentes para los atributos es necesario definir perspectivas de comparación para estas clases. Un ejemplo de perspectivas para esta clase se muestra en la figura 3.23.*

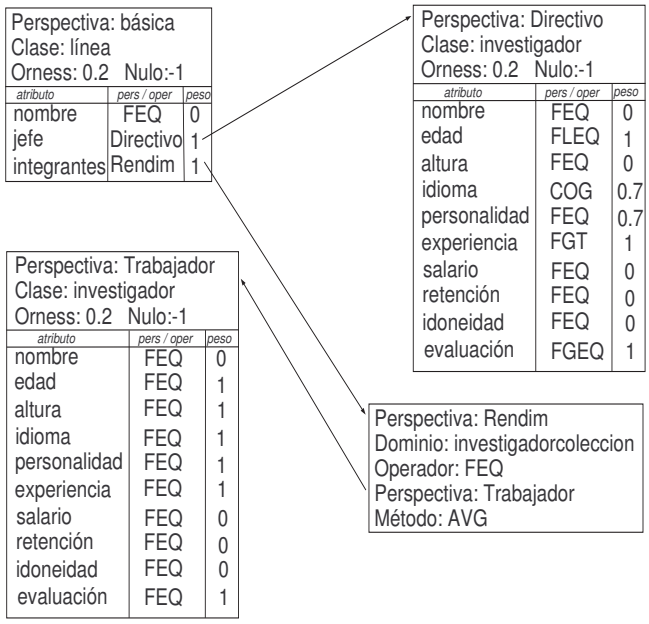


Figura 3.23: Ejemplo de perspectivas de comparación.

En la figura 3.23 se muestran una perspectiva para la clase línea. En este caso la perspectiva se llama básica*, define como número de orness 0.2 y el parámetro -1 para el tratamiento de nulos.*

En esta perspectiva no interesa, al momento de comparar, el atributo nombre*. Debido a esto se le asigna peso 0. Para los atributos* Jefe *e* Integrantes *se definen perspectivas de comparación.*

Para el atributo Jefe *se especifica una perspectiva previamente definida en la clase investigador. Al atributo* Integrantes *se le asigna una perspectiva definida para el dominio investigadorcollection, en esta perspectiva se define que se utilizará el operador FEQ para*

comparar los valores y que los objetos serán comparando teniendo en cuenta la perspectiva Trabajador *de la clase* investigador.

Volviendo al cálculo del parecido entre las dos líneas, pero en este caso teniendo en cuenta las perspectivas de comparación, se obtienen los siguientes resultados:

1. $FEQ(linea1.nombre, linea2.nombre) = \mu_S(001, 002) = 0.0$
2. $FEQ(linea1.jefe, linea2.jefe) = \mu_S(David, Esther)$

   $\mu_S(Dadid, Fernando) = 0.70$
3. $FEQ(linea1.integrantes, linea2.integrantes) = \mu_S(\{1.0/David + 0.3/Fernando$

   $+0.8/Esther\}, \{1.0/Fernando + 0.6/David + 0.4/Esther\})$

   Para obtener el parecido de los integrantes de las dos líneas se utiliza las ecuaciones (3.17) y (3.18). El resultado se obtiene calculando el parecido entre los objetos que componen los valores de los atributos. Como estos objetos son del tipo investigador *se utiliza un procedimiento similar al de comparar los* jefes *de las líneas. El resultado de esta comparación es:*

   $\mu_S(\{1.0/David + 0.3/Fernando + 0.8/Esther\},$

   $\{1.0/Esther + 0.6/David + 0.4/Fernando\}) = 0.68$

Finalmente el parecido entre las líneas es:

$$\mu_S(linea1, linea2) = 0.2max(0, 0.70, 0.68) + (1 - 0.2)min(1, 0.70, 0.68) = 0{,}684$$

A continuación se muestra otro ejemplo del uso de perspectivas de comparación en el modelo. En este caso se definen varias perspectivas para la clase *investigador*.

Ejemplo 3.11. *En el ejemplo de la clase* investigador *se desea a partir de un patrón existente de empleado, seleccionar qué empleado va a los puestos de dirigente, representante y trabajador. Para cada puesto de trabajo los requerimientos son diferentes y esos requerimientos se representarán como perspectivas de comparación entre el patrón especificado y los empleados existentes en la base de datos. El patrón definido por la dirección de la empresa es el siguiente:*

patron: nombre=patron, edad=viejo, altura=media, idiomas=1.0/español,0.8/inglés, 0.5/frances, personalidad=0.5/honesto,0.6/líder,0.7/trabajador, experiencia=alta,

salario=500, retenciones=0, idoneidad=5.5, evaluación=muy_buena

y se definen las siguientes perspectivas de comparación para la clase empleado *(ver Tabla 3.15).*

Cuadro 3.15: Perspectivas de comparación para la clase empleado

Atributo	Directivo		Representante		Trabajador	
nombre	0	FEQ	0	FEQ	0	FEQ
edad	1	FLEQ	0.8	NFLT	1	FEQ
altura	0	FEQ	0	FEQ	1	FEQ
idioma	0.7	COG	1	FEQ	1	FEQ
personalidad	0.7	FEQ	0.5	FEQ	1	FEQ
experiencia	1	FGT	0.8	FGEQ	1	FEQ
salario	0	FEQ	0	FEQ	0	FEQ
retención	0	FEQ	0	FEQ	0	FEQ
idoneidad	0	FEQ	0	FEQ	0	FEQ
evaluación	1	NFGEQ	0.8	FGEQ	1	FEQ

A la perspectiva Directivo no le interesan los atributos nombre, altura, salario, retención e idoneidad, por eso les asigna como peso 0.0. Luego considera como atributos más relevantes para el puesto la edad, la experiencia y la evaluación, peso 1.0. Y a los atributos idioma y personalidad le asigna un peso de 0.7.

Los operadores que se definen para la perspectiva son posiblemente igual difuso (FEQ) para el atributo personalidad con lo cual se pretende encontrar un empleado con una personalidad muy parecida a la fijada por el patrón. Para la edad se define el operador necesariamente menor o igual difusos (FLEQ) de forma tal que se seleccionen empleados con una edad menor o igual que la del patrón (viejo). En otras palabras no se quiere empleados muy_viejos para ese puesto. En los idiomas se requiere que el empleado seleccionado pueda entender los idiomas representados en el patrón, por eso se utiliza el operador para determinar la consistencia entre dos subconjuntos difusos (COG). Se requiere de una experiencia posiblemente mayor que alta y por eso se utiliza el operador posiblemente mayor difuso (FGT) para este atributo. Finalmente la evaluación debe ser necesariamente mayor o igual a muy_bien, garantizando esto con el operador necesariamente mayor o igual difuso (NFGEQ) en el atributo evaluación.

De forma similar pueden analizarse las restantes perspectivas de comparación. Al determinar el parecido con cada una de las perspectivas se obtienen los siguientes resultados (ver Tabla 3.16):

Cuadro 3.16: Utilización de perspectivas en la clase empleado

Patrón/Empleado	Fernando	Esther	David
$\mu_S(patron, empleado, directivo)$	0.2	0.77	0.14
$\mu_S(patron, empleado, representante)$	0.32	0.32	0.32
$\mu_S(patron, empleado, trabajador)$	0.68	0.52	0.52

Finalmente el puesto de directivo es para Esther, el de Representante para David y el de Trabajador para Fernando.

El uso de cualquiera de los dominios explicados hasta este momento en la definición de una clase en el modelo, supone que el valor del atributo será introducido por el usuario directamente cuando se construya un objeto con atributos difusos. Sin embargo, en los sistemas de bases de datos existen los atributos inferidos, atributos que toman su valor a partir de los valores asignados a otros atributos. Al modelo de bases de datos difusas orientadas a objeto propuesto en esta investigación se le han realizado extensiones de forma tal que permita la definición de este tipo de atributos.

3.3. Atributos inferidos.

Al modelo se le incorpora la posibilidad de definir atributos inferidos dentro de una clase. Estos atributos toman su valor a partir de la evaluación de reglas difusas previamente definidas que utilizan como variables independientes otros atributos definidos en la clase. El valor del atributo se actualiza al crear una instancia de la clase o se actualizan los atributos de un objeto existente. Las principales características de la propuesta son:

- El uso de reglas difusas definidas teniendo en cuenta la filosofía de Mamdani [Jag95]. Las reglas propuestas por Mandani fueron extendidas para soportar el uso de operadores, quedando definidas de la siguiente forma:
  if x_i *oper* A_1 *and ... and* x_{nx} *oper* A_n *then y is* B_1 Donde: *oper* es uno de los operadores definidos en el modelo.

- La variable en el consecuente puede estar definida sobre cualquiera de los dominios difusos soportados por el modelo o sobre un dominio preciso. En el momento de definir el sistema de reglas se trabaja con un tipo de dominio no necesariamente igual al dominio final del atributo.

- De la misma forma las variables involucradas en los antecedentes de las reglas pueden estar definidas sobre cualquiera de los dominios difusos soportados hasta el momento por el modelo, con lo que esto significa. Se incluye el uso de objetos complejos con la gran potencia de modelado que esto supone.
- El método de inferencia utilizado se basa en el método de inferencia max-min (*max-min inference method*) [MA75].
- Para evaluar los antecedentes de cada regla se pueden utilizar los operadores difusos incluidos en el modelo, la utilización de cada uno dependerá del dominio difuso sobre el que está definido el atributo del consecuente analizado.
- Es posible definir para el sistema de reglas difusas un umbral a partir del cual se dispararán las reglas.
- El valor final del atributo se obtendrá a partir de realizar una agregación del resultado de todas las reglas.

El trabajo con los atributos inferidos, su definición, la evaluación del sistema de reglas y la obtención del valor final para el atributo requiere del flujo de proceso que se muestra en la figura 3.24. Un total de siete procesos están involucrados en la obtención del valor de un atributo inferido.

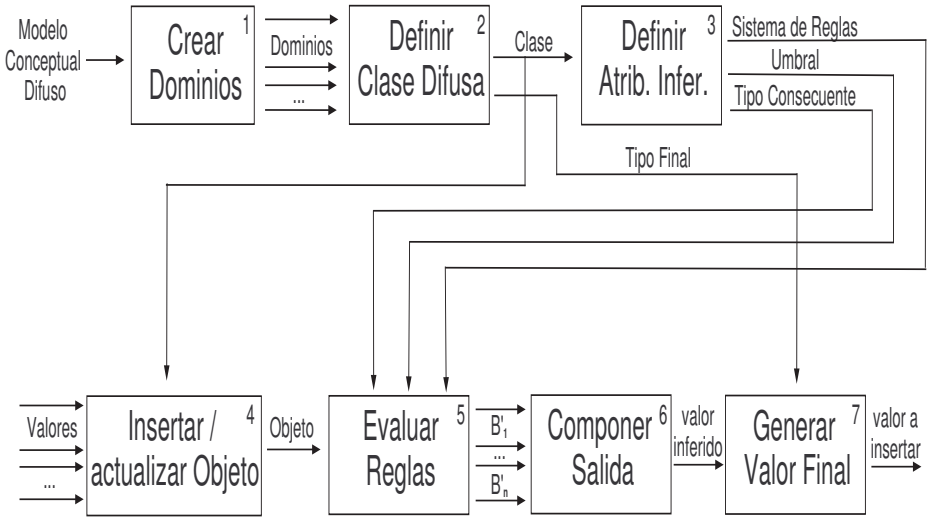


Figura 3.24: Proceso para los atributos inferidos.

El flujo comienza, como se muestra en la figura 3.24, por la creación y definición de los dominios difusos necesarios para representar un objeto. A partir de ese primer proceso quedan a disposición del usuario los dominios necesarios para crear una clase difusa dentro del modelo, según el proceso número dos.

Con una clase existente y teniendo en cuenta sus atributos es posible declarar un atributo inferido para la clase por medio del proceso tres. El resultado de este proceso será el sistema de reglas difusas, un umbral para considerar el cumplimiento de una regla dentro del sistema y un dominio para la definición del consecuente de las reglas. El dominio del consecuente, como se mostrará en próximos apartados, no tiene que ser el mismo que el dominio previamente definido en el proceso dos para el atributo inferido.

Una vez definido el atributo inferido, se inserta o actualiza un objeto dentro de la base de datos, dando valor a cada uno de sus atributos, proceso número cuatro. Ese objeto pasa a un quinto proceso donde teniendo en cuenta el sistema de reglas difusas, el umbral, el dominio del consecuente previamente definido para el atributo y los valores de cada uno de los atributos del objeto, se verifica el sistema de reglas y se obtiene el nivel de cumplimiento de cada una de las reglas.

Luego, según el método de inferencia especificado, se obtiene el valor de salida del sistema de reglas, proceso seis. Con el valor resultante y teniendo en cuenta el tipo final del atributo inferido, previamente definido al crear la clase, se determina el valor final que será insertado en el atributo inferido, proceso siete.

Los detalles generales de los principales procesos del flujo que son el proceso tres, cinco, seis y siete se mostrarán en los siguientes apartados. Se comienza describiendo la definición de reglas difusas.

Las reglas difusas son declaraciones del tipo *if-then* donde la premisa y el consecuente son proposiciones difusas del tipo *"x OPERADOR valor"* donde x es una variable. La premisa puede contener proposiciones combinadas mediante los conectores lógicos *AND* y *OR*. De forma general Mandani, [MA75], define una regla difusa de la siguiente forma:

$$IF\ x_i\ IS\ A_1^k\ AND|OR\ ...\ AND|OR\ x_n\ IS\ A_n^k\ THEN\ y_1\ IS\ B_1^k ... y_m\ IS\ B_m^k$$

Donde A_n^k son subconjuntos difusos con una relación de pertenencia previamente definida, la relación que representa una regla difusa está dada por:

$$R = I(T(A_1^k, A_2^k, ..., A_n^k), B_1^k, B_2^k ..., , B_n^k)$$

En el modelo propuesto en esta investigación se especifica la restricción de que sólo es posible definir un consecuente en la regla. Por otro lado, al definir una regla difusa en el modelo se permiten las siguientes características:

1. Las proposiciones de las reglas no siempre tienen que ser difusas, se pueden incluir variables precisas y operadores clásicos de comparación.

2. De la misma forma el consecuente puede ser una variable precisa.

3. En los antecedentes difusos se pueden utilizar los operadores definidos para cada uno de los dominios difusos.

4. Las variables incluidas en los antecedentes pueden estar definidas sobre cualquier de los dominio difuso soportados por el modelo.

Teniendo en cuenta la restricción señalada y las características mencionadas, las reglas difusas soportadas por el modelo para la definición de un sistema de reglas difusas como base del valor de un atributo inferido tendrán la siguiente forma:

$$IF\ x_i\ <operador>\ [A_1|valor]\ [AND|OR] \ldots [AND|OR]\ x_n\ <operador>\ [A_n|valor]$$

$$THEN\ y\ IS\ [B|valor]$$

Donde: x_i: atributo i del objeto.

$<$ *operador* $>$: operador compatible con el dominio del atributo x_i reflejado en el antecedente.

A_n *y* B: subconjuntos difusos representados por la relación de pertenencia $\mu_{A_n}(x_i)$ y $\mu_B(y)$, respectivamente y previamente definidos.

valor: un valor preciso

La precedencia de los operadores lógicos dentro de la regla estará dado por evaluar siempre las proposiciones conectadas por medio del operador AND y luego las conectadas por el operador OR. Si se quiere definir otro orden de precedencia se pueden utilizar paréntesis para definir con claridad la precedencia. La regla se evaluará a partir de estas reglas de precedencia. Ejemplos de definición de reglas en el modelo se muestran a continuación.

Ejemplo 3.12. *El siguiente ejemplo muestra algunas definiciones de reglas en el modelo. Se tiene una clase difusa* Empleado *caracterizada por los atributos reflejados en la tabla 3.11.*

Algunas reglas difusas relacionadas con atributos de esta clase pudieran ser:

```
a) IF salario <= 500 THEN retencion IS 0,

b) IF salario > 500 AND salario < 1500 THEN retencion IS 1.5,

c) IF m_edad FEQ muy_viejo THEN m_idoneidad IS baja,

d) IF m_edad FGT medio OR m_altura IS media THEN  m_idoneidad IS
media,

e) IF m_edad FGT medio AND name IS Alejandro AND m_altura FGEQ media
AND m_personalidad GIS
(0.9/honesto+0.8/trabajador+0.4/lider+0.6/creativo+
0.4/independiente) AND  m_experiencia FGEQ alta AND m_evaluacion IS
excelente THEN m_idoneidad IS muy_alta,
```

En el ejemplo (3.12) se definen varias reglas para atributos de la clase *Empleados* la descripción de cada regla se muestra a continuación.

- Las dos primeras reglas solamente utilizan atributos precisos tanto en las proposiciones como en el consecuente. En la regla *a)* en el antecedente se compara el atributo *salario*, definido sobre un dominio preciso, con una constante (500), si el salario es menor o igual que 500, la retención, que también es un atributo definido sobre un dominio preciso en este caso un número real, tomará valor 0. La regla *b)* tiene dos proposiciones relacionadas mediante el operador lógico *AND*, se compara el valor del atributo *salario* con dos constantes, si el *salario* es mayor que 500 y menor que 1000 la *retención* será de 1.5.

- La regla c) incorpora en el antecedente y en el consecuente atributos definido sobre dominios difusos. En este caso si la *edad* del *Empleado* es *muy_viejo* entonces la

idoneidad para el puesto es *baja*. El operador *IS* es equivalente al operador *FEQ* y funcionará en dependencia del tipo de dominio sobre le que se aplica.

- En *d)*, similar a la regla *c)* se utilizan tanto en la proposición como en el consecuente atributos difusos. En el antecedente se verifican dos atributos y se utiliza el operador lógico *OR* para enlazarlos. Es importante señalar que el atributo *m_edad*, en este caso, utiliza un operador difuso diferente a *FEQ*. En la regla se está expresando que si la *edad* del empleado es posiblemente mayor difusa que *medio* o su *altura* es media entonces la idoneidad es *media*. El valor *media* es una etiqueta lingüística existente en los dominios de los atributos utilizados en la regla.

- Por último en la regla *e)* se emplean en la proposición varios atributos difusos definidos sobre diferentes tipos de dominios y un atributo preciso. Además se emplean operadores difusos diferentes a *FEQ* en dependencia del tipo de dominio del atributo.

Estos ejemplos de definición de reglas da una idea de las posibilidades del modelo en este sentido. Finalmente las reglas deben ser combinadas para formar un sistema de reglas difusas a partir del cual se inferirá el valor de un atributo.

3.3.1. Sistema de reglas difusas.

Un atributo inferido tomará su valor a partir de un sistema de reglas difusas previamente definido por el usuario. Las características de estas reglas fueron analizadas en el apartado anterior. Ahora se analizarán las características del sistema de reglas difusas que pueden especificarse para un atributo.

Un sistema de reglas difusas es un conjunto de reglas difusas que son evaluadas de forma paralela. Para obtener el resultado final del sistema se evalúa cada regla de forma individual y luego son combinadas por medio de funciones de agregación. Aunque esta aproximación no es la única, es la más utilizada en sistemas expertos convencionales [Jag95]. El sistema de reglas estará formado por n reglas con antecedentes basados en los atributos de una clase definida en el sistema, de la forma que se muestra a continuación.

$R1 : IF\ x_i < operador > [A_1|valor]\ AND|OR...AND|OR\ x_n < operador > [A_n|valor]$
$THEN\ y\ IS\ B|valor$
$R2 : IF\ x_i < operador > [A_1|valor]\ AND|OR...AND|OR\ x_n < operador > [A_n|valor]$
$THEN\ y\ IS\ B|valor$
...
$Rn : IF\ x_i < operador > [A_1|valor]\ AND|OR...AND|OR\ x_n < operador > [A_n|valor]$
$THEN\ y\ IS\ B|valor$

En el modelo, el sistema de reglas difusas que se asocie a un atributo debe tener como único consecuente el atributo en cuestión y las reglas se analizarán de forma paralela, disparándose cuando sobrepasen el umbral definido por el usuario. Este umbral se define por defecto en el valor 0.0. Algunos ejemplos de sistemas de reglas se muestran a continuación.

Ejemplo 3.13. *En la clase difusa* Empleados *se definen un sistema de reglas difusas para inferir el valor del atributo* retención. *Para esta clase los atributos* salarios *y* retención *se definen sobre dominios precisos y representan el salario que se le paga a un empleado en un mes de trabajo y las retenciones que por tal concepto se le realizan. Para calcular el por ciento sobre el salario que representa la retención se definen rangos de valores y en dependencia del monto del salario así será la retención.*

El sistema de regla difusa que se define para este atributo quedaría de la siguiente forma:

```
IF salario <= 500 THEN retencion IS 0,

IF salario > 500 AND salario < 1500 THEN retencion IS 1.5,

IF salario >= 1500 THEN retencion IS 3
```

Para el sistema de reglas se define un umbral de 0.0 y se basa en comparar el salario con una constante. La primera regla expresa que si el salario es mayor o igual de 500 entonces, el por ciento correspondiente a la retención es de 0. La segunda si el salario está entre 500 y 1550 la retención es de 1.5 y la última si el salario es mayor de 1500 la retención será de un 3 por ciento.

Ejemplo 3.14. *Este otro ejemplo define igualmente un sistema de reglas difusas para el atributo* idoneidad *de la clase* Empleados*. En este caso la* idoneidad *es un atributo difusos que depende su valor del valor que tomen otros atributos difusos. El sistema de regla se representa de la siguiente forma:*

```
R1: IF edad FGT medio AND altura FGEQ media AND personalidad GIN
(0.9/honesto+0.8/trabajador+0.4/lider+0.6/creativo+0.4/independiente)
AND experiencia FGEQ alta AND evaluacion IS excelente THEN idoneidad
IS muy_alta,

R2: IF edad FGT medio AND altura FGEQ media AND personalidad GIN
(0.9/honesto+0.8/trabajador+0.4/lider+0.6/creativo+0.4/independiente)
AND experiencia FGEQ alta AND evaluacion IS muy_buena THEN idoneidad
IS muy_alta,

R3: IF edad IS joven AND altura FGEQ media AND personalidad GIN
(0.9/honesto+0.8/trabajador+0.4/lider+0.6/creativo+0.4/independiente)
THEN idoneidad IS alta,

R4: IF edad FEQ muy_viejo THEN idoneidad IS baja
```

Este ejemplo (ver Ejemplo 3.14) refleja el sistema de reglas para el atributo *idoneidad* que depende de los valores de los atributos, también difusos: *edad, altura, m_personalidad y experiencia*, los dominio definido para cada atributo se puede ver en la tabla (3.11). Para este sistema de reglas se define un umbral de 0.0 por lo que cualquier cumplimiento de una de las proposición de una regla, disparará la regla.

El sistema de reglas pretende asignar un valor al atributo *idoneidad*. Este atributo está definido sobre un dominio atómico con representación semántica y con referencial continuo. El dominio básico del dominio está formado por las etiquetas *baja,media,alta,muy_alta* y definido sobre el rango [0,10]. En la primera regla *(R1)* se define la siguiente combinación de proposición:

- *edad FGT medio*: la edad del empleado es posiblemente mayor difusa que medio
- *altura FGEQ media*: la altura debe ser posiblemente mayor o igual difuso a media

- *personalidad GIN (0.9/honesto+0.8/trabajador+0.4/lider+0.6/creativo +0.4/independiente)*: la personalidad debe estar incluida en el subconjunto de referencia
- *experiencia FGEQ alta*: la experiencia debe ser posiblemente mayor o igual difuso a alta
- *evaluacion IS excelente*: la evaluación debe ser igual a excelente

Si se dispara esta regla el resultado de la idoneidad será muy alta (idoneidad IS muy_alta) con el grado de cumplimiento de la regla. El resto de la reglas son similares a la anteriormente expuesta pero con consecuentes diferentes. La regla número cuatro restringe el valor de la idoneidad a la *edad viejo*.

En este ejemplo se definió un umbral para el sistema de reglas de 0.0. Por lo tanto cualquier regla que se dispare generará un resultado final para el atributo. Si por el contrario se especifica un umbral mayor que 0.0 entonces solo se tendrá en cuenta una regla si su grado de cumplimiento es igual o mayor al umbral especificado.

Con el sistema de reglas definido, el modelo tiene que ser capaz de inferir el valor de un atributo cuando se inserte o actualice un objeto, este proceso de inferencia será descrito en el siguiente apartado.

3.3.2. Resolución del sistema de reglas.

En la figura 3.24 se muestran los procesos necesarios para definir y obtener el valor de un atributo inferido. En este apartado serán tratados los procesos de evaluar el sistema de reglas (proceso cinco), componer la salida (proceso seis) y generar el valor final (proceso siete).

3.3.2.1. Evaluar sistema de reglas

La evaluación de un sistema de reglas difusas se realiza utilizando métodos de inferencia. Existen numerosos métodos de inferencia en la literatura [Jag95]. Unos de los más sencillos y más utilizados en sistemas difusos es el método conocido como *max-min* introducido por Mandani en 1974 [MA75]. En este método un operador *min*, basado en una t-norma, es utilizado para la conjunción en la premisa de las reglas y en la función de implicación y un operador *max* para la agregación, definiéndose de la siguiente forma [Jag95]:

$$\mu_{B'}(y) = \max_k(\beta_k) \tag{3.22}$$

Teniendo que k es el número de reglas definidas en el sistema difuso y β_k representa el valor obtenido al realizar la conjunción en la premisa de la regla k, este valor viene representado por:

$$B_k = \min_i \alpha_{i,k} \tag{3.23}$$

Donde $\alpha_{i,k}$ es el resultado obtenido al evaluar el antecedente i en la regla k que se determina por la siguiente ecuación:

$$\alpha_{i,k} = \sup_x(\mu_{A'_i}(x_i)) \tag{3.24}$$

En esta ecuación $\mu_{A'_i}(x_i)$ es el grado en que la proposición se cumple. El método *max-min* es la base del proceso de inferencia que realiza el modelo para determinar el valor de un atributo inferido.

El modelo teniendo en cuenta la ecuación (3.24) evalúa cada antecedente de una regla, en el cual, como se mostró anteriormente pueden incluirse operadores difusos en dependencia del tipo del consecuente. Los valores obtenidos para cada $\alpha_{i,k}$ son tenidos en cuenta para obtener, por cada regla del sistema que se dispara un valor para β_k. De esta forma se desarrolla el proceso cinco (ver Figura 3.24).

Para componer la salida del sistema de reglas, proceso seis de la figura 3.24, se realiza la agregación de todos los β_k, teniendo en cuenta la ecuación (3.22), obteniéndose un subconjunto difuso compuesto por todas los β_k resultantes como resultado o un valor preciso.

Sólo resta obtener el valor final para el atributo inferido, proceso siete (ver Figura 3.24). El valor final para el atributo dependerá del tipo final definido para el atributo inferido, por lo tanto en el próximo apartado se analiza la definición de tipos para un atributo inferido y luego se describe el proceso de obtención del valor final del atributo.

3.3.2.2. Tipo del atributo inferido.

Los tipos que se definen para el atributo inferido tienen importancia especial en el proceso mediante el cual se infiere el resultado. El modelo que se está utilizando da varias posibilidades en este sentido.

Un atributo inferido se define para un atributo de una clase. Asociados al atributo inferido se especifican meta datos con toda la información referente a dicho atributo, como

por ejemplo: el sistema de regla difusas, el umbral del sistema de reglas y el tipo de dato sobre le cual se define el atributo. En la especificación del atributo inferido se utilizan dos tipos de datos o dominios, un tipo de dato para definir el consecuente de las reglas y otro para el valor final del atributo.

El dominio real del atributo dentro de la clase será el tipo final seleccionado, mientras que el tipo del consecuente sólo se utiliza para definir el sistema de reglas y realizar el proceso de inferencia. Tanto el tipo del consecuente como el tipo final deben haber sido definidos previamente en el modelo. De esta forma pueden definirse atributos inferidos según las siguientes combinaciones (ver Tabla 3.17).

Cuadro 3.17: Tipos de datos en los atributos inferidos

Tipo del Consecuente	**Tipo Final**
Preciso	Preciso Conjuntivo sobre valores.
Atómico con referencial continuo	Preciso Atómico con referencial continuo Atómico con referencial finito de objetos Conjuntivo sobre objetos
Atómico sin representación semántica	Atómico sin representación semántica Atómico con referencial finito de objetos Conjuntivo sobre objetos
Atómico con referencial finito de valores	Atómico con referencial finito de valores Atómico con referencial finito de objetos Conjuntivo sobre objetos
Atómico con referencial finito de objetos	Atómico con referencial finito de objetos
Conjuntivo sobre valores	Conjuntivo sobre valores

El consecuente de un atributo inferido puede ser cualquiera de los dominios difusos previamente estudiados, mientras que el tipo final podrá ser: un tipo preciso, un tipo igual al tipo del consecuente o un dominio conjuntivo con referencial definido por objetos del tipo del consecuente.

3.3.2.3. Obtención del valor final

A partir del subconjunto difuso resultante del proceso seis donde se compuso una salida del sistema de regla y teniendo en cuenta la relación existente entre *tipo del consecuente - tipo final* se desarrolla un proceso para obtener el resultado final. En este proceso se utiliza uno de los siguientes métodos.

1. Obtener como valor final el valor obtenido en el proceso anterior.
2. Obtención del valor del dominio del consecuente de mayor parecido con los valores obtenidos en el subconjunto difuso resultante del proceso anterior.
3. Defusificación del subconjunto resultante en el proceso anterior.

En el **primer método** el resultado final del atributo será exactamente el valor obtenido en el proceso seis (ver Figura 3.24). Este método es utilizado en las siguientes condiciones: cuando el tipo del consecuente y el final del atributo son precisos, cuando el tipo final es un dominio conjuntivo con referencial sobre valores o sobre objetos y cuando es un dominio atómico con referencial sobre objetos. Los ejemplo (3.15) y (3.16) muestran la utilización de este método.

Ejemplo 3.15. *Utilizando la clase* Empleado *se definirá el siguiente sistema de reglas difusas para inferir el valor de la* evaluación *de un empleado. Para el ejemplo se supondrá que la evaluación del empleado está definida como un dominio atómico con referencial finito de objetos y el tipo del consecuente es un dominio atómico sin representación semántica asociada. El sistema de reglas que se define para el atributo es el siguiente:*

```
R1:
IF edad FGT medio AND altura NFGEQ media AND
 personalidad GIN (0.9/honesto+0.8/trabajador+0.4/lider+0.6/creativo+
 0.4/independiente) AND experiencia FGEQ alta THEN evaluacion IS excelente,
R2:
IF edad FGT medio AND altura NFGEQ media AND personalidad GIN
(0.9/honesto+0.8/trabajador+0.4/lider+0.6/creativo+
0.4/independiente) AND experiencia FGEQ alta THEN evaluacion IS
buena,
R3:
```

```
IF edad IS joven AND altura NFGEQ media AND personalidad
GIN (0.9/honesto+0.8/trabajador+0.4/lider+0.6/creativo+
0.4/independiente) THEN evaluacion IS buena,
R4:
IF edad IS muy_viejo THEN evaluacion IS regular
```

Al evaluar las reglas se obtiene los siguientes resultados: $\beta_1 = 0{,}88$, $\beta_2 = 0{,}5$, $\beta_3 = 0{,}8$, $\beta_4 = 0{,}6$. *Por lo tanto el resultado final de evaluar el sistema de reglas difusas es el subconjunto difusos* $B' = (0{,}88/excelente + 0{,}8/buena + 0{,}6/regular)$, *teniendo en cuenta el método de inferencia descrito anterior.*

Ejemplo 3.16. *Se tiene el atributo* retención *en la clase* Empleado *como un atributo inferido. Este atributo es preciso y para el se define un sistema de reglas difusas que al evaluarlo se obtiene como resultado el siguiente subconjunto.*

$B' = (0.5/0 + 0.7/1.5 + 0.6/1.5)$

Aplicando el método de inferencia se obtiene como resultado final para el atributo retención el valor 1.5.

En el **segundo método** se parte del subconjunto difuso resultante del proceso seis, pero se quiere que el valor final del atributo sea solamente uno de los elementos presente en el subconjunto. En todos los casos el subconjunto difuso resultante es interpretado con un referencial de objetos, por lo tanto entre sus elementos existe o puede determinarse una relación de semejanza. Se hace necesario definir un proceso que permita obtener el elemento que mayor pertenencia tenga al subconjunto resultante, teniendo en cuenta la relación de semejanza existente.

Como sólo interesa un valor del subconjunto, se interpreta con un sentido disyuntivo y en tal sentido se utiliza el operador *DIS* (3.16) previamente definido. El cálculo del resultado final para el atributo queda definido de la siguiente forma, ecuación (3.25).

$$l' = l_i \in L \mid maxl_i\{\{1...0, l_i, \}\ DIS\ B', i = 1...|L|\} \tag{3.25}$$

Donde: L es el conjunto de valores definidos para un dominio, l' es el valor resultante y B' el subconjunto resultante.

Este proceso se utiliza cuando se define el mismo dominio para el tipo final de atributo inferido y el consecuente, en los siguientes casos: atómico sin representación semántica,

atómico con referencial continuo y atómico con referencial finito de valores. El siguiente ejemplo (3.17) muestra el desarrollo de este proceso

Ejemplo 3.17. *Utilizando la clase* Empleado *se definirá el siguiente sistema de reglas difusas para inferir el valor de la* idoneidad *de un empleado. El tipo del consecuente y el tipo final del atributo son el mismo y están definidos sobre un dominio atómico con referencial continuo. El sistema de reglas que se define para el atributo es el siguiente:*

```
R1:
IF edad FEQ medio AND altura NFGEQ media AND personalidad GIN
(0.9/honesto+0.8/trabajador+0.4/lider+0.6/creativo+0.4/independiente)
AND experiencia FGEQ alta AND evaluacion IS excelente THEN idoneidad
IS muy_alta,
R2:
IF edad FGT medio AND altura NFGEQ media AND
personalidad GIN
(0.9/honesto+0.8/trabajador+0.4/lider+0.6/creativo+0.4/independiente)
AND experiencia FGEQ alta AND evaluacion IS muy_buena THEN idoneidad
IS muy_alta,
R3:
IF edad IS joven AND altura NFGEQ media AND
personalidad GIN
(0.9/honesto+0.8/trabajador+0.4/lider+0.6/creativo+0.4/independiente)
THEN idoneidad IS alta,
R4:
IF edad IS muy_viejo THEN idoneidad IS baja
```

Al evaluar las reglas se obtienen los siguientes resultados: $\beta_1 = 0{,}5$, $\beta_2 = 0{,}85$, $\beta_3 = 0{,}0$, $\beta_4 = 0{,}6$. *Se obtiene como resultado de evaluar el sistema de reglas difusas el subconjunto difusos* $(0{,}85/muy\_alto + 0{,}6/bajo)$, *partiendo del método de inferencia que se utiliza y descrito anteriormente. Teniendo en cuenta que interesa como resultado final sólo uno de las dos etiquetas involucradas en el resultado se procede a realizar el siguiente cálculo.*

$$l' = max((1.0/muy\_alto)\ DIS\ (0.85/muy\_alto + 0.6/bajo),$$
$$(1.0/bajo)\ DIS\ (0.85/muy\_alto + 0.6/bajo))$$
$$l' = max_{l_i}(0.85, 0.6) = muy\_alto$$

Entonces para el ejemplo se obtendría el valor muy_alto como resultado final del atributo. Este valor será insertado en el atributo idoneidad *del objeto que se inserte o actualice.*

El **tercer método** sólo se aplica cuando el tipo final del atributo es preciso y el tipo del consecuente es un dominio atómico con referencial continuo. Este proceso parte del resultado obtenido en el proceso seis, donde se obtiene un subconjunto difuso con un referencial definido sobre un dominio atómico con referencial continuo.

Para obtener el valor final del atributo se utiliza un proceso de defusificación que transforma la salida difusa de las reglas en un valor preciso representado por un valor numérico. Para este propósito existen dos métodos básicos: determinación de centro de gravedad y la media de los máximos [Jag95].

En el modelo propuesto se utiliza la defusificación por medio de la determinación del centro de gravedad del subconjunto resultante. Para un ambiente de dominio continuo, como es el que se tiene, el cálculo del centro de gravedad se expresa de la siguiente forma, ecuación (3.26).

$$cog(B') = \frac{\int_y \mu_{B'}(y) y dy}{\int_y \mu_{B'}(y) dy} \tag{3.26}$$

En su forma discreta este cálculo se realiza según la ecuación 3.27.

$$cog(B) = \frac{\sum_{q=1}^{N_q} \mu_{B'}(y_q) y_q}{\sum_{q=1}^{N_q} \mu_{B'}(y_q)} \tag{3.27}$$

Donde: N_q: número de valores discretos definidos en el dominio

Un ejemplo de este método es el siguiente.

Ejemplo 3.18. *Para el ejemplo (3.17) ahora se desea que el valor del atributo sea preciso. El subconjunto resultante al evaluar el sistema de regla se muestra en al figura 3.25.*

Utilizando la ecuación (3.26) se obtendría 6.19 como resultado final del atributo. Este valor será insertado en el atributo idoneidad *del objeto que se inserte o actualice.*

Como forma de resumen se presenta la tabla (3.18) donde se reflejan las combinaciones tipo final del atributo-tipo de su consecuente y el método utilizado para generar el valor

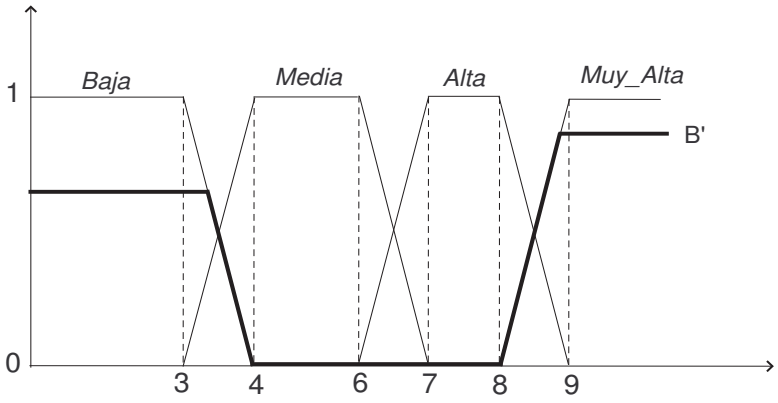


Figura 3.25: Resultado obtenido para el atributo *Idoneidad*.

final.

Hasta aquí se han presentado todos los elementos teóricos que sustentan la utilización de atributos inferidos difusos dentro del modelo. En apartados anteriores de mostró como comparar dos objetos con atributos difusos. Pero, de qué forma deben ser tratados los atributos inferidos en la comparación, una breve reflexión sobre este tema se presenta en el siguiente apartado.

3.3.3. Atributos inferidos en la comparación entre objetos.

Como fue analizado en el epígrafe 3.2, la comparación entre dos objetos descrito por atributos difusos o dos objetos complejos pasa por la comparación entre sus atributos. Si uno o varios de esos atributos son inferidos y teniendo en cuenta que su valor depende del valor de otros atributos, queda la pregunta de si interesa o no usar los atributos inferidos para comparar.

En el modelo que se presenta en esta investigación, por defecto los atributos inferidos participan como cualquier otro atributo de la comparación. Sin embargo, después de haber presentado las perspectivas de comparación (ver Epígrafe 3.2.7), queda claro que el usuario tiene todas las libertades para definir su propio esquema de comparación. De esa forma puede definir un peso a los atributos inferidos con lo cual define su interés en que participe o no en la comparación entre objetos.

Cuadro 3.18: Tipos de datos en los atributos inferidos

Tipo Consecuente	**Tipo Final**	**Método**
Preciso	Preciso	Mantener valor resultante
	Conjuntivo sobre valores	Mantener valor resultante
Atómico sin representación semántica	Atómico sin representación semántica	Comparación entre conjuntos
	Atómico con referencial finito de objetos	Mantener valor resultante
	Conjuntivo sobre objetos	Mantener valor resultante
Atómico con referencial continuo	Preciso	Defusificación
	Atómico con referencial continuo	Comparación entre conjuntos
	Atómico con referencial finito de objetos	Mantener valor resultante
	Conjuntivo sobre objetos	Mantener valor resultante
Atómico con referencial finito de valores	Atómico con referencial finito de valores	Comparación entre conjuntos
	Atómico con referencial finito de objetos	Mantener valor resultante
	Conjuntivo sobre objetos	Mantener valor resultante
Atómico con referencial finito de objetos	Atómico con referencial finito de objetos	Mantener valor resultante
Conjuntivo sobre valores	Conjuntivo sobre valores	Mantener valor resultante

Cuadro 3.19: Constantes difusas en el modelo.

Constante	Descripción
[a,b,c,d]	Distribución de posibilidad trapezoidal. Utilizada en consultas difusas.
[n,m]	Intervalo entre n y m. Utilizada en consultas difusas.
#n,l	Valor difuso aproximado. Utilizada en consultas difusas.

3.4. Constantes difusas.

En el modelo se definen un grupo de constantes difusas que pueden ser utilizadas para realizar consultas difusas sobre los datos almacenados. Estas constantes parten de los trabajos publicados por Galindo, [Gal05, Gal99]. En este modelo se da soporte a las siguientes constantes.

Estas constantes tienen su significado sólo al realizar consultas difusas sobre los datos (ver Ejemplo 3.19).

Ejemplo 3.19. *Utilizando la clase investigador y los objetos que en ella se definieron, algunas consultas que se pueden realizar utilizando las constantes difusas son:*

1. *Obtener los empleados con una idoneidad aproximadamente de 6 y un margen de 0.8.*

   $\mu_S(Fernando.idoneidad, \#6, 0.8) = 0.225$

   $\mu_S(Esther.idoneidad, \#6, 0.8) = 0.375$

   $\mu_S(David.idoneidad, \#6, 0.8) = 0.0$

2. *Obtener los empleados con una idoneidad entre 6 y 10*

   $\mu_S(Fernando.idoneidad, [6, 10]) = 1.0$

   $\mu_S(Esther.idoneidad, [6, 10]) = 0.0$

   $\mu_S(David.idoneidad, [6, 10]) = 1.0$

3. *La distribución de idoneidad exigida es la función trapezoidal con los parámetros [5.3,6,9.2,10]. En qué medida los empleados tienen la idoneidad exigida.*

   $\mu_S(Fernando.idoneidad, [5.3, 6, 9.2, 10]) = 1.0$

   $\mu_S(Esther.idoneidad, [5.3, 6, 9.2, 10]) = 0.29$

   $\mu_S(David.idoneidad, [5.3, 6, 9.2, 10]) = 0.63$

Con la descripción del uso de constantes difusas en el modelo se han presentado todas las características soportadas por el modelo utilizado en esta investigación para representar y manipular vaguedad al nivel de atributos. Elementos de su implementación en un gestor de bases de datos con soporte objeto-relacional serán analizados en otros capítulos. Ahora se mostrará otro nivel donde pueden aparecer vaguedad en las bases de datos orientas a objetos y como esta vaguedad es representado por el modelo.

3.5. Nivel de definición.

En el apartado anterior se analizó la presencia de vaguedad en el nivel de atributo de una clase, específicamente cuando no existe seguridad sobre el valor de un atributo, y se presentó como esta puede ser representada y manipulada a través del modelo de bases de datos orientados a objetos propuesto. Sin embargo, la vaguedad también puede estar presente en el momento de definir clases y objetos.

La vaguedad en la definición de un objeto surge cuando no se está seguro si un objeto tiene o no un determinado atributo. Cuando ocurren situaciones de este tipo se corre el riesgo de que se asuman valores nulos para los atributos no incluidos en el objeto. Por lo general la incertidumbre que pueda existir en la aplicabilidad de un atributo a un objeto pasa por una incorrecta definición de la clase. En tal sentido es deseado disponer de un tipo capaz de permitir expresar la vaguedad en los objetos y en las clases.

En el modelo propuesto por Marín y otros [Mar01, MVP00] se incorporan los tipos difusos como una alternativa para representar y manipular la incertidumbre que pueda surgir en el nivel de definición.

3.5.1. Tipos difusos.

Cuando se tiene un gran conocimiento del problema y se quiere trabajar con diferentes niveles de precisión en la definición de las clases se puede utilizar la definición de tipos difusos que se hace en el modelo de Marín y otros , [Mar01, MVP00]. Los niveles de cada esquema estarán definidos por el conocimiento que se quiera representar de un objeto.

Para algunos tipos de problemas es conveniente tener una definición de la clase por niveles y un mecanismo de instanciación de objetos que permita utilizar el nivel más adecuado y suficiente para representar la naturaleza de un objeto. En esto se basa el concepto de *Tipo Difuso*, [Mar01, MVP00].

La definición de *Tipo Difuso* permite enfocar los siguientes problemas de la modelación de objetos, [MVP00]:

1. Trabajar datos semiestructurados en los que, aunque hay cierta información sobre su estructura, esta es demasiado irregular para ser representada usando un enfoque relacional u orientado a objeto.

2. La presencia de gran cantidad de datos en un esquema conocido pero con una presencia abundante de valores nulos. Se pueden usar *tipos difusos* y mecanismos de instanciación que permita crear instancias que incorporen sólo un alfa-corte de los mismos.

3. En aquellos casos donde en el esquema existen atributos que indican la presencia o ausencia de información sobre otro conjunto de atributos se pueden utilizar *tipos difusos* que incorporen los atributos con grados de pertenencia determinados por capas. Cada instancia incorporara la capa que sea válida.

4. La existencia de clases artificiales en una jerarquía de herencia para agrupar características de subclases semejantes en estructura y comportamiento.

La estructura de un *Tipo Difuso* se define por medio de una estructura difusa, que no es más que un subconjunto difuso que tiene como referencial el conjunto A de todos los atributos posibles en el modelo y que se aplica en el intervalo [0, 1].

De aquí que cada atributo definido en el tipo tendrá un grado de pertenencia asociado. Entonces la estructura S queda definida de la siguiente forma.

$$S = \mu_S(a_1)/a_1 + \mu_S(a_2)/a_2 + ... + \mu_S(a_n)/a_n \tag{3.28}$$

Donde *a1, a2,. . . , a2* son los atributos que caracterizan la clase.

Al crear un objeto a partir de un *Tipo Difuso* se especificará de forma explícita cual será el $\alpha - corte$ de atributos a utilizar para representar el nuevo objeto. De esta forma es posible incorporar incertidumbre en el nivel de definición de una base de datos orientada a objetos.

Los tipos difusos dentro del modelo tienen las siguientes funcionalidades:

- En el modelo se pueden definir tipos difusos. Estos tipos difusos podrán tener todos los niveles que requiera el usuario.

- Los tipos difusos pueden estar formados por atributos difusos y por atributos precisos, aunque siempre los tipos difusos serán tratados como clases difusas dentro del sistema.

- Un Tipo Difuso puede ser el dominio de un atributo asociado a una clase.

- Es posible comparar dos objetos de un Tipo Difuso, como ya veremos, utilizando tres operadores diferentes.

El siguiente ejemplo muestra la definición de un Tipo Difuso.

Ejemplo 3.20. *Un ejemplo de tipo difusos es el siguiente, se tiene la entidad Persona Asegurada caracterizada por los siguientes atributos: nombre, apellido, DNI, dirección, tipo de coche, placa, color, número de bolsa de seguro para el coche, cantidad de incidentes, riesgo. Aquí se puede presentar el siguiente problema, si la persona no tiene coche los atributos tipo de coche, placa, color, número de bolsa de seguro, cantidad de incidentes quedarán vacíos. Un análisis similar ocurre si no tiene contratado un seguro para el coche. Este ejemplo es un esquema con atributos de presencia o ausencia. Para modelar esta entidad se utilizará un Tipo Difuso que quedaría definido de la siguiente forma.*

Se definen tres niveles de atributos:

1. *primer nivel, $\alpha = 1.0$, con los atributos nombre, apellido, DNI, fecha_nacimiento, dirección;*

2. *segundo nivel, $\alpha = 0.8$, con los atributos tipo de coche, placa, color;*

3. *tercer nivel, $\alpha = 0.6$, los atributos número de bolsa de seguro para el coche, cantidad de incidentes, riesgo (ver Figura 3.26)*

En este ejemplo de definición de tipos difusos se muestra la posibilidad de utilizar atributos definidos sobre cualquiera de los dominios difusos definidos en el modelo. Este es el caso del atributo *riesgo* del nivel 0.6 el cual está definido sobre un dominio difuso.

El valor asignado a los niveles no tiene una semántica asociada que represente alguna imperfección en el modelo. Sólo es utilizado para la identificación de los diferentes niveles del Tipo Difuso.

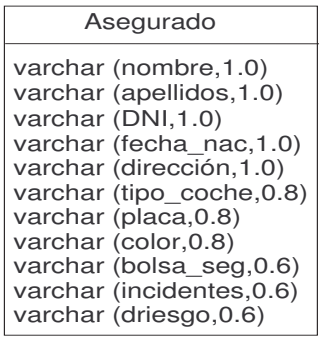


Figura 3.26: Tipo Difuso *Persona Asegurada*.

3.5.2. Operadores en Tipos difusos.

Los tipos difusos como se mostró anteriormente, son clases difusas dentro de la base de datos (el ejemplo Asegurado tiene un atributo definido sobre un dominio difuso, por lo que el Tipo Difuso es tratado como una clase difusa). De esta forma es posible utilizar el operador de comparación entre objetos difusos *FEQ*, pudiendo comparar dos objetos creados de un Tipo Difuso. Cuando se comparan dos objetos de un Tipo Difuso es posible encontrarse con las siguientes situaciones:

1. Los dos objetos que se quieren comparar pertenecen a un mismo nivel del Tipo Difuso por lo tanto incorporan los mismo atributos.

2. Cada objeto de los que se quieren comparar pertenecen a diferente nivel del Tipo Difuso por lo tanto no incorporan los mismo atributos.

En la primera situación no existe problema pues el operador realiza la comparación de la misma forma que la realiza para otros objetos difusos complejos. Sin embargo, la segunda situación plantea el siguiente problema:

"Si se comparan dos objetos de un Tipo Difuso definidos mediante el operador FEQ *se obtendrá un resultado afectado por la existencia de valores nulos y como consecuencia un grado de parecido bajo que dependerá de número orness utilizado"*

Este resultado es cierto pero no absoluto, pues al no tener ambos objetos los mismos atributos definidos se consideran diferentes. Pero qué sucede si lo único que interesa es saber cuán parecidos son los dos objetos en los atributos que sí tienen definidos en común. Por estas razones se definen tres estrategias para comparar los tipos difusos: pesimista,

optimista e indiferente. Estas estrategias responden las opciones presentadas al momento de tratar valores nulos en la comparación entre objetos (ver Epígrafe 3.2.6).

La opción pesimista supone que la semejanza entre dos atributos si uno de ellos es desconocido (no incorporado en el objeto) es igual a 0,0. En la optimista, el parecido entre dos atributos del cual se desconoce el valor de uno de ellos es 1,0. Y una indiferente donde se descartan los atributos pertenecientes al menor nivel del tipo difusos y se realiza la comparación teniendo en cuenta sólo los atributos coincidentes.

Para desarrollar estas estrategias fue necesario realizar algunas definiciones que ayudan a comprender el proceso.

Definicion 3.1. ***(Nivel de especificación de objeto -*** NO_n***)****: Se define el Nivel de especificación de un objeto definido sobre un Tipo Difuso y se nota por* NO_n *como el mínimo de los niveles de pertenencia que tienen los atributos que aparecen en el componente estructural del objeto.*

$$NO_n = \min_{\alpha \in A} \mu_S(a) \tag{3.29}$$

Donde A es el conjunto de todos los atributos definidos para el Tipo Difuso y S es el soporte finito de atributos sobre los que se define el objeto y está representado de la forma:

$$S = \mu_S(a_1)/a_1 + \mu_S(a_2)/a_2 + ... + \mu_S(a_n)/a_n$$

Definicion 3.2. ***(Nivel de especificación para semejanza entre objetos - N(T,o1,o2))*** *Se define el Nivel de especificación para semejanza entre objetos definidos sobre un Tipo Difuso* T *y se nota por* $N(T, o_1, o_2)$ *como el máximo de los Niveles de especificación de objeto de cada uno de los objetos y se define según la siguiente ecuación.*

$$N(T, o_1, o_2) = max(NO_1, NO_2) \tag{3.30}$$

La comparación entre dos objetos definidos sobre Tipos Difusos se trabaja como una sentencia de tipo II pero en este caso en vez de considerar a X como el conjunto de todos los atributos que describen un Tipo Difuso se trabaja con $X^*_{N(T,o_1,o_2)}$ restringiendo el conjunto de atributos a aquellos que forman parte del $\alpha - corte$ representado por el Nivel de especificación para semejanza entre objetos, previamente determinado. El siguiente ejemplo ilustra esta situación.

Ejemplo 3.21. *Se tiene la siguiente estructura para el Tipo Difuso* TF_1

$S = 1/a_1 + 1/a_2 + 1/a_3 + 0.8/a_4 + 0.8/a_5 + 0.6/a_6 + 0.6/a_7 + 0.6/a_8 + 0.6/a_9$

Y se tienen dos objetos o_1 *y* o_2 *definidos sobre el* $\alpha -$ *corte 0.8 el primero y 0.6 el segundo.*

$S_{o1} = 1/a_1 + 1/a_2 + 1/a_3 + 0.8/a_4 + 0.8/a_5$

$S_{o2} = 1/a_1 + 1/a_2 + 1/a_3 + 0.8/a_4 + 0.8/a_5 + 0.6/a_6 + 0.6/a_7 + 0.6/a_8 + 0.6/a_9$

¿Qué atributos deben participar de la comparación entre o_1 *y* $o_2(S(o_1, o_2))$*?*

Se determina el Nivel de especificación de objeto, para cada objeto.

$NO_1 = min(1, 1, 1, 0.8, 0.8) = 0.8$

$NO_2 = min(1, 1, 1, 0.8, 0.8, 0.6, 0.6, 0.6, 0.6) = 0.6$

Se determina el Nivel de especificación para semejanza entre objetos, para estos dos objetos.

$N(TF_1, o_1, o_2) = max(0{,}8, 0{,}6) = 0{,}8$

Por lo tanto la comparación entre los objetos o_1 y o_2, se realiza teniendo en cuenta el componente estructural del nivel 0.8.

El operador *FEQ*, por defecto, utiliza la estrategia indiferente de comparación entre objetos de un Tipo Difuso. La estrategia indiferente utiliza el cálculo del Nivel de especificación para semejanza entre objetos para determinar en qué nivel se debe hacer la comparación. Además de este operador se definieron otros dos operadores *PES* y *OPT*, que utilizan la estrategia pesimista y optimista.

En las estrategias pesimista y optimista lo que se determina es el Nivel bajo de especificación para semejanza entre objetos, según la siguiente definición.

Definicion 3.3. ***(Nivel bajo de especificación para semejanza entre objetos - Nb(T,o1,o2))*** *Se define el Nivel bajo de especificación para semejanza entre objetos definidos sobre un Tipo Difuso* T *y se nota por* $Nb(T, o_1, o_2)$ *como el mínimo de los Niveles de especificación de objeto de cada uno de los objetos y se define según la siguiente ecuación.*

$$Nb(T, o_1, o_2) = min(NO_1, NO_2) \tag{3.31}$$

Al determinar el Nivel bajo de especificación para semejanza entre objetos, el modelo utiliza este nivel para hacer la comparación, colocando 0.0 como semejanza entre un valor y *NULL* si se utiliza la estrategia pesimista y 1.0 si la estrategia a utilizar es optimista.

En el ejemplo (3.21) quedaría el nivel bajo de especificación para semejanza entre objetos como $Nb(TF_1, o_1, o_2) = min(0.8, 0.6) = 0.6$ Por lo tanto la comparación pesimista

u optimista entre los objetos o_1 y o_2, se realiza teniendo en cuenta el componente estructural del nivel 0.6.

3.5.2.1. Perspectivas de comparación en Tipos difusos.

Por ser los tipos difusos un caso especial de objetos difusos complejos soportan todas las características de estos últimos, esto incluye la posibilidad de definir pesos o importancias para sus atributos, la posibilidad de definir que operadores utilizar para comparar los pares de atributos y por consecuente definir perspectivas de comparación.

3.6. Conclusiones parciales.

En este capítulo se realizó un recorrido por los elementos teóricos que dan soporte al modelo implementado en esta investigación. Se prestó especial atención a las extensiones realizadas al modelo de bases de datos difusas orientadas a objetos de Marín y otros, explicándose las principales características. Las características del modelo de bases de datos difusas orientadas a objeto con el que se trabaja se pueden resumir de la siguiente forma:

- Brinda soporte para el uso de varios tipos de dominios difusos tanto con cardinalidad atómica como conjuntiva.
- Los dominios soportados pueden ser definidos sobre referenciales de valores o referenciales de objetos.
- Fueron definidos operadores difusos para cada uno de los dominios difusos soportados por el modelo.
- Es posible definir atributos inferidos dentro de una clase, atributos que tomarán su valor a partir de los valores de otros atributos.
- Se incorpora un tipo especial de dominio difuso que son los objetos difusos complejos que pueden ser manipulados individualmente o como valor de un atributo en un objeto.
- Se trabajan los tipos difusos como elementos para representar la vaguedad en los niveles de definición y de conducta. Los objetos definidos sobre los tipos difusos pueden ser manipulados de firma individual o como valor de un atributo de un objeto, con lo cual se consideran los tipos difusos como un tipo especial de dominio difuso.

- Se incorpora al modelo la posibilidad de definir perspectivas de comparación para los objetos difusos complejos y para los tipos difusos.

Estas características del modelo propuesto en esta investigación lo convierten en un modelo bastante completo para representar la vaguedad que pueda existir en los niveles de atributos y de definición en una base de datos difusa orientada a objetos.

Cómo fueron implementadas cada una de los elementos mostrados en este capítulo y cómo el usuario puede interactuar con el son los objetivos del próximo capítulo de esta memoria.

Capítulo 4

Implementación

En el capítulo anterior fueron presentadas las características teóricas del modelo de bases de datos difusas orientadas a objetos propuesto en esta investigación. La implementación de dicho modelo puede realizarse en un gestor de bases de datos orientadas a objetos o en gestores de bases de datos que soportan las características objeto - relacionales. Sin embargo, en esta investigación se decidió realizar la implementación utilizando *PostgreSQL*.

PostgreSQL, como se refleja en el apéndice D, es un gestor de bases de datos que incorpora algunas características objeto - relacionales y se desarrolla y distribuye bajo la filosofía de software libre. Tiene una amplia comunidad de desarrolladores que han impulsado su utilización hasta convertirlo en uno de los gestores de bases de datos más utilizados a nivel mundial en su categoría [All07, McK07]. Actualmente dentro del mercado de software libre es considerado como el mejor gestor de bases de datos a partir de las características que soporta. Si además de esto sí considera que es muy simple usarlo si quieres cosas simples, y la posibilidad de realizar exenciones si se quieren cosas complejas, se convierte en una buena elección para desarrollar la investigación.

Un elemento importante en esta elección es la características de *PostgreSQL* de estar implementado sobre una plataforma de software libre. Esto permite su distribución y utilización de forma gratuita, con lo cual los resultados de esta investigación podrán ser utilizados en cualquier contexto. Este elemento es favorable para países como Cuba al cual se le prohíbe utilizar software desarrollado en los Estados Unidos, como es el caso del Gestor de Bases de Datos Oracle y por consiguiente las extensiones que sobre el mismo se realicen. Estos elementos definieron la elección de *PostgreSQL* para implementar el modelo propuesto en esta investigación, de forma tal que el autor pueda publicar los resultados y

utilizarlos en investigaciones posteriores.

En este capítulo se mostrarán los principales elementos, desde el punto de vista de implementación, que fueron necesarios realizar para trasladar el modelo de bases de datos difusas orientadas a objetos propuesto a *PostgreSQL*. Inicialmente se dan algunos elementos tecnológicos que condicionan la implementación del modelo en *PostgreSQL* y luego el capítulo sigue un orden similar al anterior. Se comienza por describir la implementación de los dominios difusos, para luego analizar como es el trabajo con objetos y objetos complejos, la comparación y la definición de perspectivas. Se dedica un apartado a la implementación de los atributos inferidos y luego se especifica la manipulación de los Tipos Difusos. En cada uno de los casos se describen las estructuras y objetos creados para lograr la implementación y la sintaxis de las funciones creadas para interactuar con el modelo. Por ultimo se realiza una descripción del cliente Web desarrollado para trabajar con el modelo.

4.1. PostgreSQL, consideraciones iniciales.

En primer lugar, la implementación del modelo debe adaptarse a la arquitectura de *PostgreSQL*. La arquitectura de *PostgreSQL* está basada en el modelo cliente/servidor conocido como "proceso por usuario". En el servidor se definen las bases de datos del usuario y otras bases de datos que pueden ser utilizadas como plantillas (*template*). La figura 4.1, muestra los principales elementos de esta arquitectura.

El usuario crea su base de datos en *PostgreSQL* utilizando una de las plantillas existentes, por ejemplo *template1*. Por medio de una aplicación cliente (frontend) se conecta con el servidor de bases de datos utilizando el proceso *Postmaster* (supervisor) que iniciar la conexión. Luego de autentificar la conexión solicitada, el proceso *Postmaster* abre un nuevo proceso en el servidor, permitiendo desde este momento la comunicación de la aplicación del cliente con el proceso y a su vez con las bases de datos disponibles. La comunicación entre la aplicación y la base de datos se realiza por medio de sentencias *SQL*.

Teniendo en cuenta la arquitectura de *PostgreSQL*, la implementación se realizó mediante una nueva plantilla incorporada al servidor de bases de datos. Esta plantilla se denomina *pg4DB* (pOSTgRESQL Fuzzy Object Relational DataBase) e incluye todos los objetos y funciones que dan soporte al modelo propuesto en esta investigación. De esta forma el usuario dispone de una plantilla que puede utilizar cuando desee crear una base de datos con soporte para la utilización de atributos difusos y/o el trabajo con objetos difusos. Esta plantilla se distribuye junto con otros ficheros que permiten la instalación

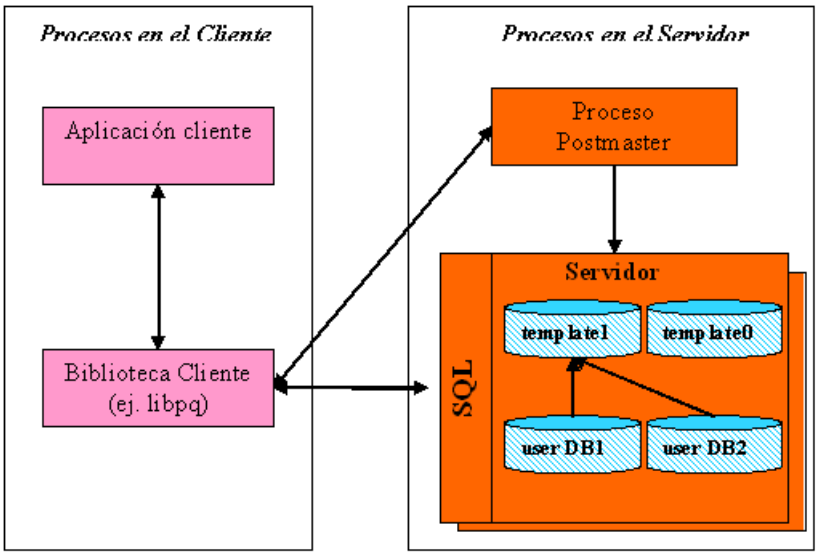


Figura 4.1: Arquitectura de PostgreSQL.

de la misma en cualquier servidor *PostgreSQL*. Los ficheros que componen el paquete de distribución pg4DB pueden consultarse en el apéndice F.

Partiendo de que el modelo propuesto en esta investigación es un modelo de bases de datos difusas orientadas a objetos, se utilizó en la implementación del mismo las características objeto-relacionales soportadas por *PostgreSQL*. *PostgreSQL* es el único gestor de bases de datos de la denominación de software libre que soporta características objeto-relacionales. Si embargo PostgreSQL no alcanza todos los niveles de los gestores de bases de datos objeto-relacional existente. En este sentido este gestor se caracteriza por a la libertad de trabajar con tipos de datos definidos por el usuario, definir índices sobre estos tipos de datos, dar soporte a objetos largos, permitir la herencia entre tablas, etc [1].

El principal elemento en los sistemas objeto-relacionales que permite identificar una orientación a objetos, es la posibilidad de trabajar con Tipos de Datos Abstractos (TDA) definidos por el usuario y el soporte de la herencia entre estos tipos. Teniendo en cuenta estos elementos, se presenta una breve comparación entre las características objeto-relacionales enunciadas por Stonebraker, [Sto99] y *PostgreSQL* (ver Tabla 4.1).

Como se puede observar en la tabla (4.1), *PostgreSQL* no es objeto-relacional propiamente. Puede considerarse que las principales características no soportadas por *Post-*

[1] http://archives.postgresql.org/pgsql-es-ayuda

Cuadro 4.1: Comparación entre las características del modelo objeto-relacional y PostgreSQL

Característica	**Objeto- Relacional** [Sto99]	**PostgreSQL**
TDA Compuestos.	Es posible definir TDA compuestos en el sistema.	Pueden definirse TDA compuesto. Al crear una tabla automáticamente se crea un TDA con la misma estructura que la tabla creada.
Métodos en un TDA	Se define el comportamiento de un TDA por medio de métodos asociados directamente a ellos.	No es posible definir métodos ni procedimientos asociados a un TDA.
Herencia.	Un TDA puede heredar la estructura y el comportamiento de otro TDA.	No existe la herencia entre TDAs.
Constructores.	Al crear un TDA se define, automáticamente, un constructor para insertar y modificar objetos del TDA.	Sólo se dispone de un único constructor, ROW, para crear objetos en todos los TDAs del sistema.
Almacenamiento.	Puede especificarse una tabla externa en la que se desean almacenar los objetos del TDA.	Sólo se almacenan los objetos en la propia tabla donde se usa el TDA como dominio de un atributo.
Anidar TDAs, colecciones.	Pueden definirse vectores de objetos de un TDA dentro de otro TDA. Esto permite modelar las relaciones uno a muchos.	No es posible definir vectores de objetos de un TDA.
Referencias a un TDA	Es posible definir atributos como una referencia a un TDA almacenado en una tabla. Este atributo tomará como valor un objeto de la tabla del TDA o de cualquiera de las tablas o tipos que hereden del TDA.	Las referencias se especifican utilizando el concepto de clave externa. La referencia no tiene en cuenta la jerarquía de herencia definida. No se puede trabajar con referencias a un TDA creado por el sistema.

greSQL, con respecto a la propuesta objeto-relacional es la imposibilidad de definir métodos y procedimientos asociados a un TDA y no soportar la herencia de métodos y procedimientos entre los TDAs, aunque si es posible la herencia entre tabla desde el punto de vista estructural. Estas son las principales dificultades encontradas para trasladar el modelo propuesto en esta investigación al gestor *PostgreSQL*.

Las carencia de *PostgreSQL* requirieron de un esfuerzo adicional para la implementación. Sin embargo, como se mostrará, utilizando las funciones definidas por el usuario y los catálogos del sistema, fue posible definir una estructura que permitió implementar el modelo teórico propuesto, simulando algunas de las características de la orientación a objetos. A continuación serán descritos los principales elementos de la implementación.

4.2. pg4DB, elementos generales.

Como se ha dicho, en este capítulo se parte del modelo de bases de datos difusas orientadas a objetos propuesto en el capítulo anterior y se realiza una traslación de cada una de las características al modelo soportado por *PostgreSQL*. Esta traslación genera la plantilla *pg4DB* (pOSTgRESQL Fuzzy Object Relational DataBase) que contendrá el modelo difuso para la base de datos. Para realizar esta traslación fue necesario apoyarse en las siguientes características de *PostgreSQL*.

- Herencia entre tablas, lo cual permitió definir una jerarquía de tablas capaz de dar todo el soporte requerido para los diferentes objetos difusos que se definan.

- Funciones definidas por el usuario. Estas funciones son definidas en el servidor de la base de datos de forma independiente, es decir no están relacionadas con ningún otro objeto de la base de datos. Se utilizaron para implementar todas las operaciones que se realizan sobre los objetos con atributos difusos y para desarrollar una interfaz entre el usuario final y el modelo. Todas las funciones fueron desarrolladas utilizando el lenguaje PLpg/SQL.

- Sobrecarga de funciones. Se sobrecargaron funciones, adaptando su comportamiento en dependencia de los parámetros introducidos. De esa forma las funciones pueden operar sobre objetos con atributos difusos de cualquier naturaleza.

- Utilización de atributos definidos sobre vectores multidimensionales.

- Utilización de los identificadores de objetos generados internamente por *PostgreSQL*.

En la plantilla resultante de la implementación *pg4DB* (ver Figura 4.2), se definió el esquema *fuzzy_schema*. En este esquema está toda la estructura que da soporte al modelo difuso, incluyendo las tablas para almacenar la definición de los tipos difusos, los valores difusos y las funciones del modelo. En el esquema *público* el usuario dispondrá de los dominios creados por el sistema, de las tablas con atributos difusos y de un conjunto de funciones que garantizan el trabajo y la comunicación con el modelo.

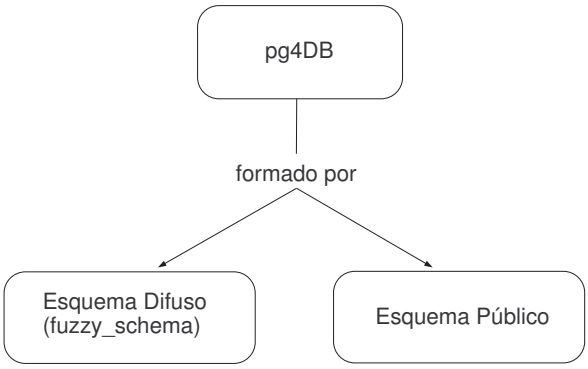


Figura 4.2: Esquemas definidos para la implementación.

A manera de resumen, para implementar la propuesta fue necesario definir los siguientes objetos en *PostgreSQL*, tabla (4.2).

Cuadro 4.2: Objetos que dan soporte a pg4DB

	Esquemas fuzzy_schema	Esquema public
Tablas	19	0
Funciones	144	69

Terminología empleada.

En el modelo teórico, propuesto durante esta investigación, se emplean terminologías que serán formalizadas a continuación con el objetivo de que que el lector entienda con claridad los aspectos de la implementación que serán descritos. De esta forma se tiene los siguientes términos:

Valor Difuso Es un valor difuso definido sobre uno de los dominios difusos soportados por el modelo.

Atributo difuso Son propiedades que toman como valor un Valor Difuso.

Objeto Los objetos están descritos por más de una propiedad. Las propiedades pueden estar definidas sobre dominios precisos o dominios difusos. Cuando están definidas sobre un dominio difuso toman como valor un Valor Difuso.

Objeto Complejos Los objetos complejos son objetos en los cuales al menos una de sus propiedades es otro objeto u otro objeto complejo.

Tabla con atributos difusos(*TWFA*) Se considera tabla con atributos difusos (*TWFA*), las tablas dentro de la base de datos que almacenan objetos con al menos una propiedad definida sobre un dominio difuso o tablas que almacenan objetos complejos.

Tipo Difuso Es un caso especial de objeto u objeto complejo en el cual se puede especificar un nivel de instanciación para el objeto, incorporando solamente las propiedades asociadas al un nivel.

Estructura de pg4DB.

La implementación del modelo propuesto, utilizando las características de *PostgreSQL* y teniendo en cuenta sus limitaciones desde el punto de vista de orientación a objetos, planteó el problema de encontrar una estructura para el modelo de datos que permita definir, almacenar y manipular objetos de una forma transparente para el usuario. Se requiere una estructura que permita realizar extensiones al lenguaje *SQL* soportado por *PostgreSQL*, de forma tal que pueda utilizarse una sintaxis lo más similar posible a la empleada en los sistemas orientados a objetos. La estructura del modelo de datos para *pg4DB* se muestra en la figura 4.3.

En este diagrama (ver Figura 4.3), se pueden identificar cuatro tipos de tablas:

Tablas de datos Son las tablas que dan soporte a las *TWFAs* del usuario y almacenan los valores para los atributos difusos. Están representadas por la primera jerarquía e incluye las tablas TFuzzyObject, TAbstracta_Domain, TDomain_CF y TFuzzyTable.

Tablas para la descripción de dominios difusos atómicos Almacenan la definición de los dominios difusos atómicos en la base de datos. Esta compuesta por la segunda jerarquía de tablas y la forman las tablas: TDomain, TDomain_WR, TDomain_DT, TDomain_DF.

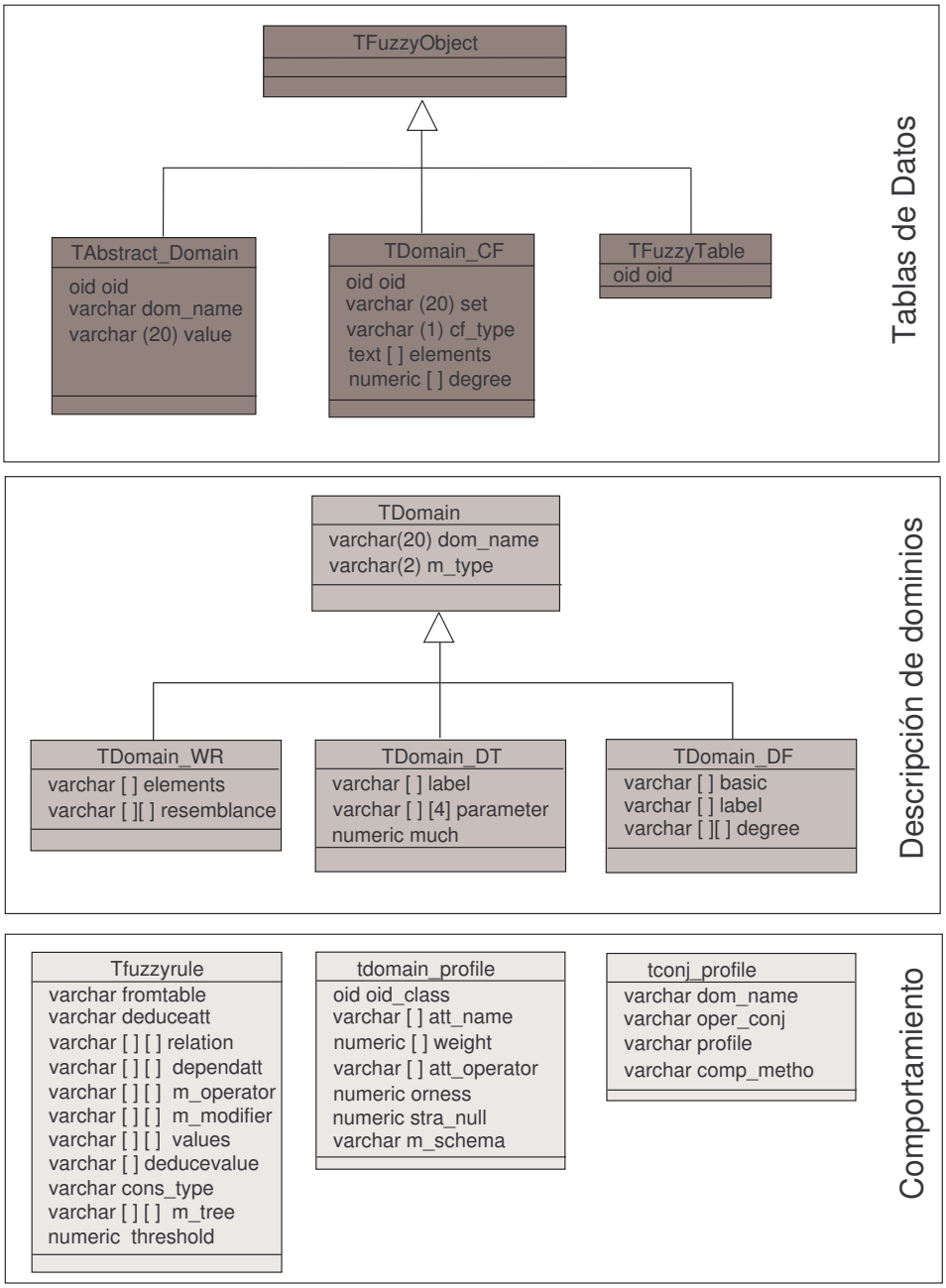


Figura 4.3: Jerarquía definida en pg4DB.

Tablas para definición de comportamiento Se encargan de almacenar la definición de parámetros que afectan el comportamiento de las *TWFAs* al momento de ser actualizadas o consultada. Tablas: tdomain_profile, tconj_profile y tfuzzyrule.

En los siguientes apartados se mostrarán los diferentes procesos implementados en *pg4DB*. Se detallarán la forma en que es almacenada la información y cuáles son las principales funciones definidas. En este punto sólo se presentarán las funciones que puede utilizar directamente el usuario del sistema. De las funciones internas del modelo sólo se mostrarán detalles cuando se considere necesario.

La presentación comienza por las extensiones realizadas al lenguaje de definición de datos (*DDL*) de *PostgreSQL* de forma que sea posible definir dominios difusos. Luego se trabajará con el lenguaje de manipulación de datos (*DML*) donde se extiende el *SQL* para permitir insertar objetos. En los apartados finales se especificarán las extensiones avanzadas realizadas a los lenguajes *DDL* y *DML* para soportar el trabajo con los atributos inferidos y los tipos difusos. Y en el último apartado se detallarán los elementos del lenguaje *DML* de consultas.

4.3. Dominios Atómicos

Un análisis de los diferentes tipos de dominios difusos que soporta el modelo permite dividirlo según su cardinalidad, desde el punto de vista de la implementación, en dos categorías : dominios atómicos y dominios conjuntivos (ver apartado 3.1). El análisis se inicia por los dominios atómicos.

Para los dominios atómicos es necesario, en primer lugar, almacenar la definición de cada uno de ellos. Analizando la definición de los dominios atómicos se nota que entre ellos existen atributos y métodos comunes como son: nombre del dominio y tipo de dominio. Mientras que cada uno de los dominios tiene sus propios atributos para almacenar su definición. Los atributos involucrados en cada dominio son:

- Atómico sin representación semántica: está definido por el vector de las etiquetas lingüísticas del dominio y un vector multidimencional con la relación de similitud definida entre las etiquetas.
- Atómico con representación semántica sobre referencial continuo: como se especificó en el capítulo anterior estos dominios están representados por funciones tra-

pezoidales. Por lo tanto sus atributos son: un vector con las etiquetas lingüísticas definidas para el dominio y un vector multidimensional con la segunda dimensión fijada a cuatro para almacenar los parámetros de las funciones.

- Atómico con representación semántica sobre referencial finito: en este dominio se necesitan tres atributos para caracterizar la definición del mismo, a saber: un vector con los valores del dominio básico, un vector con las etiquetas lingüísticas y un vector multidimensional con la relación de semejanza existente entre cada uno de los posibles valores del dominio.

Teniendo en cuenta las características de los dominio atómicos, se definió un diagrama de tabla basado en una generalización (ver Figura 4.4). Esta jerarquía de tablas sólo tiene el propósito de almacenar los meta datos de los dominios y es independiente a la estructura que da soporte a los objetos.

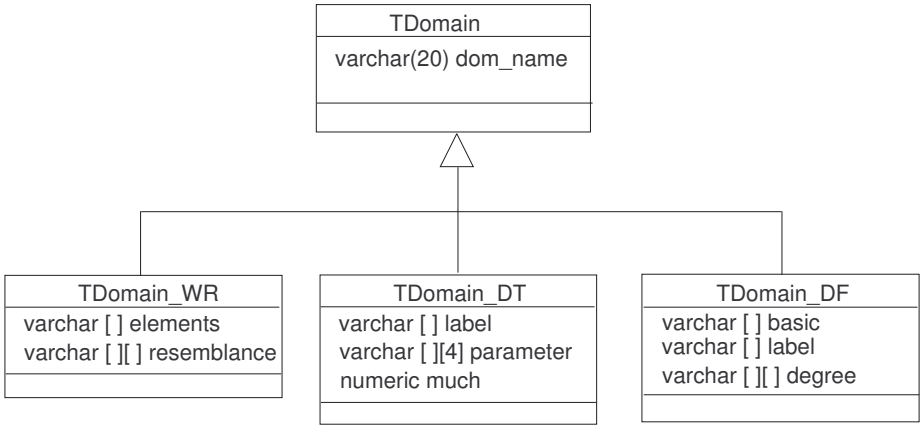


Figura 4.4: Tablas para almacenar la definición de los dominios atómicos.

Inicialmente se define una supertabla, representada por la tabla *TDomain*. De esta tabla heredarán todos los dominios atómicos.

La utilización de la tabla *TDomain*, permite identificar el tipo de un dominio atómico, para luego acceder a la tabla que corresponda y utilizar los meta atributos que definen el dominio. Gracias a esta tabla se pueden agrupar todos los dominios atómicos definidos en la base de datos, garantizando que puedan ser tratados de forma similar.

Como se muestra en la figura 4.4, además se definen las tablas TDomain_WR, TDomain_DT y TDomain_DF para almacenar los meta datos de cada uno de los dominios. A

continuación, mediante ejemplos presentados en el capítulo anterior, se muestra cómo se almacena un dominio de cada tipo (ver Ejemplo 4.1, Ejemplo 4.2 y Ejemplo 4.3).

Ejemplo 4.1. *En el caso de la calidad de un producto, ejemplo (3.1) se inserta el siguiente registro en la tabla TDomain_WR que representa la definición del dominio calidad.*

TDomain_WR		
set	***Elements***	***Resemblance***
calidad	{*alta, media, baja*}	{{*1, 0.8, 0.2*},{*0.8, 1, 0.4*},{*0.2, 0.8, 1*}}

Ejemplo 4.2. *Al definir el dominio altura correspondiente al ejemplo (3.15) se almacenaría en el modelo el siguiente registro.*

TDomain_DT			
dom_name	***label***	***parameter***	***much***
altura	{*joven, medio, viejo*}	{{*0,0,150,160*}, {*150,160,170,180*}, {*170,180,300,300*}}	*0*

Ejemplo 4.3. *Para el ejemplo de las plantas, ejemplo (**??**), es necesario definir un dominio, el cual se almacenará de la siguiente forma en la tabla TDomain_DF.*

TDomain_DF			
dom_name	***basic***	***Label***	***Degree***
Planta	{*1,2,3,4*}	{*bajo, intermedia, alto*}	{{*1,0.8,0,0*},{*0,1,1,0*}, {*0,0,0.7,1*}}

Se ha presentado de qué forma se almacena la definición de los dominios en la base de datos. Los dominios son definidos directamente por el usuario para lo cual se han creado funciones que permiten especificar la definición de un dominio como se mostrará más adelante.

Pero antes de ver como puede definirse un dominio se mostrará como el modelo gestiona los valores difusos que toma un atributo definido sobre un dominio atómico.

4.3.1. Gestión de valores de un dominio atómico

Como se explicó anteriormente, se tienen definidos los dominios de los atributos por medio de la tabla *TDomain* y las subtablas, TDomain_WR, TDomain_DT, TDomain_DF.

Estas tablas son utilizadas exclusivamente para almacenar las definiciones de los dominios y no para almacenar el valor que pueda tomar un atributo definido sobre uno de estos dominios.

Por lo tanto se hace necesario una estructura que permita el almacenamiento de los valores difusos y la utilización de estos dentro de la base de datos. Es conveniente que todos los datos definidos por medio de dominios atómicos difusos puedan ser manipulados de igual manera.

Con el objetivo de generalizar todos los valores difusos, definidos sobre dominios difusos atómicos, se ha creado la tabla *TAbstract_Domain*.

Esta tabla tiene un papel fundamental dentro de la estructura del modelo pues soporta todos los valores difusos que se introducen en la base de datos. Con este objetivo, esta tabla está dotada de tres atributos:

1. Identificador del objeto, es un atributo que identifica el registro dentro del servidor de bases de datos. Este atributo es generado internamente por PostgreSQL. *(oid)*
2. El nombre o identificador del dominio empleado para representar el valor. *(dom_name)*
3. El valor que toma ese atributo en un registro u objeto en particular. *(value)*

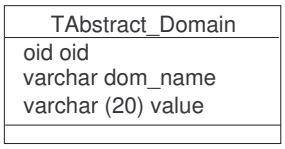


Figura 4.5: Tabla *TAbstract_Domain*.

Esta estructura da el soporte para almacenar los valores difusos, por lo tanto de ella heredarán las *TWFAs* que almacenan los valores difusos. Estas *TWFAs* se caracterizan por tener una sola propiedad que es el valor difuso.

Antes de ver cómo es que se introduce y se manipula un valor difuso en la base de datos, se mostrará de que forma se define un dominio atómico en *pg4DB*.

4.3.2. Creación de dominios atómicos

Al no tener la posibilidad de definir métodos en las tablas de la jerarquía reflejada en la figura 4.4, fue necesario recurrir a la característica que tiene *PostgreSQL* de definir

funciones del usuario y crear funciones que permitan incorporar nuevos dominios difusos en la base de datos. A partir de esto existe un procedimiento que es necesario seguir para definir un dominio atómico en el modelo (ver Figura 4.6).

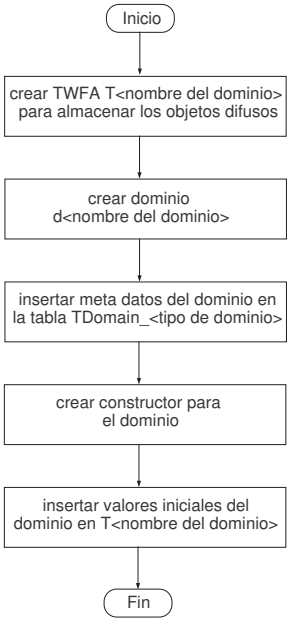


Figura 4.6: Procedimiento para crear dominios atómicos.

Como se muestra en la figura 4.6 el procedimiento consta de cinco procesos que culminan con crear toda la estructura necesaria en la base de datos para trabajar con un dominio atómico.

Este proceso lo realiza una función que es utilizada directamente por el usuario. Para cada uno de los dominios se definió una función. Estas funciones son la interfaz que debe utilizar el usuario para definir sus dominios en el modelo.

El usuario necesita especificar qué tipo de dominio desea crear y cuáles son sus parámetros. Por lo tanto, si se quiere utilizar en una *TWFA* un atributo cuyo valor difuso definido sobre un dominio atómico se debe inicialmente crear el dominio.

Las tres funciones que se definen, una para cada tipo de dominio atómico, son:

Create_Domain_WR para dominios atómicos sin representación semántica.

Create_Domain_DT para dominios atómicos con referencial continuo.

Create_Domain_DF para dominios atómicos con referencial finito.

En las tres funciones se definen parámetros diferentes, en dependencia de las características de cada dominio, pero en todos los casos uno de los parámetros es el nombre del dominio que se desea crear. Al ejecutar una de estas funciones el modelo se extiende para soportar los nuevos dominios. En la base de datos quedan creados los siguientes objetos (ver Figura 4.7).

1. Una nueva tupla en la tabla *TDomain* correspondiente al dominio, donde se representa la definición del dominio. El valor del atributo *dom_name* en esa tabla será el nombre del nuevo dominio (ver Figura 4.7-a)

2. Una nueva tabla *T<Nombre del Dominio>* que hereda directamente de la tabla *TAbstract_Domain* (ver Figura 4.7 b) y que será utilizada para almacenar los valores difusos definidos sobre el nuevo dominio.

3. Un nuevo dominio en términos de *PostgreSQL* con el nombre *D<Nombre del Dominio>*. Este dominio estará disponible para que el usuario lo asigne a un atributo que desee declarar en cualquier *TWFA* del modelo (ver Figura 4.7 c).

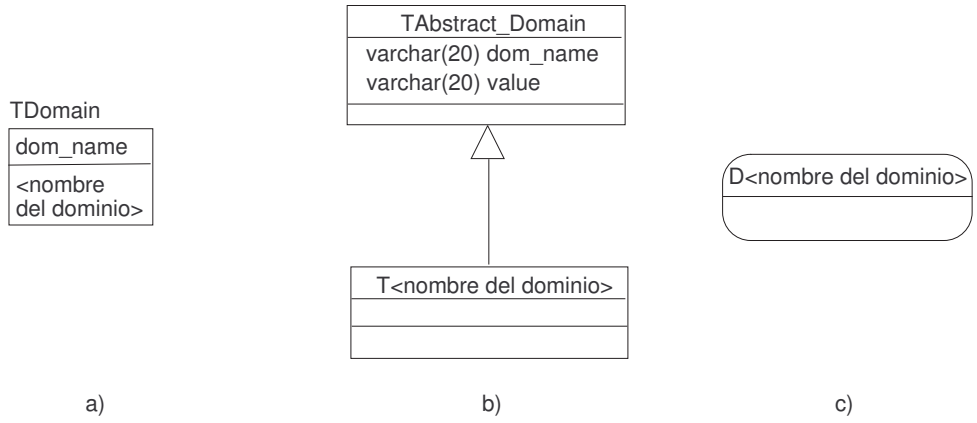


Figura 4.7: Resultado de la función *Create_Domain*.

El registro que se inserta en la tabla *TDomain* tomará como valor para el atributo *m_type* uno de los siguientes valores: WR, si es un dominio atómico sin representación semántica; DT, si es un dominio atómico definido sobre un referencial continuo y DF, si es un dominio atómico definido sobre un referencial finito de valores. Ver las tablas (4.1, 4.2, 4.3) como ejemplo. La sintaxis de cada una de las funciones para definir los diferentes tipos de dominios atómicos y ejemplos de su utilización se muestra en los siguientes apartados.

4.3.2.1. Dominios sin representación semántica.

Este tipo de dominio se puede crear por medio de la función *Create_Domain_WR ()* implementada en el modelo. La sintaxis se refleja en la tabla 4.3.

Función: Create_Domain_WR	
Create_Domain_WR (<domain_name>, <elements> , <resemblances>);	
Entradas:	
domain_name	**Tipo**: cadena de caracteres **Definición**: nombre del dominio que quiere crearse.
elements	**Sintaxis: '{**<*label*>, {**,** <*label*>}**}'** **Tipo**: cadena de caracteres **Definición**: etiquetas que forman el dominio. El parámetro estará formado por una lista de todas las etiquetas que se quieren definir.
resemblances	**Sintaxis: '{{{**<*resemblance*>, {**,**< *resemblance*>}**}**, {**,** **{**<*resemblance*>, {**,**< *resemblance*>}**}}}'** **Tipo**: valor numérico real en el rango [0,1] **Definición**: representa los valores de la relación de semejanza entre las etiquetas. Este parámetro es un vector bidimensional representando todos los valores de la relación en dependencia de la posición de los elementos en el parámetro *elements*.

Cuadro 4.3: Función *Create_Domain_WR*, sintaxis

Para definir el dominio *calidad* del ejemplo (3.1) puede escribirse el siguiente código:

```
SELECT Create_Domain_WR ('calidad','{alta, media, baja}','{{1.0,
0.8, 0.2},{0.8, 1.0, 0.4},{0.2, 0.8, 1.0}}');
```

Como se ha indicado, esta función crea inicialmente la estructura para el dominio (ver Figura 4.7) y luego inserta los meta datos del dominio en la tabla *TDomain_WR*.

Para el ejemplo presentado se crean los siguientes objetos en la base de datos (ver Figura4.8).

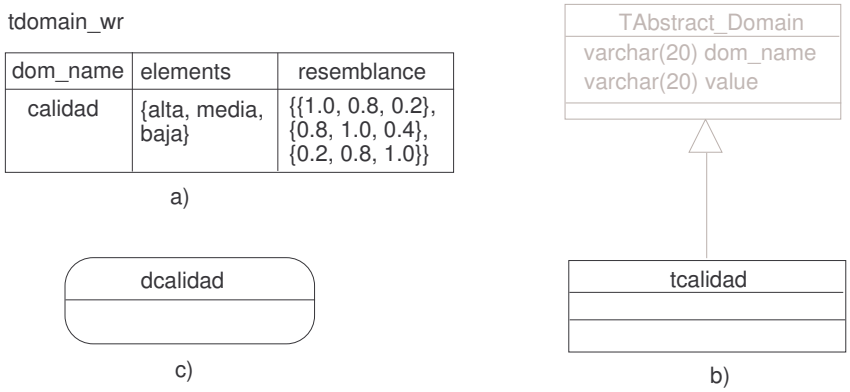


Figura 4.8: Resultado de la función *Create_Domain_WR*.

4.3.2.2. Dominio con referencial continuo.

Para la definición de dominios de este tipo al modelo, se implementó la función *Create_Domain_DT ()*. Esta función podrá ser utilizada por el usuario para definir dominios atómicos con referencial continuo (ver Tabla 4.4).

Función: Create_Domain_DT	
Create_Domain_DT ('<domain_name>',< elements>,<semantic>);	
Entradas:	
domain_name	**Tipo**: cadena de caracteres **Definición**: nombre del dominio que se desea crear.
elements	**Sintaxis: '{**<*label*> {**,** <*label*>}**}'** **Tipo**: cadena de caracteres **Definición**: etiquetas que se quieren definir para el dominio. El parámetro estará formado por una lista de todas las etiquetas.

semantic	**Sintaxis:** **'{{**<*a*>, <*b*>, <*c*>, <*d*>**}**, {, **{**<*a*>, <*b*>, <*c*>, <*d*>**}}}'** **Tipo**: valor numérico real. **Definición**: representa la semántica de las etiquetas definidas por los parámetros de las funciones que representan las distribuciones de posibilidad de las etiquetas definidas. Este parámetro es un vector de n filas y cuatro columnas que para representar todas las funciones relacionadas con las etiquetas respetando el orden definido en el parámetro *elements*.

Cuadro 4.4: Función *Create_Domain_DT*.

El dominio altura, ejemplo (3.3), es un ejemplo de este tipo de dominio y la sentencia *SQL* que permite especificar sus parámetros es la siguiente:

```
SELECT Create_Domain_DT ('altura','{bajo, medio, alto}',' {{0, 0,
150, 160},{150, 160, 170, 180}, {170, 180, 300, 300}}');
```

De forma similar a cuando se utiliza la función *Create_Domain_WR*, mostrada en el apartado anterior, al utilizar la función *Create_Domain_DT* se crean objetos en *PostgreSQL* para dar soporte a los valores de este tipo de dominio (ver Figura 4.7) y se inserta un nuevo registro en la tabla *TDomain_DT* (ver Ejemplo 4.2). El resultado en la base de datos de ejecutar esta función es el mostrado en la figura 4.9.

Para este tipo de dominio es posible definir un atributo que representa la distancia que se utilizará en los operadores difusos que incorporan la expresión mucho, tabla (3.4). En este caso se incorpora un nuevo atributo a la función para especificar este valor (ver Tabla 4.5).

Función: Create_Domain_DT	
Create_Domain_DT ('<domain_name>',< elements>,<semantic>,<much>);	
Entradas:	
nombre_dominio	**Tipo**: cadena de caracteres **Definición**: nombre del dominio que se quiere crear.

elemento	**Sintaxis: '{**<*label*> {**,** <*label*>}**}'** **Tipo**: cadena de caracteres **Definición**: etiquetas que van a definir en el dominio. El parámetro estará formado por una lista de todas las etiquetas que se quieren definir.
semantic	**Sintaxis: '{{**<*a*>**,** <*b*>**,** <*c*>**,** <*d*>**},** {**,** **{**<*a*>**,** <*b*>**,** <*c*>**,** <*d*>**}}}'** **Tipo**: valor numérico real. **Definición**: representa la semántica de las etiquetas definidas por los parámetros de las funciones que representan las distribuciones de posibilidad de las etiquetas definidas. Este parámetro es un vector de n filas y cuatro columnas que para representar todas las funciones relacionadas con las etiquetas respetando el orden definido en el parámetro *elements*.
much	**Tipo**: número real **Definición**: medida empleada en los operadores que utilizan la expresión mucho.

Cuadro 4.5: Función *Create_Domain_DT*.

4.3.2.3. Dominio con referencial finito.

De forma similar se definió la función *Create_Domain_DF ()* para los dominios atómicos con referencial finito. Esta función podrá ser utilizada por el usuario para definir dominios de este tipo. La función responde a la sintaxis mostrada en la tabla 4.6.

Función: Create_Domain_DF	
Create_Domain_DF (<*domain_name*>,<*basic*> , <*labels*> , <*degrees*>*);*	
Entradas:	
domain_name	**Tipo**: cadena de caracteres **Definición**: nombre del dominio a crear.

basic	**Sintaxis: '{**<*element*>, {**,** <*element*>}**}'** **Tipo**: cadena de caracteres. **Definición**: elementos básicos del dominio. El parámetro está formado por una lista de todos los elementos básicos.
labels	**Sintaxis: '{**<*label*>, {**,**< *label*>}**}'** **Tipo**: cadena de caracteres. **Definición**: etiquetas lingüísticas que se quiere definir en el dominio. El parámetro estará formado por una lista de todas las etiquetas.
degrees	**Sintaxis: '{{**<*degree*>, {**,**< *degree*>}**}**, {**,** **{**<*degree*>, {*','* <*degree*>}**}}}'** **Tipo**: valor numérico real definido en [0,1]. **Definición**: representa el grado de pertenencia del elemento básico a la etiqueta según el orden dado en el parámetro *básico*. Este parámetro es un vector bidimensional donde se representan cada una de las etiquetas definidas.

Cuadro 4.6: Función *Create_Domain_DF*.

El dominio planta del ejemplo (**??**) queda definido según la sentencia:

```
SELECT Create_Domain_DF ('planta', '{1, 2,3, 4}','{baja, media,
alta}',' {{1, 0.8, 0, 0},{0, 1, 1, 0},{0, 0, 0.7, 1}}');
```

De forma similar a las otras funciones *Create_Domain* presentadas, al utilizar esta función se crean los objetos que dan soporte a los valores difusos de este tipo de dominio (ver Figura 4.7) y se insertar un nuevo registro en la tabla *TDomain_DF* (ver Ejemplo 4.3). En la figura 4.10 se muestra el resultado en la base de datos al ejecutar la función.

Se presentaron como crear tres dominios atómicos con diferentes características. El resultado, en la base de datos del usuario, de los códigos mostrados en los tres apartados anteriores es la creación de tres dominio: *dcalidad*, *daltura* y *dplanta* y la siguiente estructura de *TWFA* en la base de datos (ver Figura 4.11).

Cada una de las tablas *Tcalidad*, *Taltura* y *Tplanta*, almacenan los valores difusos relacionados con los dominios *dcalidad*, *daltura* y *dplanta* respectivamente. Además, se

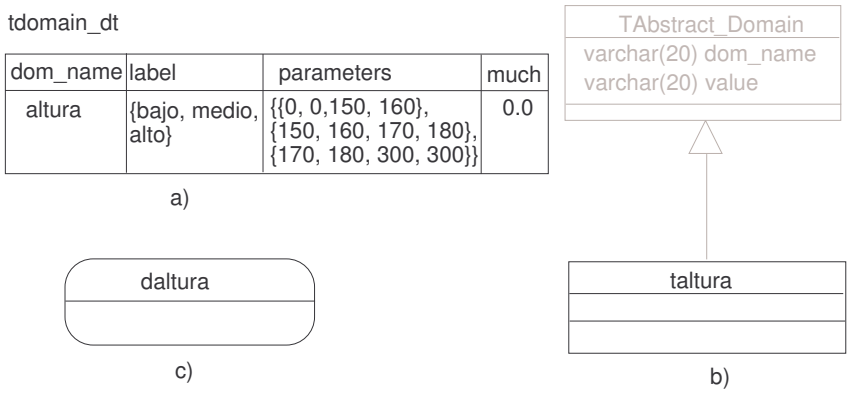


Figura 4.9: Resultado de la función *Create_Domain_DT*.

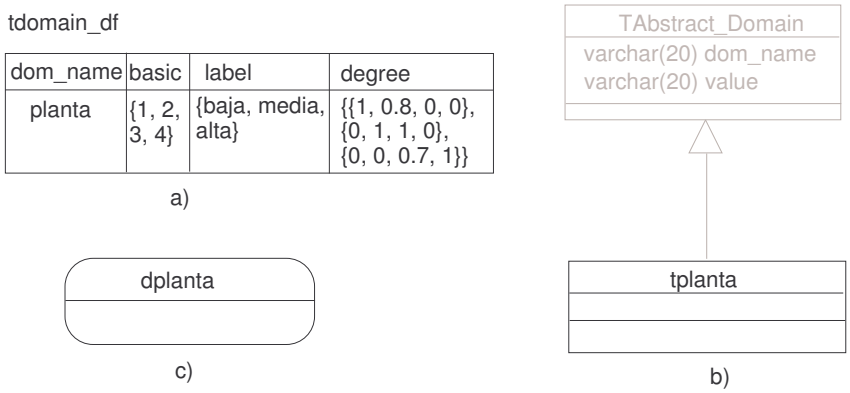


Figura 4.10: Resultado de la función *Create_Domain_DF*.

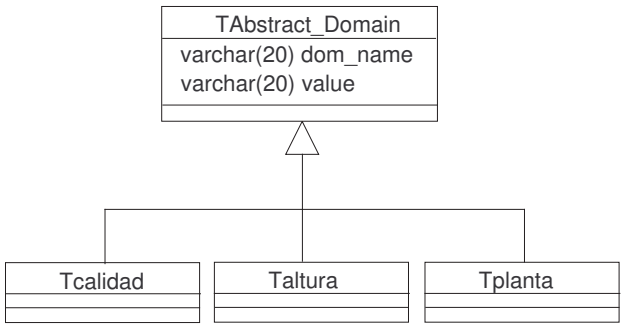


Figura 4.11: Estructura de las *TWFA* para valores difusos.

insertaron, en la jerarquía de la figura 4.5, los meta datos de cada dominio.

A manera de resumen la figura 4.12, muestra el proceso para crear un dominio difuso atómico y las funciones que para cada dominio deben utilizarse.

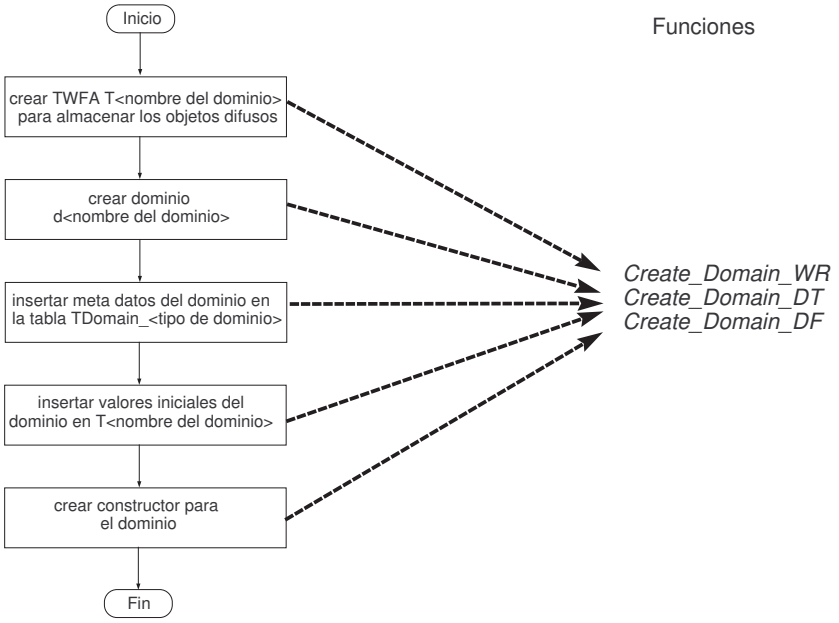


Figura 4.12: Funciones para crear un dominio atómico.

Utilizando las funciones presentadas hasta este punto el usuario puede definir sus propios dominios difusos dentro de la base de datos. Los dominios difusos podrán ser utilizado para definir atributos dentro de una *TWFA*. La forma de utilizar estos dominios en una *TWFA* y cómo crear objetos de estas *TWFA* es tema de otro apartado. Antes, se continuará describiendo las particularidades de la definición de dominios, explicando los dominios conjuntivos.

4.4. Dominios Conjuntivos con referencial sobre valores

Los dominios conjuntivos permiten que los atributos de una *TWFA* definidos sobre ellos acepten como valor un valor difuso en el cual la propiedad que lo caracteriza puede tomar como valor un subconjunto difuso. Los subconjuntos difusos pueden tener un referencial de valores o un referencial de objetos. Los elementos teóricos de estos dominios fueron presentados en el capítulo anterior.

En este apartado se muestra cómo fueron implementados estos dominios. Inicialmente sólo se presentarán los dominios conjuntivos sobre referencial de valores y luego serán retomados estos dominios para analizarlos cuando el referencial es de objetos.

Se hace necesario contar en el modelo con una estructura que permita definir y almacenar subconjuntos difusos de elementos, señalando un grado de pertenencia de cada elemento al subconjunto. Para cumplir este objetivo se define la tabla *TDomain_set (Dominio Conjunto)* (ver Figura 4.13).

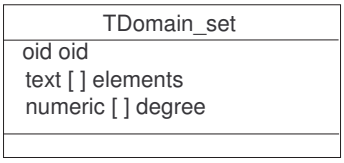


Figura 4.13: Tabla TDomain_set.

La tabla *TDomain_set* es independiente de la tabla *TAbstract_Domain*, presentada anteriormente, pues las características de los dominios conjuntivos son diferentes a la de los dominios atómicos. En el caso de los dominios atómicos se requiere almacenar información sobre los dominios, como pueden ser las funciones de pertenencia o las relaciones de similitud, mientras en el caso de los dominios conjuntivos no existe una información inicial que se requiera almacenar. Por eso en los dominios conjuntivos es suficiente con definir la *TWFA* que dará soporte a los objetos del dominio. Ambas tablas *TDomain_set* y *TAbstract_Domain* forman la jerarquía final que da soporte al trabajo con dominios difusos en el modelo.

Los atributos que caracterizan los objetos de la tabla *TDomain_CF* son:

1. Identificador del objeto, es un atributo que identifica el objeto dentro del servidor de bases de datos. Este atributo es generado internamente por *PostgreSQL*.

2. Un vector con los elementos que forman el subconjunto *(elements).*

3. Vector con los grados de pertenencia de cada elemento al subconjunto, respondiendo al orden de los elementos en el primer vector *(degree).*

Para definir un dominio de este tipo en *pg4DB*, se requiere de un procedimiento que permita definir un dominio conjuntivo. El procedimiento se muestra en la figura 4.14.

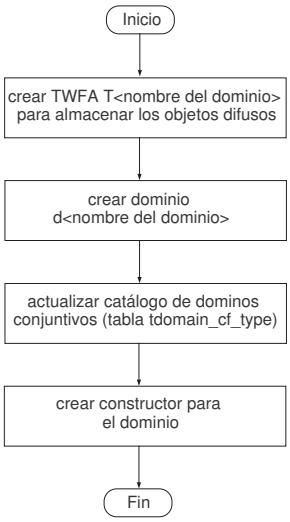


Figura 4.14: Proceso para crear un dominio conjuntivo.

Al igual que en los dominios atómicos el usuario requiere de una interfaz que le permita crear su propio dominio. Para definir este tipo de dominio el modelo incorpora una función llamada *Create_Conjuntive_Extension.*

Función: Create_Conjuntive_Extension	
Create_Conjuntive_Extension (<domain_name>);	
Entradas:	
domain_name	**Tipo:** cadena de caracteres **Definición:** nombre del dominio conjuntivo que se desea crear. Este nombre debe ser diferente a cualquier otro dominio definido dentro del modelo.

Cuadro 4.7: Función *Create_Conjuntive_Extension.*

Al ejecutar esta función, en el modelo se define una estructura que permite al usuario trabajar atributos de este tipo de dominio. En tal sentido se crean los siguientes elementos en el modelo (ver Figura 4.15).

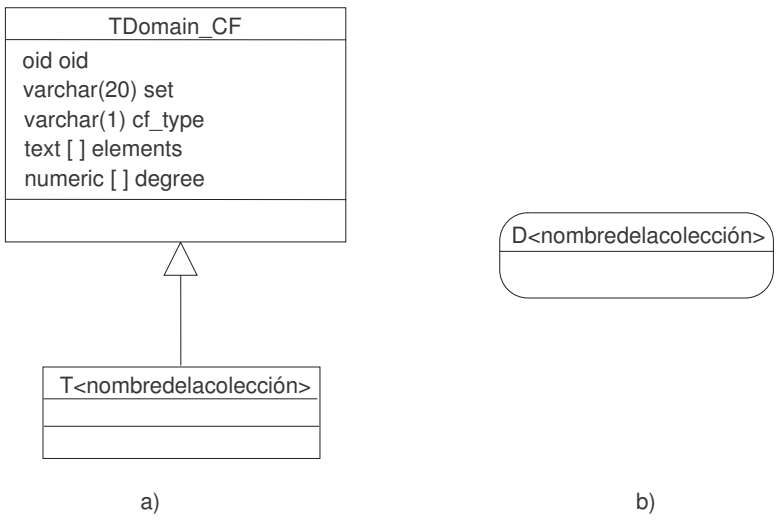


Figura 4.15: Objetos creados por la función *Create_Conjuntive_Extension.*

del dominio>. Este dominio estará disponible para que el usuario lo utilice como tipo para cualquier atributo de una *TWFA* en el modelo.

Si se quiere crear un dominio para un atributo que almacene el conjunto de idiomas que domina un estudiante se debe utilizar la siguiente sentencia *SQL*:

```
SELECT Create_Conjuntive_Extensión ('idiomasset');
```

En este ejemplo se crea un dominio con significado conjuntivo sobre un referencial de valores. Al ejecutarse la sentencia *SQL* se crea una tabla con el nombre de la *Tidiomasset* que hereda de *TDomain_set* y el dominio disponible al usuario para definir sus atributos será *Didiomasset* (ver Figura 4.16).

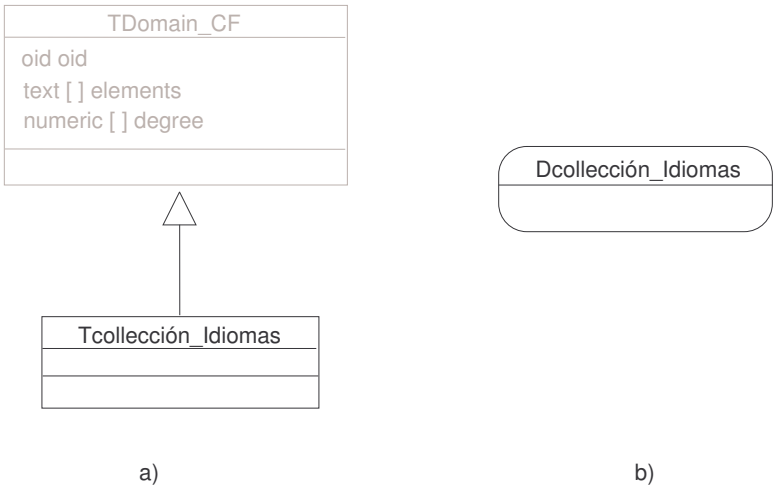


Figura 4.16: Dominio conjuntivo *colección_idiomas*.

A manera de resumen la figura 4.17 muestra el proceso de crear un dominio conjuntivo y la función que debe utilizar el usuario.

Se ha visto de que forma el usuario puede crear y definir sus dominios dentro de la base de datos, ya estos dominios quedan disponibles para ser empleados de la misma forma que un dominio genérico del gestor de bases de datos. Sólo el usuario tendrá que usarlo al momento de definir sus *TWFA*. Los detalles generales de la implementación del trabajo con objetos dentro del sistema son mostrados a continuación.

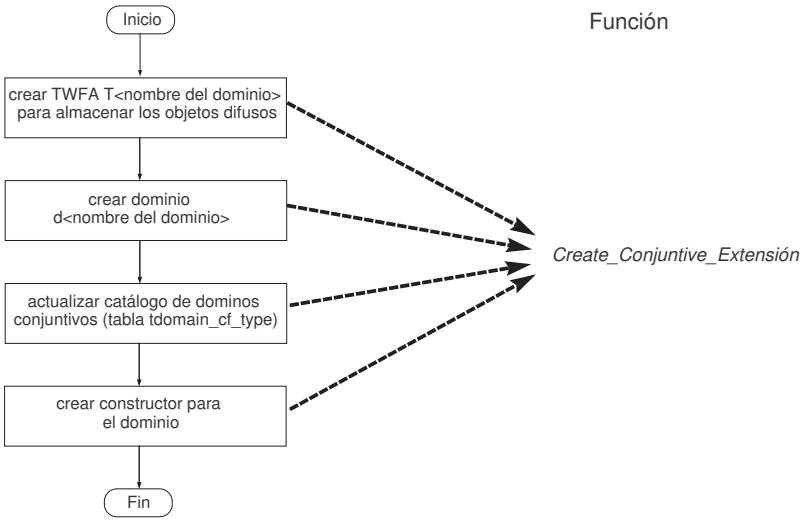


Figura 4.17: Crear un dominio conjuntivo

4.5. Objetos con atributos difusos.

La posibilidad de utilizar objetos con atributos difusos en bases de datos *PostgreSQL* es uno de los principales resultados de esta investigación. Esta característica dota a *PostgreSQL* de la capacidad de almacenar y consultar sobre objetos descritos por atributos difusos. Por lo tanto, con *pg4DB* pueden definirse y manipularse objetos, para lograr esto fue necesario desarrollar una estructura que permita definirlos y manipularlos.

Desde el punto de vista de implementación se creó una tabla superior en la jerarquía descrita hasta este momento, de forma que todos los objetos del modelo hereden de ella. La herencia a esta tabla permite el acceso por medio de un mecanismo único a cada uno de los objetos lo cual dio la posibilidad de implementar funciones únicas capaces de adaptarse a los objetos invocados en sus parámetros. Esto último es necesario por la característica de *PostgreSQL* de no permitir definir métodos y procedimiento asociados a una tabla.

Para cumplir este objetivo se creo la tabla *TFuzzyObject* (ver Figura 4.18). Esta tabla no contiene ningún atributo y sólo se utiliza como tabla abstracta para agrupar los objetos de la base de datos. Como se puede notar en la figura, las tablas *TAbstract_Domain* y *TDomain_set* presentadas en los apartados anteriores hereda directamente de *TFuzzyObject*.

Los objetos que heredan de las tablas *TAbstract_Domain* y *TDomain_set* son considerados como valores difusos dentro del modelo. En la figura 4.19 se muestra la jerarquía que va

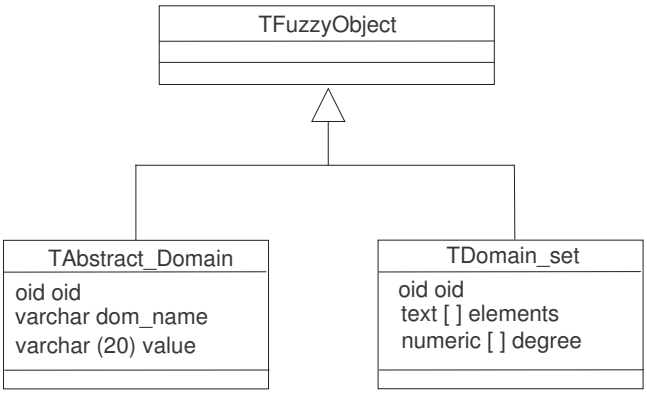


Figura 4.18: Tabla *TFuzzyObject*.

quedando a partir de los ejemplos de definición de dominios presentados en los apartados anteriores.

El usuario de la base de datos en ningún momento trabaja directamente con la tabla *TFuzzyObject* pues el principal objetivo de la misma es crear un espacio de trabajo común para la representación y la manipulación de objetos.

En el modelo es posible trabajar con objetos y con objetos complejos que se almacena en las *TWFAs* definidas por el usuario. La estructura que da soporte a esta posibilidad está formada por la tabla *TFuzzyTable*, una tabla que hereda directamente de *TFuzzyObject* (ver Figura 4.20)

El usuario puede crear y manipular sus objetos por medio de esta tabla. Este nuevo elemento en la jerarquía agrupa las *TWFAs* de objetos por lo que *TFuzzyTable* hereda de la súper tabla *TFuzzyObject*, como se muestra en la figura 4.20.

A partir de esto las *TWFAs* que defina el usuario, tienen que heredar de la tabla que da soporte a la funcionalidad, *TFuzzyTable*. El procedimiento general para crear una *TWFA* se muestra en la figura 4.21.

El diagrama de flujo de la figura 4.21, describe el proceso que se realiza para crear una *TWFA* y permitir la manipulación de la misma por el sistema y por el usuario. Cada uno de los subprocesos se irán detallando en los próximos apartados.

Se comienza con el primer proceso, *crear la* TWFA. El usuario podrá definir las *TWFAs* utilizando el comando CREATE TABLE de SQL, pero tendrá que realizar algunas precisiones. Definirá sus atributos asignándole el dominio a cada uno y especificará una herencia directa de la *TWFA* con la tabla *TFuzzyTable*. Las operaciones o funciones sobre

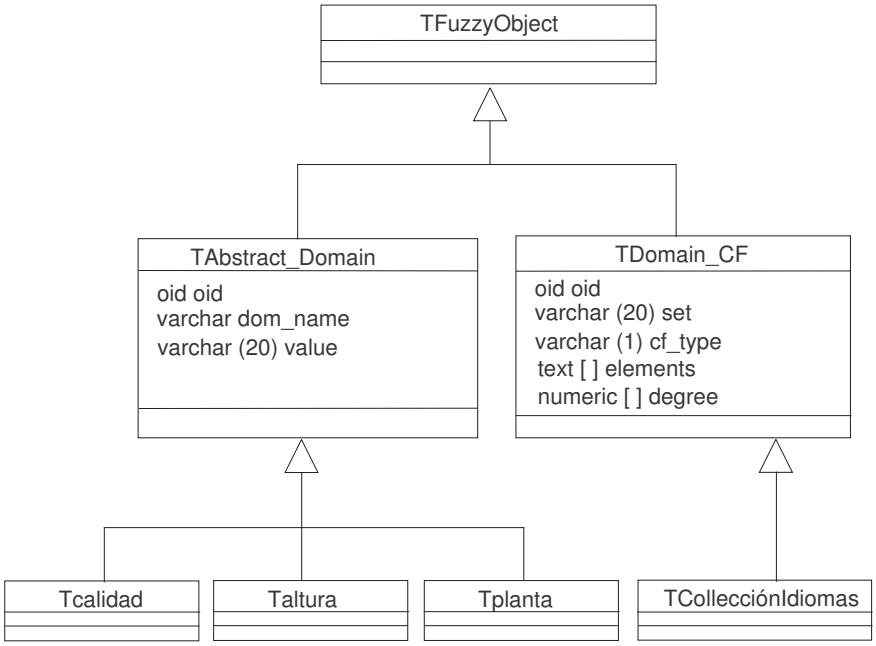


Figura 4.19: Jerarquía en la base de datos al definir los dominios difusos.

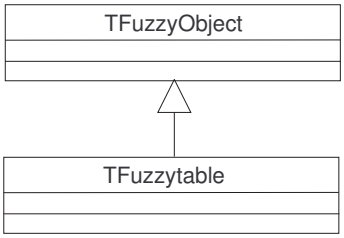


Figura 4.20: Tabla *TFuzzyTable*.

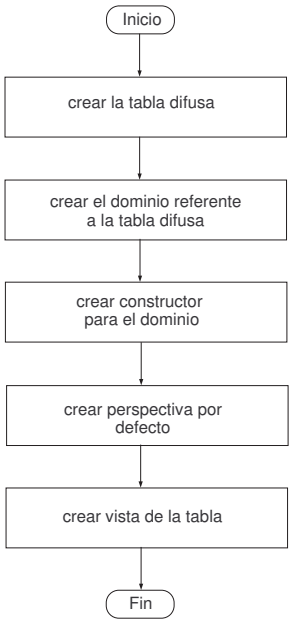


Figura 4.21: Proceso para crear una *TWFA*.

las *TWFAs* creadas, gracias a la jerarquía de tablas creada, podrán ser utilizadas de las misma forma para todas las *TWFAs* de la base de datos.

Si se quiere crear una *TWFA* llamada *Estudiantes* es necesario escribir un código como el que se muestra a continuación.

```
CREATE TABLE TEstudiante (
...
 ) INHERITS(TFuzzyTable) WITH OIDS;
```

Cuando se crea una tabla en *PostgreSQL* automáticamente se crea un *TDA* compuesto, con la misma estructura de la tabla creada. Este *TDA* puede ser utilizado como dominio de un atributo en otra tabla. Pero como se mostró en la tabla (4.1), estos *TDA* compuestos utilizados como dominio, sólo almacenan el valor de un nuevo objeto en la misma tabla en la cual se define el atributo. De esta forma no es posible realizar una referencia a ese objeto en otra tabla, además de que la existencia del objeto está asociada al registro de la tabla con lo cual no se puede manipular de forma individual.

Estas características de los tipos compuestos creados por *PostgreSQL* no es deseada

en la implementación de *pg4DB*. Se requiere utilizar el concepto de referencia para poder manipular y reutilizar los objetos. Por tales razones se creó una función que define un nuevo dominio en la base de datos respondiendo a la *TWFA* creada y dando solución a los problemas que en este sentido presenta *PostgreSQL*. Este dominio podrá ser utilizado como dominio para un atributo en otra *TWFA*, representando una referencia a un objeto del dominio.

Además fue necesario crear un constructor propio para la *TWFA*, ya que *PostgreSQL* no es capaz de definir constructores para los tipos compuestos. Por lo tanto se necesita definir otra función que creé un constructor para la *TWFA*. El dominio creado para la tabla es interpretado por el sistema como un dominio referido a objetos.

El usuario debe llamar explícitamente esta función. Las acciones de crear el dominio y crear el constructor fueron incluidad en una misma función. La función *Create_table_domain* cuya sintaxis es muy sencilla y se refleja en la tabla 4.8 es la encargada de realizar estas acciones.

Función: Create_table_domain	
Create_table_domain (<table_name>);	
Entradas:	
table_name	**Tipo:** cadena de caracteres **Definición:** nombre de una *TWFA* previamente creada en la base de datos.

Cuadro 4.8: Función Create_table_domain.

Al ejecutarse esta función se crea en el sistema los siguientes objetos:

1. El dominio d¡nombre de la *TWFA*¿que podrá ser utilizado directamente como dominio de un atributo en otra tabla y que representa un dominio de objetos.

2. Un constructor para la *TWFA* con el nombre set_<nombre de la *TWFA*> y que tendrá como parámetro todos los atributos definidos para la *TWFA*.

3. Una vista de la *TWFA* con el nombre v_<nombre de la *TWFA*> que muestra los valores de los objetos de la tabla. El objetivo de esta vista y la importancia de ella dentro del modelo será retomado en próximos apartados, luego de explicar cómo se almacenan los objetos en la base de datos.

4. Un registro en la tabla *tdomain_profile* definiendo una perspectiva por defecto para la *TWFA*. El trabajo con perspectivas presentado más adelante.

Para el ejemplo de crear la *TWFA TEstudiante*, si luego se ejecuta la siguiente sentencia:

```
SELECT Create_table_domain(TEstudiante);
```

Se crearán los objetos que se muestran en la figura 4.22 dentro de la base de datos.

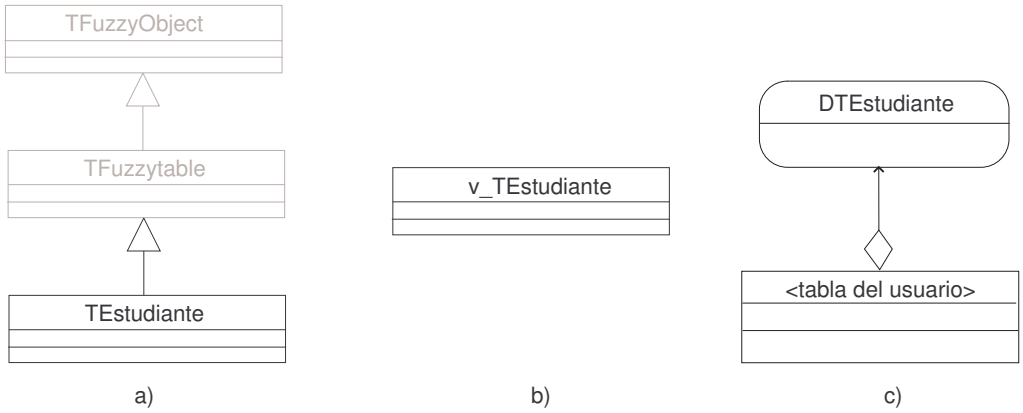


Figura 4.22: Objetos al crear una *TWFA*.

Partiendo de este ejemplo se dispone entonces en el sistema del dominio *dtestudiante* que podrá ser utilizado como un dominio para un atributo, un ejemplo se muestra en el siguiente código.

```
CREATE TABLE tesis (
...
estud dtestudiante,
...
 ) INHERITS(TFuzzyTable) WITH OIDS;
```

En este ejemplo se define la tabla *tesis* que tiene como uno de sus atributos un objeto de tipo *TEstudiante* que representa el estudiante que la está realizando.

Finalmente el proceso de crear una *TWFA* en la bases de datos se resume en la figura 4.23.

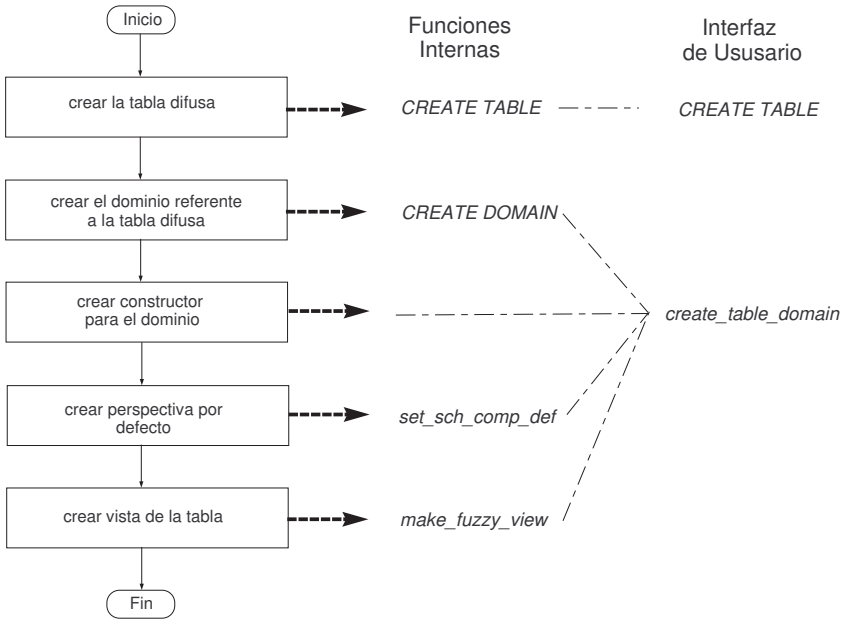


Figura 4.23: Funciones relacionadas con el proceso de crear *TWFA*.

Como puede verse en la figura 4.23, aunque se utilizan varias funciones para la implementación del proceso de crear una *TWFA*, sólo el usuario tendrá que utilizar dos sentencias, CREATE TABLE para crear las *TWFAs* y *Create_table_domain* para definir el dominio relacionado con la *TWFA*.

La jerarquía final que da soporte a los objetos difusos dentro del modelo va quedando de la siguiente forma (ver Figura 4.24). En la figura, se han incluido las tablas que almacenarán los valores difusos creados en los apartados anteriores.

En apartados anterior se mostró como definir dominios difusos sobre un referencial de valores, ahora fue mostrada la forma de crear *TWFA*. En el siguiente apartado se presentarán las extensiones realizadas a los dominios sobre referencial difuso para que pueden ser utilizados con un referencial de objetos.

4.5.1. Dominios con un referencial de objetos.

En el capítulo anterior se describieron los dominios definidos sobre referencial de objetos. Se mostró cómo fueron extendidos dos dominios para soportar objetos: los dominios atómicos con referencial finito y los dominios conjuntivos. Se debe recordar que en el modelo

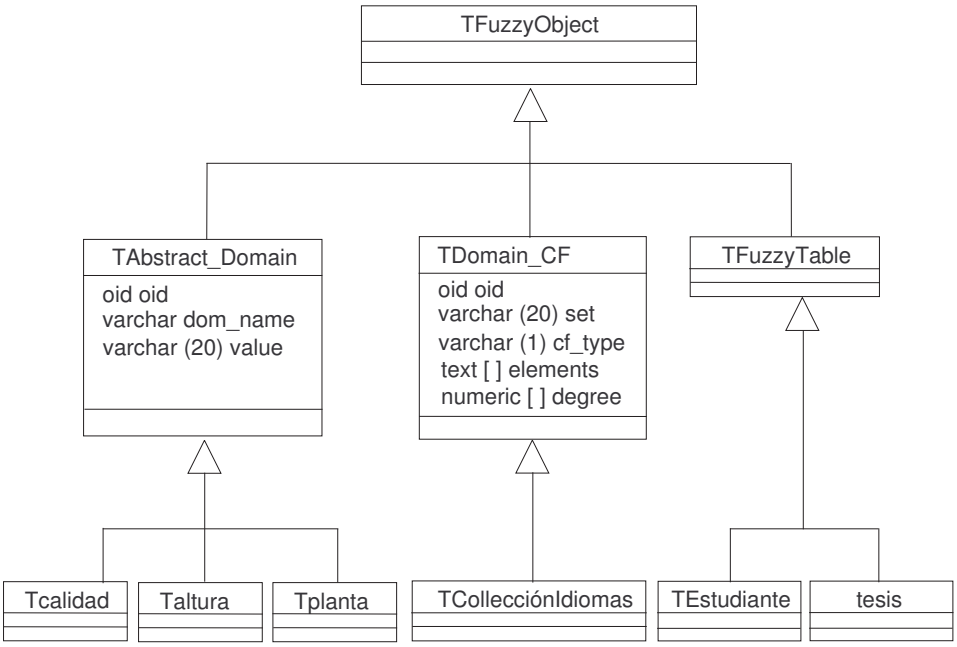


Figura 4.24: Estructura en la BD luego de crear las *TWFAs*.

es posible trabajar con valores difusos que se almacenan en objetos con una sola propiedad y heredan de la tabla *TAbstract_Domain* o de la tabla *TDomain_set* y los objetos y objetos complejos que heredan de la tabla *TFuzzyTable*.

Para dar soporte a los dominios atómicos con referencial finito y los dominios conjuntivos de forma que su referencial este definido sobre objetos se creo la siguiente estructura en la plantilla *pg4DB* (ver Figura 4.25).

Como se muestra en la figura 4.25 se incorpora una nueva tabla a la jerarquía, la tabla *tdomain_objset*. Esta tabla permite definir valores difusos como subconjuntos de objetos, caracterizado por los siguientes atributos:

1. Nombre del tipo de los objetos que constituyen el referencial del dominio *(objname)*.

2. Semántica con la cual será interpretado el conjunto. Existen dos posibles valores: C para una semántica conjuntiva y D para una semántica disyuntiva *(semantics)*.

A continuación se mostrará de qué forma se implementaron las extensiones a los dominios señalados para soportar como referencial objetos, tanto valores difusos como objetos.

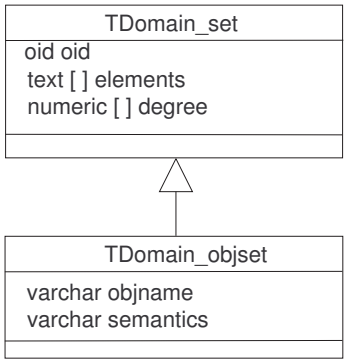


Figura 4.25: Soporte para dominios con referencial sobre objetos.

4.5.1.1. Atómico con referencial finito sobre objetos

El dominio se crea por medio de la función Create_Domain_DF() sobrecargada según la sintaxis mostrada en la tabla 4.9. Al llamar esta función se pasa como segundo parámetro el tipo de objeto sobre el que se define el referencial. Al ejecutar la función una nueva tabla que hereda de la tabla *TDomain_objset* y que tendrá como valor para el atributo *objname* el nombre del tipo de objetos de su referencial y para el atributo *semantics* el valor *D* indicando que la semántica es disyuntiva.

Función: Create_Domain_DF	
Create_Domain_DF (<collection_name>, <table_name>);	
Entradas:	
collection_name	**Tipo:** cadena de caracteres **Definición:** nombre del dominio a ser creado. Este nombre debe ser diferente a cualquier otro dominio definido dentro del modelo.
table_name	**Tipo:** cadena de caracteres **Definición:** nombre válido de una de las *TWFAs* definidas en el modelo.

Cuadro 4.9: Función Create_Domain_DF sobre objetos.

En este caso, el procedimiento para crear este dominio es el mismo que el mostrado en la figura 4.17, pero se utiliza la función *Create_Domain_DF*. Si se quiere tener un dominio de este tipo para un atributo autor que su referencial este definido sobre objetos de la tabla *tinvestigador* (ver Ejemplo 3.4) se debe escribir la siguiente sentencia *SQL*.

```
SELECT Create_Domain_DF ('autor','tinvestigador');
```

Para este caso se definirán en la bases de datos los siguientes objetos (ver Figura 4.26), dando soporte al dominio creado. En este caso se definen como valor por defecto para el atributo *objname tinvestigador* y para el atributo *semantics D*.

La otra semántica que puede tomar este dominio es conjuntiva, la cual se describe en el próximo apartado.

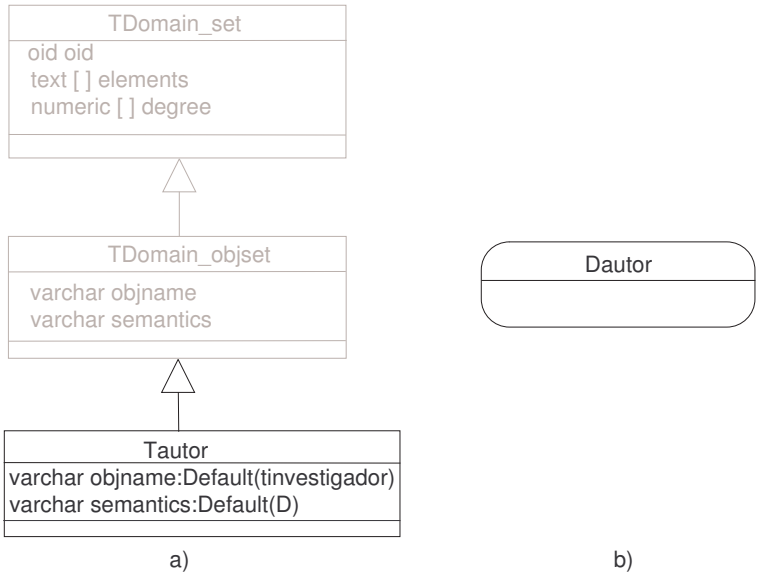


Figura 4.26: Dominio atómico sobre referencial de objetos, ejemplo autor.

4.5.1.2. Conjuntivo sobre un referencial de objetos.

El comportamiento de este tipo de dominio es similar a cuando el dominio conjuntivo está definido sobre un referencial de valores. El procedimiento para crearlo es el mismo (ver Figura 4.17). A diferencia del dominio mostrado en el anterior apartado el valor para el atributo *semantics* será C.

Para crear un dominio conjuntivo con referencial sobre objetos se utiliza la función *Create_Conjuntive_Extension*. Esta función es la misma que la que se utiliza para los dominios conjuntivos con referencial de valores (ver apartado 4.4), pero en este caso se agrega un nuevo parámetro que es el nombre de la tabla que contiene los objetos que formarán el referencial (ver Tabla 4.10).

Función: Create_Conjuntive_Extension	
Create_Conjuntive_Extension (<domain_name>, <type>);	
Entradas:	
domain_name	**Tipo:** cadena de caracteres **Definición:** nombre del dominio conjuntivo a ser creado. Este nombre debe ser diferente a cualquier otro dominio definido dentro del modelo.

type	**Tipo:** cadena de caracteres **Definición:** nombre válido de una de las tablas definidas en el modelo.

Cuadro 4.10: Función Create_Conjuntive_Extension().

La ejecución de esta función tiene como resultado la creación de las estructuras necesarias para la utilización por parte del usuario del nuevo dominio, similar a lo obtenido en la figura 4.26. Pero en este caso la semántica de dominio será conjuntiva.

Si se quiere definir el dominio para el atributo personalidad, ejemplo (3.6), se debe utilizar la función Create_Conjuntive_Extension() de la siguiente forma:

```
SELECT Create_Conjuntive_Extensión
('personalidadset','tpersonalidad');
```

En este código *personalidadset* es el nombre del dominio y *tpersonalidad* de la tabla donde se almacenan los objetos. Los valores por defecto para los atributos *objname* y *semantic* son *Tpersonalidad* y C, respectivamente (ver Figura 4.27). En este ejemplo se toma como objeto un dominio de valores difusos pero si lo que se quiere como referencial es un objeto el procedimiento es el mismo.

4.5.1.3. Definición de atributos en una *TWFA*.

Al presentar los detalles de implementación relacionados con la definición de dominios difusos se mencionó que al crear un dominio difuso automáticamente se crea un dominio en *PostgreSQL* que representa el dominio creado. Recordar que los valores para atributos definidos sobre dominios difusos se almacenan en las tablas creadas para tal efecto y que heredan de *TAbstract_Domain*, *TDomain_set* o *TDomain_objset* según sea el caso. Estas tablas tienen definida una clave en el atributo denominado *oid*. Este atributo tiene características especiales dentro del sistema de bases de datos como identificador de cada objeto. Queda entonces especificar cómo se crea o declaran los atributos en una *TWFA*.

Cada uno de los atributos con dominio difuso que se incluyan en una *TWFA* será una referencia a la *TWFA* de valores difusos que soporta el dominio previamente creado para

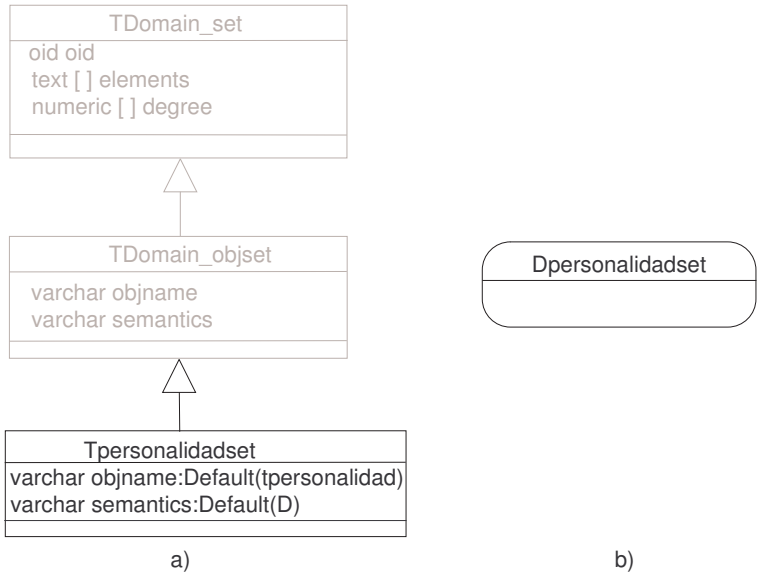


Figura 4.27: Conjuntivo sobre un referencial de objetos, ejemplo personalidadset.

representar los valores de ese atributo. Con esta estrategia se evita la redundancia de la información en las tablas y se optimiza el almacenamiento de la información en las mismas.

Por ejemplo si se tiene definido el atributo difuso *altura*, cuya definición conlleva la creación de la *TWFA* de valores difusos *Taltura* y el dominio *Daltura*. Para definir una *TWFA Estudiantes* que contenga el atributo *estatura* definido sobre el dominio *daltura* se debe escribir el siguiente código:

```
CREATE TABLE TEstudiantes(
nombre varchar (20) CONSTRAINT pk_tstudent PRIMARY KEY,
estatura Daltura REFERENCES Taltura
)INHERITS(TFuzzyTable) WITH OIDS;
```

Como se muestra en el código, es necesario crear una restricción (*constraint*) de clave externa que especifique la relación del atributo difuso con la tabla donde se almacena el valor difuso del atributo. Por otro lado notar que se debe especificar la herencia de la tabla a *TFuzzyTable*.

Utilizando los dominios hasta ahora definidos se define el siguiente ejemplo(4.4).

Ejemplo 4.4. *Se definen dos tablas una para almacenar la información referente a los estudiantes y otra para las tesis que desarrollan. La definición es según el siguiente código.*

```
CREATE TABLE Estudiantes(
nombre varchar (20) CONSTRAINT pk_tstudent PRIMARY KEY,
estatura daltura REFERENCES Taltura,
idioma didiomaset REFERENCE tidiomaset,
personalidad dpersonalidadcollec REFERENCE tpersonalidadcollec )
INHERITS(TFuzzyTable) WITH OIDS;

CREATE TABLE tesis (
titulo varchar,
eval_calidad dcalidad REFERENCE tcalidad,
estud dtestudiante,
 ) INHERITS(TFuzzyTable) WITH OIDS;
```

La figura 4.28 muestra la jerarquía que va quedando luego de definir los dominios atómicos, conjuntivos y las *TWFAs estudiante* y *tesis*.

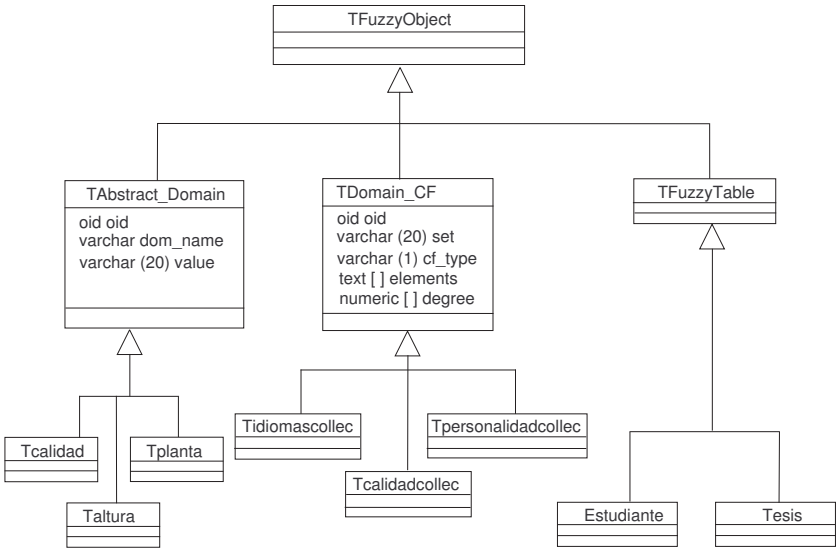


Figura 4.28: Estructura resultante en la base de datos.

De esta forma queda definida una *TWFA*. Como es de suponer para insertar objetos en

la *TWFA* se requiere de funciones especiales que puedan manipular los obejtos y permitan una interfase amigable con el usuario en este sentido y con una total transparencia.

4.6. Insertar valores difusos en un atributo.

Se ha mostrado cómo es posible definir un conjunto de atributos en una *TWFA* para describir una entidad en un problema. La inserción de nuevos objetos en la *TWFA* como valor de un atributo requerirá de especificaciones que mantengan la integridad del modelo y permitan interpretar y manipular correctamente los objetos, dándole al usuario una interfaz cercana al lenguaje propio de la base de datos.

Para insertar objetos como valor de un atributo en una *TWFA* se utiliza el comando *INSERT* de *SQL*, perfectamente soportado en *PostgreSQL*. Con la diferencia que cuando se está en presencia de un atributo definido sobre un dominio difuso la asignación se hará por medio de una expresión. Esta expresión retorna una referencia válida al objeto que representa el valor deseado en la tabla donde está definido el dominio del atributo. Estas expresiones funcionan de la siguiente forma:

- Las expresiones tienen un parámetro que es el valor que se quiere asignar al atributo.
- El sistema busca en la tabla correspondiente al dominio el valor que se quiere insertar, si este valor existe toma el *OID* y lo inserta como valor en la tabla del usuario. Si el valor no existe lo crea insertando un nuevo objeto en la tabla del dominio y retornando el *OID* asignado a ese valor. Este *OID* es el valor que se asigna al atributo.

A manera de resumen, en los atributos difusos se asigna como valor un objeto identificado por su *OID* que hace referencia al verdadero valor del objeto. El valor real estará insertado en la *TWFA* de objetos creada para el dominio.

Por ejemplo si se tiene el dominio *altura*, ejemplo (3.15), en la base de datos existe la *TWFA* de valores difusos *Taltura* que hereda de la tabla *TAbstract_Domain* donde se almacenan los valores de ese dominio (ver Figura 4.24). Luego de trabajar un tiempo con la base de datos se tiene que en la *TWFA taltura* existen los objeto que se muestran en la tabla 4.11.

Si se quiere insertar el valor *MEDIO* como valor para un atributo estatura definido sobre el dominio *Daltura* en una *TWFA* se insertará en realidad el valor 175343. Si por

Cuadro 4.11: Tabla taltura

OID	**dom_name**	**value**
175342	taltura	BAJO
175343	taltura	MEDIO
175344	taltura	ALTO
175432	taltura	172
175540	taltura	164

otro lado se quiere insertar como valor 1.85 entonces en la *TWFA taltura* se insertará un nuevo valor con *value=1.85* y el OID resultante será el valor insertado.

Las expresiones utilizadas para asignar un valor están definidas mediante funciones. Para cada uno de los dominios, soportados en el modelo, se define una función que será utilizada para asignar el valor de un atributo. Las funciones son detalladas en los siguientes apartados.

4.6.1. Insertar valores en atributos definidos sobre dominios atómicos.

Recordar que en la jerarquía está definida la tabla *TDomain* para representar los dominios atómicos con y sin representación semántica asociada. Al definir, dentro de una tabla, un atributo sobre este dominio se necesita de una función que permita insertar una referencia a un valor válido para el atributo.

Los valores de los atributos difusos están almacenados en la *TWFA* de valores difusos correspondiente al dominio creado, (*T<nombredominio>*), mientras que en las *TWFA* de objetos sólo estará la referencia a ese valor por medio del identificador del objeto (*oid*), como se explicó en el apartado anterior.

Para insertar un valor a un atributo definido sobre uno de estos dominios se definió la función *atomic_fuzzy_value* en *PostgreSQL* con la sintaxis reflejada en la tabla 4.14.

Función: atomic_fuzzy_value	
atomic_fuzzy_value (<'attribute_name'>, <'value'>);	
Entradas:	
attribute_name	**Tipo:** cadena de caracteres **Definición:** nombre de un atributo en la *TWFA*.
value	**Tipo:** cadena de caracteres **Definición:** valor valido para el atributo especificado.

Cuadro 4.12: Función *atomic_fuzzy_value*.

Entonces si se quiere insertar un valor para el atributo estatura, ejemplo (3.15), se debe escribir las siguiente sentencia:

```
INSERT INTO Estudiantes VALUES (
'Gloria',
 ...
 atomic_fuzzy_value ('estatura','MEDIO'),
 ...
 );
```

La función *atomic_fuzzy_value* permite insertar un valor difuso como valor en un atributo. Los valores difusos en el modelo están representados por objetos con una sola propiedad y su asociación con un objeto complejo se realiza por medio de una referencia. Por estos motivos la función *atomic_fuzzy_value* retorna la referencia al objeto que contiene el valor.

Como puede notarse en ningún momento es necesario especificar el dominio de los atributos; independientemente del dominio del atributo esta función retornará la referencia al valor insertado.

El nuevo objeto en la *TWFA Estudiante* va tomando los siguientes valores, tabla (4.13).

Cuadro 4.13: Insertar valor de estatura en Estudiante

nombre	estatura	idioma	personalidad
Gloria	175343	...	...

Y la *TWFA taltura* tendrá los valores mostrados en la tabla (4.11).

Anteriormente se mostró que para las *TWFA* se crean constructores que permiten crear nuevos objetos. Luego al explicar el proceso para definir dominios atómicos (ver Figura 4.6) se mostró un proceso que es para crear el constructor del dominio. Partiendo de que en los dominios difusos sus valores pueden ser tratados como objetos con una sola propiedad almacenados en una *TWFA*, se está ante el mismo problema que al crear una *TWFA*. *PostgreSQL* crea por defecto un tipo compuesto para las tablas de los dominios, este tipo

puede ser utilizado como dominio de un atributo en otra tabla y se pueden crear valores utilizando el constructor *ROW*. Sin embargo para *pg4DB* esto no es suficiente, como se explicó anteriormente, si se quisiera utilizar constructores para las tablas de los dominios estos deben responder a las características de los dominios difusos.

La función *atomic_fuzzy_value* puede ser suficiente para insertar un valor a un atributo, pero es poco intuitiva para el usuario, por lo tanto se decidió ocultar esta función al usuario por medio de la definición de constructores para cada uno de los dominios. En este caso los constructores tendrán la siguiente sintaxis (ver Tabla 4.14).

Función: set_<nombre del dominio>	
set_<domain_name> (<'value'>);	
Entradas:	
value	**Tipo:** cadena de caracteres **Definición:** especifica el valor que se desea tome el objeto en su atributo value (ver Figura 4.5). El valor puede ser un número o una etiqueta, pero siempre expresada en forma de cadena de caracteres

Cuadro 4.14: Constructores para los dominios atómicos.

En el caso del dominio *altura*, se definirá el constructor *set_altura()*. El constructor se crea automáticamente al momento de crear un dominio. Si se utiliza el constructor definido para insertar un nuevo *estudiante* se debe escribir entonces el siguiente código.

```
INSERT INTO Estudiantes VALUES (
 ...
set_altura ('MEDIA'),
 ...
 );
```

Como puede verse el uso de constructores para las tablas facilita el trabajo y hace más amigable el código *SQL* al usuario, además de extender *PostgreSQL* para que soporte esta característica.

Además de estos dominios están los conjuntivos, para insertar los valores en dominio conjuntivos se sigue un procedimiento similar, este procedimiento se muestra en el apartado

siguiente.

4.6.2. Insertar valores en atributos definidos sobre dominios conjuntivos.

Los dominios difusos conjuntivos, como se han presentado, pueden estar definidos sobre un referencial de valores o sobre un referencial de objetos. En dependencia del referencial se ha definido funciones para insertar valores en los atributos definidos sobre estos dominios. Se mostrarán en los siguientes apartados, en dependencia del referencial, como insertar valores en los dominios conjuntivos.

4.6.2.1. Referencial sobre valores

Para subconjuntos formados por valores se definió la función *simplex_fuzzy_set ()*. Esta función permite insertar un nuevo valor en la tabla que almacena el dominio y retorna una referencia válida al valor insertado (ver Tabla 4.15).

Función: simplex_fuzzy_set	
simplex_fuzzy_set (<'attribute_name'>, <elements> , <degree>);	
Entradas:	
nombre_ atributo	**Tipo**: cadena de caracteres **Definición**: nombre que tiene el atributo en la *TWFA*.
elements	**Sintaxis: '{**<*element*>, {**,** <*element*> }**}'** **Tipo**: cadena de caracteres **Definición**: el nombre del elemento que formará parte del subconjunto difuso. Este atributo es una lista de todos los elementos.
degrees	**Sintaxis: '{**<*degree*>, {**,** <*degree*>}**}'** **Tipo**: valor numérico real en el rango [0,1] **Definición**: representa el grado de pertenencia del *nombre_elemento* al subconjunto, el grado se asignará en el mismo orden que los elementos.

Cuadro 4.15: Función *simples_fuzzy_set.*

Para insertar un valor al atributo *idiomas* se utiliza una sentencia como esta:

```
INSERT INTO Estudiantes VALUES (
 ...
 simplex_fuzzy_set ('idiomas', '{español, ingles, francés}',
 '{1.0, 0.6, 0.4}'),
 ...
 );
```

En este ejemplo se define el valor para el atributo *idiomas* de un objeto de la tabla *Estudiantes*. El objeto que se está insertando en la *TWFA Estudiante*, uniendo todas las sentencias tiene los siguientes valores, tabla (4.16).

Cuadro 4.16: Insertar valor de idioma en Estudiante

nombre	estatura	idioma	personalidad
Gloria	175343	175355	...

En la tabla *tidiomacollec* se almacena el objeto que da valor al atributo *idioma* del *estudiante*, en el ejemplo se creara el siguiente objeto, tabla (4.17).

Cuadro 4.17: Objeto en la *TWFA* tidiomacollec

oid	elements	degree
175355	{español, inglés, francés}	{1.0, 0.6, 0.4}

Al igual que para los dominios atómicos, *pg4DB* crea automáticamente constructores para las tablas que almacenan los objetos relacionados con el dominio. El constructor para este tipo de dominio tendrá la sintaxis reflejada en la tabla 4.18.

Función: set_<nombre del dominio>	
set_<domain_name> (<elements> , <degrees>);	
Entradas:	
elements	**Sintaxis: '{**<*element*>, {**,** <*element*> **}}'** **Tipo**: cadena de caracteres **Definición**: el nombre del elemento que formará parte del subconjunto difuso. Este atributo es una lista de todos los elementos.

degrees	**Sintaxis: '{**<*degree*>, {**,** <*degree*>}**}'** **Tipo**: valor numérico real en el rango [0,1] **Definición**: representa el grado de pertenencia del *element* al subconjunto, el grado se asignará en el mismo orden que los elementos.

Cuadro 4.18: Constructor para dominio conjunto.

Utilizando el constructor definido para el dominio *idomacollec*, el código de insertar un valor quedaría de la siguiente forma:

```
INSERT INTO Estudiantes VALUES (
 ...
 set_idomacollec ('{español, ingles, francés} ', '{1.0, 0.6, 0.4}'),
 ...
 );
```

Si el dominio esta definido sobre un referencial de objetos entonces se sigue un procedimiento similar como se muestra en el siguiente apartado.

4.6.2.2. Referencial sobre objetos.

Ahora sólo falta insertar valores para aquellos atributos que tomarán como valor un subconjunto de elementos que a su vez son objetos. Los objetos que estarán dentro del subconjunto pueden ser de diferentes tipos, cada uno en dependencia de la tabla de la que hereda. De esa forma los objetos pueden estar definidos sobre tablas que han heredado de los dominios *TDomain*, *TFuzzyTable* y *TDomain_CF*.

Como ahora se está trabajando con objetos, es necesario utilizar su identificador dentro de la base de datos (*oid*) para poder referenciarlos. Al usuario le resulta muy incómodo trabajar con estos identificadores pues sus valores no dan ninguna información reconocible por él. Es conveniente que el usuario pueda hacer referencia a los objetos por un valor conocido por él.

En el caso de objetos que hereden de las tablas *TDomain* la mejor forma de hacer referencia a ellos es por medio de su valor, el atributo *value*. Para los objetos que pertenecen

a *TWFA* y que por consiguiente heredan de la tabla *TFuzzyTable* la mejor referencia es por medio de la clave definida por el propio usuario.

Con el objetivo de soportar todos estos elementos se ha creado la función *complex_fuzzy_set ()*. Esta función permite al usuario entrar los valores en un lenguaje conocido por él y garantiza la inserción de los valores en la tabla que almacena el dominio, al tiempo que retorna una referencia al valor. La sintaxis de la función (ver Tabla 4.19), es muy similar a la función *simplex_fuzzy_set* descrita anteriormente.

Función: complex_fuzzy_set	
complex_fuzzy_set (<'attribute_name'>, <elements> , <degrees>);	
Entradas:	
attribute_name	**Tipo**: cadena de caracteres **Definición**: nombre que tiene el atributo en la *TWFA*.
elements	**Sintaxis: '{**<*element*>, {**,** <*element*> }**}'** **Tipo**: cadena de caracteres **Definición**: el nombre del elemento que formará parte del subconjunto difuso. Este atributo es una lista de todos los elementos.
degrees	**Sintaxis: '{**<*degree*>, {**,** < *degree*>}**}'** **. Tipo**: valor numérico real en el rango [0,1] **Definición**: representa el grado de pertenencia del *element* al subconjunto, el grado se asignará en el mismo orden que los elementos.

Cuadro 4.19: Función *complex_fuzy_set.*

La gran diferencia es que ahora *nombre_elemento* se refiere al identificador de un objeto. Notar que *nombre_elemento* puede ser el valor, cuando se está trabajando con objetos definidos sobre tablas que heredan de *TDomain,* o el valor de la clave en una tabla que herede de *TFuzzyTable.*

Pero puede ser que en el caso de los objetos no siempre la *TWFA* donde se almacenan tiene definida una clave, por lo tanto la función *complex_fuzzy_set* se sobrecargo para que en el valor de los elementos acepte o la llave de la *TWFA* o el valor del OID de los objetos.

Si se quiere insertar un valor a un atributo definido sobre el dominio personalidadcollec,

ver definición del dominio en el ejemplo (3.6), puede escribirse el siguiente código:

```
INSERT INTO Estudiantes VALUES (
 ...
 complex_fuzzy_set('personalidad', '{Honesto, Creativo, Trabajador}',
 '{0.65, 0.5, 0.8}'),
 ...
 );
```

En este caso se está utilizando como referencial un valor difuso, por tanto los elementos que forman parte del subconjunto son los valores asociados al atributo *value*, de la tabla en la cual se almacenan los objetos de este dominio.

Se le dio valor al atributo *personalidad* del *Estudiante* que se esta insertan, por lo tanto el objeto se completa de la siguiente forma, tabla (4.16).

Cuadro 4.20: Insertar valor de personalidad en Estudiante

nombre	estatura	idioma	personalidad
Gloria	175343	175355	175381

En la tabla *tpersonalidadcollec* se almacena el objeto que da valor al atributo *personalidad* del *estudiante*, injertándose el siguiente objeto, tabla (4.21).

Cuadro 4.21: Objeto en la *TWFA* tpersonalidadcollec

oid	set	cf_type	elements	degree
175381	tpersonalidad	O	{Honesto, Creativo, Trabajador }	{0.65, 0.5, 0.8}

Otro ejemplo es el de insertar los investigadores vinculados a un proyecto, ejemplo (3.2.3), se puede escribir el siguiente código.

```
INSERT INTO Estudiantes VALUES (
 ...
 complex_fuzzy_set('participan', '{José, Maria, Isabel '},
 '{0.8, 0.6, 0.6}'),
 ...
 );
```

En el ejemplo cada uno de los participantes es un objeto complejo del tipo *Investigador* con una clave definida sobre el atributo *nombre* del investigador. Si la *TWFA* tiene una clave compuesta se utilizará "para separar ambos valores, el sistema se encargará de manipular la información especificada.

De la misma forma que para el resto de los dominio, para este dominio se definen constructores. En este caso el constructor seguirá la sintaxis mostrada en la tabla 4.22.

Función: set_<domain_name>	
set_<domain_name> (<elements> , <degrees>);	
Entradas:	
elements	**Sintaxis: '{**<*element*>, {**,** <*element*> }**}'** **Tipo**: cadena de caracteres **Definición**: el nombre del elemento que formará parte del subconjunto difuso. Este atributo es una lista de todos los elementos.
degrees	**Sintaxis: '{**<*degree*>, {**,** <*degree*>}**}'** **. Tipo**: valor numérico real en el rango [0,1] **Definición**: representa el grado de pertenencia del *element* al subconjunto, el grado se asignará en el mismo orden que los elementos.

Cuadro 4.22: Constructor para dominio conjunto de objetos.

De forma similar los elementos pueden ser: valores de la clave de la *TWFA* o el OID de los objetos. Los ejemplos mostrados en este apartado, utilizando los constructores quedarían de la siguiente forma:

```
INSERT INTO Estudiantes VALUES (
 ...
 set_investigarodores('{José, Maria, Isabel '}, '{0.8, 0.6, 0.6}'),
 set_personalidadcollec('{Honesto, Creativo, Trabajador '}, '{0.65, 0.5, 0.8}'),
 ..
 );
```

Los elementos estructurales básicos del modelo han sido presentados. Y la parte del modelo de manipulación de los datos, particularmente el de insertar datos también. Una vez que los datos han sido insertados al usuario le interesa consultarlos. Una de las principales consultas que se realiza en una base de datos difusa orientada a objetos es la de comparar objetos entre si para obtener el parecido. Los elementos de cómo fue implementada la comparación entre objetos en *pg4DB* se muestran en el próximo apartado.

4.7. Comparación entre objetos.

En el capítulo anterior se realizó todo el recorrido teórico sobre el modelo propuesto en esta investigación describiendo los métodos utilizados para comparar objetos. Se expuso cómo comparar valores difusos en los diferentes dominios existentes y luego se describió como realizar la comparación entre los objetos.

Desde el punto de vista de la implementación estas operaciones son por funciones que podran ser utilizada con todos los objetos creados en la base de datos. Fue necesario recurrir a una función por la imposibilidad que tiene *PostgreSQL* de asociar métodos y/o procedimientos a las tablas, unas de las principales carencias de *PostgreSQL* para ser considerado objeto-relacional puro.

Gracias a la jerarquía de tablas que se ha construido se ha podido definir una función que permite realizar el cálculo del parecido entre dos objetos cualesquiera (ver Tabla 4.23).

Función: get_resemblance	
get_resemblance (<object|value >, < object|value >);	
Entradas:	
object	**Tipo**: *oid* **Definición**: identificador del primer objeto.
value	**Tipo**: cadena de caracteres **Definición**: representa un valor, ya sea un número o una etiqueta lingüística.

Cuadro 4.23: Función *get_resemblance.*

Esta función se traduce en tres funciones:

- get_resemblance(oid,oid): permite comparar dos objetos, los objetos tienen que ser

del mismo tipo.

- get_resemblance(oid,valor) y get_resemblance(valor,oid):permite comparar un objeto con un valor. El valor deber ser una valor asociado al referencia definido para el tipo del objeto. Esta función no está disponible para todos los objetos, solo en aquellos donde es posible la comparación entre un valor y un objeto.

Esta función implementa los algoritmos de comparación descritos en el capítulo anterior para cada tipo de objeto. La función quedo implementada de la siguiente forma:

1. Llamada de la función pasándole los identificadores de los objetos a comparar.
2. Comprobar la igualdad en el tipo de los objetos pasados como parámetros.
3. En dependencia del tipo de los objetos ejecutar la función para el cálculo de semejanza.
4. Para *dominios atómicos*, se determina la forma en que está representado el dominio. Cada uno de los dominios tiene definido su propio algoritmo para el cálculo.
5. En los *dominios conjuntivos* se determina si el dominio tiene un referencial sobre valores o sobre objetos y para cada caso se define un algoritmo de cálculo.
6. Si la comparación es entre objetos complejos, tendrá al menos un atributo que es un objeto. En este caso se parte de recorrer todos los atributos del objeto e ir llamando la función para el cálculo de semejanza entre los mismos. Y luego se realiza la agregación entre las semejanzas calculadas.

Con el objetivo de hacer más intuitivo el uso de la función *get_resemblance()* se define la función con el nombre *FEQ* que acepta los mismo parámetros y funciona de forma similar los mismos parámetro. Un ejemplo de del uso de la función es:

```
SELECT FEQ(e1.oid,e2.oid) FROM testudiante as e1, testudiante as e2;
```

En este caso se está comparando los estudiantes de la *TWFA* entre si. El *oid* es el atributo que identifica a los objetos. Notar que no se puede utilizar una notación convencional orientada objeto como pudiera ser $e1.FEQ(e2)$ ya que aquí se esta trabajando con funciones definidas en el servidor de bases de datos que deben ser referenciadas por su nombre.

Además de esta función de comparación que puede ser utilizada indistintamente con cualquier tipo de objeto, se definen otras que operan sobre determinados objetos, a estas funciones se le dio el nombre genérico de operadores difusos. En los siguientes apartados se describen la implementación de los principales operadores difusos incorporados al *pg4DB*, partiendo del tipo de objeto en los que se definen.

4.7.1. Operadores difusos en dominios atómicos

En los dominios atómicos se definen operadores difusos para los dominios que tiene un referencial continuo, respondiendo a los operadores definidos en las tablas (3.3 y 3.4).

Para cada operador se definió una función que implementa la comparación, la sintaxis de la función es similar a la sintaxis de la función *FEQ* (ver Tabla 4.24).

Función: OPERATOR	
OPERATOR (<object|value >, < object|value >);	
Entradas:	
object	**Tipo**: *oid* **Definición**: identificador de objeto.
value	**Tipo**: cadena de caracteres **Definición**: representa un valor, ya sea un número o una etiqueta lingüística.

Cuadro 4.24: Operadores difusos.

OPERADOR: es uno de los operadores difusos definidos en las tablas (3.3 y 3.4). Si se quiere utilizar unos de estos operadores, por ejemplo el operador *FLT*, para comparar la estatura de dos estudiantes se debe ejecutar el siguiente código.

```
SELECT FLT(e1.estatura,e2.estatura) FROM testudiante as e1,
testudiante as e2;
```

Para el resto de los dominios atómicos solo es posible utilizar el operador *FEQ*.

4.7.2. Operadores difusos en dominios conjuntivos

Para este tipo de dominio, teniendo en cuenta el operador *FEQ*, se define un total de cinco operadores, ver epígrafe (3.2.3) y (3.2.3.1). Estos cinco operadores son para:

- Comparar subconjuntos difusos utilizando el grado de semejanza generalizado. Operador *FEQ*.
- Comparar subconjuntos difusos utilizando el grado de consistencia. Operador *COG*.
- Comparar subconjuntos difusos utilizando una media entre el grado de semejanza generalizado y el grado de consistencia. Operador *AVGC*.
- Comparar subconjuntos difusos utilizando el grado de inclusión guiada por semejanza. Operador *GIS*.
- Comparar subconjuntos difusos utilizando el grado de inclusión teniendo en cuenta la semejanza. Operador *GIN*.

La sintaxis de estos operadores se corresponde con la mostrada en la tabla 4.24, definida en el apartado anterior. Un ejemplo del uso de estos operadores es el siguiente.

```
SELECT GIS(e1.idioma,e2.idioma) FROM testudiante as e1, testudiante
as e2;
```

En este caso se están comprando los idiomas que dominan los estudiantes por medio de la inclusión guiada por semejanza.

4.7.3. Operadores difusos en objetos

Para los objetos también se definieron varias formas de compararlos. Como se vio en el apartado (3.2.1) del capítulo anterior, la comparación entre objetos se ve afectada por un coeficiente, el número de *orness*. Por lo tanto se definieron funciones que permiten trabajar con diferentes número de *orness* y se sobrecargo la función *FEQ* de forma tal que acepte la especificación de este *coeficiente*. Por lo tanto fueron implementadas las siguientes funciones.

- Utilizando número de *orness* = 1.0 expresando la existencia. Función *EXI (objeto, objeto)*.

- Utilizando número de *orness* = 0.0 expresando lo universal, todos. Función *UNI (objeto, objeto)*.
- Especificar el número de *orness*. Función *FEQ(objeto,objeto,orness)*

Ejemplo del uso de estas funciones son:

```
SELECT FEQ(e1.oid,e2.oid,0.4) FROM testudiante as e1, testudiante as
e2;

SELECT EXI(e1.oid,e2.oid) FROM testudiante as e1, testudiante as e2;
```

En el ejemplo se especifica 0.4 como numero de orness y luego se utiliza la función *EXI* para comparar los estudiantes.

4.7.4. Constantes difusas

En el capitulo anterior se presentaron las constantes difusas disponibles en el modelo propuesto, epígrafe (3.4). Estas constantes fueron implementadas por medio de funciones que retornan valores que pueden ser interpretados por la base de datos. Se debe recordar que las constantes solo pueden ser utilizadas como parámetro en las consultas de los datos, vinculadas con las funciones de comparación entre objetos vistas en lo apartados anteriores.

Por las características de estas constantes solo pueden ser utilizadas para comparar objetos definidos sobre dominios atómicos con referencial continuo. La sintaxis de cada una de estas funciones y su forma de utilización se muestra a continuación.

Constante, distribución de posibilidad trapezoidal.

Esta constante permite definir una función trapezoidal que podrá ser comparada con un objeto definido sobre dominios atómicos con referencial continuo. La constante se define por medio de la función *f* (ver Tabla 4.25).

Función: f	
f (a, b, c, d);	
Entradas:	
a, b, c, d	**Tipo**: a, b, c y d son todos de tipo numérico **Definición**: parámetros de la función trapezoidal (ver Figura 3.9)

Cuadro 4.25: Constante difusa, distribución de posibilidad trapezoidal.

Esta función puede utilizarse directamente como parámetro de las función de comparación *FEQ* o de los operadores difusos definidos para los dominios atómicos con referencial continuo. Un ejemplo del uso de la constante es el siguiente código.

Ejemplo del uso de estas funciones son:

```
SELECT FEQ(e1.estatura,f(165,170,180,185)) FROM testudiante as e1;

SELECT FGEQ(e1.estatura,f(165,170,180,185)) FROM testudiante as e1;
```

En el primer ejemplo se retorna el parecido entre la estatura de los estudiantes y la constante definida y en el segundo el valor en que la edad de los estudiantes es posiblemente mayor o igual que la constante definida.

Constante, intervalo numérico.

Esta constante representa un intervalo de valores y se representa mediante la función *fi*, a la cual se le pasan dos parámetros que definen el intervalo (ver Tabla 4.26).

Función: fi	
fi (n, m);	
Entradas:	
n, m	**Tipo**: n y m son de tipo numérico y deben cumplir la condición que n $\leq$ m. **Definición**: parámetros del intervalo numérico

Cuadro 4.26: Constante difusa, intervalo numérico.

El uso de la función es similar a la anterior y se utiliza en las mismas circunstancias, un ejemplo es:

```
SELECT FEQ(e1.estatura,fi(170,185)) FROM testudiante as e1;

SELECT FGEQ(e1.estatura,fi(170,185)) FROM testudiante as e1;
```

En este caso la comparación de la edad se realiza contra el intervalo (170,185)

Constante, valor difuso aproximado.

Como su nombre indica esta constante representa un valor difusos aproximado. Para definirlo se utiliza la función *f*, esta función ha sido sobrecargada para que acepte dos parámetro, un parámetro representando el centro del valor difusos y un segundo parámetro que representa la holgura del valor. La función que permite definir una constante de este tipo queda definida según la sintaxis mostrada en la tabla 4.27.

Función: f	
f (c, h);	
Entradas:	
c	**Tipo**: valor numérico. **Definición**: que representa el centro del valor difusos aproximado.
h	**Tipo**: valor numérico. **Definición**: que representa la holgura del valor difusos aproximado.

Cuadro 4.27: Constante difusa, valor difuso aproximado.

Si se quiere utilizar este tipo de constante en una consulta se puede escribir un código como este:

```
SELECT FEQ(e1.estatura,f(180,5)) FROM testudiante as e1;

SELECT FGEQ(e1.estatura,f(180,5)) FROM testudiante as e1;
```

Ahora la comparación se realiza entre la estatura de los estudiantes y el valor difusos aproximado definido por la función.

Comparar valor con una constante difusa.

Los tres tipos de constantes difusas implementadas en *pg4DB* puedes ser utilizada en la comparación con un número. Estos números pueden ser el valor de un atributo preciso

en una tabla. Para permitir esto fue necesario definir una función que pudiera comparar un valor numérico con una constante difusa, la función se presenta en la tabla 4.28.

Función: IS	
IS (value, constant);	
Entradas:	
value	**Tipo**: valor numérico. **Definición**: un valor nuemrico cualquiera.
constant	**Tipo**: expresión. **Definición**: una expresión que representa una de las constantes difusas definidas en el modelo

Cuadro 4.28: Función para comparar un valor numérico con una constante difusa.

Si se tiene el atributo números de incidentes (*no_incid*), definido sobre un tipo de dato *float*, en una tabla *asegurado*, y quiere comprarse con una distribución de posibilidades trapezoidal se debe escribir el siguiente código.

```
SELECT IS(no_incid, f(3,6,10,13)) FROM asegurado as e1;
```

Las constantes difusas expuestas en este apartado permiten mayor libertad y posibilidades a los usuario de consultar su base de datos difusas para obtener información que considere relevante. En la comparación entre objetos también se definen libertades que le permiten al usuario adaptar la comparación entre los objetos a sus requerimiento, esta libertad se garantiza por medio de la utilización de perspectivas de comparación.

4.7.5. Perspectivas de comparación

Cuando se trata de objetos, la comparación entre dos objetos está condicionada por perspectivas de comparación, epígrafe (3.2.7). Las perspectivas de comparación permiten definir qué peso tendrán los atributos del objeto en la comparación, que operadores se utilizarán para cada atributo en la comparación y el número de *orness* necesario para utilizar el operador de Vila [VCMP94a] en la comparación. El usuario podrá definir cuantas perspectivas desee para su objeto complejo y luego especificará en sus consultas con cual de ellas desea se realice la comparación.

Para la definición de las perspectivas de comparación el usuario dispone de la función *comparison_profile()* (ver Tabla 4.29).

Función: comparison_profile	
comparison_profile (<'table_name'>, <id_schema>,<attributes>, <weights>, <operators/profiles>), <orness>;	
Entradas:	
table_name	**Tipo**: cadena de caracteres **Definición**: nombre válido de una tabla definida en el sistema y que hereda de *TFuzzyObject*.
id_schema	**Tipo**: cadena de caracteres **Definición**: nombre que identifica un esquema.
attributes	**Sintaxis: '{**<*attribute*>, {**,** <*attribute*>}**}'** **Tipo**: cadena de caracteres **Definición**: lista formada por los nombres válidos de los atributos de la tabla especificada en *table_name*.
weights	**Sintaxis: '{**<*weight*>, {**,** < *weight*>}**}'** **Tipo**: valor numérico real en el rango [0,1] **Definición**: define el peso que tendrá el atributo dentro de la tabla. El atributo se forma por un vector y los pesos se asignarán en el mismo orden que fueron introducidos los nombres de los atributos.
operators/profiles	**Sintaxis: '{**<*operators/profiles*>, {**,** <*operators/profiles*>}**}'** **Tipo**: cadena de caracteres que representa un operador valido para el atributo o una perspectiva de comparación **Definición**: define el operador difuso o la perspectiva de comparación que se utilizará al comparar el atributo dentro del objeto. El parámetro atributo se forma por un vector y los operadores se asignarán en el mismo orden que fueron introducidos los nombres de los atributos.

orness	**Tipo**: número real en el rango [0,1] **Definición**: define el número de orness necesario para utilizar el operador de agregación de Vila en la comparación de objetos.

Cuadro 4.29: Función *comparison_profile*.

El uso de esta función (ver Tabla 4.29) definirá una perspectiva de comparación para la *TWFA* especificada que será almacenada en la tabla *tdomain_profile* (ver Figura 4.29).

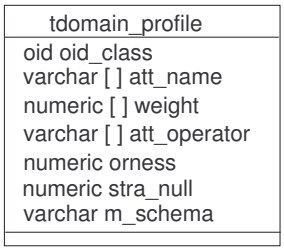


Figura 4.29: Estructura para almacenar las perspectivas de comparación.

Se debe señalar que al definirse una nueva *TWFA* se crea una perspectiva por defecto en la cual todos los atributos tiene como peso 1 y el operador que se utilizará para la comparación es FEQ.

Utilizando el ejemplo (4.4), se definirán dos perspectivas para la tabla *Estudiante* utilizando la función descrita.

```
SELECT fuzzy_schema.comparison_profile('Estudiantes','deporte',
'{nombre, estatura, personalidad}', '{0,1,0.7}',
'{FEQ,FGT,GIN}',0.2);

SELECT fuzzy_schema.comparison_profile('Estudiantes','lider',
'{nombre, estatura, personalidad}', '{0,0.3,0.9}',
'{FEQ,FGT,COG}',0.2);
```

En el ejemplo se definieron dos perspectivas para los objetos del tipo *estudiante*, una perspectiva con el nombre *deporte* y otra *líder*. Ahora se definirá una perspectiva para el

tipo *tesis*. El tipo tesis tiene uno de sus atributos definido sobre un dominio de objetos complejos. La perspectiva puede definirse como muestra el siguiente código.

```
SELECT fuzzy_schema.comparison_profile('tesis','persp1', '{titulo,
eval_calidad, estud}', '{0,1,1}', '{FEQ,FGT,FEQ}',0.2);

SELECT fuzzy_schema.comparison_profile('tesis','persp2', '{titulo,
eval_calidad, estud}', '{0,1,1}', '{FEQ,FGT,deporte}', 0.2);
```

En el primer código se define la perspectiva *persp1* donde se utilizará la función *FEQ* para comparar los atributos *estud* de dos objetos. En la perspectiva *persp2* se define que para comparar los estudiantes se utilice la perspectiva *deporte* previamente definida para el tipo *testudiante*. En esta última perspectiva se evidencia la recursividad de este concepto.

De esta forma el uso de perspectiva en el modelo tiene características recursivas que permiten adaptar el proceso de comparación entre objetos a las necesidades y requerimientos del usuario.

Se mostró cómo definir una perspectiva para una *TWFA*, sobre una tabla se pueden definirse cualquier cantidad de perspectivas por lo que es evidente la necesidad de tener una función que permita especificar la perspectiva a utilizar en la comparación entre dos objetos.

Esta función es la propia *FEQ*, vista en apartados anteriores, pero en este caso se sobrecarga para aceptar un tercer atributo que es el nombre de la perspectiva de comparación que se quiere utilizar. Por lo tanto la función queda según la sintaxis reflejada en la tabla 4.30.

Función: FEQ	
FEQ (<object>, < object>), < profile >);	
Entradas:	
object	**Tipo**: *TFuzzyObject.oid %TYPE* **Definición**: identificador de un objeto.

profile	**Tipo**: cadena de caracteres **Definición**: representa una perspectiva definida previamente para la tabla a la que pertenecen los objetos involucrados en la comparación.

Cuadro 4.30: Función *FEQ* con perspectiva.

Ahora, en este código que se presenta, la comparación de los estudiantes está condicionada por la perspectiva deporte, previamente definida.

```
SELECT FEQ(e1.oid,e2.oid,'deporte') FROM testudiante as e1,
testudiante as e2;
```

Al momento de usar la función *FEQ* sin especificar la perspectiva, siempre el sistema usará una perspectiva, pero en ese caso la perspectiva definida por defecto.

Como se presento en el capítulo anterior, en el caso de dominios conjuntivos sobre un referencial de objetos también es posible definir perspectivas de comparación (ver Epígrafe 3.2.3 y Epígrafe 3.2.8). Lo elementos de la implementación de esta perspectivas se presentan en el siguiente apartado.

4.7.6. Perspectiva de comparación para dominios conjuntivos.

En el epígrafe 3.2.8 se mostraron los detalles de las perspectivas de comparación que se definen para dominios conjuntivos con referencial sobre objetos. Estas perspectivas tiene el principal objetivo de definir, entre otros elementos, con que perspectivas deben comparase los objetos del referencial. Para definir una perspectiva en un dominio conjuntivo se creo la función *domainconj_profile* respondiendo a la sintaxis que se muestra en la tabla 4.31.

Función: comparison_profile	
domainconj_profile (*<'domain_name'>*, *<profile_name>*, *<oper_conj>*,*<profile>*, *<method_comp>*	
Entradas:	
profile_name	**Tipo**: cadena de caracteres **Definición**: nombre para la perspectiva creada.

domain_name	**Tipo**: cadena de caracteres **Definición**: nombre válido de un dominio conjuntivo definido sobre un referencial de objetos.
oper_conj	**Tipo**: cadena de caracteres **Definición**: El operador que se utilizará para comparar los subconjuntos. Estos operadores puede ser FEQ (ver Epígrafe 3.2.3), GIS (ver Ecuación 3.17) o GIN (ver Ecuación 3.21)
profile	**Tipo**: cadena de caracteres **Definición**: La perspectiva de comparación que se utilizará para comparar los objetos que forman el referencial.
method_comp	**Tipo**: valor numérico real en el rango [0,1] **Definición**: Este parámetro solo se define cuando el operador es FEQ. Sus posibles valores son: FEQ, COG, AVG (ver Epígrafe 3.2.3)

Cuadro 4.31: Función *domainconj_profile*.

Suponiendo que en el ejemplo mostrado en el apartado anterior, se tiene una tabla que refiere los datos de un equipo deportivo. Entre los datos del equipo estén los integrantes del equipo. Los integrantes del equipo toma como valor un conjunto donde cada integrante es un estudiante y tiene un grado de pertenencia al equipo. Se quiere definir una perspectiva para el dominio *dintegrantes*, que se utilizará para comparar dos equipos. La definición de la perspectiva se realiza de la siguiente forma:

```
SELECT
fuzzy_schema.domainconj\_profile('dintegrantes','integdeporte',
'FEQ', 'deporte', 'FEQ');
```

En este caso la perspectiva *integdeporte* utilizará para comparar dos valores del dominio el operador *FEQ*, la perspectiva deporte del objeto *Estudiante* y el método de comparación será el definido por defecto.

Si se quiere definir una perspectiva para la *TWFA* equipo, puede utilizarse entonces la siguiente sentencia:

```
SELECT fuzzy_schema.comparison_profile('equipo','persequipo',
'{nombre, jefe, integrantes}', '{0,1,1}',
'{FEQ,gerente,integdeporte}', 0.2);
```

En este caso se refleja totalmente la recursividad del trabajo con las perspectivas. Para un objeto de la *TWFA equipo* se definió una perspectiva que tiene en cuenta para el atributo *jefe* una otra perspectiva. Para el atributo *integrantes* se utiliza una perspectiva dentro de la cual también se define con que perspectiva se compararán los objetos.

Hasta este punto han sido presentados los principales elementos de la implementación de los dominios difusos y los objetos. En el siguiente apartado se muestras las extensiones realizadas al lenguaje de definición de datos (*DDL*) de forma tal que permita definir atributos inferidos en la base de datos.

4.8. Atributos inferidos

Los atributos inferidos son una de las extensiones propuestas en esta investigación para las bases de datos difusas orientadas a objetos. Este atributo como su nombre lo indica, es una de las características de un objeto pero tiene la particularidad que su valor no es asignado de forma explícita por el usuario, sino que es inferido a partir de un sistema de reglas difusas.

PostgreSQL no da soporte directamente para definir atributos inferidos, por lo que fue necesario crear un estructura que permitiera estas extensiones en las bases de datos.

El primer paso es definir un atributo inferido y almacenarlo en la base de datos. Partiendo de esto se requiere en la base de datos una estructura en la que puedan almacenarse los meta datos de un atributo inferido. Esa estructura se definió por medio de la tabla *tfuzzyrule* (ver Figura 4.30).

Como puede verse en la figura 4.30, en la definición de un atributo inferido intervienen varios parámetros, una breve descripción de ellos se muestra en la siguiente lista.

1. fromtable:Especifica el nombre de la tabla sobre la que se define el atributo inferido. Tipo de dato cadena de caracteres.

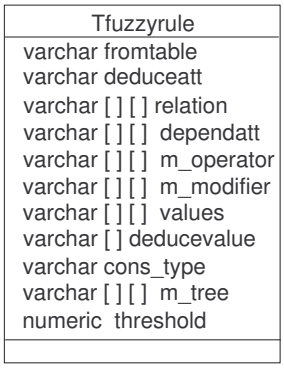


Figura 4.30: Estructura para almacenar la definición de los atributos inferidos.

2. deduceatt: Nombre del atributo inferido, se corresponde con el nombre de un atributo de la tabla almacenada en *fromtable*. Tipo de dato cadena de caracteres.

3. relation]: Se almacenan los operadores lógicos entre las preposiciones de una regla. A cada regla le corresponde una de las dimensiones del vector multidimensional. Tipo de dato vector multidimensional de cadenas.

4. dependatt: Almacena el nombre de los atributos que participan de cada preposición. Para cada regla se define un vector de atributos que se almacena en una dimensión del vector multidimensional. Tipo de dato vector multidimensional de cadenas.

5. m_operator: Operadores que se utilizan en las preposiciones. Es de tipo vector multidimensional de cadenas y se utiliza similar a *dependatt*.

6. m_modifier: Si se incluye un modificador en la preposición, por ejemplo NOT, se almacenaría en este atributo de la tabla. Tiene las mismas características que *dependatt*.

7. values: Con este atributo se completa las preposiciones, en este caso seria el valor que tomara la preposición. El tipo de datos es el mismo que el de *dependatt* y la forma de almacenar la información es igual.

8. deducevalue: Almacena el valor que tomará el consecuente en cada regla, en este caso el tipo es un vector y cada elemento del vector representa el valor del consecuente para la regla.

9. cons_type: Este atributo almacena el nombre del dominio sobre el que se define el consecuente del sistema de reglas asociado al atributo inferido.

10. m_tree: Este atributo almacena una representación en forma de árbol binario para evaluar las reglas. El tipo de dato es un vector y cada elemento del vector es define el orden en que se evaluará la regla.

11. threshold: Es un numero real en el rango [0,1] que define el umbral a partir del cual se disparan las reglas. Este umbral es único para todas las reglas del sistema.

La tabla *tfuzzyrule* esta definida en el esquema *fuzzy_schema* de la plantilla *pg4DB* y es de uso interno para el sistema por lo cual el usuario no trabaja directamente con ella por lo que no tiene que conocer su estructura para poder usar atributos inferido. En tal sentido el usuario dispone de una función que le permite definir sus atributos inferidos *new_att_derive()* (ver Tabla 4.32).

Función: Función new_att_derive	
Función new_att_derive (<fromtable>,<deduceatt>,'{$fuzzyrules_i${, $fuzzyrules_i$}}', <threshold>, <consec_domain>);	
Entradas:	
fromtable	**Tipo:** cadena de caracteres.**Definición:** especifica el nombre de una *TWFA* previamente creada por el usuario en el modelo y en la cual se definirá un atributo inferido.
deduceatt	**Tipo:** Tipo: cadena de caracteres **Definición:** nombre de un atributo de la *TWFA* especificada en fromtable el cual será definido como atributo inferido. Este atributo debe estar definido sobre un dominio difuso atómico trapezoidal.
fuzzy_rules$_i$	**Tipo:**: vector de cadenas de caracteres **Definición:** Especifica las reglas difusas que deben ser evaluadas para obtener el valor del atributo *deduceatt* previamente definido. Las reglas se definen de la forma: $IF < att_k > OPERADOR < valor > \{AND\|OR < att_{k+1} > OPERADOR < valor >\} THEN < deduceatt > IS < valor >$

	Donde: attk:es un atributo definido sobre un dominio difuso en la *TWFA* fromtable valor:posible valor que puede tomar ese atributo OPERADOR: formado por uno de los operadores difusos definidos por pg4DB para atributos difusos{FEQ,FGT,NFGT, FGEQ, NFGEQ, FLT, NFLT, FLEQ, NFLEQ,MGT,NMGT, MLT, NMLT,AVGC,COG,GIN}, dependiendo su uso con el tipo de dominio difuso del atributo y un modificador {VERY, SOMEWHAT, EXTREMELY y SLIGHTLY}
threshold	**Tipo:** Tipo: número real en el rango [0,1] **Definición:** valor a partir del cual se dispararan las reglas, el valor es común para todas las reglas.
consec_domain	**Tipo:** Tipo: cadena de caracteres **Definición:** nombre del dominio que se definirá para el consecuente del sistema de reglas.

Cuadro 4.32: Función *Función new_att_derive.*

Esta función es utilizada por el usuario una vez que ha definido todos los dominios difusos y sus *TWFA*. Al definir un atributo inferido por medio de esta función, se crearán en la base de datos los siguientes objetos.

1. Un registro en la tabla *tfuzzyrule* con la meta información del atributo inferido.

2. Una función disparador asociada a la tabla del usuario sobre la cual se definió el atributo inferido.

3. Un función asociada al disparador encargada de inferir el valor para el atributo.

Por medio de este ejemplo puede entenderse mejor cómo se definen los atributos inferidos y cual es el proceso que ocurre en la base de datos.

Se quiere definir el siguiente sistema de reglas para un atributo denominado *m_clasif* en la tabla *testudiante*. El atributo depende de otros dos atributos de la misma tabla *m_ind* y *m_reco*. El sistema de reglas es el siguiente.

```
R1: IF m_ind FLEQ REGULAR THEN m_clasif IS BAJA,
R2: IF m_ind FGT BUENO AND (m_reco IS BUENA OR m_reco IS MUY_BUENA)
THEN m_clasif IS MUY_ALTA,
R3: IF m_ind FGT BUENO AND m_reco IS REGULAR THEN m_clasif IS ALTA,
R4: IF m_ind FGT BUENO AND m_reco IS MALA THEN m_clasif IS MEDIA
```

El umbral que se define es 0.0 y el nombre del dominio del consecuente es *didoneidad*. Para definir el atributo inferido *m_clasif*, mostrado anteriormente se debe escribir la sentencia:

```
SELECT fuzzy_schema.new_att_derive(
'testudiante',
'm_clasif',
'{
m_ind FLEQ REGULAR THEN m_clasif IS BAJA,
m_ind FGT BUENO AND (m_reco IS BUENA OR m_reco IS MUY_BUENA)
THEN m_clasif IS MUY_ALTA,
m_ind FGT BUENO AND m_reco IS REGULAR THEN m_clasif IS ALTA,
m_ind FGT BUENO AND m_reco IS MALA THEN m_clasif IS MEDIA }',
0.0,
'didoneidad');
```

En la base de datos se define el atributo inferido insertando el siguiente registro en la tabla *tfuzzyrule*.

Como puede verse la principal característica de *PostgreSQL* empleada para almacenar la definición de los atributos inferidos es la posibilidad de definir atributos en las tablas como vectores multidimensionales de valores. Esta característica tiene a la vez una desventaja, la imposibilidad en *PostgreSQL* de crear vectores con longitud variable por lo tanto fue necesario completar los vectores con el valor *N*. En el caso del atributo m_tree representa la estructura de evaluación del cada regla en dependencia de los operadores lógicos y los paréntesis utilizados en su definición. Con esta estructura se garantiza la precedencia de las proposiciones en las reglas.

Toda la información que se introduce en la tabla *tfuzzyrule* es definida por el usuario, excepto el atributo *m_tree* que es determinado por el propio sistema luego de analizar las reglas. Sin embargo el usuario puede introducirla de forma muy intuitiva.

Cuadro 4.33: Definición de un Atributo Inferido.

Atributo	**Valor**
fromtable	testudiante
deduceatt	m_clasif
relation	{{N,N}, {and,or}, {and,N}, {and,N}}
dependatt	{{m_ind,N,N}, {m_ind,m_reco,m_reco}, {m_ind,m_reco,N}, {m_ind,m_reco,N}}
m_operator	{{FLEQ,N,N}, {FGT,IS,IS}, {FGT,IS,N}, {FGT,IS,N}}
m_modifier	{{N,N,N}, {N,N,N}, {N,N,N}, {N,N,N}}
values	{{REGULAR,N,N}, {BUENO,BUENA,MUY_BUENA}, {BUENO,REGULAR,N}, {BUENO,MALA,N}}
deducevalue	{BAJA,MUY_ALTA,ALTA,MEDIA}
cons_type	didoneidad
m_tree	{{!,!,1}, {—;1,—;@;2,—;@;3}, {!,—;1,—;2}, {!,—;1,—;2}}
threshold	0.0

La funcionalidad del atributo inferido en este ejemplo se completa con la definición del disparador *value4m_clasif* asociado a la tabla *testudiante* y que se ejecuta al insertar o actualizar cualquier objeto de la tabla. Además se crea la función *inf_testudiante_m_clasif()* que es la encargada de inferir el valor para el atributo *m_clasif*.

El sistema permite crear cuantos atributos inferidos se quieran para una *TWFA*, pero se debe tener en cuenta que se dispararan en el mismo orden en que fueron definidos, de esa forma el usuario tiene la libertad de utilizar el valor de un atributo inferido en las proposiciones de las reglas definidas para otro atributo inferido.

Hasta aquí se han presentado todas las características definidas en el modelo a nivel de atributos. Sólo falta describir, desde el punto de vista de implementación, el nivel de definición que en el modelo propuesto en esta investigación se basa en el uso de los tipos difusos.

4.9. Tipos Difusos

La implementación de tipos difusos en *pg4DB* sigue las mismas consideraciones de la propuesta orientada a objetos de Marín y otros. Se utilizó la característica de *PostgreSQL* de permitir la herencia entre tablas con lo cual fue posible crear una estructura que permite representar tipos difusos.

En el modelo los tipos difusos son considerados como objeto por tales razones ellos heredan directamente de la tabla *tfuzzytable*, a partir de esto se crea una jerarquía de tablas para representar cada uno de los niveles del tipo difuso (ver Figura 4.31).

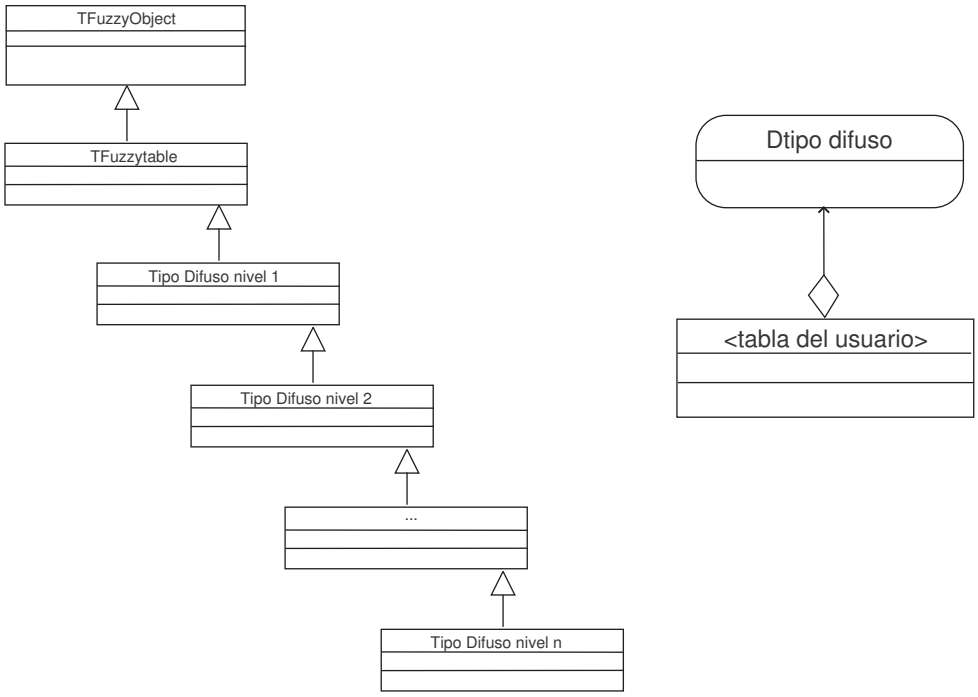


Figura 4.31: Representación de un tipo difuso en *pg4DB*.

Al ser considerados los tipos difusos como objetos, tendrán un comportamiento similar a las *TWFA* presentadas anteriormente, y al igual que estas últimas, podrán usarse los tipos difusos como dominio para un atributo en una *TWFA*. Por lo tanto al definirse en la base de datos la estructura para soportar el tipo difuso, también se crea un dominio que hace referencia a ese tipo difuso (ver Figura 4.31).

Si se aplica esta estructura al ejemplo *persona asegurada* () quedaría en la base de datos una estructura como la mostrada en la figura 4.32.

Como se muestra en la figura 4.31 los tipos difusos son tratados como *TWFA*. Es deseado que toda esta estructura sea transparente para el usuario. A partir de esto, se definen un conjunto de funciones para la creación y manipulación de los tipos. El usuario podrá utilizar estas funciones de forma muy intuitiva cuando necesite incorporar el uso de tipos difusos en la solución del problema que pretende modelar.

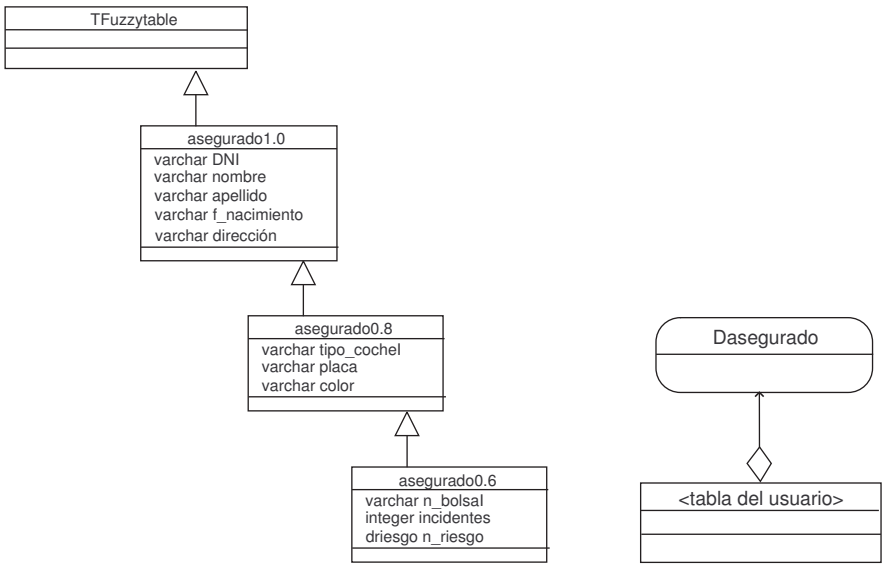


Figura 4.32: Tipo difuso *persona asegurada*.

El procedimiento para crear un tipo difuso es el siguiente (ver Figura 4.33.

En los siguientes apartados se mostrarán estos procesos y como se pueden utilizar y consultar los tipos difusos.

4.9.1. Creación de un Tipo Difuso.

La primera función que se necesita, debe permitir al usuario crear sus propios tipos difusos de una forma intuitiva. Con esta función se garantiza una transparencia total del usuario con la estructura física que se usa para representar el tipo difuso. La sintaxis de esta función queda definida como se muestra en la tabla 4.34.

Función: Create_fuzzy_type	
Create_fuzzy_type (<type_name>, <attributes>);	
Entradas:	
type_name	**Tipo**: cadena de caracteres **Definición**: nombre valido para una tabla en PostgreSQL, el nombre tiene que ser distinto a cualquiera de las otras tablas definidas en el modelo.

attributes	**Sintaxis: '{**<*attribute*>, {**,** <*attribute*>}**}'** **Tipo:** cadenas de caracteres. **Definición**: Define los atributos para el tipo difuso. Cada elemento *attribute* debe estar formado por tres valores: el nivel al que pertenece el atributo, el nombre del atributo y el tipo del atributo junto con las restricciones para el mismo.

Cuadro 4.34: Función *Create_fuzzy_type.*

Para definir el tipo difuso *persona_asegurada* se escribirá el siguiente código.

```
SELECT create_fuzzy_type ('Persona_asegurada',
 '\{
1.0 nombre varchar (20),
1.0 apellido varchar (20),
1.0 DNI varchar (11) CONSTRAINT pk_persona PRIMARY KEY,
1.0 direccion varchar (50),
0.8 tipo_coche varchar (20),
0.8 placa varchar  (7),
0.8 color varchar (10),
0.6 bolsa_seg varchar (20),
0.6 incidentes integer,
0.6 riesgo REFERENCES triesgo
 \}');
```

La función se encargará de definir toda la estructura que soporte el tipo difuso que incluye (ver Figura 4.32):

1. Dominio *dpersona_asegurada* que puede ser utilizado para definir un atributo sobre ese tipo difuso.

2. Crea los constructores para el tipo difuso, los constructores que crea serán presentados en el próximo apartado.

3. Define perspectivas de comparación por defecto.

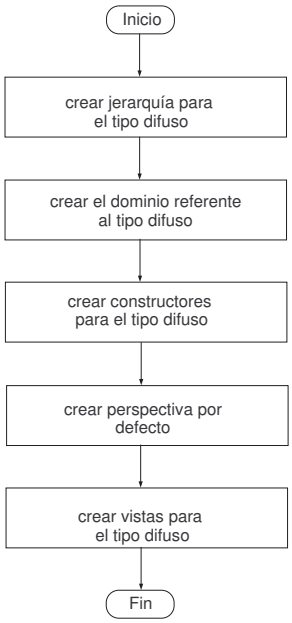


Figura 4.33: Proceso para crear tipos difusos.

4. Crea vistas del tipo difuso y de cada uno de sus niveles.

4.9.2. Creación de un objeto de un tipo difuso.

Para crear un objeto se debe insertar un registro en la estructura del tipo difuso. El comando INSERT de PostgreSQL no está diseñado para soportar la actualización de los tipos difusos, por tal razón es necesario definir funciones que permitan trabajar con los tipos difusos definidos. Para esto fue implementada la función *New_object_fuzzy_type* que permite al usuario crear un objeto a partir de la especificación del nivel que se desea incorporar al objeto (ver Tabla 4.35).

Función: New_object_fuzzy_type	
New_object_fuzzy_type (<type_name>, <level>,<list of values/expression>)	
Entradas:	
type_name	**Tipo**: cadena de caracteres **Definición**: especifica el nombre de un tipo difuso previamente definido por el usuario en el modelo.

level	**Tipo**: numérico real en el rango [0,1] **Definición**: nivel al que se quiere incorporar el nuevo objeto. El nivel especificado debe coincidir con los niveles declarados para el tipo difuso.
list of values/expression	**Tipo**: cadena de caracteres **Definición**: es una lista separada por comas de los valores y expresiones que se desean insertar en el objeto.

Cuadro 4.35: Función *New_object_fuzzy_type.*

Esta función es la base para los constructores definidos para el tipo difuso. Se define un constructor para el tipo difuso y cuantos constructores como niveles existan en el tipo difuso. La sintaxis del primero de estos constructores se muestra en la tabla 4.36.

Función: set_<fuzzy_type_name>	
set_<fuzzy_type_name> (<value/expression>)	
Entradas:	
value |expression	**Definición**: valor que se le asignará al atributo del objeto a insertar. Este parámetro es una lista de todos los valores/expresiones que se utilizarán para insertar el objeto, respondiendo al tipo de los atributos.

Cuadro 4.36: Constructor para un tipo difuso.

Como se puede notar con el uso del constructor del tipo difuso no se necesita precisar el nombre del tipo ni el nivel donde se quiere insertar el objeto, el sistema sabe por la cantidad de atributos a los que se le de valor en que nivel se encuentra el objeto.

Los otros constructores que se definen tiene un comportamiento similar a los constructores definidos para las *TWFA*, de esa forma para el ejemplo del tipo difuso *persona_asegurada* se definen los siguientes constructores.

- persona_asegurada(varchar): constructor que se ajusta a la sintaxis de a tabla 4.36. En dependencia de los valores introducido decidirá en que nivel crea el objeto.
- persona_asegurada(varchar, varchar, varchar, varchar): este constructor crea un ob-

jeto del nivel 1.0.

- persona_asegurada(varchar, varchar, varchar, varchar, varchar, varchar, varchar): este constructor crea un objeto del nivel 0.8.

- persona_asegurada(varchar, varchar, varchar, varchar, varchar, varchar, varchar, varchar, integer, oid): este constructor crea un objeto del nivel 0.6.

El usuario puede utilizar el constructor que prefiera para crear un objeto del tipo difuso. Por ejemplo para crear un objeto *persona_asegurada* en el nivel 0.6, con la función *New_object_fuzzy_type* y con los constructores se escriben las siguientes sentencias.

```
SELECT New_object_fuzzy_type (
 'Persona_asegurada',
0.6, ''Alejandra', 'García', 'x-1234567-d', 'calleA 234', 'Focus',
'1234', 'rojo', '66783-t', 2,
 set_riesgo ('medio')');

SELECT set_persona_asegurada ( ''Alejandra', 'García',
'x-1234567-d', 'calleA 234', 'Focus', '1234', 'rojo', '66783-t', 2,
 set_riesgo ('medio')');

SELECT set_persona_asegurada ( 'Alejandra', 'García', 'x-1234567-d',
'calleA 234', 'Focus', '1234', 'rojo', '66783-t', 2,
 set_riesgo ('medio'));
```

En el primer ejemplo se utiliza la función creada, en el segundo se utiliza el constructor que puede utilizarse para cualquier nivel y en la tercera sentencia se utiliza el constructor definido específicamente para el nivel 0.6.

Con estas funciones el sistema creará un nuevo objeto del tipo difuso especificado que será almacenado en la estructura definida para este fin (ver Tabla 4.37). Se puede notar la utilización del atributo *riego* definido sobre un dominio difuso y la función *set_riesgo* para manejar valores imprecisos de forma que se pueda incluir un nuevo valor en el atributo.

Cuadro 4.37: Objeto del tipo difuso Persona_Asegurada

nombre	apellido	DNI	dirección	tipo_coche	placa	color	bolsa_seg	incid.	riesgo
Alejandra	García	x-1234567-d	calleA 234	Focus	1234	rojo	66783-t	2	62613

4.9.3. Consulta de objetos de un tipo difuso.

Al comparar objetos definidos sobre tipos difusos podrá utilizarse la función FEQ previamente definida para objetos. Se debe recordar que un objeto sobre un tipo difuso es tratado dentro del modelo también como un objeto complejo.

Sin embargo, existen posibilidades al comparar tipos difusos que están relacionadas con los operadores para tipos difusos presentados en el epígrafe (3.5.2). En tal sentido fueron definidas tres funciones:

Función PES Esta función es para la estrategia pesimista donde se asume que la semejanza entre dos atributos si uno de ellos es desconocido (no incorporado en el objeto) es igual a 0,0.

Función OPT Para la estrategia optimista donde el parecido entre dos atributos del cual se desconoce el valor de uno de ellos es 1,0.

Función FEQ Es la misma función FEQ definida para los objetos pero que por defecto descartan en la comparación los atributos pertenecientes al menor nivel del tipo difusos y se realiza la comparación teniendo en cuenta sólo los atributos coincidentes.

Estas tres funciones están sobrecargadas para poder definir el número de orness en la comparación entre objetos. Existe una cuarta función FEQ con cuatro parámetros que permite definir que valor desea el usuario se consideré como parecido entre dos atributos del cual se desconoce el valor de uno de ellos. Con todas estas funciones el usuario dispone de herramientas que permiten realizar consultas sobre la base de datos con total libertad.

Al igual que se mostró con los objetos la posibilidad de definir perspectivas de comparación, en los tipos difusos también se pueden definir esas perspectivas. Se utiliza la misma función que para los objetos (ver Tabla 4.29) pero en este caso se utiliza como nombre de la tabla el propio nombre del tipo difuso. Luego el sistema se encarga de definir las perspectivas para jerarquía del tipo difuso.

4.9.4. Acceso a un atributo del tipo difuso

Como se mencionó en los apartados anteriores al crearse un tipo difusos se define en la base de datos un dominio para el tipo difuso que puede ser utilizado como dominio de un atributo en una tabla difusa o en otro tipo difuso. A partir de esto se hace necesario acceder a los valores del tipo difusos para consultarlo.

Si se desea acceder a un objeto de un tipo difuso se puede hacer directamente utilizando el OID que identifica el objeto. Mientras para acceder a un atributo del tipo difuso se deberá utilizar una función en específica en dependencia del atributo.

Las funciones para acceder a los atributos difusos son de la forma *¡nombre del atributo¿(oid)*, donde oid es el identificador del objeto definido sobre le tipo difuso al que se quiere acceder.

Como ejemplo se utilizará el tipo difuso *persona_asegurada*, como dominio de un atributo en la tabla *factura*.

```
CREATE TABLE factura( ... asegurado dpersona_asegurada ...
 )
```

Para comparar el atributo asegurado de dos factura se escribe el siguiente código.

```
SELECT FEQ(f1.asegurado,f2.asegurado) FROM factura as f1, factura as
f2;
```

Si lo que se quiere es comparar el riesgo de la persona asegurada en las facturas entonces se escribe la siguiente sentencia SQL.

```
SELECT FEQ(get_riesto(f1.asegurado),get_riesgo(f2.asegurado)) FROM
factura as f1, factura as f2;
```

De esta forma se accede a los objetos definidos sobre tipos difusos.

Un resumen de las funciones disponibles, luego de la implementación del modelo teórico, pueden verse en el apéndice (E), de la misma forma se resumen las principales tablas que dan soporte a la implementación.

4.10. Cliente Web para el modelo propuesto.

En lo presentado hasta aquí se muestra que todo el trabajo que se realiza con el modelo es por medio de sentencias *SQL* ejecutadas a nivel del servidor de base de datos. Estas sentencias *SQL* pueden ser utilizadas directamente desde un lenguaje de programación o por medio de alguno de los cliente existentes para *PostgreSQL* como son pgAdmin III, PostgreSQL Maestro, pgAcces, EMS PostgreSQL manager 3, phpPgAdmin y entre otros.

Sin embrago las herramientas que permiten interactuar con *PostgreSQL* no presentan ninguna interfaz para trabajar con los elementos difusos del modelo propuesto e implementado en esta investigación. Como las características difusas incorporadas son extensión añadidas al *PostgreSQL*, no son tenidas en cuentas por los clientes existentes, pudiendo solamente acceder a ellas por medio de sentencias *SQL*. Esta situación tiene el inconveniente de no brindar ninguna información visual que apoye al usuario con lo cual se requiere que el usuario tenga un dominio total de la sintaxis de cada función y del significado de sus parámetros.

Por lo tanto la inexistencia de una interfaz visual que permita al usuario modelar un problema que comprenda información imprecisa utilizando el modelo propuesto en esta investigación dificulta su empleo. Se hace necesario entonces implementar un cliente para *PostgreSQL* que incorpore los elementos difusos incorporados a al base de datos por medio de la presente investigación.

El primer elemento a resolver es qué tipo de aplicación se debe desarrollar. En la actualidad existen claramente definidas dos tipos de aplicaciones informática las aplicaciones desktop y las aplicaciones web. Cada uno de este tipo de aplicaciones tiene sus propias características, sus ventajas y sus desventajas, de forma general en la tabla (4.38) se muestran algunos de estos elementos.

Partiendo de que se requiere una aplicación para interactuar con *PostgreSQL* que de forma natural tiene una arquitectura cliente/servidor y teniendo en cuenta algunos de los elementos reflejados en la tabla (4.38) se decidió desarrollar una aplicación Web.

4.10.1. Aplicaciones Web

Las aplicaciones Web son sistemas informáticos que permiten a los usuarios acceder a un servidor a través de una red y ejecutar una lógica de negocio a través de un navegador (Browser). La estructura de una aplicación Web es como una aplicación de tres capas: el navegador Web es la primera capa, el cual se encarga de realizar las peticiones, la capa

Cuadro 4.38: Aplicaciones Web vs Desktop

	Desktop	**Web**
SO	Generalmente son desarrolladas para un sistema operativo en especifico.	Son independientes del sistema operativo.
Arquitectura	La aplicación radica en el ordenador y funciona de forma local.	Cliente/Servidor. La aplicación radica en un servidor y se ejecuta en el cliente.
Actualización	La aplicación debe actualizarse en cada cliente, varios ordenadores.	La aplicación solo de debe actualizar en el servidor, un solo ordenador.
Usabilidad	El usuario debe familiarizarse desde cero con el sistema desarrollado.	Se aprovecha la experiencia de los usuarios en el uso de aplicaciones Web.
Instalación	Requiere de un proceso de instalación en cada ordenador.	Solo es necesario instalar la aplicación en el servidor.
Acceso	Solo se accede a la aplicación desde el ordenador donde está instalada.	Puede acceder a la aplicación desde cualquier ordenador conectado a la red.
Robustes	Suelen ser más robustas al poderse utilizar todas las potencialidades del SO.	Suelen ser aplicaciones menos robustas.
Red	No requieren conexión a red.	Requiere conexión a red.

intermedia es un motor usando alguna tecnología Web dinámica (un lenguaje de programación para la Web), y la tercera capa es una base de datos. El navegador Web manda peticiones a la capa media, las cuales son entregadas a través de consultas y actualizaciones a la base de datos generando una interfaz para el usuario.

En el desarrollo de una aplicaciones Web pueden presentarse problemas como: dificultades de mantenibilidad (el código HTML está ligado con el código dinámico de la página por lo que la interacción entre programadores y diseñadores es muy compleja) y problemas de control de flujo (las páginas pueden ser ejecutadas fuera del orden previsto y se deben tener en cuenta muchas validaciones en cada página). Para dar solución a este problema se utilizan patrones de arquitectura de software.

El patrón de arquitectura de software más utilizado para desarrollar aplicaciones Web es el Modelo Vista Controlador (MVC). MVC es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

- Modelo: Es donde se representa la información y se almacenan los datos con los cuales opera el sistema. En el modelo se garantiza la integridad de los datos.
- Vista: Se encarga de permitir al usuario una interfaz que le permita interactuar con el sistema.
- Controlador: Controla las acciones del usuario mediante eventos que pueden provocar cambios en el modelo o consultas al modelo que serán mostradas en la vista.

El flujo de trabajo aproximado de esté patrón de diseño es el siguiente:

1. El usuario mediante la interfaz provoca alguna acción que dispara un evento.
2. El controlador recibe una notificación sombre la acción realizada por el usuario y gestiona el evento.
3. El controlador accede al modelo actualizando los datos o consultándolos.
4. El controlador envía los resultados de las acciones realizadas sobre el modelo a la vista. En algunas implementaciones de la arquitectura la vista accede directamente al modelo.
5. La vista muestra al usuario el resultado de su acción y queda a la espera de nuevas acciones.

4.10.2. Herramientas seleccionadas

La aplicación web propuesta en esta investigación como cliente para trabajar con el modelo propuesto e implementado, se desarrollo siguiendo el arquitectura Modelo Vista Controlador mencionada anteriormente. Las herramientas utilizadas en el desarrollo de la aplicación fueron las siguientes:

- Apache Tomcat: contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation que implementa las especificaciones de los servlets y de Java Server Page de Sun Microsystems. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. Un servlet es un programa escrito en Java que se ejecuta en el marco de un servicio de red (un servidor HTTP, por ejemplo), y que recibe y responde a las peticiones de uno o más clientes, o sea generan páginas Web de forma dinámica a partir de los parámetros de la petición que envíe el navegador Web. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor Web Apache. Es usado como servidor Web autónomo en entornos con alto nivel de tráfico y alta disponibilidad. Está desarrollado en el lenguaje de programación Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java. Los usuarios disponen de libre acceso a su código fuente y a su forma binaria en los términos establecidos en la Apache Software Licence. A partir de la versión 4.0, Jakarta Tomcat utiliza el contenedor de servlets Catalina. Para la aplicación se utilizo la versión 6.0.

- Java: lenguaje de programación para programar las bibliotecas de clases que funcionan como controlador. Es un lenguaje de programación desarrollado por Sun Microsystems a principios de los años 1990. Se puede utilizar de manera muy eficiente en cualquier tipo de aplicaciones. Posee varias características que ha motivado a la mayoría de los desarrolladores de software a emplearlo. Algunas de sus principales característica son: Es un lenguaje de programación orientado a objetos y de alto nivel; es distribuido ya que proporciona una colección de clases que se pueden emplear en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, lo que facilita el desarrollo de aplicaciones distribuidas; es robusto, porque proporciona numerosas comprobaciones en compilación y en tiempo de ejecución y un recolector de basura que permite liberar la memoria de forma automática, lo que le permite crear software altamente fiable; es seguro,

porque contiene barreras de seguridad implementadas en el lenguaje y en el sistema de ejecución en tiempo real; es portable, porque contiene una Máquina Virtual que se encarga de interpretar el código y es soportada en todas las plataformas; y es un lenguaje multitarea, ya que permite ejecutar varias acciones de forma sincronizada . Además está distribuido bajo la licencia GNU GPL, lo que lo convierte en un software libre. Se utlizo en la aplicación el JDK 1.6.

- JSP (Java Server Pages): es una tecnología orientada a crear páginas Web con programación en Java. Con JSP se puede crear aplicaciones Web que se ejecuten en variados Servidores Web, de múltiples plataformas, ya que Java es en esencia un lenguaje multiplataforma. Las páginas JSP están compuestas de código HTML/XML mezclado con etiquetas especiales para programar scripts de servidor en sintaxis Java. Por tanto, las páginas JSP se pueden escribir con un editor HTML/XML habitual. El motor de las páginas JSP está basado en los servlets de Java (programas en Java destinados a ejecutarse en el servidor), aunque el número de desarrolladores que pueden afrontar la programación de JSP es mucho mayor, dado que resulta mucho más sencillo de aprender que los servlets . Los archivos que se generan en JSP son de extensión .jsp que incluyen, dentro de la estructura de etiquetas HTML, las sentencias Java a ejecutar en el servidor. Antes de que sean funcionales los archivos, el motor JSP lleva a cabo una fase de traducción de esa página en un servlet, implementado en un archivo class (Byte codes de Java). Esta fase de traducción se lleva a cabo habitualmente cuando se recibe la primera solicitud de la página .jsp, aunque existe la opción de precompilar en código para evitar ese tiempo de espera la primera vez que un cliente solicita la página.

- JSTL y EL: La tecnología JSTL (JavaServer Pages Standar Tag Library) proporcionada por Sun Microsystems es una biblioteca que implementa funciones de uso frecuente en aplicaciones JSP, con el objetivo de simplificar y agilizar el desarrollo de aplicaciones Web, JSTL extiende las páginas JSP proporcionando cuatro librerías de etiquetas (Tag Libraries) con utilidades ampliamente utilizadas en el desarrollo de páginas Web dinámicas. Estas librerías de etiquetas son extensiones de la especificación de JSP, la cuál; a su vez extiende de la especificación de Servlet. Su API permite al usuario desarrollar librerías de etiquetas . En JSTL se engloban cuatro librerías: core, brinda funciones comunes de iteración sobre datos, operaciones condicionales e importación de otras páginas; xml, se utiliza para la manipulación de XML y para

XML-Transformation; sql, se emplea para gestionar conexiones a bases de datos y i18n, para la internacionalización y formateo de las cadenas de caracteres como cifras. Dentro de estas etiquetas, se utiliza un lenguaje llamado EL (Expression Language), que pretende ser un lenguaje más sencillo que Java, para realizar operaciones.

- NetBeans: IDE (Integrated Development Environment) de código abierto para desarrollar aplicaciones en Java, fundado por la Sun Microsystems, brinda la posibilidad de que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de Java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software . Es el principal rival del IDE Eclipse, es su equivalente pero patrocinado por Sun Microsystems. Actualmente existen funcionalidades disponibles en NetBeans que no lo están en Eclipse como por ejemplo el soporte nativo de las aplicaciones Web y J2EE. Al igual que ocurre con Eclipse, NetBeans se puede extender mediante la inclusión de plugins y existe una comunidad activa de desarrolladores. Está disponible también para Mac OS X. Actualmente la versión más estable del NetBeans, es la versión 6.0 que posee las siguientes características : mejoras en el editor de código (Completamiento de código más rápido e inteligente), las variables comunes se resaltan, soporte de varios lenguajes (JRuby, Ruby, Ruby on Rails) con excelentes plantillas, presenta una auto-estructuración de carpetas, dispone de plugins para PHP y Jython, tiene soporte para los lenguajes C/C++ (las librerías son fácilmente accedidas y también tiene una muy buena edición de código), presenta una interfaz gráfica de usuario (GUI) Swing (Aplicaciones de Base de Datos Swing) que probablemente sea la característica mas impresionante, es muy bueno para el desarrollo rápido de aplicaciones (RAD); otra característica es la mejora para desarrollar aplicaciones Web, por lo que incorpora el plugins Visual Web (posee JSF con lo que simplemente arrastrando los componentes se genera código de lógica de negocios como EJBs), y además posee un buen editor de JavaScript, con un depurador de error, no visto en ningún otro IDE. Se utilizo la versión 6.0.

4.10.3. Componentes de la aplicación Web

En la aplicación se definieron los tres componentes de la arquitectura de la siguiente forma (ver Figura 4.34).

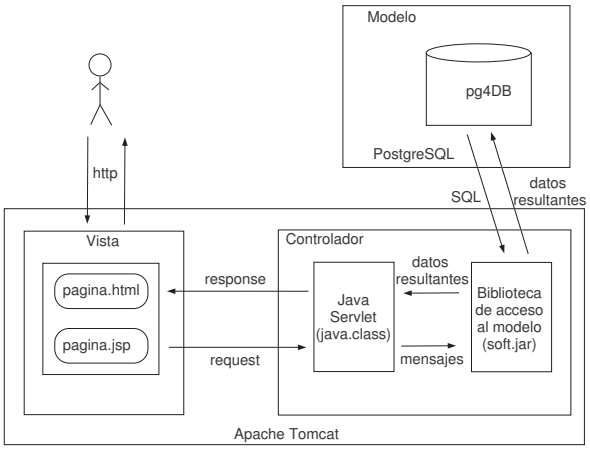


Figura 4.34: Arquitectura de la aplicación web.

El modelo en la arquitectura lo compone el servidor de bases de datos *PostgreSQL* y dentro de el las bases de datos que están definidas sobre la plantilla pg4DB que incorpora los elementos difusos propuesto en esta investigación. El acceso al modelo se realiza vía *TCP/IP* utilizando *SQL* como lenguaje de consulta. La estructura del modelo ha sido descrita en los apartados anteriores.

La vista esta formada por varias páginas *jsp* y *html*. Estas páginas muestran al usuario la información existente en el modelo y le permiten interactuar con el mismo por medio de evento. En estas páginas solo se presentan los datos y en su código se utilizan las *taglibs* (librerías de etiquetas)para determinadas funcionalidades. Dentro de las páginas solo existe código html y llamadas a etiquetas de las *tablibs*. Con este elemento se garantiza la separación entre el diseño y la manipulación de los datos.

El controlador fue implantado mediante dos elementos. Un primer elementos que son los propios *Java Servlet* encargados de manipular los objetos *request* y *response* para obtener y pasar información a la vista, respondiendo a eventos del usuario. Los *Java Servelet* fueron implementados según las características propias del modelo y esta formado por nueve clases.

El segundo elemento del controlador es una biblioteca de clases distribuida en forma de *jar*. Esta biblioteca fue implementada con la intención de acceder a todos los objetos

de *PostgreSQL* facilitando su manipulación en un ambiente de orientación a objetos. En la arquitectura planteada esta biblioteca es la única que se comunica directamente con el modelo. Cada uno de estos elementos están representados en la figura 4.34.

4.11. Conclusiones parciales

Con la implementación del modelo teórico propuesto en esta investigación en *PostgreSQL* se presenta un producto capaz de incorporar el trabajo don valores y objetos difusos. Estas extensiones serán de gran utilidad para desarrolladores y usuarios de bases de datos difusas.

EL proceso de implementación requirió de un esfuerzo adicional motivado por las carencias de *PostgreSQL* desde el punto de vista objeto-relacional. Pero finalmente utilizando de la herencia entre tablas, las funciones definidas por el usuario, la sobrecarga de funciones, la utilización de atributos definidos sobre vectores multidimensionales, la utilización de los identificadores de objetos generados internamente por *PostgreSQL* y el uso de los catálogos de *PostgreSQL* se logro implementar el modelo.

Realizando extensiones directamente sobre el *SQL* de *PostgreSQL*, se garantizo la interacción del usuario con las características difusas de la base de datos. Con esta implementación se garantiza el soporte de objetos descritos por atributos difusos y de objetos complejos en una base de datos *PostgreSQL*.

Finalmente el desarrollo de una aplicación web permite comprobar la validez de la propuesta y es un primer acercamiento para una distribución de un producto que permita de forma amigable manipular información difusos en una base de datos.

Capítulo 5

Ejemplo de aplicación

En los capítulos anteriores fue presentado el marco teórico del modelo de bases de datos difusas orientadas a objetos propuesto en esta investigación y los elementos de su implementación en el gestor de bases de datos con características objeto-relacional. En este capítulo se utilizan los resultados obtenidos en esta investigación para modelar un ejemplo, de forma que queden demostrados los aportes realizados en esta investigación y sus ventajas para la utilización en la resolución de problemas reales.

El ejemplo será presentado a partir de su modelado y su implementación mediante las extensiones realizadas al *SQL* soportado por *PostgreSQL*. Su representación se hará utilizando la aplicación web desarrollada. Finalmente se muestran algunas consultas que pueden ser realizadas en el ejemplo utilizado.

5.1. Introducción al problema/ejemplo: selección de estudiantes.

El contacto con la práctica profesional es uno de los componentes fundamentales en la formación del Ingeniero Informático en Cuba. En la Universidad de Holguín una de las formas de garantizar este componente es vincular a los estudiantes en el desarrollo de sistemas informáticos. Este proceso es dirigido por la Casa de Software, una institución dentro de la facultad que se encarga de desarrollar productos informáticos a pedido de las empresas del territorio. Uno de los procesos fundamentales dentro de la Casa de Software es la selección de estudiantes de la carrera para participar de dichos proyectos. Esta situación es la motivación del ejemplo que en este capítulo se presenta.

Por lo tanto como ejemplo se ha elegido el problema de la selección de estudiantes para participar en proyectos informáticos, en particular, el desarrollo de sistemas informáticos. La selección de personas para la realización de determinadas labores dentro de una institución es uno de los procesos de mayor importancia dentro de una empresa o entidad.

El proceso de selección de personal se basa fundamentalmente en escoger el individuo adecuado para ocupar el cargo en cuestión. Este proceso, dentro de una organización requiere primeramente de un grupo idóneo de candidatos. Estos candidatos son expuestos a una serie de pruebas como: entrevista preliminar, entrevista de selección, pruebas psicológicas, pruebas de trabajo, investigación laboral, etc. Por otro lado se tiene en cuenta el análisis del puesto a ocupar, el cual proporciona la descripción de las tareas, las especificaciones humanas y los niveles de desempeño que requiere cada puesto. Comparando ambos elementos, los especialistas seleccionan al personal que consideran idóneo.

En el problema de selección de estudiantes para participar en el desarrollo de proyectos informáticos se siguen los principios de un proceso básico de selección. Sin embargo por cuestión se simplicidad, el ejemplo que se presenta a continuación solo tendrá en cuenta las exigencias especificadas para trabajar un tema específico en un equipo de desarrollo y las características de cada estudiante, de forma tal que utilizando las capacidades de modelado y comparación de objetos y objetos complejos, soportada por el modelo de bases de datos difusas propuesto en esta investigación, se pueda realizar de forma sencilla una selección de los estudiantes que trabajarán en cada tema.

Además, en el ejemplo mostrado se almacena información de los proyectos y de los equipos, todo en un contexto donde se utiliza información imprecisa para representar los diferentes objetos involucrados en el problema.

5.2. Entidades y conceptos principales.

En el contexto relacionado con la selección de estudiantes para participar en proyectos informáticos intervienen varios factores y se almacena información de varias entidades. Los principales conceptos y entidades involucradas en este proceso están representadas en el modelo de la figura 5.1.

A partir de estos conceptos pueden definirse qué entidades son necesarias representar en el problema y a partir de éstas obtener las tablas para darle soporte en la base de datos.

Los principales características de cada una de las tablas son las siguientes:

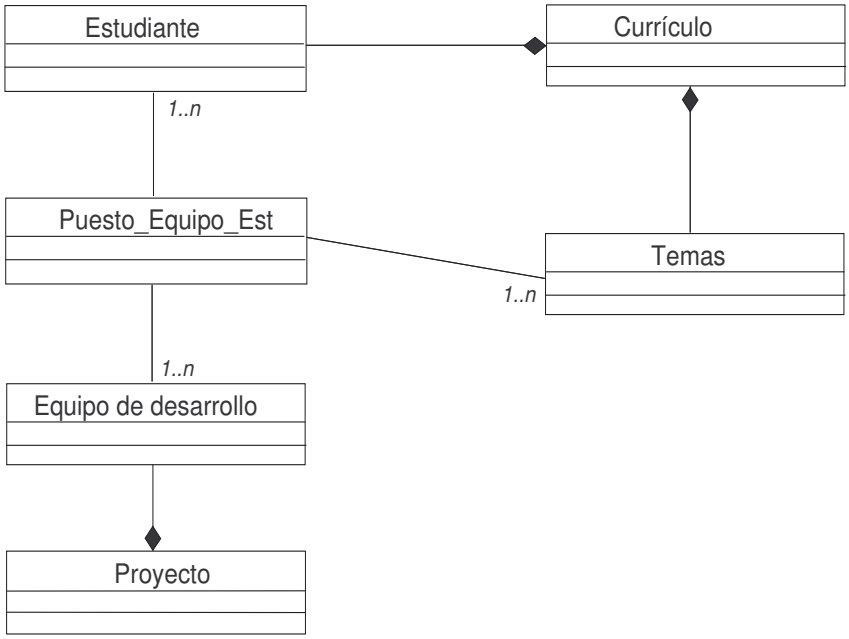


Figura 5.1: Diagrama de clases del problema.

Currículo Almacena las características de un estudiante desde el punto de vista de su posible participación en un proyecto informático. Además permite especificar los requisitos exigidos para un determinado tema.

Estudiante Almacena los datos generales del estudiante y además incorpora un atributo definido sobre la tabla currículo donde se especifican las características del estudiante.

Tema Almacena los datos sobre los temas y los requisitos exigidos para el mismo por medio de un objeto de tipo currículo.

Equipo de desarrollo Almacena la información referida al equipo que realizará un proyecto informático.

Proyecto Almacena los datos de un proyecto.

Tema_Equipo_Estudiante Almacena que estudiantes participan en un determinado tema dentro de un equipo de desarrollo.

En el próximo apartado se presentará un análisis detallado de la estructura de cada tabla. Se mostrarán los detalles de los atributos que caracterizan cada tabla.

5.3. Definición de atributos.

Realizando un análisis de los atributos involucrados en el problema se detecta que algunos de ellos tienen naturaleza imprecisa por lo que se utilizará el modelo propuesto en esta investigación para su representación.

A nivel de definición, la tabla *currículo* tiene niveles de precisión pues ella está definida por un conjunto de atributos que no siempre van a estar presentes en los objetos que de ella se creen. Esto está dado porque en dependencia del año en que está el estudiante, así serán los atributos dentro de su currículo que tendrán valor. Si el estudiante está en primer año no interesa almacenar el índice general que tiene porque aún no hay valores para ese atributo. Otros detalles se mostrarán más adelante.

Por otra parte varios de los atributos que caracterizan a las tablas presentan imprecisión. En esta investigación se dispone de un modelo con una amplia variedad de dominios difusos y otros elementos que permiten trabajar la imprecisión a nivel de atributos y de definición. Se mostrará como sacar provecho a este modelo para la implementación de una base de datos para el ejemplo tratado. Los detalles de cada atributo, así como su definición serán mostrados en el siguiente apartado.

5.3.1. Tabla currículo.

Como se mencionó anteriormente, el currículo de un estudiante dependerá del año en que esté matriculado, por lo que un objeto sobre esta tabla no siempre tendrá valor en todos sus atributos. Una representación plana de los objetos en estas situaciones provocará la existencia de valores nulos, situación esta que no es deseada. Por tales motivos se decidió representar esta tabla como un tipo difuso una forma de arreglar este problema.

Para el tipo difuso fueron definidos tres niveles, un primer nivel 1.0 que corresponde a estudiantes matriculados en el primer año de la carrera, un segundo nivel 0.8 para los estudiantes en segundo año y un tercer nivel 0.6 para los matriculados en tercero, cuarto y quinto año. Los valores de los grados no son relevantes para el problema y solo pretenden crear tres alfa cortes para el objeto. La definición de cada uno de los niveles se muestra en los próximos apartados.

5.3.1.1. Nivel 1.0 del tipo difuso currículo.

El nivel 1.0 del tipo difuso currículo corresponde a los estudiantes del primer año y en el se definen los atributos: disponibilidad, año, responsabilidad, personalidad, idioma, habilidades de comunicación, relaciones interpersonales, capacidad de análisis, idoneidad. Además de estos atributos se utiliza el *OID* proporcionado por *PostgreSQL* para identificar los objetos a falta de una llave en la relación. La definición de cada atributo es la siguiente.

1. **Disponibilidad**: Representa la disponibilidad del estudiante para participar en algún proyecto, viene dado por la disponibilidad personal del estudiante y por la participación del mismo en otras tareas. Se mide la disponibilidad en un rango de [0-100] siendo 0 disponibilidad nula y 100 máxima disponibilidad. Este atributo se representa mediante un *dominio atómico con representación semántica sobre referencial continuo* en el cual se definen las etiquetas lingüísticas: *ESCASA, BAJA, MEDIA, ALTA, MUY_ALTA y CASI_TOTAL* (ver Figura 5.2).

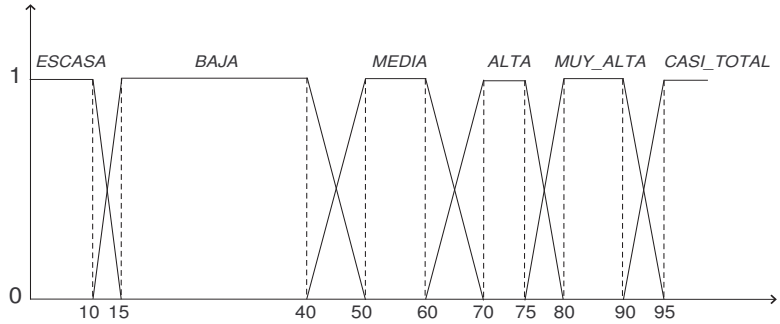


Figura 5.2: Dominio para el atributo disponibilidad.

2. **Año**: Se refiere al año que cursa actualmente el estudiante. El estudiante puede estar en primer año, segundo año, tercer año, cuarto año o quinto año. Es conveniente referirse al año que cursa un estudiante por medio de etiquetas lingüísticas definidas sobre el referencial de años, para lo cual se definieron las siguientes etiquetas: *INICIALES,MEDIOS y TERMINALES*, que representan, como su nombre lo indica, en que momento de la carrera se encuentra el estudiante. Estas etiquetas quedan definidas mediante las siguientes distribuciones de posibilidad:

   INICIALES=1.0/primero,0.85/segundo,0.5/tercero,0.25/cuarto

   MEDIOS=0.2/primero,0.8/segundo,1/tercero,0.8/cuarto,0.2/quinto

TERMINALES=0.25/segundo,0.65/tercero,0.85/cuarto,1/quinto

Para este atributo se definió un dominio *atómico con representación semántica sobre referencial finito de valores* con el siguiente dominio básico {*primero, segundo, tercero, cuarto, INICIALES, MEDIOS, TERMINALES*}

En este caso al introducir el año en que está un estudiante es una información precisa, sin embargo, se decidió tratar el año como un dominio difusos pensando en ser utilizado en consultas sobre la base de datos.

3. **Responsabilidad:** Representa el nivel de responsabilidad que tiene un estudiante y al no existir un referencial asociado a este tipo de atributo se utiliza un *dominio atómico sin representación semántica*. Se definen cuatro posibles valores: *MUY_POCA, POCA, MEDIA y MUCHA*. La relación de semejanza entre ellos se muestra en la tabla 5.1.

Cuadro 5.1: Relación de semejanza para el atributo *responsabilidad*.

	MUY_POCA	*POCA*	*MEDIA*	*MUCHA*
MUY_POCA	1.0	0.6	0.4	0.2
POCA		1.0	0.5	0.3
MEDIA			1.0	0.7
MUCHA				1.0

4. **Personalidad:** Se quiere almacenar los rasgos de la personalidad del estudiante y en qué grado posee cada rasgo. En tal sentido se define un *dominio conjuntivo sobre un referencia de objetos* difusos simples que en su caso representa rasgos de personalidad. El referencial del dominio son objetos difusos con un atributo definido sobre un dominio atómico sin representación semántica que tiene una relación de semejanza según la tabla 5.2.

Cuadro 5.2: Relación de semejanza para el atributo *personalidad*.

	Honesto	*Trabajador*	*Independiente*	*Creativo*	*Líder*
Honesto	1.0	0.8	0.4	0.3	0.3
Trabajador		1.0	0.6	0.4	0.3
Independiente			1.0	0.8	0.9
Creativo				1.0	0.9
Líder					1.0

5. **Idiomas:** Son los idiomas que domina el estudiante. El valor de este atributo es un subconjunto difuso de idiomas y la pertenencia de cada idioma al subconjunto representa el nivel de dominio del idioma por el estudiante. Para este atributo se utilizará un *dominio conjuntivo sobre un referencial de valores.*

6. **Comunicación:** Representa la habilidad del estudiante de comunicarse con sus compañeros. Por ello, se utiliza un *dominio atómico sin representación semántica*, donde se definen las etiquetas y la relación de semejanza existente entre ellas, como muestra la tabla (5.3).

Cuadro 5.3: Relación de semejanza para el atributo *comunicación.*

	MUY_POCA	*POCA*	*MEDIA*	*ALTA*	*MUY_ALTA*
MUY_POCA	1.0	0.8	0.5	0.3	0.0
POCA		1.0	0.7	0.4	0.2
MEDIA			1.0	0.65	0.4
ALTA				1.0	0.85
MUY_ALTA					1.0

7. **Relaciones interpersonales:** Las relaciones personales del estudiante con el resto de los estudiantes es un atributo que interesa almacenar. Para medir el nivel en que establece las relaciones interpersonales el estudiante se definieron las siguientes etiquetas lingüísticas: *MUY_MALAS, MALAS, ACEPTABLES, BUENAS, MUY_BUENAS y EXCELENTE.* Se utiliza un *dominio atómico sin representación semántica* para el atributo y se establece la siguiente relación de semejanza (ver Tabla 5.4).

Cuadro 5.4: Relación de semejanza para el atributo *relaciones interpersonales.*

	M_MAL.	*MALA*	*ACEPT.*	*BUENAS*	*M_BUE.*	*EXC.*
MUY_MALAS	1.0	0.85	0.6	0.4	0.25	0.0
MALAS		1.0	0.8	0.45	0.35	0.1
ACEPTABLES			1.0	0.6	0.4	0.25
BUENAS				1.0	0.8	0.5
MUY_BUENAS					1.0	0.85
EXCELENTE						1.0

8. **Capacidad de análisis:** La capacidad de análisis que tenga el estudiante es una información importante para participar en equipos de desarrollo de software. Este atributo, por sus características, se define utilizando un *dominio atómico sin representación semántica* donde se definen las etiquetas: *BAJA, MEDIA, ALTA y MUY_ALTA*. La relación de semejanza que se establece entre las etiquetas es la representada en la tabla 5.8.

Cuadro 5.5: Relación de semejanza para el atributo *capacidad de análisis*.

	BAJA	*MEDIA*	*ALTA*	*MUY_ALTA*
BAJA	1.0	0.5	0.35	0.2
MEDIA		1.0	0.7	0.4
ALTA			1.0	0.8
MUY_ALTA				1.0

9. **Idoneidad:** La idoneidad es un atributo que refleja el nivel en que un estudiante tiene las características deseadas para participar en un equipo de desarrollo de software. La idoneidad toma valor en el rango [0,10], siendo 10 el valor que refleja la mejor idoneidad. El atributo se define sobre un *dominio atómico con representación semántica sobre referencial continuo*, en el cual se especifican cuatro etiquetas para referir la idoneidad (ver Figura 5.3).

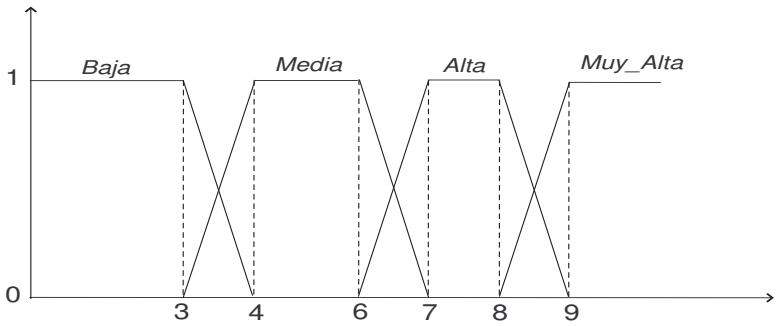


Figura 5.3: Dominio para el atributo idoneidad.

Este atributo toma su valor a partir del valor de otros atributos, por tal razón se considera un *atributo inferido*. Se define el siguiente sistema de reglas difusas para el atributo.

```
R1: IF m_disponibilidad FGT MUY_ALTA AND m_anio FEQ MEDIOS AND
```

```
 m_responsabilidad FEQ MUCHA AND m_rel_int FEQ ACEPTABLES AND
  m_cap_anal FEQ ALTA THEN m_idoneidad IS MUY_ALTA,
R2:
IF m_disponibilidad FEQ BAJA AND m_anio FEQ TERMINALES
THEN m_idoneidad IS BAJA,
R3:
IF m_disponibilidad FEQ MEDIA OR m_disponibilidad FEQ ALTA AND
 m_responsabilidad FEQ MEDIA AND m_cap_anal FEQ ALTA
 THEN m_idoneidad IS MEDIA,
R4:
IF m_disponibilidad FGEQ MUY_ALTA AND m_anio FEQ INICIALES AND
 m_responsabilidad FEQ MUCHA AND m_cap_anal FEQ MUY_ALTA
 THEN m_idoneidad IS ALTA,
R5:
m_anio FEQ 5 THEN m_idoneidad IS BAJA
```

El consecuente del atributo queda definido de la misma forma sobre el dominio de la idoneidad y se trabajará con un umbral de 0.0. Con el objetivo de una mayor claridad y sencillez del ejemplo la semántica de este sistema de reglas tiene un objetivo demostrativo, por lo tanto no se adapta totalmente a la realidad del problema aunque si pretende un acercamiento.

En este caso el estudiante tendrá una idoneidad muy alta si su disponibilidad es posiblemente mayor difuso que muy alta, el año que cursa es de los años medios, la responsabilidad es mucha, tiene aceptables relaciones interpersonales y alta capacidad de análisis. La idoneidad es baja si la disponibilidad del estudiante es baja y esta en los años terminales de la carrera. Tendrá una idoneidad media si la disponibilidad es media o alta y la responsabilidad es media y la capacidad de análisis alta.

Si la disponibilidad es posiblemente mayor o igual difuso que muy alta, el estudiante esta en los años iniciales y tiene mucha responsabilidad y muy alta capacidad de análisis la idoneidad es alta. Finalmente si el estudiante está en el quito año de la carrera la idoneidad es baja.

5.3.1.2. Nivel 0.8 del tipo difuso currículo.

Para los estudiantes de segundo año se incorporan nuevos atributos a su currículo como son: índice general, índice en la disciplina de matemática, índice en la disciplina de física, índice en la disciplina de idioma, índice en la disciplina de sistemas digitales, índice en la disciplina de ciencias empresariales, índice en la disciplina de matemática aplicada, índice en la disciplina de inteligencia artificial, índice en la disciplina de tecnología de la programación, índice en la disciplina de informática industrial, capacidad para el trabajo en equipo, independencia para el trabajo y rendimiento. Estos atributos serán junto con los del nivel 1.0, el soporte del nivel 0.8 del tipo difuso currículo. Los detalles de cada atributo son los siguientes:

1. **Índice General:** Este atributo representa el índice general que tiene acumulado el estudiante. El índice es un promedio de las notas obtenidas por el estudiante en las asignaturas cursadas. El valor de este atributo estará entre [3,6], es un valor preciso pero con el objetivo de realizar consultas difusas sobre este dato, se define sobre un dominio difuso. Es conveniente tratar esta información por medio de etiquetas lingüísticas y con este objetivo se definieron las etiquetas: *MUY_BAJO, BAJO, MEDIO, ALTO y MUY_ALTO*, con relaciones de pertenencia representadas por funciones trapezoidales (véase Figura 5.4).

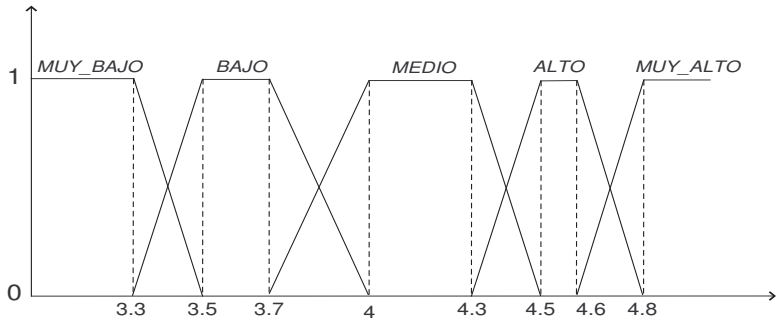


Figura 5.4: Dominio para los atributos índice.

2. **Índice por disciplinas:** En este caso se necesita tener un grupo de atributos que representan los índices alcanzados por el estudiante en cada una de las disciplinas que se imparten en la carrera. Una disciplina esta compuesta por varias asignaturas y las asignaturas tiene un orden en el que deben ser cursadas y aprobadas por los

estudiantes. En este caso también se utiliza un dominio difuso para poder realizar consultas en este sentido. Los índices que se almacenarán en cada estudiante son:

- Índice en la disciplina de matemática.
- Índice en la disciplina de física.
- Índice en la disciplina de idioma.
- Índice en la disciplina de sistemas digitales.
- Índice en la disciplina de ciencias empresariales.
- Índice en la disciplina de matemática aplicada.
- Índice en la disciplina de inteligencia artificial.
- Índice en la disciplina de tecnología de la programación.
- Índice en la disciplina de informática industrial.

El dominio de estos atributos será un *dominio atómico con representación semántica sobre referencial continuo*. Se emplea el mismo dominio definido para el atributo *índice general* (ver Figura 5.4).

3. **Trabajo en equipo:** Para pertenecer a un equipo de desarrollo de un producto informático se requiere que el estudiante tenga aptitudes para el trabajo en equipo. Este atributo almacenará un valor para la aptitud de trabajo en grupo del estudiante. El atributo quedará definido sobre un *dominio atómico sin representación semántica*, para el cual se definen los valores: *MUY_POCA, POCA, MEDIA, ALTA y MUY_ALTA* y una relación de semejanza (ver Tabla 5.6).

Cuadro 5.6: Relación de semejanza para el atributo *trabajo en equipo*.

	MUY_POCA	*POCA*	*MEDIA*	*ALTA*	*MUY_ALTA*
MUY_POCA	1.0	0.8	0.5	0.3	0.0
POCA		1.0	0.7	0.4	0.2
MEDIA			1.0	0.65	0.4
ALTA				1.0	0.85
MUY_ALTA					1.0

4. **Capacidad para el trabajo independiente:** De la misma forma que es necesario conocer las aptitudes para el trabajo en equipo, se requiere conocer cuán independiente es el estudiante durante el trabajo. Para almacenar esto se crea un atributo

que estará definido sobre el mismo dominio que el atributo *trabajo en equipo* pero que representará la independencia del estudiante.

5. **Rendimiento:** El rendimiento es un atributo que refleja un valor de como ha rendido el estudiante hasta el momento, el valor de este atributo dependerá de los valores que tenga la idoneidad del estudiante y el índice general que tiene. Por lo tanto se define como un atributo inferido que tendrá en cuenta el siguiente sistema de reglas difusa.

```
R1:
IF m_idoneidad FGEQ ALTA AND indice_general FGT ALTO
THEN m_rendimiento IS MUY_ALTA,
R2:
IF m_idoneidad FGEQ ALTA AND indice_general FEQ MEDIO
THEN m_rendimiento IS ALTA,
R3:
IF m_idoneidad FEQ MEDIA AND indice_general FEQ MUY_ALTO
THEN m_rendimiento IS ALTA,
R4:
IF m_idoneidad FEQ MEDIA AND indice_general FGEQ MUY_ALTO
THEN m_rendimiento IS MEDIA,
R5:
IF indice_general FLT MEDIO THEN m_rendimiento IS BAJA,
R6:
IF m_idoneidad FEQ BAJA THEN m_rendimiento IS BAJA
```

El dominio del atributo será el mismo que se definió para el atributo idoneidad y el tipo del consecuente del sistema de reglas será ese mismo dominio. Se utiliza un umbral de 0.0. La semántica en este caso también es demostrativa y queda expresada de la siguiente forma: si la idoneidad es posiblemente mayor o igual que alta y el índice general es mayor que alto entonces el rendimiento del estudiante es muy alto; si la idoneidad es posiblemente mayor o igual que alta y el índice general es medio el rendimiento es alto; si la idoneidad es media y el índice general muy alto el rendimiento es alto; si la idoneidad es media y el índice general posiblemente mayor o igual que muy alto el rendimiento es medio; si el índice general es menor de medio el rendimiento es bajo y por último si la idoneidad es baja el rendimiento es bajo.

Es importante notar que para este atributo inferido su valor depende del resultado de otro valor inferido.

5.3.1.3. Nivel 0.6 del tipo difuso currículo.

El último de los niveles definidos para el tipo difuso currículo es el 0.6, en este nivel además de los atributos incluidos en el resto de los niveles se incluyen los siguientes atributos: índice en la disciplina de ingeniería de software, roles que puede desarrollar el estudiante, lenguajes de programación que domina el estudiante, experiencia que tiene el estudiante en diferentes tipos de aplicaciones. La descripción de cada atributo es la siguiente:

1. **Índice de la disciplina ingeniería de software:** Este atributo almacena el índice obtenido por el estudiante en las asignaturas agrupadas en esta disciplina. El dominio para el atributo es el mismo que para el atributo *índice general* definido para el nivel 0.8 (véase Figura 5.4).

2. **Roles desarrollados:** Durante la carrera los estudiantes pueden ir desarrollando los diferentes roles que existen en la ingeniería de software: Analista, Diseñador, Arquitecto, Programador, Probador, Planificador, Especialista en Soporte, Especialista en Calidad, Especialista en Seguridad, Gestor de Versiones, Gestor de Cambios y Config., Implantador, Jefe de Proyecto, otros. Este atributo pretende almacenar el nivel de desarrollo que tiene el estudiante en determinados roles, para esto se definió el atributo sobre un *dominio conjuntivo sobre un referencial de valores*.

3. **Lenguajes de programación:** Es importante conocer cuáles son los lenguajes de programación que domina un estudiante. En los tres primeros años de la carrera el estudiante se forma como programador y a partir de ese momento es que se puede considerar que domina un lenguaje. Como un estudiante puede dominar varios lenguajes, cada uno en determinado nivel, es necesario tener un valor conjuntivo para este atributo. En tal sentido se definió el atributo con un *dominio conjuntivo sobre un referencial de objetos*. Los objetos del referencial serán sencillos pues tendrá una sola propiedad definida sobre un *dominio atómico sin representación semántica*, donde se refleja la semejanza que puede existir entre los diferentes lenguajes de programación (ver Tabla 5.7).

Cuadro 5.7: Relación de semejanza entre lenguajes de programación

	Java	JScript	JSP	C	C++	C#	Vis Basic	VB Script	Pascal	Obj Pascal	PHP	VB Net	ASP Net
Java	1	0,45	0,95	0,6	0,7	0,65	0,1	0,1	0,25	0,3	0,1	0,35	0,23
Java Script		1	0,5	0,4	0,4	0,4	0,15	0,8	0,1	0,1	0,6	0,4	0,5
JSP			1	0,6	0,65	0,55	0,1	0,2	0,2	0,3	0,65	0,3	0,6
C				1	0,9	0,75	0,15	0,1	0,45	0,37	0,35	0,25	0,2
C++					1	0,8	0,15	0,1	0,37	0,47	0,37	0,3	0,25
C#						1	0,15	0,1	0,3	0,37	0,37	0,35	0,7
Visual Basic							1	0,65	0,25	0,2	0,15	0,7	0,5
VB Script								1	0,1	0,1	0,33	0,3	0,45
Pascal									1	0,82	0,15	0,1	0,1
Object Pascal										1	0,15	0,15	0,1
PHP											1	0,3	0,33
VB .Net												1	0,83
ASP .Net													1

4. **Experiencia en el desarrollo de aplicaciones:** La experiencia que pueda tener un estudiante en el desarrollo de determinados tipos de aplicaciones es importante. Se quiere almacenar la experiencia del estudiante en las siguientes aplicaciones:
   - Aplicaciones Desktop
   - Aplicaciones Web
   - Aplicaciones Multimedia
   - Aplicaciones Empresariales
   - Aplicaciones Educativas

La experiencia viene dada por la cantidad de aplicaciones de cada tipo en las que ha participado el estudiante. Para este atributo se definió un dominio *atómico con representación semántica sobre referencial finito de valores* sobre el cual se definieron las etiquetas: *MUY_POCA,POCA,MEDIA y BASTANTE* con las siguientes distribuciones de posibilidad.

MUY_POCA=1.0/1,0.8/2,0.5/3,0.2/4
POCA=0.3/1,0.8/2,1.0/3,1.0/4,0.85/5,0.4/6,0.2/7
MEDIA=0.3/2,0.45/3,0.8/4,1.0/5,1.0/6,0.85/7,0.6/8,0.3/9,0.1/10
BASTANTE=0.4/4,0.75/5,0.9/6,1.0/7,1.0/8,1.0/9,1.0/10

5.3.2. Tabla Estudiante.

La tabla estudiante incluye entre sus atributos un objeto del tipo currículo y los atributos DNI, nombre, fecha de nacimiento, dirección, municipio, índice de ingreso, recomendación e idoneidad del estudiante. Los detalles de cada atributo son los siguientes:

1. Los atributos DNI (identificación personal del estudiante), nombre, fecha de nacimiento, dirección y municipio se definen sobre dominios precisos soportados por el gestor de bases de datos.
2. **Becado:** Atributo preciso que representa si el estudiante está becado o no.
3. **Horario:** Representa el horario en que el estudiante recibe las clases, los posibles valores son: mañana y tarde. Es un atributo preciso.
4. **Alumno Ayudante:** Los estudiantes pueden ser o no alumnos ayudantes, con lo cual este atributo es preciso y sus valores posibles son sí y no.

5. **Índice de ingreso**: Se refiere al índice con que ingresó el estudiante a la universidad. Este índice toma un valor entre [60,100] y para el se definieron las siguientes etiquetas lingüísticas: *MALO, REGULAR, BUENO y EXCELENTE* con las funciones de pertenencias que se muestran en la figura (5.5).

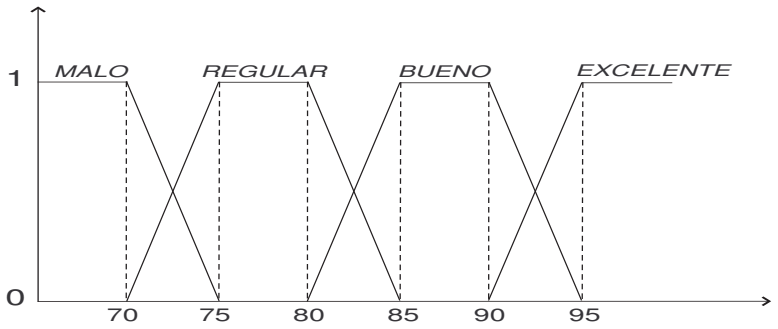


Figura 5.5: Dominio para el atributo índice de ingreso.

6. **Recomendación:** Al estudiante terminar su pre universitario se le realiza un dictamen sobre los resultados obtenidos y comportamiento mantenido durante sus estudios. Esta información se almacena en el atributo recomendación que es un atributo definido sobre un *dominio atómico sin representación semántica*. Las etiquetas definidas para este dominio y las relación de semejanza existente entre ellas se muestra en la tabla 5.8.

Cuadro 5.8: Relación de semejanza para el atributo *capacidad de análisis*.

	MALA	*REGUL.*	*BUENA*	*MUY_BUE.*
MALA	1.0	0.55	0.25	0.0
REGULAR		1.0	0.75	0.55
BUENA			1.0	0.85
MUY_BUENA				1.0

7. **Clasificación:** A partir del valor del índice de ingreso y la recomendación que trae el estudiante de su enseñanza anterior se clasifica al estudiante. El atributo que representa esta clasificación queda definido sobre el dominio especificado para la idoneidad (ver Figura 5.3). Se considera este atributo como un atributo inferido para el cual se define el siguiente sistema de reglas difusas.

```
R1:
```

```
IF m_ind FLEQ REGULAR THEN m_clasif IS BAJA,
R2:
IF m_ind FGT BUENO AND (m_reco IS BUENA OR m_reco IS MUY_BUENA)
THEN m_clasif IS MUY_ALTA,
R3:
IF m_ind FGT BUENO AND m_reco IS REGULAR THEN m_clasif IS ALTA,
R4:
IF m_ind FGT BUENO AND m_reco IS MALA THEN m_clasif IS MEDIA
```

Como se aprecia el consecuente de las reglas se define también sobre el dominio idoneidad y se especifica para el sistema de reglas un umbral de 0.0. En este caso la semántica del sistema de regla es a modo ilustrativo y en el se define que se clasifica al estudiante de forma baja si su índice de ingreso (m_ind) es menor o igual que regular; si el índice de ingreso es mayor que bueno y la recomendación (m_reco) es buena o muy buena entonces la clasificación es muy alta; la clasificación es alta si el índice es mayor que bueno y la recomendación es regular; y por último la clasificación es media si el índice de ingreso es mayor que bueno y la recomendación es mala.

5.3.3. Tabla Proyecto.

La tabla proyecto tiene los siguientes atributos: código del proyecto, nombre del proyecto, institución, fecha inicio, duración, financiación y tipo de proyecto. Los dominios que se definen para cada atributo son los siguientes:

- Los atributos código del proyecto (identifica al proyecto), nombre del proyecto (varchar), institución (varchar), fecha inicio (date) se definen sobre dominios precisos.
- **Duración:** Es la duración que tendrá el proyecto. Esta duración se mide en meses y para ella se definió un *dominio atómico con representación semántica sobre referencial continuo* representado en la figura 5.6.
- **Financiación:** Representa el monto de la financiación de un proyecto. Para este atributo también se define un *dominio atómico con representación semántica sobre referencial continuo* con las etiquetas lingüísticas: *POCO, MEDIO, GRANDE y MUY_GRANDE*. Las distribuciones de posibilidad de cada etiqueta se muestran en la figura 5.7.

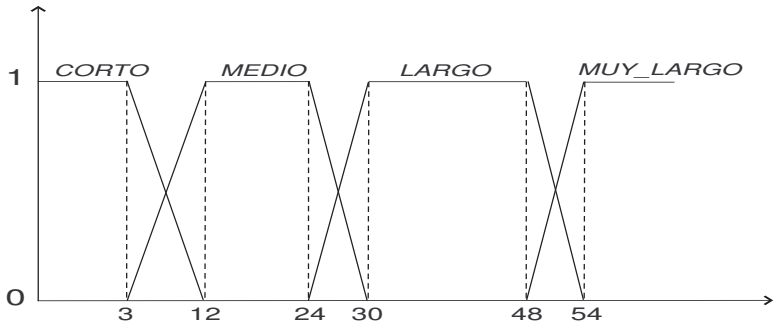


Figura 5.6: Dominio para el atributo duración de un proyecto.

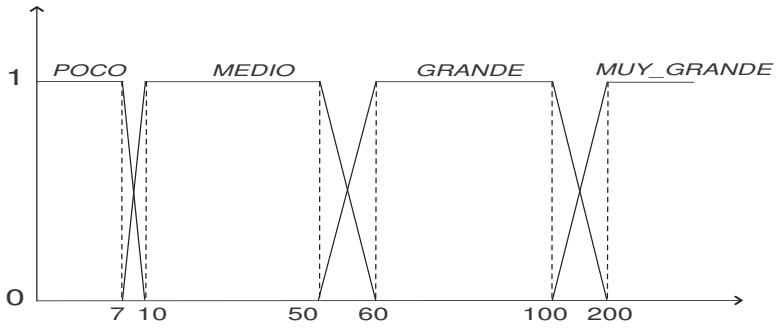


Figura 5.7: Dominio para el atributo financiación de un proyecto.

- **Tipo de proyecto:** El tipo del proyecto es un atributo que toma su valor a partir de lo valores especificados para la duración y la financiación del proyecto. Este atributo queda definido sobre un dominio *atómico sin representación semántica* en el que se definen los siguientes valores: *SERVICIO, PRODUCCION y ESTRATEGICO* y con una relación de semejanza entre los valore según la tabla 5.9.

Cuadro 5.9: Relación de semejanza para el atributo *capacidad de análisis*.

	SERVICIO	*PRODUC*	*ESTRATEG*
SERVICIO	1.0	0.6	0.3
PRODUCCION		1.0	0.85
ESTRATEGICO			1.0

Como el valor del atributo depende del valor que tomen otros atributos, se define este atributo como un atributo inferido y en tal sentido se define el siguiente sistema de reglas.

```
R1:
IF duracion FEQ MUY_LARGO AND finan FGEQ GRANDE
THEN tipoproy IS ESTRATEGICO,
R2:
IF duracion FGT LARGO AND finan FEQ MEDIO
THEN tipoproy IS PRODUCCION,
R3:
IF duracion FLEQ MEDIO THEN tipoproy IS SERVICIO
```

Como se aprecia el consecuente de las reglas se define también sobre el dominio tipo de proyecto y se especifica para el sistema de reglas un umbral de 0.0. La semántica del sistema de reglas es que si la duración del proyecto es muy larga y la financiación es mayor o igual que grande el proyecto es estratégico; si la duración es es mayor que larga y la financiación es media el proyecto es de producción y por último si la duración es menor o igual que media el proyecto es de servicio.

5.3.4. Tabla Tema.

La tabla tema describe las características de cada tema. Esta tabla incluye los atributos: código del tema, nombre del tema, currículo y dedicación requerida por el tema.

El atributo currículo es del tipo difuso currículo definido anteriormente (ver Epígrafe 5.3.1). La dedicación representa el nivel de dedicación que se requiere para el tema; este atributo queda definido por un dominio atómico con representación semántica sobre referencial continuo y su comportamiento es el mismo que el del dominio disponibilidad (véase Figura 5.2).

5.3.5. Tabla Equipo de trabajo.

Esta tabla representa los equipos de trabajo que se forman para ejecutar un proyecto y sus atributos son: código del equipo, nombre del equipo, proyecto con el que esta asociado y un conjunto de los puestos que se incluyen en el equipo y su grado de vinculación con el equipo.

Los atributos código del equipo, nombre del equipo son de dominios precisos. El atributo proyecto es un objeto de la tabla proyecto y el conjunto de puestos se define sobre un dominio conjuntivo de objetos, en este caso de objetos de la tabla tema.

5.3.6. Tabla Tema-Equipo-Estudiante.

La tabla Tema-Equipo-Estudiante representa la asignación de un estudiante a un puesto para trabajar en un equipo. Además se almacena la dedicación que requerirá el estudiante en ese tema y la labor que desempeñará en el mismo. La dedicación se define mediante el dominio disponibilidad (ver Figura 5.2).

Luego de definidas las tablas, los atributos que las caracterizan y los detalles de los dominios difusos involucrados en el problema se puede utilizar el modelo propuesto en esta investigación para implementar la base de datos.

5.4. Implementación utilizando SQL99 y la aplicación Web.

A partir del análisis de los atributos que caracterizarán cada tabla de la base de datos, puede notarse que en todos los casos se esta en presencia de tablas con atributos difusos ($TWFA$). A partir de este momento todas las tablas de la base de datos serán referenciadas de esta forma.

Se utilizará el modelo *pg4DB* creado en esta investigación para implementar una base de datos difusa que permita representar un modelo de datos para el problema con el que se trabaja. El primer elemento es crear la base de datos.

Se creará una base de datos que tendrá como plantilla el modelo *pg4DB* con lo cual se le incorporará todas las funcionalidades descritas en el transcurso de esta memoria. La sentencia básica en SQL para crear la nueva base de datos es la siguiente:

```
CREATE DATABASE SelecEst TEMPLATE pg4DB;
```

Con la aplicación Web se crea la base de datos, ya con la plantilla *pg4DB* asignada como se muestra en la figura 5.8.

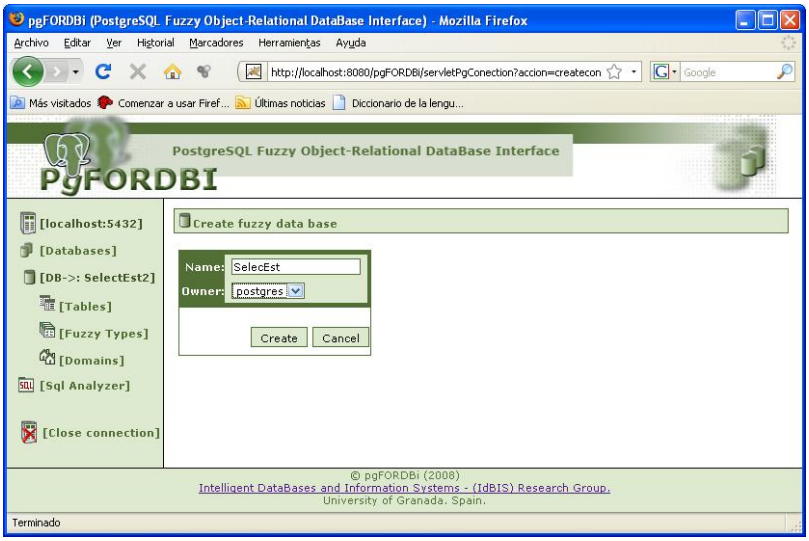


Figura 5.8: Crear una base de datos.

De esta forma en el servidor queda creada la base de datos *SelecEst* con todas las funcionalidades difusas desarrolladas en esta investigación.

Una vez creada la base de datos se debe seguir la metodología mostrada en la figura 5.9 para realizar la implementación del modelo de dato en *pg4DB*.

El proceso comienza por la definición de los dominios difusos, una vez que se tengan los dominios difusos se pueden crear las *TWFAs* del sistema o los tipos difusos. Si una *TWFA*

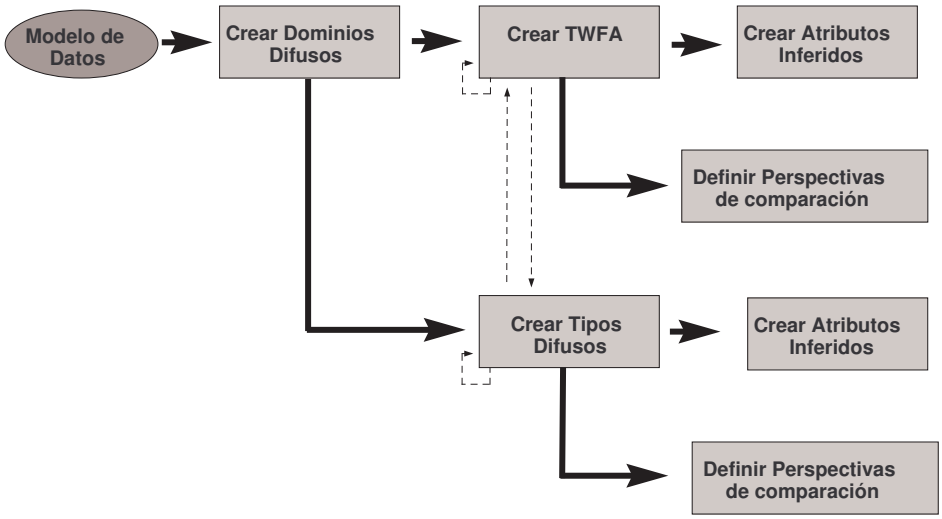


Figura 5.9: Metodología para utilizar *pg4DB*.

tiene un atributo definido sobre un tipo difuso, debe crearse inicialmente el tipo difuso. Un tipo difuso puede tener también un atributo definido sobre una *TWFA*. Además una *TWFA* puede tener una atributo definido sobre otra *TWFA* y de la misma forma sucede con los tipos difuso.

Con las *TWFAs* y los tipos difusos creados se puede especificar atributos inferidos y perspectivas de comparación para cada uno de ellos. A partir de aquí ya el modelo esta listo para introducir datos y realizar consultas sobre los mismos.

Se seguirá la metodología propuesta en la figura 5.9 para ir mostrando la implementación del ejemplo.

5.4.1. Crear dominios difusos.

Como se vio en apartados anteriores es necesario definir dominios difusos de casi todos los tipos soportados por el modelo. La definición de cada uno de los dominios necesarios para el problema abarcaría mucho espacio por lo que se presenta un ejemplo de cómo se define cada tipo de dominio.

El dominio atómico sin representación semántica asociada es utilizado en varios atributos, por lo que es necesario definir los siguientes dominios: responsabilidad, personalidad, comunicación, relaciones personales, capacidad de análisis, trabajo en equipo, lenguajes de programación, recomendación y tipo de proyecto. Además se deben definir los dominios

personalidad y lenguajes de programación que se utilizarán como referencial para dominios conjuntivos.

La definición del dominio responsabilidad se realiza de la siguiente forma:

```
SELECT fuzzy_schema.Create_Domain_WR('responsabilidad',
'{MUY_POCA,POCA,MEDIA,MUCHA}',
'{{1,0.6,0.4,0.2},{0.6,1,0.5,0.3},{0.4,0.5,1,0.7},{0.2,0.5,0.7,1}}');
```

Este tipo de dominio utilizando la aplicación Web desarrollada se define según las figuras (5.10, 5.11 y 5.12)

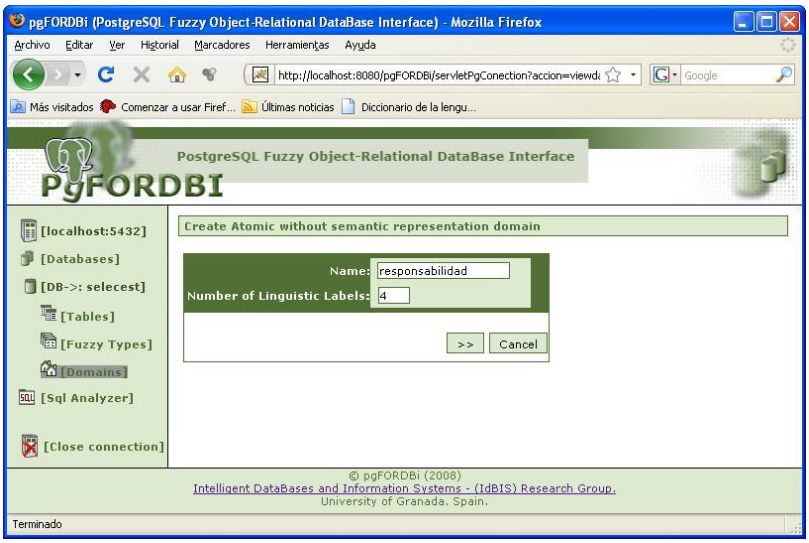


Figura 5.10: Crear dominio responsabilidad 1/3.

De la misma forma los atributos disponibilidad, idoneidad, índice, índice de ingreso, tiempo y financiación se definen sobre un dominio atómico con referencial continuo. La creación del dominio disponibilidad se realiza mediante estas sentencias *SQL*.

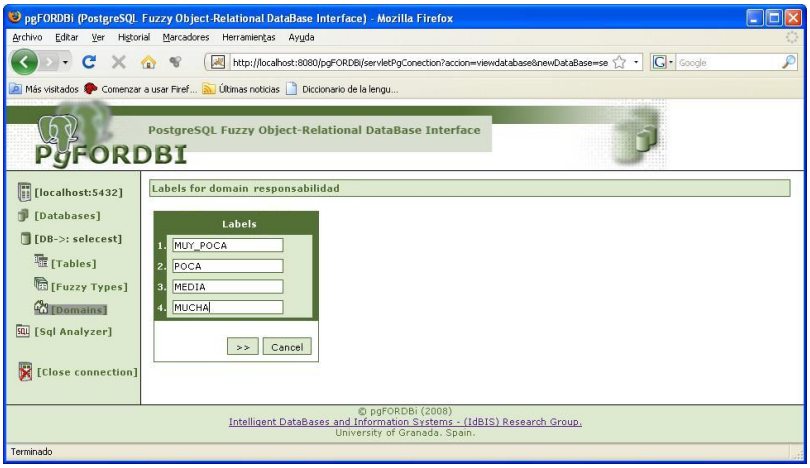


Figura 5.11: Crear dominio responsabilidad 2/3.

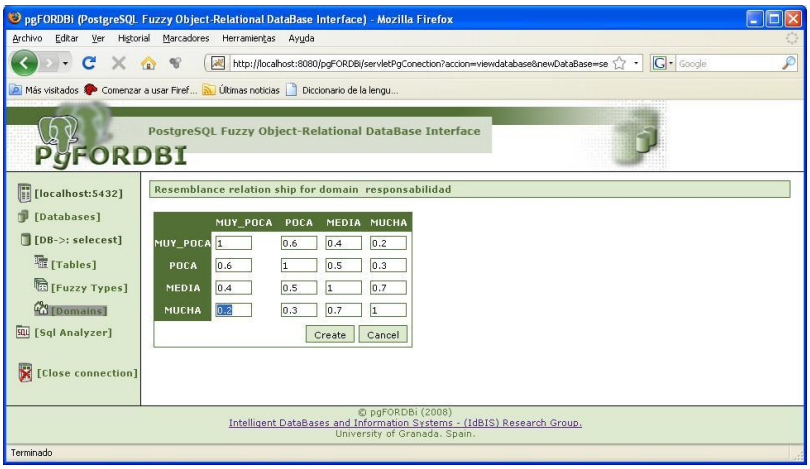


Figura 5.12: Crear dominio responsabilidad 3/3.

```
SELECT fuzzy_schema.Create_Domain_DT('disponibilidad',
'{ESCASA,BAJA,MEDIA,ALTA,MUY_ALTA,CASI_TOTAL}',
'{{0,0,10,15},{10,15,40,50},{40,50,60,70},{60,70,75,80},
 {75,80,90,95},{90,95,100,100}}');
```

La figura 5.13 muestra la creación de este dominio mediante la aplicación Web.

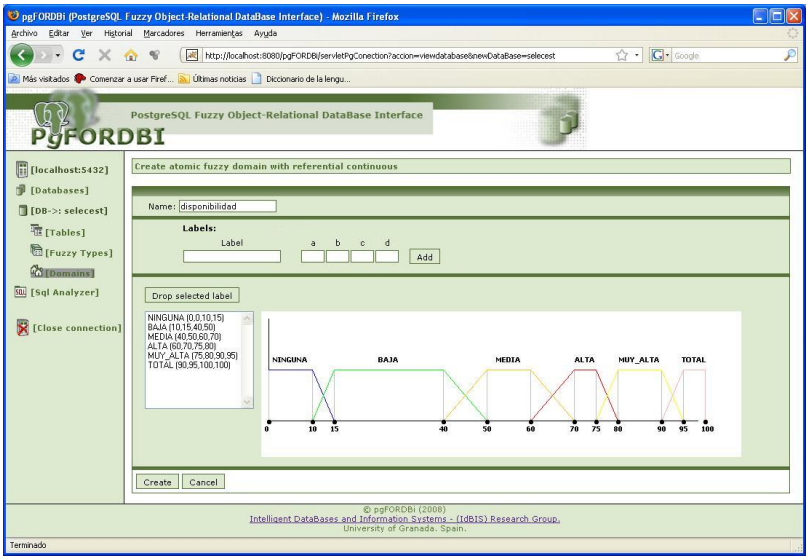


Figura 5.13: Crear dominio disponibilidad.

Los dominios atómicos definidos en un referencial finito sobre valores se emplean en los atributos año y experiencia. En el caso del año se utilizan las siguientes sentencias SQL para su definición.

```
SELECT fuzzy_schema.Create_Domain_DF('anio',
'{1,2,3,4,5}','{INICIALES,MEDIOS,TERMINALES}',
'{{1,0.85,0.5,0.25,0},{0.2,0.8,1,0.8,0.2},{0,0.25,0.65,0.85,1}}');
```

Con la aplicación Web el dominio se crea según las figuras 5.14, 5.15 y 5.16.

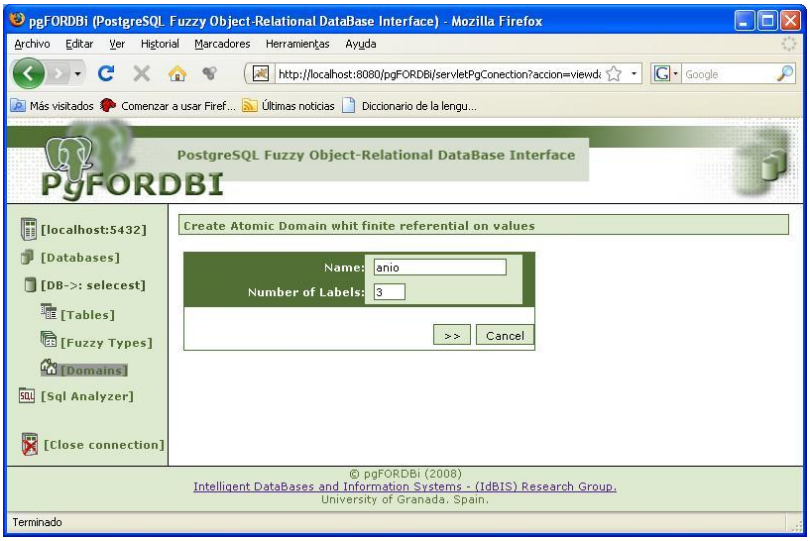


Figura 5.14: Crear dominio año académico 1/3.

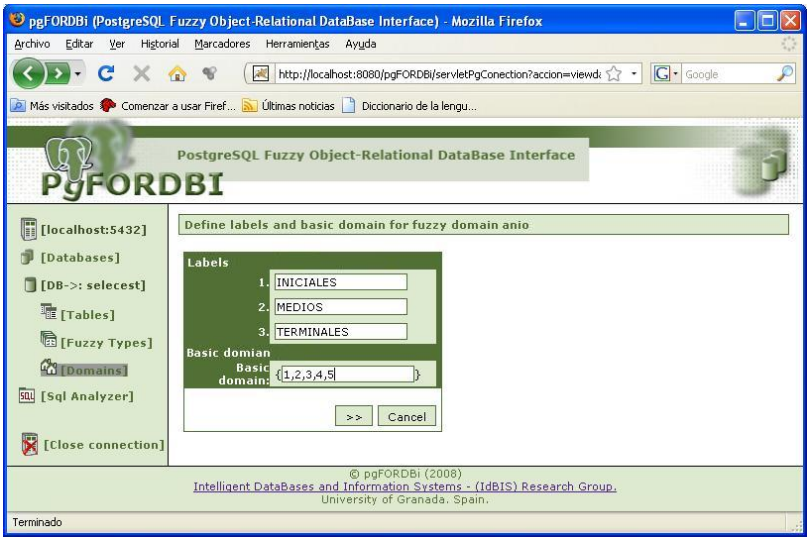


Figura 5.15: Crear dominio año académico 2/3.

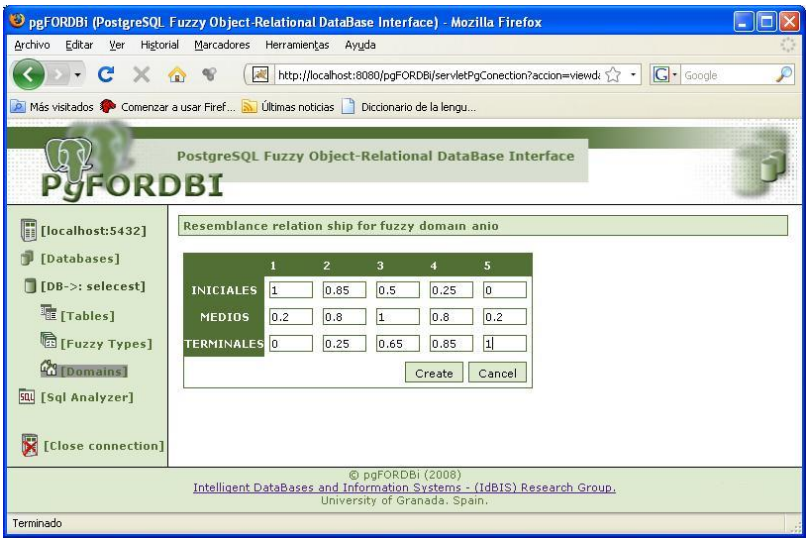


Figura 5.16: Crear dominio año académico 3/3.

Los atributos, idiomas que domina el estudiante y roles que puede desarrollar se definen por medio de dominios conjuntivos con un referencial de valores. La definición del atributo roles se muestra a continuación:

```
SELECT fuzzy_schema.Create_Conjuntive_Extension('rolescollection');
```

Con la aplicación Web se crea el dominio para los idiomas como se muestra en la figura 5.17 y el procedimiento es el mismo.

Un dominio conjuntivo, pero esta vez con un referencial definido sobre objetos, se utiliza para los atributos: lenguajes de programación, personalidad y puestos en un equipo. En el caso de puestos de un equipo se trata de objetos complejos, pero su definición es similar a la que se muestra a continuación, que corresponde a la definición de lenguajes de programación. Notar que inicialmente se define el dominio sin representación semántica asociada para los lenguajes y luego el dominio conjuntivo cuyo nombre es leng_progcollection.

```
SELECT fuzzy_schema.Create_Domain_WR('leng_prog',
'{Java,JavaScript,JSP,C,C++,C#,VisualBasic,VBScript,Pascal,
```

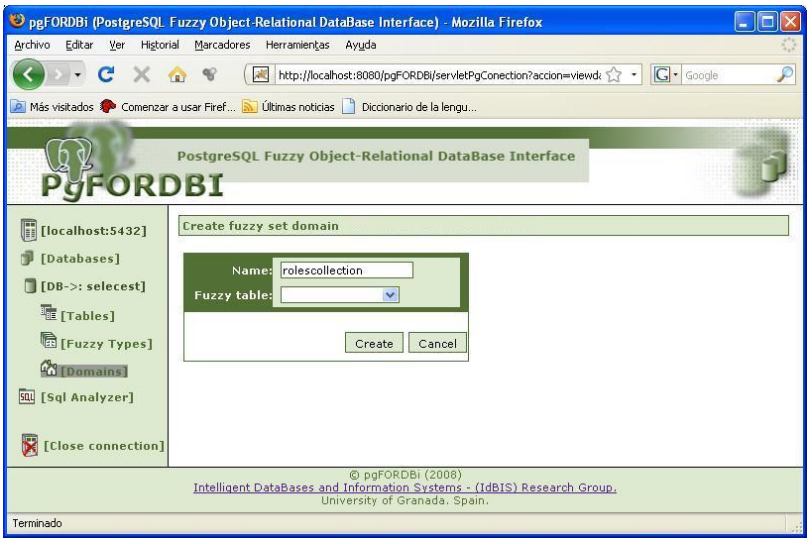


Figura 5.17: Crear dominio idiomas.

```
ObjectPascal,PHP,VBNet,ASPNet}',
'{{1,0.45,0.95,0.6,0.7,0.65,0.1,0.1,0.25,0.3,0.1,0.35,0.23},
{0.45,1,0.5,0.4,0.4,0.4,0.15,0.8,0.1,0.1,0.6,0.4,0.5},
{0.95,0.5,1,0.6,0.65,0.55,0.1,0.2,0.2,0.3,0.65,0.3,0.6},
{0.6,0.4,0.6,1,0.9,0.75,0.15,0.1,0.45,0.37,0.35,0.25,0.2},
{0.7,0.4,0.65,0.9,1,0.8,0.15,0.1,0.37,0.47,0.37,0.3,0.25},
{0.65,0.4,0.55,0.75,0.8,1,0.15,0.1,0.3,0.37,0.37,0.35,0.7},
{0.1,0.15,0.1,0.15,0.15,0.15,1,0.65,0.25,0.2,0.15,0.7,0.5},
{0.1,0.8,0.2,0.1,0.1,0.1,0.65,1,0.1,0.1,0.33,0.3,0.45},
{0.25,0.1,0.2,0.45,0.37,0.3,0.25,0.1,1,0.82,0.15,0.1,0.1},
{0.3,0.1,0.3,0.37,0.47,0.37,0.2,0.1,0.82,1,0.15,0.15,0.1},
{0.1,0.6,0.65,0.35,0.37,0.37,0.15,0.33,0.15,0.15,1,0.3,0.33},
{0.35,0.4,0.3,0.25,0.3,0.35,0.7,0.3,0.1,0.15,0.3,1,0.83},
{0.23,0.5,0.6,0.2,0.25,0.7,0.5,0.45,0.1,0.1,0.33,0.83,1}}');

SELECT fuzzy_schema.Create_Conjuntive_Extension('leng_progcollection',
'tleng_prog');
```

Con la interfaz se muestra la creación del dominio colección de lenguajes (ver Figura 5.18).

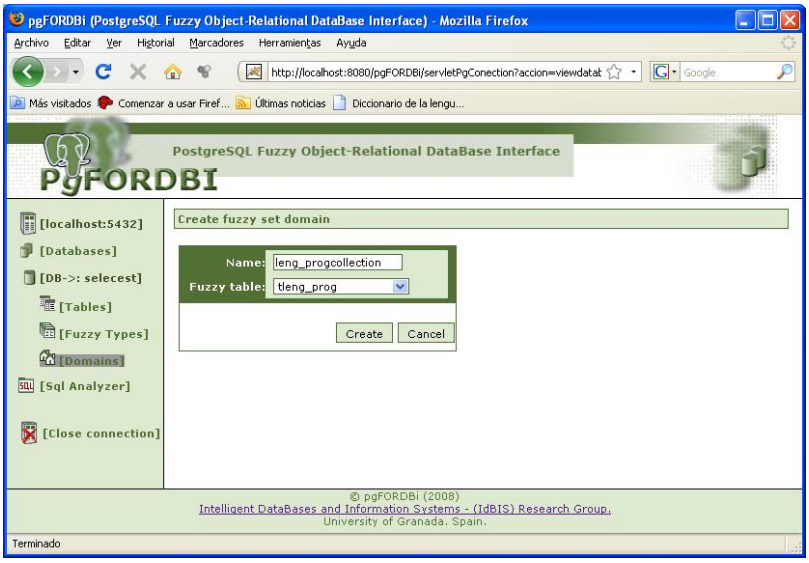


Figura 5.18: Crear dominio personalidad.

Todos los dominios creados para el ejemplo se muestran en esta ventana de la aplicación Web (véase Figura 5.19).

5.4.2. Tipo difuso currículo.

El tipo difuso currículo fue descrito brevemente en apartados anteriores, aquí se muestra la sentencia *SQL* que creará el tipo difuso en la base de datos. Anteriormente fueron definidos cada uno de los dominios utilizados en la definición del tipo difuso.

```
SELECT fuzzy_schema.create_fuzzy_type('curriculo','{
1.0 m_disponibilidad ddisponibilidad
                    REFERENCES fuzzy_schema.tdisponibilidad,
1.0 m_anio danio REFERENCES fuzzy_schema.tanio,
1.0 m_responsabilidad dresponsabilidad
```

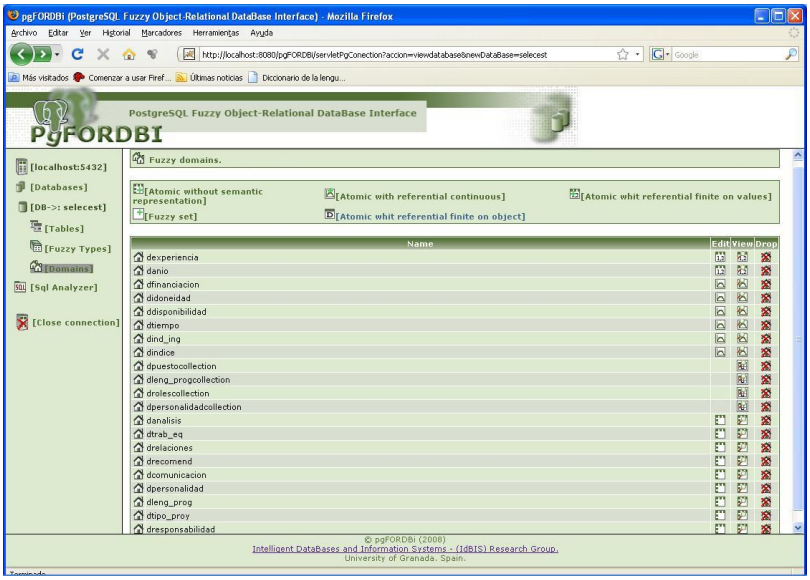


Figura 5.19: Dominios definidos para el ejemplo.

```
                    REFERENCES fuzzy_schema.tresponsabilidad,
1.0 m_personalidad dpersonalidadcollection
                    REFERENCES fuzzy_schema.tpersonalidadcollection,
1.0 m_idioma didiomacollection REFERENCES fuzzy_schema.tidiomacollection,
1.0 m_hab_com dcomunicacion REFERENCES fuzzy_schema.tcomunicacion,
1.0 m_rel_int drelaciones REFERENCES fuzzy_schema.trelaciones,
1.0 m_cap_anal danalisis REFERENCES fuzzy_schema.tanalisis,
1.0 m_idoneidad didoneidad REFERENCES fuzzy_schema.tidoneidad,
0.8 indice_general dindice REFERENCES fuzzy_schema.tindice,
0.8 indice_mat dindice REFERENCES fuzzy_schema.tindice,
0.8 indice_fis dindice REFERENCES fuzzy_schema.tindice,
0.8 indice_idioma dindice REFERENCES fuzzy_schema.tindice,
0.8 indice_sistdig dindice REFERENCES fuzzy_schema.tindice,
0.8 indice_empresa dindice REFERENCES fuzzy_schema.tindice,
0.8 indice_matapli dindice REFERENCES fuzzy_schema.tindice,
0.8 indice_ia dindice REFERENCES fuzzy_schema.tindice,
0.8 indice_prog dindice REFERENCES fuzzy_schema.tindice,
```

```
0.8 indice_infind dindice REFERENCES fuzzy_schema.tindice,
0.8 trab_equipo dtrab_eq REFERENCES fuzzy_schema.ttrab_eq,
0.8 indep dtrab_eq REFERENCES fuzzy_schema.ttrab_eq,
0.8 m_rendimiento didoneidad REFERENCES fuzzy_schema.tidoneidad,
0.6 indice_is dindice REFERENCES fuzzy_schema.tindice,
0.6 roles drolescollection REFERENCES fuzzy_schema.trolescollection,
0.6 lengua_prog dleng_progcollection
                        REFERENCES fuzzy_schema.tleng_progcollection,
0.6 exp_desktop dexperiencia REFERENCES fuzzy_schema.texperiencia,
0.6 exp_web dexperiencia REFERENCES fuzzy_schema.texperiencia,
0.6 exp_mutim dexperiencia REFERENCES fuzzy_schema.texperiencia,
0.6 exp_empresa dexperiencia REFERENCES fuzzy_schema.texperiencia,
0.6 exp_educat dexperiencia REFERENCES fuzzy_schema.texperiencia
}');
```

El tipo difuso se crea por medio de la aplicación Web según las figuras 5.20, 5.21, 5.22 y 5.23.

5.4.3. Tablas con atributos difusos.

Siguiendo el orden asumido corresponde definir las *TWFAs* en el modelo y como ejemplo se seleccionó la *TWFA* Estudiante y de forma similar deberán ser creadas el resto de las *TWFAs*.

```
CREATE TABLE testudiante (
DNI varchar(20) PRIMARY KEY,
nombre varchar(50),
f_nacimiento date,
direccion varchar (50),
municipio varchar (50),
becado varchar(2),
horario varchar(10),
alumno_ayud varchar(2),
m_ind dind_ing REFERENCES fuzzy_schema.tind_ing,
```

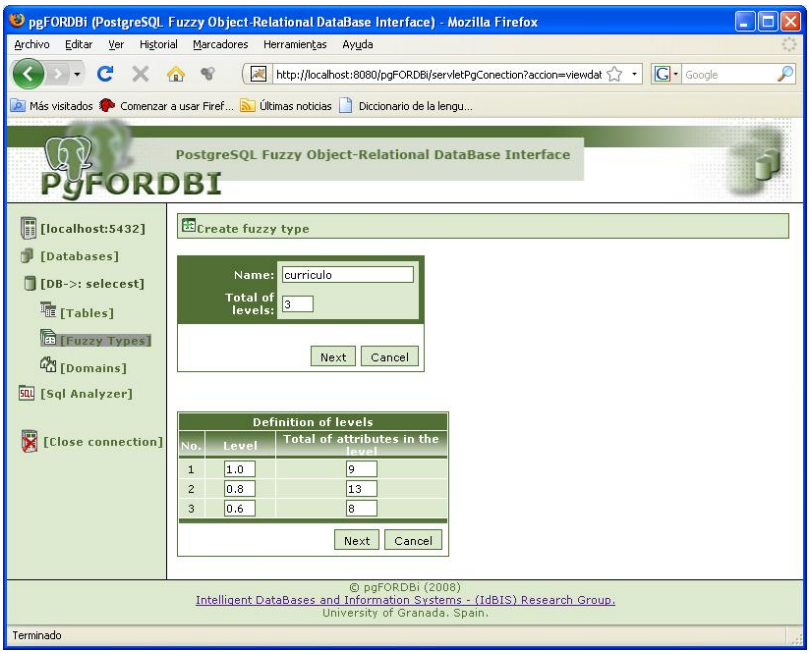


Figura 5.20: Crear tipo difuso currículo 1/4.

```
m_reco drecomend REFERENCES fuzzy_schema.trecomend,
m_clasif didoneidad REFERENCES fuzzy_schema.tidoneidad,
curr dcurriculo
)INHERITS(fuzzy_schema.TFuzzyTable) WITH OIDS;
```

Con la aplicación Web, se crea la *TWFA* estudiante como se muestra en las figuras 5.24 y 5.25.

La definición de los dominios difusos y la creación de las TWFA requeridas para la base de datos del ejemplo, generan toda una estructura que será utilizada para almacenar y manipular los objetos en la base de datos. La estructura resultante para este ejemplo queda representada en la figura del apéndice G. En esta figura quedan sombreadas las *TWFA* del usuario, definidas en el esquema *public* de la base de datos. El resto de las *TWFAs* se definen en el esquema *fuzzy_schema*.

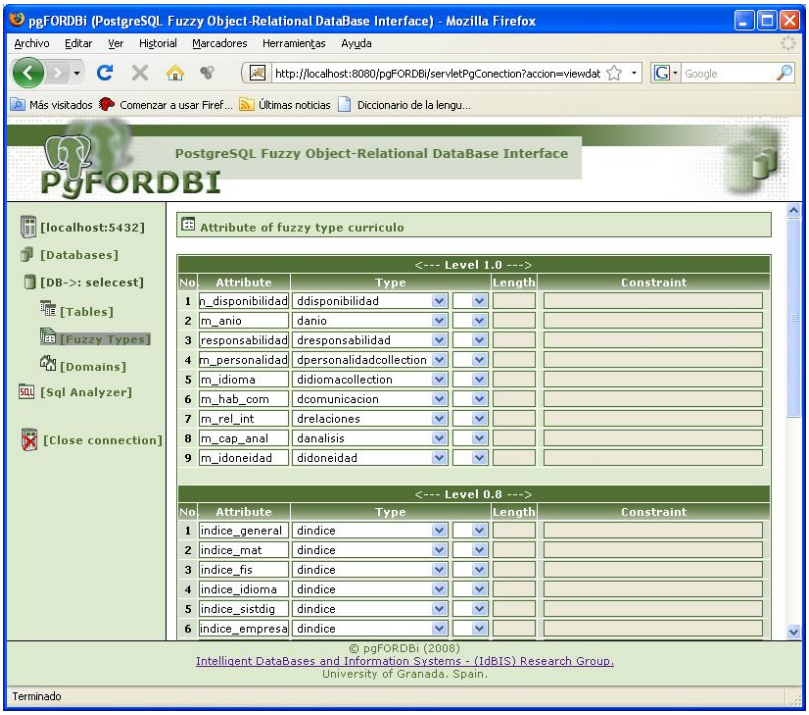


Figura 5.21: Crear tipo difuso currículo 2/4.

5.4.4. Definir atributos inferidos.

Como se mostró al describir los atributos de cada *TWFA*, algunos de ellos se definen como atributos inferidos. Se ejemplifica la creación de un atributo inferido utilizando el atributo idoneidad para el tipo difuso currículo. Notar que el atributo inferido se crea sobre un tipo difuso; pero este mismo procedimiento es utilizado para crear atributos inferidos en una *TWFA*. La sentencia *SQL* a escribir para definir el atributo inferido idoneidad en el tipo difusos currículo es la siguiente:

```
SELECT fuzzy_schema.new_att_derive('curriculo','m_idoneidad','{
m_disponibilidad NFGT MUY_ALTA AND m_anio FEQ MEDIOS AND
m_responsabilidad FEQ MUCHA AND m_rel_int FEQ ACEPTABLES AND
m_cap_anal FEQ ALTA THEN m_idoneidad IS MUY_ALTA,
m_disponibilidad FEQ BAJA AND m_anio FEQ TERMINALES
THEN m_idoneidad IS BAJA,
m_disponibilidad FEQ MEDIA OR m_disponibilidad FEQ ALTA AND
 m_responsabilidad FEQ MEDIA AND m_cap_anal FEQ ALTA
```

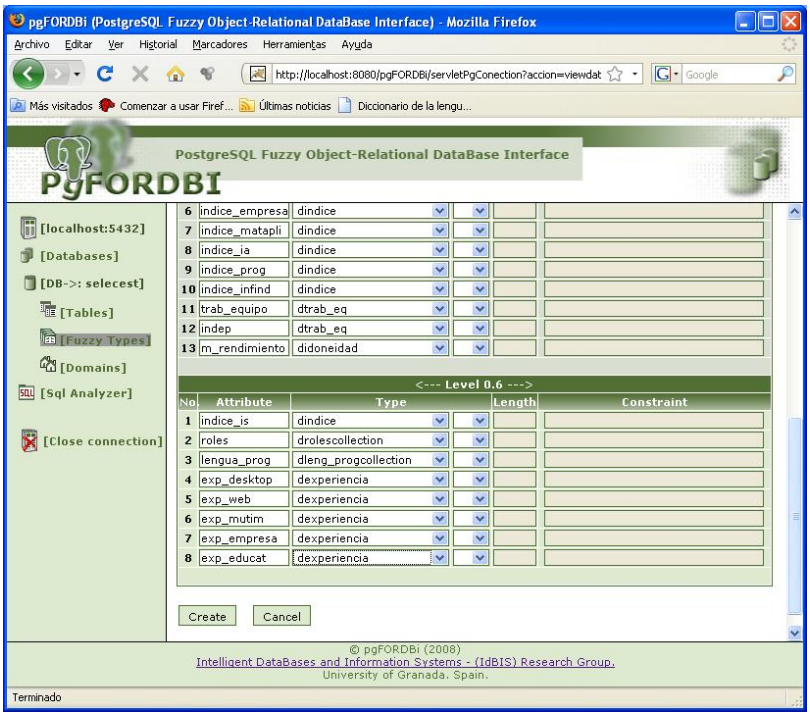


Figura 5.22: Crear tipo difuso currículo 3/4.

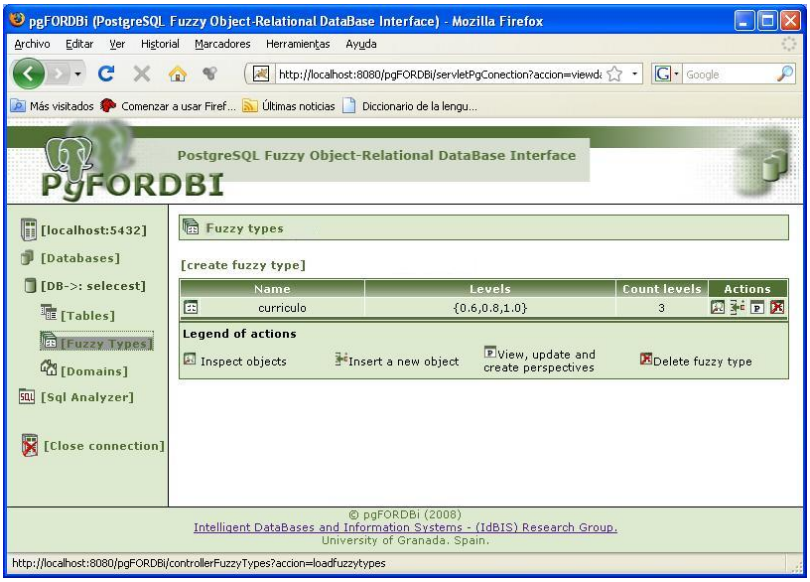


Figura 5.23: Crear tipo difuso currículo 4/4.

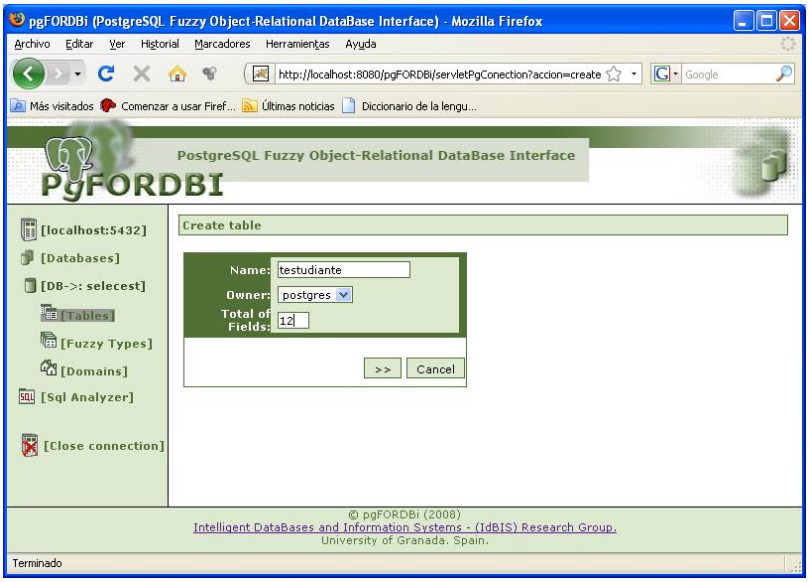


Figura 5.24: Crear *TWFA* estudiante 1/2.

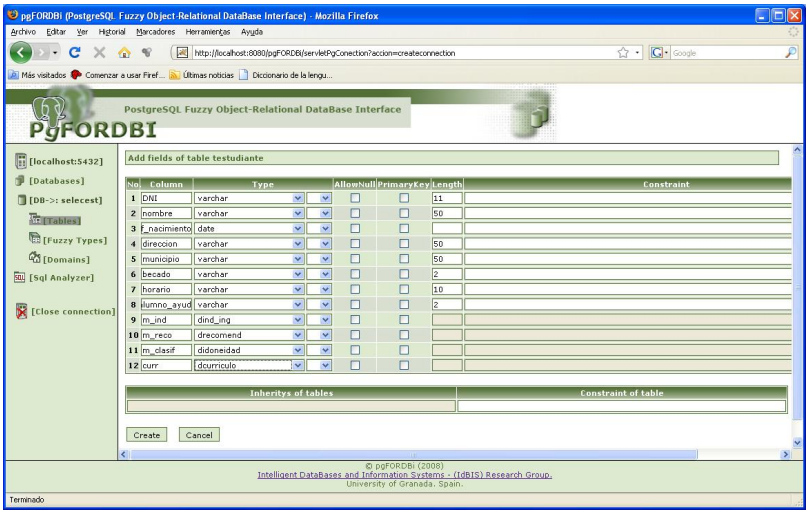


Figura 5.25: Crear *TWFA* estudiante 2/2.

```
 THEN m_idoneidad IS MEDIA,
m_disponibilidad FGEQ MUY_ALTA AND m_anio FEQ INICIALES AND
m_responsabilidad FEQ MUCHA AND m_cap_anal FEQ MUY_ALTA
THEN m_idoneidad IS ALTA,
m_anio FEQ 5 THEN m_idoneidad IS BAJA
}','didoneidad');
```

Puede utilizarse la aplicación Web propuesta, de la misma forma que con el resto de las operaciones, para definir un atributo inferido. Como ejemplo se utilizará también el atributo *idoneidad* del tipo difuso *currículo*. El atributo inferido queda definido según la figura 5.26. La figura 5.27, muestra como se inserta la primera regla del sistema de reglas difusas asociada al atributo inferido *idoneidad*, de forma similar se hace para el resto de las reglas hasta obtener el resultado mostrado en la figura 5.26.

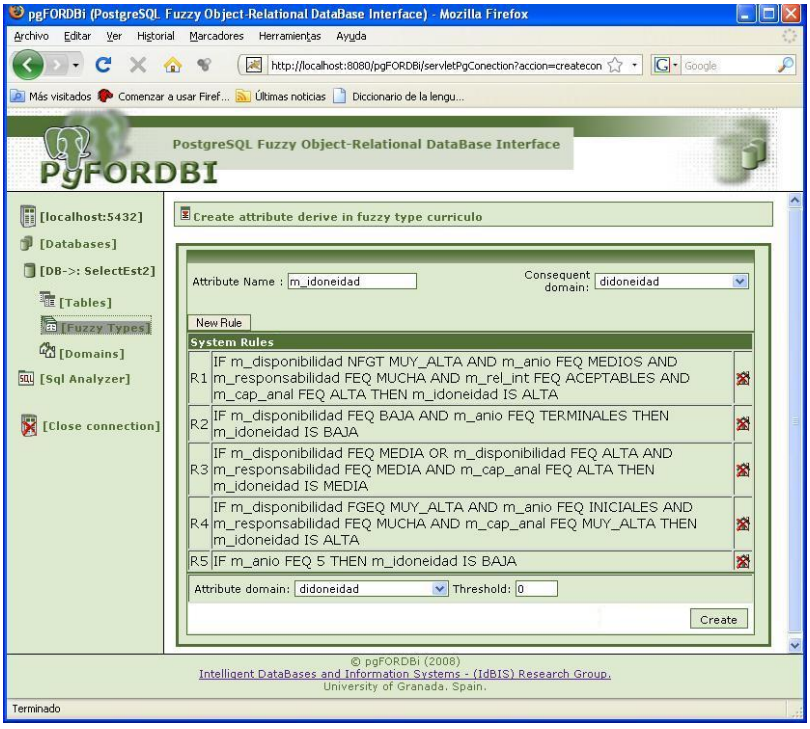


Figura 5.26: Crear atributo inferido idoneidad 1/2.

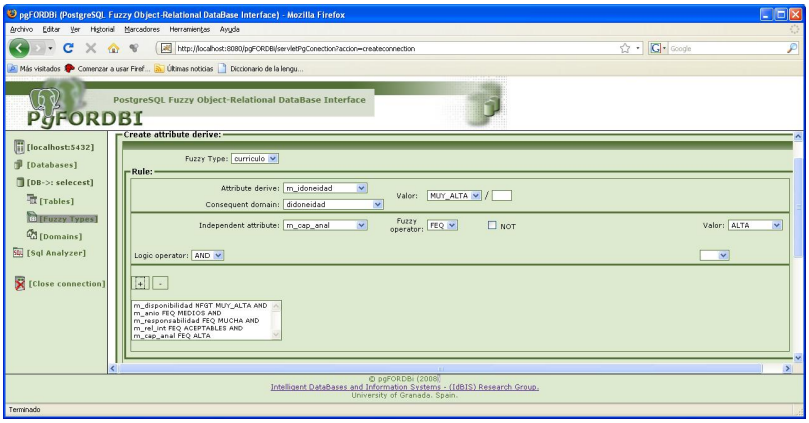


Figura 5.27: Crear atributo inferido idoneidad 2/2.

En este ejemplo el sistema automáticamente creará el atributo inferido idoneidad para el nivel 1.0 del tipo difusos. A partir de esto se puede crear el atributo inferido *rendimiento* que en este caso se define para el nivel 0.8 y tiene en cuenta el valor del atributo *idoneidad* previamente definido. Al dispararse las reglas se infiere inicialmente el atributo *idoneidad* y luego el atributo *rendimiento*.

La *TWFA estudiante* también tiene un atributo inferido denominado *clasificación* que se define de forma similar a lo mostrado hasta este punto. El código *SQL* necesario en este caso es el siguiente:

```
SELECT fuzzy_schema.new_att_derive('testudiante','m_clasif','{
m_ind FLEQ REGULAR THEN m_clasif IS BAJA,
m_ind FGT BUENO AND (m_reco IS BUENA OR m_reco IS MUY_BUENA)
THEN m_clasif IS MUY_ALTA,
m_ind FGT BUENO AND m_reco IS REGULAR THEN m_clasif IS ALTA,
m_ind FGT BUENO AND m_reco IS MALA THEN m_clasif IS MEDIA
}','didoneidad');
```

Si se quiere utilizar la aplicación WEB para definir el atributo *clasificación* se procede de forma similar a lo mostrado en las figuras 5.26 y 5.27. El mismo proceso se utiliza para definir el atributo inferido *tipo de proyecto* de la *TWFA proyecto*.

5.4.5. Definir perspectivas de comparación.

Las perspectivas de comparación, para una *TWFA* o para un tipo difuso, definen un patrón de comparación donde se especifican los operadores que se van a utilizar y el peso que tendrá cada atributo en la comparación de los objetos (ver Apartado anterior).

En esta base de datos se da la siguiente situación. Se desea realizar un proyecto en el cual uno de sus temas se elaborará una aplicación Web para el control de la contabilidad en una empresa. La aplicación será realizada utilizando como lenguaje de programación Java y JSP. Respondiendo a esta aplicación se definió un tema donde se especifica que características deben tener los estudiantes que participen en dicho tema (ver columna uno de la Tabla 5.10). Se debe recordar que las características de un tema están representadas por medio del tipo difuso currículo.

Se desean cubrir tres posiciones, la de programador, jefe de grupo, analista y probador. Cada una de estas posiciones exige un cumplimiento diferente de las características especificadas para el tema, por lo tanto se decidió definir cuatro perspectivas de comparación (ver columnas dos, tres, cuatro y cinco de la Tabla 5.10).

Puede notarse que al momento de definir la perspectiva los atributos que no interesan que se tengan en cuenta se le asigna como peso 0 y para los atributos más importantes el peso es 1. Para definir la perspectiva *programador* en el tipo difuso *currículo* se debe escribir la siguiente sentencia *SQL* con la cual se crea un esquema de comparación para el tipo difuso.

```
SELECT fuzzy_schema.comparison_profile('curriculo','programador',
'{m_disponibilidad,m_anio,m_responsabilidad,m_personalidad,m_idioma,
m_hab_com,m_rel_int,m_cap_anal,m_idoneidad,indice_general,indice_mat,
indice_fis,indice_idioma,indice_sistdig,indice_empresa,indice_matapli,
indice_ia,indice_prog,indice_infind,trab_equipo,indep,m_rendimiento,
indice_is,roles,lengua_prog,exp_desktop,exp_web,exp_mutim,exp_empresa,
exp_educat}',
'{1,1,0.8,0,0.3,0.7,0.6,0.9,1,0.7,0.8,0,0,0,0.8,0,0,1,0,0.85,0.8,0.7,
0.4,0,1,0,1,0,0.8,0}',
```

Cuadro 5.10: Perspectivas de comparación para el tipo difuso currículo

	Tema	**Programador**		**Jefe Grupo**		**Analista**		**Probador**	
Atributo	**Valor**	**Oper**	**Peso**	**Oper**	**Peso**	**Oper**	**Peso**	**Oper**	**Peso**
disponibilidad	ALTA	FGT	1	FGT	1	FGEQ	0.9	FGT	0.8
año	MEDIOS	FEQ	1	FEQ	0.7	FEQ	0.8	FEQ	1
responsabilidad	MEDIA	FEQ	0.8	FEQ	1	FEQ	0.8	FEQ	1
personalidad	honesto/0.65	FEQ	0						
idioma	1.0/Espanol,0.3/Ingles	GIN	0.3	GIN	0.3	GIN	0.3	GIN	0.3
comunicación	ALTA	FEQ	0.7	FEQ	1	FEQ	1	FEQ	0.9
relaciones personales	BUENAS	FEQ	0.6	FEQ	0.9	FEQ	0.9	FEQ	0.8
capacidad de analisis	ALTA	FEQ	0.9	FEQ	1	FEQ	1	FEQ	0.3
idoneidad	ALTA	FGEQ	1	FGT	1	FEQ	0.7	FEQ	0.5
indice general	MEDIO	FGEQ	0.7	FGEQ	0.7	FGEQ	0.7	FGEQ	0.5
indice matatemática	MEDIO	FGEQ	0.8	FGEQ	0.5	FGEQ	0.5	FGEQ	0.4
indice física	BAJO	FEQ	0	FEQ	0	FEQ	0	FEQ	0
indice idioma	BAJO	FEQ	0	FEQ	0	FEQ	0	FEQ	0
indice sistemas digitales	BAJO	FEQ	0	FEQ	0	FEQ	0	FEQ	0
indice ciencias empresariales	ALTO	FGEQ	0.8	FGT	1	FGT	1	FGEQ	1
indice matemática aplicada	MUY_ALTO	FEQ	0	FEQ	0.8	FEQ	0.8	FEQ	0
indice inteligencia artíficial	BAJO	FEQ	0	FEQ	0	FEQ	0	FEQ	0
indice programación	ALTO	FGT	1	FGEQ	0.85	FGEQ	0.4	FLT	0.8
indice informática industrial	BAJO	FEQ	0	FEQ	0	FEQ	0	FEQ	0
trabajo en equipo	MEDIA	FEQ	0.85	FEQ	1	FEQ	1	FEQ	0.85
trabajo independiente	ALTA	FEQ	0.8	FEQ	0.7	FEQ	0	FEQ	0.9
rendimiento	MEDIA	FGEQ	0.7	FGT	0.7	FGT	0.7	FLEQ	0.3
indice ingenieria de software	ALTO	FLEQ	0.4	FGEQ	0.9	FGT	1	FLEQ	0.4
roles	programador/0.7	FEQ	0	FEQ	0	FEQ	0	FEQ	0
lenguajes de programación	0.8/JAVA,0.6/JSP	GIS	1	GIN	1	GIN	0.8	FEQ	0
experiencia apliaciones desktop	MUY_POCA	FEQ	0	FEQ	0	FEQ	0	FEQ	0
experiencia apliaciones web	MEDIA	FEQ	1	FEQ	1	FEQ	0.8	FEQ	0
experiencia apliaciones mutimedia	MUY_POCA	FEQ	0	FEQ	0	FEQ	0	FEQ	0
experiencia apliaciones empresariales	MEDIA	FEQ	0.8	FEQ	1	FEQ	1	FEQ	0.6
experiencia apliaciones educativas	MUY_POCA	FEQ	0	FEQ	0	FEQ	0	FEQ	0

```
'{FGT,FEQ,FEQ,FEQ,GIN,FEQ,FEQ,FEQ,FGEQ,FGEQ,FGEQ,FEQ,FEQ,FEQ,FGEQ,
FEQ,FEQ,FGT,FEQ,FEQ,FEQ,FGEQ,FLEQ,FEQ,GIS,FEQ,FEQ,FEQ,FEQ,FEQ}',0.4,-1.0);
```

En la definición de la perspectiva programador se definió utilizar 0.4 como número de *orness* y se tendrá una posición indiferente al comparar dos objetos de un tipo difuso pertenecientes a niveles diferentes. En la aplicación web la definición de esta perspectiva se muestra en las figuras 5.28 y 5.29

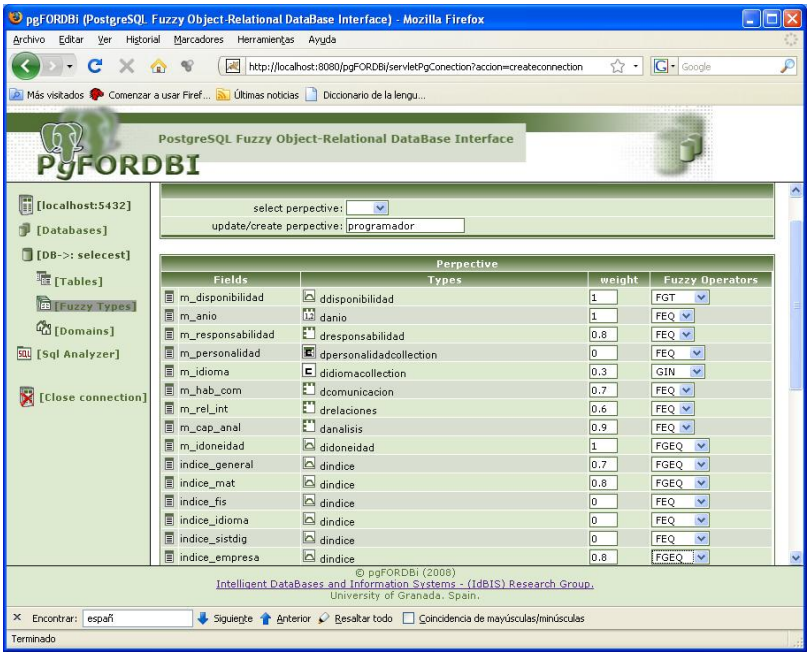


Figura 5.28: Perspectiva programador 1/2.

En la tabla 5.10, la columna tema se refiere a valores que va a tomar al atributo currículo en un objeto de la tabla tema. Esta información luego será utilizada para hacer consultas en la base de datos.

Por otro lado se desea seleccionar un estudiante como jefe del grupo de desarrollo a partir de un patrón que se exige. En el patrón se tiene en cuenta el currículo del estudiante que debe adaptarse al tema *sistemacontab*; pero con un perfil de programador y que además tenga un índice de ingreso posiblemente mayor difuso que BUENO y preferiblemente esté becado. Para obtener este resultado se define la siguiente perspectiva de comparación para la *TWFA* testudiante.

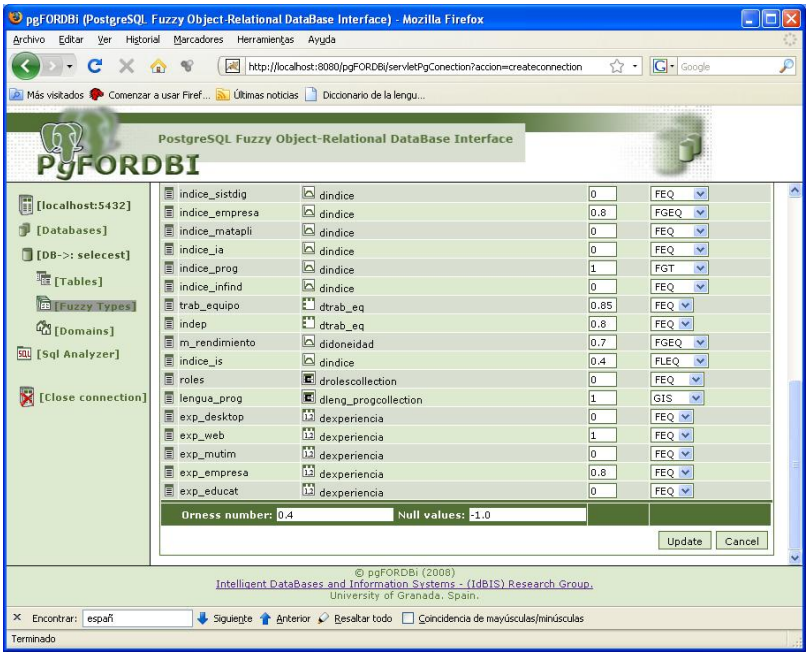


Figura 5.29: Perspectiva programador 2/2.

```
SELECT fuzzy_schema.comparison_profile('testudiante','jprogram',
'{DNI,nombre,f_nacimiento,direccion,municipio,becado,horario,alumno_ayud,
m_ind,m_reco,m_clasif,curr}', '{0,0,0,0,0,0.85,0,0,0.4,0,1,1}',
'{FEQ,FEQ,FEQ,FEQ,FEQ,FEQ,FEQ,FEQ,FGT,FEQ,FEQ,programador}', 0.8);
```

Notar como en este caso se utiliza la perspectiva *programador* previamente creada para el tipo difuso *currículo*. Una consulta a la base de datos utilizando esta perspectiva se muestra en el apartado de consultas en este capítulo.

5.4.6. Crear objetos.

Por último, se está en condiciones de crear objetos en la base de datos. Se mostrará la creación de dos objetos *testudiante*, uno perteneciente al primer año de la carrera y otro perteneciente al cuarto año. Las sentencias a escribir son las siguientes:

Estudiante de primer año de la carrera.

```
INSERT INTO testudiante VALUES( 87061723164,
```

```
'Michel Batista',
'06/17/1986',
'C.16-A N.11 E/5 Y 7',
'Freyre',
'SI',
'Mañana',
'NO',
set_ind_ing('94.44'),
set_recomend('BUENA'),
set_idoneidad('1'),
set_curriculo (
    set_disponibilidad('ALTA'),set_anio('1'),
    set_responsabilidad('MEDIA'),
    set_personalidadcollection('{HONESTO,INDEPENDIENTE,TRABAJADOR,
        CREATIVO}','{0.6,0.5,0.8,0.7}'),
    set_idiomacollection('{español,ingles}','{1.0,0.8}'),
    set_comunicacion('ALTA'),
    set_relaciones('ACEPTABLES'),
    set_analisis('MEDIA'),
    set_idoneidad('1')
    )
);
```

Estudiante del cuarto año de la carrera.

```
INSERT INTO testudiante VALUES(
80050123543,
'Damian Calderon',
'01/05/1980',
'EDIF.39 ESC.D APTO.9 RP.ABEL S',
'Holguin',
'NO',
'Tarde',
'SI',
set_ind_ing('92.21'),
```

```
set_recomend('REGULAR'),
set_idoneidad('1'),
set_curriculo (
    set_disponibilidad('ALTA'),
    set_anio('4''),
    set_responsabilidad('MEDIA'),
    set_personalidadcollection('{HONESTO,INDEPENDIENTE,TRABAJADOR,
        CREATIVO}','{0.7,0.55,0.8,0.6}'),
    set_idiomacollection('{español,ingles}','{1.0,0.9}'),
    set_comunicacion('MEDIA'),
    set_relaciones('ACEPTABLES'),
    set_analisis('MEDIA'),
    set_idoneidad('1'),
    set_indice('3.95'),
    set_indice('3.59'),
    set_indice('3.99'),
    set_indice('3.49'),
    set_indice('4.50'),
    set_indice('3.70'),
    set_indice('4.76'),
    set_indice('3.28'),
    set_indice('3.48'),
    set_indice('4.32'),
    set_trab_eq('ALTA'),
    set_trab_eq('MEDIA'),
    set_idoneidad('1'),
    set_indice('4.41'),
    set_rolescollection('{Programador,Diseñador,Analista,Probador}',
        '{0.5,0.7,0.7,0.6}'),
    set_leng_progcollection('{Java,VisualBasic,C++}','{0.7,0.2,0.9}'),
    set_experiencia('1'),
    set_experiencia('2'),
    set_experiencia('1'),
    set_experiencia('1'),
```

```
        set_experiencia('2')'
        )
);
```

En la aplicación web se inserta un nuevo objeto *testudiante* según lo mostrado en la figura 5.30.

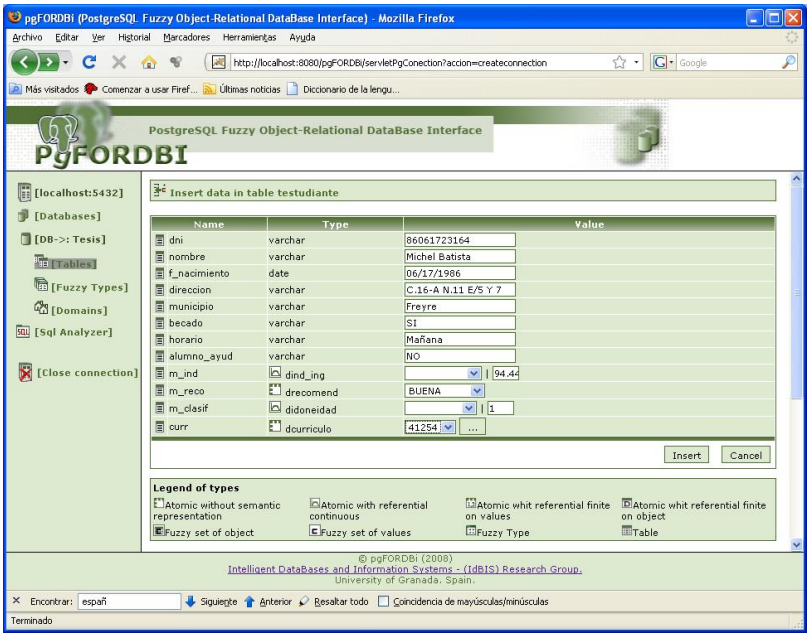


Figura 5.30: Crear un objeto estudiante.

Una muestra de algunos de los estudiantes insertados en la base de datos se muestra en el apéndice H.1. El apéndice H.3 muestra la creación de un objeto tipo *tpuesto*. Estos objetos serán utilizados en las consultas que se realizarán sobre la base de datos.

5.5. Consultas ejemplo.

Luego de introducir los objetos en la base de datos lo más importante es obtener información de esos objetos. Por medio de consultas *SQL* es posible consultar la base de datos teniendo en cuenta la información difusa almacenada en ella. Se mostrarán a continuación algunos de los tipos de consultas que pueden realizarse sobre la base de datos.

1. Obtener el grado de parecido entre el tiempo de duración del proyecto CITMA y la del proyecto Che12H.

```
SELECT p.cod_proy, p1.cod_proy, FEQ(p.duracion, p1.duracion) FROM
tproyecto as p, tproyecto as p1 WHERE p.cod_proy='Che12H' AND
p1.cod_proy='CITMA'
```

   El resultado de estas consultas se muestra en la tabla 5.11.

Cuadro 5.11: Resultado de la consulta

cod_proy	cod_proy	feq
Che12H	CITMA	0.89

2. Listar los nombre de aquellos estudiantes que ingresaron con un índice EXCELENTE en un grado igual o mayor de 0.8.

```
SELECT e.nombre, FEQ('EXCELENTE',e.m_ind) AS po FROM testudiante as
e WHERE FEQ('EXCELENTE',e.m_ind) >=0.8;

o

SELECT e.nombre, FGEQ (e.m_ind,'EXCELENTE') AS po FROM testudiante
as e WHERE FGEQ(e.m_ind,'EXCELENTE') >=0.8;
```

   El resultado de estas consultas se muestra en la tabla 5.12.

Cuadro 5.12: Resultado de la consulta

nombre	degree
Michel Batista	0.89
Miguel Aguilera	1.00
Rafale Barata	1.00
Odalis Pita	0.92
Anniet Llorente	0.88

3. Listar los nombre de aquellos estudiantes que ingresaron con un índice de alrededor de 95 puntos +/-1.5.

```
SELECT e.nombre, FEQ(e.m_ind, f(95,1.5)) FROM testudiante as e WHERE
FEQ(e.m_ind, f(95,1.5)) <> 0.0
```

El resultado de estas consultas se muestra en la tabla 5.12.

Cuadro 5.13: Resultado de la consulta

nombre	feq
Michel Batista	0.63
Rafale Barata	0.12
Odalis Pita	0.72
Anniet Llorente	0.59

4. Obtener los estudiantes que su currículo es similar al solicitado para trabajar en el tema *sistcontbweb*.

```
SELECT e.dni, t.cod_tema, FEQ(e.curr,t.curr) FROM testudiante as e,
ttema as t WHERE t.cod_tema='sistcontbweb'
```

El resultado de esta consulta se muestra en la tabla 5.14.

Cuadro 5.14: Resultado de la consulta

dni	cod_tema	feq
86061723164	sistcontbweb	0.00
87123026023	sistcontbweb	0.00
86052121698	sistcontbweb	0.60
86110221788	sistcontbweb	0.52
85072422382	sistcontbweb	0.20
85120722363	sistcontbweb	0.20
80050123543	sistcontbweb	0.20
82090326066	sistcontbweb	0.20
84012425135	sistcontbweb	0.20
83103119402	sistcontbweb	0.20
85102722975	sistcontbweb	0.20
86011021716	sistcontbweb	0.20
85050521681	sistcontbweb	0.20

5. Obtener los estudiantes cuyo currículo se adapta más al solicitado para trabajar en el tema *sistcontbweb* como programadores. En este caso se utiliza la perspectiva de

comparación previamente creada para el tipo difuso currículo.

```
SELECT e.dni, t.cod_tema, FEQ(e.curr,t.curr,'programador') FROM
testudiante as e, ttema as t WHERE t.cod_tema='sistcontbweb'
```

El resultado de esta consulta se muestra en la tabla 5.15.

Cuadro 5.15: Resultado de la consulta

dni	cod_tema	feq
86061723164	sistcontbweb	0.00
87123026023	sistcontbweb	0.00
86052121698	sistcontbweb	0.70
86110221788	sistcontbweb	0.82
85072422382	sistcontbweb	0.52
85120722363	sistcontbweb	0.40
80050123543	sistcontbweb	0.40
82090326066	sistcontbweb	0.32
84012425135	sistcontbweb	0.34
83103119402	sistcontbweb	0.36
85102722975	sistcontbweb	0.40
86011021716	sistcontbweb	0.40
85050521681	sistcontbweb	0.40

6. Obtener el grado en el que cada estudiante puede ocupar el puesto de "jefe del grupo de desarrollo". El patrón a utilizar en la consulta se define en el apéndice H.2.

```
SELECT e.nombre, FEQ(e.oid,e1.oid,'jprogram') FROM testudiante as e,
testudiante as e1 where e1.dni='00000000001'
```

El resultado de estas consultas se muestra en la tabla 5.16.

Utilizando la aplicación web el usuario puede auxiliarse del *SQL Analyzer* incorporado a la misma, para confeccionar sus consultas (ver Figura 5.31 y Figura 5.32).

Cuadro 5.16: Resultado de la consulta

nombre	feq
Michel Batista	0.68
Carlos Ferra	0.80
Marlen Machado	0.56
Juan Vinaldell	0.66
Yurieski Arcaya	0.68
Miguel Aguilera	0.68
Damian Calderon	0.40
Rafale Barata	0.32
Odalis Pita	0.68
Yandi Fernandez	0.29
Anniet Llorente	0.68
Susel Tamayo	0.80
Pedro Camacho	0.32

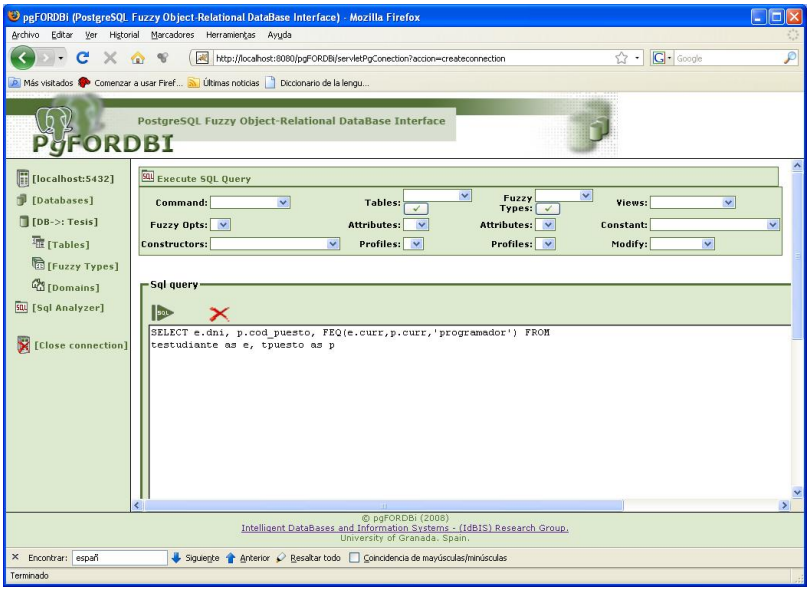


Figura 5.31: Ejemplo de consulta utilizando la aplicación Web 1/2.

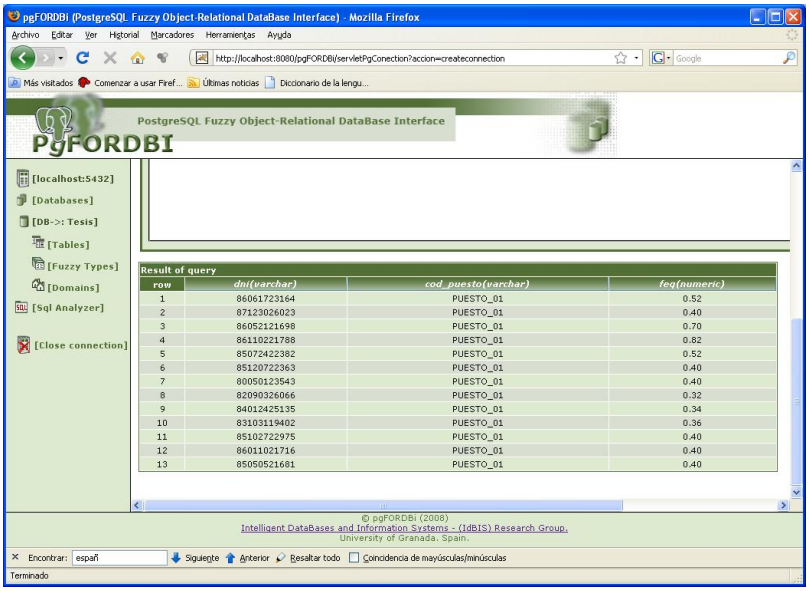


Figura 5.32: Ejemplo de consulta utilizando la aplicación Web 1/2.

Capítulo 6

Conclusiones y Trabajos Futuros

Este capítulo presentará los principales resultados y conclusiones obtenidas durante la investigación y dará una pauta hacia posibles trabajos futuros que pueden realizarse en esta área de estudio.

6.1. Conclusiones

El objetivo general de esta investigación fue desarrollar un modelo de bases de datos difusas orientas a objetos y su implementación en un gestor de bases de datos distribuido como software libre y con características objeto-relacional. En tal sentido se propuso dicho modelo y se implementó utilizando *PostgreSQL*. Los detalles de los resultados obtenidos se reflejan a continuación.

Se logró desarrollar un modelo teórico de bases de datos difusas a partir de extensiones realizadas al modelo de bases de datos difusas orientadas a objetos de Marín y otros. Las extensiones al modelo se realizaron en el nivel de atributo y en el nivel de definición.

En el nivel de atributos se trabajó con las diferentes formas de representar y manipular la imperfección en los atributos logrando los siguientes resultados:

1. Se proponen cuatro formas de representar la imprecisión en los valores de un atributo. Los cuatro dominios son:

    - Cuando el valor por su naturaleza no tiene una representación semántica asociada a él, como son los ejemplo de la calidad de un producto, rasgos de la personalidad de una persona, grado de responsabilidad, nivel de comunicación

entre otros. Para estos valores se define un dominio atómico sin representación semántica.

- Situaciones en las que se tiene un referencial continuo bien definido y un conjunto de etiquetas lingüísticas que representan la imprecisión del valor. Esta etiquetas están definidas sobre ese referencial por medio de funciones. Ejemplo de este tipo de dominio son la altura de una persona, el índice académico, la financiación de un proyecto, etc . Para estos casos se trabaja con dominio atómico sobre un referencial continuo.
- Existe un referencial bien definido para el dominio pero este referencial es finito y sobre él se definen varias etiquetas lingüísticas por medio de distribuciones de posibilidad. En este caso se dispone de un dominio atómico con referencial finito. Los atributos dureza de un material, planta, experiencia en el desarrollo de aplicaciones, entre otros, son ejemplos de este tipo de dominio.
- Y por último, los caso en los cuales el valor de un atributo está representado por medio de un subconjunto difuso de valores. Para estos casos se define un dominio conjuntivo sobre un referencial de valores. Ejemplo de este dominio son los idiomas que domina una estudiante.

2. Para los dominios sobre referencial de valores se definen formas de comparar sus valores por medio del cálculo de la semejanza entre ellos. Y en algunos casos se definen operadores difusos para la comparación, los detalles de estos operadores son:

   - Para el dominio atómico con referencial continuo se proponen operadores de posibilidad que permiten determinar entre dos valores los estados de mayor difuso, mayor e igual difuso, menos difuso, menor e igual difuso, mucho mayor difuso y mucho menor difuso.
   - Para el dominio conjuntivo se proponen como operadores difusos diferentes formas de igualdad a partir de combinar la inclusión entre dos subconjuntos. De esta forma se pueden combinar utilizando una t-norma, una t-conorma o el promedio.

3. En el modelo propuesto es posible definir y manipular objetos descritos por atributos difusos permitiendo la comparación entre dos objetos. El trabajo con objetos en el modelo se caracteriza por:

- Un objeto descrito por atributos difusos tiene al menos un atributo definido sobre un dominio difuso.
- La comparación de dos objetos pasa por comparar cada uno de sus atributos y luego hacer una agregación con los resultados obtenidos.
- La comparación entre dos objetos puede estar condicionada por la utilización de perspectivas de comparación que permiten definir el comportamiento de la comparación entre objetos según los requerimientos del usuario. Esto incluye, definir el peso que tendrá cada atributo en la comparación y los operadores que se utilizarán en la misma.

4. El modelo permite de la misma forma el trabajo con objetos complejos. Un objeto es complejo si al menos uno de sus atributos es un objeto. Estos objetos complejos tienen el mismo comportamiento que los otros objetos pero se deben recalcar los siguientes elementos:

   - Al comparar dos objetos complejos ocurre un proceso de recursividad. Esto viene dado porque al comparar dos objetos se compara cada atributo y si uno de estos atributo es otro objeto se repite el proceso de comparar objetos.
   - La perspectivas de comparación definidas para los objetos complejos tienen también recursividad. En este caso se debe especificar en las perspectivas de un objeto complejo con qué perspectiva quiere que se compare un atributo definido como objeto.

5. Los dominios difusos propuestos para un referencial de valores fueron extendidos para que soportaran un referencial de objetos y de objetos complejos. En este sentido fueron extendidos dos dominios:

   - Atómico con referencial finito sobre objetos. La extensión de este dominio a un referencial de objeto provocó la necesidad de definir la forma de determinar la igualdad entre dos valores del dominio. Ejemplo de este dominio es el atributo autor de un artículo. En este caso la semántica con la cual se interpretan los subconjuntos difusos es disyuntiva.
   - Conjuntivo sobre un referencial de objetos. Este caso el cálculo de la igualdad entre dos subconjuntos parte de tener en cuenta la semejanza existente entre los elementos del conjunto. De esta forma se utiliza el grado de inclusión guiado

por semejanza para determinar la inclusión. La inclusión puede combinarse por medio de una t-norma, una t-conorma o el promedio. Además, se definieron operadores no simétricos basados en la inclusión de un subconjunto en otro, pero en este caso teniendo en cuenta la semejanza. Este proceso dio como resultado dos operadores difusos. Ejemplos de este dominio son los investigadores de una línea, el conjunto de rasgos de la personalidad y otros. Para este tipo de dominio, en el caso de que su referencial sean objetos complejos es posible definir perspectivas de comparación. En estas perspectivas se especifica con qué perspectiva quiere que se comparen los objetos que forman el subconjunto.

6. Una de las extensiones propuestas en esta investigación consiste en utilizar atributos inferidos que no son más que atributos difusos que toman su valor a partir del valor de otros atributos por medio de la inferencia resultante de un sistema de reglas. Las características de estos atributos son las siguientes:

   - El uso de reglas difusas definidas teniendo en cuenta la filosofía de Mamdani [Jag95]. Las reglas propuestas por Mandani fueron extendidas para soportar el uso de operadores, quedando definidas de la siguiente forma:
     if x_i *oper* A_1 *and ... and* x_{nx} *oper* A_n *then y is* B_1 Donde: *oper* es uno de los operadores definidos en el modelo.
   - La variable en el consecuente puede estar definida sobre cualquiera de los dominios difusos soportados por el modelo o sobre un dominio preciso. En el momento de definir el sistema de reglas se trabaja con un tipo de dominio no necesariamente igual al dominio final del atributo.
   - De la misma forma las variables involucradas en los antecedentes de las reglas pueden estar definidas sobre cualquiera de los dominios difusos soportados hasta el momento por el modelo, con lo que esto significa. Se incluye el uso de objetos complejos con la gran potencia de modelado que esto supone.
   - El método de inferencia utilizado se basa en el método de inferencia max-min (*max-min inference method*) [MA75].
   - Para evaluar los antecedentes de cada regla se pueden utilizar los operadores difusos incluidos en el modelo, la utilización de cada uno dependerá del dominio difuso sobre el que está definido el atributo del consecuente analizado.

- Es posible definir para el sistema de reglas difusas un umbral a partir del cual se dispararán las reglas.
- El valor final del atributo se obtendrá a partir de la agregación del resultado de todas las reglas.

El nivel de definición en el modelo es soportado por el uso de tipos difusos. Los tipos difusos son un tipo especial de objeto u objeto complejo. Los objetos definidos sobre los tipos difusos pueden ser manipulados de firma individual o como valor de un atributo de un objeto, con lo cual se consideran los tipos difusos como un tipo especial de dominio difuso. Algunos elementos a resaltar de los tipos difusos son:

- Se definieron operadores para los tipos difusos a partir del tratamiento que se le da a la comparación entre objetos de diferentes niveles de un tipo difuso. De esta forma se consideran tres variantes: considerar como parecido igual *uno* la comparación entre un atributo existente en un objeto y no existente en el otro; considerar como *cero* ese parecido; solo tener en cuenta para la comparación los atributos que estén presentes en ambos objetos.
- Es posible definir perspectivas de comparación para los tipos difusos de forma similar a la perspectivas definidas para un objeto u objeto complejo. En este caso se define también el comportamiento al momento de comparar atributos no existentes.

Las extensiones propuestas en esta investigación al modelo de bases de datos difusas orientadas a objetos de Marín y otros [Mar01], han logrado un modelo teórico con un mayor número de prestaciones en un ambiente de bases de datos difusas. *El modelo se enriquece desde el punto de vista conceptual al poder representar atributos inferidos difusos, dando soporte al uso de sistemas de reglas difusas en bases de datos. Es de gran valor para el usuario la incorporación de las características que permiten definir y trabajar con perspectivas de comparación para los objetos difusos complejos y el uso de diferentes estrategias al comparar tipos difusos.*

El modelo teórico propuesto fue incorporado como una contribución a *PostgreSQL* por medio de una plantilla que se instala en el servidor de bases de datos. Las principales características de esta implementación son las siguientes:

- A partir de que *PostgreSQL* es un gestor que no incorpora las principales características objeto-relacional, fue necesario crear una estructura que permitiera la implemen-

tación de todas las características del modelo teórico propuesto y su comportamiento lo más familiar posible a la orientación a objetos.

- Se utilizaron en la implementación principalmente la herencia entre tablas, las funciones definidas por el usuario, la sobrecarga de funciones, los procedimientos almacenados, la utilización de vectores multidimensionales como valor para un atributo.

- Toda la implementación fue realizada utilizando el lenguaje PLpg/SQL soportado por *PostgreSQL*.

- El resultado final de la implementación es una extensión al *SQL* de *PostgreSQL* para dar soporte a todas las características del modelo propuesto.

La implementación del modelo teórico propuesto en un gestor de base de datos como *PostgreSQL*, por medio de la extensión del *SQL*, permite a los desarrolladores utilizar las características difusas de las bases de datos sin necesidad de intérpretes intermedios. El desarrollador dispondrá de un conjunto de funciones que podrá usar directamente sobre sentencias *SQL* tanto en el servidor como en una aplicación cliente. Este resultado puede ser distribuido y utilizado de forma gratuita por cualquier persona o institución sin que esto implique violación alguna.

Además se desarrolló una interfaz Web para que los usuarios puedan interactuar con los elementos difusos de la base de datos. La interfaz está orientada completamente al usuario de forma que facilite la utilización del modelo de la forma más transparente posible. Esta interfaz permitirá la utilización de los resultados de esta investigación por profesionales de especialidades no informáticos.

6.2. Trabajos Futuros

Esta investigación da continuidad a otras investigaciones similares donde las bases de datos difusas son estudiadas y desarrolladas a profundidad, sin embargo aún quedan puntos donde se pueden enfocar las futuras aportaciones en esta área. Algunos de posibles áreas de trabajo son:

1. En esta investigación se utiliza como función de agregación para determinar el parecido entre dos objetos u objetos complejos el propuesto por Vila y otros [VCMP94a].

Este no es la única forma de agregar los resultados de la comparación entre los atributos por lo tanto es conveniente incluir al modelo otras funciones de agregación para determinar el parecido entre dos objetos e incorporar la elección de la función de agregación en la definición de las perspectivas de comparación.

2. El modelo ha sido implementado pensando en que los usuarios compartirán la definición de los dominios y las perspectivas de comparación. Sin embargo, la definición de un dominio no tiene que ser igual para todos los usuarios. Por otro lado los usuarios pueden tener necesidades diferentes para las cual definir una perspectiva. Será útil incorporar al modelo la posibilidad de que a nivel de usuario se gestiones los dominios y las perspectivas de comparación.

3. En el nivel de atributo puede existir imperfección en los datos debido a una imprecisión de los mismos o a la existencia de incertidumbre. El modelo propuesto en esta investigación da soporte para representar y manipular la imprecisión pero no así la incertidumbre. Trabajos futuros pueden centrase en el análisis de la representación y manipulación de incertidumbre explícita en los datos dentro del modelo.

4. Cómo se a mostrado a lo largo de la memoria, *PostgreSQL* como gestor de bases de datos objeto-relacional tiene algunas carencias que han hecho más difícil el proceso de implementación, con un coste de eficiencia intrínseco. Será conveniente implementar el modelo propuesto en esta investigación en un gestor de bases de datos objeto - relacional más avanzado como lo es Oracle que si incorpora todas lass características del modelo objeto-relacional.

5. Varias investigaciones han propuesto modelos conceptuales para modelar bases de datos difusas. Es conveniente Establecer enlaces con modelos conceptuales existentes para que permitan anticipar en la etapa de diseño las nuevas capacidades incorporadas al modelo como son: la definición de atributos inferidos, los tipos difusos y las perspectivas de comparación.

6. No existe una metodología formalizada que permita a un usuario desarrollar una base de datos difusa lo cual incluye en primer lugar técnicas de recopilación de información para determinar los tipos de dominios necesarios en la base de datos y su definición. Trabajar en una metodología que oriente en la definición, representación y adquisición

de información imprecisa utilizando el modelo propuesto en esta investigación puede ser otra de las líneas de aportación.

7. Aplicación de los resultados obtenidos. La aplicación de los resultados obtenidos en otras áreas del conocimiento como puede ser la administración, el turismo, la agricultura, las ciencias pedagógicas, etc. es uno de los principales retos que tiene esta área de investigación. Los aportes de esta investigación abren el diapasón de aplicaciones.

Apéndice A

Modelo de bases de datos relacional

El modelo relacional fue presentado por E.F.Codd en 1970, convirtiéndose en uno de los modelos matemáticos más importantes para la representación de las bases de datos. Es un modelo basado en registros, como lo es también el modelo de red y el jerárquico, pero supuso un gran avance con respecto a estos dos modelos, fundamentalmente por la forma de conexión entre los registros. Mientras en los modelos de red y jerárquico la conexión entre los registros se realiza mediante punteros, en el modelo relacional se utiliza el valor de los registros. Esto último permitió una definición matemática formal del modelo [KS98].

Este modelo se basa en la teoría matemática de las relaciones, y su funcionamiento esta definido sobre un modelo matemático específico: el álgebra relacional y el cálculo relacional.

El modelo relacional se ha establecido como el principal modelo de datos para aplicaciones comerciales de procesamiento de datos. Hoy en día, la gran mayoría de los sistemas de bases de datos están desarrollados sobre este modelo y la tendencia es que seguirá siendo el modelo líder para un grupo de aplicaciones, fundamentalmente en el área de gestión empresarial.

Este modelo de datos tiene tres elementos claramente definidos, a saber, la parte estructural del modelo, la de integridad y la de manipulación, y un cuarto elemento que son las contribuciones que se le han hecho al modelo original. De una forma resumida se presentarán las tres partes principales, figura A.1

Estructura del Modelo Relacional

En la parte estructural el elemento principal de representación en el modelo relacional

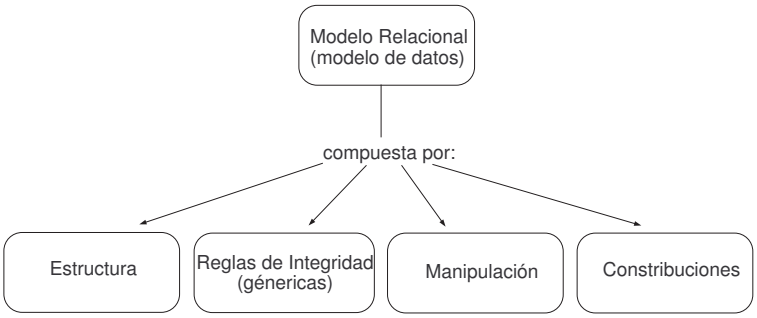


Figura A.1: Principales elementos del modelo relacional.

son las *relaciones*. Sean los dominios $D_1, D_2, ..., D_n$, no necesariamente distintos. Una *relación* R es un subconjunto del producto cartesiano de los dominios. La relación estará formada por un conjunto de n-tuplas de la forma $d_1, d_2, ..., d_n$ tales que $d_1 \in D_1, d_2 \in D_2, ..., d_n \in D_n$.

Las relaciones se representan mediante tablas bidimensionales donde cada fila representa una tupla y cada columna un atributo. La cardinalidad de la relación está dada por la cantidad de tuplas que la conforman. En la terminología relacional las tablas son denominadas como relaciones.

Los atributos de la relación, representados por las columnas de la tabla, tiene cada uno un nombre y un tipo o dominio sobre del cual se definen. Es característica de este modelo que los atributos no tienen un orden fijo al igual que las tuplas que forman la relación.

Para referir un atributo de una relación R se utilizará el nombre definido para el atributo pudiéndose utilizar la notación $R. < nombredelatributo >$ si se desea ser más especifico en la referencia. En cada relación se define una clave que distinguirá una tupla de otra, evitando que dos tuplas en una misma relación sean iguales. La clave de una relación podrá estar formada por un subconjunto de los atributos que forma la relación.

Una característica fundamental de este modelo de base de datos es permitir relaciones entre relaciones, en este caso se utilizan claves externas. Las claves externas son claves primarias en una tabla y atributos en otra tabla que tienen como objetivo representar la relación entre las dos tablas.

Con el objetivo de ganar en eficiencia y evitar anomalías en las operaciones de actualización, inserción y eliminación de tuplas sobre la base de datos se desarrolló un conjunto de directrices de calidad denominadas como *niveles de normalización*. Estas directrices parten de la atomicidad del valor de los atributos y van inspeccionando las dependencias entre los atributos de las relaciones, creando nuevas relaciones en aquellos casos donde se encuen-

tren problemas de redundancia o inconsistencia. Así se tienen varios criterios de calidad que una relación debe cumplir, conocidos como *Formas Normales*. De esta formas existe la *1ra Forma Normal*,*2da Forma Normal*,*3ra Forma Normal*, *Forma Normal de Boyce-Codd*, *4ta Forma Normal* y *5ta Forma Normal*. Aunque estas no son las únicas formas normales, si son las más aceptadas desde el punto de vista práctico en el diseño de una base de datos relacional. Para acceder a un estudio detallado de la teoría de dependencias funcionales, multivaluadas y de reunión, así como de los procesos de normalización, consultar [KS98, UW99].

A.1. Integridad de los datos.

Cualquier sistema gestor de bases de datos debe garantizar una consistencia de los datos que almacena evitando en todo momento que se introduzca información que no pueda ser identificada. De la misma forma no se permiten referencias a un dato que no exista en la base de datos. Para esto los sistemas de bases de datos relacionales definen dos reglas de integridad: integridad de entidad e integridad referencial (figura A.2).

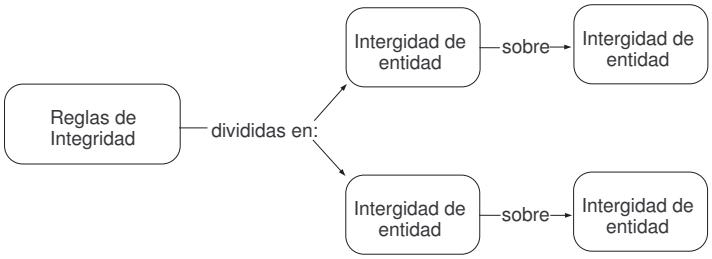


Figura A.2: Reglas de integridad en el Modelo Relacional.

La regla de integridad de entidad no permite que un componente de la clave primaria sea nulo, garantizando de esta forma la identidad lógica de cada tupla. Y la integridad referencial tiene el objetivo de garantizar que las relaciones entre los registros de tablas relacionadas sean válidas y que no se eliminen ni modifiquen accidentalmente datos relacionados. Las claves externas de una relación tienen que corresponderse con la clave primaria de la relación a la que hace referencia.

A.2. Manipulación de los datos.

La manipulación de los datos en el modelo relacional se realiza por medio de dos lenguajes relacionales equivalentes entre si: el álgebra relacional y el cálculo relacional. El primero proporciona un conjunto de operadores mediante los cuales se especifican operaciones sobre las tablas y el segundo proporciona una sintaxis para expresar aquello que se desea obtener de las relaciones sin tener que especificar el mecanismo para obtenerlo. El álgebra relacional es un lenguaje de consulta procedural, mientras que el cálculo relacional es un lenguaje declarativo. Como resultado de cada operación sobre el modelo relacional se obtiene otra relación, denominada relación derivada, sobre la cual se pueden aplicar nuevamente los operadores (Figura A.3).

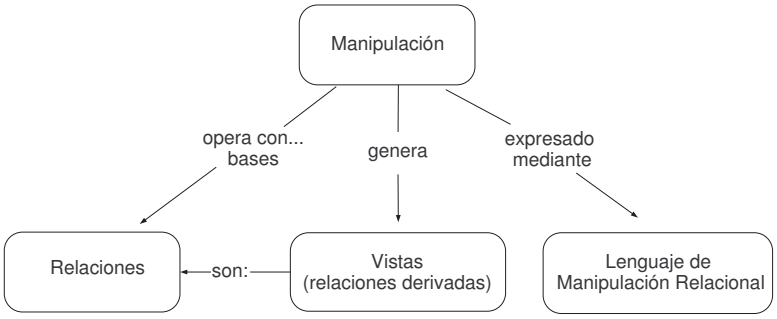


Figura A.3: Manipulación en el Modelo Relacional.

Cuando se implementa el modelo relacional sobre un sistema de gestión de bases de datos se definen dos lenguajes, el lenguaje de definición de datos (*DDL*) y el lenguaje de manipulación de datos (*DML*). El *DDL* permite la creación, modificación y eliminación de elementos de la base de datos. Estos elementos pueden ser: tablas, vistas, índices y otros más sofisticados en dependencia del *SGBD*. El *DML* permite la consulta, modificación e inserción de datos en la base de datos. La función de consulta es las más importante y se basa en el álgebra relacional y/o el cálculo relacional.

Para la implementación de los lenguajes comerciales *DDL* y *DML* se utilizan los llamados *lenguajes relacionales*. Esto lenguajes son conocidos también como lenguajes de consulta y le permiten al usuario interactuar con la base de datos de una forma amigable. En el mercado han prevalecido tres lenguajes: *SQL (Structured Query Language), QBE (Query By Example)* y *Quel (Query Language)*. *QBE* esta basado en el cálculo relacional de dominios; *Quel* en el cálculo relacional de tuplas y *SQL* usa una combinación de cons-

trucciones del álgebra relacional y del cálculo relacional. Un estudio de estos tres lenguajes existe en [KS98].

El lenguaje *SQL* se ha establecido claramente como el lenguaje de bases de datos relacional estándar. Este lenguaje está compuesto por sentencias que realizan las diferentes funciones sobre la base de datos.

Principales comando de SQL

SQL es un lenguaje muy poderoso en expresividad e incluye las operaciones fundamentales del álgebra relacional y del cálculo relacional proporcionando un conjunto de comandos que definen la estructura de la base de datos y que permiten la consulta de sus datos. Los diferentes comandos de *SQL* pueden ser consultados en [KS98, UW99], sin embargo, en esta sección se recuerda la sintaxis de tres de sus comandos más importantes y más utilizados, los comandos CREATE TABLE, INSERT y SELECT.

CREATE TABLE – Este comando permite la creación de tablas (relaciones) en el modelo, definiendo un grupo de restricciones para los valores que puede tomar cada columna (atributo) de la tabla. Las restricciones en una columna podrán especificar cláusulas como DEFAULT, NOT NULL, NULL y CHECK, además de definir si el atributo se comportará como una clave primaria o externa. Esto último se realiza utilizando las cláusulas PRIMARY KEY y FOREIGN KEY respectivamente. Si las claves que se definen en la tabla están compuestas por más de una columna, entonces las restricciones PRIMARY KEY y FOREIGN KEY se definirán en las restricciones de la tabla.

Ejemplo A.1. *Creación de la tabla Jugadores utilizando SQL*

```
CREATE TABLE Jugadores (
Codigo  NUMBER  NOT NULL PRIMARY KEY,
Nombre VARCHAR(30),
Equipo VARCHAR(4)  NOT NULL CONSTRAINT CEquipo REFERENCES
Equipos (ID_Equipo),
Contrato VARCHAR(10) DEAFULT ´Temporal´ CONSTRAINT CContrato
CHECK (Contrato IN(´Temporal´, ´Permanente´)));
```

En el ejemplo A.1 se crea la tabla *Jugadores* con cinco campos. *Código* es la clave primaria de la relación. *Equipo* es una clave externa que relaciona las tablas *Jugador* y *Equipos,* en este caso la tabla *Equipos* debe existir en la base de datos. *Contrato* es un campo que toma el valor *Temporal* por defecto y cuyo valor esta restringido por una lista.

INSERT - este comando se utiliza para insertar nuevas tuplas o registros en una relación, especificando los valores que tomará cada columna. El comando insertará los valores especificados en una *lista de expresiones* como valores de una nueva tupla en una tabla. La asignación del valor de cada columna se realiza en el mismo orden en que fue definida la tabla. Es posible limitar los atributos de la tupla que tomarán valores especificándolos en la *lista_atributos*, ejemplo A.2.

Ejemplo A.2. *Inserción de registros en la tabla Jugadores utilizando SQL. Se quiere adicionar un nuevo jugador en la tabla Jugadores. La sentencia SQL quedaría de la siguiente forma:*

```
INSERT INTO Jugadores
VALUES (003, 'Oscar Gil', 'HOLG', 'Permanente');
```

SELECT - es la instrucción utilizada para consultar la base de datos y extraer información a partir del cumplimiento de determinados criterios. Es el comando más utilizado dentro de los lenguajes de manipulación de datos. La instrucción SELECT genera un producto cartesiano entre varias tablas especificadas en la cláusula FROM y aplica las restricciones que se imponen en la cláusula WHERE, finalmente proyecta el resultado teniendo en cuenta los campos indicados en la cláusula SELECT, ejemplo A.3.

Ejemplo A.3. *Consulta de datos de la tabla Jugadores utilizando SQL. Obtener el nombre, el código y el nombre del equipo de todos los jugadores contratados de forma permanente.*

```
SELECT Nombre, Codigo, NombreEquipo FROM Jugadores, Equipos
WHERE Contrato = 'Permanente'
AND Jugadores.Equipo=Equipos.ID_Equipo;
```

A.3. Sistemas Gestores de Bases de Datos Relacionales.

Desde la aparición del Modelo Relacional muchos fabricantes de sistemas comenzaron a incluir el modelo en sus propuestas. Actualmente la gran mayoría de los sistemas de bases de datos que podemos encontrar en el mercado están fabricados sobre la tecnología relacional. Entre los principales vendedores de soportes para el desarrollo de sistemas de bases de datos que utilizan el Modelo Relacional se encuentra:

1. *DB2* de *IBM (Copyright©1994,2008 International Business Machines Corporation,* www.ibm.com/software/data/db2*)*. Este es considerado el primer producto que incorpora el lenguaje *SQL,* según *Michael Stonebraker*, aunque aun persiste la pelea entre *Oracle* y *DB2* respecto a quién fue el primero en usar *SQL*. Su principal uso ha estado en microcomputadoras.

2. *dBase (Copyright©2004 dataBased Intelligence, Inc,* www.dbase.com*),* es uno de los primeros sistemas de bases de datos relacionales para microcomputadoras. Tuvo una gran popularidad en la década de los 80.

3. *Informix*, (*Copyright©1994,2008 International Business Machines Corporation,* www.ibm.com/informix/) gestor de bases de datos desarrollado por *Informix Software*. A pesar de tener gran éxito en los 90 (fue el segundo sistema de bases de datos más popular, después de *Oracle*) su popularidad ha ido desapareciendo progresivamente.

4. *InterBase (Copyright©1994-2008 Borland Software Corporation*, www.borland .com /interbase*)*, desarrollado y comercializado por *Borland*. Las características de este sistema que lo distinguen del resto son pocos requisitos de administración y una arquitectura multigeneracional; se ejecuta en *Windows*, *Linux*, y *Solaris*. Este producto actualmente se distribuye libremente.

5. *Microsoft SQL Server (Copyright©2008 Microsoft Corporation,* www.microsoft.com /sql*)*, propuesto por *Microsoft*. Es uno de los principales gestores de bases de datos utilizados en negocios e instituciones, se emplea en pequeñas y medianas bases de datos. Su evolución va en ascenso buscando nuevos nichos de mercado.

6. *Oracle (Copyright©2008, Oracle Corporation, www.oracle.com)*. Fue en la década de los 90 el sistema de bases de datos más popular. Actualmente es líder en bases de datos de gran tamaño como las que se utilizan en grandes compañías, hospitales, etc.

7. *SQL/DS* también de *IBM (Copyright©1994,2008 International Business Machines, www.ibm.com/us)*, la primera implementación de un DBMS comercial para mainframe desarrollado por IBM.

8. *Microsoft Access (Copyright©2004 Microsoft Corporation,* www.microsoft.com /office /access) sistema de bases de datos relacional creado por Microsoft orientado a las pequeñas empresas.

9. *PostgreSQL*(www.postgresql.org) Sistema de bases de datos con soporte para el modelo relacional distribuido de forma gratuita. En los últimos años ha aumentado significativamente su utilización en diferentes tipos de aplicaciones y por sus prestaciones ha obtenido premios como el mejor sistema de bases de datos .

10. *Berkeley DB* (www.sleepycat.com) Es una base de datos empotrada que puede ser utilizada por varios lenguajes de programación. Se distribuye con dos licencia una de ellas con cláusulas similares a GNU GPL.

11. *MySQL* (www.mysql.com) uno de los gestores de bases de datos relacionales más instalados en el mundo. Aunque sus comienzos fueron totalmente como software libre actualmente está patrocinado y es propiedad de una empresa privada, aunque aún existe una distribución GPL.

12. *Firebird* (firebird.sourceforge.net) Es el sistema de bases de datos libre de Borland considerado una versión de InterBase.

13. *Apache Derby* (db.apache.org/derby/) Es una base de datos relacional open source implementada enteramente en Java. Es una base de datos de poco tamaño (2 MB) que puede ser embebida en los programas Java. Es una de las bases de datos de mayor crecimiento, respecto a su utilización, en los últimos años.

Existen muchos más sistemas gestores de bases de datos pero en esta lista están los más significativos y populares. La gama de software catalogado como libre que también soportan el modelo relacional han quitando parte del mercado a los reconocidos DBMS, aunque aún está lejos el momento en que los puedan sustituir por completo.

A.4. Limitaciones del Modelo Relacional.

Las aplicaciones de bases de datos inicialmente se centraron en problemas que cumplían de forma general las siguientes características [KS98]:

1. Uniformidad en los datos.

2. Orientación a registros con longitudes fijas.

3. Datos pequeños, registros cortos de 100 bytes o menos.

4. Campos atómicos. No existencia de una estructura definida en los campos.
5. Transacciones cortas.
6. Esquema de conceptos estático. El esquema de la base de datos no cambia con frecuencia.

Estas características son soportadas de forma muy eficiente por el Modelo Relacional. Sin embargo, han surgido nuevas aplicaciones de bases de datos. Tratando de adaptar la tecnología de bases de datos a otras aplicaciones que están fuera del ámbito del procesamiento tradicional de datos como pueden ser: *Diseño Asistido por Ordenador, Ingeniería de Software Asistida por Ordenador, Bases de datos multimedia, Sistemas de información de oficina, Sistemas expertos de bases de datos, etc*, fue que surgieron nuevos requerimientos para las bases de datos [KS98, Sto89], como son:

1. Estructuras de datos no convencionales. Valores no atómicos para los datos.
2. Definiciones de reglas y funciones para los elementos de la base de datos como tuplas o relaciones. Posibilidad de definir un comportamiento para los objetos.
3. Definición de reglas generales para la aplicación. Meta conocimiento.
4. Un mayor acercamiento al modelo semántico de los datos.

Además de esto el Modelo Relacional tiene un conjunto de limitaciones en las propias características que lo hacen tan eficiente. Por ejemplo, el reflejo y la manipulación de todos los datos en forma de tabla es un problema cuando existen relaciones entre datos con estructuras sofisticadas, puesto que la única salida es establecer relaciones entre tablas y la utilización de claves externas. En [PMM$^+$03, Sto89] se exponen las principales limitaciones del modelo que se pueden resumir en :

1. Para representar objetos complejos o datos altamente relacionados se requiere de numerosas tablas que den soporte a la estructura. Como resultado se obtendrá un modelo poco expresivo y de difícil manejo, provocando que en la práctica su funcionamiento sea lento.
2. El modelo relacional no ofrece soporte para expresar estructuras recursivas o anidadas. Ni colecciones de datos que correspondiendo al mismo tipo de entidad, tienen un tipo similar pero no idéntico.

3. Capacidad de creación de tipos definidos por el usuario muy limitada.

4. Las aplicaciones que actualmente se están desarrollando para la gestión de datos, utilizan lenguajes con estructuras de datos propias. Esto provoca un salto de impedancia entre el lenguaje de la aplicación y los lenguajes *(SQL)* que soportan las bases de datos y con la propia representación de los datos (*impedance mismatch*).

5. En el Modelo Relacional no está previsto almacenar el código junto con los datos, para ocultar los datos o limitar el acceso a ellos. Esto puede provocar que algún código intruso pueda afectar los datos.

El surgimiento de estas necesidades en el modelado de las bases de datos y las propias limitaciones del Modelo Relacional dieron un impulso en el desarrollo de nuevos modelos.

Apéndice B

El Modelo Orientado a Objetos.

Hasta la década de los 80 los sistemas de bases de datos han estado dominados por el modelo relacional. Con el modelo relacional ha sido posible modelar e implementar la mayoría de los sistemas actuales de bases de datos. Sin embargo, durante esos años comenzaban a surgir aplicaciones con características muy particulares y con una definición de datos más compleja. Producto de la propia complejidad de los datos se requerían de estructuras complejas en la base de datos que el modelo relacional no podía brindar [Sto89].

Estas nuevas aplicaciones se caracterizan por contener procesos que leen información de una estructura compleja, procesan esa información utilizando algoritmos de una determinada complejidad y luego escriben en la estructura. Por lo que los requerimientos de consulta sobre los datos son menores en este tipo de aplicaciones [DD98].

Este tipo de problemas ha motivado el desarrollo de una nueva tecnología de bases de datos, los Sistemas de Bases de Datos Orientados a Objetos (*Object-Oriented Database Management Systems*), conocidos por sus iniciales en castellano *SGBDOO* o en inglés *SGBDOO*.

Esta tecnología se basa fundamentalmente en los conceptos de orientación a objetos (*OO*) los cuales son aplicados a sistemas de bases de datos. Esta es una de las áreas de la industria del software y del ámbito académico que más atención ha tenido en los últimos años. El término *OO* indica más una forma de diseñar y una metodología para el de desarrollo de software que un lenguaje de programación específico, aunque los diferentes tipos de lenguajes han realizado sus propios compiladores para soportar este tipo de programación. Es el caso de lenguajes como C++ [Dei03], Java [Mey00],o el más reciente C# [http://msdn.microsoft.com/vcsharp/].

La filosofía de trabajo *OO* surge en la década de los 80 provocando mejoras de amplio alcance en la forma de diseño, desarrollo y mantenimiento del software, tratando de ofrecer una solución a largo plazo a los problemas y preocupaciones que han existido desde el comienzo del desarrollo del software: la falta de portabilidad del código, la reusabilidad, facilidad de modificación del código, ciclos de desarrollo más cortos y técnicas de codificación más intuitivas.

El elemento básico de este paradigma no es la función, como en el caso de la programación estructurada, sino un ente denominado objeto. Se trata de representar un modelo de la realidad en forma de conjuntos de objetos que interactúan entre sí por medio de mensajes. Toda la metodología se sustenta sobre algunos conceptos básicos como: objeto, clase, encapsulación, herencia y polimorfismo.

Objetos.

Son la base de la metodología orientada a objetos. Por medio de los objetos podemos representar cualquier entidad compleja del mundo real a través del valor de sus atributos esenciales y definir un comportamiento por medio de operaciones. Por lo tanto un objeto es un elemento del mundo real, caracterizado por los valores de sus atributos y con un comportamiento definido. Teniendo un significado a efectos del problema que se esté modelando.

Un objeto puede ser real o abstracto, por ejemplo: una organización, una factura, una figura en un dibujo, una pantalla de usuario, un avión, un vuelo de avión, etc. Dentro del software orientado a objetos, un objeto es cualquier cosa, real o abstracta, acerca de la cual almacenamos datos y los métodos que controlan dichos datos. A la vez un objeto puede estar compuesto por otros objetos. Y estos últimos a su vez también pueden estar compuestos por otros objetos. Esta intrincada estructura es la que permite construir objetos muy complejos.

Un objeto se caracteriza por tres elementos: su identificador, sus propiedades y sus métodos:

1. **Identificador**: En el mundo real cada entidad está identificada, de la misma forma nuestros objetos estarán identificado por un identificador de objeto *(OID)*. El *OID* expresa que aunque dos objetos sean exactamente iguales en sus atributos, su identidad es diferente. De esta forma un objeto tiene una existencia independiente de los valores de sus atributos. Esta es una diferencia crucial con respecto al modelo relacional. El *OID* del objeto lo genera automáticamente el sistema, identificando

de forma unívoca al objeto. Por lo tanto, al comparar dos objetos estos pueden ser iguales por su *OID*, logrando una igualdad de identidad o por sus atributos logrando una igualdad de valor. Como es de suponer una igualdad por *IDO* infiere una por valor, no cumpliéndose de forma inversa.[Mar01]

2. **Propiedades**: Son las propiedades que caracterizan el estado de un objeto. Las propiedades pueden ser a su vez otros objetos definiendo de esta forma las relaciones entre los objetos.

3. **Métodos**: Es el conjunto de operaciones que pueden realizarse sobre un objeto. Estas operaciones están soportadas por los procedimientos o funciones que definen el comportamiento de un objeto. Los métodos están caracterizados por su nombre, parámetros y valor que retorna, pueden ser de tres tipos: observadores, que devuelven información acerca del estado del objeto; modificadores, que cambian un objeto de un estado válido a otro; y constructores, que permiten la creación de objetos.

Mensajes

Para operar sobre un objeto se utilizan señales sobre el objeto. Las señales hacen que se realice una determinada operación. Una operación ejecutará el o los métodos apropiados y, de manera opcional, producirá una respuesta. Esta señal es lo que se denomina *mensaje.* Un mensaje siempre debe identificar el objeto receptor y el método a utilizar, además de los argumentos necesarios para realizar la operación. Por tanto un mensaje estará formado por: nombre del objeto, nombre de una operación y, si fuera necesario, un grupo de parámetros.

Tipos y Clases.

Los objetos presentes en nuestro modelo, al igual que las entidades del mundo real, se pueden clasificar según tipos de objetos. En el tipo se definirán los rasgos comunes de un conjunto de objetos con las mismas características y el mismo comportamiento. De esta forma un tipo define los datos a almacenar de un conjunto de objetos con las mismas características. Define también el conjunto de operaciones que se pueden realizar sobre esos objetos.

Las clases son abstracciones que permiten describir un conjunto de objetos de un determinado tipo. El concepto de clase engloba dos vertientes: el conjunto de objetos y el tipo que los caracteriza. En la definición del tipo de una clase se pueden especificar, aparte de los atributos y métodos que caracterizan el tipo de sus objetos, otros atributos que van a estar ligados a la clase como tal, teniendo un valor compartido por todos los objetos de

esa clase. Estos atributos son llamados como *atributos de clase*. De la misma manera se pueden definir *métodos de clase* que son métodos dirigidos a la propia clase y que actúan sobre todos los objetos de la clase y no tienen necesidad de la definición de un receptor particular (el receptor es la propia clase).

Encapsulamiento.

La principal característica de una clase es la de encapsular la estructura y el comportamiento de los objetos, impidiendo ver la implementación interna de una operación, y permitiendo al mismo tiempo su reutilización. La interacción con los objetos se realiza por medio de la interfaz definida para cada objeto; sólo se necesita conocer qué hace y no cómo se hace.

Esto se realiza estableciendo en la definición del tipo de una clase la visibilidad de los atributos y los métodos de la misma. Usando la palabra *Private* para aquellos métodos y atributos que no podrán ser accedidos por ningún procedimiento externo; *Protect* para aquellos métodos y atributos que podrán ser accedidos sólo por subtipos de la clase y *Public* para los que no tienen ninguna restricción.

La aplicación entera se reduce a un agregado de objetos. El encapsulamiento protege a los datos asociados a un objeto contra su modificación directa por quien no tenga derecho a acceder a ellos, eliminando efectos secundarios.

Instanciación de clases

Es el mecanismo por el cual podemos crear objetos de una clase, los cuales estarán caracterizados por los atributos y métodos definidos en el tipo de la clase. Al hablar de una instancia de una clase nos estamos refiriendo a un objeto de esa clase.

Todos los objetos se almacenan de forma independiente, aunque la definición del comportamiento es la misma. La instanciación de los objetos se suele implementar mediante un método de clase especial denominado *constructor*.

Ejemplo B.1. *Definición de la clase Alumno.*

```
class Alumno {
1. private:
2.  char *nombre;
3.  int edad;
4. Residencia * reside;
5. static int cant_est;
6. public:
```

```
7.  Alumno (char * n, int e){...};
8.  void SetEdad(int e) {...};
9.  void SetNombre(char * n) {...};
10. char *GetNombre(){return nombre;}
11. int GetEdad (){ return edad;}
12. ...
};
```

El ejemplo B.1 está codificado en *C++* y en él se declara una clase *Alumno*. Todos los objetos o instancias de esta clase se caracterizaran por las propiedades *nombre, edad* y *residencia* (líneas 2-4) y su comportamiento estará definido por los métodos definidos (líneas 7-12). La propiedad *reside* es compleja pues refleja una referencia a otro objeto de tipo *Residencia*. La propiedad *cant_est* es un atributo de la clase que tendrá el mismo valor para todos los objetos. Observe que la visibilidad de las propiedades de los objetos es privada, es decir solo podrá ser accedida por los métodos del objeto.

Para acceder a un método de un objeto se le debe pasar un mensaje especificando el método y los parámetros, ejemplo B.2. Los métodos de clase se puede acceder directamente mediante el operador (::).

Ejemplo B.2. *Mensajes a un objeto.*

```
Alumno a1(), a2(‘‘Miguel Barnet’’, 26);
a1.SetNombre (‘‘Enrique Nuñez’’);
```

Persistencia.

La duración del almacenamiento de un objeto es una propiedad estrechamente relacionada con el tipo de almacenamiento. Se define como el mínimo potencial de vida que tiene la zona donde se almacena un objeto. Por lo tanto determina el periodo en el que los objetos pueden tener existencia real, es decir, estar alojados físicamente en memoria. Esta propiedad acompaña al objeto desde el instante de su creación (*definición*) y depende del modo en que se realizó esta definición.

Cuando la duración de los objetos es persistente se mantienen sus existencias más allá del procedimiento que le dio existencia y en ocasiones es deseable que los objetos permanezcan más allá del tiempo de ejecución del software.

Herencia.

Las clases que conforman un modelo no están aisladas, sino que se relacionan entre sí en forma de jerarquía. La jerarquía organiza las clases en forma de árbol, permitiendo la reutilización de las estructuras y tipos definidos por el programador, al tiempo que sirve de base para capacidades avanzadas de la orientación a objetos, como son la redefinición de propiedades, el enlazado dinámico y el polimorfismo Esta jerarquía se forma por medio del mecanismo de herencia, en el cual una clase puede estar definida sobre la base de otra, heredando de ella las propiedades que la caracterizan y los métodos que definen su comportamiento e incorporando nuevas propiedades y métodos para completar su definición. También se puede redefinir variables o funciones miembro ya existentes.

Cuando se trabaja con herencias se definen los supertipos y los subtipos. Un subtipo es un tipo que hereda de otro tipo y el supertipo es un tipo del cual hereda un subtipo. De la misma forma están las subclases y las superclases. Las subclases heredan de las superclases. En el siguiente ejemplo se muestra el proceso de herencia de una clase con respecto a otra.

Ejemplo B.3. *Herencia entre clases, figura B.1.*

```
class Alumno_Extranj:public Alumno {
2.  char *pais;
3.  Institucion * procedencia;
4. public:
5.  Alumno_Extranj ():Alumno();
6.  Alumno (char *p, Institucion *i);
7.  void Setpais(int p);
8. void Setprocedencia(char * i);
9.  char *Getpais (){return pais;}
10. Institucion * Getprocedencia (){ return procedencia;}
};
```

En el ejemplo B.3 se tiene una clase *A_Extranj* que hereda los atributos y propiedades de la clase *Alumno* y adiciona nuevos atributos y procedimientos a la clase. Todos los objetos que sean instanciados a partir de una clase son miembros también de las superclases de la clase.

Existen dos mecanismos de herencia, la herencia simple y la herencia múltiple. En la herencia simple la clase hereda solo de una superclase, mientras que en la herencia múltiple

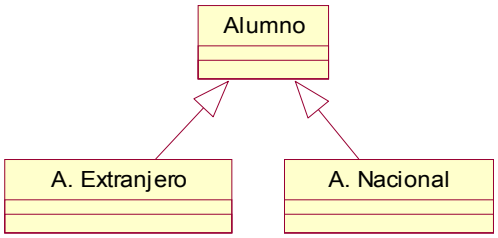


Figura B.1: Herencia entre clases.

la clase hereda las propiedades y el comportamiento de más de una superclase, ejemplo B.4.

Ejemplo B.4. *Herencia múltiple*

```
class Alumno_Extranjero: public Alumno, public Extranjero {
};
```

En este ejemplo se define una nueva clase *Alumno_Extranjero* que hereda tanto de la clase *Alumno* como de la clase *Extranjero*.

El mecanismo de herencia presenta múltiples ventajas evidentes. La principal ventaja es la posibilidad de reutilizar código sin tener que escribirlo de nuevo. Esto es posible porque todas las subclases pueden utilizar el código de las superclases sin tener que volver a definirlo en cada una de ellas. La presencia de este mecanismo es la principal diferencia entre un lenguaje de programación orientado a objetos y otro que no lo es.

Como se ha señalado, una clase puede redefinir las definiciones hechas por sus superclases permitiendo personalizar la subclase. Por lo general los atributos no son redefinidos y esta facilidad se centra más en la redefinición de algunos métodos para matizar la operación que realiza, sin introducir grandes cambios desde el punto de vista semántico en la función de la operación.

En algunos casos una clase no tiene otra utilidad que la de ser superclase para otras clases que se deriven de ella. A este tipo de superclase, que nunca pueden ser instanciadas debido a que no tienen completa su definición, se les denomina *clases abstractas* y su función es la de agrupar miembros comunes de otras clases que se deriven de ellas y sentar las bases que permitan escribir código polimórfico para manipular los objetos de las subclases.

Polimorfismo.

El polimorfismo es un mecanismo que permite invocar funciones definidas con una misma interfaz pero implementadas de forma diferente. Estas funciones pueden actuar sobre objetos distintos dentro de una jerarquía de clases, sin tener que especificar el tipo exacto de los objetos. Por lo tanto el mecanismo del polimorfismo permite definir e invocar funciones que comparten la misma interfaz pero tienen una implementación diferente.

Al llamar un método abstracto el compilador no decide en tiempo de compilación cuál será la función que se debe utilizar en un momento dado del programa. Esa decisión se toma en tiempo de ejecución. A este proceso de decisión en tiempo de ejecución se le *denomina vinculación dinámica o tardía*, en oposición a la habitual vinculación estática o temprana, consistente en decidir en tiempo de compilación qué función se aplica en cada caso.

Ejemplo B.5. *Uso del polimorfismo.*

```
Objeto_Geometrico *obj1 = new Punto();
Objeto_Geometrico *obj2 = new Linea();
obj1.dibuja();
obj2.dibuja();
```

En el ejemplo B.5 se declararon dos punteros a una superclase *Objeto_Geometrico* que apuntarán a dos subclases distintas *Punto* y *Línea*. Al llamar la función *dibujar* se ejecutará la implementación que corresponda al tipo de objeto apuntado. Todo este proceso se realiza en tiempo de ejecución.

B.1. Sistemas Gestores de Bases de Datos Orientadas a Objetos.

Hasta aquí se presentaron los principales conceptos de la tecnología orientada a objetos que le darán funcionalidad a los *SGBDOO*, ahora se relacionarán las principales características de estos tipos de sistemas, sus particularidades, sus ventajas y sus desventajas.

Generalidades sobre SGBDOO.

Los sistemas *SGBDOO* comienzan en la década de los 80 a generalizarse en múltiples aplicaciones. La utilización de estos sistemas se vio beneficiada por el aumento en el tamaño de la memoria principal disponible, la velocidad de las unidades centrales de procesamiento,

la reducción del coste del hardware y la mejora en el entendimiento de la gestión de la base de datos que se ha logrado en los últimos años [KS98].

La mayoría de los *SGBDOO* tienen su base en los modelos semánticos de datos y en los lenguajes de programación orientados a objetos. Los primeros pasos en este tipo de sistemas se dieron en una primera generación por los conocidos proyectos *Iris, Orion, GemStone, O2, Ontos* [DD98].

Producto de esta intensa actividad, fue necesario definir de una manera precisa el concepto de *SGBDOO*. En efecto, contrariamente al caso de los sistemas relacionales que primero fueron definidos formalmente en el artículo original de *Codd*, después se generaron prototipos y finalmente transformados en producto, no hubo un principio de especificación precisa de lo que debía ser un *SGBDOO*.

De forma general la definición de base de datos orientada a objetos se muestra como una colección de objetos en la que su estado, comportamiento y relaciones son definidos de acuerdo con un modelo de datos orientado a objetos. Por lo tanto un *SGBDOO* será un sistema de bases de datos que permite la definición y manipulación de una base de datos orientada a objetos.

Los *SGBDOO* combinan características de orientación a objetos y lenguajes de programación orientados a objetos con capacidades de bases de datos. Estos sistemas están basados - en la mayoría de los casos - en la arquitectura de un lenguaje de programación de bases de datos. Las aplicaciones son escritas utilizando una extensión de un lenguaje de programación existente, extendidos para incorporar funcionalidad de base de datos. El objetivo de estos sistemas es llegar a integrarse con múltiples lenguajes. Esto puede suponer un problema debido a lo cercano de la asociación que se requiere entre el sistema de tipos de la base de datos y el sistema de tipos del lenguaje de programación.

En [Atk90] se definió el conjunto de características que debía cumplir un *SGBDOO*. Estas características se dividieron en tres grupos:

1. Obligatorias. Son características que deben estar presentes en los *SGBDOO*. Entre ellas están: soporte para objetos complejos, identidad de objetos, encapsulación, tipos, clases, herencia, extensibilidad, persistencia, administración de almacenamiento secundario, concurrencia, recuperación y facilidad de consultas.

2. Opcionales. Características que pueden adicionarse al sistema pero que no son obligatorias: herencia múltiple, chequeo de tipos, diseño de transacciones y versiones.

3. Abiertas. Es un grupo de características que el propio diseñador desee incorporar. Es un grado de libertad que se le deja a los desarrolladores de sistemas *SGBDOO*.

Cómo enfocar estas características y cuáles son los principales elementos que se deben tener en cuenta en cada una, se puede encontrar en [Atk90].

La incorporación de la noción de objetos complejos en *SGBDOO* optimiza la representación de las estructuras complejas, acortando la distancia que separa el modelado de un escenario del mundo real y su implementación. Los objetos complejos permiten así un modelado más intuitivo de datos de una aplicación y su presentación en la base de datos estará más cerca de la realidad. La estructura compleja de los objetos es manejada por punteros lógicos. Esos punteros reemplazan la utilización de uniones relacionales, e inducen así una ganancia de desempeño, particularmente cuando la cantidad de datos es importante. Un esquema típico de un *SGBDOO* se muestra en la Figura B.2 [Man94].

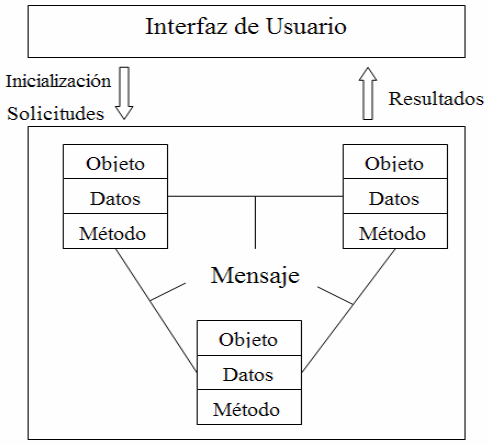


Figura B.2: Un *SGBDOO* tradicional [Man94]

El concepto de identidad de objeto, tiene repercusiones particularmente interesantes en el contexto de un *SGBDOO*. Un objeto en el sistema puede ser identificado de forma única por medio de su identificador. Esto permitirá una mejor gestión de actualización, ganando en rapidez. Además de ofrecer una gestión automática de integridad referencial al desaparecer las referencias a objetos destruidos, problema crucial de las bases de datos.

A principios de la década de los 90', la comercialización efectiva de este tipo de sistemas progresó rápidamente y fueron desarrolladas varias aplicaciones. A raíz de esto se fue

desarrollando un modelo, el *ODMG* [Cat00], que en nuestros días ya esta en su versión 3.0 (www.odbms.org/odmg.html).

ODMG: Un estándar para bases de datos orientadas a objeto puras.

Un grupo de investigadores realizaron trabajos con el objetivo de asegurar la portabilidad de las aplicaciones realizadas sobre *SGBDOO*. En este objetivo se definen tres componentes [Cat00]:

1. Lenguajes de especificación de objetos. Existen dos lenguajes definidos. El *ODL* (*Object Definition Language*) que permite definir el modelo de datos especificando la definición de objetos complejos, la relación entre esos objetos y los métodos asociados a dichos objetos. Y el *OIF (Object Interchange Format),* un lenguaje utilizado para descargar y cargar el estado actual de un *ODMS* desde o hacia un archivo o conjunto de archivos.

2. Lenguajes de consulta de objetos. *OQL* (*Object Query Language*). Es un lenguaje de requerimientos que permite consultar objetos de estructuras complejas, enviar mensajes a objetos, efectuar *join* y otras operaciones de tipo asociativo. Este lenguaje permite además actualizar los objetos. Es un lenguaje no procedural basado en el *SQL*.

3. Lenguajes de conexión: se definen los principales elementos para la implementación de SGBDOO en lenguajes como *C++*, *Smalltalk* y *Java*.

Estos componentes forman la base y marcan las pautas a seguir para la implementación de un *SGBDOO*.

Principales características de los sistemas SGBDOO

Es posible resumir las principales características de los sistemas *SGBDOO* reflejando las ventajas de su utilización en sistemas de bases de datos [Vil02]:

1. Un modelo orientado a objetos que respalda objetos complejos, identidad del objeto, encapsulamiento, clases, extensibilidad e integridad.

2. Un *DDL* (*Data Definition Language*) real con rasgos de manipulación de esquemas. Que permite al usuario definir tipos de objetos, procedimientos y variables asociadas con los tipos de objetos.

3. La capacidad de almacenar y manipular tanto los datos (objetos) como los metadatos (clases y métodos) en el propio sistema.

4. Un *DML* (*Data Manipulation Language*), a través de un lenguaje de consulta completo, declarativo.

5. La capacidad de desarrollar aplicaciones completas en un medio único.

6. En relación al desempeño, los *SGBDOO* dominan con respecto a los sistemas relacionales empleados en aplicaciones que manipulan objetos complejos. Esto es por una sencilla razón; que los sistemas relacionales fueron hechos y diseñados para efectuar algunas operaciones simples como selección, proyección, etc, y los *SGBDOO* tienen por finalidad manipular objetos estructurados y complejos.

7. Los *SGBDOO* han permitido llegar a nuevos dominios para los cuales las bases de datos tradicionales encuentran más dificultades para ser aceptadas. Su fuerte es en ambientes donde hay una necesidad de datos no estándar, es decir, aquellos donde se manipulan textos estructurados o no estructurados, imágenes, gráficos, sonidos, videos, documentos o programas. Se trata entonces de ambientes donde la estructura de los datos es tan compleja que representarla en un modelo tradicional es ineficaz. Por ejemplo: *CAD, Gestión de datos técnicos, Cartografía, Multimedia, Sistemas distribuidos y cliente/servidor, Bases de datos multimedia, Correo por voz, GIS.*

8. Disminuyen los costos de desarrollo por todas las ventajas que en el desarrollo tienen los sistemas orientados a objetos.

Principales SGBDOO.

Entre los principales gestores de bases de datos orientados a objetos que hoy están en el mercadoestán:

1. Objectivity/DB *(Copyright ©2000-2008 Objectivity Inc,* www.objectivity.com*)*

2. Objectstore (*Copyright 1993-2008. Progress Software Corporation* www.progress.com/objectstore/)

3. Versant *(Copyright ©2003-2006 Versant Corporation Inc,* www.versant.com*)*

4. GemStone/S and GemStone Facets *(Copyright ©2008, The GemStone Systems, Inc, www.gemstone.com)*

5. ObjectDB for Java/JDO *(Copyright 2001-2007 ObjectDB Software, www.objectdb.com)*

6. Progress ObjectStore PSE Pro (Copyright 1993-2008. Progress Software Corporation, www.progress.com/ objectstore/)

7. db4Object (Copyright 2000-2008 db4objects, Inc., www.db4o.com)

Abundante información sobre los SGBDOO y otros SGBDOO puede consultarse en http://www.odbms.org/.

Problema DE las bases de datos orientadas a objetos

A pesar de la adecuada filosofía de los *SGBDOO*, los sistemas existentes actualmente no están en absoluto exentos de problemas. Algunos de estos problemas son:

1. Código intruso para gestionar la base de datos. La no utilización de un estándar como gestor persistente de datos para trabajar con las *SGBDOO* provoca una especie de impedancia entre los objetos definidos en un lenguaje y su manipulación desde otro lenguaje orientado a objetos.

2. Carencia de interoperabilidad. Integración de múltiples lenguajes de programación de forma que objetos que hayan sido creados en una base de datos que emplea como lenguaje de programación C++ puedan ser recuperados desde otra base de datos con otro lenguaje de programación orientado a objetos diferente.

3. Difícil portabilidad. Los sistemas se construyen para plataformas específicas.

4. Extensibilidad limitada. Poder de adaptarse al tipo de datos a gestionar, seleccionando por ejemplo las técnicas de indexación adecuadas. Esto exige un sistema flexible que permita la extensibilidad y adaptabilidad del sistema.

5. Problemas relacionados con los sistemas operativos. La base de estos problemas está en que un sistema operativo soporta un sencillo conjunto de abstracciones con una implementación simple para cada una de ellas. Si se requiere una implementación diferente para una de ellas, la implementación tiene que ser modificada o bien el constructor del *SGBD* tiene que re-implementar la abstracción.

6. Optimización de consultas. Este es el mayor problema de los *SGBDOO*. En ocasiones al implementar una solución de consulta se necesita involucrar a toda la base de datos provocando un bajo desempeño.

7. El Modelo Relacional de bases de datos por muchos años ha sido aplicado para el desarrollo de aplicaciones. Por lo tanto el uso de bases de datos orientadas a objetos en una institución provoca los siguientes inconvenientes: necesidad de disponer de dos servidores uno para las bases de datos relacionales previamente existentes y otro para las orientadas a objetos, necesidad de contratar a personal para atender los dos sistemas o tener gastos de formación del personal antigua que atiende los sistemas relacionales.

Estos problemas han motivado que la cuota de mercado de estos productos se haya visto reducida a aquellas aplicaciones donde su capacidad de modelado es esencial, mientras que en aplicaciones convencionales se sigue imponiendo la filosofía relacional.

Apéndice C

El Modelo Objeto Relacional.

Hasta hace poco tiempo, sólo se podía elegir entre sistemas relacionales *SGBDR* o sistemas orientados a objetos *SGBDOO*. Pensando en la utilización de las ventajas de *SGBDOO* y manteniendo la forma de almacenar y acomodar los datos, surgen los sistemas de gestión de bases de datos relacionales orientados a objetos, *SGBDOR*.

Un *SGBDOR* es un modelo en el cual se cumplen los conceptos de orientación a objetos y de *BD* relacional. *SGBDOR = SGBDOO + SGBDR.* El principal objetivo de los *SGBDOR* fue lograr los beneficios de los modelos *SGBDR* y *SGBDOO,* como es la escalabilidad del modelo y el soporte de una rica definición de tipos de datos. Los *SGBDOR* emplean un modelo de datos que intenta incorporar características *OO* en un *SGBDR*[Sto99], figura C.1.

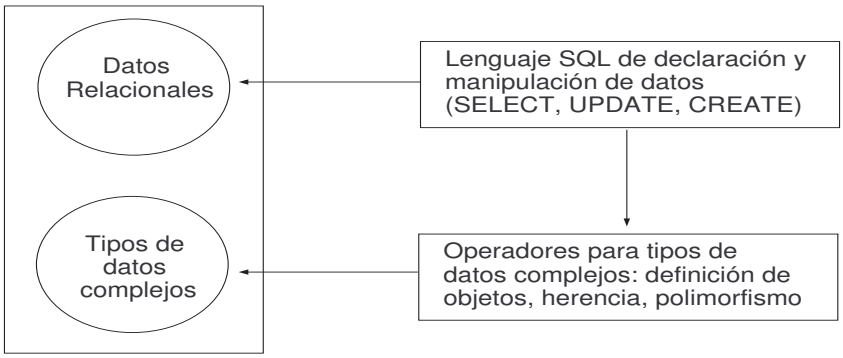


Figura C.1: Modelo de un *SGBDOR*

Los sistemas basados en *SGBDOR* hacen un uso extensivo de elementos de orientación

a objetos tales como: sistema de tipos extensible por el usuario, encapsulamiento, herencia, polimorfismo, asociación dinámica de métodos, objetos complejos sin la primera forma normal e identidad de objetos. Todo esto soportado por un *SGBDR* extendido.

Estas ideas son las que subyacen en el desarrollo de los nuevos sistemas de bases de datos relacional. No todos los sistemas han seguido el mismo enfoque, por lo que existen distintos modelos cuyas características dependen de la forma y el grado en que se haya extendido el modelo relacional. No obstante todos los modelos comparten el concepto de tabla y el lenguaje de consulta, incluyen algún concepto de *objeto* y permiten almacenar conjuntamente datos y métodos a través del concepto de *disparador* o procedimiento.

Se ha utilizado una terminología muy diversa para nombrar a este tipo de sistemas, inicialmente denominados *Sistemas Objeto-Relacional*. Sin embargo últimamente se ha impuesto la terminología *Relacional Orientado a Objetos*, (*SGBDOR*) que es la utilizada por las tres grandes casas comerciales que han apostado más fuertemente por estos nuevos modelos que son: *Oracle e IBM*.

Como era de esperar, los lenguajes asociados a estos modelos son extensiones de *SQL*. De hecho, existe un estándar de *SQL* que incorpora los elementos objeto-orientados, el *SQL:1999*una extensión del *SQL2* [EM99]. Por tal razón el modelo relacional tiene que ser drásticamente modificado para soportar características clásicas de la programación orientada a objetos. Poco a poco los investigadores han ido ajustando la lista de requerimientos de un sistema *SGBDOR*. Las propiedades postuladas en la literatura se dividen en tres categorías:

1. *Modelo de datos objeto-relacional* (*MDOR, ORDM*) con tipos definidos por el usuario (*TDA*), tablas tipificadas y herencia sobre tipos y tablas tipificadas producto de rutinas definidas por el usuario (*RDU,UDR*)

2. *Infraestructura extensible* integrando nuevos operadores en los lenguajes de consulta, contenedores de datos, métodos para acceso a tablas o árboles de búsqueda genéricos.

3. *Mejora en el funcionamiento* con sistemas de reglas de propósito general.

C.0.0.1. Ventajas e Inconvenientes de los SGBDOR

La principal ventaja de extender el modelo relacional radica en la posibilidad de reutilizar y compartir. Dicha posibilidad aparece cuando se considera que se pueden asociar a

los tipos almacenados, funcionalidades que antes se debían asociar con las aplicaciones. Por ejemplo los métodos de manejo de los tipos de datos especiales se pueden compartir por todas las aplicaciones que utilicen dichos datos. Esta es una ventaja típica de la orientación a objetos.

Otra ventaja obvia es que el extender el modelo relacional permite preservar el cuerpo de conocimiento y la experiencia obtenida en el desarrollo de aplicaciones relacionales. Así se desarrollan adecuadamente las nuevas funcionalidades de los sistemas. Como elemento relevante en este tipo de sistema es que de la misma forma que los *SGBDOO* eliminan el saltó de impedancia permitiendo modelar mucho mejor los problemas de la realidad.

Inconvenientes de los SGBDOR

Los inconvenientes presentados a este tipo de modelo surgen de los colectivos que son más radicales en la defensa, bien del modelo relacional puro, bien del modelo orientado a objetos puro.

Los defensores del modelo relacional [Dat01] argumentan que el éxito de dicho modelo se debe a sus propiedades de esencialidad y sencillez de manejo, y que dichas propiedades se pierden en las extensiones. Consideran además que la actual tecnología de sistemas de bases de datos relacionales no garantiza un rendimiento adecuado para estas extensiones. Por último no consideran que dichas extensiones sean necesarias ya que el número de aplicaciones que las necesitan son minoritarias frente a la gran cantidad de aplicaciones existentes para el modelo relacional.

Los defensores del modelo orientado a objetos puro alegan que la terminología del modelo relacional extendido es engañosa. Los vendedores de sistemas relacionales orientados a objetos intentan considerar el objeto como una extensión del modelo relacional, y esto es de todo punto incierto. El punto de vista de la orientación a objetos se pierde por completo, y no se tiene en cuenta la gran diferencia semántica entre ambos enfoques. Sencillamente, las aplicaciones orientadas a objetos no están centradas en los datos, como las relacionales; sino que son mucho más dinámicas, ya que encapsulan tanto información como conducta, y esta visión se pierde con una simple extensión de los datos.

Esta polémica se ha reflejado en los distintos manifiestos referentes a los nuevos modelos de bases de datos [Atk90, DD98, Sto89]. Sin embargo consideramos que se ha llegado a un punto intermedio entre las bases de datos relacionales y orientados a objetos, donde sin perder las características fundamentales del modelo relacional se han podido incorporar nuevas funcionalidades del paradigma orientado a objeto. Actualmente en el mercado se dispone de herramientas de este tipo con rendimientos adecuados, muy cercanos a los de un

base de datos relacional y permitiendo un acercamiento entre el modelo de datos abstracto y el mundo real que se modela.

C.0.0.2. Características de los SGBDOR

Los *SGBDOR* tienen características relacionadas con la orientación a objetos como son: definición y utilización de datos complejos, herencia de tipos y comportamiento de los objetos y mantiene un total soporte al modelo clásico de bases de datos relacional. El uso de estas características en sistemas de bases de datos se vio formalizada mediante el SQL3 o SQL:1999 [EM99] a partir de la experiencias en el desarrollo de este tipo de sistema, adquirido por varias empresas (ORACLE8 de Oracle, Informix Universal Server, DB2 Universal Database de IBM y Cloudscape, entre otras). Un recorrido por los principales elementos se presenta a continuación.

Datos complejos.

La creación de datos complejos en un *SGBDOR* está basada en la utilización de tipos de objetos largos (LOB, Large Object Data Type), colecciones, esquemas definidos por el usuario (Tipo definidos por el Usuario(*user-defined type, UDT*)) y referencias. Sus principales características son las siguientes:

Tipos Largos Uno de los principales tipos de datos incorporados al SQL:1999 son los tipos largos(LARGE OBJECT, LOB) divididos en Binary Large Object (BLOB) para almacenar objetos multimedia: audio, video, mapas, imagen y los Character Large Object (CLOB) para almacenar texto. Al declararse en una tabla es necesario especificar su capacidad en KBytes, MBytes o GBytes, por ejemplo

```
 video_film
BLOB (2G)
```

cualquier columna de una tabla pudiendo seleccionar su valor, insertarlo, eliminarlo y modificarlo. Aunque no todas las operaciones de SQL estan permitidas sobre este tipo de dato.

Datos compuestos A los tipos de datos compuestos o colecciones de SQL se incorporan los tipos ARRAY y ROW que permiten almacenar directamente en una columna colecciones de datos. Esta posibilidad desde cierto punto de vista rompe con la 1era Forma Normal que debe ser cumplida en el modelo relacional, pero puede considerarse

que el valor es atómico conceptualmente, aunque pueda descomponerse. ARRAY permite definir un atributo de una tabla como una colección de valores de uno de los tipos de datos existente de forma que pueda descomponerse, por ejemplo los días de la semana

```
 dia_semana VARCHA(10)
ARRAY[7]
```

en una columna de la tabla. Por ejemplo se define el siguiente atributo en una tabla estudinate

```
 dirección
ROW(calle VARCHAR, numero INTEGER, cod_post VARCHAR(5))
```

estudiante de la siguiente forma:

```
e.direccion.numero
```

Tipos definidos por el usuario Una de las facilidades incorporadas por el SQL:1999 es la posibilidad de definir Tipos Distintos (Distinct Type), estos tipos se definen por medio de la sentencia CREATE TYPE y el SQL:1999 incorpora la prohibición de que dos tipos diferentes se puedan comparar entre si o realizar cualquier tipo de operación, aunque estén definidos sobre el mismo tipo de dato. De esa forma se tiene, por ejemplo, los atributos *edad* y *años de experiencia* no se pueden comparar aunque los dos estén definidos sobre el tipo de datos INTEGER. Siempre el usuario tendrá la posibilidad de utilizar una función CAST para forzar la comparación. Además, existen los tipos estructurados definidos por el usuario debido a su importancia serán analizados en los próximos subtemas.

Tipos de datos estructurados definidos por el usuario. La principal facilidad que brinda SQL:1999 para dar soporte a la orientación a objeto son los tipos estructurados definidos por el usuario (structured user defined type), como su nombre lo indica estos tipos de datos contienen una estructura y sus principales características son:

- Su estructura puede estar formada por uno o varios atributos definidos sobre cualquiera de los tipos de datos soportados por el lenguaje.

- El comportamiento de la estructura se soporta en método, funciones y procedimientos.
- El acceso a los atributos es por medio de las funciones *set* y *get* con lo cual se logra encapsular los datos.
- La comparación de sus valores solo puede realizarse por medio de funciones definidas por el usuario.
- Puede participar de jerarquías de tipos donde los subtipos tienen todos los atributos y rutinas de los supertipos mas los nuevos atributos y rutinas que se quieran adicionar al subtipo.

Se pueden encontrar en SQL:1999 dos formas diferentes de utilizar los tipos de datos estructurados definidos por el usuario, también conocidos como Tipos de Datos Abstractos (TDA), en forma de *tipos de tupla* o como propiamente un *tipo de dato*.

La diferencia fundamental entre los *tipos tupla* y los *tipos dato* propiamente es que los *tipos tupla* no contemplan en absoluto el encapsulamiento, inherente a la orientación a objetos. Al no permitir el encapsulamiento se puede acceder al valor de los datos a través de métodos públicos no propios. Consultar no es peligroso para la integridad de una clase pero si modifican los elementos de la misma.

TDA tipo tupla. La definición de un *tipo tupla* se realiza utilizando la sentencia CREATE ROW TYPE y a partir de este tipo de datos se puede definir una nueva tabla o utilizar el nuevo tipo como dominio de un atributo en una tabla o en otro TDA. La identidad de objetos propia de los lenguajes orientados a objetos se incorpora en SQL:1999 a través del concepto de referencia, este concepto es fundamental para la utilización de los TDA.Una referencia es un puntero a un tipo de constructor que puede ser una fila *(row)*, una colección *(collection)* o un tipo de dato *(data type)*.

Todo tipo tupla T genera un tipo asociado $REF(T)$ que no es más que un tipo puntero a las tuplas de T. La referencia tiene la misma funcionalidad de la identidad de un objeto, de manera que si se quiere incluir dentro de otro tipo. Un atributo de tipo T, se utilizará $REF(T)$, como tipo de dicho atributo. Esto sirve también para mantener la integridad referencial cuando se trabaja con tablas.

Por ejemplo, si se quiere incluir en los datos de un actor, su mejor película y además tener una relación que recoja la actuación de cada actor en cada película tendremos que hacer las siguientes declaraciones, ejemplo C.1:

Ejemplo C.1. *Creando ROW TYPE.*

```
CREATE ROW TYPE TDireccion (calle CHAR (50), ciudad CHAR (50));

CREATE ROW TYPE Película (titulo CHAR (30), año INTEGER, enColores
BIT (1));

CREATE ROW TYPE Actor (nombre CHAR (30), direccion TDireccion,
mejorpelicula REF (Pelicula));

CREATE ROW TYPE ActorPelicula (actor REF (Actor),
 pelicula REF (Pelicula));

CREATE TABLE Peliculas OF TYPE Pelicula;

CREATE TABLE Actores OF TYPE Actor;

CREATE TABLE Actores_en_Peliculas OF TYPE ActorPelicula;
```

Se puede notar que la declaración de tipos tupla como el tipo *actor* tiene como atributo un objeto de tipo *TDireccion*. El objeto *ActorPelicula* tiene un tributo que es una referencia a una tupla de tipo *Actor* y otra de tipo *Película*.

Una vez que se acepta la posibilidad de definir un atributo en un *tipo* como una referencia a otro *tipo*, se establece la manera de acceder a los elementos del tipo referenciado. Esto se hace mediante el operador$\rightarrow$. Es decir si x es una referencia a una tupla t y a es un atributo de t, entonces $x \rightarrow a$ es el valor de a en el la tupla t.

Se ha comentado que las referencias se hacen con respecto a un *tipo tupla*; pero es posible que un *tipo tupla* esté asociado a varias tablas e interese que un determinado atributo haga referencia sólo a las instancias del tipo pertenecientes a una determinada tabla. Esto se hace especificando el ámbito de un atributo mediante la declaración:

```
SCOPE FOR <atributo> IS <relacion>
```

Por ejemplo supongamos que vamos a considerar dos relaciones asociadas al tipo *Actor*:

ActorCine y *ActorTeatro.* La referencia a *Actor* que aparece en la relación *ActorPelicula* es la de *ActorCine.*

Ejemplo C.2. *Creación de tablas a partir de tipos.*

```
CREATE TABLE ActorCine OF TYPE Actor; CREATE TABLE ActorTeatro OF
TYPE Actor;

CREATE TABLE Actores_en_Peliculas OF TYPE ActorPelicula SCOPE FOR
actor is ActorCine;
```

Es un principio general en orientación a objetos que los identificadores de los objetos sean internos al sistema y que no sean accesibles a través de un lenguaje de consulta. No obstante no hay razón en principio por la que no se pueda acceder a dicho identificador interno siempre que no se modifique por parte del usuario. SQL:1999 hace uso de esta posibilidad permitiendo declarar, para cada tabla, un atributo adicional que recoge los valores de dicho identificador interno que puede ser referenciado en una consulta. Dicho atributo juega el papel de clave primaria de la relación en cuestión y se declara mediante la cláusula:

```
VALUES FOR <atributo> ARE SYSTEM GENERATED
```

Entonces el valor de dicho atributo será la referencia a la tupla donde se encuentra. De hecho, este atributo sirve a la vez como clave primaria de la relación y como identificador de objeto para sus tuplas.

Por ejemplo es posible declarar dos atributos generados por el sistema para las tablas actor de cine y película y expresar de una forma más convencional las consultas, del ejemplo C.3.

Ejemplo C.3. *Creación de tablas a partir de tipos.*

```
CREATE TABLE ActorCine OF TYPE Actor VALUES FOR actor_id ARE SYSTEMS
GENERATED; CREATE TABLE Peliculas OF TYPE Pelicula VALUES FOR
pelicula_id ARE SYSTEMS GENERATED; SELECT Peliculas.titulo FROM
Actores_en_Peliculas, ActorCine, Peliculas WHERE
```

```
Actores_en_Peliculas.actor= ActorCine.actor_id AND
Actores_en_Peliculas.pelicula=Peliculas.peliculas_id AND
ActorCine.nombre='Luis Alberto García';
```

Obsérvese que los atributos generados por el sistema se igualan directamente con las referencias que son atributos de la tabla *Actores_en_Peliculas*, y que el uso directo de las referencias es un estilo más orientado a objetos, tal y como aparece en el Ejemplo 1-12 y permite simplificar las consultas.

TDA tipo dato.

Cómo se mencionó existe otra forma de definir clases mediante el concepto de tipo de dato abstracto (*TDA*) de forma que soporte el encapsulamiento. En este caso los objetos pertenecientes a un *TDA tipo dato* se conciben para ser componentes de una tupla, no tuplas por si mismos; pero ello no implica que no puedan tener una estructura de tupla. La diferencia radica en que la extensión de la clase de un *TDA tipo dato* no tiene las características de una tabla como ocurre con los *tipo tupla*. Estos tipos son objetos generales que sólo pueden ser usados como componentes de una tupla, es decir, como dominios para definir un atributo de una tupla. La forma general de definición de un *TDA* es como sigue:

```
1. CREATE TYPE <nombre de tipo> ( 2. lista de atributos y sus tipos
3. declaraciones opcionales de los operadores =, >, <, etc..,
asociados al tipo 4. declaraciones de funciones (métodos) asociados
al tipo );
```

En este esquema general, la línea (3) nos indica que es posible establecer operadores de comparación específicos para el tipo que definimos. Estos operadores se implementarán posteriormente como funciones. La forma genérica de este tipo de declaraciones es:

```
OPERADOR <nombre de la función que implementa el operador>
```

Donde OPERADOR se sustituirá en su caso por EQUALS, LESS THAN, etc. Hay que tener en cuenta que para poder incluir comparaciones asociadas a estos tipos de datos en una sentencia *SQL*, cuando estos aparezcan como valores de un atributo, es necesario tener definidos estos operadores previamente, dado que los conceptos de igualdad y orden son

altamente dependientes del tipo de dato. La línea (4) nos indica que se pueden declarar métodos adicionales. *SQL:1999* proporciona ciertas funciones previamente construidas que no necesitan ser declaradas.

Estas funciones son:

1. Una función constructor que devuelve un nuevo objeto del tipo. Todos los atributos del objeto creado son inicialmente nulos. Si T es el nombre del tipo, $T()$ es la función constructor.

2. Funciones observadoras para cada uno de los atributos del objeto. Si X es una variable de un determinado objeto entonces $A(X)$ es el valor del atributo A para dicha variable. También se suele usar $X.A$ para nombrar dicho valor.

3. Funciones modificadoras. Estas funciones sirven para asignar un valor a cada atributo de un objeto. Se utilizan normalmente en el lado izquierdo de una sentencia de asignación y hay que hacer notar que si se desea un verdadero encapsulamiento es necesario evitar que estas funciones sean de uso público. En *SQL:1999* se utiliza para ello un mecanismo de gestión de privilegios.

Ejemplo C.4. *Definición de tipos y creación de tablas.*

```
CREATE TYPE DireccionTDA ( calle CHAR(50), ciudad CHAR(20), EQUALS
DirEq, LESS THAN DirLT, FUNCTION DirecCompleta (:a1
DireccionTDA)RETURNS char (82), DECLARE EXTERNAL buscaCP CHAR(50)
CHAR(20) RETURNS CHAR (10) LANGUAGE C );

CREATE TABLE Direcciones OF DireccionTDA; INSERT INTO Direcciones
VALUES ('Santiago', 'Granada'); SELECT calle FROM Direcciones WHERE
ciudad = ''Madrid''
```

En este ejemplo se declaran dos operadores EQUALS y LESS THAN, la implementación de dichos operadores es similar a la implementación de cualquier otro método o función del *TDA*. Como declarar funciones con SQL:1999 se muestra en los siguientes subtemas.

Tipos de herencia.

Una de las principales características de la *OO* es que podemos crear toda una jerarquía de clases que heredan una de las otras. Al heredar una clase de otra de un nivel superior

en la jerarquía estamos diciendo que la subclase tendrá los mismos atributos y métodos que la clase padre, en otras palabras que la subclase es del tipo de la clase padre. A la vez se puede definir nuevos atributos y métodos para la subclase o simplemente redefinir los existentes en la clase base.

En *SQL:1999* se garantiza un proceso de herencia entre un subtipo y un tipo. En este caso el subtipo reutiliza todos los atributos del tipo y puede contener atributos adicionales específicos del subtipo (ejemplo C.5 y figura C.2).

Ejemplo C.5. *Herencia en SQL3.*

Cuadro C.1: Tabla *Empleados*

Nombre	**Salario**
Silvia González	500.00

Nombre	**Salario**	**Lenguaje**	**Proyecto**
Esteban Morales	600.00	C++	DOR

Cuadro C.2: Tabla *Programadores*

Nombre	**Salario**	**Región**
Akiko Yokomoto	700.00	Asia

Cuadro C.3: Tabla *Representantes*

Note que *Programadores* y *Representantes* son subtipos de *Empleados*, pues tienen atributos en común, *nombre* y *salario*. La jerarquía de herencia quedaría (figura C.2) y se define con SQL:1999 de la siguiente forma:

```
CREATE TYPE Empleado AS OBJECT ( Nombre VARCHAR2(20), Salario
NUMBER(6,2)) NOT FINAL;

CREATE TYPE Programador UNDER Empleado ( Languaje VARCHAR2(12),
Proyecto VARCHAR2(30));

CREATE TYPE Representante UNDER Empleado( Región  VARCHAR2(30));
```

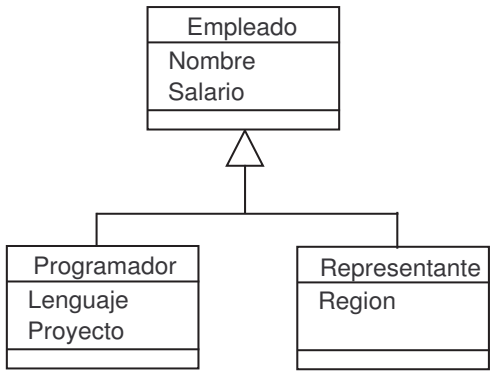


Figura C.2: Jerarquía *Empleado-Programador-Representante*

```
CREATE TABLE Empleados OF Empleado; CREATE TABLE Programadores OF
Programador; CREATE TABLE Representantes OF Representante;

INSERT INTO Empleados  VALUES (Empleado('Silvia González',
 500.00));
INSERT INTO Programadores  VALUES (Programador( 'Esteban Morales',
600.00, 'C++', 'DOR')); INSERT INTO Representante  VALUES
(Representante( 'Akiko Yokomoto', 700.00, 'Asia'));
```

Los subtipos *Programador* y *Representante* heredan todos los atributos de *Empleado*. Una petición a los objetos *Empleados* del tipo *Empleado* significa además una petición para los objetos de los tipos *Programadores* y *Representantes*. Ejemplo C.6:

Ejemplo C.6. *Consulta sobre una jerarquía.*

```
SELECT e.Nombre FROM Empleados e;
```

El resultado será:

Comportamiento de los objetos.

En el SQL:1999 el comportamiento de los objetos se define por medio de métodos, los métodos se definen como funciones con algunas restricciones y mejoras como son:

Nombre
Silvia González
Esteban Morales
Akiko Yokomoto

Cuadro C.4: Resultado **Ejemplo 1-18**

- Los métodos están asociados a un TDA en especifico mientras las funciones no.
- Los métodos se almacenan en el mismo esquema donde se almacena la estructura del TDA al cual esta asociado, mientras que las funciones no tienen una área especifica donde almacenarse.
- Los métodos utilizan cómo tipo de dato por defecto de los argumentos no declarados el TDA para el cual fue creado, mientras que en las funciones esto no existe.
- Las funciones pueden ser polimórficas pero solo una función es seleccionada durante la compilación, en dependencia de los atributos que se le pasan a la función. Los métodos también son polimórficos pero la especificación del argumento por defecto permite seleccionar el método exacto a ser invocado en tiempo de ejecución.

El uso de métodos en los TDA se aproxima bastante a la definición de un objeto verdadero. Algunos *SGBDOR* como *Oracle* y *DB2* permiten definir métodos dentro de la propia definición de los tipos de datos con lo cual permiten desarrollar una aplicación *SQL* similar a una aplicación orientada a objetos. Para acceder a los atributos y métodos de un TDA se pueden utilizar la notación punto (dot notation) en la forma

```
tuplo.atributo o tuplo.metodo
```

el caso de los atributos es posible también utilizar una notación funcional de la forma:

```
atributo(tuplo)
```

Después de una lista de atributos de un *TDA*, se puede añadir una lista de declaración de funciones. La forma de declarar funciones es la siguiente:

```
FUNCTION <nombre> (<argumentos>) RETURNS <tipo>;
```

Las funciones pueden ser de dos tipos: internas y externas. Las funciones externas se escriben en el lenguaje anfitrión (ej. *lenguaje C*) y las funciones internas se escriben en un *SQL* extendido que tiene las siguientes características:

1. Se usa para la asignación.
2. Se pueden declarar variables locales dando su nombre precedido por un signo de exclamación y seguido de su tipo.
3. Se puede acceder a las componentes de una estructura utilizando el punto.
4. Los valores *Booleans* se pueden expresar como en las cláusulas WHERE.
5. Para iniciar y terminar el cuerpo de una función se utiliza BEGIN y END.

Como definir una función interna se muestra en el ejemplo C.7

Ejemplo C.7. *Definición de la función DireccionTDA*

```
FUNCTION DireccionTDA (:ca CHAR(50), :ci CHAR(50)) RETURNS
DireccionTDA; BEGIN a:= DireccionTDA (); a.calle=:ca; a.ciudad=:ci;
RETURN :a; END;

FUNCTION addrEq(:a1 DireccionTDA, :a2 DireccionTDA) RETURNS BOOLEAN
RETURN (:a1.calle=:a2.calle AND :a1.ciudad=:a2=ciudad);

FUNCTION addrLT(:a1 DireccionTDA, :a2 DireccionTDA) RETURNS BOOLEAN
RETURN ((:a1.calle < a2.calle) OR (:a1.calle = a2.calle AND
 :a1.ciudad < :a2.ciudad));

FUNCTION DirecCompleta (a: DireccionTDA) RETURNS char(82) :z
CHAR(10); BEGIN z:=buscaCP(:a.calle,:a.ciudad) RETURN(a:calle||' '||
a:ciudad||' '|| :z); END;
```

Las funciones externas (*buscaCP*) son métodos escritos en un lenguaje anfitrión e la siguiente forma:

```
DECLARE EXTERNAL <nombre><signatura> LANGUAGE 'nombre de lenguaje'
```

Y el siguiente ejemplo declarar la función *buscaCP* antes de ser usada. La función existe externamente escrita en el lenguaje C.

```
DECLARE EXTERNAL buscaCP(CHAR(50),CHAR(20)) RETURNS CHAR (10)
LANGUAGE C
```

C.0.0.3. Principales Gestores Objeto-Relacionales.

En el mercado actual de bases de datos han aparecido un grupo de gestores que dan soporte a SQL:1999 o a extensiones de estos lenguajes. Estas aplicaciones incluyen en cierta medida las características antes expuestas para estos tipos de sistemas. Alguno de ellos son:

1. *Oracle (Copyright©2008, Oracle Corporation, www.oracle.com)*
2. Cloudscape y DB2 *(Copyright©2008 International Business Machines Corporation 2,* www.ibm.com/software/*)*
3. *FirstSQL/J (Copyright ©2001-07 FFE Software, Inc.,* www.firstsql.com)
4. *Cache (Copyright ©InterSystems Corporation 1996-2008,*www.intersystems.com)
5. *TITANIUM (*www.mdbs.com
6. *Valentina (*1998-2008 Paradigma Software Inc www.paradigmasoft.com*)*
7. *PostgreSQL (*Copyright 1996 - 2008 PostgreSQL Global Development Group www. postgresql. org*)*

Otros productos *SGBDOR* puede consultarse en
www.service-architecture.com/products/object-relational_mapping.html

Apéndice D

PostgreSQL

La investigación descrita en esta memoria utiliza a PostgreSQL como gestor de bases de datos, por estas razones algunas de sus principales características serán comentadas. El sistema Gestor de Bases de Datos *PostgreSQL* fue desarrollado originalmente en el *Departamento de Ciencia de la Computación de la Universidad de California en Berkeley.* Este sistema fue pionero en la incorporación de los conceptos de bases de datos objeto - relacional, ofreciendo soporte para los lenguajes *SQL92* y *SQL3*, integridad de transacciones y extensibilidad de tipos de datos.

Actualmente es un descendiente de dominio público y código abierto del código original de *Berkeley.* Está basado en un sistema de bases de datos relacional pero potenciado sustancialmente con la incorporación de conceptos como: clases, herencia, tipos y funciones, como vía para que los usuarios puedan extender fácilmente el sistema.

Además *PostgreSQL* permite el uso de restricciones (*constraints*), disparadores *(triggers)*, reglas *(rules)* e integridad transaccional. Todas estas características colocan a *PostgreSQL* en la categoría de Bases de Datos Objeto-Relacional.

Actualmente su desarrollo está en mano de una amplia comunidad de programadores que constantemente incorporan al mercado nuevas versiones. La última versión para esta fecha es la 8.3.x Como regla se pone a disposición del público nuevas versiones en un periodo de 6 meses. Todos estos años de desarrollo han convertido a *PostgreSQL* en el gestor de bases de datos de código abierto más avanzado hoy en día, ofreciendo control de concurrencia multi-versión, soportando casi todas las sintaxis *SQL* y contando también con un amplio conjunto de enlaces con lenguajes de programación *(C, C++, Java, Perl, Tcl, Python).*

Este sistema está en constante desarrollo e implementación y en cada versión surgen nuevas funciones, nuevas características o se solucionar inconvenientes de anteriores versiones. Esta propia dinámica de desarrollo provoca el principal inconveniente que desde nuestro punto de vista podemos señalar a *PostgreSQL*:

1. Una pobre documentación, desactualizada en muchos casos, con muy poca información y en ocasiones nula información sobre algunas de las características soportadas por *PostgreSQL*.

2. Limitaciones en algunas de sus características provocadas por implementaciones que no tienen en cuenta todas las posibilidades.

Debido a estas dos limitaciones nuestro trabajo con esta herramienta se basó, en lo fundamental, en la experimentación de sus características y el estudio de sus posibilidades. Es justo señalar que existe una gran comunidad de *PostgreSQL* que mantienen listas de discusión las cuales han sido de gran valor para el trabajo.

Principales características.

El *PostgreSQL* incorpora todas las características de un Sistema Gestor de Bases de Datos Relacional y además un conjunto de características que responden al Modelo Objeto-Relacional. En este subtópico veremos de forma muy general las principales características que incorpora *PostgreSQL* y que responden a las declaración de *Tercera Generación de Sistemas de Bases de Datos* planteadas por Stonebraker [Sto89].

1. **Identificación de objetos *(Object Identification)*:** El identificador de objetos, *OID* según sus siglas en Inglés, permite identificar un objeto en todo el esquema de la base de datos. El usuario tiene la posibilidad de elegir, al momento de crear una tabla, si desea que los objetos creados para esa tabla contengan un atributo *OID* que lo identifique o no. Aunque puede ser usado como claves de la tabla, por identificar de forma única cada uno de los objetos, se recomienda que el usuario defina sus propias claves en dependencia del problema modelado [DD03]. El *OID* es un número positivo representado en 32 bits. Este valor en la tabla siempre está oculto y para poder acceder a el se debe precisar de forma explícita en una sentencia SELECT. Es posible crear en una tabla una columna de tipo *OID* para almacenar de forma explícita una referencia a otro objeto, por lo general en otra tabla. *PostgreSQL* soporta operaciones lógicas entre valores OID y valores enteros (*int4*).

2. **Uso de tipos definidos por el usuario**: *PostgreSQL* soporta dos tipos de datos; los tipos básicos y los tipos definidos por el usuario. Entre los tipos básicos se incluyen tipos de tipo numérico entero, numérico de punto flotante, cadenas de caracteres, fecha, etc. Los tipos definidos por el usuario permiten que el usuario defina sus propios tipos y los pueda utilizar como dominio de algún atributo en una tabla. El usuario podrá definir sus propios tipos utilizando el comando CREATE TYPE. Al momento de crearlos el usuario deberá definir un nombre para el tipo, un tamaño en bit y las funciones de entrada y salida del tipo. Estas funciones tienen el objetivo de transformar un valor cadena de caracteres en una estructura de datos definida para el tipo y la operación inversa de forma que se garantice la interacción de *PostgreSQL* con el tipo creado. Estas funciones se implementan en lenguaje *C*. Además de este tipo, es posible crear un tipo en forma de registro (CREATE TYPE AS) pero este tipo sólo podrá ser utilizado como valor de retorno de una función y no como dominio de una tabla en particular.

3. **Uso de vectores**: Es posible definir columna de tablas como vectores multidimencionales tanto de tipos base como de tipos definidos por el usuario. De esta forma podemos almacenar en nuestra tablas valores no atómicos rompiendo con la normalización en la segunda generación de sistemas de bases de datos.

4. **Funciones definidas por el usuario**: El programador de aplicaciones podrá definir sus propias funciones en el modelo de la base de datos. Estas funciones podrán ser utilizadas dentro de las sentencias de SQL como funciones propias del lenguaje. *PostgreSQL* define tres tipos de funciones:

   - Funciones de lenguaje de consulta (escritas en SQL)
   - Funciones de lenguaje procedural (escrita en uno de los lenguajes procedurales soportados por PostgreSQL: pl/Pgslq, pl/Tcl, pl/Perl, pl/Python)
   - Funciones internas.
   - Funciones en lenguaje C.

   Cada uno de estos tipos de funciones puede tomar como argumento un tipo base, un tipo creado por el usuario o una combinación de tipos. Además cada tipo de función puede retornar tanto un tipo base como un tipo definido por el usuario.

5. **Operadores:** *PostgreSQL* dispone de operadores unarios izquierdos y derechos y de operadores binarios. Sin embargo los operadores pueden ser sobrecargados. Es posible definir un operador con el mismo nombre y que su resultado dependa del tipo de dato de los operando. Un papel fundamental en la base de datos juegan los operadores de *casting* permitiendo transformar datos de un tipo en otros. Todos los operadores definidos en el modelo tendrán asociados una función implementada por cualquiera de las vías antes mostradas, tomando como parámetros los operando del operador y retornando el valor que se desee en la operación.

6. **Funciones de agregación**: De la misma forma podremos definir nuestras propias funciones de agregación. Estas funciones actuarán sobre un determinado tipo de dato que será especificado en su definición.

7. **Herencia entre tablas**: Es posible definir una jerarquía entre las tablas, especificando relaciones de herencia entre ellas. Cuando una tabla hereda de otra incorpora automáticamente todos los atributos de la supertabla. Una tabla puede heredar de una, ninguna o varias tablas y al especificarse una consulta sobre una tabla se pueden consultar sus objetos y los objetos de todas las subtablas.

8. **Reglas en el sistema**: Se pueden definir un conjunto de reglas asociadas a una tabla o una vista a partir del cumplimiento de una condición. Las reglas pueden ser definidas para la ocurrencia de los eventos SELECT, INSERT, DELETE, UPDATE. Al ejecutarse estas reglas pueden afectar de forma simultánea varios objetos.

9. **Definición de Triggers**: igualmente para los eventos SELECT, INSERT, DELETE, UPDATE de una tabla es posible definir *disparadores*. La principal diferencia con las reglas es que estos se ejecutan en un solo objeto por vez.

Un estudio detallado sobre todas las características de *PostgreSQL* y la forma de implementación se puede encontrar en [DD03, Pos08]. Los ejemplos y el modelo propuesto en esta memoria fueron implementados totalmente sobre la versión 8.0 y tienen compatibilidad total con las versiones posteriores de *PostgreSQL*. Una discusión sobre el alcance de las características objeto-relacionales de *PostgreSQL* con respecto al estacar SQL:1999 fijado se presenta en el capítulo [**?**].

Apéndice E

Resumen del modelo difuso para *PostgreSQL.*

Para facilitar el trabajo con el modelo se presenta un resumen de las principales funciones de *pg4DB*, tabla (E.1).

Cuadro E.1: Resumen de las funciones del modelo propuesto.

Objeto	**Tema**	**Descripción**	**Ref.**
Atomic_fuzzy_value (varchar)	Dominios atómicos.	Insertar referencias a objetos simples definidos sobre dominios atómicos.	Tabla 4.14
AVGC (oid,oid)	Dominios conjuntivos.	Determina el promedio al combinar la inclusión entre dos subconjuntos.	Apartado 4.7.2
COG (oid,oid)	Dominios conjuntivos.	Determina el grado de consistencia entre dos subconjuntos.	Apartado 4.7.2
comparison_profile (varchar, varchar, varchar, varchar, varchar)	Objetos complejos y tipos difusos.	Definir una perspectiva de comparación para un objeto complejo o para un tipo difuso.	Tabla 4.29

Objeto	**Tema**	**Descripción**	**Ref.**
Complex_fuzzy_set (varchar, varchar, varchar)	Dominios conjuntivos sobre objetos.	Insertar referencias a objetos simples definidos sobre dominios conjuntivos con refrencial sobre objetos.	Tabla 4.19
Create_AtomicinObjetc (varchar, varchar)	Dominio atómico referencial obejetos.	Crear dominio atómicos con referencial finito sobre objetos.	Tabla 4.9
Create_Conjutive_Extension (varchar)	Dominios conjuntivos.	Crear dominios conjuntivo con referencial de valores.	Tabla 4.7
Create_Conjutive_Extension (varchar, varchar)	Dominios conjuntivos.	Crear dominios conjuntivo con referencial de objetos.	Tabla 4.10
Create_Domain_DF (varchar, varchar, varchar, varchar)	Dominios atómicos finitos sobre valores.	Crear dominios atómico con referencial finito sobre valores.	Tabla 4.6
Create_Domain_DT (varchar, varchar, varchar)	Dominios atómicos continuos.	Crear dominios atómico con referencial continuo.	Tabla 4.4
Create_Domain_DT (varchar, varchar, varchar, numeric)	Dominios atómicos continuos.	Crear dominios atómico con referencial continuo especificando el parametro much.	Tabla 4.5
Create_Domain_WR (varchar, varchar, varchar)	Dominios atómicos sin representación semántica.	Crear dominios atómico sin representación semántica.	Tabla 4.3
Create_fuzzy_type (varchar, varchar[])	Tipos Difusos	Crear en el modelo un tipo difuso.	Tabla 4.34
Create_table_domain (varchar)	Objetos complejos.	Crear el dominio referente a un objeto complejo.	Tabla 4.8

Objeto	**Tema**	**Descripción**	**Ref.**
EXI(oid,oid)	Objetos.	Calcular el parecido entre dos objetos, considerando numero de orness 1.0.	Apartado 4.7.3
f (numeric, numeric)	Consultas.	Definición de una constante difusa, valor difuso aproximado.	Tabla 4.27
f (numeric, numeric, numeric,numeric)	Consultas.	Definición de una constante difusa, distribución de posibilidad trapezoidal.	Tabla 4.25
FEQ(oid, oid, numeric)	Objetos complejos y tipos difusos.	Comparar objetos de un tipo disuso especificando el numero orness.	Apartado 4.9.3
FEQ(oid, oid, numeric, numeric)	Tipos difusos.	Comparar objetos de un tipo difuso especificando el valor al comparar valores nulos y el número orness.	Apartado 4.9.3
FEQ(oid, oid, varchar)	Objetos complejos y tipos difusos.	Especificar la perspectiva al comparar objetos complejos.	Tabla 4.30
FEQ(oid,oid)	Objetos.	Calcular el parecido entre dos objetos.	Apartado 4.7
FEQ(oid,varchar)	Objetos simples.	Calcular el parecido entre un objetos y un valor.	Apartado 4.7
FEQ(varchar,oid)	Objetos simples.	Calcular el parecido entre dos objetos y un valor.	Apartado 4.7
FGEQ (oid,oid)	Dominio atómico con referencia continuo.	Compara dos objetos buscan si el primero es Posiblemente Mayor o igual Difuso que el segundo.	Tabla 4.24

Objeto	**Tema**	**Descripción**	**Ref.**
FGEQ (oid,varchar)	Dominio atómico con referencia continuo.	Compara si el objeto es Posiblemente Mayor o igual Difuso que un valor.	Tabla 4.24
FGEQ (varchar,oid)	Dominio atómico con referencia continuo.	Compara si un valor es Posiblemente Mayor o igual Difuso que un objeto.	Tabla 4.24
FGT (oid,oid)	Dominio atómico con referencia continuo.	Compara dos objetos buscan si el primero es Posiblemente Mayor Difuso que el segundo.	Tabla 4.24
FGT (oid,varchar)	Dominio atómico con referencia continuo.	Compara si el objeto es Posiblemente Mayor Difuso que un valor.	Tabla 4.24
FGT (varchar, oid)	Dominio atómico con referencia continuo.	Compara si un valor es Posiblemente Mayor Difuso que un objeto.	Tabla 4.24
fi (numeric, numeric)	Consultas.	Definición de un intervalo numérico.	Tabla 4.26
FLEQ (oid,oid)	Dominio atómico con referencia continuo.	Compara dos objetos buscan si el primero es Posiblemente menor o igual difuso que el segundo.	Tabla 4.24
FLEQ (oid,varchar)	Dominio atómico con referencia continuo.	Compara si el objeto es Posiblemente menor o igual difuso que un valor.	Tabla 4.24

Objeto	**Tema**	**Descripción**	**Ref.**
FLEQ (varchar,oid)	Dominio atómico con referencia continuo.	Compara si un valor es Posiblemente menor o igual difuso que un objeto.	Tabla 4.24
FLT (oid,oid)	Dominio atómico con referencia continuo.	Compara dos objetos buscan si el primero es Posiblemente Menor Difuso que el segundo.	Tabla 4.24
FLT (oid,varchar)	Dominio atómico con referencia continuo.	Compara si el objeto es Posiblemente Menor Difuso que un valor.	Tabla 4.24
FLT (varchar,oid)	Dominio atómico con referencia continuo.	Compara si un valor es Posiblemente Menor Difuso que un objeto.	Tabla 4.24
get_resemblance (oid, oid)	Objetos.	Calcular el parecido entre dos objetos.	Tabla 4.23
get_resemblance (oid, varchar)	Objetos simples.	Calcular el parecido entre un objetos y un valor.	Tabla 4.23
get_resemblance (varchar, oid)	Objetos simples.	Calcular el parecido entre dos objetos y un valor.	Tabla 4.23
GIN (oid,oid)	Dominios conjuntivos.	"Determina el grado de inclusión teniendo en cuenta la semejanza entre dos subconjuntos."	Apartado 4.7.2
GIS (oid,oid)	Dominios conjuntivos.	Determina el grado de inclusión guiado por semejanza entre dos subconjuntos.	Apartado 4.7.2
IS (float, varchar)	Consultas.	Comparar un número con una distribución de posibilidad trapezoidal.	Tabla 4.28

Objeto	**Tema**	**Descripción**	**Ref.**
MGT (oid,oid)	Dominio atómico con referencia continuo.	Compara dos objetos buscan si el primero es Posiblemente mucho mayor difuso que el segundo.	Tabla 4.24
MGT (oid,varchar)	Dominio atómico con referencia continuo.	Compara si el objeto es Posiblemente mucho mayor difuso que un valor.	Tabla 4.24
MGT (varchar,oid)	Dominio atómico con referencia continuo.	Compara si un valor es Posiblemente mucho mayor difuso que un objeto.	Tabla 4.24
MLT (oid,oid)	Dominio atómico con referencia continuo.	Compara dos objetos buscan si el primero es Posiblemente mucho manor difuso que el segundo.	Tabla 4.24
MLT (oid,varchar)	Dominio atómico con referencia continuo.	Compara si el objeto es Posiblemente mucho menor difuso que un valor.	Tabla 4.24
MLT (varchar,oid)	Dominio atómico con referencia continuo.	Compara si un valor es Posiblemente mucho menor difuso que un objeto.	Tabla 4.24
new_att_derive (varchar, varchar, varchar[], numeric, varchar, boolean, varchar)	Objetos complejos y tipos difusos.	Definir un atributo inferido.	Tabla 4.32
new_object_fuzzy_type (varchar, varchar)	Tipos Difusos	Crear un objeto de un tipo difuso.	Tabla 4.35

Objeto	**Tema**	**Descripción**	**Ref.**
NFGEQ (oid,oid)	Dominio atómico con referencia continuo.	Compara dos objetos buscan si el primero es Necesariamente Mayor o igual Difuso que el segundo.	Tabla 4.24
NFGEQ (oid,varchar)	Dominio atómico con referencia continuo.	Compara si el objeto es Necesariamente Mayor o igual Difuso que un valor.	Tabla 4.24
NFGEQ (varchar,oid)	Dominio atómico con referencia continuo.	Compara si un valor es Necesariamente Mayor o igual Difuso que un objeto.	Tabla 4.24
NFGT (oid,oid)	Dominio atómico con referencia continuo.	Compara dos objetos buscan si el primero es Necesariamente Mayor Difuso que el segundo.	Tabla 4.24
NFGT (oid,varchar)	Dominio atómico con referencia continuo.	Compara si el objeto es Necesariamente Mayor Difuso que un valor.	Tabla 4.24
NFGT (oid,varchar)	Dominio atómico con referencia continuo.	Compara si un valor es Necesariamente Mayor Difuso que un objeto.	Tabla 4.24
NFLEQ (oid,oid)	Dominio atómico con referencia continuo.	Compara dos objetos buscan si el primero es Necesariamente menor o igual difuso que el segundo.	Tabla 4.24
NFLEQ (oid,varchar)	Dominio atómico con referencia continuo.	Compara si un valor es Necesariamente menor o igual difuso que un objeto.	Tabla 4.24

Objeto	**Tema**	**Descripción**	**Ref.**
NFLEQ (varchar,oid)	Dominio atómico con referencia continuo.	Compara si el objeto es Necesariamente menor o igual difuso que un valor.	Tabla 4.24
NFLT (oid,oid)	Dominio atómico con referencia continuo.	Compara dos objetos buscan si el primero es Necesariamente Menor Difuso que el segundo.	Tabla 4.24
NFLT (oid,varchar)	Dominio atómico con referencia continuo.	Compara si el objeto es Necesariamente Menor Difuso que un valor.	Tabla 4.24
NFLT (varchar,oid)	Dominio atómico con referencia continuo.	Compara si un valor es Necesariamente Menor Difuso que un objeto.	Tabla 4.24
NMGT (oid,oid)	Dominio atómico con referencia continuo.	Compara dos objetos buscan si el primero es Necesariamente mucho mayor difuso que el segundo.	Tabla 4.24
NMGT (oid,varchar)	Dominio atómico con referencia continuo.	Compara si el objeto es Necesariamente mucho mayor difuso que un valor.	Tabla 4.24
NMGT (varchar,oid)	Dominio atómico con referencia continuo.	Compara si un valor es Necesariamente mucho mayor difuso que un objeto.	Tabla 4.24
NMLT (oid,oid)	Dominio atómico con referencia continuo.	Compara dos objetos buscan si el primero es Necesariamente mucho menor difuso que el segundo.	Tabla 4.24

Objeto	**Tema**	**Descripción**	**Ref.**
NMLT (oid,varchar)	Dominio atómico con referencia continuo.	Compara si el objeto es Necesariamente mucho menor difuso que un valor.	Tabla 4.24
NMLT (varchar,oid)	Dominio atómico con referencia continuo.	Compara si un valor es Necesariamente mucho menor difuso que un objeto.	Tabla 4.24
OPT(oid,oid)	Tipos Difusos	Comparar objetos de un tipo difuso considerando los valores nulos desde un punto de vista optimista.	Apartado 4.9.3
OPT(oid,oid,numeric)	Tipos Difusos	Comparar objetos de un tipo difuso considerando los valores nulos desde un punto de vista optimista, especificando el numero orness.	Apartado 4.9.3
PES(oid,oid)	Tipos Difusos	Comparar objetos de un tipo difuso considerando los valores nulos desde un punto de vista pesimista.	Apartado 4.9.3
PES(oid,oid,numeric)	Tipos Difusos	Comparar objetos de un tipo difuso considerando los valores nulos desde un punto de vista pesimista, especificando el numero orness.	Apartado 4.9.3
set_<nombre del dominio>(varchar)	Dominios atómicos.	Constructor para los objetos simples definidos sobre dominios atómicos.	Tabla 4.14

Objeto	**Tema**	**Descripción**	**Ref.**
set_<nombre del dominio>(varchar, varchar)	Dominio conjuntivos y dominio atomico con referencial finito de objetos.	Constructor para los objetos simples definidos sobre dominios conjuntivos..	Tabla 4.18 y 4.22
set_<nombre del tipo difuso>(oid)	Tipos Difusos	Constructor para los objetos definidos sobre tipos difusos.	Tabla 4.36
set_<nombre del tipo difuso>(varchar)	Tipos Difusos	Constructor para los objetos definidos sobre tipos difusos.	Tabla 4.36
simples_fuzzy_set (varchar, varchar, varchar)	Dominios conjuntivos con referencial de valores.	Insertar referencias a objetos definidos sobre dominios conjuntivos con referencial de valores.	Tabla 4.15
UNI(oid,oid)	Objetos.	Calcular el parecido entre dos objetos, considerando numero de orness 0.0.	Apartado 4.7.3

Las principales tablas que dan el soporte a la plantilla pg4DB se resumen en la tabla (E.2).

Cuadro E.2: Resumen de las principales tablas del modelo *pg4DB*.

Tabla	**Descripción**	**Ref.**
tabstract_domain	Soporte para la definición de objetos simples definidos sobre dominios atómicos.	Apartado 4.3.1
tdomain	Soporte para almacenar la definición de dominios atómicos difusos.	Apartado 4.3
tdomain_cf	Soporte para la definición de objetos definidos sobre dominios con un significado conjuntivo.	Apartado 4.4
tdomain_df	Almacenar la definición de dominios atómicos con representación semántica, representados por conjuntos finitos.	Apartado 4.3
tdomain_dt	Almacenar la definición de dominios atómicos con representación semántica y con distribuciones de posibilidad representada por funciones.	Apartado 4.3
tdomain_wr	Almacenar la definición de dominios atómicos sin representación semántica asociada.	Apartado 4.3
tdomain_profile	Almacena los metadatos de las perspectivas de comparación entre obejtos.	Apartado 4.7.5
tfuzzyobject	Soporte para la definición de objetos.	Apartado 4.5
tfuzzyrule	Almacena los meta datos de los atributos inferidos.	Apartado 4.8
tfuzzytable	Soporte para la definición de objetos complejos.	Apartado **??**

Apéndice F

Elementos de instalación de pg4DB.

La plantilla *pg4DB* se distribuye en un script de *PostgreSQL* que debe ser previamente instalado en el servidor de bases de datos. Para esto el usuario dispone de un paquete de instalación compuesto por los siguientes ficheros:

fuzzy_schema.sql contiene un script que crea el esquema, objetos y funciones que dan soporte al uso de atributos difusos y objetos con atributos difusos en la base de datos. El script está codificado en PLpg/SQL.

pgfordb-init.bat es un fichero para la instalación automática, en el sistema operativo Windows, de *pg4DB* en un servidor *PostgreSQL*.

pgfordb-init es un fichero para la instalación automática, en el sistema operativo GNU/Linux, de *pg4DB* en un servidor *PostgreSQL*.

BSD license.txt describe licencia de desarrollo y distribución, de pg4DB. pg4DB se desarrolló sobre *PostgreSQL* por lo que incluye y respeta sus licencias.

Installation notes.txt conjunto de notas que describen el proceso de instalación de *pg4DB*.

El usuario debe ejecutar los ficheros pgfordb-init.bat o pgfordb-init, en dependencia del sistema operativo que tenga instalado. El sistema de pedirá una calve de un usuario

administrador de la base de datos y automáticamente instalará en el servidor de bases de datos la plantilla con todo el soporte para el trabajo con bases de datos difusas.

Apéndice G

Estructura de la base de datos ejemplo.

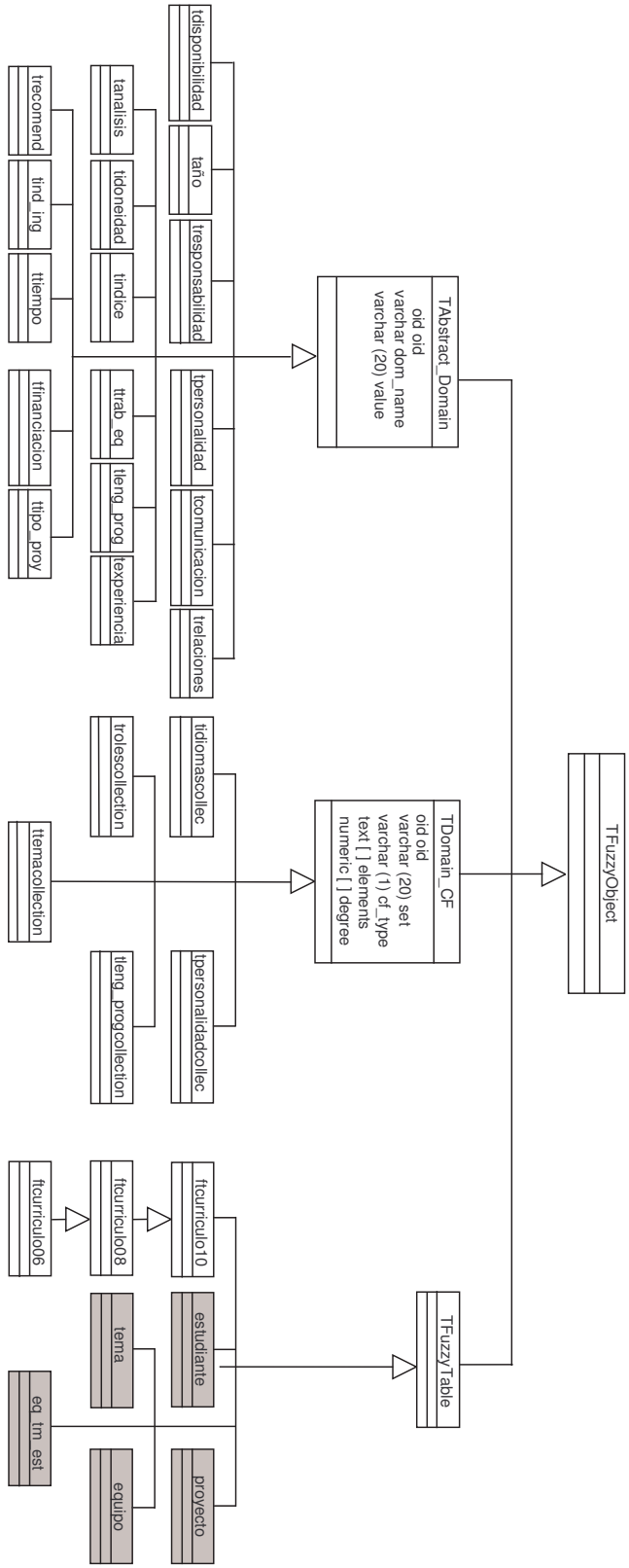


Figura G.1: Estructura creada para el ejemplo del capítulo 5.

Apéndice H

Objetos insertados en el ejemplo.

H.1. Estudiantes insertados en la base de datos

H.2. Patrón de estudiante jefe del grupo de desarrollo.

```
INSERT INTO testudiante VALUES(
'00000000001',
'JefeTema_sistcontbweb',
'01/01/2009',
'',
'',
'SI',
'',
'',
set_ind_ing('BUENO'),
set_recomend('BUENA'),
set_idoneidad('1'),
set_curriculo (
    set_disponibilidad('ALTA'),
    set_anio('MEDIOS'),
    set_responsabilidad('MEDIA'),
    set_personalidadcollection('{HONESTO,INDEPENDIENTE,TRABAJADOR,
        CREATIVO}','{0.7,0.4,0.9,0.2}'),
```

Cuadro H.1: Estudiantes insertados en la base de datos 1/3

Atributos	Est1	Est2	Est3	Est4
DNI	86061723164	87123026023	86052121698	86110221788
nombre	Michel Batista	Carlos Ferra	Marlen Machado	Juan Vinaldell
f_nacimiento	17/06/1986	30/12/1987	21/05/1986	02/11/1986
direccion	C.16-A N.11 E/5 Y 7 R.FREYRE	CTRA GIBARA 503 RPTO. LUZ	31 EDIF.6-B AP.5 ESQ.4 RP.LENI	RASTRO 39 AGRAMONTE Y GARAYAL
municipio	Freyre	Gibara	Holguin	Holguin
becado	SI	SI	NO	NO
horario	Mañana	Mañana	Tarde	Tarde
alumno_ayud	NO	NO	SI	SI
indice de ingreso	94,44	81,77	93,24	84,31
recomendación	BUENA	BUENA	REGULAR	BUENA
clasificación	MUY_ALTA	BAJA	ALTA	BAJA
disponibilidad	ALTA	MEDIA	ALTA	MUY_ALTA
año	1	1	2	2
responsabilidad	MEDIA	MEDIA	POCA	MEDIA
personalidad	0.6/HONESTO+ 0.5/INDEPENDIENTE+ 0.8/TRABAJADOR+ 0.7/CREATIVO	0.6/HONESTO+ 0.7/INDEPENDIENTE+ 0.9/TRABAJADOR+ 0.4/CREATIVO	0.7/HONESTO+ 0.6/INDEPENDIENTE+ 0.5/TRABAJADOR+ 0.8/CREATIVO	0.9/HONESTO+ 0.45/INDEPENDIENTE+ 0.65/TRABAJADOR+ 0.8/CREATIVO
idioma	1.0/español+ 0.8/ingles	1.0/español+ 0.3/ingles	1.0/español+ 0.6/ingles	1.0/español+ 0.75/ingles
comunicación	ALTA	MUY_ALTA	ALTA	MUY_ALTA
relaciones personales	ACEPTABLES	BUENAS	ACEPTABLES	MUY_BUENAS
capacidad de analisis	MEDIA	MEDIA	MEDIA	ALTA
idoneidad	MEDIA	MEDIA	MEDIA	ALTA
indice general			3,94	3,82
indice matatemática			3,10	3,75
indice física			3,68	3,42
indice idioma			4,21	4,83
indice sistemas digitales			4,58	4,14
indice ciencias empresariales			3,58	4,01
indice matemática aplicada			3,65	3,40
indice inteligencia artífical			4,58	3,86
indice programación			3,15	3,46
indice informática industrial			4,94	3,47
trabajo en equipo			MEDIA	POCA
trabajo independiente			ALTA	ALTA
rendimiento			BAJA	BAJA
indice ingeniería de software				
roles				
lenguajes de programación				
experiencia apliaciones desktop				
experiencia apliaciones web				
experiencia apliaciones mutimedia				
experiencia apliaciones empresariales				
experiencia apliaciones educativas				

Cuadro H.2: Estudiantes insertados en la base de datos 2/3

Atributo	Est5	Est6	Est7	Est8
DNI	85072422382	85120722363	80050123543	82090326066
nombre	Yurieski Arcaya	Miguel Aguilera	Damian Calderon	Rafale Barata
f_nacimiento	24/07/1985	07/12/1985	01/05/1980	03/09/1982
direccion	18 SUR N.508 E/5 Y 7 U.NORIS	C.35 N.1402 E/14A Y 16 S.GERMA	EDIF.39 ESC.D APTO.9 RP.ABEL S	3RA N.25 E/14 Y AV CAP URB
municipio	Urbano	Urbano	Holguin	Holguin
becado	SI	SI	NO	NO
horario	Mañana	Mañana	Tarde	Mañana
alumno_ayud	NO	NO	SI	NO
indice de ingreso	93,46	96,82	92,21	96,32
recomendación	MUY_BUENA	REGULAR	REGULAR	MUY_BUENA
clasificación	MUY_ALTA	ALTA	MEDIA	MUY_ALTA
disponibilidad	ALTA	MEDIA	ALTA	MEDIA
año	3	3	4	5
responsabilidad	MUCHA	MEDIA	MEDIA	MUCHA
personalidad	0.6/HONESTO+ 0.5/INDEPENDIENTE+ 0.6/TRABAJADOR+ 0.7/CREATIVO	0.9/HONESTO+ 0.6/INDEPENDIENTE+ 0.6/TRABAJADOR+ 0.85/CREATIVO	0.7/HONESTO+ 0.55/INDEPENDIENTE+ 0.8/TRABAJADOR+ 0.6/CREATIVO	0.65/HONESTO+ 0.8/INDEPENDIENTE+ 0.7/TRABAJADOR+ 0.85/CREATIVO
idioma	1.0/español+ 0.3/ingles	1.0/español+ 0.65/ingles	1.0/español+ 0.9/ingles	1.0/español+ 0.1/ingles
comunicación	POCA	MUY_ALTA	MEDIA	POCA
relaciones personales	EXCELENTE	ACEPTABLES	ACEPTABLES	BUENAS
capacidad de analisis	MEDIA	MEDIA	MEDIA	MEDIA
idoneidad	MEDIA	MEDIA	MEDIA	BAJA
indice general	4,11	4,17	3,95	4,27
indice matatemática	3,95	3,73	3,59	4,43
indice física	3,81	4,13	3,99	4,24
indice idioma	3,72	4,46	3,49	4,23
indice sistemas digitales	3,78	3,66	4,50	3,30
indice ciencias empresariales	4,46	4,39	3,70	4,31
indice matemática aplicada	4,98	4,24	4,76	4,31
indice inteligencia artífical	3,76	4,80	3,28	4,50
indice programación	4,64	3,34	3,48	4,28
indice informática industrial	3,59	4,48	4,32	4,18
trabajo en equipo	ALTA	MEDIA	ALTA	ALTA
trabajo independiente	MEDIA	MUY_ALTA	MEDIA	M
rendimiento	BAJA	BAJA	BAJA	BAJA
indice ingeniería de software	3,36	4,51	4,41	4,89
roles				
lenguajes de programación				
experiencia apliaciones desktop	2	2	1	6
experiencia apliaciones web	3	3	2	0
experiencia apliaciones mutimedia	0	0	1	0
experiencia apliaciones empresariales	3	3	1	4
experiencia apliaciones educativas	2	2	2	0

Cuadro H.3: Estudiantes insertados en la base de datos 3/3

Estudiante	Est9	Est10	Est11	Est12
DNI	84012425135	83103119402	85102722975	86011021716
nombre	Odalis Pita	Yandi Fernandez	Anniet Llorente	Susel Tamayo
f_nacimiento	24/01/1984	31/10/1983	27/10/1985	10/01/1986
dirección	C.13 N.77 16 Y 1/ENE SIBONEY	JOSUE PAIS 47 E/MAR.P Y HO ADU	C.30 N.4109-A E.41 Y 45 VE-LASC	C/6TA N.65 E/20 DE MA./INDEP
municipio	Santiago	Holguin	Gibara	Holguin
becado	SI	NO	SI	NO
horario	Mañana	Mañana	Tarde	Mañana
alumno_ayud	SI	SI	SI	SI
indice de ingreso	94,58	83,80	94,38	81,95
recomendación	BUENA	REGULAR	MUY_BUENA	MUY_BUENA
clasificación	MUY_ALTA	BAJA	MUY_ALTA	BAJA
disponibilidad	BAJA	ALTA	MUY_ALTA	CASI_TOTAL
año	5	5	4	3
responsabilidad	MEDIA	MEDIA	MEDIA	MUCHA
personalidad	0.55/HONESTO+ 0.9/INDE-PENDIENTE+ 0.75/TRABAJA-DOR+ 0.7/CREATIVO	0.6/HONESTO+ 0.6/INDEPEN-DIENTE+ 0.7/TRABAJADOR+ 0.6/CREATIVO	0.55/HONESTO+ 0.4/INDE-PENDIENTE+ 0.9/TRABAJA-DOR+ 0.5/CREATIVO	0.7/HONESTO+ 0.9/INDEPEN-DIENTE+ 0.7/TRABAJADOR+ 0.5/CREATIVO
idioma	1.0/español+ 0.75/ingles	1.0/español+ 0.85/ingles	1.0/español+ 0.4/ingles	1.0/español+ 0.6/ingles
comunicación	MEDIA	ALTA	POCA	ALTA
relaciones personales	EXCELENTE	MUY_BUENAS	ACEPTABLES	EXCELENTE
capacidad de analisis	MEDIA	ALTA	MUY_ALTA	MUY_ALTA
idoneidad	BAJA	BAJA	MEDIA	ALTA
indice general	3,95	4,11	4,36	4,05
indice matatemática	3,86	4,56	4,77	4,38
indice física	3,16	4,04	4,23	3,28
indice idioma	4,57	4,50	3,89	4,51
indice sistemas digitales	4,01	3,52	4,87	3,21
indice ciencias empresariales	3,98	4,59	3,73	4,59
indice matemática aplicada	3,48	3,45	4,78	3,69
indice inteligencia artífical	3,43	4,39	3,66	3,78
indice programación	3,40	3,36	4,47	4,13
indice informática industrial	3,87	4,64	4,20	3,24
trabajo en equipo	MEDIA	POCA	MEDIA	MUY_ALTA
trabajo independiente	M	ALTA	MUY_ALTA	ALTA
rendimiento	BAJA	BAJA	BAJA	ALTA
indice ingeniería de software	4,75	3,56	4,97	3,69
roles				
lenguajes de programación				
experiencia apliaciones desktop	3	0	2	2
experiencia apliaciones web	2	7	3	5
experiencia apliaciones mutimedia	1	0	3	0
experiencia apliaciones empresariales	4	3	2	4
experiencia apliaciones educativas	1	0	3	0

Cuadro H.4: Otros datos de los estudiantes insertados

Estudiante	roles	lenguajes de programación
Est1		
Est2		
Est3		
Est4		
Est5	0.7/Programador+ 0.5/Diseñador+ 0.3/Analista	0.65/Java+ 0.2/VisualBasic+ 0.8/C+++ 0.5/JSP
Est6	0.6/Programador+ 0.4/Diseñador+ 0.2/Analista+ 0.8/Probador	0.7/Java+ 0.4/VisualBasic+ 0.85/C++
Est7	0.5/Programador+ 0.7/Diseñador+ 0.7/Analista+ 0.6/Probador	0.7/Java+ 0.2/VisualBasic+ 0.9/C++
Est8	0.45/Programador+ 0.75/Diseñador+ 0.8/Analista+ 0.9/Probador	0.4/Java+ 0.6/VisualBasic+ 0.8/C++
Est9	0.75/Programador+ 0.65/Diseñador+ 0.8/Analista	0.35/Java+ 0.5/VisualBasic+ 0.75/C++
Est10	0.8/Programador+ 0.5/Diseñador+ 0.7/Probador	0.5/VisualBasic+ 0.9/ASP+ 0.75/C++
Est11	0.9/Programador+ 0.4/Diseñador+ 0.5/Analista	0.6/Java+ 0.1/VisualBasic+ 0.9/C+++ 0.8/JSP
Est12	0.5/Programador+ 0.4/Diseñador+ 0.5/Analista+ 0.2/Probador	0.8/Java+ 0.35/VisualBasic+ 0.8/C++

```
    set_idiomacollection('{español,ingles}','{1.0,0.3}'),
    set_comunicacion('ALTA'),
    set_relaciones('BUENAS'),
    set_analisis('ALTA'),
    set_idoneidad('1'),
    set_indice('MEDIO'),
    set_indice('MEDIO'),
    set_indice('BAJO'),
    set_indice('BAJO'),
    set_indice('BAJO'),
    set_indice('ALTO'),
    set_indice('MUY_ALTO'),
    set_indice('BAJO'),
    set_indice('ALTO'),
    set_indice('BAJO'),
    set_trab_eq('MEDIA'),
    set_trab_eq('ALTA'),
    set_idoneidad('1'),
    set_indice('ALTO'),
    set_rolescollection('{Programador,Diseñador,Analista}','{0.9,0.4,0.6}'),
    set_leng_progcollection('{Java,JSP}','{0.8,0.6}'),
    set_experiencia('MUY_POCA'),
    set_experiencia('MEDIA'),
    set_experiencia('MUY_POCA'),
    set_experiencia('MEDIA'),
    set_experiencia('MUY_POCA')
)
);
```

H.3. Temas

```
INSERT INTO ttema VALUES(
'sistcontbweb',
'SISTEMA CONTAB WEB',
```

```
set_curriculo (
    set_disponibilidad('ALTA'),
    set_anio('MEDIOS'),
    set_responsabilidad('MEDIA'),
    set_personalidadcollection('{HONESTO,INDEPENDIENTE,TRABAJADOR,
        CREATIVO}','{0.7,0.4,0.9,0.2}'),
    set_idiomacollection('{español,ingles}','{1.0,0.3}'),
    set_comunicacion('ALTA'),
    set_relaciones('BUENAS'),
    set_analisis('ALTA'),
    set_idoneidad('1'),
    set_indice('MEDIO'),
    set_indice('MEDIO'),
    set_indice('BAJO'),
    set_indice('BAJO'),
    set_indice('BAJO'),
    set_indice('ALTO'),
    set_indice('MUY_ALTO'),
    set_indice('BAJO'),
    set_indice('ALTO'),
    set_indice('BAJO'),
    set_trab_eq('MEDIA'),
    set_trab_eq('ALTA'),
    set_idoneidad('1'),
    set_indice('ALTO'),
    set_rolescollection('{Programador,Diseñador,Analista}','{0.9,0.4,0.6}'),
    set_leng_progcollection('{Java,JSP}','{0.8,0.6}'),
    set_experiencia('MUY_POCA'),
    set_experiencia('MEDIA'),
    set_experiencia('MUY_POCA'),
    set_experiencia('MEDIA'),
    set_experiencia('MUY_POCA')
),
set_disponibilidad ('MEDIA')
```

```
);
```

H.4. Proyectos insertados en la base de datos

Cuadro H.5: Proyectos insertados en la base de datos

oid	cod_proy	nombre_proy	instit.	f_inicio	dura.	finan	tipoproy
45297	Che	Inform. Niquel	UHO-MINBAS	25/01/2001	23	23000	ESTRATEGICO
45298	Gav	Inform. Cadena Gaviota	UHO-MINTUR	23/11/2002	40	MUY_GRANDE	ESTRATEGICO
45299	FELTON	Inform. Termoelectrica Feltón	UHO-MINBAS	12/12/2003	7	8500	SERVICIO
45304	Che12H	Moa 12 Horas	UHO-MINBAS	15/09/2004	11	1300	PRODUCCION
45305	MED	Inform. Emp. Medicamentos	UHO-MINBAS	13/04/2005	4	2500	SERVICIO
45306	CITMA	Inform Gestion Ciencia	UHO-CITMA	13/03/2009	MEDIO	GRANDE	SERVICIO

Apéndice I

Manual de usuario para pg4DB.

Este apéndice presenta un breve manual de usuario de la aplicación web pgFORDBi. Esta aplicación fue utilizada en el ejemplo mostrado en el capítulo 5 y tiene como objetivo mostrar la validez del modelo y la implementación propuesta en esta investigación. Se describirán las principales capacidades y funcionalidades de la aplicación mostrando la forma en la que el usuario puede interactuar con ella.

I.1. Conexión al servidor de base de datos.

El primer paso para el trabajo con la aplicación es conectarse al servidor de la base de datos. En este caso se utilizará una interfaz como la que se muestra en la figura I.1.

Como se muestra en la figura I.1 se requiere de la siguiente información para realizar la conexión: servidor, base de datos, usuario y contraseña. Cualquier error que se produzca al intentar realizar la conexión se le mostrara un mensaje al usuario con la descripción del error (ver Figura I.2)

I.2. Opciones de la interfaz.

Una vez que el usuario proporcionó la información necesaria para poder realizar una conexión con una base de datos la aplicación entra en su página principal (ver Figura I.3)

Como se muestra en la figura I.3 la página principal de la aplicación esta dividida en cuatro zonas. Una primera zona que es el encabezado con información de la aplicación (banner), una segunda zona que es el pie de la página con información sobre la autoría de

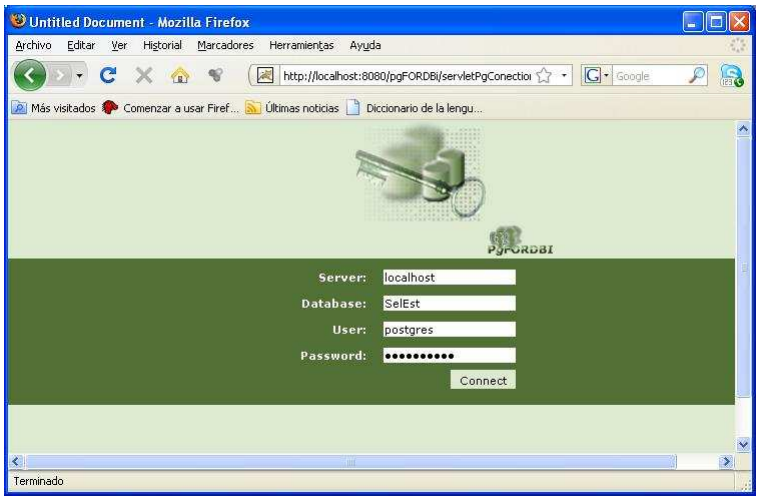


Figura I.1: Conexión al servidor de base de datos.

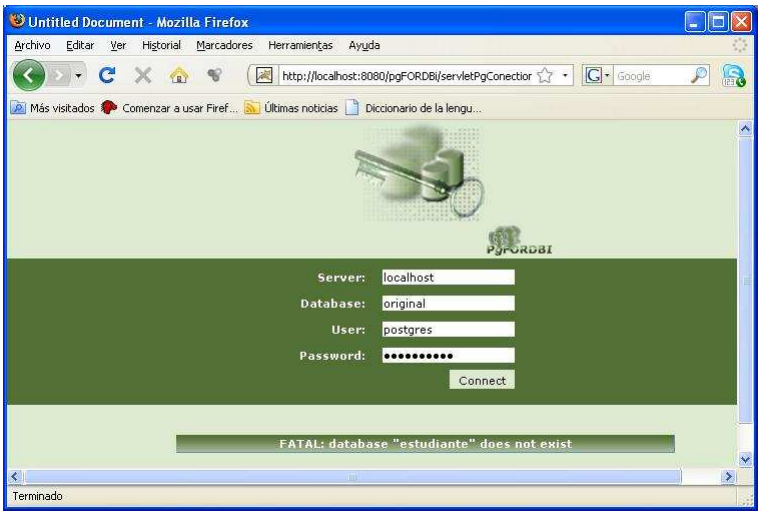


Figura I.2: Error en el sistema.

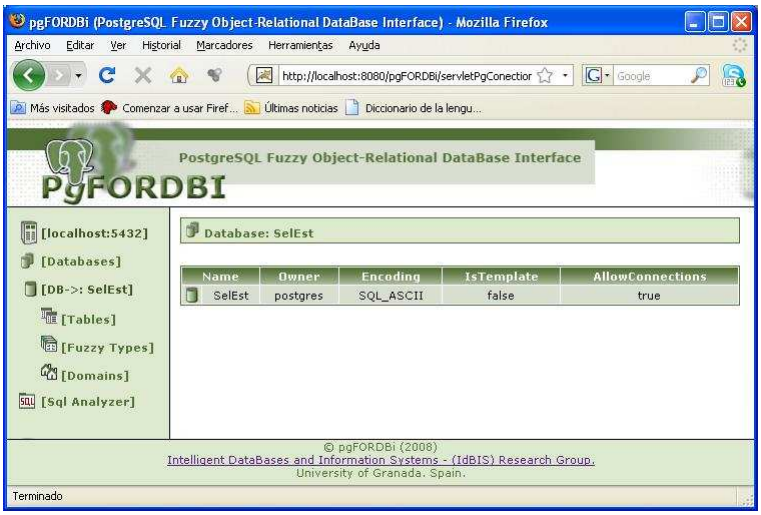


Figura I.3: Página principal.

la aplicación. La tercera parte es la zona de la derecha en la cual se mostraran todas las páginas que permitirán interactuar con el modelo. Y una cuarta zona a la izquierda con un menu que permite navegar por la aplicación. En este menu existen las siguientes opciones:

- Información sobre la conexión, ip y puerto de conexión.
- Opción Databases: Esta opción permite gestionar las bases de datos que están en el servidor de bases de datos.
- Información de cual base de datos se está conectado en ese momento.
- Opción Tables: Permite gestionar las tablas que existen en la base de datos.
- Opción Fuzzy Type: Para gestionar los tipos difusos definidos en la base de datos seleccionada.
- Opción Domains: Permite gestionar todo lo relacionado con los dominios difusos presentes en la base de datos.
- SQL Analyzer: Esta opción permite al usuario escribir sus consultas SQL para la base de datos.
- Cerrar conexión: Cierra la conexión con el servidor de base de datos.

Cada una de estas opciones se irán detallando en el transcurso de este apéndice.

I.3. Gestión de base de datos.

Como se mostró, en el proceso de conexión el usuario especifica con que base de datos quiere conectarse. Al entrar al sistema se encontrará con una página que muestra algunas de las propiedades de la bases de datos a la que se conecto (ver Figura) esta propiedades son: nombre de la base de datos, propietario de la base de datos, codificación utilizada para la base de datos, si es una plantilla y si permite conexiones.

Por medio de la opción *Database* del menu se pueden ver todas las bases de datos que existentes en el servidor (ver Figura). En esa página se muestra un listado de todas las bases de datos existente y se resalta la base de datos activa en ese momento.

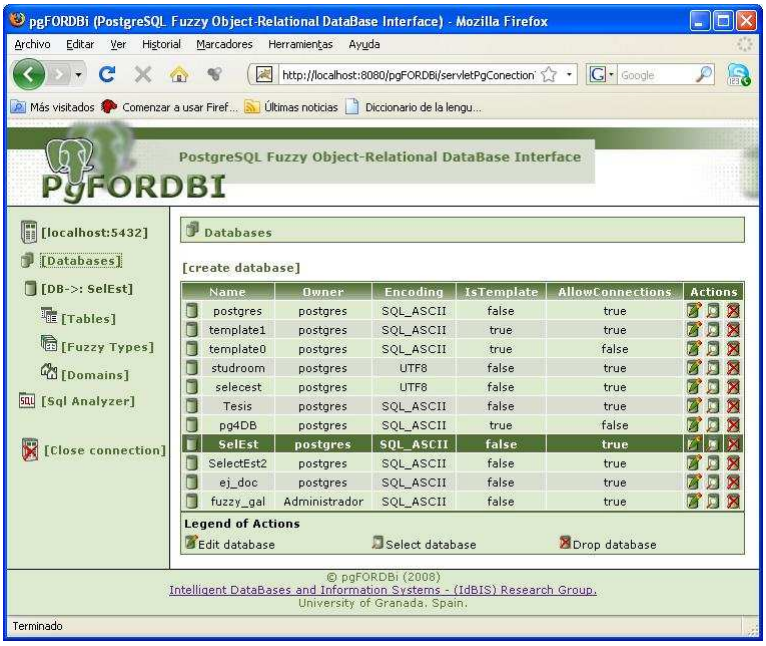


Figura I.4: Gestión de bases de datos.

El usuario tiene la posibilidad de realizar las siguientes acciones: editar una base de datos, seleccionar una base de datos y eliminar una base de datos.

Al editar una base de datos (ver Figura I.5) el usuario podrá cambiar el nombre de la base de datos y el propietario. Al seleccionar una base de datos se pondrá activa esa base de datos para poder navegar dentro de ella. La opción de eliminar base de datos borra la base de datos del servidor.

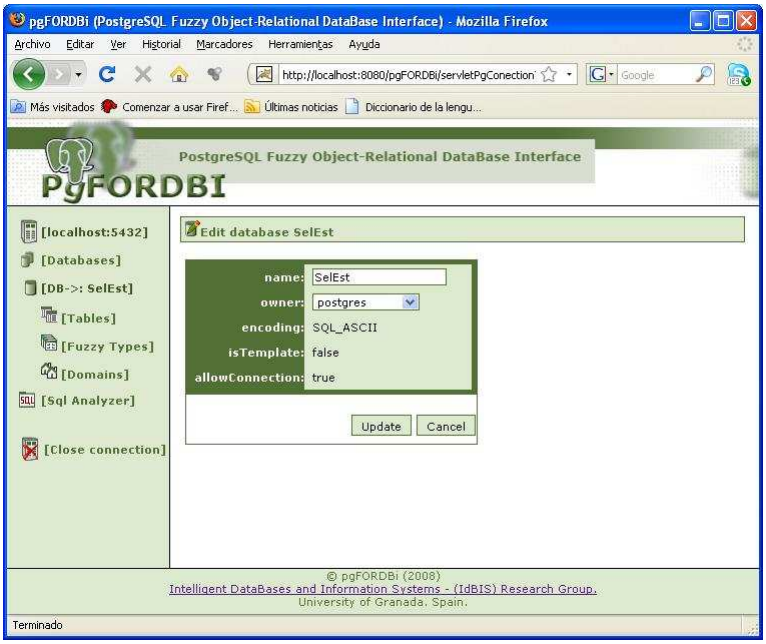


Figura I.5: Editar bases de datos.

I.4. Gestión de tablas.

La gestión de las tablas de la base de datos se realiza por medio de la opción *Tables* del menu. Al acceder a esta opción aparecerá un listado de todas las tablas de la base de datos (ver Figura I.4.3) mostrando las propiedades nombre, propietario, si es difusa, y esquema donde esta definida.

El primer elwmento en la gestión de tablas es la creación de nuevas tablas. A esta opción se accede por medio del link *Create Table* de la figura I.4.3. Al seleccionar esta opción el usuario podrá crear una nueva tabla en el sistema, en dependencia de los tipos de datos que seleccione para los atributos de la tabla se creara una tabla difusa o una tabla precisa.

La primera información necesaria para crear una tabla que se le solicita al usuario es el nombre de la tabla, el propietario y la cantidad de atributos que quiere que tenga la tabla (ver Figura I.7).

Luego el usuario podrá ir definiendo sus atributos según la página que se muestra en la figura I.8. Para ganar en sencillez solo se muestran dos atributos. El usuario puede definir para cada atributo el nombre, el tipo de dato, es la columna permite valores nulos, si es forma parte de la clave, la longitud y restricciones a nivel de columna. También puede

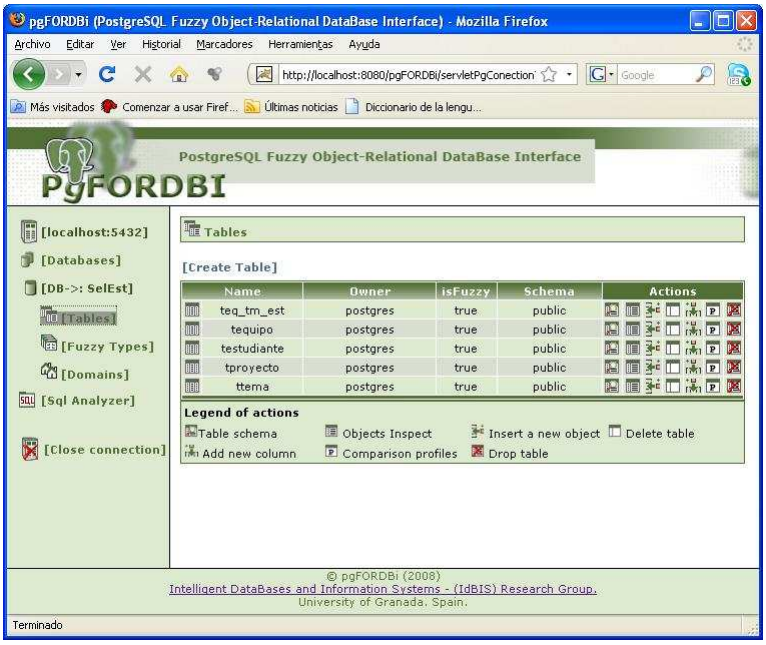


Figura I.6: Tablas en la bases de datos.

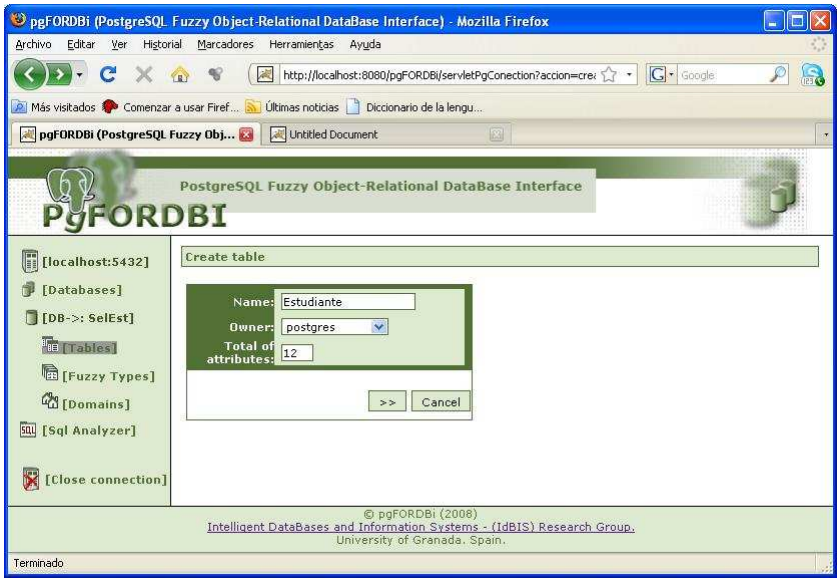


Figura I.7: Crear una tabla.

definir para la tabla si hereda de alguna otra y restricciones a nivel de tabla.

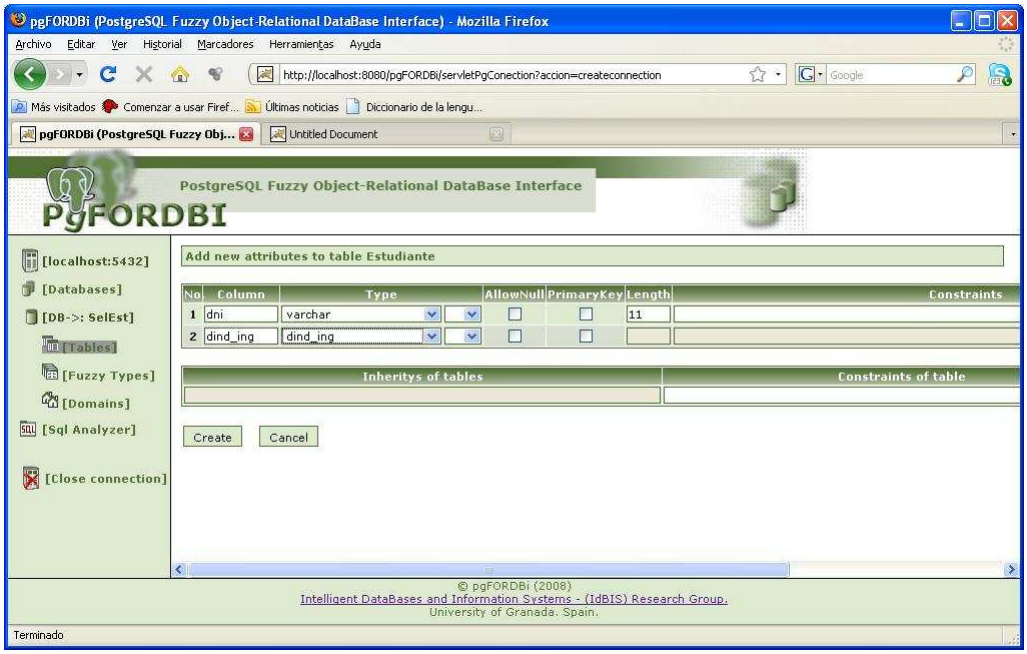


Figura I.8: Definir atributos en una tabla.

Si en el proceso de definir atributos para la tabla (ver Figura I.8) se especifica como dominio de uno de los atributos un dominio difuso, automáticamente el sistema asume que se esta creando una tabla difusa y desactiva las opciones no validas, como se muestra en la figura. En el ejemplo de la figura al seleccionar un dominio difuso para el atributo *dind_ing* se desactivo la posibilidad de definir restricciones para ese atributo y la opción de herencia entre tablas. Estos elementos lo manipulará el sistema.

Como puede notarse en la figura con las tablas se pueden realizar varias acciones: gestionar su esquema, ver los objetos, insertar nuevos objetos, borrar la tabla, adicionar nuevos atributos, gestionar perspectivas de comparación y eliminar la tabla. Estas acciones serán descritas a continuación.

I.4.1. Esquema de una tablas.

El esquema de una tabla se muestra al seleccionar la acción correspondiente (ver Figura I.9).

En la página mostrada por la figura I.9 se muestra un listado de los atributos que

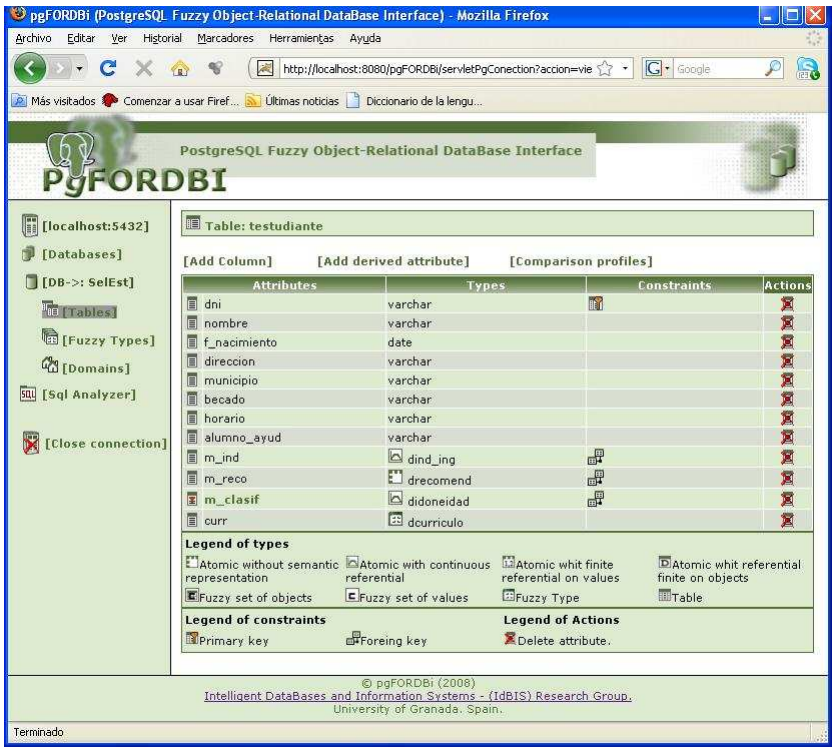


Figura I.9: Esquema de una tabla.

describen los objetos almacenados en la tabla seleccionada. Este listado incluye el nombre del atributo, el tipo de dato con icono identificando el tipo de dato en caso de ser difuso (ver la leyenda en la misma figura) y si el atributo es una clave extranjera o forma parte de la clave de la relación. En la página se da la posibilidad de eliminar el atributo por medio de la acción *Delete Attribute*.

Además en una tabla se puede adicionar un nuevo atributo por medio del link *Add Column*, en esta caso se muestra la página de la figura I.10 en la cual se puede adicionar un nuevo atributo.

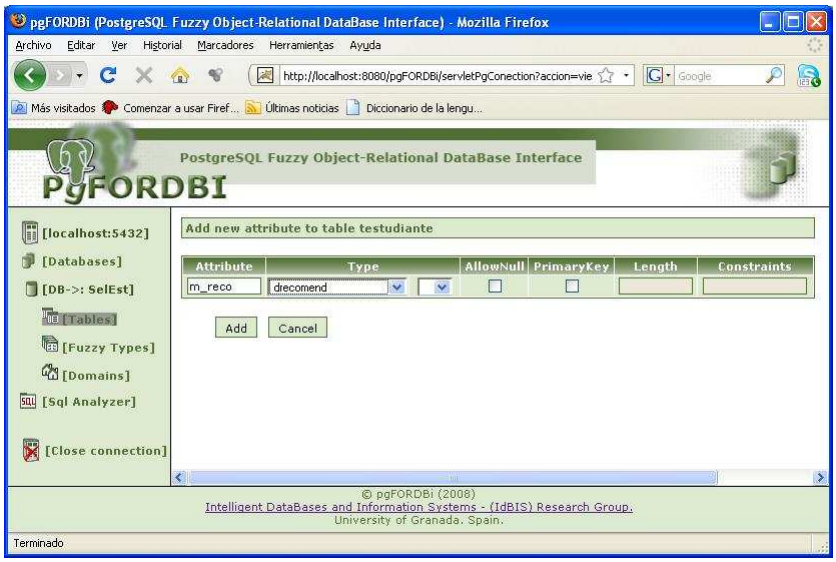


Figura I.10: Adicionar atributo a una tabla.

También es posible adicionar un nuevo atributo inferido por medio del link *Add derived attribute*, las páginas para adicionar atributos inferidos serán presentadas posteriormente en este mismo apéndice. Y la posibilidad de definir perspectivas de comparación por medio del link *Comparison profiles* que también será presentado más adelante. Estas dos opciones son similares para las tablas y los tipos difusos, por eso se mostrarán de forma independiente.

I.4.2. Ver objetos.

Los objetos almacenados en una tabla pueden verse por medio de la acción *Objects Inspect*. Por ejemplo para la tabla *tproyecto* se mostrarán los objtos almacenados en ella según la figura I.11.

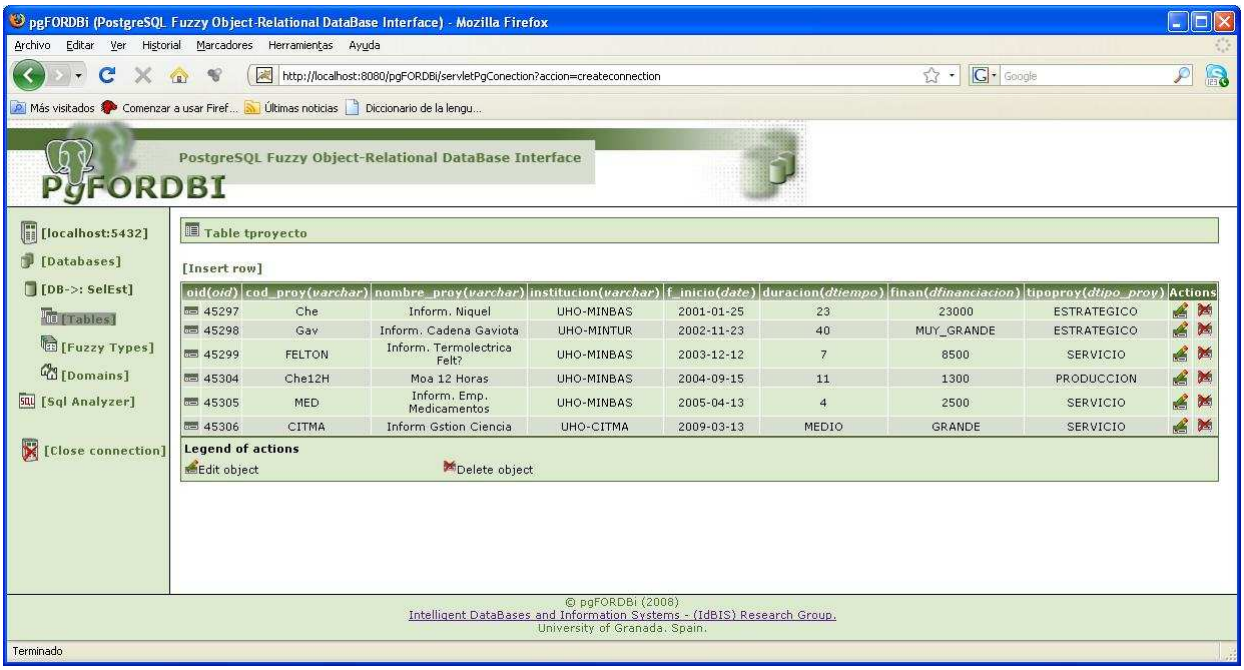


Figura I.11: Ver objetos de una tabla.

En la figura I.11 se listan los objetos existentes en la tabla dando la posibilidad de editar el objeto o eliminarlo por medio de las acciones correspondientes. Y la posibilidad de insertar un nuevo objeto por medio del link *Insert row*. Esta opción será descrita en el proximo apartado.

I.4.3. Insertar objetos.

Se pueden insertar nuevos objetos en una tabla por medio de la acción *Insert a new Object* de la figura . En este caso se mostrará una interfaz que dependerá de los atributos existentes en la tabla y el tipo de dominio sobre el que están definidos. Un ejemplo para la tabla *tproyecto* es el mostrado en la figura I.12.

Como se menciono la interfaz para insertar objetos se adaptara a la cantidad de atributos existentes en la tabla y al tipo de dominio sobre el que está definido el atributo, de esa forma se tiene las siguientes opciones.

Atributos precisos Los atributos precisos se insertan directamente en un control de edición (ver Figura I.13).

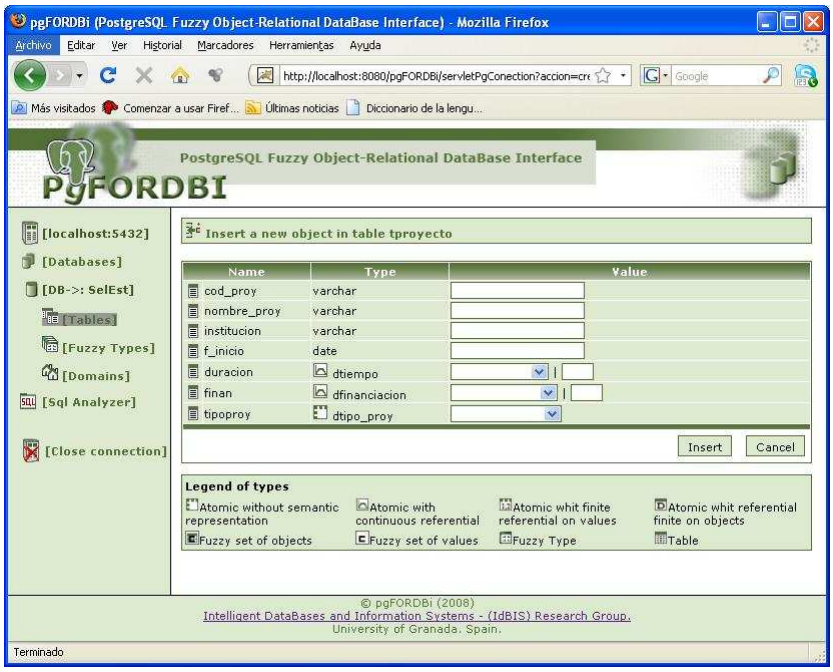


Figura I.12: Ver objetos de una tabla.

Figura I.13: Insertar valor preciso.

Atributo definido sobre un dominio atómico sin representación semántica En este caso se utiliza una lista de selección para solo poder seleccionar un valor entre las etiquetas definidas para el dominio (ver Figura).

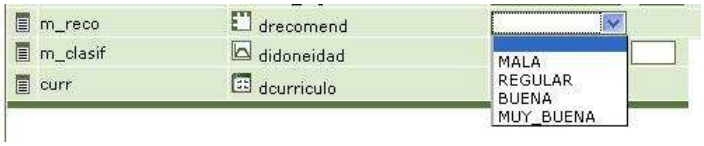


Figura I.14: Insertar valor dominio atómico sin representación semántica.

Atributo definido sobre un dominio atómico con referencial continuo Se utilizan dos controles, una lista de selección para seleccionar un valor entre las etiquetas definidas para el dominio y un control de edición para introducir un valor diferente a las etiquetas definidas (ver Figura I.15).

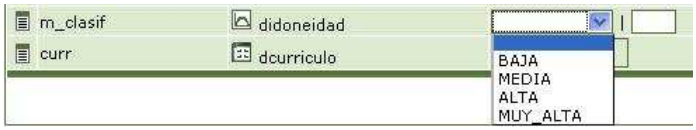


Figura I.15: Insertar valor dominio atómico con referencial continuo.

Atributo definido sobre un dominio atómico con referencial finito sobre valores Por medio de una lista de selección el usuario puede seleccionar un valor entre las etiquetas definidas para el dominio o el dominio básico definido (ver Figura I.16).

Figura I.16: Insertar valor dominio atómico con referencial finito sobre valores.

Atributo definido sobre un dominio conjuntivo de valores En este caso se define un asistente para ir creando el subconjunto que será el valor del atributo. Este asistente esta formado por dos controles de edición para ir introduciendo el elemento y el grado. Luego el elemento y el grado pueden ser adicionados al subconjunto resultan-

te por medio del botón +. También puede eliminarse un elemento del subconjunto seleccionándolo y presionado - (ver Figura I.17).

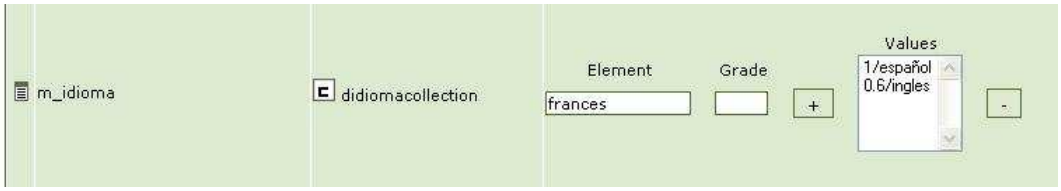


Figura I.17: Insertar valor dominio conjuntivo de valores.

Atributo definido sobre un dominio conjuntivo de objetos En este caso se define un asistente similar al de los dominio conjuntivo de valores, solo que en este caso los elementos que pueden formar parte del subconjunto son seleccionados de una lista (ver Figura figman:insertaconjobj). Este procedimiento es valido para cualquier tipo de objeto: objetos, objetos complejos, tipos difusos. Este mismo asistente es utilizado para dominios atómicos con referencial finito sobre objetos.

Figura I.18: Insertar valor dominio conjuntivo de objetos.

Atributo definido sobre objetos En el caso de atributos que son objetos o tipos difusos el valor se puede seleccionar desde una lista. En esa lista parecerá el valor de la clave de los objetos o su identificados en caso de no existir una clave definida (ver Figura I.19 y Figura I.20).

Figura I.19: Insertar objeto como valor de un atributo.

Para ayudar al usuario se proporciona una opción, botón ... que muestra todos los atributos de los objetos existentes, pudiendo seleccionar uno de ellos. Automáticamente el objeto seleccionado será el valor para el atributo (ver Figura).

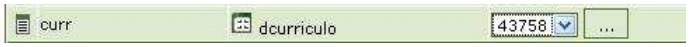


Figura I.20: Insertar tipo difuso como valor de un atributo.

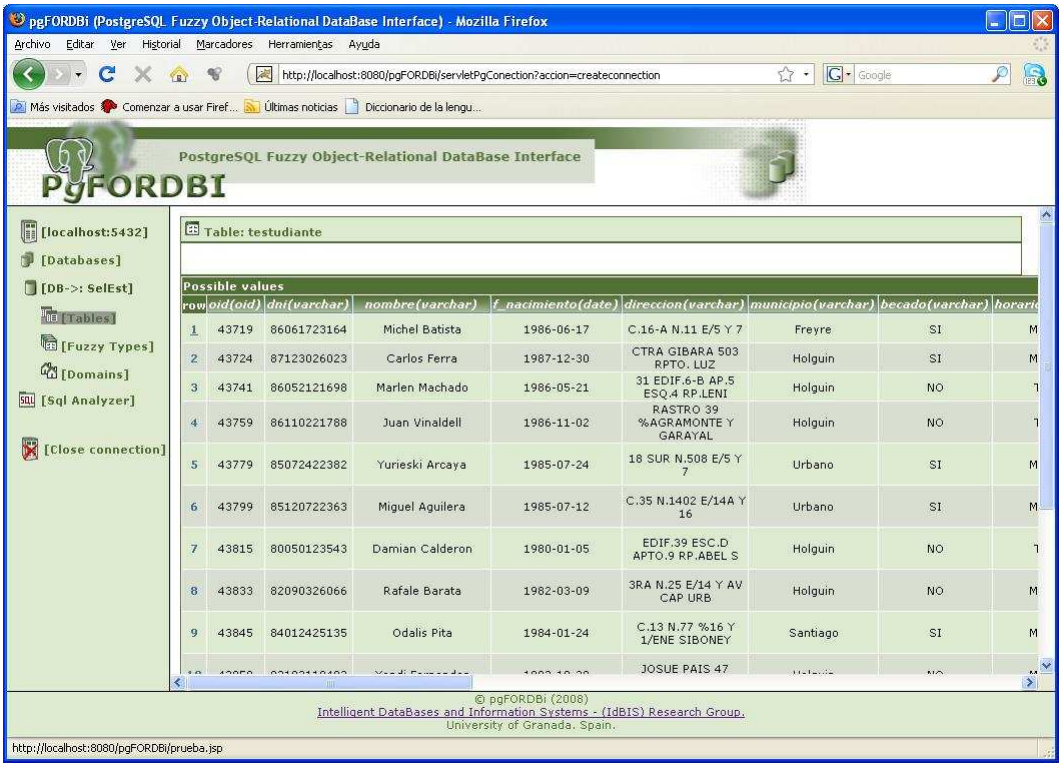


Figura I.21: Ayuda para insertar objeto como valor.

I.4.4. Borrar Tabla, Adicionar nuevos atributos y Eliminar tabla

Esta opción de borrar tabla, elimina todos los objetos de la tabla seleccionada. Antes de borrar solicita una confirmación de la operación.

Adicionar nuevos atributos, su propio nombre indica la operación que se realizará. El proceso y las páginas que se utilizan son las mostradas en la figura I.10.

Y eliminar tabla borra la tabla de la base de datos.

I.4.5. Gestionar Perspectivas de Comparación

Esta opción permite definir nuevas perspectivas de comparación para la tabla o actualizar una perspectiva ya existente. El procedimiento que se realiza en este caos es valido también para cuando se quieren gestionar las perspectiva en los tipos difusos.

Al seleccionar esta opción al usuario le saldrá una página como la que se muestra en la figura I.22.

LA definición de la perspectiva parte de haber seleccionado previamente una tabla. En el control *Select profile* el usuario puede seleccionar una de las perspectivas existentes. Automáticamente el nombre de la perspectiva aparece también en el control *Update o create profile*. Luego se muestra un listado con todos los atributos de la tabla mostrando: el nombre, el tipo, el peso que tiene en la perspectiva y el operador o perspectiva que se utilizará en la comparación.

El peso puede ser actualizado en el control edit asociado a cada atributo y los operadores/perspectiva podrán ser actualizados seleccionando de una lista. Esta lista está limitada a los operadores validos según el tipo de dominio. Para el caso de tipos objeto o conjuntivo sobre objetos se mostrarán además las perspectivas definidas para el tipo.

El usuario puede definir su perspectiva, si mantiene el mismo nombre en el control *Update o create profile* se actualizará la perspectiva seleccionada. De lo contrario, si se especifica otro nombre se creara una nueva perspectiva.

I.5. Escribir consultas

Para escribir y ejecutar consultas sobre la base de datos se tiene la opción del menu *SQL Analyzer*. Cuando se accede a esta opción se muestra la página de la figura I.23.

El usuario podrá escribir su consulta y ejecutarla presionado el botón SQL. El resultado de la consulta aparecerá en la misma página en la parte inferior. Para ayudar al usuario

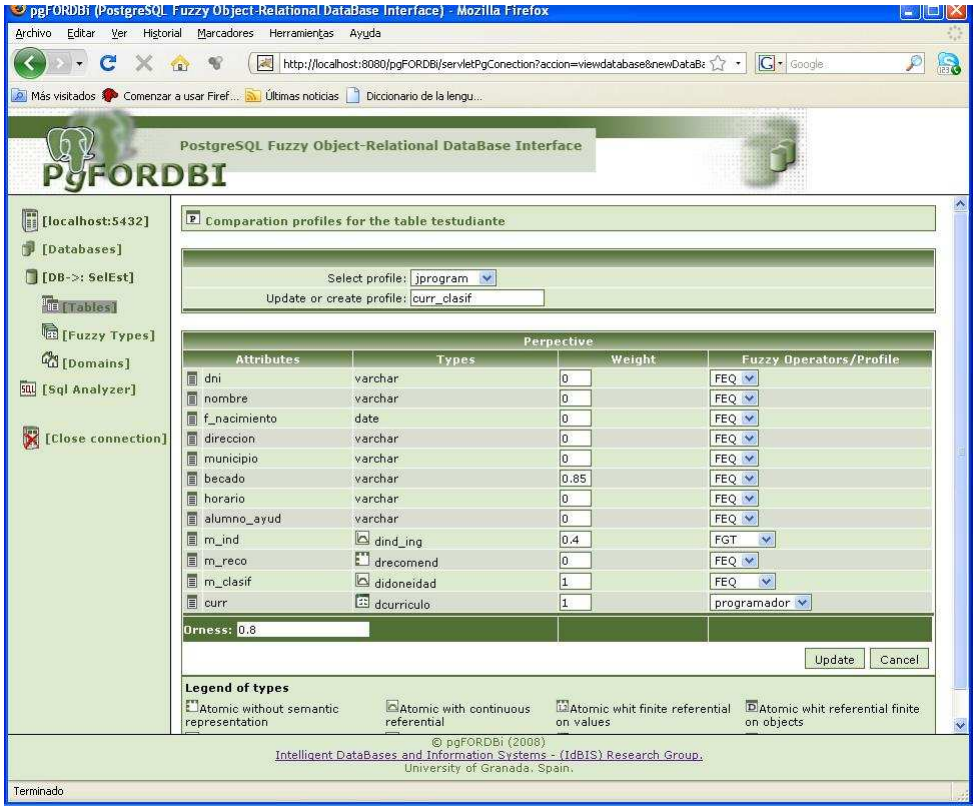


Figura I.22: Definir una perspectiva de comparación.

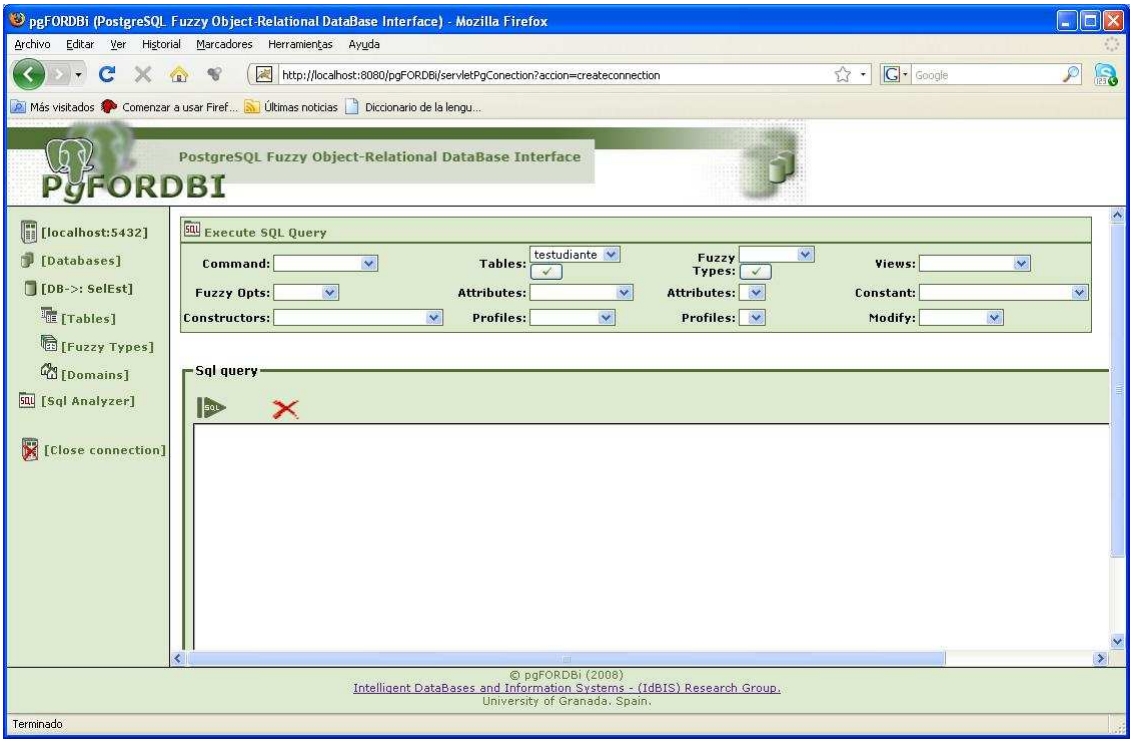


Figura I.23: Escribir sentencias SQL.

se incluyeron unos controles que le permiten acceder a los elementos más utilizados en las consultas, con solo seleccionar un elemento, este se insertará en la zona de escritura del SQL. Esto le permite al usuario escribir sus consultas con mayor facilidad. Los elementos de ayuda son:

- Comando SQL: es una lista de los principales comandos SQL
- Operadores difusos: una lista de todos los operadores difusos soportados por el modelo.
- Constructores: Son los constructores asociados a los dominios difusos creados en el sistema. LA lista muestra los constructores con sus parámetros.
- Tabla: Son las tablas de la base de datos. En este caso para insertar el nombre de la tabla en la zona de escritura debe presionarse el botón que esta al lado.
- Atributos: Atributos que tiene la tabla seleccionada.
- Perspectivas: Listado de todas las perspectivas definidas para la tabla seleccionada.
- Tipos difusos: Todos los tipos difusos existentes en la base de datos. Para insertar el nombre de la tabla en la zona de escritura debe presionarse el botón que esta al lado. Los controles Attribute y Profiles que están debajo están condicionado por el tipo difuso seleccionado.
- Vistas: un listado de las vistas disponibles en la base de datos.
- Constantes: Son las constantes difusas que estan disponibles en el modelo.
- Modificadores: Algunos de los modificadores que pueden utilizarse en consultas difusas.

I.6. Cerrar conexión

Como su nombre lo indica cierra la conexión existente con el servidor de base de datos, luego de que el usuario confirme la acción. Retornando a la página de conexión de la aplicación.

Bibliografía

[AH87] S. Abiteboul and R. Hull. Ifo: A formal semantic database model. *ACM Transc. On Database Systems*, 12(4):525–565, 1987.

[All07] Open Solutions Alliance. 2007 inter-open customer forum series summary report. http://opensolutionsalliance.org/osa/index.html, 2007.

[AM97] Philippe Smets Amihai Motro. *Uncertainty Management in information systems.* Kluwer Academic Publishers, 1997.

[Atk90] M. Atkinson. The object oriented database systems manifesto. In *Proceedings First International Conference on Deductive and Object Oriented Database.*, Tokio, Japon, 1990.

[Bal79] J. F. Baldwin. A new approach to approximate reasoning using a fuzzy logic. *Fuzzy Sets and Systems*, 2:309–325, 1979.

[BC02] G. Bordogna and S. Chiesa. Linguistic representation of imperfect spatial information in a fuzzy object oriented database. *IEEE*, 2002.

[BCMP04] Carlos D. Barranco, Jesús R. Campaña, Juan Miguel Medina, and Olga Pons. Immosoftweb: a web based fuzzy application for real estate management. pages 196–206, 2004.

[BGMP92] Daniel Barbará, Hector Garcia-Molina, and Daryl Porter. The management of probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 4(5):487–502, 1992.

[BKP05] Patrick Bosc, Donald H. Kraft, and Frederick E. Petry. Fuzzy sets in database and information systems: Status and opportunities. *Fuzzy Sets and Systems*, 156(3):418–426, 2005.

[BLP94] G. Bordogna, D. Lucarella, and G. Pasi. A fuzzy object oriented data model. In *Proceedings of FUZZ-IEEE 94*, pages 313–318, 1994.

[BMCMSH06] Carlos D. Barranco, Juan Miguel Medina, Jesús Chamorro-Martínez, and José M. Soto-Hidalgo. Using a fuzzy object-relational database for colour image retrieval. pages 307–318, 2006.

[BP82] E. P. Buckles and F.E. Petry. A fuzzy representation of data for relational database. *Fuzzy Sets and Systems*, 7(3):213–226, 1982.

[BP01] G. Bordogna and G. Pasi. Graph-based interaction in a fuzzy object oriented database. *International Journal of Intelligent Systems*, 16:821–841, 2001.

[Cal97] Rita de. Caluwe. Fuzzy and uncertain object-oriented database: Concepts and models. *Advances in Fuzzy Systems - Applications and Theory*, 13, 1997.

[Cat00] R. G. G. Cattell. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[CF00] V. Cross and A. Firat. Fuzzy objects for geographical information systems. *Fuzzy Sets and Systems*, 113:19–36, 2000.

[CGMP07] Jesús R. Campaña, M. Carmen Garrido, Nicolás Marín, and Olga Pons. A fuzzy set-based approach to temporal databases. In *SUM*, pages 31–44, 2007.

[CJ04] Medina J.M. Pons O. Vila M.A. Cubero J.C., MarinÑ. Fuzzy object management in an object-relational framework. In *X Intl. Conf. of information processing and management of uncertainty in knowledge-based systems*, pages 1767–1774, 2004.

[CKV92] G. Chen, E. E. Kerre, and J. Vandenbulcke. A general treatment of data redundancy in a fuzzy relational data model. *Journal of the American Society for Information Science*, 43(4):304–311, 1992.

[CKV94] G. Chen, E. E. Kerre, and J. Vandenbulcke. A computational algorithm for the ffd closure and a complete axiomatization of fuzzy functional de-

pendency (ffd). *International Journal of Intelligent Systems*, 9(5):421–439, 1994.

[CKV96] G. Chen, E. E. Kerre, and J. Vandenbulcke. Normalization based on functional dependency in a fuzzy relational data model. *Information Systems*, 21(3):299–310, 1996.

[Cod79] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst.*, 4(4):397–434, 1979.

[Cod86] E. F. Codd. Missing information (applicable and inapplicable) in relational databases. *SIGMOD Record*, 15(4):53–78, 1986.

[Cod87] E. F. Codd. More commentary on missing information in relational databases (applicable and inapplicable information). *SIGMOD Record*, 16(1):42–50, 1987.

[Cod90a] E. F. Codd. The relational model for database management, version 2. 1990.

[Cod90b] E.F Codd. *The Relational Model for Database Management: version 2.* Addison Wesley Publishing Company, 1990.

[CP87] Roger Cavallo and Michael Pittarelli. The theory of probabilistic databases. pages 71–81, 1987.

[Cro01] V. Cross. Fuzzy extensions for relationships in a generalized object model. *International Journal of Intelligent Systems*, 16(7):843–861, 2001.

[CV94] J. C. Cubero and M. A. Vila. Wad and strong resemblances in fuzzy functional dependencies. In *Proceedings of FUZZ-IEEE*, pages 44–47, Orlando, EUA, 1994.

[CV95] J. C. Cubero and M. A. Vila. A new definition of fuzzy functional dependency in fuzzy relational databases. *International Journal of Intelligent Systems*, 9(5):441–448, 1995.

[CVK91] G. Chen, J. Vandenbulcke, and E. E. Kerre. A step towards the theory of fuzzy relational database design. In *Proceedings of the 4th World Congress of International Fuzzy Systems Association, IFSA 91*, pages 44–47, 1991.

[Dat01] C. J. Date. *Introducción a los Sistemas de Bases de Datos. 7ma Edición.* Addison Wesley, 2001.

[DD98] C. J. Date and H. Darwen. *The Third Manifesto.* Addison Wesley, 1998.

[DD03] K. Douglas and S. Douglas. *PostgreSQL. A comprehensive guide to building, programming, and administering PostgreSQL database.* Sam Publishing, 2003.

[Dei03] Deitel. *C++ cómo programar.* Prentice Hall, 2003.

[DP94] D. Dubois and H. Prade. Quotient operators in fuzzy relational databases. In *Proceedings of EUFIT 94*, pages 357–360, 1994.

[DPR91] D. Dubois, H. Prade, and J.P. Rossazza. Vagueness, typicality and uncertainty in class hierarchies. *International Journal of Intelligent Systems*, 6:167–183, 1991.

[EM99] Andrew Eisenberg and Jim Melton. Sql: 1999, formerly known as sql 3. *SIGMOD Record*, 28(1):131–138, 1999.

[Fel71] W. Feller. *An Introduction to Probability Theory and Its Applications.* Wiley Text Books., 1971.

[Gal99] José Galindo. *Tratamiento de la Imprecisión en bases de datos relacionales: extensión dle modelo y adaptación de los SGBD actuales.* PhD thesis, Dpto. Ciencia de la Computación e Inteligencia Artificial, ETSI Informática, Universidad de Granada, 1999.

[Gal02] Aranda C. Caro J. L. Guevara A. Aguayo A. Galindo, J. Applying fuzzy databases and fsql to the management of rural accommodation. *Tourism Management*, 23 6:623 – 629, 2002.

[Gal05] Jose Galindo. *Fuzzy Databases: Modeling, Design And Implementation.* Idea Group Publishing (October 19, 2005), 2005.

[GBP91] R. George, B.P. Buckles, and F.E. Petry. An object-oriented data model to represent uncertainty in coupled artificial intelligence database systems. In Springer-Verlag, editor, *The next generation of information systems: From*

data to knowledge, pages 37–48. Lecture Notes in Artificial Intelligence, Berlín, 1991.

[GBP93] R. George, B.P. Buckles, and F.E. Petry. Modelling class hierarchies in the fuzzy object-oriented data model. *Fuzzy Sets and Systems*, 60(3):259–272, 1993.

[GC96] N.V. Gyseghem and Rita de. Caluwe. Fuzzy inheritance in the ufo database model. *IEEE*, pages 1365–1370, 1996.

[GC98] N.V. Gyseghem and Rita de. Caluwe. Imprecision and uncertainty in the ufo database model. *Journal of the American Society for Information Science*, 49(3):236–252, 1998.

[GCV93] N.V. Gyseghem, Rita de. Caluwe, and R. Vandenberghe. Ufo: Uncertainty and fuzziness in an object-oriented model. *IEEE*, pages 773–778, 1993.

[Gra88] C. Granger. An application of possibility theory to object recognition. *Fuzzy Sets and Systems*, 28(3):351–362, 1988.

[Jag95] René Jager. *Fuzzy Logic in Control*. PhD thesis, Delft University of Technology,Department of Electrical Engineering,Control laboratory, 1995.

[JUGZ05] Leoncio Jiménez, Angélica Urrutia, José Galindo, and Pascale Zaraté. Implementación de una base de datos relacional difusa. un caso en la industria del cartón. *Revista Comlombiana de Computación*, 6(2):48–58, 2005.

[KS98] H.F. Korth and A. Silberschatz. *Fundamentos de Bases de Datos 3ra Edición*. McGraw-Hill, 1998.

[KY95] G.J. Klir and B. Yuan. *Fuzzy Set and Fuzzy Logic: Theory and Applications*. Prentice Hall, 1995.

[LFKL03] Jonathan Lee, Y.Y. Fanjiang, J.Y. Kuo, and Y.Y. Lin. Modelling imprecise requirements with xml. *Information & Software Technology*, 45(7):445–460, 2003.

[LSZ93] D. Lucarella, S.Parisotto, and A Zanci. More: Multimedia object retrieval environment. In *Proceedings of the fifth ACM Conference on Hypertext*, pages 39–50, 1993.

[MA75] E. H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1):1–13, 1975.

[Man94] F. Manola. An evaluation of object-oriented dbms developments: 1994 edition. *GTE Laboratories Incorporated*, TR-0263-08-94-165, 1994.

[Mar01] N. Marín. *Estudio de la Vaguedad en los Sistemas de Bases de Datos Orientados a Objetos: Tipos difusos y sus Aplicaciones.* PhD thesis, Universidad de Granada, 2001.

[MBPV03] N. Marín, F. Berzal, O. Pons, and M. A. Vila. Foodbi: Managing fuzzy object-oriented data on top of the java platform. In *International Fuzzy Systems Association World Congress, IFSA'2003*, pages 384–387, Istanbul, Turkey, 2003.

[MBS02] Arun K. Majumdar, Indrajit Bhattacharya, and Amit K. Saha. An object-oriented fuzzy data model for similarity detection in image databases. *IEEE Trans. Knowl. Data Eng.*, 14(5):1186–1189, 2002.

[McK07] Joe McKendrick. Open source in the enterprise. new software disrupts the technology stack. Independent Oracle Users Group: For the Complete Technology & Database Professional. http://www.ioug.org/, 2007.

[Mey00] N. Meyer. *Programación java en Linux.* Prentice Hall, 2000.

[MMP+03] N. Marín, J.M. Medina, O. Pons, D. Sánchez, and M. A. Vila. Complex object comparison in a fuzzy context. *Information & Software Technology*, 45:431–444, 2003.

[Mou99] N. Mouaddib. The nuanced relational division. In *Proceeding of FUZZIEEE*, volume 2, pages 1419–1424, 1999.

[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.

[MPV01] N. Marín, O. Pons, and M. A. Vila. A strategy for adding fuzzy types to an object-oriented database system. *International Journal of Intelligent Systems*, 16(7):863–880, 2001.

[MVP00] N. Marín, M. A. Vila, and O. Pons. Fuzzy types: Softening structures. In *Proceedings of the ninth IEEE international conference on fuzzy systems.*, pages 774–779, 2000.

[MZM00] Z.M. Ma, W.J. Zhang, and W.Y. Ma. Semantic measure of fuzzy data in extended possibility-based fuzzy relational databases. *international Journal of Intelligent Systems*, 15(8):705–716, 2000.

[MZMC00] Z.M. Ma, W.J. Zhang, W.Y. Ma, and G. Chen. Functional dependencies in extended possibility based fuzzy relational databases. In *Proceeding of Ninth IEEE International Conference Fuzzy Systems*, volume 2, pages 929–932., 2000.

[PMM+03] O. Pons, N. Marín, J.M. Medina, S. Acid, and M. A. Vila. *Introducción a las Bases de Datos.* 2003.

[Pos08] PostgreSQL. Postgresql 8.0.0 documentation, 2008.

[PT84] H. Prade and C. Testemale. Generalizing database relational algebra for the treatment of incomplete or uncertain information and vague queries. *Information Science*, 34(2):115–143, 1984.

[PY99] G. Pasi and R.R Yager. Calculating attribute values using inheritance structures in fuzzy object-oriented data models. *IEEE Transaction Systems Man and Cybernet*, 29(4):556–564, 1999.

[RDP98] J. Rossazza, D. Dubois, and H. Prade. A hierarchical model of fuzzy classes. In *In: De Caluwe R,. editor. Fuzzy and Uncertain Object - oriented Databases. Concepts and Models.*, pages 21–61. 1998.

[RHB89] E.A. Rundensteiner, L.W. Hawkes, and W. Bandler. On nearness measures in fuzzy relational data models. *International Journal of Approximate Reasoning*, 3(3):267–298, 1989.

[RM88] K. Raju and A. Majumdar. Fuzzy functional dependencies and lossless join decomposition on fuzzy relational database systems. *ACM Transactions on Database Systems (TODS)*, 13(2):129–166, 1988.

[Rus86] E.H. Ruspini. Imprecision and uncertainty in the entity-relationship model. In *Prade H. y Negiota C. V. (Eds) Fuzzy Logic and Knowledge Engineering*, pages 18–28. 1986.

[SB75] E. Shortliffe and B. Buchanan. A model of inexact reasoning in medicine. *Mathematical Biosciences*, 23:351–379, 1975.

[Sch76] G. Schafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[SLK01] A. Schenker, M. Last, and A. Kandel. Fuzzification of an object-oriented database systems. *International Journal of Fuzzy System*, 3(2):432–441, 2001.

[SM89] S. Shenoi and A. Melton. Proximity relations in the fuzzy relational databases. *Fuzzy Sets and Systems*, 31(3):285–296, 1989.

[Sto89] M. Stonebraker. Third - generation database system manifesto. In *ACM SIGMOD*, 1989.

[Sto99] M. Stonebraker. *Object-Relational DBMSs*. Morgan Kaufmann Publishers., 1999.

[TC03] Guy de Tré and Rita de. Caluwe. Level-2 fuzzy set and their usefulness in object-oriented database modelling. *Fuzzy Sets and Systems*, 140:29–49, 2003.

[TCP08] Guy De Tré, Rita M. M. De Caluwe, and Henri Prade. Null values in fuzzy databases. *J. Intell. Inf. Syst.*, 30(2):93–114, 2008.

[TKS91] K. Tanaka, S. Kobayashi, and T. Sakanoue. Uncertainty management in object-oriented database system. In *Database and Expert Systems Applications (DEXA) 91*, pages 251–256, 1991.

[Uma82] M. Umano. Freedom-o: A fuzzy database system. *Gupta M. M. y E.Sánchez (Eds.) Fuzzy Information and Decision Processes.*, pages 339–347, 1982.

[UW99] J.D. Ullman and J. Widom. *A First Course in Database Systems.* Prentice Hall, 1999.

[VC91] R. Vandenberghe and Rita de. Caluwe. An entity - relationship approach to the modelling of vagueness in databases. In *Proccedings of ECSQAU - Symbolic and Quantitative Approaches to Uncertainty 1991*, pages 338–343, 1991.

[VCMP94a] M. A. Vila, J. C. Cubero, J.M. Medina, and O. Pons. The generalized selection: an alternative way for the quotient operations in fuzzy relational databases. In *Proceedings of IPMU 94*, pages 23–30, 1994.

[VCMP94b] M. A. Vila, J. C. Cubero, J.M. Medina, and O. Pons. A logic approach to fuzzy relational databases. *International Journal of Intelligent Systems*, 9(5):449–461, 1994.

[VCMP95] M. A. Vila, J. C. Cubero, J.M. Medina, and O. Pons. Dealing with uncertain and imprecise information in semantic database models. In *Proceeding of the Sixth IFSA Congress*, pages 237–240, S. Paulo. Brazil, 1995.

[VCMP96a] M. A. Vila, J. C. Cubero, J.M. Medina, and O. Pons. A conceptual approach to deal with imprecision and uncertainty in object.based data models. *International Journal of Intelligent Systems*, 11:791–806, 1996.

[VCMP96b] M. A. Vila, J. C. Cubero, J.M. Medina, and O. Pons. Structural vagueness in semantic data models. In *Proceedings of IPMU 96*, pages 1189–1196, 1996.

[VCMP97] M. A. Vila, J. C. Cubero, J.M. Medina, and O. Pons. Inheritance problems in fuzzy semantic data models. In *Proceedings of EUFIT 97*, 1997.

[VCMP98] M. A. Vila, J. C. Cubero, J.M. Medina, and O. Pons. A fuzzy object-oriented data model represented by means of a semantic data model. Technical report, DECSAI, 1998.

[Vil02] M.A. Vila. Modelos de datos avanzados: Bases de datos orientadas a objetos. In *Apuntes de clase*, 2002.

[Yag91] R.R Yager. Fuzzy quotient operators for fuzzy relational data bases. In *Proceedings of the International Fuzzy Engineering Symposium IFE 91*, volume 3, pages 13–15, 1991.

[Yag92] R.R Yager. Fuzzy quotient operators. In *Proceedings of IPMU 92*, pages 317–322, 1992.

[YAG96] A. Yazici, D. Aksoy, and R. George. The similarity-based fuzzy object-oriented data model. In *Proceeding of IPMU 96*, volume 3, pages 1177–1182, 1996.

[YC98] A. Yazici and A. Cinar. Conceptual design of fuzzy object-oriented database. In *International Conference on Knowledge-Based Intelligent Electronic Systems 98*, pages 299–305, 1998.

[YGA98] A. Yazici, R. George, and D. Aksoy. Design and implementation issues in the fuzzy object-oriented data model. *Journal of Information Sciences.*, 108:241–260, 1998.

[YK97] A. Yazici and M. Koyuncu. Fuzzy object-oriented database modelling coupled with fuzzy logic. *Fuzzy Sets and Systems*, 89(1):1–26, 1997.

[Zad75] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1975.

[Zad83] L.A. Zadeh. A computational approach to fuzzy quantifiers in natural languages. *Computer and Mathematic*, 9:149–184, 1983.

[ZC86] A. Zivieli and P.P. Chen. Entity-relationship modelling and fuzzy databases. In *Procceding of the Second International Conference on Data Engineering 86 - IEEE*, pages 18–28, 1986.

[ZK96] Slawomir Zadrozny and Janusz Kacprzyk. Fquery for access: towards human consistent querying user interface. pages 532–536, 1996.