Grado en Matemática Computacional

Prácticas Externas

Y

Proyecto de Final de Grado

Bases de Datos Relacionales Difusas.

Autora:
Delia Sanchis Minguez

Supervisor:
José Trias
Tutor académico:
Juan José Font Ferrandis

Fecha de lectura: 16 de Septiembre de 2015
Curso académico 2014/2015

Resumen

Este documento consta de dos partes, primero presenta un informe de la estancia en prácticas donde se especifican los objetivos y las tareas realizadas correspondientes a dicho periodo de prácticas, y a continuación, presenta un modelo de Bases de Datos Relacionales Difusas cuyas características principales son: la integración en un entorno común de modelos precedentes, la capacidad de representación de una amplia gama de información difusa y el tratamiento coherente y flexible de la misma. Todo ellos con el ánimo de ofrecer un modelo con el que resolver cada problema de representación y manipulación de información difusa, atendiendo a la naturaleza del mismo, pudiendo modelar la consulta, mediante la elección del operador de comparación y de la medida de compatibilidad difusa a emplear. Además, te ofrece la posibilidad de actuar de forma selectiva sobre la precisión con la que los atributos han de satisfacer las condiciones de una consulta.

Palabras clave

Bases de datos relacionales, Lógica difusa y modelo GEFRED.

Keywords

Relational database, Fuzzy logic and GEFRED Model.

Índice general

Capítulo 1

Prácticas externas

1.1. Descripción

Durante mi estancia en prácticas debido a la asignatura MT1030, he estado realizando las 290 horas de las prácticas en la empresa Molbe Studio. Esta empresa se dedica a la comercialización on-line de azulejos. Durante este tiempo mi supervisor en la empresa ha sido José Trias y se ha encargado de revisar mi proyecto según ha ido avanzando.

1.2. Objetivos

El trabajo a desarrollar fue:

-Estadísticas de precios en diferentes zonas.

-Cálculo de márgenes según costes y zonas de venta.

-Implementación de un sistema para calcular los costes y los precios de venta según zonas geográficas en base a los costes de gestión y distribución.

1.3. Tareas

Para llegar al cumplimiento de estos objetivos, nos basamos en los datos almacenados en la base de datos, la cual tenía algunos errores serios que tuvimos que ir modificando sobre la marcha.

Empezamos recopilando los datos de facturación del año 2014 y a continuación evaluamos los márgenes necesarios para aumentar el volumen de facturación de la empresa y si es factible, con más tiempo, hacer un programa que permita calcular precios de venta mínimo, según el

tipo de cliente y la zona geográfica.

1.4. Planificación

Durante las dos primeras semanas tuve un primer contacto con la base de datos de la empresa, es decir, examiné las tablas y profundicé en lo atributos, claves primarias y claves ajenas para poder realizar las consultas deseadas sobre éstas. También realicé las consultas necesarias para extraer la información que tuvimos que declarar en Hacienda del año 2014.

Durante las siguientes dos semanas calculé los pesos totales por facturas y los agrupé por zonas de forma que pudimos sacar unos nuevos umbrales que para la empresa significaron un ahorro importante a la hora de su distribución.

Una vez adquiridos los conocimientos necesarios y con la supervisión tanto del tutor como del responsable en la empresa, fui avanzando en el proyecto en la medida de lo posible.

1.4.1. Primer quincena

Exploración a fondo de la base de datos, conocer la distribución y los atributos para poder luego trabajar con ellos y extraer la información que me piden estudiar los atributos de cada tabla. Empezar a hacer una hoja Excel con las facturaciones del año 2014.

1.4.2. Segunda quincena

Continuamos puliendo la consulta anterior. Una vez estaba correcta, elaboramos un programa JAVA para cambiar la estructura del archivo ".csv".

Consultas para calcular el peso total por factura y para sacar el producto más vendido cada mes.

1.4.3. Tercera quincena

Nuevas consultas SQL para exportar los archivos ".csv" de la base de datos y luego hacer un programa en java para tratar con estos datos y así ahorrar tiempo y trabajo a los trabajadores de la empresa.

Calculo de márgenes de beneficio.

Elaboración de un programa JAVA que calcula los dos transportistas más baratos para cada pedido.

1.4.4. Cuarta quincena

Solucionar el problema de convertir un archivo ".java" en ".exe".

Sistematizar la forma de tratar los pedidos cada día mediante consultas SQL y un programa JAVA.

Sacar los productos y muestras más vendidas cada mes para elaborar estrategias de mejoras de costes.

Pensar una forma de comunicarse entre el encargado de preparar los paquetes de los pedidos y la persona que se encarga de atender los pedidos de los clientes.

1.4.5. Quinta quincena

Seguir puliendo la nueva forma de gestionar toda la información de los pedidos.

Aprender el funcionamiento del programa de contabilidad Ciel Compta Evolution y preparar los ficheros que éste necesita.

Preparar las hojas de cálculo para el mes de mayo.

1.4.6. Sexta quincena

Empezar a crear gráficos y márgenes de costes y beneficios con la información que hemos obtenido de los productos y muestras más vendidas desde el año 2014 hasta ahora.

Estudiar el tipo de cliente por zona geográfica, ya que próximamente se ampliará el mercado a Alemania, España y Reino Unido, y los precios y productos variarán dependiendo de la zona.

1.5. Informe de la estancia en prácticas

El primer día empecé leyendo un artículo del periódico *El Mundo* (listas de Spotify) para entender mis objetivos en la empresa. Luego me explicaron las distintas plataformas con las que trabaja la empresa (Prestashop, Dolibarr, phpMyAdmin). Los siguientes días ya empecé a manejar la base de datos. Elaboré un esquema de la base de datos que resultaría más útil a la hora de trabajar (artículos, líneasfac, facturas, clientes, vendedores, provincias, etc).

Para crear la hoja Excel, empecé recopilando la información de la primera semana de Enero del 2014 de la empresa. Para ello tuve que buscar en qué tablas de la base de datos de *Dolibarr* se encontraba dicha información (en qué tabla está, cómo se llama el atributo en dicha tabla). Una vez tuve la información necesaria localizada en las tablas, empecé a escribir las consultas

SQL necesarias para que, por ejemplo, me apareciese el IVA facturado en cada factura con su código de factura correspondiente.

También tuve que hacer algunas comprobaciones con algunas referencias de productos en la base de datos de *Dolibarr*, ya que habían errores en los albaranes de compra a proveedores. Resulta que en el pedido aparecía la referencia correcta del producto, pero en el albarán no salía la misma referencia.

Poco a poco iba localizando los atributos que me pedían para el Excel en las tablas de la bases de datos, precio del transporte de cada factura con y sin IVA, etc. Empezaba a hacer consultas, como por ejemplo: para sacar el precio del material sin IVA, el cual no estaba en la base de datos. Tuve que obtenerlo restando la columna del importe total de la factura sin IVA al precio del transporte sin IVA.

La consulta se complica. Por ejemplo, cuando hacemos la subconsulta en el FROM para sacar el descuento de cada factura, hay que poner a NULL las facturas que no se les haya aplicado ningún descuento y luego con el comando COALESCE, cambiar el NULL por un 0.

La consulta SQL es la siguiente:

```
1
2 SELECT datef, nom, facnumber, fk_pays, tva, total_ht, `erp_facture`.total_ttc,
      total, erp_paiement.note, si.dto
3 FROM `erp_facture`
4 LEFT OUTER JOIN `erp_paiement`
5 USING ( rowid )
6 JOIN `erp_societe`
7 JOIN `erp_facturedet`
8 JOIN (
9 SELECT facnumber, COALESCE( s.dto, 0 ) AS dto
10 FROM (
11 SELECT facnumber, erp_facturedet.total_ttc AS dto
12 FROM `erp_facturedet` JOIN `erp_facture`
13 WHERE erp_facturedet.fk_facture = erp_facture.rowid
14 AND erp_facturedet.total_ttc LIKE ( '-%' )
15 AND datef LIKE ( '2014-01-%' )
16 ) AS s
17 RIGHT OUTER JOIN erp_facture
18 USING ( facnumber )
19 WHERE datef LIKE ( '2014-01-%' )
20 ) AS si
21 USING ( facnumber )
22 WHERE erp_facture.fk_soc = erp_societe.rowid
23 AND erp_facturedet.fk_facture = erp_facture.rowid
24 AND ( description LIKE ( 'TRANS%' ) OR description LIKE ( 'Frais%'))
25 AND `erp_facture`.`datef` LIKE ( '2014-01%' )
26 AND erp_facture.paye LIKE 1
27 ORDER BY `erp_facture`.`datef` ASC LIMIT 0 , 30
```

Continuamos puliendo la consulta. Tenemos que contemplar cuándo una factura ha sido anulada. Hay facturas que no tienen entrada en la tabla erp_ payment. Esto lo arreglamos con un RIGHT OUTER JOIN, ya que éste era un motivo por el cual no me salían algunas facturas. Volvemos a comprobar que el resultado de mi consulta coincide con las facturas impresas.

Cuando analizo las facturas de todos los meses del año 2014 observo que el total facturado de cada mes no coincide con el que sale en la web de *Dolibarr*. El motivo de la ausencia de algunas facturas era que en la descripción de la línea de factura donde va el transporte, habían algunas facturas que tenían mal escrita la descripción (traSnporte) o que también lo escriben como "shipping". Además habían unas facturas con el total en negativo que tampoco salían porque tenían otra descripción, éstas eran devoluciones.

La consulta queda de la siguiente forma:

```
1
2 SELECT DISTINCT datef, nom, facnumber, fk_pays, tva, trans, `erp_facture`.
      total_ttc, total, erp_paiement.note, si.dto
3 FROM `erp_facture`
4 LEFT OUTER JOIN `erp_paiement`
5 USING ( rowid )
6 JOIN `erp_societe`
7 JOIN `erp_facturedet`
8 JOIN (
9 SELECT facnumber, COALESCE( s.dto, 0 ) AS dto
10 FROM (
11
12 SELECT facnumber, SUM(erp_facturedet.total_ttc) AS dto
13 FROM `erp_facturedet`
14 JOIN `erp_facture`
15 WHERE erp_facturedet.fk_facture = erp_facture.rowid
16 AND erp_facturedet.total_ttc LIKE (
17 '-%'
18 )
19 AND datef LIKE (
20 '2014-03-%'
21 )
22 GROUP BY facnumber
23 ) AS s
24 RIGHT OUTER JOIN erp_facture
25 USING ( facnumber )
26 WHERE datef LIKE (
27 '2014-03-%'
28 )
29 ) AS si
30 USING ( facnumber )
31 JOIN (
32
33 SELECT facnumber,description,COALESCE(total_ht,0) as trans
34 FROM(
35 SELECT datef,facnumber, erp_facturedet.description,total_ht
36 FROM `erp_facture` JOIN `erp_facturedet`
37 WHERE erp_facturedet.fk_facture = erp_facture.rowid
38 AND(
39 description LIKE (
40 'TRANS%
41 )
42 OR description LIKE (
43 'TRASN%
44 )
45 OR description LIKE (
46 'Frais%
47 )
48 OR description LIKE (
49 'Ship%
```

```
50)
51OR description LIKE (
52 'AVOIR_F%
53)
54OR description LIKE (
55 'ENVIO'
56)
57)AND datef LIKE (
58 '2014-03-%'
59)
60) AS factsConTrans RIGHT OUTER JOIN
61 `erp_facture` USING (facnumber)
62WHERE erp_facture.datef LIKE (
63 '2014-03-%'
64)
65
66) AS trans USING (facnumber)
67WHERE erp_facture.fk_soc = erp_societe.rowid
68AND erp_facturedet.fk_facture = erp_facture.rowid
69AND `erp_facture`.`datef` LIKE (
70 '2014-03%'
71)
72AND erp_facture.paye LIKE 1
73ORDER BY `erp_facture`.`datef` ASC
```

Para hacer las comprobaciones necesarias lo que hicimos fue ver en la cuenta de la empresa la cantidad de dinero que se ha ingresado cada mes y, comprobar que esa cantidad coincide con la que he obtenido yo.

Una vez vemos que es así, tenemos que cambiar la estructura del documento ".csv" ya que la contable francesa nos específica la forma en la que lo quiere, por ejemplo:

-Yo lo tengo con esta estructura

2014-01-01 / anne / FA1401-0979 / 1 / 7,46 / 19,21 / 18,83 / 45,5 /

Figura 1.1: Entrada

-Tengo que cambiarlo a esta:

Fecha	Código de factura	Código	Nombre-Código de factura		Importe
2014-01-01	FA1401-0979	707100	anne	FA1401-0979	18,83
2014-01-01	FA1401-0979	708500	anne	FA1401-0979	19,21
2014-01-01	FA1401-0979	445710	anne	FA1401-0979	7,46
2014-01-01	FA1401-0979	0	anne	FA1401-0979	0.0
2014-01-01	FA1401-0979	580001	anne	FA1401-0979	45,5

Figura 1.2: Salida

Para ello he tenido que crear un programa JAVA que maneje ficheros ".cs" (importación y exportación de ellos). Me he apoyado en tutoriales ya que nunca antes había hecho nada parecido con este tipo de ficheros.

El programa se encuentra en el **Anexo A.1.**

También he hecho la consulta para calcular el peso total por factura, con fecha, numfac y código postal:

```
SELECT datef, `erp_facturedet`.fk_facture, SUM(`erp_facturedet`.qty * `erp_product`.weight ) AS pesosTotFact, zip
FROM `erp_facturedet`
JOIN `erp_product`
JOIN `erp_facture`
JOIN `erp_societe`
WHERE erp_product.rowid = `erp_facturedet`.fk_product
AND `erp_facturedet`.fk_facture = `erp_facture`.rowid
AND erp_facture.fk_soc = erp_societe.rowid
AND `erp_facture`.datef LIKE (
'2014-01-%'
)
AND erp_facture.paye LIKE 1
GROUP BY datef, `erp_facturedet`.fk_facture, zip
```

Pero los resultados de esta búsqueda no son 100 % fiables ya que los datos en la base de datos no están correctos, pues hay muchas facturas (de fin de año la mayoría) que no salen porque no tienen identificador de producto, es decir, el producto no está en la tabla erp_ product. También aparecen pesos de productos negativos, y esto no es posible.

Además, he desarrollado la consulta para sacar el producto más vendido cada mes:

```
SELECT MAX(f.cantidadVendida), f.ref
FROM(
SELECT SUM(`ps_order_detail`.product_quantity) as cantidadVendida ,`ps_product`.reference as ref
FROM `ps_product` JOIN `ps_order_detail` JOIN `ps_order_invoice`
WHERE `ps_product`.id_product=`ps_order_detail`.product_id
AND `ps_order_invoice`.id_order_invoice= `ps_order_detail`.id_order_invoice
AND `ps_order_invoice`.delivery_date LIKE ('2014-01%')
GROUP BY `ps_order_detail`.product_id,`ps_product`.reference
) as f
```

También he creado una nueva consulta SQL para obtener los datos necesarios con los que luego se elaborará un documento ".csv" a partir del cual se podrá hacer un seguimiento económico y financiero para saber si han habido pérdidas o ha habido algún error como, por ejemplo, no haber cobrado el transporte a un cliente.

La consulta SQL con la que obtenemos dicha información (fecha en la que se hace el pedido, fecha de facturación, número de factura, código interno del cliente, cálculos para sacar el peso total por factura, precio del transporte con y sin IVA, coste del material (a la empresa)) es la siguiente:

```
1
2 SELECT `erp_facture`.datef,`erp_facture`.date_valid,`erp_facture`.facnumber,
      ref_client, SUM( `erp_facturedet`.qty * `erp_product`.weight ) AS
      pesosTotFact,zip,trans.transSINiva,trans.transCONiva, `erp_facture`.total as
      facturado ,SUM(COALESCE(`erp_product_fournisseur_price`.price,0) * `
      erp_facturedet`.qty) AS costeMat
3 FROM `erp_facture`
4 JOIN `erp_facturedet`
5 JOIN `erp_product`
6 LEFT OUTER JOIN `erp_product_fournisseur_price` ON ( `
      erp_product_fournisseur_price`.fk_product = erp_product.rowid )
7 JOIN `erp_societe`
8 JOIN (
9 SELECT facnumber,description,COALESCE(total_ht,0) as transSINiva,COALESCE(
      ttconiva,0) as transCONiva
10 FROM(
11 SELECT datef,facnumber, erp_facturedet.description,total_ht,`erp_facturedet`.
      total_ttc as ttconiva
12 FROM `erp_facture` JOIN `erp_facturedet`
13 WHERE erp_facturedet.fk_facture = erp_facture.rowid
14 AND(
15 description LIKE (
16 'TRANS%
17 )
18 OR description LIKE (
19 'TRASN%
20 )
21 OR description LIKE (
22 'Frais%
23 )
24 OR description LIKE (
25 'Ship%
26 )
27 OR description LIKE (
28 'AVOIR_F%
29 )
30 OR description LIKE (
31 'ENVIO'
32 )
33 )AND datef LIKE (
34 '2014-05-%'
35 )
36 ) AS factsConTrans RIGHT OUTER JOIN
37 `erp_facture` USING (facnumber)
38 WHERE erp_facture.datef LIKE (
39 '2014-05-%'
40 )
41
42 ) AS trans USING (facnumber)
43
44 WHERE erp_product.rowid = `erp_facturedet`.fk_product
45 AND `erp_facturedet`.fk_facture = `erp_facture`.rowid
46 AND erp_facture.fk_soc = erp_societe.rowid
47 AND `erp_facture`.datef LIKE (
48 '2014-05-%'
49 )
50 AND erp_facture.paye LIKE 1
51 GROUP BY `erp_facture`.facnumber,`erp_facture`.datef, `erp_facture`.date_valid,
      ref_client,zip,trans.transSINiva,trans.transCONiva,facturado
```

```
ORDER BY `erp_facture`.`datef` ASC , `erp_facture`.`rowid` ASC
```

Esta consulta tiene un problema ya que no muestra las facturas que tienen como inicio en el código de factura AV, es decir, las facturas que son devoluciones, ya que, al no tener código de producto, cuando haces JOIN entre las tablas, esta información se pierde.

Una vez tenemos la información necesaria para elaborar el documento ".csv", tenemos que implementar un programa JAVA que calcule el margen de beneficio y que sea capaz de seleccionar el transportista más barato para ese destino y con dicho peso.

Para ello tenemos que importar al programa JAVA un documento ".csv" que contiene todos los precios de los diferentes transportistas dependiendo del peso y del destino del pedido.

La clase que presentamos en el **Anexo A.2.** es la encargada de tratar el archivo ".csv" que obtenemos de exportar el resulta de la consulta SQL en la base de datos.

En el **Anexo A.3.** encontramos la clase encargada de tratar el archivo ".csv" donde se encuentra toda la información de los transportistas (precio, peso, destino).

Una vez creadas las clases que van a tratar nuestros ficheros, implementamos nuestro programa JAVA (**Anexo A.4.**).

Aquí podemos observar un pequeño ejemplo de su resultado:

Fecha envio	Fecha pedido	CUdigo de factura	CUdigo de pedido	Peso	Destino	pr. transp
2014-05-01	2014-05-01	FA1405-1654	CO1405-1607	15.0	69	18.32
2014-05-01	2014-05-01	FA1405-1655	CO1405-1608	5.0	31	18.32
2014-05-01	2014-05-01	FA1405-1656	CO1405-1609	12.0	6	18.32
2014-05-01	2014-05-01	FA1405-1657	CO1405-1610	20.0	71	35.61

Figura 1.3: Salida del programa JAVA

DSV	TRANSP	Transporte sin IVA	Transporte con IVA	PR REAL	DIF
	SPAINTIR	19.2	23.04000000	30.67	-11.470000
	SPAINTIR	19.2	23.04000000	30.67	-11.470000
	SPAINTIR	19.2	23.04000000	30.67	-11.470000
27.86	UCX	25.15	30.18000000	40.21	-15.060000

Figura 1.4: Salida del programa JAVA

Ttcfac sin IVA	Coste material	Coste total	Margen	Margen%
19.27	0.0	30.67	-11.400000000000002	-59.159314
19.2	0.0	30.67	-11.470000000000002	-59.739583
19.21	0.0	30.67	-11.46	-59.656428
25.16	0.0	40.21	-15.05	-59.817170

Figura 1.5: Salida del programa JAVA

También he creado un programa que permite calcular los dos transportistas más baratos, introduciendo por pantalla el código del país donde se va a enviar el pedido, las dos primeras cifras del código postal y el peso del pedido.

El código JAVA se encuentra en el **Anexo A.5.**

Para que los trabajadores de la empresa sean capaces de utilizar mi programa tenemos que transformar mi archivo ".java" en un ejecutable ".exe". Así tendríamos una aplicación de escritorio fácil de manejar para sus usuarios. Para hacer este cambio de formato he tenido que buscar mucha información y descargarme muchos tutoriales de internet. He probado con un programa que se llama *JSmoothGen* que convierte archivos ".class" o ".java" en ".exe", pero tras varios intentos, no ha funcionado. He llegado a la conclusión de que es debido a que mi programa no usa interfaces gráficas, todos los datos son pedidos por consola y el resultado también se muestra en consola.

A continuación elaboré el mismo programa pero usando *interfaces gráficas* para ver si así se solucionaba mi problema.

El código JAVA usando interfaces gráficas se encuentra en el **Anexo A.6.** Pero el problema no se soluciona con esto.

Después de contactar con mi profesor de programación, Pedro García, he conseguido solucionar el problema. Lo que pasaba era que cuando el programa iba a buscar la información necesaria para calcular el resultado, no sabía dónde tenía que ir a buscar, ya que cuando construyo el archivo ".jar", éste sólo comprime los archivos que tienen el formato ".class".

Para construir el archivo ".jar" he tenido que descargar un *plugin* de Eclipse que se llama: **net.sf.fjep.fatjar_ 0.0.31**.

Para solucionar el problema he declarado una variable con el nombre dirDatos, y ahí he escrito la ruta de los archivos ".csv" necesarios para poder ejecutar el programa con éxito.

Además hemos modernizado la gestión administrativa de la empresa para llevar un control de ésta. Ahora los pedidos los tratamos de la siguiente forma: todas las mañanas hago una consulta SQL en la base de datos para obtener los pedidos de los clientes que se han hecho el día anterior. La consulta es la siguiente:

```
SELECT id_order, id_customer, lastname, firstname, email, total_paid, invoice_number,
    payment_method, `ps_orders`.date_add, total_products_wt
FROM `ps_orders` JOIN `ps_customer` USING (id_customer) JOIN `ps_order_payment`
WHERE `ps_orders`.date_add LIKE (
'2015-04-%'
)
AND `ps_order_payment`.order_reference= `ps_orders`.reference
ORDER BY `ps_orders`.`invoice_number` ASC
```

La clase encargada de tratar el archivo ".cvs" de la consulta se encuentra en el **Anexo A.7.**

También he hecho un programa JAVA que trata el resultado de esta consulta y lo transforma en un ".csv" con el que vamos a trabajar. He creado direcciones Gmail para los trabajadores de la empresa que se dedican al tema administrativo ya que nos vamos a ayudar de una herramienta que nos ofrece éste: **Google Drive**, una aplicación para crear, compartir y guardar todas tus cosas donde colaborar con otros usuarios al que puedes acceder en todos tus dispositivos. Lo que he hecho ha sido crear un Excel en el Drive y compartirlo con ellos. He protegido el documento

por celdas de forma que cada trabador sólo puede modificar las celdas que le he asignado. Así es menos posible que el Excel contenga errores, y si los contiene, que se sepa quién ha sido el autor del fallo. He aprendido a manejar muchas herramientas del Excel que desconocía como, por ejemplo:

-Vistas de filtros: El objetivo de las vistas de filtro es poder crear, nombrar y guardar filtros que podemos utilizar sin cambiar el modo en que otros ven la hoja de cálculo, ayudando así a trabajar de forma independiente viendo lo que más nos interesa, sin afectar al resto de personas que usen la misma hoja.

-Validación de datos: Te permite crear listas desplegables en celdas. Tiene dos opciones: validación en relación con un intervalo o listas personalizadas. Si en una celda con una lista desplegable introduces un valor diferente de las opciones que se ofrecen, aparecerá un error de validación de datos en la celda.

-Formato condicional: Cuando tienes una tabla con muchos datos, es difícil saber qué es lo importante. El formato condicional permite cambiar el estilo de las celdas según sus valores.

Cada dos días hago una copia de seguridad de este Excel, aunque Google Drive tiene un panel de historial de revisión que permite ver de un vistazo todos los cambios que ha realizado en un documento cada colaborador. Aunque no funciona exactamente como una herramienta de control de cambios, el historial de revisión te permite ver y volver a versiones anteriores de hoja de cálculo, así como ver qué colaboradores han realizado cambios en cada una de las versiones.

Como este Excel contiene mucha información y se hace pesado trabajar con él, ya que tienes que moverte por todo el documento, lo que he hecho ha sido crear una vista para cada trabajador, pues cada uno necesita unas celdas específicas. Mediante el uso de relaciones entre las tablas consigo que los cambios sean visibles en el resto.

Con esta nueva forma de trabajar conseguimos, además de un buen control de la gestión de pedidos, que los trabajadores no necesiten preguntarse entre ellos e ir a buscar en correos o donde sea ya que toda la información está en el Excel.

El programa JAVA se encuentra en el **Anexo A.8.**

Un ejemplo de su ejecución sería:

4390	3903			Laurence	275.88	41
4391	3852			Aurelie	348.55	41
4392	4063			Nicolas	25.78	41
4393	1077			Valerie	1805.26	41
4394	4064			Claire	26.77	41
4395	4066			DÃ©borah	112.76	41
4396	3780			Carine	1933.38	41
4397	4068			Sophie	777.74	41
4398	4070			Catherine	429.8	41
4399	3603			GÃ©rard	536.84	41
4400	4071			Delphine	534.27	41
4401	4072			Delphine	349.16	41
4402	4073			Martine	26.77	41
4403	4074			Laure	29.74	41
4404	4075			Claire	30.73	41
4405	4079			Raphael	139.0	41
4406	4081			Magali	28.74	41

Figura 1.6: Salida

Carte bancaire PAYBOX		Préparation en cours	4/7/15 11:50	MAINZU
Carte bancaire PAYBOX		En cours de livraison	4/7/15 11:51	
Carte bancaire PAYBOX		Préparation en cours	4/7/15 12:09	ECHANTILLON
Carte bancaire PAYBOX		Préparation en cours	4/7/15 15:48	**TOTAL**
				CEVICA
				MAINZU
				PERONDA
				PAZOLA
Carte bancaire PAYBOX		En cours de livraison	4/7/15 17:40	ECHANTILLON
Carte bancaire PAYBOX		En cours de livraison	4/7/15 18:26	MAINZU
Carte bancaire PAYBOX		En cours de livraison	4/7/15 21:32	**TOTAL**
				MAINZU
	14842708/14-04			CIFRE
Carte bancaire PAYBOX		Préparation en cours	4/7/15 22:26	**TOTAL**
	14842700/14-04			VIVES
				COMPLEMENTTO
Carte bancaire PAYBOX		Préparation en cours	4/8/15 9:17	PAZOLA
Carte bancaire PAYBOX		En cours de livraison	4/8/15 9:28	PERONDA
Devis			4/8/15 9:56	
Carte bancaire PAYBOX		Préparation en cours	4/8/15 10:17	MAINZU
Carte bancaire PAYBOX		Livré	4/8/15 10:30	ECHANTILLON
Carte bancaire PAYBOX		Livré	4/8/15 11:57	ECHANTILLON
Carte bancaire PAYBOX		Préparation en cours	4/8/15 13:02	ECHANTILLON
Carte bancaire PAYBOX		Préparation en cours	4/8/15 15:16	AZULIBER
Carte bancaire PAYBOX		En cours de livraison	4/8/15 17:28	ECHANTILLON

Figura 1.7: Salida

También he creado las consultas para obtener el producto más vendido y la muestra más vendida por meses y con el resultado de ésta lo que hacemos es mirar los márgenes de beneficio. Para ello cogemos las 30 muestras y los 30 productos más vendidos de cada mes, así sabemos los productos que son tendencia y podemos ganar tiempo, por ejemplo, pidiendo palets de estos productos para así siempre tener stock de estos en la nave. De esta forma el pedido estaría listo antes.

La consulta SQL que te muestra los productos más vendidos por meses es la siguiente:

```
1
2SELECT SUM(  `ps_order_detail`.product_quantity ) AS cantidadVendida,  `
    ps_product`.reference AS ref,  `ps_order_detail`.product_name,  `ps_product`.
    supplier_reference
3FROM  `ps_product`
4JOIN  `ps_order_detail`
5JOIN  `ps_order_invoice`
6WHERE  `ps_product`.id_product =  `ps_order_detail`.product_id
7AND  `ps_order_invoice`.id_order_invoice =  `ps_order_detail`.id_order_invoice
8AND  `ps_order_invoice`.date_add LIKE (
9 '2014-10%'
10)
11AND  `ps_order_detail`.product_price >5
12GROUP BY  `ps_order_detail`.product_id,  `ps_product`.reference,  `
    ps_order_detail`.product_name,  `ps_product`.supplier_reference
13ORDER BY  `cantidadVendida` DESC
```

La consulta SQL que te devuelve las muestras más pedidas por meses es la siguiente:

```
1
2SELECT SUM(  `ps_order_detail`.product_quantity ) AS cantidadVendida,  `
    ps_product`.reference AS ref,  `ps_order_detail`.product_name,  `ps_product`.
    supplier_reference
3FROM  `ps_product`
4JOIN  `ps_order_detail`
5JOIN  `ps_order_invoice`
6WHERE  `ps_product`.id_product =  `ps_order_detail`.product_id
7AND  `ps_order_invoice`.id_order_invoice =  `ps_order_detail`.id_order_invoice
8AND  `ps_order_invoice`.date_add LIKE (
9 '2014-10%'
10)
11AND  `ps_order_detail`.product_price <5
12GROUP BY  `ps_order_detail`.product_id,  `ps_product`.reference,  `
    ps_order_detail`.product_name,  `ps_product`.supplier_reference
13ORDER BY  `cantidadVendida` DESC
```

Para la gestión de costes hemos comprado el programa de gestión financiera *Ciel Compta Evolution*. Este programa es una herramienta que nos ayuda a controlar la contabilidad (general y analítica, y presupuestaria) de nuestra empresa, y analiza y administra a medida su actividad. Algunas de las características que nos ofrece son:

-Te permite registrar rápidamente escrituras gracias a la afectación automática de la cuenta de IVA (proveedores, clientes, cargas, productos) y editar sus estados contables.

-Calcular estadísticas, gráficos, los resultados del mes...

-Controlar el presupuesto y las desviaciones con sus previsiones

-Consultar la tesorería preventiva para anticipar necesidades de financiación o hacer simulaciones.

-Generar automáticamente el cuadro de amortización.

Para ello lo que hago es pasarle al programa un documento ".csv" de cada mes que contiene a fecha de facturación, el código de factura, el código del país, el IVA facturado, el precio del transporte sin IVA, precio total de el/los producto/s sin IVA, total facturado con IVA, total facturado sin IVA y el descuento aplicado a la factura.

Este documento lo extraigo de la base de datos con esta consulta SQL:

```
1
2
3 SELECT DISTINCT datef, nom, facnumber, fk_pays, tva, trans, `erp_facture`.
      total_ttc, total, erp_paiement.note, si.dto
4 FROM `erp_facture`
5 LEFT OUTER JOIN `erp_paiement`
6 USING ( rowid )
7 JOIN `erp_societe`
8 JOIN `erp_facturedet`
9 JOIN (
10 SELECT facnumber, COALESCE( s.dto, 0 ) AS dto
11 FROM (
12
13 SELECT facnumber, SUM(erp_facturedet.total_ttc) AS dto
14 FROM `erp_facturedet`
15 JOIN `erp_facture`
16 WHERE erp_facturedet.fk_facture = erp_facture.rowid
17 AND erp_facturedet.total_ttc LIKE (
18 '-%'
19 )
20 AND datef LIKE (
21 '2015-01-%'
22 )
23 GROUP BY facnumber
24 ) AS s
25 RIGHT OUTER JOIN erp_facture
26 USING ( facnumber )
27 WHERE datef LIKE (
28 '2015-01-%'
29 )
30 ) AS si
31 USING ( facnumber )
32 JOIN (
33
34 SELECT facnumber,description,COALESCE(total_ht,0) as trans
35 FROM(
36 SELECT datef,facnumber, erp_facturedet.description,total_ht
37 FROM `erp_facture` JOIN `erp_facturedet`
38 WHERE erp_facturedet.fk_facture = erp_facture.rowid
39 AND(
40 description LIKE (
41 'TRANS%'
42 )
43 OR description LIKE (
44 'TRASN%'
45 )
46 OR description LIKE (
47 'Frais%'
48 )
49 OR description LIKE (
50 'Ship%'
51 )
```

```
52 OR description LIKE (
53 'AVOIR_F%'
54 )
55 OR description LIKE (
56 'ENVIO'
57 )
58 OR description LIKE (
59 'PORT'
60 )
61 )AND datef LIKE (
62 '2015-01-%'
63 )
64 ) AS factsConTrans RIGHT OUTER JOIN
65 `erp_facture` USING (facnumber)
66 WHERE erp_facture.datef LIKE (
67 '2015-01-%'
68 )
69
70 ) AS trans USING (facnumber)
71 WHERE erp_facture.fk_soc = erp_societe.rowid
72 AND erp_facturedet.fk_facture = erp_facture.rowid
73 AND `erp_facture`.`datef` LIKE (
74 '2015-01%'
75 )
76 AND erp_facture.paye LIKE 1
77 ORDER BY `erp_facture`.`datef` ASC
```

Un ejemplo de su exportación sería:

Fecha	Nombre	numero de factura	Código de país	IVA	Precio transporte (sin iva
2015-01-02		FA1501-3294	1	45,18	57,
2015-01-02		FA1501-3295	1	121,11	74,1
2015-01-02		FA1501-3296	1	26,1	40,8
2015-01-03		FA1501-3292	1	281,09	
2015-01-03		FA1501-3297	1	4,93	19,8
2015-01-03		FA1501-3298	1	28,94	40,8
2015-01-03		FA1501-3299	1	5,57	19,8
2015-01-03		FA1501-3300	1	31,42	40,8
2015-01-04		FA1501-3301	1	263,59	
2015-01-04		FA1501-3302	1	4,77	19,8
2015-01-04		FA1501-3303	1	24,86	40,8
2015-01-04		FA1501-3304	1	4,13	19,8
2015-01-04		FA1501-3305	1	4,93	19,8
2015-01-05		FA1501-3306	1	4,93	19,8
2015-01-05		FA1501-3307	1	4,29	19,8
2015-01-05		FA1501-3308	1	11,49	19,8
2015-01-05		FA1501-3309	1	5,73	19,8
2015-01-05		FA1501-3310	1	128,03	74,1
2015-01-05		FA1501-3311	1	17,8	40,8
2015-01-05		FA1501-3312	1	4,13	19,8
2015-01-05		FA1501-3313	1	46,52	57,
2015-01-06		FA1501-3293	140	5,89	19,8
2015-01-06		FA1501-3314	1	4,13	19,8
2015-01-06		FA1501-3315	1	5,09	19,8
2015-01-06		FA1501-3316	1	17,8	40,8
2015-01-06		FA1501-3317	1	12,31	19,8
2015-01-06		FA1501-3318	1	5,09	19,8
2015-01-07		FA1501-3319	1	8,79	19,8
2015-01-07		FA1501-3320	1	76,26	74,1
2015-01-07		FA1501-3321	1	4,77	19,8
2015-01-07		FA1501-3322	1	161,58	
2015-01-07		FA1501-3358	1	48,77	57,
2015-01-08		FA1501-3323	1	8,95	19,8
2015-01-08		FA1501-3324	1	73,47	74,1
2015-01-08		FA1501-3325	1	21,48	40,8
2015-01-08		FA1501-3326	1	4,77	19,8
2015-01-09		FA1501-3327	1	73,41	
2015-01-09		FA1501-3330	1	24,12	40,8

Figura 1.8: Salida

Con la ayuda de Google me he descargado tutoriales para aprender el funcionamiento del programa Ciel Compta Evolution.

Además he seguido trabajando con los distintos documentos excels con los que ahora se controlan todos los pedidos de la empresa, completándolos para su mejora. Por ejemplo, he añadido el código de facturación de *Dolibarr*. Esto ha sido bastante difícil ya que realmente no existe ninguna tabla que relacione *Dolibarr* con *Prestashop* y toda la información que ponía en el Excel era de *Prestashop*. Para ello lo que hago es comparar las fechas de las facturas y los e-mails de los clientes.

La consulta SQL es la siguiente:

```
1
2SELECT `ps_orders`.id_order, `ps_orders`.id_customer, `ps_customer`.lastname,
    `ps_customer`.firstname, `ps_customer`.email, `ps_orders`.total_paid, `
    ps_orders`.invoice_number,`erp_facture`.facnumber, `ps_orders`.payment, `
    ps_orders`.date_add
3FROM `ps_orders`
4JOIN `ps_customer`
5USING (id_customer)
6JOIN `erp_societe`
7JOIN `erp_facture`
8WHERE `ps_customer`.email=`erp_societe`.email
9AND `erp_societe`.rowid=`erp_facture`.fk_soc
10AND `ps_orders`.total_paid=`erp_facture`.total_ttc
11AND `erp_facture`.datef LIKE (
12'2015-04-%'
13)
14AND `ps_orders`.date_add LIKE (
15 '2015-04-%'
16)
17ORDER BY `ps_orders`.`invoice_number` ASC
```

Pero antes de hacer esta consulta tengo que hacer la otra, ya que los presupuestos no tienen este número de facturación y si sólo hiciese esta última perdería información, además las facturas de *Dolibarr* sólo se ven reflejadas en esta última consulta. He creado las nuevas hojas de cálculo de Google para la gestión de los pedidos de Mayo tal como hice en la quincena anterior correspondiente al mes de Abril, y todo lo que ello conlleva.

También tenemos que analizar algunos presupuestos de importes bastante altos para ver si seríamos capaces de afrontarlos ya que este tipo de presupuestos no se cobran hasta que el cliente ha recibido su pedido, pero nosotros sí que tenemos que pagar a los proveedores antes. Analizar el riego que tenemos en caso de que saliese mal la compra y el cliente no nos pagase, cuanto nos costaría recuperarnos de esa perdida.

Sigo introduciendo todos los pedidos en el Excel cada día para que cada trabajador pueda completar sus columnas y tener toda la información necesaria para poder hacer su trabajo del día a día.

Empezamos a mirar el programa de contabilidad, intentamos importarle las escrituras, que era la información que yo había sacado de le base de datos la semana anterior, pero no nos aclaramos, además de que el programa es en francés y no tiene opción de cambiar el idioma.

Además empezamos una nueva tarea, limpiar nuestra base de datos de productos. Para ello lo primero que vamos a hacer es descatalogar algunos productos que nunca se hayan vendido o que, por ejemplo, se hayan pedido sólo muestras de estos pero nunca se hayan vendido o se hayan vendido muy pocas veces. Para ello he creado una consulta en SQL que muestra los productos y muestras que se habían vendido durante el año 2014 y lo que llevamos del 2015, la cantidad de veces que se habían vendido, el precio de venta y los proveedores. De esta forma podíamos decidir con qué proveedores dejar de trabajar y qué productos descatalogar.

La consulta SQL es la siguiente:

```
1
2SELECT `ps_supplier`.name,`ps_product`.id_category_default,`ps_category_lang`.
    name,`ps_product`.reference,`ps_order_detail`.product_price, SUM(  `
    ps_order_detail`.product_quantity ) AS cantidadVendida
3FROM `ps_order_detail`  JOIN  `ps_product` JOIN `ps_product_supplier` JOIN `
    ps_supplier` JOIN `ps_orders` JOIN `ps_category_lang`
4WHERE `ps_product`.id_product=`ps_order_detail`.product_id
5AND `ps_product_supplier`.id_product=`ps_order_detail`.product_id
6AND `ps_product_supplier`.id_product_attribute=`ps_order_detail`.
    product_attribute_id
7AND `ps_supplier`.id_supplier=`ps_product_supplier`.id_supplier
8AND `ps_orders`.id_order=`ps_order_detail`.id_order
9AND `ps_category_lang`.id_category=`ps_product`.id_category_default
10AND
11 (
12`ps_orders`.date_add LIKE (
13 '2015-%'
14)
15OR
16
17`ps_orders`.date_add LIKE (
18 '2014-%'
19)
20)
21GROUP BY `ps_supplier`.name,  `ps_product`.id_category_default,  `ps_product`.
    reference,  `ps_order_detail`.product_price;
```

Y un ejemplo de su exportación sería:

Proveedor	categ	Nombre categoría	ref producto	precio de venta	Cantidad vendida
ARCANA	391	Treewood 22x90 cm	TR2004001	0.000000	22
ARCANA	391	Treewood 22x90 cm	TR2004001	0.825000	88
ARCANA	391	Treewood 22x90 cm	TR2004001	1.500000	22
ARCANA	391	Treewood 22x90 cm	TR2004002	0.825000	66
ARCANA	391	Treewood 22x90 cm	TR2004002	0.833334	22
ARCANA	391	Treewood 22x90 cm	TR2004003	0.000000	22
ARCANA	391	Treewood 22x90 cm	TR2004003	0.825000	22
ARCANA	391	Treewood 22x90 cm	TR2004003	1.500000	44
ARCANA	391	Treewood 22x90 cm	TR2004004	0.825000	88
ARCANA	393	Universal 20x50 cm	UN2001001	0.825000	22
ARCANA	393	Universal 20x50 cm	UN2001002	0.825000	22
ARCANA	393	Universal 20x50 cm	UN2001003	0.825000	22
ARCANA	393	Universal 20x50 cm	UN2001007	0.825000	22
ARCANA	393	Universal 20x50 cm	UN2001008	0.000000	22
ARCANA	393	Universal 20x50 cm	UN2001008	0.825000	44
ARCANA	393	Universal 20x50 cm	UN2001009	4.180602	308
ARCANA	393	Universal 20x50 cm	UN2001009	18.000000	176
ARCANA	393	Universal 20x50 cm	UN2001010	0.825000	22
ARCANA	393	Universal 20x50 cm	UN2001010	1.500000	22
ARCANA	529	Tempo	TE2001001	0.000000	44
ARCANA	529	Tempo	TE2001001	0.825000	110
ARCANA	529	Tempo	TE2001001	1.500000	66
ARCANA	529	Tempo	TE2001001	41.880000	572
ARCANA	529	Tempo	TE2001002	0.000000	22
ARCANA	529	Tempo	TE2001002	0.825000	88
ARCANA	529	Tempo	TE2001002	1.500000	22
ARCANA	529	Tempo	TE2001002	41.880000	154
ARCANA	529	Tempo	TE2001003	0.000000	22
ARCANA	529	Tempo	TE2001003	0.825000	22
ARCANA	529	Tempo	TE2001003	41.880000	374
ARCANA	529	Tempo	TE2001004	0.000000	22
ARCANA	529	Tempo	TE2001004	1.500000	22
ARCANA	529	Tempo	TE2001005	0.000000	44
ARCANA	529	Tempo	TE2001005	0.825000	66
ARCANA	529	Tempo	TE2001005	1.500000	44
ARCANA	529	Tempo	TE2001005	41.880000	330
ARCANA	529	Tempo	TE2001006	0.000000	22
ARCANA	529	Tempo	TE2001006	0.825000	132
ARCANA	529	Tempo	TE2001006	41.880000	506
AZULIBER	411	Pizarra 30x60 cm	PI1304001	0.825000	44
AZULIBER	411	Pizarra 30x60 cm	PI1304001	1.500000	66
AZULIBER	411	Pizarra 30x60 cm	PI1304001	25.000000	1188
AZULIBER	411	Pizarra 30x60 cm	PI1304002	0.825000	22
AZULIBER	411	Pizarra 30x60 cm	PI1304002	1.500000	22
AZULIBER	411	Pizarra 30x60 cm	PI1304003	0.825000	44
AZULIBER	411	Pizarra 30x60 cm	PI1304003	25.000000	462
AZULIBER	411	Pizarra 30x60 cm	PI1304101	7.441472	1408
AZULIBER	412	Calzada 35x35 cm	CA1302001	0.000000	22
AZULIBER	412	Calzada 35x35 cm	CA1302001	0.825000	44
AZULIBER	412	Calzada 35x35 cm	CA1302002	0.000000	22
AZULIBER	412	Calzada 35x35 cm	CA1302003	0.825000	44

Figura 1.9: Salida

También he tenido que crear un Excel para los trabajadores de la nave con todas las referencias internas de las muestras y productos y la referencia de proveedor de cada una.

La consulta SQL es la siguiente:

```
1
2SELECT `ps_product_attribute`.reference, `ps_product`.supplier_reference,`
    ps_supplier`.name
3FROM `ps_product_attribute` JOIN `ps_product` JOIN `ps_product_supplier` JOIN `
    ps_supplier`
4WHERE `ps_product`.id_product=`ps_product_attribute`.id_product
5AND `ps_product_supplier`.id_product = `ps_product`.id_product
6AND `ps_supplier`.id_supplier=`ps_product_supplier`.id_supplier
```

Y un ejemplo de su resultado es:

ID17050010	IDOLE WHITE 25X70	VENUS
ID17050011	IDOLE WHITE 25X70	VENUS
ID17050030	IDOLE GRAPHITE 25X70	VENUS
ID17050031	IDOLE GRAPHITE 25X70	VENUS
ID17050040	IDOLE WAVE WHITE 25X70	VENUS
ID17050041	IDOLE WAVE WHITE 25X70	VENUS
ID17050060	IDOLE STATUE WHITE 25X70	VENUS
ID17050061	IDOLE STATUE WHITE 25X70	VENUS
ID17050080	IDOLE SEMIRAMIS WHITE 25X70	VENUS
ID17050081	IDOLE SEMIRAMIS WHITE 25X70	VENUS
ID17050120	IDOLE PRISMA WHITE 25X70	VENUS
ID17050121	IDOLE PRISMA WHITE 25X70	VENUS
ID17050140	IDOLE OPTICAL WHITE 25X70	VENUS
ID17050141	IDOLE OPTICAL WHITE 25X70	VENUS
ID17050160	IDOLE WAVE GRAPHITE 25X70	VENUS
ID17050161	IDOLE WAVE GRAPHITE 25X70	VENUS
ID17050170	IDOLE STATUE GRAPHITE 25X70	VENUS
ID17050171	IDOLE STATUE GRAPHITE 25X70	VENUS
ID17050180	IDOLE SEMIRAMIS GRAPHITE 25X70	VENUS
ID17050181	IDOLE SEMIRAMIS GRAPHITE 25X70	VENUS
ID17050190	IDOLE PRISMA GRAPHITE 25X70	VENUS
ID17050191	IDOLE PRISMA GRAPHITE 25X70	VENUS
ID17050200	IDOLE OPTICAL GRAPHITE 25X70	VENUS
ID17050201	IDOLE OPTICAL GRAPHITE 25X70	VENUS
ON17010010	ONIRA WHITE	VENUS
ON17010011	ONIRA WHITE	VENUS
ON17010020	ONIRA POIS YELLOW	VENUS
ON17010021	ONIRA POIS YELLOW	VENUS
ON17010030	ONIRA POIS TURQUOISE	VENUS
ON17010031	ONIRA POIS TURQUOISE	VENUS
ON17010040	ONIRA POIS WATERMELON	VENUS
ON17010041	ONIRA POIS WATERMELON	VENUS
ON17010050	ONIRA POIS VIOLET	VENUS
ON17010051	ONIRA POIS VIOLET	VENUS
ON17010060	ONIRA POIS BLACK	VENUS
ON17010061	ONIRA POIS BLACK	VENUS
ON17010070	ONIRA YELLOW	VENUS
ON17010071	ONIRA YELLOW	VENUS
ON17010080	ONIRA TURQUOISE 25X70	VENUS
ON17010081	ONIRA TURQUOISE 25X70	VENUS
ON17010090	ONIRA WATERMELON 25X70	VENUS
ON17010091	ONIRA WATERMELON 25X70	VENUS
ON17010100	ONIRA VIOLET 25X70	VENUS
ON17010101	ONIRA VIOLET 25X70	VENUS
ON17010110	ONIRA BLACK 25X70	VENUS
ON17010111	ONIRA BLACK 25X70	VENUS
ON17011120	ONIRA WHITE 33.6X33.6	VENUS
ON17011121	ONIRA WHITE 33.6X33.6	VENUS
ON17011130	ONIRA YELLOW 33.6X33.6	VENUS
ON17011131	ONIRA YELLOW 33.6X33.6	VENUS

Figura 1.10: Salida

Si la referencia interna acaba en 0 es una muestra, y si acaba en 1, es un producto.

También me he dedicado a sacar los márgenes de beneficio de las ventas que hemos hecho durante el mes de Abril. Cuando nos hemos puesto a mirar porque algunos son negativos nos hemos dado cuenta de que esto era porqué hubo una semana que los pedidos no pasaron en *Dolibarr*. Hasta que el informático no pase esos pedidos y solucione el problema, no tendremos los márgenes de Abril bien.

Para calcular estos márgenes he tenido que introducir en la base de datos esta consulta para obtener la información necesaria y de ahí poder calcular los márgenes de beneficio:

```
1
2SELECT `erp_facture`.datef,`erp_facture`.facnumber,ROUND(SUM(`ps_order_detail`.
    product_quantity * `ps_order_detail`.product_weight),2) AS pesosTotFact, ZIP
    ,trans.transSINiva,trans.transCONiva,`ps_orders`.total_paid_tax_excl,ROUND(
    SUM(`ps_product_supplier`.product_supplier_price_te* `ps_order_detail`.
    product_quantity),2) AS costeMat,fk_pays
3FROM `ps_orders` JOIN `ps_order_detail`
4JOIN `ps_customer`
5USING (id_customer)
6JOIN `erp_societe` JOIN `erp_facture`
7JOIN (
8SELECT facnumber,description,COALESCE(total_ht,0) as transSINiva,COALESCE(
    ttconiva,0) as transCONiva
9FROM(
10SELECT datef,facnumber, erp_facturedet.description,total_ht,`erp_facturedet`.
    total_ttc as ttconiva
11FROM `erp_facture` JOIN `erp_facturedet`
12WHERE erp_facturedet.fk_facture = erp_facture.rowid
13AND(
14description LIKE (
15 'TRANS%
16)
17OR description LIKE (
18 'TRASN%
19)
20OR description LIKE (
21 'Frais%
22)
23OR description LIKE (
24 'Ship%
25)
26OR description LIKE (
27 'AVOIR_F%
28)
29OR description LIKE (
30 'ENVIO'
31)
32OR description LIKE (
33 'PORT'
34)
35)AND datef LIKE (
36 '2015-04-%'
37)
38) AS factsConTrans RIGHT OUTER JOIN
39 `erp_facture` USING (facnumber)
40WHERE erp_facture.datef LIKE (
41 '2015-04-%'
42)
43
44) AS trans USING (facnumber)
```

```
45 JOIN `ps_product_supplier`
46 WHERE `ps_orders`.id_order=`ps_order_detail`.id_order
47 AND `ps_customer`.email=`erp_societe`.email
48 AND `erp_societe`.rowid=`erp_facture`.fk_soc
49 AND `ps_orders`.total_paid=`erp_facture`.total_ttc
50 AND `ps_product_supplier`.id_product=`ps_order_detail`.product_id
51 AND `ps_order_detail`.product_attribute_id=`ps_product_supplier`.
       id_product_attribute
52 AND `erp_facture`.datef LIKE (
53 '2015-04-%'
54 )
55 AND `ps_orders`.date_add LIKE (
56  '2015-04-%'
57 )
58 GROUP BY `erp_facture`.datef,`erp_facture`.facnumber
59 ORDER BY  `erp_facture`.`datef` ASC
```

Para comprobar que el resultado de la consulta es correcto, lo hago con el mes de marzo y me doy cuenta de que no aparecen todas las facturas, hay unas 20 que se pierden. Rastreando cada una de éstas que no aparecen he podido concluir de momento que:

-algunas es debido a que el producto no está introducido en la tabla de producto_proveedor de *Prestashop*.

-otras es debido a que la factura es de *Dolibarr*. Entonces, al hacer los JOINS con las tablas de *Prestashop* se pierden.

-otras son las devoluciones (AV) que aún no está claro por qué no aparecen, pero éstas no las tenemos en cuenta a la hora de calcular los márgenes.

-otras es debido a que la forma de unir las tablas de *Prestashop* con las de *Dolibarr* era mediante el email de los clientes. Como hay clientes que no tienen guardado el email, lo que he hecho ha sido otra consulta donde uso el nombre y apellidos del cliente para enlazar las tablas. Como en *Dolibarr* aparece en diferentes columnas el nombre y los apellidos he tenido que concatenarlas.

La consulta SQL es la siguiente:

```
1
2 SELECT `erp_facture`.datef,`erp_facture`.facnumber,ROUND(SUM(`ps_order_detail`.
      product_quantity * `ps_order_detail`.product_weight),2) AS pesosTotFact, ZIP
      ,trans.transSINiva,trans.transCONiva,`ps_orders`.total_paid_tax_excl,ROUND(
     SUM(`ps_product_supplier`.product_supplier_price_te* `ps_order_detail`.
     product_quantity),2) AS costeMat,fk_pays
3 FROM `erp_facture` JOIN `erp_societe` JOIN (SELECT CONCAT(firstname, ' ',
     lastname) AS nom, `ps_customer`.*
4 FROM `ps_customer`) as nyp USING (nom) JOIN `ps_orders` JOIN `ps_order_detail`
     JOIN `ps_product_supplier`
5 JOIN (
6 SELECT facnumber,description,COALESCE(total_ht,0) as transSINiva,COALESCE(
     ttconiva,0) as transCONiva
7 FROM(
8 SELECT datef,facnumber, erp_facturedet.description,total_ht,`erp_facturedet`.
     total_ttc as ttconiva
9 FROM `erp_facture` JOIN `erp_facturedet`
```

```
WHERE erp_facturedet.fk_facture = erp_facture.rowid
AND(
description LIKE (
 'TRANS%
)
OR description LIKE (
 'TRASN%
)
OR description LIKE (
 'Frais%
)
OR description LIKE (
 'Ship%
)
OR description LIKE (
 'AVOIR_F%
)
OR description LIKE (
 'ENVIO'
)
OR description LIKE (
 'PORT'
)
)AND datef LIKE (
 '2015-03-%'
)
) AS factsConTrans RIGHT OUTER JOIN
`erp_facture` USING (facnumber)
WHERE erp_facture.datef LIKE (
 '2015-03-%'
)

) AS trans USING (facnumber)
WHERE `erp_facture`.datef LIKE (
'2015-03-%'
)
AND `erp_societe`.rowid=`erp_facture`.fk_soc
AND `ps_orders`.id_customer = nyp.id_customer
AND  `ps_orders`.id_order=`ps_order_detail`.id_order
AND `ps_product_supplier`.id_product=`ps_order_detail`.product_id
GROUP BY `erp_facture`.datef,`erp_facture`.facnumber
ORDER BY  `erp_facture`.`datef` ASC
```

En esta consulta saca facturas que antes no me aparecían, pero aún no consigo que me salgan todas.

Un ejemplo donde utilizamos el resultado de esta consulta con el programa java que creé hace unas semanas, que calcula los márgenes de beneficio, es el siguiente:

Fecha envio	Código de factur	Peso	Destino	pr. transp	DSV	TRANSP
01/04/2015	FA1504-4667	10,42	35	11,26		UCX_EXPACK
01/04/2015	FA1504-4668	6,41	61	8,78		UCX_EXPACK
01/04/2015	FA1504-4669	175	87	60,25	40,71	SPAINTIR
01/04/2015	FA1504-4670	3,66	76	7,74		UCX_EXPACK
01/04/2015	FA1504-4671	2,91	69	7,47		UCX_EXPACK
01/04/2015	FA1504-4672	177,5	13	60,25	43,91	SPAINTIR
02/04/2015	FA1504-4674	1260	75	193,18		UCX
02/04/2015	FA1504-4675	5	75	8,07		UCX_EXPACK
02/04/2015	FA1504-4679	54,5	69	40,86	31,77	SPAINTIR
02/04/2015	FA1504-4680	27,25	22	17,7		UCX_EXPACK
02/04/2015	FA1504-4681	3,29	93	7,74		UCX_EXPACK
02/04/2015	FA1504-4682	2	75	7,18		UCX_EXPACK
03/04/2015	FA1504-4683	81,75	95	40,86	32,98	SPAINTIR
03/04/2015	FA1504-4684	13,48	44	12,1		UCX_EXPACK
03/04/2015	FA1504-4685	708,5	95	136,18		UCX
03/04/2015	FA1504-4686	62,5	78	40,86	32,98	SPAINTIR
03/04/2015	FA1504-4687	1,7	25	7,18		UCX_EXPACK
03/04/2015	FA1504-4688	305,73	69	79,86	71,11	UCX
03/04/2015	FA1504-4689	8,7	30	9,26		UCX_EXPACK
03/04/2015	FA1504-4691	16	33	12,8		UCX_EXPACK
04/04/2015	FA1504-4692	37,5	63	30,79	29,79	SPAINTIR
04/04/2015	FA1504-4693	18,68	61	14,02		UCX_EXPACK
04/04/2015	FA1504-4694	100	92	40,86	32,98	SPAINTIR
04/04/2015	FA1504-4695	44,52	38	30,77	27,86	SPAINTIR
04/04/2015	FA1504-4696	1610,44	77	244,48		UCX
05/04/2015	FA1504-4697	176	31	60,25	32,74	SPAINTIR
05/04/2015	FA1504-4698	43	68	30,77		SPAINTIR
05/04/2015	FA1504-4699	294	33	85,92	60,9	SPAINTIR
05/04/2015	FA1504-4700	3,68	68	7,74		UCX_EXPACK
05/04/2015	FA1504-4701	5,99	13	8,54		UCX_EXPACK
06/04/2015	FA1504-4702	1	59	6,66		UCX_EXPACK
06/04/2015	FA1504-4703	8,9	75	9,26		UCX_EXPACK
06/04/2015	FA1504-4704	39,12	75	30,77	29,06	SPAINTIR
06/04/2015	FA1504-4705	400	74	95,7	89,46	UCX
06/04/2015	FA1504-4706	331,5	75	79,86	74,73	UCX

Figura 1.11: Salida

Capítulo 2

Proyecto de fin de grado

2.1. Introducción

Existen datos que no tienen una definición precisa, como por ejemplo, "ser joven", "temperatura alta", "estatura media", "estar cerca de", etc. Para solucionar estos problemas aparecen los conjuntos difusos, como una extensión de los conjuntos clásicos, mediante los cuales podemos manipular información que tiene un alto grado de incertidumbre. Las bases de los conjuntos difusos (o borrosos) fueron presentados en 1965 por el profesor Lofti Zadeh de la Universidad de California, en Berkeley, con un artículo titulado "Fuzzy Sets". En este artículo Zadeh presenta unos conjuntos sin límites precisos, los cuales, según él, juegan un papel importante en el reconocimiento de formas, interpretación de significados, y especialmente en la abstracción, la esencia del proceso del razonamiento humano.

Como se ha comentado, los conjuntos difusos son una generalización de los conjuntos clásicos que nos permiten describir nociones imprecisas. De este modo, la pertenencia de un elemento a un conjunto pasa a ser cuantificada mediante un "grado de pertenencia". Dicho grado toma un valor en el intervalo [0,1]; si este grado toma el valor 0 significa que el elemento no pertenece al conjunto, si es 1 pertenece al conjunto y si es otro valor del intervalo (0,1), pertenece con cierto grado al conjunto. En lugar del intervalo [0,1], también se suelen considerar otros conjuntos ordenados más generales, como cadenas, retículos (completos), multiretículos, etc. Actualmente, los conjuntos difusos se utilizan en multitud de campos; en ciencias de la computación para recuperación de información, bases de datos relacionales difusas (SGBDRD), control difuso, ecuaciones diferenciales difusas, B-splines difusos, etc.

La Lógica Difusa es una lógica de múltiples valores, que permite que sean definidos los valores intermedios entre las evaluaciones convencionales como: verdadero o falso, sí o no, alto o bajo, etc. Nociones como "más alto" o "muy rápido" pueden ser formuladas matemáticamente y procesadas por computadoras a fin de aplicar una forma de "pensamiento humano" a la programación de computadores.

En los últimos años la Lógica Difusa se ha utilizado en distintos tipos de instrumentos, máquinas y en diversos ámbitos de la vida cotidiana. Algunos casos, por ejemplo, son los es-

tabilizadores de imágenes en grabadoras de vídeo, controladores de ascensores e ingeniería de terremotos. También se ha usado esta técnica en la industria, obteniéndose excelentes resultados, como en el caso del metro de Sendai en Japón, ya que permitía que el metro arrancara y frenara con gran suavidad, sin producir alteraciones entre los pasajeros.

Si dividimos todo lo que se ha conseguido gracias a la Lógica Difusa, podemos hacer tres grupos:

- **Productos creados para el consumidor:** Lavadoras difusas, hornos microondas (establece y afina la energía y el programa de cocción), sistemas térmicos, traductores lingüísticos, cámaras de vídeo, televisores, estabilizadores de imágenes digitales y sistemas de foco automático en cámaras fotográficas.
- **Sistemas:** Ascensores (reduce el tiempo de espera a partir del numero de personas), trenes, automóviles (como puede ser la transmisión, frenos y mejora de la eficiencia del uso del combustible en motores), controles de tráfico, sistemas de control de acondicionadores de aire que evitan las oscilaciones de temperatura y sistemas de reconocimiento de escritura.
- **Software:** Diagnóstico médico, seguridad, comprensión de datos, tecnología informática y bases de datos difusas para almacenar y consultar información imprecisa (uso del lenguaje Fuzzy Structured Query Language).

2.2. Conjuntos Difusos

2.2.1. Introducción

Sabemos que un conjunto es una colección de objetos bien especificados que poseen una propiedad común. Recordemos que se puede definir de diversas formas:

- Por enumeración de los elementos que lo componen. Para un conjunto E finito, de n elementos, tendríamos, por ejemplo, la siguiente representación: $E = \{a_1, a_2, ..., a_n\}$.
- Por descripción analítica de una propiedad que caracterice a todos los miembros del conjunto. Por ejemplo, $A = \{x \in \mathbb{R} | x \leq 7\}$.
- Usando la función característica (también llamada función de pertenencia) para definir sus elementos. Si llamamos $m_A : U \to \{0, 1\}$ a dicha función de pertenencia, siendo U el conjunto universal, tendremos que,

$$m_A(x) = \begin{cases} 1 & \text{si } x \in A \\ 0 & \text{si } x \notin A \end{cases}$$

  Así, un conjunto A está completamente definido por el conjunto de pares:

$$A = \{(x, m_A(x) : x \in U, m_A(x) \in \{0, 1\}\}.$$

  Es decir, si la función de pertenencia para un valor dado de x toma el valor 1, ese valor es un elemento del conjunto; por el contrario, si toma el valor cero, no pertenece al conjunto.

Ejemplo 1: Si $E = \{a, e, i, o, u\}$ es el conjunto de las vocales del alfabeto y $A = \{a, i, u\}$ un subconjunto del mismo, podríamos representarlos en la siguiente forma:

$$E = \{(a, 1), (e, 1), (i, 1), (o, 1), (u, 1)\}$$
$$A = \{(a, 1), (e, 0), (i, 1), (o, 0), (u, 1)\}$$

Para un conjunto difuso, sin embargo, la cuestión de pertenencia de un elemento al conjunto no es cuestión de todo o nada, sino que hay diferentes grados de pertenencia. La función de pertenencia puede tomar cualquier valor en el intervalo real [0,1].

Definición 1. Se define *Universo de Discurso* como el conjunto U de posibles valores que puede tomar la variable x.

Definición 2. La *función de pertenencia* $m_A(x)$ de un conjunto difuso A es una función:

$$m_A : U \to [0, 1],$$

quedando perfectamente definido un conjunto difuso A como sigue:

$$A = \{(x, m_A(x)) : x \in U, m_A(x) \in [0, 1]\}$$

Así, cualquier elemento x en U tiene grado de pertenencia $m_A(x) \in [0, 1]$.

Ejemplo 2: Supóngase que alguien quiere describir la clase de *animales terrestres veloces*. Algunos animales pertenecen definitivamente a esta clase, como el guepardo ó la gacela, mientras otros, como la tortuga o la araña, no pertenecen. Pero existe otro grupo de animales para los que es difícil determinar si son veloces o no. Utilizando notación difusa, el conjunto difuso para los animales *veloces* sería

$$\{(Guepardo, 1), (Avestruz, 0{,}9), (Liebre, 0{,}8), (Gacela, 0{,}7), (Gato, 0{,}4), ...\}$$

es decir, la liebre pertenece con grado de 0.8 a la clase de animales *veloces*, la gacela con grado de 0.7 y el gato con grado de 0.4.

Si se supone que C es un conjunto clásico finito $\{x_1, x_2, ..., x_n\}$, entonces una notación alternativa es

$$C = x_1 + x_2 + ... + x_n$$

siendo + una enumeración. A partir de ella, Zadeh propuso una notación más conveniente para conjuntos difusos. Así, otra forma de escribir el conjunto de los animales *veloces* del ejemplo 1 sería:

$$1/Guepardo + 0{,}9/Avestruz + 0{,}8/Liebre + 0{,}7/Gacela + 0{,}4/Gato$$

Es decir, se puede describir el conjunto difuso como sigue:

$$A = \frac{m_A(x_1)}{x_1} + \frac{m_A(x_2)}{x_2} + ... + \frac{m_A(x_n)}{x_n} = \sum_{i=1}^{n} \frac{m_A(x_i)}{x_i}$$

donde el símbolo de división no es más que un separador de los conjuntos de cada par, y el sumatorio es la operación de unión entre todos los elementos del conjunto. Cuando U es incontable o es continuo, se escribe la ecuación anterior como:

$$A = \int_U \frac{m_A(x)}{x}$$

Ejemplo 3: La figura 2.1 muestra algunos conjuntos difusos definidos en el universo de discurso Edad. Concretamente, se representan las funciones de pertenencia de los conjuntos difusos "joven", "maduro", "viejo".

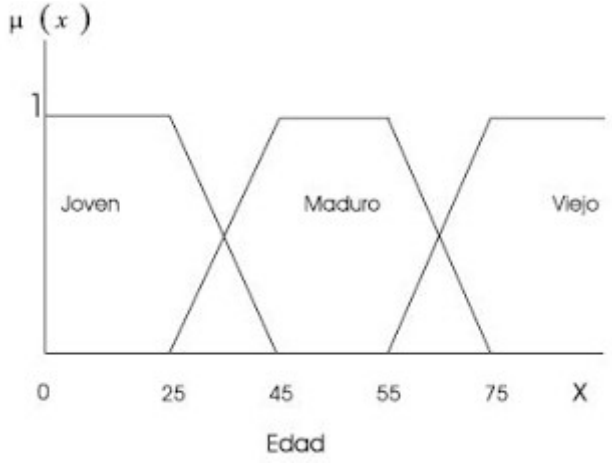

Figura 2.1: Ejemplo de conjuntos difusos

Se puede ver que los conjuntos difusos se superponen, de manera que un individuo podría tener un grado de pertenencia en dos conjuntos: "joven" y "maduro", indicando que posee cualidades asociadas a ambos conjuntos. Por ejemplo, una persona con 35 años tiene un grado de pertenencia 0.5 para el conjunto "joven" y 0.5 para el conjunto "maduro".

Algunas definiciones relacionadas con los conjuntos difusos son:

Definición 3. El *soporte* de un conjunto difuso A es el conjunto clásico que contiene todos los elementos de A cuyos grados de pertenencia no son cero. Esto se define por $S(A)$. Es decir:

$$S(A) = \{x \in U | m_A(x) > 0\}$$

Definición 4. Un conjunto difuso A es *convexo* si y sólo si X es *convexo* y

$$\forall x, y \in U, \forall \alpha \in [0,1] / m_A(\alpha x + (1-\alpha)y) \geq min(m_A(x), m_A(y)).$$

En la teoría de control difuso, es usual tratar sólo con conjuntos difusos *convexos.*

Definición 5. Se define la *altura* de un conjunto difuso A sobre X, que se denota por $Alt(A)$, como:

$$Alt(A) = \sup_{x \in U} m_A(x)$$

Definición 6. Dado un número $\alpha \in [0,1]$ y un conjunto difuso A, definimos el α-*corte* de A como el conjunto clásico A_α que tiene la siguiente función de pertenencia:

$$m_{A_\alpha}(x) = \begin{cases} 1 & \text{cuando } m_A(x) \geq \alpha \\ 0 & \text{en cualquier otro caso} \end{cases}$$

En definitiva, el α-*corte* se compone de aquellos elementos cuyo grado de pertenencia supera o iguala el umbral α.

2.2.2. Funciones de pertenencia

Existe una gran variedad de formas para las funciones de pertenencia asociadas a un conjunto difuso. Las más comunes son las siguientes:

1. Forma Singleton

$$A(x) = \begin{cases} 1 & \text{cuando } x = a \\ 0 & \text{cuando } x \neq a \end{cases}$$

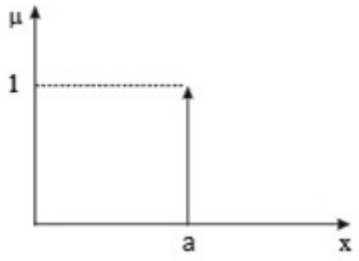

Figura 2.2: Ejemplo Forma Singleton

2. Forma Triangular

$$A(x) = \begin{cases} 0 & \text{si } x \leq a \\ (x-a)/(m-a) & \text{si } x \in (a, m] \\ (b-x)/(b-m) & \text{si } x \in (m, b) \\ 1 & \text{si } x \geq b \end{cases}$$

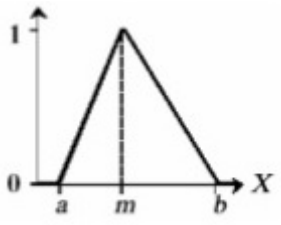

Figura 2.3: Ejemplo Forma Triangular

3. Forma S

$$A(x) = \begin{cases} 0 & \text{si } x \leq a \\ 2\{(x-a)/(b-a)\}^2 & \text{si } x \in (a, m] \\ 1 - 2\{(x-a)/(b-a)\}^2 & \text{si } x \in (m, b) \\ 1 & \text{si } x \geq b \end{cases}$$

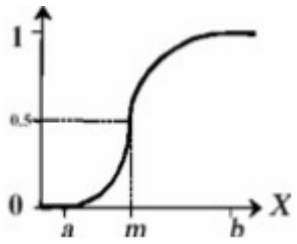

Figura 2.4: Ejemplo Forma S

4. Forma Trapezoidal

$$A(x) = \begin{cases} 0 & \text{si } (x \leq a) \text{ o } (x \geq d) \\ (x-a)/(b-a) & \text{si } x \in (a,b] \\ 1 & \text{si } x \in (b,c) \\ (d-x)/(d-c) & \text{si } x \in (b,d) \end{cases}$$

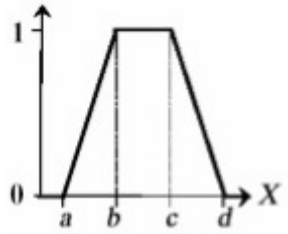

Figura 2.5: Ejemplo Forma Trapezoidal

2.2.3. Operaciones básicas con conjuntos difusos

Las operaciones más usadas en el contexto de los conjuntos difusos son las siguientes:

- Diremos que un conjunto A es un *subconjunto* difuso de B, es decir, $A \subseteq B$, si

  $$\forall x \in U : m_A(x) \leq m_B(x),$$

  y si existe, al menos, un punto $x \in U$ tal que $m_A(x) \neq m_B(x)$, entonces escribiremos que $A \subset B$.

- *Intersección*: La intersección de dos conjuntos, A y B, tiene la siguiente función de pertenencia:

  $$m_{A \cap B}(x) = \text{mín}\{m_A(x), m_B(x)\}$$

- *Unión*: La unión de dos conjuntos difusos, A y B, tiene la siguiente función de pertenencia:

  $$m_{A \cup B}(x) = \text{máx}\{m_A(x), m_B(x)\}$$

- *Negación*: La negación de A se denota por $\overline{A}$, y tiene la siguiente función de pertenencia:

$$m_{\overline{A}}(x) = 1 - m_A(x)$$

Ejemplo 5: En la figura 2.6 se observa un trapecio difuso A entre 5 y 8 (rojo), un triángulo difuso B entorno al 4 (verde) y la intersección de ambos conjuntos difusos (el resultado es la línea de color azul):

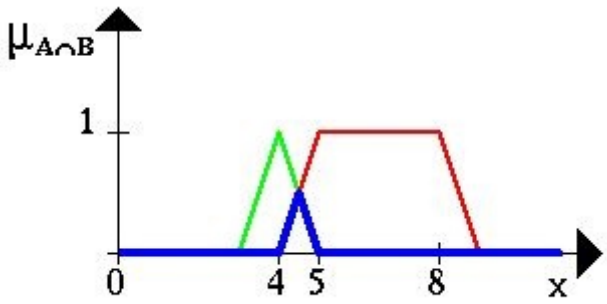

Figura 2.6: Ejemplo de Intersección

La figura 2.7 muestra la unión de ambos conjuntos difusos (el resultado es la línea de color azul):

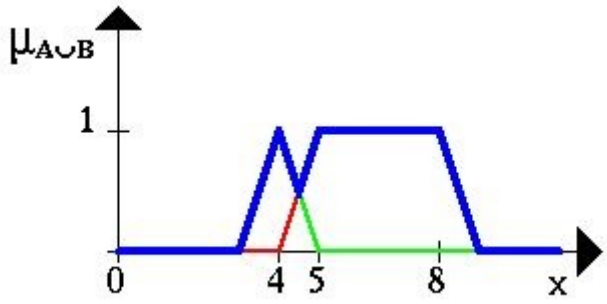

Figura 2.7: Ejemplo de Unión

La figura 2.8 muestra la negación del conjunto difuso A (el resultado es la línea de color azul):

2.2.4. Variables lingüísticas

Una variable lingüística, como su nombre sugiere, es una variable cuyos valores son palabras o sentencias en un lenguaje natural o sintético (no números). Por ejemplo, *Velocidad* es una variable lingüística cuyos valores pueden ser “alta”, “no alta”, “baja”, “no baja” y “muy baja”

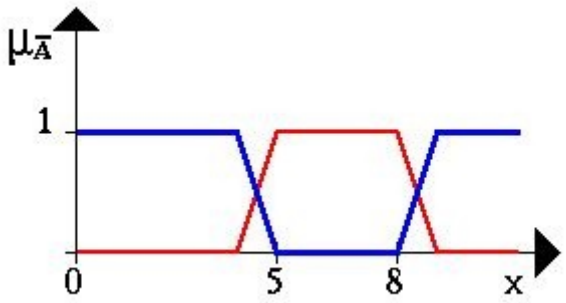

Figura 2.8: Ejemplo de Negación

(ver Figura 2.9). Si disponemos de un valor concreto de velocidad, lo podríamos representar mediante un punto en el conjunto, mientras que una etiqueta lingüística es una colección de puntos (velocidades posibles).

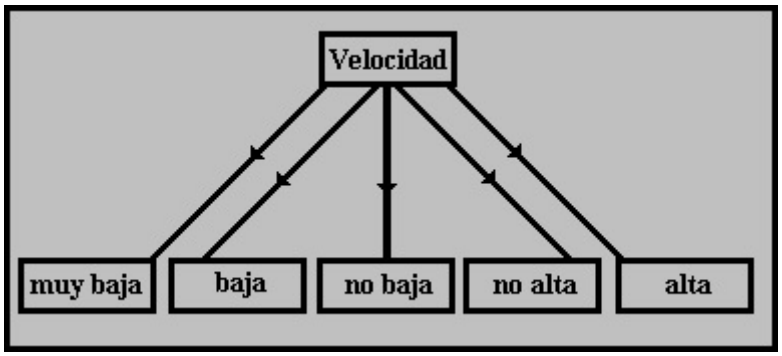

Figura 2.9: Valores lingüísticos de la variable difusa *Velocidad*.

Cada valor de una variable lingüística representa un conjunto difuso en un universo determinado como se muestra en la figura 2.10.

Estrictamente, una variable lingüística esto formada por cinco partes (v, T(v), U, G, M) donde:

- v: nombre de la variable.
- T(v): conjunto de términos o valores lingüísticos de v.
- U: universo de discurso donde se define T(v).
- G: regla sintáctica para generar los términos lingüísticos de v.
- M: regla semántica para asociar cada valor a su significado.

Por ejemplo, *Velocidad* se puede considerar una variable lingüística v. El conjunto de térmi-

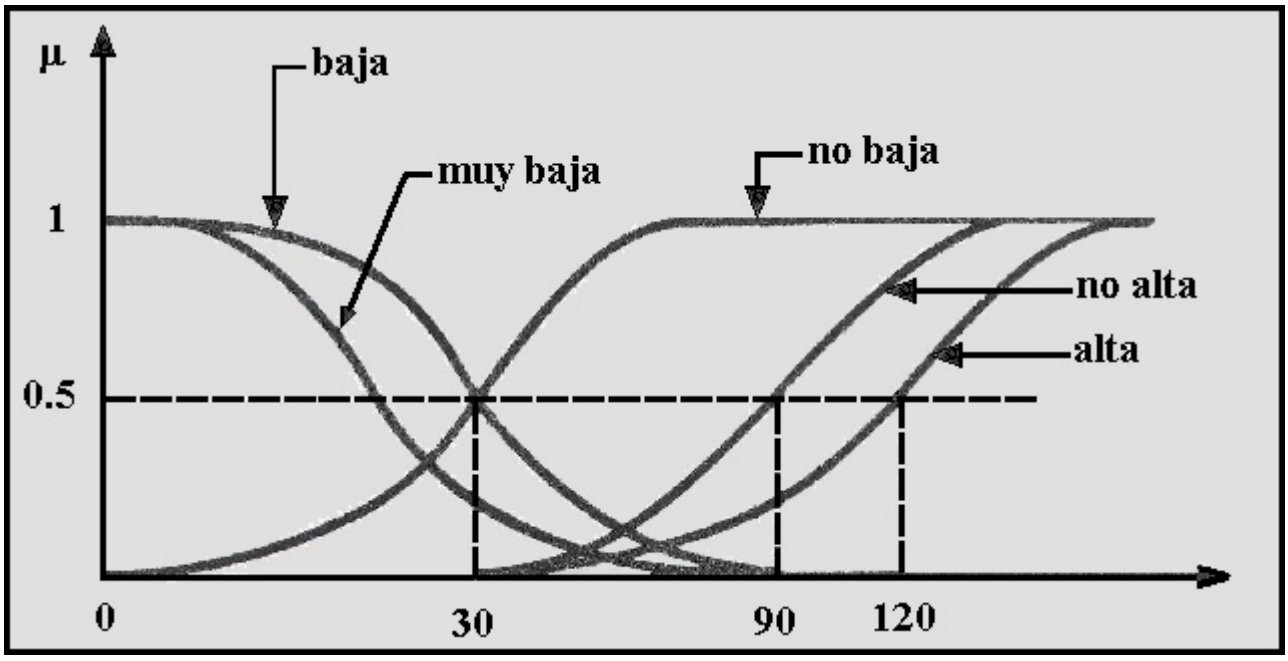

Figura 2.10: Conjuntos difusos de la variable lingüística *Velocidad* con sus correspondientes funciones de pertenencia.

nos lingüísticos (partición difusa de su universo) es:

$$T(Velocidad) = \{\text{muy alta, alta, no alta, lento, muy lento}\}$$

Cada termino en T(Velocidad) está caracterizado por un conjunto difuso en el universo de discurso U=[0,200] km/h. La regla sintáctica (G) determina el orden de las palabras de los términos lingüísticos de Velocidad. Por ejemplo *no* y *muy* son modificadores que preceden al término primario *alta* y que sirven para distinguir un término lingüístico del otro. La regla semántica (M) asocia cada término lingüístico con su significado: alta es alrededor de 180 km/h, y baja es alrededor de 30 km/h, etc.

2.3. Bases de datos relacionales ([7])

2.3.1. Conceptos de bases de datos

Una *base de datos* es un conjunto de datos almacenados en una memoria externa que están organizados mediante una estructura de datos. Cada base de datos se diseña para satisfacer los requisitos de información de una empresa u otro tipo de organización, como por ejemplo, una universidad o un hospital.

Se puede percibir como un gran almacén de datos que se define y se crea una sola vez, y que se utiliza al mismo tiempo por distintos usuarios. En una base de datos todos los datos se integran

con una mínima cantidad de duplicidad. De este modo, la base de datos no pertenece a un sólo departamento sino que se comparte por toda la organización. Además, la base de datos no sólo contiene los datos de la organización, también almacena una descripción de dichos datos. Esta descripción es lo que se denomina *metadatos*, se almacena en el *diccionario de datos* o *catálogo* y es lo que permite que exista *independencia de datos lógica-física*. Si esta no existe significa que cuando los datos se separan en distintos ficheros, es más complicado acceder a ellos, ya que el programador de aplicaciones debe sincronizar el procesamiento de los distintos ficheros implicados para garantizar que se extraen los datos correctos. Además, ya que la estructura física de los datos se encuentra especificada en los programas de aplicación, cualquier cambio en dicha estructura es difícil de realizar.

2.3.2. Sistema de gestión de bases de datos

El *sistema de gestión de la base de datos* (en adelante **SGBD**) es una aplicación que permite a los usuarios definir, crear y mantener la base de datos, además de proporcionar un acceso controlado a la misma. Se denomina *sistema de bases de datos* al conjunto formado por la base de datos, el SGBD y los programas de aplicación que dan servicio a la empresa u organización.

El modelo seguido con los sistemas de bases de datos es muy similar al modelo que se sigue en la actualidad para el desarrollo de programas con lenguajes orientados a objetos, en donde se da una implementación interna de un objeto y una especificación externa separada. Los usuarios del objeto solo ven la especificación externa y no se deben preocupar de cómo se implementa internamente el objeto. Una ventaja de este modelo, conocido como abstracción de datos, es que se puede cambiar la implementación interna de un objeto sin afectar a sus usuarios ya que la especificación externa no se ve alterada. Del mismo modo, los sistemas de bases de datos separan la definición de la estructura física de los datos de su estructura lógica, y almacenan esta definición en la base de datos. Todo esto es gracias a la existencia del SGBD, que se sitúa entre la base de datos y los programas de aplicación.

Generalmente, un SGBD proporciona los servicios que se citan a continuación:

- El SGBD permite la definición de la base de datos mediante un *lenguaje de definición de datos*. Este lenguaje permite especificar la estructura y el tipo de los datos, así como las restricciones sobre los datos.

- El SGBD permite la inserción, actualización, eliminación y consulta de datos mediante un *lenguaje de manejo de datos*. El hecho de disponer de un lenguaje para realizar consultas reduce el problema de los sistemas de ficheros, en los que el usuario tiene que trabajar con un conjunto fijo de consultas, o bien, dispone de un gran número de programas de aplicación costosos de gestionar. Hay dos tipos de lenguajes de manejo de datos: los *procedurales* y los *no procedurales*. Estos dos tipos se distinguen por el modo en que acceden a los datos. Los lenguajes procedurales manipulan la base de datos registro a registro, mientras que los no procedurales operan sobre conjuntos de registros. En los lenguajes procedurales se especifica qué operaciones se debe realizar para obtener los datos resultado, mientras que en los lenguajes no procedurales se especifica qué datos deben obtenerse sin decir cómo hacerlo. El lenguaje no procedural más utilizado es el

SQL (*Structured Query Language*) que, de hecho, es un estándar y es el lenguaje de los SGBD relacionales.

- El SGBD proporciona un acceso controlado a la base de datos mediante:
  - Un sistema de seguridad, de modo que los usuarios no autorizados no puedan acceder a la base de datos.
  - Un sistema que mantiene la integridad y la consistencia de los datos.
  - Un sistema de control de concurrencia que permite el acceso compartido a la base de datos.
  - Un sistema de control de recuperación que restablece la base de datos después de que se produzca un fallo del *hardware* o del *software*.
  - Un diccionario de datos o catálogo, accesible por el usuario, que contiene la descripción de los datos de la base de datos.

El SGBD se ocupa de la estructura física de los datos y de su almacenamiento. Con esta funcionalidad, el SGBD se convierte en una herramienta de gran utilidad. Sin embargo, desde el punto de vista del usuario, se podría discutir que los SGBD ha complicado las cosas, ya que ahora los usuarios ven más datos de los que realmente quieren o necesitan, puesto que ven la base de datos completa. Conscientes de este problema, los SGBD proporcionan un mecanismo de *vistas* que permite que cada usuario tenga su propia vista o visión de la base de datos. El lenguaje de definición de datos permite definir vistas como subconjuntos de la base de datos. Todos los SGBD no presentan la misma funcionalidad, depende de cada producto. En general, los grandes SGBD multiusuario ofrecen todas las funciones que se acaban de citar e incluso más. Los sistemas modernos son conjuntos de programas extremadamente complejos y sofisticados, con millones de líneas de código y con una documentación consistente en varios volúmenes. Lo que se pretende es proporcionar un sistema que permita gestionar cualquier tipo de requisitos y que tenga un 100 % de fiabilidad ante cualquier tipo de fallo.

Los SGBD están en continua evolución, tratando de satisfacer los requisitos de todo tipo de usuarios. Por ejemplo, muchas aplicaciones de hoy en día necesitan almacenar datos imprecisos (difusos), etc. Para satisfacer a este mercado, los SGBD deben evolucionar. Conforme vaya pasando el tiempo, irán surgiendo nuevos requisitos, por lo que los SGBD nunca permanecerán estáticos.

2.3.3. Modelo relacional

Modelos de datos

Una de las características fundamentales de los sistemas de bases de datos es que proporcionan cierto nivel de abstracción de datos, al ocultar las características sobre el almacenamiento físico que la mayoría de usuarios no necesita conocer. Los *modelos de datos* son el instrumento principal para ofrecer dicha abstracción a través de su jerarquía de niveles. Un modelo de datos es un conjunto de conceptos que sirven para describir la estructura de una base de datos, es decir, los datos, las relaciones entre los datos y las restricciones que deben cumplirse sobre los

datos. Los modelos de datos contienen también un conjunto de operaciones básicas para la realización de consultas (lecturas) y actualizaciones de datos. Además, los modelos de datos más modernos incluyen mecanismos para especificar acciones compensatorias o adicionales que se deben llevar a cabo ante las acciones habituales que se realizan sobre la base de datos.

Los modelos de datos se pueden clasificar dependiendo de los tipos de conceptos que ofrecen para describir la estructura de la base de datos, formando una jerarquía de niveles. Los modelos de datos de alto nivel, o *modelos conceptuales*, disponen de conceptos muy cercanos al modo en que la mayoría de los usuarios percibe los datos, mientras que los modelos de datos de bajo nivel, o *modelos físicos*, proporcionan conceptos que describen los detalles de cómo se almacenan los datos en el ordenador. Los conceptos de los modelos físicos están dirigidos al personal informático, no a los usuarios finales. Entre estos dos extremos se encuentran los *modelos lógicos*, cuyos conceptos pueden ser entendidos por los usuarios finales, aunque no están demasiado alejados de la forma en que los datos se organizan físicamente. Los modelos lógicos ocultan algunos detalles de cómo se almacenan los datos, pero pueden implementarse de manera directa en un SGBD.

Los modelos conceptuales utilizan conceptos como entidades, atributos y relaciones. Una *entidad* representa un objeto o concepto del mundo real como, por ejemplo, un cliente de una empresa o una de sus facturas. Un *atributo* representa alguna propiedad de interés de una entidad como, por ejemplo, el nombre o el domicilio del cliente. Una *relación* describe una interacción entre dos o más entidades, por ejemplo, la relación que hay entre un cliente y las facturas que se le han realizado.

Cada SGBD soporta un modelo lógico, siendo los más comunes el *relacional*, el de *red* y el *jerárquico*. Estos modelos representan los datos valiéndose de estructuras de registros, por lo que también se denominan *modelos orientados a registros*. Hay una familia más moderna de modelos lógicos, son los *modelos orientados a objetos*, que están más próximos a los modelos conceptuales. En el modelo relacional los datos se describen como un conjunto de tablas con referencias lógicas entre ellas, mientras que en los modelos jerárquico y de red, los datos se describen como conjuntos de registros con referencias físicas entre ellos (punteros).

Los modelos físicos describen cómo se almacenan los datos en el ordenador: el formato de los registros, la estructura de los ficheros (desordenados, ordenados, agrupados) y los métodos de acceso utilizados (índices, tablas de dispersión).

A la descripción de una base de datos mediante un modelo de datos se le denomina *esquema de la base de datos*. Este esquema se especifica durante el diseño, y no es de esperar que se modifique a menudo. Sin embargo, los datos que se almacenan en la base de datos pueden cambiar con mucha frecuencia: se insertan datos, se actualizan, se borran, etc. Los datos que la base de datos contiene en un determinado momento conforman el *estado de la base de datos*.

La distinción entre el esquema y el estado de la base de datos es muy importante. Cuando definimos una nueva base de datos, sólo especificamos su esquema al SGBD. En ese momento, el estado de la base de datos es el *estado vacío*, sin datos. Cuando se cargan datos por primera vez, la base datos pasa al *estado inicial*. De ahí en adelante, siempre que se realice una operación de actualización de la base de datos, se tendrá un nuevo estado. El SGBD se encarga, en parte, de garantizar que todos los estados de la base de datos sean estados válidos que satisfagan la estructura y las restricciones especificadas en el esquema. Por lo tanto, es muy importante que

el esquema que se especifique al SGBD sea correcto y se debe tener gran cuidado al diseñarlo. El SGBD almacena el esquema en su *catálogo* o *diccionario de datos*, de modo que se pueda consultar siempre que sea necesario.

En 1970, el modo en que se veían las bases de datos cambió por completo cuando E. F. Codd introdujo el **modelo relacional**. En aquellos momentos, el enfoque existente para la estructura de las bases de datos utilizaba punteros físicos (direcciones de disco) para relacionar registros de distintos ficheros. Si, por ejemplo, se quería relacionar un registro A con un registro B, se debía añadir al registro A un campo conteniendo la dirección en disco (un puntero físico) del registro B. Codd demostró que estas bases de datos limitaban en gran medida los tipos de operaciones que los usuarios podían realizar sobre los datos. Además, estas bases de datos eran muy vulnerables a cambios en el entorno físico. Si se añadían los controladores de un nuevo disco al sistema y los datos se movían de una localización física a otra, se requería una conversión de los ficheros de datos. Estos sistemas se basaban en el modelo de red y el modelo jerárquico, los dos modelos lógicos que constituyeron la primera generación de los SGBD.

El modelo relacional representa la segunda generación de los SGBD. En él, todos los datos están estructurados a nivel lógico como tablas formadas por filas y columnas, aunque a nivel físico pueden tener una estructura completamente distinta. Un punto fuerte del modelo relacional es la sencillez de su estructura lógica.

Dada la popularidad del modelo relacional, muchos sistemas de la primera generación se han modificado para proporcionar una interfaz de usuario relacional, con independencia del modelo lógico que soportan (de red o jerárquico).

En los últimos años, se han propuesto algunas extensiones al modelo relacional para capturar mejor el significado de los datos, para disponer de los conceptos de la orientación a objetos y para disponer de capacidad deductiva.

El modelo relacional, como todo modelo de datos, tiene que ver con tres aspectos de los datos, que son los que se presentan en los siguientes apartados de este capítulo: qué características tiene la estructura de datos, cómo mantener la integridad de los datos y cómo realizar el manejo de los mismos.

Estructura de datos relacional

La estructura de datos del modelo relacional se llama *relación*. En este apartado se presenta esta estructura de datos, sus propiedades, los tipos de relaciones y qué es una clave de una relación. Para facilitar la comprensión de las definiciones formales de todos estos conceptos, se dan antes unas definiciones informales que permiten asimilar dichos conceptos con otros que resulten familiares.

Relaciones

Definiciones informales

El modelo relacional se basa en el concepto matemático de *relación*, que gráficamente se representa mediante una tabla. Codd, que era un experto matemático, utilizó una terminología perteneciente a las matemáticas, en concreto de la teoría de conjuntos y de la lógica de predicados.

Una relación es una tabla con columnas y filas. Un SGBD sólo necesita que el usuario pueda percibir la base de datos como un conjunto de tablas. Esta percepción sólo se aplica a la estructura lógica de la base de datos, no se aplica a la estructura física de la base de datos, que se puede implementar con distintas estructuras de almacenamiento.

Un atributo es el nombre de una columna de una relación. En el modelo relacional, las relaciones se utilizan para almacenar información sobre los objetos que se representan en la base de datos. Una relación se representa gráficamente como una tabla bidimensional en la que las filas corresponden a registros individuales y las columnas corresponden a los campos o atributos de esos registros. Los atributos pueden aparecer en la relación en cualquier orden.

Un dominio es el conjunto de valores legales de uno o varios atributos. Los dominios constituyen una poderosa característica del modelo relacional. Cada atributo de una base de datos relacional se define sobre un dominio, pudiendo haber varios atributos definidos sobre el mismo dominio.

El concepto de dominio es importante porque permite que el usuario defina, en un lugar común, el significado y la fuente de los valores que los atributos pueden tomar. Esto hace que haya más información disponible para el sistema cuando éste va a ejecutar una operación relacional, de modo que las operaciones que son semánticamente incorrectas se pueden evitar. Por ejemplo, no tiene sentido comparar el nombre de una calle con un número de teléfono, aunque los dos atributos sean cadenas de caracteres. Sin embargo, el importe mensual del alquiler de un inmueble no estará definido sobre el mismo dominio que el número de meses que dura el alquiler, pero sí tiene sentido multiplicar los valores de ambos dominios para averiguar el importe total al que asciende el alquiler. Los SGBD relacionales no ofrecen un soporte completo de los dominios ya que su implementación es extremadamente compleja.

Una tupla es una fila de una relación. Los elementos de una relación son las tuplas o filas de la tabla. Las tuplas de una relación no siguen ningún orden.

El grado de una relación es el número de atributos que contiene. El grado de una relación no cambia con frecuencia.

La cardinalidad de una relación es el número de tuplas que contiene. Ya que en las relaciones se van insertando y borrando tuplas a menudo, la cardinalidad de las mismas varía constantemente.

Una base de datos relacional es un conjunto de relaciones normalizadas. Una relación está normalizada si en la intersección de cada fila con cada columna hay un solo valor.

Definiciones formales

Una *relación* R definida sobre un conjunto de *dominios* $D_1, D_2, ..., D_n$ consta de:

- *Cabecera*: conjunto fijo de pares *atributo:dominio*

$$(A_1 : D_1), (A_2 : D_2), ..., (A_n : D_n)$$

  donde cada atributo A_j corresponde a un único dominio D_j y todos los A_j son distintos, es decir, no hay dos atributos que se llamen igual. El grado de la relación R es n.

- *Cuerpo*: conjunto variable de *tuplas*. Cada tupla es un conjunto de pares *atributo:valor* :

$$(A_1 : v_{i1}), (A_2 : v_{i2}), ..., (A_n : v_{in})$$

  con $i = 1, 2, ..., m$, donde m es la cardinalidad de la relación R. En cada par $(A_j : v_{ij})$ se tiene que $v_{ij} \in$D$_j$.

Las relaciones se suelen representar gráficamente mediante tablas. Los nombres de las columnas corresponden a los nombres de los atributos, y las filas son cada una de las tuplas de la relación. Los valores que aparecen en cada una de las columnas pertenecen al conjunto de valores del dominio sobre el que está definido el atributo correspondiente.

Propiedades de las relaciones

Las relaciones tienen las siguientes características:

- Cada relación tiene un nombre, y éste es distinto del nombre de todas las demás.
- Los dominios sobre los que se definen los atributos son escalares, por lo que los valores de los atributos son atómicos. De este modo, en cada tupla, cada atributo toma un solo valor. Se dice que las relaciones están *normalizadas*.
- No hay dos atributos que se llamen igual.
- El orden de los atributos no importa: los atributos no están ordenados.
- Cada tupla es distinta de las demás: no hay tuplas duplicadas.
- El orden de las tuplas no importa: las tuplas no están ordenadas.

Tipos de relaciones

En un SGBD relacional hay dos tipos de relaciones:

- *Relaciones base*. Son relaciones reales que tienen nombre, y forman parte directa de la base de datos almacenada. Se dice que las relaciones base son relaciones autónomas.
- *Vistas*. También denominadas relaciones virtuales, son relaciones con nombre y derivadas (no autónomas). Que son derivadas significa que se obtienen a partir de otras relaciones; se representan mediante su definición en términos de esas otras relaciones. Las vistas no poseen datos almacenados propios, los datos que contienen corresponden a datos almacenados en relaciones base.

Claves

Ya que en una relación no hay tuplas repetidas, éstas se pueden distinguir unas de otras, es decir, se pueden identificar de modo único. La forma de identificarlas es mediante los valores de sus atributos. Se denomina *superclave* a un atributo o conjunto de atributos que identifican de modo único las tuplas de una relación. Se denomina *clave candidata* a una superclave en la que ninguno de sus subconjuntos es una superclave de la relación. El atributo o conjunto de atributos K de la relación R es una clave candidata para R si, y sólo si, satisface las siguientes propiedades:

- *Unicidad*: nunca hay dos tuplas en la relación R con el mismo valor de K.
- *Irreducibilidad (minimalidad)*: ningún subconjunto de K tiene la propiedad de unicidad, es decir, no se pueden eliminar componentes de K sin destruir la unicidad.

Cuando una clave candidata está formada por más de un atributo, se dice que es una *clave compuesta*. Una relación puede tener varias claves candidatas.

Para identificar las claves candidatas de una relación no hay que fijarse en un estado u ocurrencia de la base de datos. El hecho de que en un momento dado no haya duplicados para un atributo o conjunto de atributos, no garantiza que los duplicados no sean posibles. Sin embargo, la presencia de duplicados en un estado de la base de datos sí es útil para demostrar que cierta combinación de atributos no es una clave candidata. El único modo de identificar las claves candidatas es conociendo el significado real de los atributos, ya que esto permite saber si es posible que aparezcan duplicados. Sólo usando esta información semántica se puede saber con certeza si un conjunto de atributos forman una clave candidata.

Se denomina *clave primaria* de una relación a aquella clave candidata que se escoge para identificar sus tuplas de modo único. Ya que una relación no tiene tuplas duplicadas, siempre hay una clave candidata y, por lo tanto, la relación siempre tiene clave primaria. En el peor caso, la clave primaria estará formada por todos los atributos de la relación, pero normalmente habrá un pequeño subconjunto de los atributos que haga esta función.

Las claves candidatas que no son escogidas como clave primaria son denominadas *claves alternativas*.

2.3.4. Reglas de integridad

Una vez definida la estructura de datos del modelo relacional, pasamos a estudiar las reglas de integridad que los datos almacenados en dicha estructura deben cumplir para garantizar que son correctos.

Al definir cada atributo sobre un dominio se impone una restricción sobre el conjunto de valores permitidos para cada atributo. A este tipo de restricciones se les denomina *restricciones de dominios*. Hay además dos reglas de integridad muy importantes que son restricciones que se deben cumplir en todas las bases de datos relacionales y en todos sus estados (las reglas se deben cumplir todo el tiempo). Estas reglas son la *regla de integridad de entidades* y la *regla de integridad referencial*. Antes de definirlas, es preciso conocer el concepto de *nulo*.

Nulos

Cuando en una tupla un atributo es desconocido, se dice que es *nulo*. Un nulo no representa el valor cero ni la cadena vacía ya que éstos son valores que tienen significado. El nulo implica ausencia de información, bien porque al insertar la tupla se desconocía el valor del atributo, o bien porque para dicha tupla el atributo no tiene sentido.

Ya que los nulos no son valores, deben tratarse de modo diferente, lo que causa problemas de implementación. De hecho, no todos los SGBD relacionales soportan los nulos.

Regla de integridad de entidades

La primera regla de integridad se aplica a las claves primarias de las relaciones base: *ninguno de los atributos que componen la clave primaria puede ser nulo*.

Por definición, una clave primaria es una clave irreducible que se utiliza para identificar de modo único las tuplas. Que es irreducible significa que ningún subconjunto de la clave primaria sirve para identificar las tuplas de modo único. Si se permitiera que parte de la clave primaria fuera nula, se estaría diciendo que no todos sus atributos son necesarios para distinguir las tuplas, con lo que se estaría contradiciendo la irreducibilidad.

Nótese que esta regla sólo se aplica a las relaciones base y a las claves primarias, no a las claves alternativas.

Regla de integridad referencial

La segunda regla de integridad se aplica a las claves ajenas: *si en una relación hay alguna clave ajena, sus valores deben coincidir con valores de la clave primaria a la que hace referencia, o bien, deben ser completamente nulos.*

La regla de integridad referencial se enmarca en términos de estados de la base de datos: indica lo que es un estado ilegal, pero no dice cómo puede evitarse. Por lo tanto, una vez establecida la regla, hay que plantearse qué hacer si estando en un estado legal, llega una petición para realizar una operación que conduce a un estado ilegal. Existen dos opciones: rechazar o aceptar la operación y realizar operaciones adicionales compensatorias que conduzcan a un estado legal.

Para hacer respetar la integridad referencial se debe contestar, para cada clave ajena, a las tres preguntas que se plantean a continuación y que determinarán su comportamiento:

- *Regla de los nulos*: «¿Tiene sentido que la clave ajena acepte nulos?»
- *Regla de borrado*: «¿Qué ocurre si se intenta borrar la tupla referenciada por la clave ajena?»
  - *Restringir*: no se permite borrar la tupla referenciada.
  - *Propagar*: se borra la tupla referenciada y se propaga el borrado a las tuplas que la referencian mediante la clave ajena.
  - *Anular*: se borra la tupla referenciada y las tuplas que la referenciaban ponen a nulo la clave ajena (sólo si acepta nulos).
  - *Valor por defecto*: se borra la tupla referenciada y las tuplas que la referenciaban ponen en la clave ajena el valor por defecto establecido para la misma.
- Regla de modificación: «¿Qué ocurre si se intenta modificar el valor de la clave primaria de la tupla referenciada por la clave ajena?»
  - *Restringir*: no se permite modificar el valor de la clave primaria de la tupla referenciada.
  - *Propagar*: se modifica el valor de la clave primaria de la tupla referenciada y se propaga la modificación a las tuplas que la referencian, mediante la clave ajena.
  - *Anular*: se modifica la tupla referenciada y las tuplas que la referenciaban ponen a nulo la clave ajena (sólo si acepta nulos).
  - *Valor por defecto*: se modifica la tupla referenciada y las tuplas que la referenciaban ponen en la clave ajena el valor por defecto establecido para la misma.

Reglas de negocio

Además de las dos reglas de integridad anteriores, es posible que sea necesario imponer ciertas restricciones específicas sobre los datos que forman parte de la estrategia de funcionamiento de la empresa. A estas reglas se las denomina *reglas de negocio.*

Por ejemplo, si en cada oficina de una determinada empresa sólo puede haber hasta veinte empleados, el SGBD debe dar la posibilidad al usuario de definir una regla al respecto y debe hacerla respetar. En este caso, no debería permitir dar de alta a un empleado en una oficina que ya tiene los veinte permitidos. No todos los SGBD relacionales permiten definir este tipo de restricciones y hacerlas respetar.

2.3.5. Lenguajes relacionales

Son varios los lenguajes utilizados por los SGBD relacionales para manejar las relaciones. Algunos de ellos son *procedurales*, lo que quiere decir que el usuario indica al sistema exactamente cómo debe manipular los datos. Otros son *no procedurales*, que significa que el usuario indica qué datos necesita, en lugar de establecer cómo deben obtenerse. Se puede decir que el álgebra relacional es un lenguaje procedural de alto nivel, mientras que el cálculo relacional es un lenguaje no procedural. Sin embargo, ambos lenguajes son equivalentes: para cada expresión del álgebra, se puede encontrar una expresión equivalente en el cálculo, y viceversa.

El álgebra relacional (o el cálculo relacional) se utiliza para medir la potencia de los lenguajes relacionales. Si un lenguaje permite obtener cualquier relación que se pueda derivar mediante el álgebra relacional, se dice que es *relacionalmente completo*. La mayoría de los lenguajes relacionales son relacionalmente completos, pero tienen más potencia que el álgebra o el cálculo porque se les han añadido operadores especiales.

Tanto el álgebra como el cálculo son lenguajes formales no muy amigables; sin embargo, es conveniente estudiarlos porque sirven para ilustrar las operaciones básicas que todo lenguaje de manejo de datos debe ofrecer. Además, han sido la base para otros lenguajes relacionales de manejo de datos de más alto nivel.

Álgebra relacional

El álgebra relacional es un lenguaje formal con una serie de operadores que trabajan sobre una o varias relaciones para obtener otra relación resultado, sin que cambien las relaciones originales. Tanto los operandos como los resultados son relaciones, por lo que la salida de una operación puede ser la entrada de otra operación. Esto permite anidar expresiones del álgebra, del mismo modo que se pueden anidar las expresiones aritméticas. A esta propiedad se le denomina *clausura*: las relaciones son cerradas bajo el álgebra, del mismo modo que los números son cerrados bajo las operaciones aritméticas.

Hay cinco operadores que son fundamentales: *restricción, proyección, producto cartesiano, unión y diferencia*. Los operadores fundamentales permiten realizar la mayoría de las operaciones de obtención de datos. Los operadores no fundamentales son la *concatenación (JOIN)*, la *intersección* y la *división*, que se pueden expresar a partir de los cinco operadores fundamentales.

La restricción y la proyección son operaciones *unarias* porque operan sobre una sola relación. El resto de las operaciones son *binarias* porque trabajan sobre pares de relaciones.

A continuación explicamos brevemente los operadores del álgebra relacional:

- **Restricción: R WHERE condición.**

  La restricción, también denominada selección, opera sobre una sola relación R y da como resultado otra relación cuyas tuplas son las tuplas de R que satisfacen la condición especificada. Esta condición es una comparación en la que aparece al menos un atributo de R, o una combinación booleana de varias de estas comparaciones.

- **Proyección:** $R[a_i, ..., a_k]$**.**

  La proyección opera sobre una sola relación R y da como resultado otra relación que contiene un subconjunto vertical de R, extrayendo los valores de los atributos especificados y eliminando duplicados.

- **Producto cartesiano: R TIMES S.**

  El producto cartesiano obtiene una relación cuyas tuplas están formadas por la concatenación de todas las tuplas de R con todas las tuplas de S.

- **Unión: R UNION S.**

  La unión de dos relaciones R y S, con P y Q tuplas respectivamente, es otra relación que tiene como mucho P + Q tuplas siendo éstas las tuplas que se encuentran en R o en S o en ambas relaciones a la vez. Para poder realizar esta operación, R y S deben ser compatibles para la unión.

- **Diferencia: R EXCEPT S.**

  La diferencia obtiene una relación que tiene las tuplas que se encuentran en R y no se encuentran en S. Para realizar esta operación, R y S deben ser compatibles para la unión.

- **Concatenación (Join): R JOIN S.**

  La concatenación de dos relaciones R y S obtiene como resultado una relación cuyas tuplas son todas las tuplas de R concatenadas con todas las tuplas de S que en los atributos comunes (áquellos que se llaman igual) tienen los mismos valores. Estos atributos comunes aparecen una sola vez en el resultado.

- **Concatenación externa (Outer-join): R LEFT OUTER JOIN S.**

  La concatenación externa por la izquierda es una concatenación en la que las tuplas de R (que se encuentra a la izquierda en la expresión) que no tienen valores en común con ninguna tupla de S, también aparecen en el resultado.

- **Intersección: R INTERSECT S.**

  La intersección obtiene como resultado una relación que contiene las tuplas de R que también se encuentran en S. Para realizar esta operación, R y S deben ser compatibles para la unión.

- **División: R DIVIDE BY S.**

  Suponiendo que la cabecera de R es el conjunto de atributos A y que la cabecera de S es el conjunto de atributos B, tales que B es un subconjunto de A, y si C = A - B (los atributos de R que no están en S), la división obtiene una relación cuya cabecera es el conjunto de atributos C y que contiene las tuplas de R que están acompañadas de todas las tuplas de S.

- **Agrupación: SUMMARIZE R GROUP BY** $(a_i, ..., a_k)$ **ADD cálculo AS atributo.**

  Esta operación agrupa las tuplas de R que tienen los mismos valores en los atributos especificados y realiza un cálculo sobre los grupos obtenidos. La relación resultado tiene como cabecera los atributos por los que se ha agrupado y el cálculo realizado, al que se da el nombre especificado en atributo.

Cálculo relacional

El álgebra relacional y el cálculo relacional son formalismos diferentes que representan distintos estilos de expresión del manejo de datos en el ámbito del modelo relacional. El álgebra relacional proporciona una serie de operaciones que se pueden usar para indicar al sistema cómo construir la relación deseada a partir de las relaciones de la base de datos. El cálculo relacional proporciona una notación para formular la definición de la relación deseada en términos de las relaciones de la base de datos.

El cálculo relacional toma su nombre del cálculo de predicados, que es una rama de la lógica. Hay dos tipos de cálculo relacional, el orientado a tuplas, propuesto por Codd, y el orientado a dominios, propuesto por otros autores. El estudio del cálculo relacional se hará aquí mediante definiciones informales.

En el cálculo de predicados (lógica de primer orden), un predicado es una función con argumentos que se puede evaluar a verdadero o falso. Cuando los argumentos se sustituyen por valores, la función lleva a una expresión denominada proposición, que puede ser verdadera o falsa. Por ejemplo, las frases «Paloma Poy es una vendedora de la empresa» y «Paloma Poy es jefa de Natalia Guillén» son proposiciones, ya que se puede determinar si son verdaderas o falsas. En el primer caso, la función «es una vendedora de la empresa» tiene un argumento (Paloma Poy) y en el segundo caso, la función «es jefa de» tiene dos argumentos (Paloma Poy y Natalia Guillén).

Si un predicado tiene una variable, como en «x es una vendedora de la empresa», esta variable debe tener un rango asociado. Cuando la variable se sustituye por alguno de los valores de su rango, la proposición puede ser cierta; para otros valores puede ser falsa. Por ejemplo, si el rango de x es el conjunto de todas las personas y reemplazamos x por Paloma Poy, la proposición «Paloma Poy es una vendedora de la empresa» es cierta. Pero si reemplazamos x por el nombre de una persona que no es vendedora de la empresa, la proposición es falsa.

Si F es un predicado, la siguiente expresión devuelve el conjunto de todos los valores de x para los que F es cierto: x WHERE F(x). Los predicados se pueden conectar mediante AND, OR y NOT para formar predicados compuestos.

2.4. Bases de datos relacionales difusas

2.4.1. Introducción

Las Bases de datos relacionales difusas intentan aplicar una forma más humana de pensar en la programación de computadoras usando lógica difusa. Por tanto diremos que es una metodología que proporciona una manera simple y elegante de obtener una conclusión a partir de información de entrada vaga, ambigua, imprecisa, con ruido o incompleta. En general la lógica difusa imita cómo un persona toma decisiones basada en información con las características mencionadas. Una de las ventajas de la lógica difusa es la posibilidad de implementar sistemas basados en ella tanto en hardware como en software o en combinación de ambos.

Esta lógica es una lógica multievaluada y sus características principales, presentadas por Zadeh en la referencia antes mencionada son:

- En la lógica difusa, el razonamiento exacto es considerado como un caso particular del razonamiento aproximado.
- Cualquier sistema lógico puede ser trasladado a términos de lógica difusa.
- En lógica difusa, el conocimiento es interpretado como un conjunto de restricciones flexibles, es decir, difusas, sobre un conjunto de variables.
- La inferencia es considerada como un proceso de propagación de dichas restricciones. En lógica difusa, todo problema es un problema de grados.

2.4.2. Bases de datos difusas

Las bases de datos difusas nacen de unir la teoría de bases de datos, principalmente del modelo relacional con la teoría de conjuntos difusos, para permitir, básicamente dos objetivos:

- El almacenamiento de información difusa (además de información no difusa).
- El tratamiento y consulta de esta información de forma difusa o flexible.

Las bases de datos tradicionales son muy limitadas: No permiten ni almacenar ni tratar con datos imprecisos. Sin embargo las personas manejamos datos imprecisos muy a menudo y muy eficientemente.

¿Dónde podemos ver reflejada dicha imprecisión? o, lo que sería lo mismo, ¿cuáles son las diferencias entre una base de datos relacional difusa y una clásica?

A la definición del formato interno de la BDRD, y su esquema global de implementación, se le denomina **FIRST** (Fuzzy Interface for Relational SysTems).

La necesidad inicial de FIRST es debida a que se pretende construir un Sistema de BDRD sobre un SGBD ya existente. En particular, se ha usado el SGBD Oracle, por su potencia, flexibilidad, popularidad y robustez.

Los fundamentos de **FIRST**, que se explicarán más adelante son: *Atributos Difusos* y *FMB* (Fuzzy Metaknowledge Base, Base de Metaconocimiento Difuso). La *FMB* está incluida dentro del catálogo o diccionario del sistema y almacena información relacionada con la extensión difusa del la base de datos.

Oracle, una compañía de software que desarrolla bases de datos y sistemas de gestión de bases de datos, sólo permite el uso de SQL y no de *FSQL* (Fuzzy Structured Query Language. Por eso, ha sido necesario construir un programa que permita el uso de FSQL en la BDRD definida por **FIRST**.

A continuación desarrollamos los conceptos para entender las diferencias entre las bases de datos relacionales y la bases de datos relacionales difusas.

Tipos de atributos difusos

Se distinguen dos clases de atributos difusos: Atributos cuyos valores son conjuntos difusos y atributos cuyos valores son grados difusos.

- **Atributos con Conjuntos Difusos**: Estos atributos pueden clasificarse en cuatro tipos según FSQL, de acuerdo al tipo de referencial (dominio subyacente o eje X donde se definen los conjuntos difusos). En todos ellos se incluyen los valores Unknown, Undefined, y Null:
  - **Tipo 1**: Son atributos precisos (sin imprecisión o *crisp*). Sin embargo, se permite definir etiquetas lingüísticas en su dominio y podremos usarlas en consultas difusas. Se almacenan igual que un atributo normal, pero puede ser transformado o manipulado usando condiciones difusas. Este tipo es útil para extender bases de datos tradicionales para permitir consultas difusas en sus dominios clásicos. Por ejemplo, preguntas del tipo: "Dame los empleados que ganan mucho más que el salario mínimo".
  - **Tipo 2**: Son atributos "imprecisos sobre un referencial ordenado". Admiten valores *crisp* y difusos, en forma de distribuciones de posibilidad o conjuntos difusos, sobre un dominio subyacente ordenado. Es una extensión del Tipo 1 que sí permite el almacenamiento de información imprecisa, tal como el valor: "aproximadamente 2 metros". Por simplicidad, estos conjuntos difusos suelen ser una función trapezoidal (Figura 2.11, donde el eje Y es el grado difuso).
  - **Tipo 3**: Son atributos difusos con datos "discretos sobre dominio subyacente no ordenado con analogía". Aquí se pueden definir etiquetas (Rubio, Moreno...) que son escalares con una relación de similitud o proximidad definida sobre esas etiquetas, para indicar en qué medida se parecen cada par de etiquetas. También se admiten distribuciones de posibilidad (o conjuntos difusos) sobre este dominio, como por ejemplo, el valor {1/Rubio, 0.4/Moreno}, que expresa que cierta persona es más Rubia que Morena. Observe que el dominio subyacente de ese valor difuso es el conjunto de etiquetas y este conjunto carece de orden.

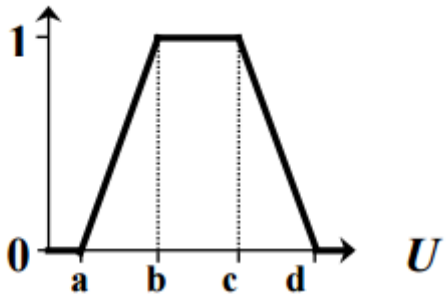

Figura 2.11: Función trapezoidal

- **Tipo 4**: Estos originales atributos son similares al Tipo 3, pero sin precisar la relación de similitud. En este caso suponemos que no necesitamos tal relación o que no existe.

- **Atributos con Grados Difusos**: El dominio de estos grados puede estar en el intervalo [0,1], aunque otros valores podrían también permitirse, tal como una distribución de posibilidad (usualmente sobre este intervalo unidad). Para simplificar, nosotros sólo usaremos grados en dicho intervalo, ya que otras opciones no ofrecen grandes ventajas.

  El significado de estos grados es variado y depende de su uso. El procesamiento de los datos será diferente según este significado, por lo que el sistema debe almacenar y considerar este significado. Los significados más importantes de estos grados según distintos autores son : grado de cumplimiento, grado de incertidumbre, grado de posibilidad y grado de importancia. Por supuesto, podremos definir y usar otros significados.

  Estos grados pueden ser asociados a diferentes conceptos:

  - **Grado en cada valor de un atributo (Tipo 5)**: Algunos atributos pueden tener un grado difuso asociado a ellos. Esto implica que cada valor de este atributo (en cada tupla o instancia) tiene un grado asociado que mide el nivel de "imprecisión"de ese valor/atributo. Para interpretarlo, necesitamos saber el significado de ese grado y el significado del atributo asociado.
  - **Grado en un conjunto de valores de diferentes atributos (Tipo 6)**: Aquí, el grado está asociado a algunos atributos. Este es un caso poco usual, pero puede ser muy útil en casos específicos. Une la imprecisión de varios atributos en un único grado.
  - **Grado en la instancia completa de la relación (Tipo 7)**: Este grado está asociado a la tupla completa (u objeto) de la relación y no exclusivamente al valor de un atributo específico de la tupla. Normalmente, puede representar algún "grado de pertenencia" del objeto a la relación o tabla de la base de datos. También suelen usarse para medir el "grado de importancia" de cada objeto.
  - **Los grados no asociados (Tipo 8)** son útiles cuando la información imprecisa que queremos representar, se puede representar usando únicamente el grado, sin asociar este grado a otro valor o valores. Por ejemplo, la peligrosidad de un medicamento puede ser expresado usando un grado difuso de este tipo.

FMB (Fuzzy Metaknowledge Base)

Los datos de metaconocimiento difuso son el conocimiento necesario sobre la base de datos difusa y especialmente sobre los atributos difusos. Esta información se almacena en formato relacional en la llamada FMB (Fuzzy Metaknowledge Base). Vamos primero a definir la información que se almacena en la FMB y posteriormente explicamos su estructura y sus tablas. Así pues, la FMB almacena esta información:

1. Atributos con capacidades difusas.

2. El metaconocimiento requerido para cada atributo depende de su tipo:

- Tipos 1 y 2: Estos atributos almacenan en la FMB la definición (conjunto difuso) de cada etiqueta lingüística, el "margen"para valores aproximados, y la mínima distancia para considerar dos valores como muy separados (valor "much"). Este último valor se usa en comparadores como "mucho mayor que", comparador MGT de FSQL.
- Tipos 3 y 4: Valor n (explicado antes), nombre de las etiquetas y, para el Tipo 3, la relación de similitud entre cada dos etiquetas.
- Tipos 5 y 6: Significado del grado, y el atributo (Tipo 5) o atributos (Tipo 6) a los que el grado está asociado.
- Tipos 7 y 8: Significado del grado.

3. Otros objetos: Éstos incluyen cualificadores difusos (asociados a un atributo y usados para establecer umbrales en consultas), y cuantificadores difusos (asociados a una tabla o a un atributo). Los cuantificadores son usados en consultas, tales como "Dame los empleados que pertenecen a la *mayoría* de los proyectos", y también en restricciones difusas, como por ejemplo "Un empleado debe trabajar en *muchos* proyectos".

Si dos atributos difusos de los Tipos 1, 2, 3 ó 4 necesitan las mismas definiciones, podemos registrar tales atributos como compatibles, para simplificar la FMB.

A continuación tratamos de dar una idea de la utilidad de cada tabla:

- $FUZZY\_COL\_LIST$: Describe los atributos difusos de cada tabla. El valor F_TYPE se usa para el tipo de atributo difuso, de 1 a 8. LEN es el valor n. $CODE\_SIG$ indica el significado del grado si $F\_TYPE \in [5, 8]$.
- $FUZZY\_DEGREE\_SIG$: Guarda los significados de los grados de la base de datos.
- $FUZZY\_OBJECT\_LIST$: Esta tabla contiene las declaraciones de los objetos difusos relacionados con los atributos difusos, tales como, etiquetas lingüísticas, cualificadores y cuantificadores difusos. Los cuantificadores pueden ser absolutos o relativos, y pueden tener uno o dos argumentos.
- $FUZZY\_LABEL\_DEF$: Define las etiquetas usando funciones trapezoidales (Figura 2.11).

- $FUZZY\_APPROX\_MUCH$: Valores para el "margen" y el "much"para los Tipos 1 y 2.
- $FUZZY\_NEARNESS\_DEF$: Relaciones de similitud para los Tipo 3.
- $FUZZY\_COMPATIBLE\_COL$: Atributos difusos compatibles, con iguales etiquetas.
- $FUZZY\_QUALIFIERS\_DEF$: Definición de cualificadores difusos.
- $FUZZY\_DEGREE\_COLS$: Esta tabla establece los atributos (o columnas) asociados a grados difusos (sólo para los Tipos 5 y 6). Observa que un grado Tipo 5 tiene sólo un atributo asociado, mientras que un Tipo 6 puede tener múltiples. Además, un atributo puede tener muchos grados difusos asociados a él, pero todos de los Tipos 5 ó 6. Por supuesto, los Tipos 7 y 8 no usan esta tabla.
- $FUZZY\_ER\_LIST$: Esta tabla almacena las entidades difusas y las relaciones difusas. $DEGREE\_TYPE$ vale "M"para entidades difusas, "C"para entidades difusas con grados calculados automáticamente, "E.e"I"para entidades débiles difusas (dependencia de existencia o de identificación) y, finalmente, "R"para relaciones difusas representadas por una tabla.
- $FUZZY\_TABLE\_QUANTIFIERS$: Definición de cuantificadores asociados a una relación o tabla (no un atributo). Estos cuantificadores son usados en restricciones difusas y pueden ser absolutos o relativos.

Entidades difusas

Se define **tipo de entidad difusa** como una entidad a la que se le añade un atributo que expresa un grado.

- **Grado de pertenencia (G0)**: El grado de pertenencia mide la pertenencia a un conjunto difuso.
- **Grado de cumplimiento (G1) de una condición o de una propiedad**. Puede verse también como grado de pertenencia al conjunto difuso de los elementos que cumplen tal condición o propiedad. Puede almacenarse o simplemente calcularse a partir de una consulta difusa.
- **Grado de incertidumbre (G2)**: El grado de Incertidumbre mide en qué medida es cierto el dato.
- **Grado de posibilidad (G3)**: El grado de Posibilidad mide lo posible que es el dato.
- **Grado de importancia (G4)**. En una relación, distintos objetos pueden tener diferente nivel de importancia y esa diferencia puede ser útil considerar para algunos propósitos.

Cada instancia de una entidad tiene un grado para medir la relación de esa instancia con su tipo de entidad.

Interrelaciones difusas

Una interrelación se considera difusa si tiene un atributo difuso que relacione las entidades asociadas.

Participación difusa

Participación: Especifica si la existencia de una entidad depende de que esté relacionada con la otra entidad. Puede ser total o parcial.

Tipo de Correspondencia: Restringe el número de elementos de la interrelación en los que puede participar una entidad.

Los más comunes son:

- "uno a uno"(1:1)
- "uno a muchos"(1:N)
- "muchos a muchos"(N:M)

Completitud difusa

Restricciones para una especialización o generalización:

- De completitud total: Toda entidad de la superclase debe pertenecer a alguna subclase de la especialización.
- De completitud parcial: Existen instancias de la superclase que no pertenecen a ninguna subclase.

Agregación difusa

Una agregación permite representar tipos de entidades compuestas que se obtienen por la unión de otras más simples.

Existen dos tipos de agregaciones:

- De tipos de entidades. Ejemplo: un coche se compone de chasis, motor e interior. Consideremos que el atributo motor tiene una importancia de 0.8 y el interior de 0.5.
- De atributos. Ejemplo: el atributo Identificador tiene una importancia de 1, seguido por el año (0.8), el dueño (0.7) y el color(0.6).

2.4.3. Principales modelos de bases de datos difusas

Los aspectos más importantes de la información que tratamos son: incertidumbre e imprecisión.

Un **SBDRD** (Sistema de Bases de Datos Relaionales Difuso) debe satisfacer las siguientes condiciones:

- Proporcionar mecanismos adecuados para poder representar la información difusa.
- Representar un marco adecuado para almacenar el significado de la información difusa que albergue.
- Facilitar un número mínimo de operadores.
- Satisfacer los requisitos de un modelo relacional.

Aunque aún no hemos logrado satisfacer completamente las características anteriores. Los diferentes niveles "difusos"que se pueden cubrir actualmente mediante un SBDRD son:

- Obtención de información difusa a partir de datos almacenados de forma precisa.
- Representación y recuperación de información difusa.
- Representación y recuperación de información difusa, y el tratamiento de la misma.

Existen tres enfoques diferentes para implementar la gestión de bases de datos difusas:

1. Modelo Relacional Difuso Básico.
2. Modelo de Unificación Mediante Relaciones de Similitud.
3. Modelos Relacionales sobre Distribuciones de Posibilidad.

Los modelos más importantes basados en el último enfoque son:

- el modelo de **Umano-Fukami**
- el modelo de **Prade-Testemale**
- el modelo de **Zemankova-Kaendel**
- el modelo **GEFRED**

Nosotros nos centraremos en el modelo **GEFRED**.

2.5. GEFRED. Un modelo generalizado de bases de datos difusas

GEFRED es un modelo generalizado de bases de datos difusas que fue introducido en 1994 por Medina J.M., Pons O. y Vila A., de la Universidad de Granada, en la revista *Information Sciences 76 (1), 87-109*. Este modelo se desarrolla en un ambiente posibilístico pero incorporando un tratamiento coherente para aquella información difusa cuyo origen o representación no es posibilística. Las líneas básicas que sustentan el modelo se desglosan en los apartados relativos a la estructura y manipulación de los mismos.

2.5.1. Estructura de datos

Uno de los objetivos del modelo es el de proporcionar una amplia cobertura, en cuanto a la representación y tratamiento, de información difusa. De acuerdo con esto, el tipo de datos sobre los que opera este modelo se enumera en la **Figura 2.12**.

1 Un escalar simple (Ej. Aptitud=buena).
2 Un número simple (Ej. Edad=28).
3 Un conjunto de posibles asignaciones excluyentes de escalares (Ej. Aptitud={mala,buena}).
4 Un conjunto de posibles asignaciones excluyentes de números (Ej. Edad= {20,21}).
5 Una distribución de posibilidad en el dominio de los escalares (Ej. Aptitud= {0.6/mala,0.7/regular}).
6 Una distribución de posibilidad en el dominio de los números (Ej. Edad= {0.4/23, 1.0/24, 0.8/25}, números difusos, etiquetas lingüísticas)
7 Un número Real $\in [0,1]$ representando grados de cumplimiento (Ej. Calidad=0.9).
8 Un valor desconocido **Unknown** dado por la distribución de posibilidad **Unknown**=$\{1/u : u \in U\}$ sobre el dominio considerado.
9 Un valor indefinido **Undefined** dado por la distribución de posibilidad **Undefined**=$\{0/u : u \in U\}$ sobre el dominio considerado.
10 Un valor nulo **NULL** dado por **NULL**={1/Unknown,1/Undefined}

Figura 2.12: Tabla 1: Tipos de datos

El término escalar en informática viene heredado del Álgebra Lineal y se refiere a un dato atómico y unidimensional, en contraposición al concepto de Vector, que se refiere a un dato multidimensional (int, bool, char, short, long, float, double, etc).

Los elementos del modelo que estructuran toda esta información son:

- DOMINIO DIFUSO GENERALIZADO, D_G.

  Definición 7. Si D es el dominio de discurso, $\tilde{P}(D)$ el conjunto de todas las distribuciones de posibilidad definidas sobre D, incluidas las que definen los tipos Unknown y Undefined, (Tipos 8 y 9 resp. de la **Figura 2.12**), y NULL el tipo cuya definición aparece en la citada tabla, llamaremos **Dominio Difuso Generalizado**, D_G, a $D_G \subseteq \tilde{P}(D) \cup NULL$.

  El Dominio Difuso Generalizado constituye el elemento estructural sobre el que se articulará la representación de los datos recogidos en la **Figura 2.12**.

- RELACION DIFUSA GENERALIZADA, R_{FG}.

  Definición 8. Una **Relación Difusa Generalizada**, R_{FG}, viene dada por un par de conjuntos "cabecera", $(\mathcal{H})$ y "cuerpo", $(\mathcal{B})$, $R_{FG} = (\mathcal{H}, \mathcal{B})$ definidos como sigue:

  - La "cabecera" consiste en un conjunto fijo de ternas atributo-dominio-pertenencia,

  $$\mathcal{H} = \{(A_{G_1} : D_{G_1}[, \mu_{A_{G_1}}]),\ (A_{G_2} : D_{G_2}[, \mu_{A_{G_2}}]),...,(A_{G_n} : D_{G_n}[, \mu_{A_{G_n}}])\}$$

  donde a cada atributo A_{Gj} , le subyace un dominio difuso generalizado, no necesariamente distinto, D_{Gj} $(j = 1, 2, ..., n)$ y $\mu_{A_{G_j}}$ es un"atributo de pertenencia" que toma valores en el intervalo [0,1].

  - El "cuerpo" consiste en un conjunto de tuplas difusas generalizadas, donde cada tupla está compuesta por un conjunto de ternas atributo-valor-grado,

  $$\mathcal{B} = \{(A_{G_1} : \tilde{d}_{i1}[, \mu_{i1}]),\ (A_{G_2} : \tilde{d}_{i2}[, \mu_{i2}]),...,\ (A_{G_n} : \tilde{d}_{in}[, \mu_{in}])\}$$

  $(i = 1, 2, ..., m$, siendo m el número de tuplas de la relación), donde $\tilde{d}_{ij}$ representa el valor de dominio que toma la tupla i sobre el atributo A_{Gj} y $\mu_{ij} \neq 0$ el grado de pertenencia de este valor a la tupla i.

  Observaciones:

  - Los corchetes que encierran a $\mu_{A_{G_j}}$ y a μ_{ij} denotan un carácter opcional para el *atributo de pertenencia* y por tanto para los *grados de pertenencia*, en cuanto a su aparición, de forma explícita, en la *cabecera* y *cuerpo* de la relación respectivamente. En una relación difusa generalizada R_{FG}, en la que para un atributo dado, A_{Gj}, no tengamos definido en la *cabecera* el *atributo de pertenencia* $\mu_{A_{G_j}}$ correspondiente, entenderemos que todos los valores de dominio que adopte el *cuerpo* de la relación para dicho atributo, $\tilde{d}_{ij}$, presentarán, implícitamente, un *grado de pertenencia* a la tupla i igual a 1, o sea, $\mu_{ij} = 1\ \forall\ i = 1, 2, ..., m$ siendo m el número de tupla de la relación.
  - Una tupla i, que contenga un valor cero para alguno de los μ_{ij} que la componen, no pertenecerá a la relación R_{FG}.
  - La definición de *relación difusa generalizada* contempla las relaciones clásicas como un caso particular, donde los dominios clásicos sobre los que se construyen los atributos que constituyen la *cabecera* de la relación son casos particulares de la definición de *dominio difuso generalizado* y donde no aparecen los *atributos de pertenencia* en la *cabecera* ni los *grados de pertenencia* en el *cuerpo*, ya que estos son considerados igual a uno.

Definición 9. Sea R_{FG} una relación difusa generalizada dada por:

$$R_{FG} = \left\{ \begin{array}{c} \mathcal{H} = \{(A_{G_1} : D_{G_1}[, \mu_{A_{G_1}}]),\ (A_{G_2} : D_{G_2}[, \mu_{A_{G_2}}]),...,(A_{G_n} : D_{G_n}[, \mu_{A_{G_n}}])\} \\ \mathcal{B} = \{(A_{G_1} : \tilde{d}_{i1}[, \mu_{i1}]),\ (A_{G_2} : \tilde{d}_{i2}[, \mu_{i2}]),...,\ (A_{G_n} : \tilde{d}_{in}[, \mu_{in}])\} \end{array} \right.$$

Llamaremos **Componente de valor** de una relación difusa generalizada y lo denotaremos R^v_{FG}, a la parte de la relación dada por:

$$R^v_{FG} = \left\{ \begin{array}{c} \mathcal{H}^v = \{(A_{G_1} : D_{G_1}),...,\ (A_{G_n} : D_{G_n})\} \\ \mathcal{B}^v = \{(A_{G_1} : \tilde{d}_{i1}),...,\ (A_{G_n} : \tilde{d}_{in})\} \end{array} \right.$$

$\mathcal{H}^v$ y $\mathcal{B}^v$ son las **componentes de valor** de la "cabecera" y el "cuerpo" respectivamente, así mismo llamaremos **componentes de pertenencia** de una relación difusa generalizada y lo denotaremos R^μ_{FG}, a la parte de la relación dada por:

$$R^\mu_{FG} = \left\{ \begin{array}{c} \mathcal{H}^\mu = \{[\mu_{A_{G_1}}]),...,\ [\mu_{A_{G_n}}]\} \\ \mathcal{B}^\mu = \{[\mu_{i1}]),...,\ [\mu_{in}])\} \end{array} \right.$$

$\mathcal{H}^\mu$ y $\mathcal{B}^\mu$ son las **componentes de pertenencia** de la "cabecera" y el "cuerpo" respectivamente.

- CLAVE PRIMARIA GENERALIZADA, K_G.

  Definición 10. Sea R_{FG} una relación difusa generalizada dada por:

$$R_{FG} = \left\{ \begin{array}{c} \mathcal{H} = \{(A_{G_1} : D_{G_1}[, \mu_{A_{G_1}}]),...,(A_{G_n} : D_{G_n}[, \mu_{A_{G_n}}])\} \\ \mathcal{B} = \{(A_{G_1} : \tilde{d}_{i1}[, \mu_{i1}]),...,\ (A_{G_n} : \tilde{d}_{in}[, \mu_{in}])\} \end{array} \right.$$

  con $i = 1, 2, ..., m$, siendo m el número de tuplas de la relación, llamaremos **clave primaria generalizada**, K_G, a un subconjunto de $\mathcal{H}$ expresado como:

$$K_G \subseteq \mathcal{H}, K_G = \{(A_{G_s} : D_{G_s}) \colon s \in S \subseteq \{1, ..., n\}\}$$

  que cumple:

  - $\forall\ s \in S$, D_{G_s} es un dominio numérico o escalar simple (tipos 1 y 2 de la **Figura 2.12**).
  - $\forall\ i, i' \in \{1, ..., m\}$, $\exists s \in S : (A_{G_s} : d_{is}) \neq (A_{G_s} : d_{i's})$, donde el operador $\neq$ recoge el significado clásico de la desigualdad.

  Con la clave primaria definida de esta forma se garantiza, de un lado, la accesibilidad de cada una de las tuplas de la relación y, de otro, la posibilidad de proporcionar un tratamiento difuso a los atributos incluidos en la clave primaria cuando esto resulte conveniente. Además, la existencia de una clave primaria posibilita que se pueda exigir a una base de datos relacional difusa realizada según el modelo propuesto que satisfaga las reglas de identidad y referencial.

2.5.2. Manipulación de los datos

El modelo describe un álgebra relacional difusa generalizada con la que se manipulan las R_{FG},s. Las operaciones clásicas (Unión, Intersección, Diferencia, Producto cartesiano, Proyección, JOIN y Selección) quedan extendidas a fin de que puedan operar sobre las R_{FG},s de forma coherente.

La selección y el JOIN basan su funcionamiento en el empleo de comparadores difusos construidos en torno a alguna de las medidas de compatibilidad difusas aparecidas en la literatura. Nuestro modelo adopta un marco básico, sobre el que construir los comparadores difusos partiendo del significado de las operaciones de comparación definidas sobre el dominio de discurso, y extendiendo dicho significado al tratamiento de valores del *dominio generalizado difuso*.

Operadores de comparación difusa

Un aspecto crítico que ha de abordar cualquier SBRD es la definición de los operadores que se ven envueltos en las operaciones de Selección y JOIN. Este problema lo resuelve el modelo mediante la definición de operadores de comparación a dos niveles: a nivel del Dominio de discurso D y a nivel de D_G. Sobre el primer nivel definiremos el *comparador extendido*, θ_e y con su ayuda construiremos el *comparador difuso generalizado*, θ_G, en el segundo.

- COMPARADOR EXTENDIDO (θ_e)

  Definición 10. Sea D el dominio de discurso considerado, llamaremos **Comparador Extendido**, θ_e, a cualquier operador definido sobre D que pueda ser expresado en la forma:

  $$\theta_e : D \times D \rightarrow [0,1]$$
  $$\theta_e(d_i, d_j) \rightarrow [0,1]$$

  con $d_i, d_j \in D$

  El **comparador extendido** recoge todas las modalidades de relación existentes entre los valores del dominio de discurso D considerado, atendiendo a la naturaleza de dichos valores y el carácter (clásico, difuso o de similitud) de la relación existente entre los mismos.

  Así definido, el **comparador extendido** permite modelar en forma consistente los siguientes operadores de comparación:

  - Comparadores clásicos del álgebra relacional tales como: $=, \neq, >, \geq, <, \leq$. Por ejemplo, el operador *igual extendido* quedaría expresado mediante $=_e (d_i, d_j) = \delta(d_i, d_j)$ donde $\delta(d_i, d_j) = 1$ para $d_i = d_j$ y $\delta(d_i, d_j) = 0$ para $d_i \neq d_j$.
  - Comparadores difusos tales como"aproximadamente igual", "mucho mayor que", etc. Este tipo de comparadores vendrán expresados mediante funciones de pertenencia. Por ejemplo, se podría modelar el operador "aproximadamente igual" mediante el correspondiente comparador extendido $\simeq_e$ con la siguiente función de pertenencia:

    $$\mu_{\simeq e}(d_i, d_j) = e^{-\beta|d_i - d_j|} \text{ con } \beta > 0$$

- Operadores de similitud, los cuales operan sobre datos escalares en los que se ha establecido una *relación de similitud*.
- Comparadores en base a propiedad. Estos comparadores operan sobre dominios discretos y basan su actuación en la definición previa de la relación existente entre cada par de valores pertenecientes a dicho dominio en base a una propiedad observada sobre lo mismo. Contienen en forma explícita el grado en que cada par de valores del producto cartesiano $D \times D$ son compatibles según la propiedad que modela este comparador.

Ejemplo 1 Supongamos $P = \{malo, regular, bueno, excelente\}$, el dominio finito de escalares asociado a un atributo que almacena el rendimiento de un empleado. Se puede considerar sobre este dominio la propiedad "más eficiente que", la cual nos proporciona en qué medida son más eficientes entre sí cada valor del dominio. Así, por ejemplo, esta propiedad pone de manifiesto que el valor de dominio "bueno" es "más eficiente que" el valor "regular" y algo más que "malo". Esto nos conduce, a la hora de modelar este tipo de operadores, a asignar a cada par de valores un grado de compatibilidad, comprendido entre 0 y 1, con la propiedad "más eficiente que". Con lo cual, por ejemplo, podríamos modelar la propiedad "más eficiente que" definida sobre el dominio P mediante la siguiente relación:

d	d'	$mas\_eficiente\_que_e(d, d')$
malo	*malo*	*0*
regular	*malo*	*0.4*
⋮	⋮	⋮
excelente	*bueno*	*0.5*
excelente	*excelente*	*0*

Figura 2.13: Tabla 2

De forma similar se podrían modelar otras propiedades definidas sobre el dominio P, como por ejemplo: "mucho más eficiente que", "menos eficiente que", etc.

- COMPARADOR DIFUSO GENERALIZADO (θ_G)

Definición 11. Sea D el dominio de discurso considerado, sea D_G el dominio difuso generalizado construido sobre él y sea θ_e un comparador extendido definido en D. Consideremos una función θ_G definida:

$$\theta_G : D_G \times D_G \rightarrow [0, 1]$$
$$\theta_G(\tilde{d_1}, \tilde{d_2}) \in [0, 1]$$

Diremos que θ_G es un **comparador difuso generalizado** sobre D_G inducido por el comparador extendido θ_e, si cumple:

$$\theta_G(\tilde{d_1}, \tilde{d_2}) = \theta_e(d_1, d_2) \qquad \forall\ d_1, d_2 \in D$$

donde $\tilde{d_1}, \tilde{d_2}$ representan las distribuciones de posibilidad, $1/d_1$, $1/d_2$, inducidas, respectivamente, por los valores d_1, d_2.

Álgebra relacional difusa generalizada

Con los comparadores difusos anteriormente definidos y con la estructura de datos adoptada por el modelo, estamos en condiciones de extender el significado de los operadores del álgebra relacional, a fin de que podamos manipular en forma conveniente las "relaciones difusas generalizadas".

- UNIÓN DIFUSA GENERALIZADA, $\cup_G$

  Definición 12. Sean R_{FG} y R'_{FG} dos relaciones difusas generalizadas dadas por:

$$R_{FG} = \begin{cases} \mathcal{H} = \{(A_{G_1} : D_{G_1}[,\mu_{A_{G_1}}]),..., (A_{G_n} : D_{G_n}[,\mu_{A_{G_n}}])\} \\ \mathcal{B} = \{(A_{G_1} : \tilde{d}_{i1}[,\mu_{i1}]),..., (A_{G_n} : \tilde{d}_{in}[,\mu_{in}])\} \end{cases}$$

$$R'_{FG} = \begin{cases} \mathcal{H}' = \{(A_{G_1} : D_{G_1}[,\mu_{A_{G_1}}]),..., (A_{G_n} : D_{G_n}[,\mu_{A_{G_n}}])\} \\ \mathcal{B}' = \{(A_{G_1} : \tilde{d}'_{k1}[,\mu'_{k1}]),..., (A_{G_n} : \tilde{d}'_{kn}[,\mu'_{kn}])\} \end{cases}$$

  con $i = 1,,...,m$ y $k = 1,,...,m'$, siendo m y m' las respectivas cardinalidades, entonces la **unión difusa generalizada** de R_{FG} y R'_{FG} vendrá definida por:

$$R_{FG} \cup_G R'_{FG} = \begin{cases} \mathcal{H}_{\cup_G} = \{(A_{G_1} : D_{G_1}[,\mu_{A_{G_1}}]),..., (A_{G_n} : D_{G_n}[,\mu_{A_{G_n}}])\} \\ \mathcal{B}_{\cup_G} = \begin{cases} \mathcal{B}^v_{\cup_G} = \mathcal{B}^v \cup \mathcal{B}'^v \\ \mathcal{B}^\mu_{\cup_G} = \{[\mu''_{l1}],...,[\mu''_{ln}]\} \end{cases} \end{cases}$$

  con $l = 1,...,m''$, siendo m'' la cardinalidad de la unión, donde

$$\mu''_{lj} = max\{\mu_{lj}, \mu'_{lj}\}$$

  μ_{lj} el grado de pertenencia de d''_{lj} a la tupla l en la relación R_{FG} y μ'_{lj} el grado de pertenencia de pertenencia de d''_{lj} a la tupla l en la relación R'_{FG}.

  Por tanto, la unión difusa generalizada de dos relaciones difusas generalizadas, contiene aquellas tuplas que pertenecen a la unión de ambas relaciones, adoptando, para el grado de pertenencia a cada tupla de la unión, μ''_{lj}, el máximo de los grados de pertenencia adoptados por dichos valores en las relaciones de origen, $\mu''_{lj} = max\{\mu_{lj}, \mu'_{lj}\}$.

- INTERSECCIÓN DIFUSA GENERALIZADA, $\cap_G$

  Definición 13. Sean R_{FG} y R'_{FG} dos relaciones difusas generalizadas dadas por:

$$R_{FG} = \begin{cases} \mathcal{H} = \{(A_{G_1} : D_{G_1}[,\mu_{A_{G_1}}]),..., (A_{G_n} : D_{G_n}[,\mu_{A_{G_n}}])\} \\ \mathcal{B} = \{(A_{G_1} : \tilde{d}_{i1}[,\mu_{i1}]),..., (A_{G_n} : \tilde{d}_{in}[,\mu_{in}])\} \end{cases}$$

$$R'_{FG} = \begin{cases} \mathcal{H}' = \{(A_{G_1} : D_{G_1}[,\mu_{A_{G_1}}]),..., (A_{G_n} : D_{G_n}[,\mu_{A_{G_n}}])\} \\ \mathcal{B}' = \{(A_{G_1} : \tilde{d}'_{k1}[,\mu'_{k1}]),..., (A_{G_n} : \tilde{d}'_{kn}[,\mu'_{kn}])\} \end{cases}$$

  con $i = 1,,...,m$ y $k = 1,,...,m'$, siendo m y m' las respectivas cardinalidades, entonces la **intersección difusa generalizada** de R_{FG} y R'_{FG} vendrá definida por:

$$R_{FG} \cap_G R'_{FG} = \left\{ \begin{array}{c} \mathcal{H}_{\cap_G} = \{(A_{G_1} : D_{G_1}[, \mu_{A_{G_1}}]),..., (A_{G_n} : D_{G_n}[, \mu_{A_{G_n}}])\} \\ \mathcal{B}_{\cap_G} = \left\{ \begin{array}{c} \mathcal{B}^v_{\cap_G} = \mathcal{B}^v \cap \mathcal{B}'^v \\ \mathcal{B}^\mu_{\cap_G} = \{[\mu''_{l1}], ..., [\mu''_{ln}]\} \end{array} \right. \end{array} \right.$$

con $l = 1, ..., m''$, siendo m'' la cardinalidad de la intersección, donde

$$\mu''_{lj} = min\{\mu_{lj}, \mu'_{lj}\}$$

μ_{lj} el grado de pertenencia de d''_{lj} a la tupla l en la relación R_{FG} y μ'_{lj} el grado de pertenencia de pertenencia de d''_{lj} a la tupla l en la relación R'_{FG}.

Así pues, la intersección difusa generalizada de dos relaciones difusas generalizadas contiene aquellas tuplas que pertenecen a la intersección de ambas relaciones, adoptando, para el grado de pertenencia a cada tupla de la unión, μ''_{lj}, el mínimo de los grados de pertenencia adoptados por dichos valores en las relaciones de origen, $\mu''_{lj} = min\{\mu_{lj}, \mu'_{lj}\}$.

- DIFERENCIA DIFUSA GENERALIZADA, $-_G$

  Definición 14. Sean R_{FG} y R'_{FG} dos relaciones difusas generalizadas dadas por:

$$R_{FG} = \left\{ \begin{array}{c} \mathcal{H} = \{(A_{G_1} : D_{G_1}[, \mu_{A_{G_1}}]),..., (A_{G_n} : D_{G_n}[, \mu_{A_{G_n}}])\} \\ \mathcal{B} = \{(A_{G_1} : \tilde{d}_{i1}[, \mu_{i1}]),..., (A_{G_n} : \tilde{d}_{in}[, \mu_{in}])\} \end{array} \right.$$

$$R'_{FG} = \left\{ \begin{array}{c} \mathcal{H}' = \{(A_{G_1} : D_{G_1}[, \mu_{A_{G_1}}]),..., (A_{G_n} : D_{G_n}[, \mu_{A_{G_n}}])\} \\ \mathcal{B}' = \{(A_{G_1} : \tilde{d}'_{k1}[, \mu'_{k1}]),..., (A_{G_n} : \tilde{d}'_{kn}[, \mu'_{kn}])\} \end{array} \right.$$

  con $i = 1, , ..., m$ y $k = 1, , ..., m'$, siendo m y m' las respectivas cardinalidades, entonces la **diferencia difusa generalizada** de R_{FG} y R'_{FG} vendrá definida por:

$$R_{FG} -_G R'_{FG} = \left\{ \begin{array}{c} \mathcal{H}_{-_G} = \{(A_{G_1} : D_{G_1}[, \mu_{A_{G_1}}]),..., (A_{G_n} : D_{G_n}[, \mu_{A_{G_n}}])\} \\ \mathcal{B}_{-_G} = \left\{ \begin{array}{c} \mathcal{B}^v_{-_G} = \mathcal{B}^v - \mathcal{B}'^v \\ \mathcal{B}^\mu_{-_G} = \{[\mu''_{l1}], ..., [\mu''_{ln}]\} \end{array} \right. \end{array} \right.$$

  con $l = 1, ..., m''$, siendo m'' la cardinalidad de la diferencia, donde

$$\mu''_{lj} = min\{\mu_{lj}, 1 - \mu'_{lj}\}$$

  μ_{lj} el grado de pertenencia de d''_{lj} a la tupla l en la relación R_{FG} y μ'_{lj} el grado de pertenencia de pertenencia de d''_{lj} a la tupla l en la relación R'_{FG}.

  Como puede observarse, la diferencia difusa generalizada de dos relaciones difusas generalizadas contiene aquellas tuplas que pertenecen a la diferencia de ambas relaciones, adoptando, para el grado de pertenencia a cada tupla de la diferencia, μ''_{lj}, el mínimo entre el grado de pertenencia a la tupla en la relación R_{FG} y el complemento del grado de pertenencia a la tupla de la relación R'_{FG}, $\mu''_{lj} = min\{\mu_{lj}, 1 - \mu'_{lj}\}$.

- PRODUCTO CARTESIANO DIFUSO GENERALIZADO, $\times_G$

  Definición 15. Sean R_{FG} y R'_{FG} dos relaciones difusas generalizadas dadas por:

$$R_{FG} = \left\{ \begin{array}{c} \mathcal{H} = \{(A_{G_1} : D_{G_1}[, \mu_{A_{G_1}}]),..., (A_{G_n} : D_{G_n}[, \mu_{A_{G_n}}])\} \\ \mathcal{B} = \{(A_{G_1} : \tilde{d}_{i1}[, \mu_{i1}]),..., (A_{G_n} : \tilde{d}_{in}[, \mu_{in}])\} \end{array} \right.$$

$$R'_{FG} = \begin{cases} \mathcal{H}' = \{(A_{G_1} : D_{G_1}[,\mu_{A_{G_1}}]),..., (A_{G_{n'}} : D_{G_{n'}}[,\mu_{A_{G_{n'}}}])\} \\ \mathcal{B}' = \{(A_{G_1} : \tilde{d}'_{k1}[,\mu'_{k1}]),..., (A_{G_{n'}} : \tilde{d}'_{kn'}[,\mu'_{kn'}])\} \end{cases}$$

con $i = 1,,...,m$ y $k = 1,,...,m'$, siendo m y m' las respectivas cardinalidades y n, n' los respectivos grados, entonces el **producto cartesiano difuso generalizado** se define como:

$$R_{FG} \times_G R'_{FG} = \begin{cases} \mathcal{H}_{\times_G} = \mathcal{H} \times \mathcal{H}' \\ \mathcal{B}_{\times_G} = \mathcal{B} \times \mathcal{B}' \end{cases}$$

Como se puede observar el producto cartesiano difuso generalizado opera igual que el producto cartesiano usual, solo que considera a los atributos de pertenencia y a sus valores como partes integrantes de las relaciones.

- PROYECCIÓN DIFUSA GENERALIZADA, P_G

  Definición 16. Sea R_{FG} una relación difusa generalizada dada por:

$$R_{FG} = \begin{cases} \mathcal{H} = \{(A_{G_1} : D_{G_1}[,\mu_{A_{G_1}}]),..., (A_{G_n} : D_{G_n}[,\mu_{A_{G_n}}])\} \\ \mathcal{B} = \{(A_{G_1} : \tilde{d}_{r1}[,\mu_{r1}]),..., (A_{G_n} : \tilde{d}_{rn}[,\mu_{rn}])\} \end{cases}$$

  con $r = 1,,...,m$, siendo m el número de tuplas de la relación.

  Sea X un subconjunto de $\mathcal{H}$ expresado como:

$$X \subseteq \mathcal{H}, X = \{(A_{G_s} : D_{G_s}[,\mu_{A_{G_{s'}}}]) : s \in S, s' \in S'; S, S' \subseteq \{1,...,n\}\}$$

  entonces diremos que la **proyección difusa generalizada** de R_{FG} sobre X, $P_G(R_{FG}; X)$ es una relación difusa generalizada dada por:

$$P_G(R_{FG}; X) = \begin{cases} \mathcal{H}_P = X \\ \mathcal{B}_P = \{(A_{G_s} : \tilde{d}_{rs}[,\mu_{rs'}])\} \end{cases}$$

  con $s \in S, s' \in S', S, S' \subseteq \{1,...,n\}$.

  La proyección se puede realizar sobre cualquier subconjunto de la *cabecera*, pudiendo incluir éste "atributo de valor" así como "atributo de pertenencia". El resultado de la proyección es una "selección vertical" sobre el *cuerpo* de la relación.

- SELECCIÓN DIFUSA GENERALIZADA, S_G

  Definición 17. Sea R_{FG} una relación difusa generalizada dada por:

$$R_{FG} = \begin{cases} \mathcal{H} = \{(A_{G_1} : D_{G_1}[,\mu_{A_{G_1}}]),..., (A_{G_n} : D_{G_n}[,\mu_{A_{G_n}}])\} \\ \mathcal{B} = \{(A_{G_1} : \tilde{d}_{r1}[,\mu_{r1}]),..., (A_{G_n} : \tilde{d}_{rn}[,\mu_{rn}])\} \end{cases}$$

  con $r = 1,,...,m$, siendo m el número de tuplas de la relación.

  Sea $\tilde{a} \in D_G$ una constante, sea θ_G un "comparador difuso generalizado" y sea $\gamma \in [0,1]$ un"umbral de cumplimiento", entonces la **selección difusa generalizada** realizada sobre R_{FG} por la condición inducida por θ_G compuesto con $\tilde{a}$ sobre el atributo A_{Gi} y cualificada por γ, $S_G(R_{FG}; \theta_G(A_i; \tilde{a}) \geq \gamma)$, es una relación R^S_{FG} dada por:

$$R^S_{FG} = \left\{ \begin{array}{c} \mathcal{H} = \{(A_{G_1} : D_{G_1}[, \mu_{A_{G_1}}]),..., (A_{Gn} : D_{Gn}[, \mu_{A_{G_n}}])\} \\ \mathcal{B} = \{(A_{G1} : \tilde{d}_{r'1}[, \mu_{r'1}]),..., (A_{Gi} : \tilde{d}_{r'i}[, \mu_{r'i}]),..., (A_{Gn} : \tilde{d}_{r'n}[, \mu_{r'n}])\} \end{array} \right.$$

con

$$\mu'_{r'i} = \theta_G(\tilde{d}_{r'i}; \tilde{a}) \geq \gamma$$

$r' = 1, ..., m'$, siendo m' la cardinalidad de la selección.

- Reglas de composición sobre la selección generalizada, S_G

Definición 18. Sea R_{FG} una relación difusa generalizada dada por:

$$R_{FG} = \left\{ \begin{array}{c} \mathcal{H} = \{(A_{G_1} : D_{G_1}[, \mu_{A_{G_1}}]),..., (A_{G_n} : D_{G_n}[, \mu_{A_{G_n}}])\} \\ \mathcal{B} = \{(A_{G_1} : \tilde{d}_{r1}[, \mu_{r1}]),..., (A_{G_n} : \tilde{d}_{rn}[, \mu_{rn}])\} \end{array} \right.$$

con $r = 1, , ..., m$, siendo m el número de tuplas de la relación y

- sea $S \subseteq \{1, ..., n\}$ siendo n el "grado" de R_{FG},
- sean $\tilde{a}_i \in D_{G_i}$, $i \in S$ un conjunto de constantes no necesariamente distintas y A_{G_i}, $i \in S$ un conjunto de atributos de la cabecera de R_{FG} no necesariamente distintos,
- sean $\theta_{G_i}, i \in S$, un conjunto de comparadores difusos generalizados, no necesariamente distintos,
- sean $\gamma_i \in [0, 1]$, un conjunto de "umbrales de cumplimiento",
- y sean $S_G(R_{FG}; \theta_{G_i}(A_i, \tilde{a}_i) \geq \gamma_i), i \in S$, el conjunto de selecciones difusas generalizadas inducidas por las constantes $\tilde{a}_i$ y los comparadores difusos generalizados, θ_{G_i} sobre los atributos A_{G_i} de R_{FG} y cualificadas por γ_i.

Entonces definimos las siguientes composiciones sobre la selección generalizada:

- Llamaremos **complemento de la selección generalizada**, $\neg S_G(R_{FG}; \neg\theta_{G_i}(A_i, \tilde{a}_i) \geq \gamma_i) = S_G(R_{FG}; \theta_{G_i}(A_i, \tilde{a}_i) \geq \gamma_i)$, a la relación difusa generalizada, $R^{\neg S}_{FG}$ dada por:

$$R^{\neg S}_{FG} = \left\{ \begin{array}{c} \mathcal{H} = \{(A_{G_1} : D_{G_1}[, \mu_{A_{G_1}}]),..., (A_{G_n} : D_{G_n}[, \mu_{A_{G_n}}])\} \\ \mathcal{B} = \{(A_{G_1} : \tilde{d}_{r'1}[, \mu_{r'1}]),..., (A_{G_i} : \tilde{d}_{r'i}[, \mu_{r'i}]),..., (A_{G_n} : \tilde{d}_{r'n}[, \mu_{r'n}])\} \end{array} \right.$$

  con

$$\mu'_{r'i} = 1 - \theta_{G_i}(\tilde{d}_{r'i}; \tilde{a}) \geq \gamma$$

  $r' = 1, ..., m'$, siendo m' la cardinalidad de la selección.
- Se define la **conjunción sobre la selección generalizada** mediante la expresión:

$$\bigwedge S_G(R_{FG}; \theta_{G_i}(A_i, \tilde{a}_i) \geq \gamma_i) = \bigcap_G(R_{FG}; \theta_{G_i}(A_i, \tilde{a}_i) \geq \gamma_i)$$

- La **disyunción sobre la selección generalizada** viene dada por la expresión:

$$\bigvee S_G(R_{FG}; \theta_{G_i}(A_i, \tilde{a}_i) \geq \gamma_i) = \bigcup_G(R_{FG}; \theta_{G_i}(A_i, \tilde{a}_i) \geq \gamma_i)$$

La **selección difusa generalizada** así definida, proporciona un mecanismo por el cual se ajustan los "grados de pertenencia" de los "valores atributo" según las condiciones impuestas en la consulta y se eliminan aquellas tuplas que no satisfacen un "umbral de cumplimiento" establecido. Para ello, se hace uso de los "comparadores difusos generalizados" empleados para modelar las condiciones. También se han introducido reglas mediante las cuales componer consultas complejas a partir de unas más simples.

- JOIN DIFUSO GENERALIZADO, $\bowtie_G$

  Definición 19. Sean R_{FG} y R'_{FG} dos relaciones difusas generalizadas dadas por:

$$R_{FG} = \begin{cases} \mathcal{H} = \{(A_{G_1} : D_{G_1}[, \mu_{A_{G_1}}]),..., (A_{G_n} : D_{G_n}[, \mu_{A_{G_n}}])\} \\ \mathcal{B} = \{(A_{G_1} : \tilde{d}_{i1}[, \mu_{i1}]),..., (A_{G_n} : \tilde{d}_{in}[, \mu_{in}])\} \end{cases}$$

$$R'_{FG} = \begin{cases} \mathcal{H}' = \{(A_{G_1} : D_{G_1}[, \mu_{A_{G_1}}]),..., (A_{G_{n'}} : D_{G_{n'}}[, \mu_{A_{G_{n'}}}])\} \\ \mathcal{B}' = \{(A_{G_1} : \tilde{d}'_{k1}[, \mu'_{k1}]),..., (A_{G_{n'}} : \tilde{d}'_{kn'}[, \mu'_{kn'}])\} \end{cases}$$

  con $i = 1, , ..., m$ y $k = 1, , ..., m'$, siendo m y m' las respectivas cardinalidades y n, n' los respectivos grados.

  Sea θ_G un "comparador difuso generalizado" y sea $\gamma \in [0, 1]$ un "umbral de cumplimiento", entonces la **JOIN difusa generalizada** realizada sobre $R_{FG} \times_G R'_{FG}$ mediante la condición θ_G inducida sobre los atributos A_{G_i} de R_{FG} y $A'_{G_{i'}}$ de R'_{FG}, cualificada por

$$\gamma'', \bowtie_G (R_{FG} \times_G R'_{FG}; \theta_G(A_i, A'_{i'}) \geq \gamma)$$

  viene dada por:

$$R^{\bowtie}_{FG} = \begin{cases} \mathcal{H}^{\bowtie} = \mathcal{H} \times \mathcal{H}' \\ \mathcal{B}^{\bowtie} = \{(A_{G_1} : \tilde{d}_{r1}[, \mu_{r1}]),..., (A_{G_i} : \tilde{d}_{ri}, \mu''_{ri}),...,(A'_{G_{i'}} : \tilde{d}'_{ri'}, \mu''_{ri'}), \\ ...,(\text{A'}_{G_{n''}} : \tilde{d}'_{rn''}[, \mu_{rn''}])\} \end{cases}$$

  con $n'' = n + n'$, donde r es la cardinalidad del JOIN y

$$\mu''_{ri} = \mu''_{ri'} = \theta_G(\tilde{d}_{ri}, \tilde{d}'_{ri'}) \geq \gamma$$

  La *JOIN difusa generalizada* se puede entender como una especie de *selección difusa generalizada* realizada sobre el *producto cartesiano difuso generalizado* de las dos relaciones implicadas, y cuyo resultado es la variación de los grados de pertenencia de los atributos sobre los que se ejecuta la JOIN y la supresión de aquellas tuplas que no presenten un "grado de JOIN" superior o igual a un umbral establecido.

2.5.3. Ejemplo de representación y manipulación de una BD relacional difusa según GEFRED

Para ilustrar el funcionamiento del modelo vamos a emplear el ejemplo de una relación que recoge datos, algunos de ellos de naturaleza difusa, sobre un conjunto de empleados. Este

ejemplo no agota todas las posibilidades del modelo GEFRED, tanto en la representación, como en la manipulación de información difusa.

Suponemos que la información que almacenan los atributos NOMBRE y CALLE son de naturaleza precisa. Los dominios que subyacen a EDAD y SALARIO, son numéricos y tienen definidos sobre ellos las distribuciones de posibilidad (expresadas en forma trapezoidal) etiquetadas y definidas como muestras las **Figuras 2.14, 2.15 y 2.16**. Se observa la presencia de valores "Unknown" en ambos atributos pero, dada la naturaleza de la relación, no existen valores "undefined" y por tanto tampoco "NULL". El dominio que subyace al atributo RENDIMIENTO es el tipo 1 de la **Figura 2.12**. En la **Figura 2.18** se especifican los valores del dominio así como la relación de similitud que existe entre ellos. También permite en principio valores "Unknown".

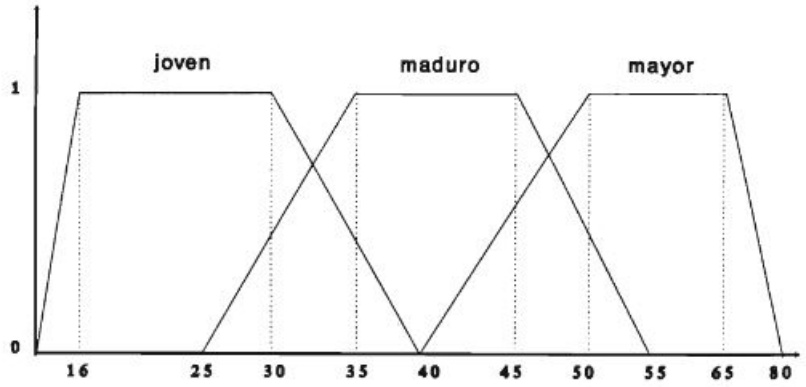

Figura 2.14: Etiquetas lingüísticas definidas sobre el dominio EDAD

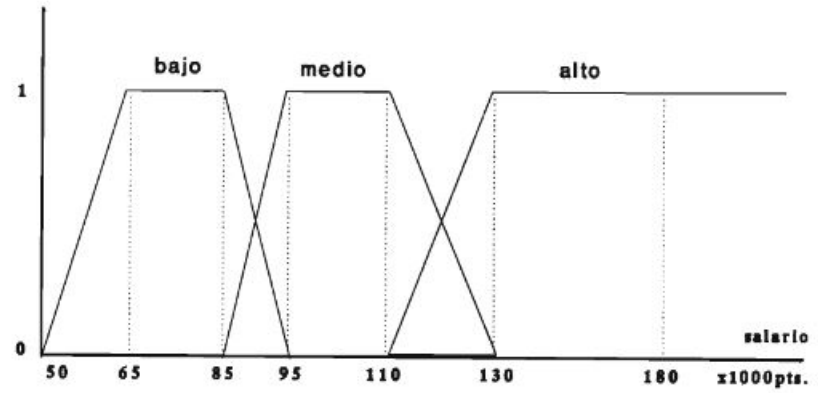

Figura 2.15: Etiquetas lingüísticas definidas sobre el dominio SALARIO

De acuerdo con el modelo GEFRED, la "cabecera" de la relación difusa generalizada que estructura la información vendría dada por:

$$\Lambda = \{NOMBRE : D_{NOMBRE}, CALLE : D_{CALLE}, EDAD : D_{EDAD}, RENDIMIENTO : D_{RENDIEMIENTO}, SALARIO : D_{SALARIO}\}$$

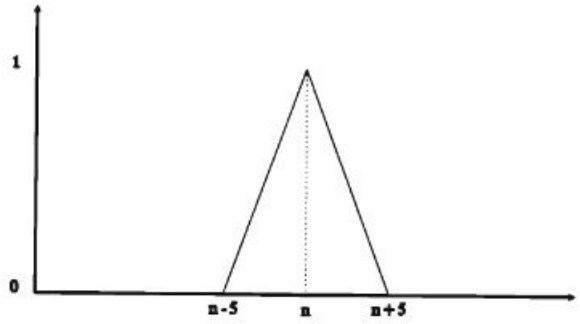

Figura 2.16: Etiqueta "aproximadamente" sobre el dominio EDAD

$\mathcal{H}$	NOMBRE	CALLE	EDAD	RENDIMIENTO	SALARIO
$\mathcal{B}$	Luis	Recogidas	&.8/30,1/31	Bueno	110000
	Antonio	Reyes Católicos	Maduro	Regular	100000
	Juan Carlos	Camino Ronda	Joven	Malo	90000
	Francisco	P. A. Alarcón	Mayor	Excelente	150000
	Julia	Puerta Real	Joven	Bueno	130000
	Inés	Manuel de Falla	#28	Bueno	125000
	Javier	Gran Vía	*30,35	Regular	105000

El símbolo # significa "aproximadamente" y su definición aparece en la figura 3, & indica distribución de posibilidad y * intervalo.

Figura 2.17: Tabla 2:Tabla de empleados, **Emp**

Donde no aparecen los atributos de pertenencia es porque el grado de pertenencia de cada valor de atributo a tupla es igual a 1. El "cuerpo" de la relación está constituido por todas las tuplas mostradas en la relación de la **Figura 2.17**.

$s_e(d, d')$	Malo	Regular	Bueno	Excelente
Malo	1	0.8	0.5	0.1
Regular	0.8	1	0.7	0.5
Bueno	0.5	0.7	1	0.8
Excelente	0.1	0.5	0.8	1

Figura 2.18: Tabla 3: Relación de analogía sobre el dominio RENDIMIENTO.

Para poner de manifiesto el funcionamiento del *álgebra relacional difuso generalizado*, vamos a estudiar el proceso de resolución de una consulta ejemplo.

En la primera consulta estamos interesados en encontrar empleados que con un rendimiento "bueno" perciban un salario "alto", o que con una edad "mayor" tengan un salario "medio". En términos más precisos, podríamos formular la siguiente consulta:

Consulta: Dame el NOMBRE, la EDAD, el RENDIMIENTO y el SALARIO, así como el grado en que cada atributo satisface la condición, de aquellos empleados que presenten RENDIMIENTO "bueno" (con un grado $\geq 0{,}9$) y un SALARIO "alto" (con un grado $\geq 0{,}7$) ó que tengan una EDAD "mayor" (con un grado $\geq 0{,}6$) y un SALARIO "medio" (con un grado $\geq 0{,}7$).

Los "comparadores difusos generalizados" implícitos en la consulta deberán, o estar definidos en la implementación, o bien estar almacenados cómo componentes de metaconocimiento de la Base de Datos Difusa. GEFRED, en este aspecto, no precisa como se ha de modelar el comparador de forma específica. Esto es así porque la naturaleza de la información condiciona, además de la representación de los datos, la forma de operar con ellos. En nuestro ejemplo vamos a emplear los siguientes modelos para los comparadores implicados en la consulta:

- Para la compatibilidad en los atributos EDAD y SALARIO empleamos el "comparador extendido" habitual $=_e (d, d') = \delta(d, d')$ y el "comparador difuso generalizado" inducido:

$$\theta^{=_e}(\tilde{d}, \tilde{d'}) = sup(d, d') \in D \times D \ / \ min(=_e (d, d') = \delta(d, d'), \pi_{\tilde{d}}(d), \pi_{\tilde{d'}}(d'))$$

  donde, $\pi_{\tilde{d}}(d)$ y $\pi_{\tilde{d'}}(d')$ son las distribuciones de posibilidad asociadas a $\tilde{d}$ y $\tilde{d'}$, respectivamente.

- Para RENDIMIENTO emplearemos el "comparador extendido" $s_e(\tilde{d}, \tilde{d'})$ dado en la **Figura 2.18**. El "comparador difuso generalizado" usado será:

$$\theta^{s_e}(\tilde{d}, \tilde{d'}) = sup(d, d') \in D \times D \ / \ min(s_e(d, d') = \delta(d, d'), \pi_{\tilde{d}}(d), \pi_{\tilde{d'}}(d'))$$

  donde nuevamente, $\pi_{\tilde{d}}(d)$ y $\pi_{\tilde{d'}}(d')$ son las distribuciones de posibilidad asociadas a $\tilde{d}$ y $\tilde{d'}$, respectivamente.

Una vez precisados los comparadores, la consulta en los términos de álgebra de GEFRED queda expresada como sigue.

$$\mathcal{P}_G((S_1 \wedge_G S_2) \vee_G (S_3 \wedge_G S_4; X)$$

$\mathcal{H}$	NOMBRE	CALLE	EDAD	REND	μ_{REND}	SALARIO
$\mathcal{B}$	Luis	Recogidas	&.8/30,1/31	Bueno	1	110000
	Julia	Puerta Real	Joven	Bueno	1	130000
	Inés	Manuel de Falla	#28	Bueno	1	125000

Los valores de μ_{REND} se obtienen de la relación de similitud mostrada en la Tabla 3. Las tuplas que presentan un valor $< \gamma = 0.9$ para el grado de pertenecia del atributo RENDIMIENTO no pertenecen a la relación.

Figura 2.19: Tabla 4: Cálculo de S_1

$\mathcal{H}$	NOMBRE	CALLE	EDAD	RENDIMIENTO	SALARIO	μ_{SAL}
$\mathcal{B}$	Francisco	P. A. Alarcón	Mayor	Excelente	150000	1
	Julia	Puerta Real	Joven	Bueno	130000	1
	Inés	Manuel de Falla	#28	Bueno	125000	0.75

Los valores de μ_{SAL}, que se cálculan según la ecuación 26, para etiquetas trapezoidales pueden ser calculados de forma trigonométrica. Las tuplas que presentan un valor $< \gamma = 0.7$ para el grado de pertenecia del atributo SAL no pertenecen a la relación.

Figura 2.20: Tabla 5: Cálculo de S_2

donde:

- $S_1 = S_G(Emp, \theta^{S_e}(RENDIMIENTO, Bueno) \geq 0{,}9)$ con "Bueno" $\in D_{REND}$.
- $S_2 = S_G(Emp, \theta^{=_e}(SALARIO, Alto) \geq 0{,}7)$ siendo "Alto" la etiqueta que referencia la distribución de posibilidad mostrada en la **Figura 2.15**.
- $S_3 = S_G(Emp, \theta^{=_e}(EDAD, Mayor) \geq 0{,}6)$ siendo "Mayor" la etiqueta que referencia la distribución de posibilidad mostrada en la **Figura 2.14**.
- $S_4 = S_G(Emp, \theta^{=_e}(SALARIO, Medio) \geq 0{,}7)$ siendo "Medio" la etiqueta que referencia la distribución de posibilidad mostrada en la **Figura 2.15**.
- $X = \{NOMB, EDAD, \mu_{EDAD}, REND, \mu_{REND}, SAL, \mu_{SAL}\}$

En las **Figuras 2.19, 2.20, 2.21, 2.22, 2.23, 2.24, 2.25 y 2.26**, se detallan las diferentes operaciones llevadas a cabo para resolver la consulta.

$\mathcal{H}$	NOMBRE	CALLE	EDAD	μ_{EDAD}	RENDIMIENTO	SALARIO
$\mathcal{B}$	Antonio	Reyes Católicos	Maduro	0.75	Regular	100000
	Francisco	P. A. Alarcón	Mayor	1	Excelente	150000
	Julia	Puerta Real	Joven	1	Bueno	130000

Los valores de μ_{EDAD}, que se cálculan según la ecuación 26, para etiquetas trapezoidales pueden ser calculados de forma trigonométrica. Las tuplas que presentan un valor $< \gamma = 0.6$ para el grado de pertenecia del atributo EDAD no pertenecen a la relación.

Figura 2.21: Tabla 6: Cálculo de S_3

$\mathcal{H}$	NOMBRE	CALLE	EDAD	RENDIMIENTO	SALARIO	μ_{SAL}
$\mathcal{B}$	Luis	Recogidas	&.8/30,1/31	Bueno	110000	1
	Antonio	Reyes Católicos	Maduro	Regular	100000	1
	Inés	Manuel de Falla	#28	Bueno	125000	0.75
	Javier	Gran Vía	*30,35	Regular	105000	1

Los valores de μ_{SAL}, que se cálculan según la ecuación 26, para etiquetas trapezoidales pueden ser calculados de forma trigonométrica. Las tuplas que presentan un valor $< \gamma = 0.7$ para el grado de pertenecia del atributo SAL no pertenecen a la relación.

Figura 2.22: Tabla 7: Cálculo de S_4

$\mathcal{H}$	NOMBRE	CALLE	EDAD	REND	μ_{REND}	SALARIO	μ_{SAL}
$\mathcal{B}$	Julia	Puerta Real	Joven	Bueno	1	130000	1
	Inés	Manuel de Falla	#28	Bueno	1	125000	0.75

Para el cálculo de la "Conjunción Generalizada Difusa" se halla la "Intersección Generalizada Difusa" de las relaciones calculadas por las consultas $\mathcal{S}_1$ y $\mathcal{S}_2$.

Figura 2.23: Tabla 8: Cálculo de $S_1 \wedge_G S_2$

$\mathcal{H}$	NOMBRE	CALLE	EDAD	μ_{EDAD}	REND	SALARIO	μ_{SAL}
$\mathcal{B}$	Antonio	Reyes Católicos	Maduro	0.75	Regular	100000	1

Para el cálculo de la "Conjunción Generalizada Difusa" se halla la "Intersección Generalizada Difusa" de las relaciones calculadas por las consultas $\mathcal{S}_3$ y $\mathcal{S}_4$.

Figura 2.24: Tabla 9: Cálculo de $S_3 \wedge_G S_4$

$\mathcal{H}$	...	EDAD	μ_{EDAD}	RENDIMIENTO	μ_{REND}	SALARIO	μ_{SAL}
$\mathcal{B}$	...	Maduro	0.75	Regular	1	100000	1
	...	Mayor	1	Excelente	0.8	150000	1
	...	Joven	1	Bueno	1	130000	1
	...	#28	1	Bueno	1	125000	0.75

Para el cálculo de la "Disyunción Generalizada Difusa" se halla la "Unión Generalizada Difusa" de $(\mathcal{S}_1 \wedge_G \mathcal{S}_2)$ y $(\mathcal{S}_3 \wedge_G \mathcal{S}_4)$.

Figura 2.25: Tabla 10: Cálculo de $((S_1 \wedge_G S_2) \vee_G (S_3 \wedge_G S_4))$

2.5.4. Conclusiones

El modelo que presentamos en este trabajo aporta las siguientes características frente a las otras propuestas de modelos:

- Proporciona tratamiento difuso a la información: tanto a los datos difusos, como a las relaciones que existen entre ellos.
- Los tipos de datos con los que opera son bastante extenso en comparación con otras

$\mathcal{H}$	NOMBRE	EDAD	μ_{EDAD}	RENDIMIENTO	μ_{REND}	SALARIO	μ_{SAL}
$\mathcal{B}$	Antonio	Maduro	0.75	Regular	1	100000	1
	Julia	Joven	1	Bueno	1	130000	1
	Inés	#28	1	Bueno	1	125000	0.75

Se proyecta sobre los atributos, ya sean de valor ó de pertenencia.

Figura 2.26: Tabla 11: $P_G((S_1 \wedge_G S_2) \vee_G (S_3 \wedge_G S_4; X))$

propuestas. Su naturaleza difusa es más dispersa que en modelos precedentes.

- Organiza de forma consistente la información difusa. Las "relaciones difusas generalizadas" estructuran tanto la información de partida como la que resulta de las operaciones realizadas sobre la misma. Una tupla difusa generalizada se entiende como una serie de valores de atributo, difusos o no, que presentan un grado de pertenencia, mayor o menor, a la relación que dicha tupla expresa. En el ejemplo anterior una de las tuplas que aparece como resultado de la consulta, p.e., es {Antonio, (Maduro,0.75), (Regular,1), (100000,1)}; esto significa que en la base de datos hay una tupla que corresponde a las condiciones de la consulta con grado 0.75 para los requerimientos de edad, y con un grado 1, para los de rendimiento y salario.
- Dentro del conjunto de metainformación que el experto o el usuario debe proporcionar a la Base de Datos se encuentra, además de la definición que adopta para las diferentes etiquetas, la del modelo de comparador difuso que va a asociar a cada una de las comparaciones a establecer para cada dominio definido.
- El modelo presenta gran flexibilidad para el tratamiento y evaluación de la información difusa, basado en la libertad a la hora de seleccionar un comparador difuso generalizado u otro y en que se puede controlar el grado en que se satisfacen las condiciones individuales de una consulta.
- Una gran parte de los modelos precedentes se pueden considerar, bajo ciertas condiciones, como casos particulares de GEFRED. Por tanto, este modelo se puede emplear para representarlos.

Por último, el modelo también presenta una serie de desventajas, como por ejemplo:

- El lenguaje de consulta resulta incómodo debido al gran número de parámetros que deben utilizarse.
- Los comparadores abstractos hacen difícil la decisión de cuál debemos usar.
- Falta de estandarización derivado de la poca popularidad de este tipo de bases de datos pues es un campo novedoso y, por tanto, el número de especialistas en dicho campo es muy bajo.

Bibliografía

[1] J. M. Medina , M. A. Vila y O. Pons, *GEFRED. A Generalized Model of Fuzzy Relational Databases*, Information Sciences, 76, 1-2, pp 87-109. (1994)

[2] Galingo J., Urrutia A. y Piattini M., *Fuzzy databases modeling, design and implemetation*, United States of America y United Kingdom, 2006

[3] http://www.koala-soft.com/bases-de-datos-difusas *Bases de datos difusas*, 2012.

[4] Medina Rodriguez, J.M. y A. Vila Miranda, *GEFRED. Un modelo generalizado de Bases de datos Difusas*, Granada, España, 1994

[5] Angélica Urrutia, José Galindo y Alejandro Sepúlveda, *Implementación de una base de datos difusa con FIRST-2 y PostgreSQL*, Huelva, España, 2010

[6] Marcela Varas Contreras y Angélica Urrutia Sepúlveda, *Bases de datos difusas modeladas con UML*, Chile

[7] Mercedes Marqués, *Bases de datos*, Castellón , España, 2011

Appendices

Apéndice A

Programas JAVA

A.1. Programa 1

```
1
2 public class Usuario {
3
4     public String fecha;
5     public String codfac;
6     public int cod;
7     public String nom;
8     public String num;
9
10  public Usuario(String fecha, String codfac, int cod, String nom, String num) {
11          this.fecha=fecha;
12          this.codfac=codfac;
13          this.cod=cod;
14          this.nom=nom;
15          this.num=num;
16
17      }
18 }
19
20 import java.io.BufferedReader;
21 import java.io.File;
22 import java.io.FileNotFoundException;
23 import java.io.FileReader;
24 import java.io.FileWriter;
25 import java.io.IOException;
26 import java.util.ArrayList;
27 import java.util.List;
28 import java.util.Vector;
29
30 import com.csvreader.CsvWriter;
31
32 import dominio.Usuario;
33
34 public class ReadCVS {
35
36  public static void main(String[] args) {
37
```

```
38    ReadCVS obj = new ReadCVS();
39    obj.run();
40
41  }
42
43  public void run() {
44
45    String csvFile = "facts_Diciembre.csv";
46    BufferedReader br = null;
47    String line = "";
48
49    try {
50
51        List<Usuario> usuarios = new ArrayList();
52
53        br = new BufferedReader(new FileReader(csvFile));
54
55        while ((line = br.readLine()) != null ) {
56
57
58            String[] palabras = line.split("/");
59
60            String fecha=palabras[0];
61            String nombre=palabras[1];
62            String codfac=palabras[2];
63            Integer codpais=Integer.parseInt(palabras[3].trim());
64            String iva= palabras[4];
65            String trans= palabras[5];
66            String prod= palabras[6];
67            String ttconIVA= palabras[7];
68            String ttsinIVA= palabras[8];
69            String metodoPago= palabras[9];
70            Double dto= Double.parseDouble(palabras[10].trim());
71            //String metodoPago2=palabras[11];
72
73            int codv,codt,codi,codmp,coddto;
74            int[] v;
75            Vector<Integer> ue=new Vector<Integer>();
76            ue.add(2);
77            ue.add(5);
78            ue.add(59);
79            ue.add(78);
80            ue.add(79);
81            ue.add(76);
82            ue.add(80);
83            ue.add(201);
84            ue.add(202);
85            ue.add(4);
86            ue.add(89);
87            ue.add(94);
88            ue.add(102);
89            ue.add(18);
90            ue.add(8);
91            ue.add(3);
92            ue.add(133);
93            ue.add(139);
94            ue.add(140);
95            ue.add(148);
96            ue.add(17);
97            ue.add(184);
```

```
98              ue.add(25);
99              ue.add(7);
100             ue.add(188);
101             ue.add(20);
102
103
104             //Ventas
105             if( codpais==1){
106                 codv=707100;
107                 codt=708500;
108                 codi=445710;
109                 if (dto.equals(0.0)){
110                     coddto=0;
111                 }else{
112                     coddto=709700;
113                 }
114
115             }
116             else if( ue.contains(codpais) ){ //dentro UE
117
118                 codv=707950;
119                 codt=708501;
120                 codi=445711;
121                 if (dto.equals(0.0)){
122                     coddto=0;
123                 }else{
124                     coddto=709750;
125                 }
126             }
127             else {
128                 codv=707951;
129                 codt=708501;
130                 codi=0;
131                 if (dto.equals(0.0)){
132                     coddto=0;
133                 }else{
134                     coddto=709751;
135                 }
136             }
137             codmp=580001;
138
139             String nom=nombre+"-"+codfac;
140
141
142             //System.out.println(fecha + codfac + codv + nom + prod);
143             //System.out.println(fecha + codfac + codt + nom + trans);
144             //System.out.println(fecha + codfac + codi + nom + iva);
145             //System.out.println(fecha + codfac + coddto + nom + dto);
146             //System.out.println(fecha + codfac + codmp + nom + ttconIVA);
147
148
149             String nuedto=String.valueOf(dto);
150
151             usuarios.add(new Usuario(fecha,codfac, codv , nom, prod));
152             usuarios.add(new Usuario(fecha,codfac, codt , nom, trans));
153             usuarios.add(new Usuario(fecha,codfac, codi , nom, iva));
154             usuarios.add(new Usuario(fecha,codfac, coddto , nom, nuedto));
155             usuarios.add(new Usuario(fecha,codfac, codmp , nom, ttconIVA));
156
157
```

```
158         }
159
160         //Hago la exportación
161
162         String outputFile = "diciembre_export.csv";
163         boolean alreadyExists = new File(outputFile).exists();
164
165         if(alreadyExists){ //Si existe lo borro
166             File ficheroUsuarios = new File(outputFile);
167             ficheroUsuarios.delete();
168         }
169
170         try {
171
172             CsvWriter csvOutput = new CsvWriter(new FileWriter(outputFile, true),
                    ',');
173
174             csvOutput.write("Fecha");
175             csvOutput.write("Código de factura");
176             csvOutput.write("Código");
177             csvOutput.write("Nombre-Código de factura");
178             csvOutput.write("Importe");
179
180             csvOutput.endRecord();
181
182             for(Usuario us : usuarios){
183
184                 csvOutput.write(us.fecha);
185                 csvOutput.write(us.codfac);
186                 csvOutput.write(String.valueOf(us.cod));
187                 csvOutput.write(us.nom);
188                 csvOutput.write(us.num);
189
190                 csvOutput.endRecord();
191             }
192
193             csvOutput.close();
194
195         } catch (IOException e) {
196             e.printStackTrace();
197         }
198
199     } catch (FileNotFoundException e) {
200         e.printStackTrace();
201     } catch (IOException e) {
202         e.printStackTrace();
203     } finally {
204         if (br != null) {
205             try {
206                 br.close();
207             } catch (IOException e) {
208                 e.printStackTrace();
209             }
210         }
211     }
212
213     System.out.println("Done");
214
215 }
216
```

```
217
218}
```

A.2. Programa 2

```
1
2public class Linea {
3        public String fechaE;
4    public String fechaP;
5    public String codfac;
6    public String codpedido;
7    public double peso;
8    public String codpos;
9    public double transSINiva;
10   public String transCONiva;
11   public double ttcSINiva;
12   public double costeMat;
13
14   public Linea(String fechaE,String fechaP,String codfac, String codpedido,
         double peso, String codpos, double transSINiva, String transCONiva,double
         ttcSINiva, double costeMat ){
15       this.fechaE=fechaE;
16       this.fechaP=fechaP;
17       this.codfac=codfac;
18       this.codpedido=codpedido;
19       this.peso=peso;
20       this.codpos=codpos;
21       this.transSINiva=transSINiva;
22       this.transCONiva=transCONiva;
23       this.ttcSINiva=ttcSINiva;
24       this.costeMat=costeMat;
25   }
26
27}
```

A.3. Programa 3

```
1
2public class Simulador {
3        public String transportista;
4        public int departamento;
5        public double pesoInf;
6        public double pesoSup;
7        public int peso;
8        public double precio;
9
10       public Simulador(String transportista, int departamento, double pesoInf,
             double pesoSup, int peso, double precio){
11               this.transportista=transportista;
12               this.departamento=departamento;
```

```
13                  this.pesoInf=pesoInf;
14                  this.pesoSup=pesoSup;
15                  this.peso=peso;
16                  this.precio=precio;
17
18          }
19
20}
```

A.4. Programa 4

```
1import java.io.BufferedReader;
2import java.io.File;
3import java.io.FileNotFoundException;
4import java.io.FileReader;
5import java.io.FileWriter;
6import java.io.IOException;
7import java.util.ArrayList;
8import java.util.List;
9import java.util.Vector;
10
11import com.csvreader.CsvWriter;
12
13import dominio.Linea;
14import dominio.Simulador;
15import dominio.Usuario;
16
17public class ReadCVS {
18
19  public static void main(String[] args) {
20
21    ReadCVS obj = new ReadCVS();
22    obj.run();
23
24  }
25
26
27  public void run() {
28
29            String csvFile = "trans_mayo.csv";
30            BufferedReader br = null;
31            String line = "";
32
33            try {
34
35                List<Linea> lineas = new ArrayList();
36
37                br = new BufferedReader(new FileReader(csvFile));
38
39                while ((line = br.readLine()) != null ) {
40
41
42                    String[] palabras = line.split("/");
43
44                    String fechaE=palabras[0];
45                    String fechaP=palabras[1];
```

```
46                      String codfac=palabras[2];
47                      String codpedido=palabras[3];
48                      Double peso= Double.parseDouble(palabras[4].trim());
49                      String codpos= palabras[5];
50                      Double transSINiva= Double.parseDouble(palabras[6].trim());
51                      String transCONiva= palabras[7];
52                      Double ttcSINiva= Double.parseDouble(palabras[8].trim());
53                      Double costeMat= Double.parseDouble(palabras[9].trim());
54                      Integer codpais=Integer.parseInt(palabras[10].trim());
55
56
57                      lineas.add(new Linea(fechaE,fechaP, codfac , codpedido, peso,
                            codpos, transSINiva, transCONiva, ttcSINiva, costeMat,
                            codpais)); //Faltaría añadir codpais
58
59                  }
60
61                  System.out.println("Hago la exportación");
62                  //Hago la exportación
63
64                  String outputFile = "trans_mayo_export.csv";
65                  boolean alreadyExists = new File(outputFile).exists();
66
67                  if(alreadyExists){ //Si existe lo borro
68                      File ficheroUsuarios = new File(outputFile);
69                      ficheroUsuarios.delete();
70                  }
71
72                  try {
73
74                      CsvWriter csvOutput = new CsvWriter(new FileWriter(outputFile
                            , true), ',');
75
76                      csvOutput.write("Fecha envio");
77                      csvOutput.write("Fecha pedido");
78                      csvOutput.write("Código de factura");
79                      csvOutput.write("Código de pedido");
80                      csvOutput.write("Peso");
81                      csvOutput.write("Destino");
82                      csvOutput.write("pr. transp");
83                      csvOutput.write("DSV");
84                      csvOutput.write("TRANSP");
85                      csvOutput.write("Transporte sin IVA");
86                      csvOutput.write("Transporte con IVA");
87                      csvOutput.write("PR REAL");
88                      csvOutput.write("DIF");
89                      csvOutput.write("Ttcfac sin IVA");
90                      csvOutput.write("Coste material");
91                      csvOutput.write("Coste total");
92                      csvOutput.write("Margen");
93                      csvOutput.write("Margen%");
94
95                      csvOutput.endRecord();
96
97                      String csvsim=null;
98
99                  System.out.println("Segunda importación");
100
101                     for(Linea us : lineas){
102
```

```
103                 if(us.codpais==1){ //Francia
104                         csvsim = "Simulador_fr_barras.csv";
105                 }else if(us.codpais==2){
106                         csvsim="Simulador_belgica.csv";
107                 }else if(us.codpais==5){
108                         csvsim="Simulador_alemania.csv";
109                 }else if(us.codpais==41){
110                         csvsim="Simulador_austria.csv";
111                 }else if(us.codpais==140){
112                         csvsim="Simulador_lux.csv";
113                 //}else if(us.codpais==xx){
114                         //csvsim="Simulador_holanda.csv";
115                 }else if(us.codpais==184){
116                         csvsim="Simulador_polonia.csv";
117                 }else if(us.codpais==7){
118                         csvsim="Simulador_reinoUnido.csv";
119                 }else if(us.codpais==79){
120                         csvsim="Simulador_repche.csv";
121                 }else if(us.codpais==4){
122                         csvsim="Simulador_España.csv";
123                 }
124
125
126                 BufferedReader brsim = null;
127                     String linesim = "";
128
129                     List<Simulador> lineasim = new ArrayList();
130
131                 brsim = new BufferedReader(new FileReader(csvsim));
132                 //System.out.println(us.codpais);
133                 while ((linesim = brsim.readLine()) != null ) {
134
135
136                     String[] palabras = linesim.split("/");
137
138                     String transportista=palabras[0];
139                     Integer departamento=Integer.parseInt(palabras[1].
                            trim());
140                     Double pesoInf=Double.parseDouble(palabras[2].trim())
                            ;
141                     Double pesoSup=Double.parseDouble(palabras[3].trim())
                            ;
142                     Integer peso= Integer.parseInt(palabras[4].trim());
143                     Double precio= Double.parseDouble(palabras[5].trim())
                            ;
144
145                     lineasim.add(new Simulador(transportista,departamento
                            , pesoInf ,pesoSup, peso,precio));
146
147                 }
148
149
150                     String codigo=null;
151
152                     if ( us.codpos.length() == 6){
153                             codigo=us.codpos.substring(0,2);
154                     }
155                     else{
156                             codigo=us.codpos.substring(0,3);
157                     }
```

```
158
159                         Integer cod=Integer.parseInt(codigo.trim()); //
                                cod=departamento
160
161
162
163                         int j =0;//pos buena
164                         Double precio=1000.0;
165                         for(int i=0;i<lineasim.size(); i++){
166                                 if(lineasim.get(i).departamento == cod ||
                                        lineasim.get(i).departamento==-1){
167                                     //ahora comparo pesos y luego
                                            elijo precio
168                                     if(lineasim.get(i).pesoInf < us.
                                            peso && lineasim.get(i).
                                            pesoSup> us.peso){
169                                         if (precio>lineasim.get(i
                                                ).precio){
170                                             precio=lineasim.
                                                    get(i).precio;
171                                             j=i;
172                                         }
173                                     }
174                                 }
175                         }
176
177
178                         int k=0; //pos segundo mas barato
179                         Double precio2=1000.0;
180                         for(int i=0;i<lineasim.size(); i++){
181                                 if((lineasim.get(i).departamento == cod
                                        || lineasim.get(i).departamento==-1)
                                       && !lineasim.get(j).transportista.
                                        equals(lineasim.get(i).transportista))
                                        {
182                                     //ahora comparo pesos y luego
                                            elijo precio
183                                     if(lineasim.get(i).pesoInf < us.
                                            peso && lineasim.get(i).
                                            pesoSup> us.peso){
184                                         if (precio2>lineasim.get(
                                                i).precio ){
185                                             precio2=lineasim.
                                                    get(i).precio;
186                                             k=i;
187                                         }
188                                     }
189                                 }
190                         }
191
192                         csvOutput.write(us.fechaE);
193                         csvOutput.write(us.fechaP);
194                         csvOutput.write(us.codfac);
195                         csvOutput.write(us.codpedido);
196                         csvOutput.write(String.valueOf(us.peso));
197                         csvOutput.write(String.valueOf(cod));
198
199                         String cad="DSV";
200
```

```
201                             int res = lineasim.get(j).transportista.compareTo
                                    (cad) ;
202
203                             if(res!=1){ //Si DSV no es el mas barato
204
205                                     csvOutput.write(String.valueOf(lineasim.
                                            get(j).precio));
206
207                                     if(!lineasim.get(k).transportista.equals(
                                            "DSV")){
208                                             csvOutput.write("");
209                                     }
210                                     else{//Si DSV fuese el segundo mas
                                            barato
211                                             csvOutput.write(String.valueOf(
                                                    lineasim.get(k).precio));
212                                     }
213                                     csvOutput.write(lineasim.get(j).
                                            transportista);
214                             }
215                             else{ //Si DSV es el mas barato
216                                     csvOutput.write(String.valueOf(lineasim.
                                            get(k).precio));//Precio segundo mas
                                            barato
217                                     csvOutput.write(String.valueOf(lineasim.
                                            get(j).precio));
218                                     csvOutput.write(lineasim.get(k).
                                            transportista);
219                             }
220
221                             csvOutput.write(String.valueOf(us.transSINiva));
222                             csvOutput.write(us.transCONiva);
223                             Double prReal=  lineasim.get(j).precio + 12.35;
224                             csvOutput.write(String.valueOf(prReal));
225                             Double dif=us.transSINiva-prReal;
226                             csvOutput.write(String.valueOf(dif));
227                             csvOutput.write(String.valueOf(us.ttcSINiva));
228                             csvOutput.write(String.valueOf(us.costeMat));
229                             Double costeTotal= us.costeMat + prReal;
230                             csvOutput.write(String.valueOf(costeTotal));
231                             Double margen=us.ttcSINiva-costeTotal;
232                             csvOutput.write(String.valueOf(margen));
233                             Double margen100 = (margen/us.ttcSINiva)*100;
234                             csvOutput.write(String.valueOf(margen100));
235
236                             csvOutput.endRecord();
237
238                     }
239
240                         csvOutput.close();
241
242                 } catch (IOException e) {
243                     e.printStackTrace();
244                 }
245
246             } catch (FileNotFoundException e) {
247                 e.printStackTrace();
248             } catch (IOException e) {
249                 e.printStackTrace();
250             } finally {
```

```
251                 if (br != null) {
252                     try {
253                         br.close();
254                     } catch (IOException e) {
255                         e.printStackTrace();
256                     }
257                 }
258             }
259             System.out.println("Adios");
260 
261         }
```

A.5. Programa 5: Transportista más barato

```
1 
2 package Trans;
3 
4 import java.io.BufferedReader;
5 import java.io.FileNotFoundException;
6 import java.io.FileReader;
7 import java.io.IOException;
8 import java.util.ArrayList;
9 import java.util.List;
10 import java.util.Scanner;
11 import java.util.Vector;
12 
13 import dominio.Simulador;
14 
15 public class Transporte {
16 
17         public static void main(String[] args)throws IOException {
18 
19                 int cod;
20                 double peso;
21 
22 
23                 String entrada1 = "";
24                 String entrada2="";
25             Scanner entradaEscaner = new Scanner (System.in); //Creación de un
                    objeto Scanner
26             System.out.println ("Por favor introduzca las dos primeras cifras del
                    código postal por teclado:");
27             entrada1 = entradaEscaner.nextLine (); //Invocamos un método sobre un
                     objeto Scanner
28             System.out.println ("Ahora introduzca el peso:");
29                 entrada2=entradaEscaner.nextLine ();
30 
31             cod=Integer.parseInt(entrada1);
32             peso=Double.parseDouble(entrada2);
33 
34             System.out.println("Código:"+cod+ " Peso:"+ peso);
35 
36             MiHebra h=new MiHebra(cod,peso);
37             Vector<String> ret= h.run();
38 
39             System.out.println( "Transportista: " + ret.get(0) );
```

```
40              System.out.println( "Precio: " + ret.get(1) );
41          }
42
43  }
44
45  class MiHebra{
46
47          int cod;
48          double peso;
49
50          public MiHebra(int cod,double peso){
51                  this.cod=cod;
52                  this.peso=peso;
53          }
54
55          public Vector<String> run(){
56                  String csvsim=null;
57
58          //if(us.codpais==1){ //Francia
59                   csvsim = "Simulador_fr_barras.csv";
60              //}
61
62          BufferedReader brsim = null;
63              String linesim = "";
64              Vector<String> v=new Vector<String>();
65
66              try {
67
68              List<Simulador> lineasim = new ArrayList();
69
70          brsim = new BufferedReader(new FileReader(csvsim));
71
72          while ((linesim = brsim.readLine()) != null ) {
73
74
75              String[] palabras = linesim.split("/");
76
77              String transportista=palabras[0];
78              Integer departamento=Integer.parseInt(palabras[1].trim());
79              Double pesoInf=Double.parseDouble(palabras[2].trim());
80              Double pesoSup=Double.parseDouble(palabras[3].trim());
81              Integer peso= Integer.parseInt(palabras[4].trim());
82              Double precio= Double.parseDouble(palabras[5].trim());
83
84              lineasim.add(new Simulador(transportista,departamento, pesoInf ,
                    pesoSup, peso,precio));
85
86          }
87
88              int j =0;//pos buena
89                  Double precio=1000.0;
90                  for(int i=0;i<lineasim.size(); i++){
91                          if(lineasim.get(i).departamento == cod){
92                                  //ahora comparo pesos y luego elijo precio
93                                  if(lineasim.get(i).pesoInf < peso && lineasim.get
                                        (i).pesoSup>peso){
94                                      if (precio>lineasim.get(i).precio){
95                                              precio=lineasim.get(i).precio;
96                                              j=i;
97                                      }
```

```
98
99                                  }
100                             }
101                 }
102
103
104             v.add(String.valueOf(lineasim.get(j).transportista));
105             v.add(String.valueOf(lineasim.get(j).precio));
106             } catch (FileNotFoundException e) {
107                 e.printStackTrace();
108             } catch (IOException e) {
109                 e.printStackTrace();
110             } finally {
111                 if (brsim != null) {
112                     try {
113                         brsim.close();
114                     } catch (IOException e) {
115                         e.printStackTrace();
116                     }
117                 }
118             }
119
120             return v;
121
122         }
123
124 }
```

A.6. Programa 6: Transportista más barato usando interfaces gráficas

```
1
2
3 import java.awt.*;
4 import java.awt.event.*;
5 import java.io.BufferedReader;
6 import java.io.FileNotFoundException;
7 import java.io.FileReader;
8 import java.io.IOException;
9 import java.util.ArrayList;
10 import java.util.List;
11 import java.util.Vector;
12
13 import javax.swing.*;
14
15 import dominio.Simulador;
16
17 public class binario extends JFrame implements ActionListener{
18     private JLabel eti1,eti2,eti3,eti4,eti5,eti6,eti7;
19     JTextField entrada1,entrada2,entrada3,bin1,bin2,bin3,bin4;
20     JButton calcular;
21
22     public binario(){
23         super("Calculo transportista mas barato");
24         Container contenedor=getContentPane();
25         contenedor.setLayout(new FlowLayout());
```

```
26
27          eti1=new JLabel("Por favor introduzca el código del país. Los códigos son
                los siguientes: Francia: 1, Bélgica: 2, Alemania: 5, Austria: 41,
                Luxemburgo: 140, Polonia: 184, Reino Unido: 7, República Checa: 79, España
                : 4   ");
28          contenedor.add(eti1);
29          entrada1=new JTextField(10);
30          contenedor.add(entrada1);
31          eti2=new JLabel("Por favor introduzca las dos primeras cifras del código
                postal por teclado, en el caso de que solo tuviese 4 cifras,
                introduzca solo la primera:");
32          contenedor.add(eti2);
33          entrada2=new JTextField(10);
34          contenedor.add(entrada2);
35          eti3=new JLabel("Ahora introduzca el peso:");
36          contenedor.add(eti3);
37          entrada3=new JTextField(10);
38          contenedor.add(entrada3);
39
40
41          eti4=new JLabel("Transportista más barato:");
42          contenedor.add(eti4);
43          bin1=new JTextField(10);
44          contenedor.add(bin1);
45          eti5=new JLabel("Precio:");
46          contenedor.add(eti5);
47          bin2=new JTextField(10);
48          contenedor.add(bin2);
49          eti6=new JLabel("Segundo transportista más barato:");
50          contenedor.add(eti6);
51          bin3=new JTextField(10);
52          contenedor.add(bin3);
53          eti7=new JLabel("Precio:");
54          contenedor.add(eti7);
55          bin4=new JTextField(10);
56          contenedor.add(bin4);
57          calcular=new JButton("Calcular");
58          contenedor.add(calcular);
59          calcular.addActionListener(this);
60
61
62
63          setSize(1500,300);
64          setVisible(true);
65      }
66
67      public static void main(String args[]){
68          binario aplicacion=new binario();
69          aplicacion.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
70      }
71
72      int cod;
73      double peso;
74      int codpais;
75
76
77      public void actionPerformed(ActionEvent evento){
78
79          codpais=Integer.parseInt(entrada1.getText());
80          cod=Integer.parseInt(entrada2.getText());
```

```
81          peso=Double.parseDouble(entrada3.getText())+7;
82
83          if(evento.getSource()==calcular){
84
85              String csvsim=null;
86
87              if(codpais==1){ //Francia
88                  csvsim = "Simulador_fr_barras.csv";
89               }else if(codpais==2){
90                  csvsim="Simulador_belgica.csv";
91               }else if(codpais==5){
92                  csvsim="Simulador_alemania.csv";
93               }else if(codpais==41){
94                  csvsim="Simulador_austria.csv";
95               }else if(codpais==140){
96                  csvsim="Simulador_lux.csv";
97               //}else if(codpais==xx){
98                  //csvsim="Simulador_holanda.csv";
99               }else if(codpais==184){
100                 csvsim="Simulador_polonia.csv";
101              }else if(codpais==7){
102                 csvsim="Simulador_reinoUnido.csv";
103              }else if(codpais==79){
104                 csvsim="Simulador_repche.csv";
105              }else if(codpais==4){
106                 csvsim="Simulador_España.csv";
107              }
108
109
110             BufferedReader brsim = null;
111             String linesim = "";
112             Vector<String> v=new Vector<String>();
113
114             try {
115
116             List<Simulador> lineasim = new ArrayList();
117
118             brsim = new BufferedReader(new FileReader(csvsim));
119
120             while ((linesim = brsim.readLine()) != null ) {
121
122
123                 String[] palabras = linesim.split("/");
124
125                 String transportista=palabras[0];
126                 Integer departamento=Integer.parseInt(palabras[1].trim());
127                 Double pesoInf=Double.parseDouble(palabras[2].trim());
128                 Double pesoSup=Double.parseDouble(palabras[3].trim());
129                 Integer peso= Integer.parseInt(palabras[4].trim());
130                 Double precio= Double.parseDouble(palabras[5].trim());
131
132                 lineasim.add(new Simulador(transportista,departamento, pesoInf ,
                        pesoSup, peso,precio));
133
134             }
135
136             String s1="DSV";
137             String s2="SERLOGIS";
138
139                 int j =0;//pos buena
```

```
140             Double precio=1000.0;
141             for(int i=0;i<lineasim.size(); i++){
142                 if((lineasim.get(i).departamento == cod || lineasim.get(i).
                        departamento==-1) &&(lineasim.get(i).transportista.
                        compareTo(s1)!=1 && lineasim.get(i).transportista.
                        compareTo(s2)!=1) ){
143                     //ahora comparo pesos y luego elijo precio
144                     if(lineasim.get(i).pesoInf < peso && lineasim.get(i).
                            pesoSup>peso){
145                         if (precio>lineasim.get(i).precio){
146                             precio=lineasim.get(i).precio;
147                             j=i;
148                         }
149
150                     }
151                 }
152             }
153
154             int k=0; //pos segundo mas barato
155             Double precio2=1000.0;
156             for(int i=0;i<lineasim.size(); i++){
157                 if((lineasim.get(i).departamento == cod || lineasim.get(i).
                        departamento==-1) && !lineasim.get(j).transportista.equals
                        (lineasim.get(i).transportista) && (lineasim.get(i).
                        transportista.compareTo(s1)!=1 && lineasim.get(i).
                        transportista.compareTo(s2)!=1)){
158                     //ahora comparo pesos y luego elijo precio
159                     if(lineasim.get(i).pesoInf < peso && lineasim.get(i).
                            pesoSup>peso){
160                         if (precio2>lineasim.get(i).precio ){
161                             precio2=lineasim.get(i).precio;
162                             k=i;
163                         }
164                     }
165                 }
166             }
167
168             v.add(String.valueOf(lineasim.get(j).transportista));
169             v.add(String.valueOf(lineasim.get(j).precio));
170             v.add(String.valueOf(lineasim.get(k).transportista));
171             v.add(String.valueOf(lineasim.get(k).precio));
172         } catch (FileNotFoundException e) {
173             e.printStackTrace();
174         } catch (IOException e) {
175             e.printStackTrace();
176         } finally {
177             if (brsim != null) {
178                 try {
179                     brsim.close();
180                 } catch (IOException e) {
181                     e.printStackTrace();
182                 }
183             }
184         }
185
186         bin1.setText(v.get(0));
187         bin2.setText(v.get(1));
188         bin3.setText(v.get(2));
189         bin4.setText(v.get(3));
190
```

```
191
192         }
193     }
194
195 }
```

A.7. Programa 7

```
1
2 public class Ps {
3
4         public String idOr;
5         public String idCus;
6         public String last;
7         public String first;
8         public String email;
9         public double ttcPagado;
10        public String codfac;
11        public String modpag;
12        public String dateAdd;
13        public double peso;
14
15        public Ps(String idOr, String idCus, String last, String first, String
              email, double ttcPagado, String codfac,String modpag, String dateAdd,
              double peso){
16                this.idOr=idOr;
17                this.idCus=idCus;
18                this.last=last;
19                this.first=first;
20                this.email=email;
21                this.ttcPagado=ttcPagado;
22                this.codfac=codfac;
23                this.modpag=modpag;
24                this.dateAdd=dateAdd;
25                -this.peso=peso;
26
27        }
28
29 }
```

A.8. Programa 8

```
1
2  public void run() {
3
4             String csvFile = "ps_abril.csv";
5             BufferedReader br = null;
6             String line = "";
7
8             try {
9
10                List<Ps> ps = new ArrayList();
11
```

```
12              br = new BufferedReader(new FileReader(csvFile));
13
14              while ((line = br.readLine()) != null ) {
15
16
17                  String[] palabras = line.split("/");
18
19                  String idOrder=palabras[0];
20                  String idCus=palabras[1];
21                  String last=palabras[2];
22                  String first=palabras[3];
23                  String email=palabras[4];
24                  Double ttcPagado= Double.parseDouble(palabras[5].trim());
25                  String codfac= palabras[6];
26                  String modpag= palabras[7];
27                  String dateAdd= palabras[8];
28
29                  Double peso= Double.parseDouble(palabras[9].trim());
30
31                  ps.add(new Ps(idOrder,idCus, last, first, email,ttcPagado,
                        codfac,modpag,dateAdd,peso));
32
33              }
34
35              //Hago la exportación
36
37              String outputFile = "ps_export.csv";
38              boolean alreadyExists = new File(outputFile).exists();
39
40              if(alreadyExists){ //Si existe lo borro
41                  File ficheroUsuarios = new File(outputFile);
42                  ficheroUsuarios.delete();
43              }
44
45              try {
46
47                  CsvWriter csvOutput = new CsvWriter(new FileWriter(outputFile
                        , true), ',');
48
49                  csvOutput.write("id_order");
50                  csvOutput.write("id_customer");
51                  csvOutput.write("Apellidos");
52                  csvOutput.write("Nombre");
53                  csvOutput.write("e-mail");
54                  csvOutput.write("tt_paye");
55                  csvOutput.write("nº_factura");
56                  csvOutput.write("modo_de_pago");
57                  csvOutput.write("Estado_pedido");
58                  csvOutput.write("Fecha_creación");
59                  csvOutput.write("Proveedor");
60                  csvOutput.write("Pedido_fábrica");
61                  csvOutput.write("Confirmación_fabrica");
62                  csvOutput.write("Peso_automático");
63                  csvOutput.write("Peso_real");
64
65                  csvOutput.endRecord();
66
67                  for(Ps us :ps){
68
69                      csvOutput.write(us.idOr);
```

```
                        csvOutput.write(us.idCus);
                        csvOutput.write(us.last);
                        csvOutput.write(us.first);
                        csvOutput.write(us.email);
                        csvOutput.write(String.valueOf(us.ttcPagado));
                        csvOutput.write(us.codfac);
                        csvOutput.write(us.modpag);
                        csvOutput.write("");
                        csvOutput.write(us.dateAdd);
                        csvOutput.write("");
                        csvOutput.write("");
                        csvOutput.write("");
                        csvOutput.write(String.valueOf(us.peso));

                        csvOutput.endRecord();
                    }

                    csvOutput.close();

                } catch (IOException e) {
                    e.printStackTrace();
                }

            } catch (FileNotFoundException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                if (br != null) {
                    try {
                        br.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }

            System.out.println("Done");

        }
```

Índice de figuras