ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERO EN INFORMÁTICA

VISUAL FSQL: GESTIÓN VISUAL DE BASES DE DATOS DIFUSAS EN ORACLE A TRAVÉS DE INTERNET USANDO FSQL

Realizado por

RAFAEL FRANCISCO OLIVA MORENO

Dirigido por

JOSÉ GALINDO GÓMEZ

Departamento

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA

MÁLAGA, DICIEMBRE DE 2003

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERO EN INFORMÁTICA

Reunido el tribunal examinador en el día de la fecha, constituido por:

Presidente Dº/Dª. _____

Secretario Dº/Dª. _____

Vocal Dº/Dª. _____

Para juzgar el proyecto Fin de Carrera titulado:

VISUAL FSQL: GESTIÓN VISUAL DE BASES DE DATOS DIFUSAS EN ORACLE A TRAVÉS DE INTERNET USANDO FSQL

del alumno Dº/Dª. **RAFAEL FRANCISCO OLIVA MORENO**

dirigido por Dº/Dª. **JOSÉ GALINDO GÓMEZ**

ACORDÓ POR _____ OTORGAR LA CALIFICACIÓN DE _____

Y PARA QUE CONSTE, SE EXTIENDE FIRMADA POR LOS COMPARECIENTES DEL TRIBUNAL, LA PRESENTE DILIGENCIA.

Málaga, a de del 200__

El Presidente El Secretario El Vocal

Fdo: Fdo: Fdo:

VISUAL FSQL

v. 1.0

GESTIÓN VISUAL DE BASES DE DATOS EN ORACLE A TRAVÉS DE INTERNET USANDO FSQL

RAFAEL FRANCISCO OLIVA MORENO

Agradecimientos

Quiero expresar mi agradecimiento más sincero por el apoyo incondicional de mis padres Rafael y M.Carmen, mi hermana M.Carmen, mis sobrinos Carlos y Natalia, mi cuñado Carlos y toda mi familia en general.

También merece mi agradecimiento José Galindo Gómez, tutor de este proyecto, por la paciencia demostrada durante el desarrollo del mismo y por su colaboración y sugerencias.

ÍNDICE GENERAL

Introducción

El objetivo del proyecto es desarrollar una aplicación web para acceder a ella a través de internet y poder realizar consultas a bases de datos difusas o tradicionales, localizadas en el mismo servidor que la aplicación ó un servidor remoto de Bases de Datos.

Con esta aplicación podremos conectarnos al servidor de Bases de Datos (BD) y tendremos la oportunidad de realizar consultas difusas mediante el lenguaje *FSQL (FUZZY SQL)*, que es una extensión del *SQL (Sequence Query Language)* tradicional. Esto no significa necesariamente que el usuario deba conocer dicho lenguaje, puesto que las consultas se realizarán a través de una interfaz visual, fácil de utilizar y que nos guiará paso a paso, por lo que se posibilita notablemente que cualquier persona pueda manejar el programa. También podremos realizar consultas directamente en FSQL para quien así lo desee.

Una vez realizada la consulta, la aplicación enviará al servidor de BD la sentencia y éste comprobará su validez, en cuyo caso responderá a la aplicación con el código equivalente generado en SQL. Este código SQL lo utilizará nuestra aplicación para realizar la consulta definitiva a la BD y el resultado de esta consulta se mostrará al usuario.

Hay que aclarar que la verdadera consulta SQL a la BD es la que se lleva a cabo al final. La primera consulta, se realiza en FSQL y la procesa una función del llamado Servidor FSQL que la analiza y la traduce a SQL de acuerdo a los elementos y estructuras definidas en el lenguaje FSQL.

El sistema FSQL para ORACLE y algunos de los artículos científicos publicados sobre este lenguaje pueden obtenerse de la dirección web http://www.lcc.uma.es/~ppgg/FSQL.

Métodos

Para la arquitectura de la aplicación se va a utilizar la metodología de las 3 capas recomendada para aplicaciones distribuidas: La capa de presentación, la capa de negocio y la capa de acceso a datos. La parte de presentación se desarrollará en componentes visuales a través de páginas web debido a la flexibilidad y potencia de este entorno para desarrollar aplicaciones visuales. En cuanto a las capas de negocio y acceso a datos utilizaremos componentes más específicos y que se ubicarán en el servidor. La aplicación tendrá una interfaz amigable para que cualquier usuario se pueda conectar a un servidor de Bases de Datos Difusas a través de Internet, y posteriormente por medio de una serie de opciones poder realizar consultas difusas, finalmente se procesará esta consulta y se mostrarán los resultados.

El sistema implementado se ha programado en *ASP (Active Server Pages)*, páginas activas de servidor, que se encargan de dar soporte a nivel de acceso a datos y gestión de páginas dinámicas, mientras que *Javascript* se ha utilizado en la interfaz de usuario para la funcionalidad en el cliente del navegador. Como servidor de internet se ha utilizado *Internet Information Server (IIS)*.

Por su parte, en el servidor se utilizará el modelo teórico *GEFRED*, que es un modelo de Bases de Datos Relacionales Difusas. La implementación de este modelo se encargará de gestionar las consultas *FSQL* y traducirlas a SQL (mediante una función implementada en el servidor y llamada desde el cliente). Todo esto funcionará bajo el *SGBD ORACLE* que será el encargado de atender las consultas definitivas en SQL. La aplicación irá en un servidor web, mientras que la BD puede ir en el mismo servidor ó bien utilizar otro remoto.

Todo el software de desarrollo y gestión se instalará bajo entorno Windows 2000, puesto que es uno de los más utilizados para aplicaciones de este tipo. De todas formas esta aplicación se puede instalar en cualquier sistema operativo con soporte de páginas web (p.ej. Windows XP y Windows 2003 Server).

Para acceder a la aplicación se podrán utilizar el *Microsoft Internet Explorer 5.5* ó superior o bien el *Netscape Navigator*, aunque con esté último no tendremos toda la funcionalidad ni optimización que con el navegador de Microsoft.

Las distintas fases de trabajo para el desarrollo del proyecto han sido las siguientes:

- Recopilar documentación y bibliografía de los sistemas y entornos a utilizar, así como elegir el software de desarrollo más adecuado para el proyecto.
- Instalar el sistema Windows 2000, el Servidor de Bases de Datos ORACLE, el servidor web y los entornos de programación a utilizar (Macromedia Dreamweaver MX y Microsoft Visual Studio 6.0).
- Utilizar un primer prototipo para comprobar las comunicaciones a través de Internet, el servidor web, las consultas SQL, las consultas FSQL en línea de comandos, etc.
- Desarrollar la aplicación del programa Cliente Visual FSQL.
- Añadir la ayuda a la aplicación.
- Documentación del programa y el manual de usuario.
- Memoria del proyecto.

Medios materiales

Para la realización del proyecto se han utilizado los siguientes medios materiales.

- Hardware:
  - ✓ Ordenador PC (Pentium IV 1.8GHz., 512Mb RAM, 80Gb Disco Duro).
  - ✓ Impresora de inyección de tinta (Canon BJC-2100).

- Software:
  - ✓ Microsoft Windows 2000.
  - ✓ Macromedia Dreamweaver MX.
  - ✓ Microsoft Visual Studio.
  - ✓ ORACLE 8.1.7
  - ✓ ODBC de Oracle.
  - ✓ Servidor FSQL 1.2.

La memoria está organizada en los siguientes puntos:

- Capítulo 1: Aquí se hace una breve introducción a la Lógica Difusa.
- Capítulo 2: En este capítulo se especifica la arquitectura utilizada para la Base de Datos Relacional Difusa y se explica el Servidor FSQL.
- Capítulo 3: Se muestra una visión general del desarrollo de aplicaciones web, los servidores y lenguajes utilizados, y su funcionalidad a través de Internet.
- Capítulo 4: En este capítulo se muestran las principales tecnologías existentes para acceder a Bases de Datos a través de internet.
- Capítulo 5: Se describe la arquitectura y desarrollo de la aplicación cliente Visual FSQL.
- Capítulo 6: Explicación muy detallada de la instalación del sistema en un entorno operativo.
- Capítulo 7: Manual de Usuario de Visual FSQL.
- Apéndice A. Mensajes de Error en la conexión de Oracle.
- Conclusiones y Líneas Futuras.
- Bibliografía.
- Índices de Figuras y Tablas.

Capítulo 1

Introducción a la Lógica Difusa

En el presente capítulo se trata de exponer una breve introducción a la Lógica Difusa.

1.1. Significado y Origen

Antes de comenzar describiendo brevemente la lógica difusa y su origen preguntémonos ¿qué significa *fuzzy*?; término sobre el cual se sustenta una forma de expresar las leyes, modos y formas del conocimiento científico (lógica).

Originalmente el término *fuzzy* procede de *fuzz*, que sirve para denominar la pelusa que recubre el cuerpo de los polluelos al poco de salir del huevo. Este término en inglés significa "confuso, borroso, no definido o desenfocado". Este término se traduce por "*flou"* en francés y se pronuncia "*aimai"* en japonés. La traducción de esta palabra al castellano es difuso o borroso, aunque *fuzzy*, en los ámbitos académico y tecnológico, está aceptado tal cual, de forma similar a como los es *"bit"*. *Fuzzy* significa ambiguo o vago, en el sentido del razonamiento humano, más que en la acepción de probabilidad de algo.

La lógica difusa nació cuando el Profesor Lotfi A.Zadeh publicó un artículo titulado *"Fuzzy Sets"* (Conjuntos Difusos) [24]. En este artículo el Dr. Zadeh presentó unos conjuntos sin límites precisos los cuales, según él, juegan un importante papel en el reconocimiento de formas, interpretación de significados, y especialmente abstracción, la esencia del proceso de razonamiento del ser humano.

En la lógica clásica sólo es posible tratar información que sea totalmente cierta o totalmente falsa; no le es posible manipular aquella información imprecisa o incompleta inherente a un problema y como información que es contiene datos que permitirían una mejor

resolución del mismo. Con ello se podría decir que la lógica difusa es una extensión de los sistemas clásicos. La lógica difusa es la lógica que soporta modos de razonamiento aproximados en lugar de exactos. Su importancia radica en que muchos modos de razonamiento humano, en especial el razonamiento según el sentido común, son aproximados por naturaleza.

Esta lógica es una lógica multievaluada, 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.

La lógica difusa se ha convertido en un tema muy común en control de máquinas como el resultado de hacerlas más "capaces" y "responsables". Se podría decir que la lógica difusa permite a los ordenadores trabajar no sólo con métodos cuantitativos sino también cualitativos, se trata pues de un intento de aplicar una forma más humana de pensar en la programación de computadoras.

1.2. Teoría de Conjuntos Difusos

En este apartado se va a introducir algunas nociones elementales sobre la teoría de conjuntos difusos, así como la notación utilizada al respecto a lo largo de esta memoria. En este resumen nos detendremos en los aspectos semánticos y de representación relacionadas con esta potente herramienta. En la literatura podemos encontrar una gran cantidad de trabajos sobre esta teoría, como en [11] donde se puede encontrar una recopilación de algunos de los artículos más interesantes publicados sobre el tema por L.A. Zadeh. En [25] es posible

encontrar recopilados los aspectos más importantes que constituyen la teoría de conjuntos difusos así como la teoría de la posibilidad.

1.2.1. Introducción a los Conjuntos Difusos (Fuzzy-Sets)

Los Conjuntos Difusos son una generalización de los (sub)conjuntos clásicos en el sentido de que los amplían pues permiten la descripción de nociones "vagas" e "imprecisas". Relajan la restricción de los conjuntos clásicos de pertenencia o no-pertenencia absoluta al mismo. Es necesario hacer notar que muchos de estos conceptos con naturaleza "imprecisa", si no todos, responden a criterios subjetivos. Esto es, la definición de esa imprecisión depende en mayor o menor medida de la persona que la expresa. Dicha generalización nos lleva a que:

- La pertenencia de un elemento a un conjunto pasa a ser un concepto *"difuso o borroso"*. Para algunos elementos puede no estar clara su pertenencia o no al conjunto.
- Dicha pertenencia puede ser cuantificada por un grado. Dicho grado se denomina habitualmente como **"grado de pertenencia"** de dicho elemento al conjunto y toma un valor en el intervalo $[0,1]\in\Re$ por convenio.

  - ✓ Nótese la gran potencialidad que presenta este último punto al permitir expresar de forma cuantitativa algo cualitativo (difuso) mediante el grado de pertenencia. De forma más formal podemos definir un conjunto difuso como sigue:

Definición 1.1 Un **Conjunto Difuso** A sobre un universo de discurso Ω (intervalo finito o infinito dentro del cual el conjunto difuso puede tomar un valor) es un conjunto de pares

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

donde $\mu_A(x)$ se denomina **grado de pertenencia** del elemento x al conjunto difuso A. Este grado oscila entre los extremos **0** y **1** del dominio de los nº reales:

$\mu_A(x) = 0$ indica que x no pertenece en absoluto al conjunto difuso A.

$\mu A(x) = 1$ indica que x pertenece totalmente al conjunto difuso A**.**

A veces, en vez de dar una lista exhaustiva de todos los pares que forman el conjunto (valores discretos), se da una definición para la función $\mu A(x)$, llamada función característica o **función de pertenencia.**

Si la función de pertenencia sólo produce valores del conjunto {0,1}, entonces el conjunto que genera no es difuso, sino "**Crisp**" (puede traducirse como un valor concreto, preciso).

De la definición de conjunto difuso se derivan los siguientes conceptos:

- **Universo de Discurso:** Como se ha mencionado anteriormente existen dos posibilidades a la hora de establecer el intervalo de valores válido para el conjunto difuso y son:

  A través de un universo de discurso *finito o discreto:*

  $$\Omega = \{x_1, x_2, \ldots, x_n\}$$

  un conjunto difuso A se puede representar como:

  $$A = \mu_1 / x_1 + \mu_2 / x_2 + \ldots + \mu_n / x_n$$

  donde μ_i representa el grado de pertenencia del elemento x_i, con i=1,2,..,n. Habitualmente los elementos con grado cero no se listan. Aquí la suma no hace el papel de la suma aritmética sino que tiene el sentido de agregación y la / no es el operador de división sino que tiene el significado de asociación de ambos valores.

  Y al expresar el conjunto difuso a través de su función de pertenencia en un universo de discurso *infinito*, así un conjunto difuso A sobre Ω puede representarse como:

  $$A = \int \mu A(x) / x$$

- **Etiqueta Lingüística:** Es aquella palabra, en lenguaje natural, que expresa o *identifica a un conjunto difuso*, que puede estar formalmente definido o no. Así la función de pertenencia de un conjunto difuso A, $\mu_A(x)$, expresa el grado en que x verifica la categoría especificada por A.

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

Ejemplo: Para ilustrar lo mencionado anteriormente tomemos como ejemplo el siguiente caso:

Si expresamos el concepto cualitativo "joven" mediante un conjunto difuso, donde el eje de abcisas (eje X) representa el universo de discurso *edad* y el eje de ordenadas (eje Y) representa los grados de pertenencia en el intervalo [0,1]. El conjunto difuso que representa dicho concepto podría expresarse en la forma siguiente, considerando un universo discreto:

Joven = {1/0,..., 1/20, 1/25, 0.9/26, 0.8/27, 0.7/28, 0.6/29, 0.5/30,..., 0.1/34}

La "edad" (en años enteros) sería el universo de discurso de "joven". La etiqueta lingüística "joven" identificaría a este conjunto difuso representado por una función de pertenencia si consideramos un universo de discurso no discreto, de otros como "adulto", "viejo"... y así:

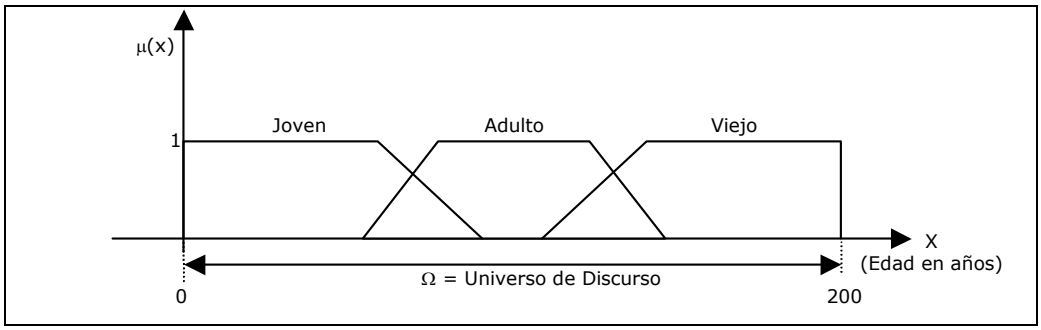


Figura 1.1. Gráfico que ilustra tres etiquetas lingüísticas.

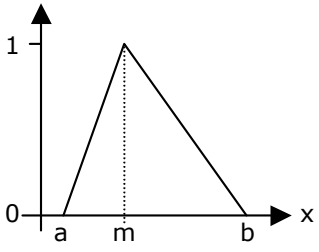
1.2.2. Tipos de Funciones de Pertenencia

Según la forma de la función de pertenencia, se tendrá distintas clases de conjuntos difusos. Zadeh propuso una serie de funciones de pertenencia que se podrían clasificar en dos grupos, las formadas por líneas rectas "lineales" y las que presentan formas gaussianas, es decir, "curvas".

A continuación se comentan estos tipos de funciones de pertenencia propuestos por Zadeh, estos tipos de conjuntos difusos son los denominados conjuntos difusos *convexos* en la teoría de conjuntos difusos, finalizando con la aportación por nuestra parte de la función *trapecio extendido* la cual podríamos clasificar como una función lineal *no convexa*, es decir, una función que alternativamente es creciente y decreciente en su dominio.

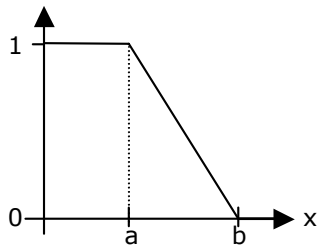
1) **<u>Triangular:</u>** Definido por sus límites inferior **a** y superior **b**, y el valor modal **m**, tal que **a<m<b.**

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



2) <u>Función L:</u> Se trata de una función definida por dos parámetros de la siguiente forma:

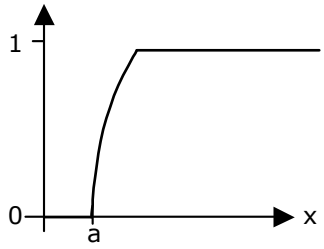
$$L(x) = \begin{cases} 1 & \text{si } x \le a \\ \dfrac{a-x}{b-a} & \text{si } a < x \le b \\ 0 & \text{si } x > b \end{cases}$$



3) <u>Función Γ (Gamma):</u> Definida por su límite inferior **a** y el valor **k>0**. Dos definiciones:

$$\Gamma(x) = \begin{cases} 0 & \text{si } x \le a \\ 1 - e^{-k(x-a)^2} & \text{si } x > a \end{cases}$$

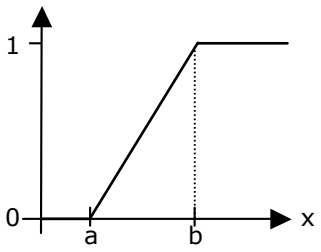
$$\Gamma(x) = \begin{cases} 0 & \text{si } x \le a \\ \dfrac{k(x-a)^2}{1+k(x-a)^2} & \text{si } x > a \end{cases}$$



- Esta función se caracteriza por un rápido crecimiento a partir de **a**.
- Cuanto mayor es el valor de **k**, el crecimiento es más rápido aún.
- La primera definición tiene un crecimiento más rápido que la segunda.
- Asíntota horizontal en 1.

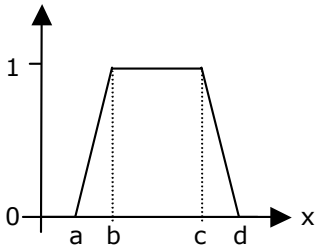
De forma lineal también se expresa la función gamma como:

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



4) **Función Trapezoidal:** Definida por sus límites inferior **a** y superior **d**, y los límites de su núcleo, **b** y **c**, inferior y superior respectivamente.

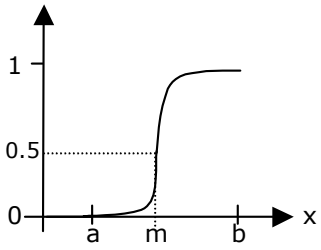
$$T(x) = \begin{cases} 0 & \text{si } (x \le a) \text{ o } (x \ge d) \\ (x-a)/(b-a) & \text{si } x \in (a,b] \\ 1 & \text{si } x \in (b,c) \\ (d-x)/(d-c) & \text{si } x \in (b,d) \end{cases}$$



5) **Función S:** Definida por sus límites inferior **a** y superior **b**, y el valor **m**, o punto de inflexión tal que **a<m<b**.

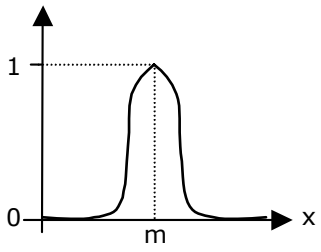
- Un valor típico es: **m=(a+b) / 2.**
- El crecimiento es más lento cuanto mayor sea la distancia **a-b**.

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



6) **Función Gaussiana:** Definida por su valor medio **m** y el valor de **k>0**.

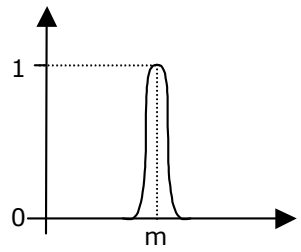
$$G(x) = e^{-k(x-m)^2}$$



- Es la típica campana de Gauss.
- Cuanto mayor es **k**, más estrecha es la campana.

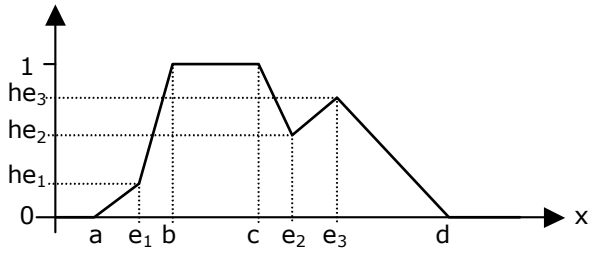
7) **<u>Función Pseudo-Exponencial:</u>** Definida pro su valor medio **m** y el valor **k>1**.

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



- Cuanto mayor es el valor de **k**, el crecimiento es más rápido aún y la "campana" es más estrecha.

8) **<u>Función Trapecio Extendido:</u>** Definida por los cuatro valores de un trapecio **[a,b,c,d]**, y una lista de puntos entre **a** y **b** y/o entre **c** y **d**, con su valor de pertenencia asociado a cada uno de esos puntos (e_i,he_i).



Observaciones:

- En general, la función **Trapezoidal** se adapta bastante bien a la definición de cualquier concepto, con la ventaja de su fácil definición, representación y simplicidad de cálculos.
- En casos particulares, el **Trapecio Extendido** puede ser de gran utilidad. Éste permite gran expresividad aumentando su complejidad.
- En general, usar **una función más compleja no añade mayor precisión**, pues debemos recordar que se está definiendo un concepto **difuso**.

En control difuso, se busca expresar las nociones de "incremento", "decremento" y "aproximación" y para ello se utilizan los tipos de funciones de pertenencia comentados

anteriormente. Las funciones de pertenencia Γ, S se usan para representar etiquetas lingüísticas como "alto", "caliente"" en el dominio de la altura y la temperatura. Etiquetas lingüísticas como "pequeño" y "frío" se expresarían a través de la función L. Por otro lado la noción de aproximación a veces resulta difícil de expresar con una palabra, en el dominio de la temperatura debería ser "confortable"; lo cual se expresaría a través de las funciones triángulo, trapecio, gaussiana,...

1.2.3. Cálculo de la Función de Pertenencia

Las funciones de pertenencia pueden calcularse de diversas formas. El método a elegir depende de la aplicación en particular, del modo en que se manifieste la incertidumbre y en el que ésta sea medida durante los experimentos.

1) Método Horizontal:

- Se basa en las respuestas de un grupo de **N** "expertos".
- La pregunta tiene el formato: "¿Puede x ser considerado compatible con el concepto A?".
- Sólo se acepta un "SÍ" o un "NO", de forma que: $A(x)=(Respuestas\ Afirmativas)/N$.

2) Método Vertical:

- Se escogen varios valores para α, para construir sus α-cortes.
- Ahora la pregunta es la siguiente, efectuado esos valores de α predeterminados: "¿Identifique los elementos de X que pertenecen a A con grado no menor que α?".
- A partir de esos α-cortes se identifica el conjunto difuso A (usando el llamado Principio de Identidad o Teorema de Representación que se verá más adelante).

3) Método de Comparación de Parejas [Saaty80]:

- Suponemos que tenemos ya el conjunto difuso **A**, sobre el Universo **X** de **n** valores **$(x_1, x_2, \ldots, x_n)$**.
- Se calcula la **Matriz Recíproca $M=[a_{hi}]$**, matriz cuadrada **n x n** :

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

  - ✓ La diagonal principal es siempre 1.
  - ✓ Propiedad de Reciprocidad: **$(a_{hi}, a_{ih})=1$**
  - ✓ Propiedad Transitiva: **$(a_{hi}, a_{ik})=a_{hk}$**

- El proceso es el inverso:
  - ✓ Se calcula la matriz **M**.
  - ✓ Se calcula **A** a partir de **M**.

- Para calcular **M**, se cuantifica numéricamente el nivel de prioridad o mayor pertenencia de una pareja de valores: **x_i** con respecto a **x_j**.
  - ✓ El número de comparaciones: **n(n-1)/2**
  - ✓ La transitividad es difícil de conseguir (el autovalor de la matriz sirve para medir la consistencia de los datos, de forma que si es muy bajo, deberían repetirse los experimentos).

4) Método basado en la Especificación del Problema:

- Requieren una función numérica que quiera ser aproximada.
- El error se define como un conjunto difuso: Mide la calidad de la aproximación.

5) Método basado en la Optimización de Parámetros:

- La forma de un conjunto difuso A, depende de unos parámetros, denotados por el vector **p**: Representándolo por **A(x; p).**
- Se obtiene algunos resultados experimentales, en la forma de parejas (elemento, grado de pertenencia): **(E_k, G_k)** con **k=1,2, ..., N.**
- El problema consiste en optimizar el vector **p**, por ejemplo minimizando el error cuadrático: $\min_p \sum_{k=1}^{N} [G_K - A(E_K; p)]^2$

6) Método basado en la Agrupación Difusa

- Se trata de agrupar los objetos del Universo en grupos (solapados) cuyos niveles de pertenencia a cada grupo son vistos como grados difusos.
- Existen varios algoritmos de *Fuzzy Clustering*, pero el más aceptado es el algoritmo de *"fuzzy isodata"* [3].

Algoritmo "Fuzzy Isodata"

- Supongamos **N** elementos **(x_1, x_2, ... , x_n)**, entre los que existe una medida de **distancia** entre cada dos elementos: $\| x_i - x_j \|$.
- Crear una matriz **F=[f_{ij}]**, de **c** filas y **N** columnas, donde $f_{ij} \in$ **[0,1]**, denota el grado de pertenencia de x_j al grupo i-ésimo y se **cumple** que:

$$\forall j = 1,2,\ldots,N : \sum_{i=1}^{c} f_{ij} = 1, \text{ y que } \forall i = 1,2,\ldots,c : \sum_{j=1}^{N} f_{ij} \in (0, N).$$

- Fila **i**: Grados de pertenencia de los **N** elementos al grupo **i**-ésimo.
- Algoritmo:

  - ✓ **1. K:=0;** Hallar una matriz inicial **F(0)**.
  - ✓ **2.** Usando **F(k)**, calcular los centroides $v_i(k) = \dfrac{\sum_{j=1}^{N} f^2_{ij}(k) x_j}{\sum_{j=1}^{N} f^2_{ij}(k)}$

- ✓ **3.** Calcular **F(k+1)**: $(f_{ij}(k+1))^{-1} = \sum_{h=1}^{c} \left(\frac{||x_j - v_i||}{||x_j - v_h||} \right)^2$

- ✓ **4.** Comparar **F(k)** con **F(k+1)**: Si son suficientemente parecidos, **PARAR**. en otro caso, **k:=k+1**; Ir al paso **2**.

- Obtenemos soluciones locales a la siguiente optimización no lineal, **cumpliendo** la matriz $[f_{ij}]$ las condiciones anteriores:

$$\min_{v_i f_{ij}} \sum_{j=1}^{N} \sum_{i=1}^{c} f_{ij}^{2} ||x_j - v_i||^2$$

1.2.4. Conceptos sobre Conjuntos Difusos

Sobre conjuntos difusos se definen una serie de conceptos que nos permiten tratar y comparar conjuntos difusos:

- **Igualdad** ***(Equality)*** de conjuntos difusos sobre un mismo universo de discurso:

  Definición 1.2 Dos conjuntos difusos A y B sobre Ω se dicen iguales si cumplen:

$$A = B \Leftrightarrow \forall x \in \Omega,\ \mu_A(x) = \mu_B(x)$$

- **Inclusión** ***(Inclusión)*** de un conjunto difuso en otro:

  Definición 1.3 Dados dos conjuntos difusos A y B sobre Ω, decimos que A está incluido en B si cumplen:

$$A \subseteq B \Leftrightarrow \forall x \in \Omega,\ \mu_A(x) \leq \mu_B(x)$$

- **Soporte** de un conjunto difuso:

  Definición 1.4 El soporte (support) de un conjunto difuso A definido sobre Ω es un subconjunto de dicho universo que satisface:

$$\text{supp}(A) = \{x \in \Omega,\ \mu_A(x) > 0\}$$

- El **α-corte** de un conjunto difuso:

  Definición 1.5 Denotándolo por A_α, es un subconjunto no difuso (clásico) de elementos de Ω, cuya función de pertenencia toma un valor mayor o igual que algún valor concreto α de dicho universo de discurso que satisface:

$$A_\alpha = \{x : x \in \Omega,\ \mu A(x) \geq \alpha, \alpha \in [0,1]\}$$

  El Teorema de Representación permite representar cualquier conjunto difuso A mediante la unión de sus α-cortes. La Figura 1.2 ilustra el concepto de α-corte:

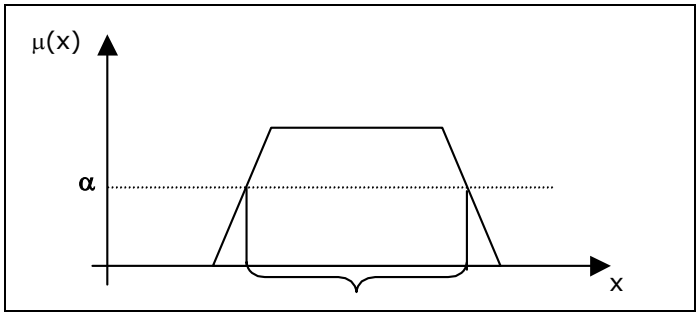


Figura 1.2. α-corte en un trapecio.

- **Teorema de Representación:**

  Definición 1.6 Todo subconjunto difuso A puede ser obtenido a partir de la unión de sus α-cortes:

$$A = \bigcup_{\alpha \in [0,1]} A_\alpha$$

- **Conjunto Difuso Convexo o Cóncavo:**

  Definición 1.6 Haciendo uso del Teorema de Representación se establece el concepto de conjunto difuso como aquel en que todos sus α-cortes son convexos:

$$\forall x, y \in \Omega, \forall \lambda \in [0,1]: \mu A(\lambda \cdot x + (1 - \lambda) \cdot y) \geq \min(\mu A(x), \mu A(y))$$

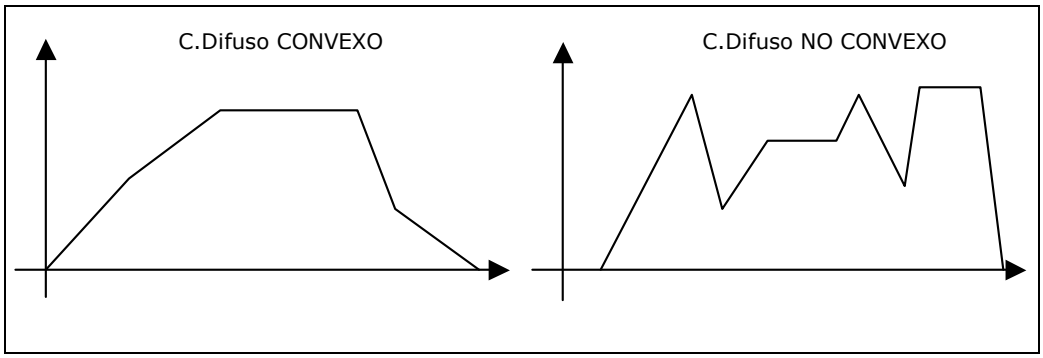


Figura 1.3. Ejemplos de conjuntos difusos convexos y no convexos.

Por último, indicar que si dos conjuntos difusos son convexos, también lo es su intersección.

- **Núcleo** *(Core)*:

  Definición 1.7 El núcleo de un conjunto difuso A, definido sobre Ω es un subconjunto de dicho universo que satisface:

$$\text{Kern}(A) = \{x \in \Omega,\ \mu_A(x) = 1\}$$

- **Altura** *(Height)*:

  Definición 1.8 La altura de un conjunto difuso A, definido sobre Ω se define como:

$$\text{Hgt}(A) = \sup_{x \in \Omega} \mu_A(x)$$

- Conjunto difuso **Normalizado:**

  Definición 1.9 Un Conjunto Difuso es normalizado sí y sólo sí:

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

1.2.5. Operaciones sobre Conjuntos Difusos

El hecho de que la teoría de conjuntos difusos generalice la teoría de conjuntos clásica, da lugar a que los conjuntos difusos admitan las operaciones de unión, intersección y complemento. En [14] podemos encontrar estas y otras operaciones, como la *concentración* (elevar al cuadrado la función de pertenencia), la *dilatación* (efectuar la raíz cuadrada de la

función de pertenencia) y la *intensificación* que pueden utilizarse cuando se usan modificadores lingüísticos (linguistic hedges) como "muy" o "poco".

1.2.5.1. Unión e Intersección

Definición 1.10 Si A y B son dos conjuntos difusos sobre un universo de discurso Ω, la función de pertenencia de la unión de ambos conjuntos, $A \cup B$, viene dada por

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

donde f es una T-conorma [15].

Definición 1.11 Si A y B son dos conjuntos difusos sobre un universo de discurso Ω, la función de pertenencia de la intersección de ambos conjuntos, $A \cap B$, viene dada por

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

donde g es una T-norma [15].

Las definiciones anteriores no son únicas ya que existen varios operadores que satisfacen el concepto de T-norma y de T-conorma, como las presentadas en [16]. Los mas importantes son:

- Operadores *idempotentes:* El **máximo** y el **mínimo** para la unión y la intersección respectivamente. Satisfacen, además de la idempotencia, la propiedad distributiva aplicada sobre ambos y son estrictamente crecientes. Estos operadores son los más utilizados por que conservan gran cantidad de las propiedades de los operadores booleanos.

- Operadores *arquimedianos:* Emplean la **suma probabilística**, (x+y-x*y), y el **producto**, (x*y), para la unión y la intersección, respectivamente. Estos operadores no satisfacen la propiedad distributiva ni son idempotentes.

- Operadores *acotados (bounded):* Los operadores dados por, min(1,x+y) y max(0,x+y-1), representan la unión y la intersección respectivamente. Estos operadores no satisfacen la idempotencia, la propiedad distributiva ni la propiedad de absorción. Por el contrario, satisfacen las propiedades conmutativa, asociativa y de identidad.

1.2.5.2. Complemento o Negación

La noción de complemento se puede construir a partir del concepto de negación fuerte de E.Trillas [17]:

> **Definición 1.12** Una función C de [0,1] en [0,1] es una **negación fuerte** si satisface las siguientes condiciones:
>
> A. C(0)=1
> B. C(C(a))=a (involución)
> C. C es estrictamente decreciente
> D. C es continua

Aunque existen varios tipos de operadores que satisfacen tales propiedades o versiones relajadas de las mismas, nosotros, para el complemento, emplearemos principalmente la versión proporcionada por Zadeh en [10], en el cual:

$$C(x) = 1 - x$$

Por tanto, para un conjunto difuso A sobre un universo de discurso Ω, la función de pertenencia del **complemento** de A, $\neg A$, viene dada por:

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

1.2.6. Números Difusos

El concepto de número difuso fue introducido por primera vez en [10] con el propósito de analizar y manipular valores numéricos aproximados, por ejemplo: “próximo a 0”, “casi

5", etc. El concepto ha sido refinado sucesivamente y en esta memoria entenderemos por número difuso lo siguiente:

Definición 1.13 Sea A un subconjunto difuso de Ω y $\mu_A(x)$ su función de pertenencia cumpliendo:

$\forall x, y \in \Omega, \forall \mu_A(t) \geq \min(\mu_A(x), \mu_A(y))$, es decir, que es CONVEXO.

$\mu_A(x)$ es semicontinua superiormente.

El soporte de A es un conjunto acotado.

entonces diremos que A es un **número difuso**.

En la Figura 1.4 podemos ver la representación gráfica de un número difuso.

Algunos autores incluyen en la definición la necesidad de que el subconjunto difuso esté normalizado.

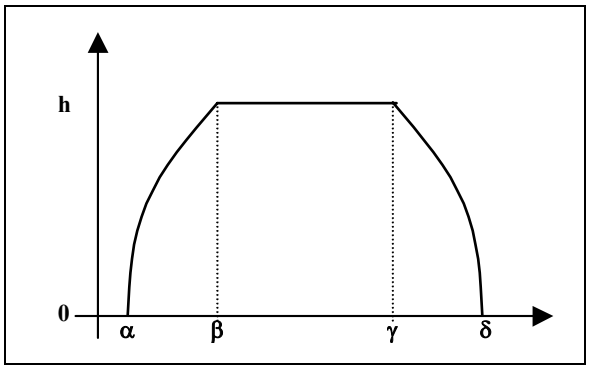


Figura 1.4. Número difuso general.

La forma general de la función de pertenencia de un número difuso A, es la siguiente:

$$\mu_A(x) = \begin{cases} r_A(x) & \text{si } x \in [\alpha, \beta) \\ h & \text{si } x \in [\beta, \gamma] \\ s_A(x) & \text{si } x \in (\gamma, \delta] \\ 0 & \text{en otro caso} \end{cases}$$

donde r_A, s_A:$\Omega \rightarrow [0,1]$, r_M no decreciente, s_M no creciente y

$r_A(\beta) = h = s_A(\gamma)$

con $h \in (0,1]$ y $\alpha,\beta,\gamma,\delta \in \Omega$.

Al número h se le denomina *altura* del número difuso, al intervalo $[\beta,\gamma]$ *intervalo modal* y a los números β-α y δ-γ *holguras izquierda y derecha* respectivamente.

A lo largo de esta memoria utilizaremos a menudo un caso particular de números difusos que se obtiene cuando consideramos a las funciones r_A y s_A como funciones lineales. En este caso la función de pertenencia adopta la forma:

$$\mu_A(x) = \begin{cases} h + \dfrac{(x-\beta)h}{\beta-\alpha} & \text{si } x \in [\alpha,\beta) \\ h & \text{si } x \in [\beta,\gamma] \\ h - \dfrac{(x-\gamma)h}{\delta-\gamma} & \text{si } x \in (\gamma,\delta] \\ 0 & \text{en otro caso} \end{cases}$$

A un número difuso de este tipo lo llamaremos *triangular o* ***trapezoidal.*** Usualmente trabajaremos con números difusos normalizados por lo que h=1, en este caso podremos caracterizar un número difuso trapezoidal normalizado A, mediante el empleo de los 4 parámetros que son realmente los imprescindibles (ver Figura 1.5):

$A \equiv (\alpha,\beta,\gamma,\delta)$

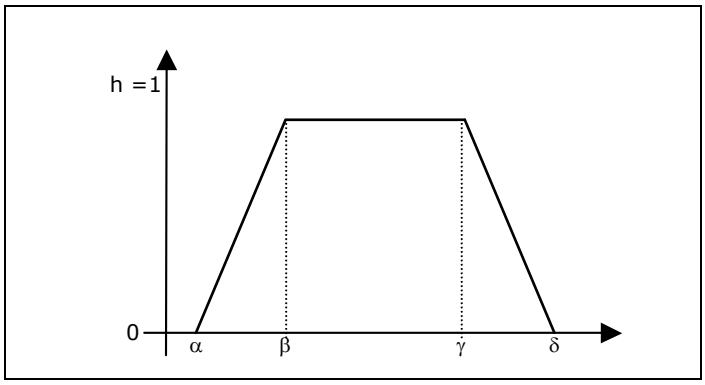


Figura 1.5. Número difuso trapezoidal normalizado.

1.2.6.1. El principio de Extensión (*Extension Principle*)

Una de las nociones más importantes en teoría de conjuntos difusos es el *principio de extensión*, propuesto en [9]. Proporciona un método general que permite extender conceptos matemáticos no difusos para el tratamiento de cantidades difusas. Se usa para transformar cantidades difusas, que tengan iguales o distintos universos, según una función de transformación en esos universos.

Sea A una cantidad difusa definida sobre el universo de discurso X y *f* una función de transformación no difusa tal que *f: X* $\rightarrow$ Y. El propósito es extender *f* en A de forma que opere sobre X y devuelva una cantidad difusa B sobre Y. Dicho objetivo se obtiene por uso de la composición del *Sup-Min* como a continuación se comenta de forma generalizada en el caso del producto cartesiano de *n* universos de discursos.

En la Figura 1.6 se puede observar la representación gráfica del *Principio de Extensión*.

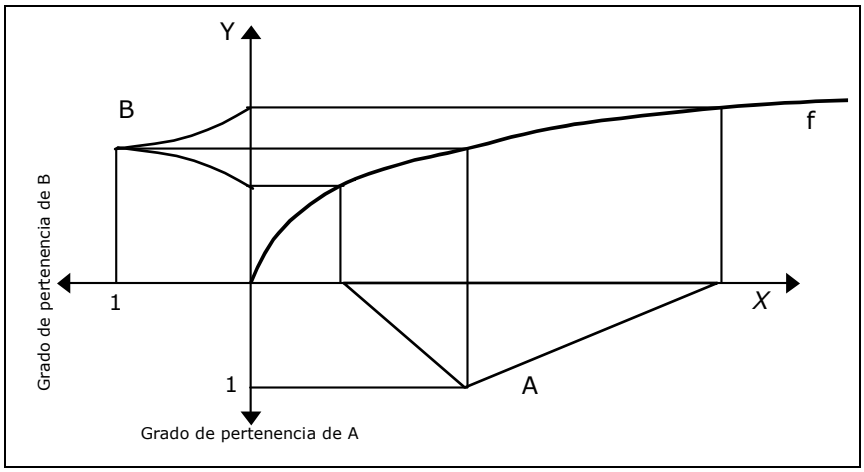


Figura 1.6. Representación gráfica del principio de extensión.

Definición 1.14 Sea Ω un producto cartesiano de universos tal que $\Omega = \Omega_1 \text{ X } \Omega_2 \text{ X...X } \Omega_n$, y $A_1, A_2, \ldots, A_n$, n conjuntos difusos de $\Omega_1, \Omega_2, \ldots, \Omega_n$ respectivamente, f una función desde Ω al universo Ω', entonces un conjunto difuso B de Ω' viene definido por:

$$B = \int_{\Omega'} \mu_B(y) / y$$

donde $y = f(A_1, A_2, \ldots, A_n)$ $(y \in \Omega')$, y

$$\mu_B(y) = \sup_{\substack{\Omega 1,\ldots,\Omega n \\ f(\Omega 1,\ldots,\Omega n)=\Omega'}} \min(\mu_{A_1}(x_1),\ldots,\mu_{A_n}(x_n))$$

si $f^{-1}(y) \neq 0$.

en otro caso $\mu_B(y) = 0$.

Ejemplo X.Y Sean X e Y, ambos, el universo de los números naturales.

- Función sumar 4: *y=f(x)=x+4:*
  - ✓ A=0.1/2+0.4/3+1/4+0.6/5;
  - ✓ B=f(A)=0.1/6+0.4/7+1/8+0.6/9;

- Función suma: $y=f(x_1,x_2)=x_1+x_2$:
  - ✓ A_1=0.1/2+0.4/3+1/4+0.6/5;
  - ✓ A_2=0.4/5+1/6;
  - ✓ B=f(A_1, A_2)=0.1/7+0.4/8+0.4/9+1/10+0.6/11;

1.2.6.2. Aritmética Difusa

Gracias al Principio de Extensión es posible extender las operaciones aritméticas clásicas al tratamiento de números difusos. De esta forma las cuatro operaciones principales quedan extendidas en:

- **Suma Extendida:** Dadas dos cantidades difusas A_1 y A_2, la función de pertenencia de la suma viene dada por la expresión:

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

De este modo, la suma, queda expresada en términos de la operación del supremo. La suma extendida es una operación conmutativa, asociativa y no existe el concepto de número simétrico.

- **Diferencia Extendida:** Dadas dos cantidades difusas A_1 y A_2, la función de pertenencia de la suma viene dada por la expresión:

$$\mu_{A_1 - A_2}(y) = \text{Sup}\{\text{Min}(\mu_{A_1}(y + x), \mu_{A_2}(x)) / x \in \Omega\}$$

De estas definiciones se puede obtener fácilmente que si A_1 tiene *n* términos y A_2 tiene *m* términos, el número de términos de A_1+A_2 y de A_1-A_2 es (n-1)+(m-1)+1, o lo que es lo mismo: n+m-1.

- **Producto Extendido:** El producto de dos cantidades difusas $A_1 * A_2$ se obtiene:

$$\mu_{A_1 \bullet A_2}(y) = \begin{cases} \sup\{\min(\mu_{A_1}(z/y), \mu_{A_2}(y)) / y \in \Omega - \{0\}\} & \text{si } z \neq 0 \\ \max(\mu_{A_1}(0), \mu_{A_2}(0) & \text{si } z = 0 \end{cases}$$

- **División Extendida:** La división de dos cantidades difusas se define mediante:

$$\mu_{A_1 \div A_2}(y) = \text{Sup}\{\text{Min}(\mu_{A_1}(y.z), \mu_{A_2}(y)) / y \in \Omega\}$$

Basándose en una expresión particular del principio de incertidumbre adaptada al empleo de α-cortes y en un tipo de número difuso similar al descrito anteriormente, denominado número L-R, en [16] se describen fórmulas de cálculo rápido para las anteriores operaciones aritméticas.

1.2.7. Teoría de la Posibilidad

Esta teoría se basa en la idea de variables lingüísticas y cómo estas están relacionadas con los conjuntos difusos [10]. Así, se puede evaluar la *posibilidad* de que una determinada variable X sea o pertenezca a un determinado conjunto A, como el grado de pertenencia de los elementos de X en A.

Definición Sea un conjunto difuso A definido sobre Ω con su función de pertenencia $\mu_A(x)$ y una variable X sobre Ω (que desconocemos su valor). Entonces, la proposición "X es A" define una **Distribución de Posibilidad**, de forma que se dice que la "posibilidad" de que X = u vale $\mu_A(u)$, para todo valor $u \in \Omega$.

Capítulo 2

Arquitectura de la BDRD: El Servidor FSQL (Fuzzy SQL)

Vamos a ver como la arquitectura de la BDRD (Base de Datos Relacional Difusa) muestra los elementos básicos con los que está construida y cómo se relacionan unos con otros. Esos elementos son de distinta naturaleza. Estructuras de datos (tablas, vistas...), gramáticas para la definición de DML (Data Manipulation Language) y DDL (Data Definition Language), programas (en distintos lenguajes), datos...

El objetivo de esta arquitectura es, como ya se ha indicado, poder almacenar y tratar información imprecisa, difusa, y para ello necesitamos formalizar un método de hacerlo. Es decir, es absolutamente imprescindible definir cómo se van a almacenar estos nuevos tipos de información y cómo es posible comunicarse con la BDRD para conseguir nuestro propósito. Este canal de comunicación será una vía para que los usuarios, sea directamente o a través de aplicaciones específicas puedan trabajar con la BDRD. Para la construcción de ese canal hemos preferido modificar un canal ya existente y ampliamente empleado: El Lenguaje de Consulta Estructurado **SQL** (Structured Query Language).

Así pues, hemos modificado el lenguaje SQL para adaptarlo a las necesidades de una BDRD, de forma que permita expresar valores difusos, condiciones difusas, atributos difusos, grados de compatibilidad, umbrales de cumplimiento... naciendo asé el que hemos denominado SQL Difuso, Fuzzy SQL o FSQL.

Para que el lenguaje FSQL cobrara vida hemos creado un Servidor FSQL multiusuario, implantado de acuerdo con el modelo Cliente/Servidor, para uno de los SGBD (Sistemas Gestores de Bases de Datos) más potentes y difundidos en la actualidad: Oracle. El programa Cliente FSQL, que permite consultar esa BDRD de forma fácil, cómoda y flexible usando el

Servidor FSQL, será explicado en el siguiente Capítulo. Además del Servidor FSQL, hemos construido un programa Cliente FSQL, llamado VisualFSQL para acceso web.

En este capítulo explicaremos primero cómo hemos conseguido almacenar valores difusos partiendo de un SGDB clásico (Oracle) explicando cómo se ha implementado el modelo GEFRED [13] de BDRD. Esta es la base de nuestro Sistema de BDRD. A continuación daremos una definición de la sintaxis y la semántica del lenguaje FSQL, (tanto de sentencias DML como DDL). Posteriormente definiremos la arquitectura de nuestro modelo Cliente/Servidor de BDRD y cómo se ha programado el Servidor de Consultas FSQL (versión 1.2). Terminaremos el capítulo con una lista de las mejoras más importantes que pueden incorporarse tanto al lenguaje FSQL como a próximas versiones del Servidor FSQL.

2.1. Implementación de la BDRD: FIRST

En [12] se expuso un módulo para permitir extender la capacidad de un SGBDR clásico para que pueda representar y manipular información "imprecisa". Este módulo, llamado FIRST (Fuzzy Interface for Relational SysTems, Interface Difuso para Sistemas Relacionales), utiliza GEFRED [13] como modelo teórico y los recursos del modelo relacional clásico para poder representar este tipo de información.

Además, para poder efectuar las operaciones típicas sobre la BDRD (creación de tablas, de etiquetas, consultas flexibles...) hemos extendido el lenguaje SQL para que permita tratar los nuevos datos. Esta extensión la hemos llamado FSQL (Fuzzy SQL).

2.1.1. Esquema General de FIRST

La Figura 2.1 muestra el esquema general de FIRST. Como la idea de partida es la de construirlo sobre un gestor de bases de datos convencional, todos los desarrollos a realizar toman a dicho gestor como el elemento principal al que van dirigidas todas las peticiones. En breve, explicamos cada uno de esos módulos:

- **SGBDR** (Sistema Gestor de Bases de Datos Relacionales, Relational DabaBase

Management System, RDBMS): Todas las operaciones que hayamos concebido para la extensión difusa que representa nuestra implementación, se traducirán a peticiones al SGBDR anfitrión (en general en SQL). Las peticiones al SGBDR se realizarán empleando el lenguaje SQL o FSQL, dependiendo si la consulta involucra o no relaciones y/o condiciones difusas. Como veremos, las sentencias FSQL serán procesadas por el Servidor FSQL.

- **BD** (Base de Datos): Almacena, en formato relacional toda la información que sea de interés, igual que cualquier otra base de datos. La única diferencia es que nuestra base de datos permitirá el almacenamiento de información difusa en sus tablas. La forma en que se representan los datos en dichas tablas dependerá de la naturaleza de los mismos y se verá en los próximos apartados.
- **FMB** (Fuzzy Metaknowledge Base, Base de Metaconocimiento Difuso): El "diccionario o "catálogo del sistema" de un SGBDR representa aquella parte del sistema que almacena información sobre los datos recogidos en la base de datos, así como otro tipo de informaciones: Usuarios, permisos, accesos, datos de control... Dentro de este catálogo hemos incluido la FMB, que extiende esta parte del sistema a fin de que pueda recoger aquella información necesaria relacionada con la naturaleza "imprecisa" de la nueva colección de datos a procesar. Su organización y la información que almacena se verán más adelante.
- **Servidor FSQL**: Su objetivo es captar las sentencias en lenguaje FSQL y traducirlas a un lenguaje que entienda el SGBDR, el lenguaje SQL. Para efectuar esta traducción utilizará la información almacenada en la FMB. Este Servidor está íntegramente programado en el lenguaje PL/SQL de Oracle y se encuentra implantado como una colección de módulos almacenados en el Servidor Oracle. Nuestro Servidor FSQL podrá ser instalado en cualquier plataforma donde exista una implementación de Oracle.
- **Cliente FSQL**: Se trata de un programa que hace de interfaz entre el hombre (u otro programa) y el Servidor FSQL. Este programa puede ser muy simple, pues el trabajo principal de una operación FSQL será efectuado por el Servidor FSQL. El programa Cliente FSQL puede ser programado para cualquier Sistema Operativo y en cualquier lenguaje de programación. Nosotros hemos desarrollado un Cliente FSQL muy completo para poder editar y construir sentencias difusas visualmente (en modo asistente), sin la necesidad de conocer la sintaxis de FSQL ni la de SQL (siempre es

recomendable tener algunas nociones aunque sean de SQL), además este cliente da acceso a Bases de Datos difusas a través de Internet o Intranets, lo cual facilita mucho la accesibilidad y ámbito de aplicabilidad del lenguaje FSQL.

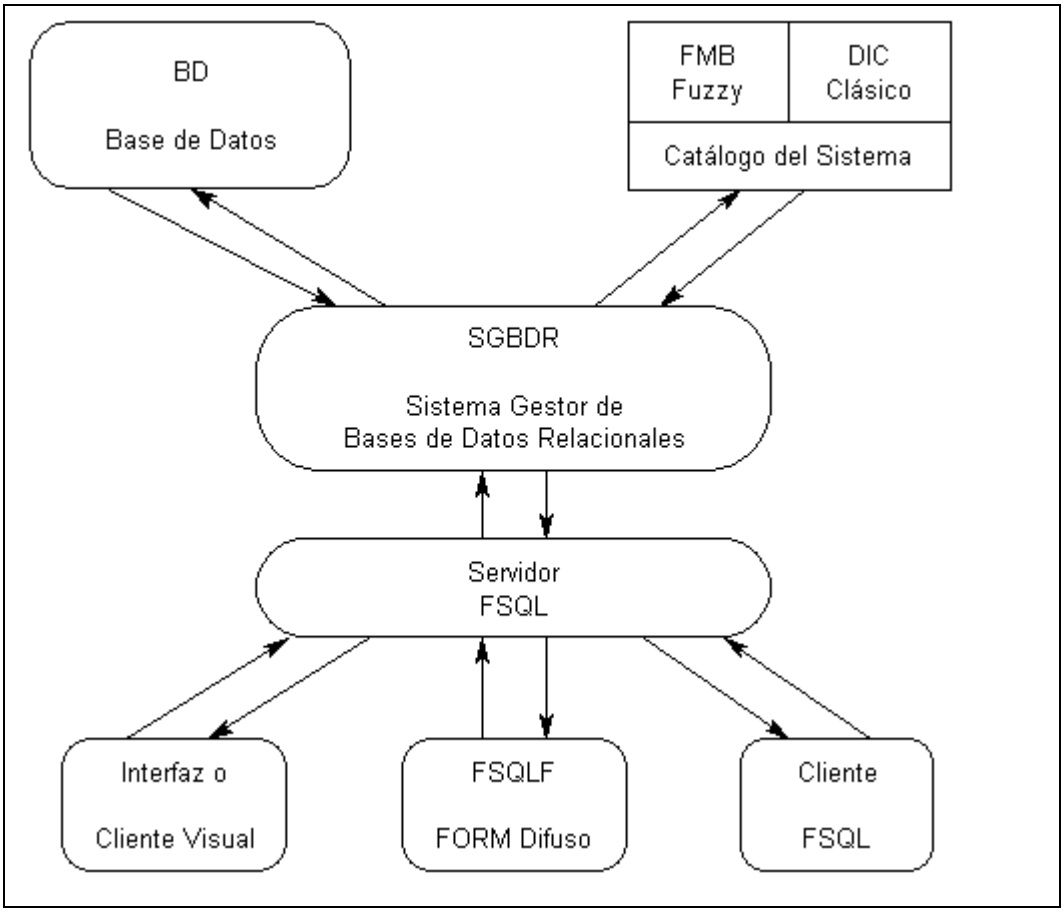


Figura 2.1. Esquema General de FIRST.

2.1.2. Representación del Conocimiento Impreciso

A continuación resumimos los criterios adoptados para la representación del conocimiento impreciso: Datos difusos, comparadores difusos, umbrales de cumplimiento y cuantificadores de la consulta.

2.1.2.1. Representación de los Datos Difusos y/o con Tratamiento Difuso

Para los diferentes tipos de datos que constituían la definición de dominio difuso generalizado, acordamos los siguientes criterios de representación:

1. Datos Precisos (crisp, clásicos): Se empleará la representación que proporcione el SGBDR anfitrión. Es decir, se adoptarán los formatos para cadenas alfanuméricas, valores numéricos, fechas, horas... e incluso objetos, soportados por el sistema sobre el que se construya FIRST.

2. Datos Imprecisos (fuzzy, difusos): Los datos de naturaleza imprecisa soportados por GEFRED pueden ser clasificados en dos grupos con distintas representaciones para cada uno de ellos: Sobre referencial ordenado y sobre referencial discreto no ordenado. Por "referencial" se entiende el dominio subyacente del atributo en cuestión. Así, por ejemplo, un atributo Altura puede ser considerado difuso para almacenar valores como "Alto" (ver Figura 2.3), "Bajo"... pero el dominio subyacente sobre el que se construyen las distribuciones de posibilidad es ordenado y corresponde a los centímetros de altura posibles.

DATOS IMPRECISOS SOBRE REFERENCIAL ORDENADO

Este grupo de datos contiene distribuciones de posibilidad sobre dominios continuos o discretos sobre los que existe una relación de orden. A este grupo pertenecen el tipo 6 de la Tabla 2.4. Cada dato de este tipo tiene asociada una función de pertenencia. Por cuestiones de simplicidad de representación y de eficiencia en el cálculo, adoptaremos las siguientes representaciones para este tipo de datos:

- **Distribución de Posibilidad Trapezoidal**: Esta representación determina la función de pertenencia asociada al dato mediante el uso de cuatro parámetros [α, β, γ, δ], tal y como se muestra en la Figura 2.2. Utilizamos funciones de pertenencia normalizadas, aquellas que poseen un núcleo no vacío, i.e., tienen posibilidad máxima (1) para, al menos, un valor del dominio subyacente.

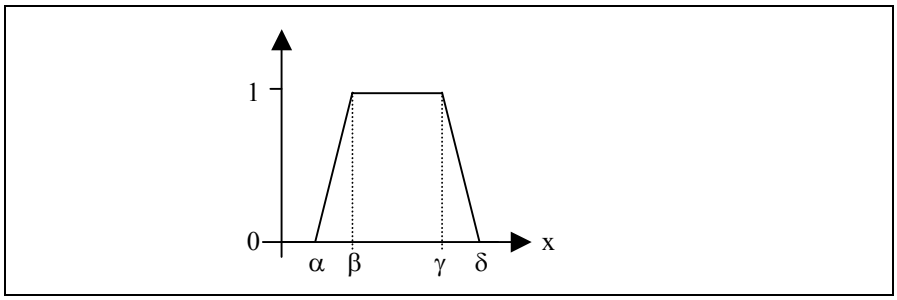


Figura 2.2. Formato de una distribución de posibilidad trapezoidal.

- **Etiqueta Lingüística**: Los datos expresados mediante una etiqueta lingüística hacen referencia a un concepto impreciso, a veces subjetivo, que lleva asociado una distribución de posibilidad. Por ejemplo, la etiqueta lingüística "Alto", puede llevar asociada la distribución de posibilidad en representación trapezoidal que muestra la Figura 2.3.

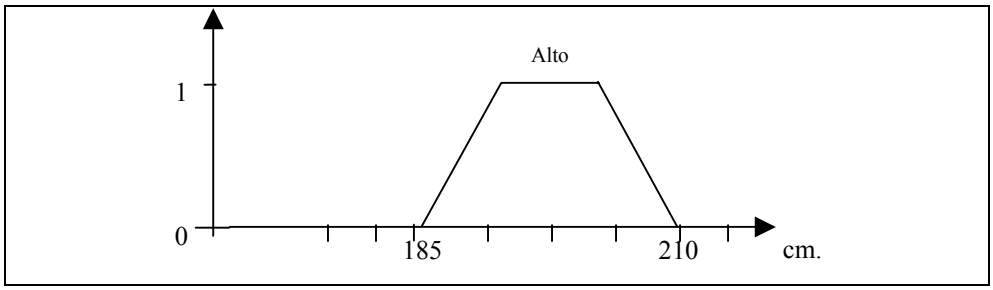


Figura 2.3. Ejemplo de una Etiqueta Lingüística para el concepto "Alto".

- **Valores Aproximados**: Dado un valor, n, perteneciente al dominio subyacente, podemos dar una representación del concepto impreciso "aproximadamente n" mediante un valor, llamado "margen", a partir del cual podemos construir su función de pertenencia como una distribución de posibilidad triangular, ver Figura 2.4. Nuevamente empleamos funciones de pertenencia normalizadas.

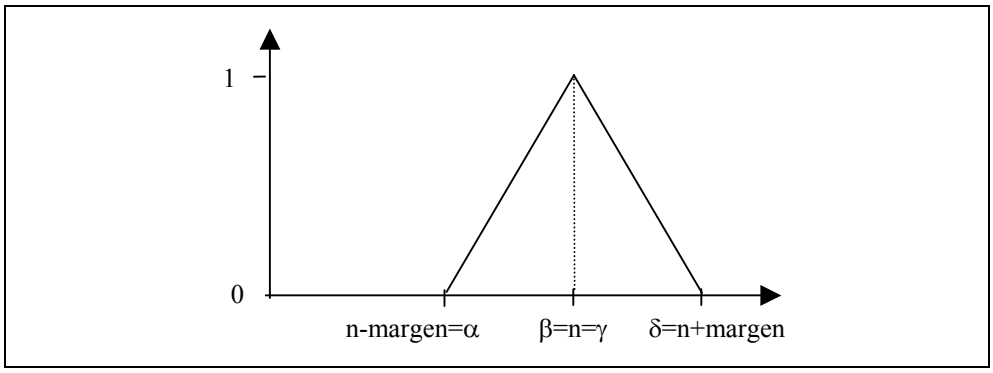


Figura 2.4. Distribución de posibilidad para "Aproximadamente n" (#n).

- **Intervalos de posibilidad**: Son un caso de distribuciones de posibilidad trapezoidales en el que las pendientes de ambos lados del trapecio son infinitas y por tanto todos los valores entre los dos extremos son los únicos que son totalmente posibles (posibilidad 1), como se muestra en la Figura 2.5. Se opera con ellos de forma similar a como se hace con la distribuciones de posibilidad trapezoidales.

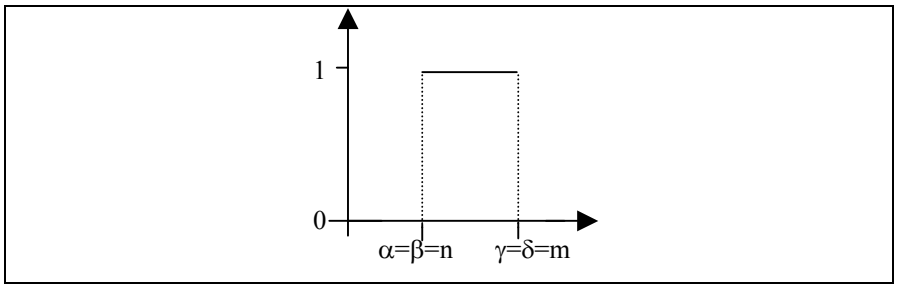


Figura 2.5. Distribución de posibilidad para el intervalo [n,m].

DATOS CON ANALOGIA SOBRE REFERENCIAL NO ORDENADO

Este grupo de datos está construido sobre dominios subyacentes discretos no ordenados en los que se encuentran definidas "relaciones de semejanza" o similitud entre los valores que lo constituyen. Para este tipo de datos tendremos que proporcionar almacenamiento para la representación de los mismos, así, como para las "relaciones de semejanza" definidas sobre los valores del dominio. Los diferentes tipos que podemos representar dentro de este grupo son:

- **Escalares Simples**: Se considera como una distribución de posibilidad con una única pareja de datos en la que el valor de posibilidad es 1. O sea, se considera que el valor del escalar d es el único posible y su posibilidad es 1 (para que esté normalizada).

- **Distribución de Posibilidad sobre Escalares**: A un dato impreciso de este tipo le asociamos una representación en la que se describen los valores del dominio de discurso que la componen con los respectivos valores de posibilidad para cada uno de ellos.

Por otra parte, existen otros 3 valores especiales que pueden almacenarse en cualquiera de los tipos de datos "imprecisos" que acabamos de describir.

- **UNKNOWN** (Desconocido, pero aplicable): Un dato de este tipo refleja el desconocimiento con respecto al valor que toma un atributo. Sabemos, sin embargo, que el atributo puede tomar algún valor del dominio de discurso. Esto implica que es posible que tome cualquiera de ellos, por tanto representaremos el tipo UNKNOWN mediante la distribución de posibilidad, $\{1/u, \forall u \in U\}$ donde U es el dominio subyacente. La Figura 2.6 muestra gráficamente esta distribución de posibilidad, la cual toma el valor 1 para todo el dominio subyacente.

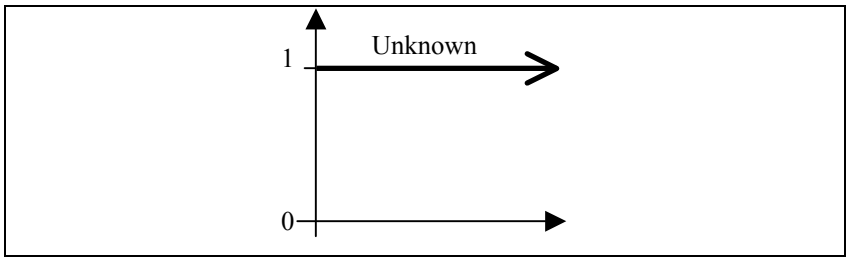


Figura 2.6. Distribución de posibilidad para el tipo UNKNOWN.

- **UNDEFINED** (No aplicable): Cuando un atributo toma el valor UNDEFINED refleja el hecho de que ninguno de los valores de dominio sobre el que está definido es aplicable. Esto se puede entender como que ninguno de los valores de dominio es posible, por lo que lo representaremos mediante la distribución de posibilidad, {0/u,

$\forall u \in U\}$ donde U es el dominio subyacente. La distribución de posibilidad se muestra en la Figura 2.7, la cual toma el valor 0 para todo el dominio subyacente.

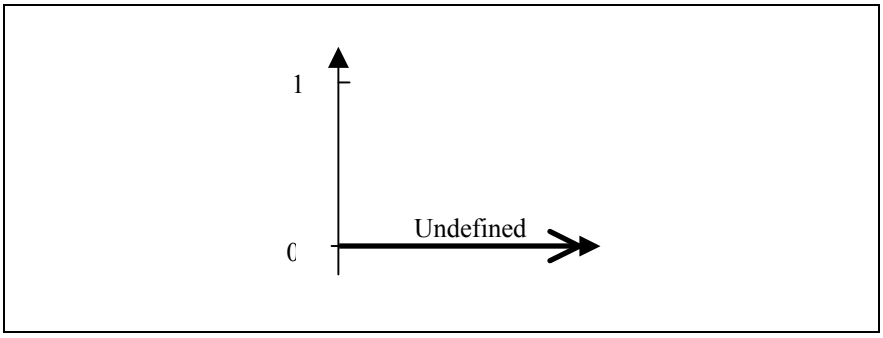


Figura 2.7. Distribución de posibilidad para el tipo UNDEFINED.

- **NULL** (Ignorancia absoluta): Sobre un atributo tenemos un valor NULL cuando no aportamos información, bien porque no la conocemos (UNKNOWN) o porque no es aplicable (UNDEFINED). Mediante el conjunto (1/UNKNOWN,1/UNDEFINED) podemos modelar este tipo de dato.

2.1.2.2. Comparadores Difusos Generalizados

En la literatura pueden encontrarse diversos métodos de comparación de números difusos, los cuales pueden clasificarse en dos categorías: los que emplean una función del conjunto de números difusos a un conjunto ordenado y los que utilizan relaciones difusas para el proceso de comparación. Nosotros basaremos la implementación que estamos describiendo en comparadores construidos en torno al segundo tipo. Concretamente, partiremos del esquema formulado en [18], si bien su representación ha sido simplificada y adaptada a las peculiaridades tanto de GEFRED como de FIRST.

Como vimos en el apartado dedicado a GEFRED, la definición de comparador difuso generalizado permitirá modelar una amplia variedad de modalidades de comparación. FIRST no pone límite con respecto al número y tipo de los comparadores que pueden emplearse. Como veremos, en la implementación del modelo de BDRD que aquí presentamos se han implementado 15 comparadores diferentes. De ellos, 14 son para datos sobre referencial

ordenado y se muestran en la Tabla 2.4 y se definen en el apartado 2.2.1.2 . Además, en el apartado 2.2.1.4 se define 1 comparador para los datos sobre referencial no ordenado.

2.1.2.3. Umbral de Cumplimiento de una Condición Difusa: Cualificadores

Cuando planteamos una consulta en una base de datos con imprecisión se establecen una serie de condiciones a cumplir por las tuplas que la satisfagan. Dada la naturaleza imprecisa de los operadores y de los datos sobre los que operamos, existe un grado de cumplimiento para cada condición involucrada en una consulta. Este grado de cumplimiento se halla comprendido entre 0 y 1. Mediante el empleo de un umbral mínimo, entre 0 y 1, para el grado de cumplimiento podemos ejercer algún control sobre la precisión con que se satisfacen cada una de las condiciones de la consulta. Si establecemos un umbral de cumplimiento 1 para una condición envuelta en una consulta, eliminaremos aquellas tuplas que no igualen o superen el umbral para esa condición.

En nuestro modelo es posible asociar un cualificador lingüístico (una palabra) a un umbral determinado, de forma que puedan efectuarse consultas poniendo como umbral dicho cualificador, haciendo que las consultas contengan más significado.

2.1.2.4. Representación de Cuantificadores Difusos de la Consulta

Como se vio en el apartado 2.1.2.1, los cuantificadores difusos absolutos se representan como distribuciones de posibilidad trapezoidales en el intervalo [0, +∞] y los relativos como distribuciones de posibilidad trapezoidales en el intervalo [0,1]. En la FMB se almacenarán los valores de un trapecio en esos rangos.

2.1.3. Implementación de FIRST en Oracle

A continuación vamos a describir en líneas generales, cuales son los niveles sobre los que podemos actuar para llevar a cabo la implementación de los principales módulos de FIRST empleando los recursos antes mencionados:

- A nivel de la Base de Datos. La base de datos es aquella colección de datos persistentes que constituyen la representación de una fracción de conocimiento del universo. Como nuestro sistema contempla la representación de conocimiento impreciso, debemos determinar la forma en que éste se almacena en la base de datos. Por tanto, habrá que extender la representación de los datos para albergar este tipo de información. En realidad, como veremos, este "tipo" de información imprecisa puede ser, de hecho, de muchos "tipos" diferentes.
- A nivel del Catálogo del Sistema. En un SGBDR clásico existe una parte del sistema en la que se recoge toda aquella información que el gestor necesita saber acerca de los datos que almacena, "datos sobre los datos" (denominados a veces "metadatos"). Esta parte representa habitualmente dicha información mediante el empleo de tablas o relaciones organizadas siguiendo un esquema similar al empleado por la propia base de datos. FIRST necesita tener información sobre los elementos de la base de datos que contienen información imprecisa así como de la naturaleza y representación de dicha información. Denominaremos Base de Metaconocimiento Difuso (FMB), a aquella extensión del Catálogo del Sistema que recoge la información necesaria sobre los datos de naturaleza imprecisa existentes en la base de datos. La FMB será explicada en el apartado 2.1.3.2 y siguientes.
- A nivel del Gestor de FIRST. El gestor posee conocimiento implementado sobre las operaciones de naturaleza imprecisa que puede realizar y de cómo hacerlo. El Servidor FSQL es la parte del sistema que contiene las rutinas que se encargan de implementar esas operaciones.

A continuación exponemos el sistema utilizado para la representación del conocimiento impreciso y para el almacenamiento de la FMB.

2.1.3.1. Representación del Conocimiento Impreciso en la Base de Datos Oracle

En el sistema que se está describiendo existen tres tipos de atributos susceptibles de tratamiento impreciso. La clasificación adoptada se basa en criterios de representación y de tratamiento de los datos "imprecisos": Se clasifican según el tipo del dominio que les subyace y por si permiten el almacenamiento de información imprecisa o sólo permiten el tratamiento impreciso de datos sin imprecisión:

ATRIBUTOS DIFUSOS TIPO 1

Son atributos con "datos precisos", clásicos o crisp, que pueden tener etiquetas lingüísticas definidas sobre ellos. Este tipo de atributos reciben una representación igual que los datos precisos. Sin embargo, cuentan con información en la FMB, donde se almacenarán las etiquetas, también recogerá información acerca de la naturaleza de estos atributos. Por tanto, son atributos clásicos que admiten el tratamiento difuso y por tanto podremos efectuar consultas difusas (flexibles) sobre ellos, aunque no permitan almacenar valores difusos. Este tipo de atributos son los que permitirán que bases de datos relacionales clásicas puedan beneficiarse de nuestro trabajo, permitiendo el tratamiento difuso de datos no difusos, mediante el empleo de consultas flexibles sobre una base de datos tradicional. Como veremos, esto puede ser utilizado para, por ejemplo, operaciones de Data Mining.

ATRIBUTOS DIFUSOS TIPO 2

Son atributos que pueden recoger "datos imprecisos sobre referencial ordenado". Estos atributos recogen el tipo de datos cuya representación ha sido descrita en el apartado 2.1.2.1. Permiten también la representación de información incompleta en forma de datos de tipo UNKNOWN, UNDEFINED y NULL. Lógicamente, también admiten "información precisa".

Tipos de Valores	Atributos de la BD para cada Tipo 2				
	FT	F1	F2	F3	F4
UNKNOWN	0	NULL	NULL	NULL	NULL
UNDEFINED	1	NULL	NULL	NULL	NULL
NULL	2	NULL	NULL	NULL	NULL
CRISP	3	NULL	NULL	NULL	NULL
LABEL	4	FUZZY_D	NULL	NULL	NULL
INTERVALO [n,m]	5	N	NULL	NULL	m
APROXIMADAMENTE(d)	6	D	d-margen	D+margen	margen
TRAPECIO [α,β,γ,δ]	7	α	β-α	γ-δ	δ

Tabla 2.1. Representación interna de Atributos Difusos Tipo 2.

En la Tabla 2.1 mostramos el sistema utilizado para representar a los atributos difusos Tipo 2. Para un atributo Difuso Tipo 2 llamado F, la representación usa 5 atributos, FT para almacenar el código de tipo que le corresponde a cada valor y los atributos F1, F2, F3, y F4, para almacenar los parámetros de cada dato. Los valores NULL que aparecen en los atributos tienen el significado de valor "no-aplicable" en el SGBDR anfitrión.

Así, vemos que un atributo difuso Tipo 2, llamado por ejemplo F, está compuesto, de hecho, por 5 atributos clásicos:

- **FT**: Almacena el tipo de valor que corresponde al dato que queremos almacenar, indicando su representación. Según lo visto, puede ser: UNKNOWN (0), UNDEFINED (1), NULL (2), CRISP (3), LABEL (4), INTERVALO (5), APROXIMADAMENTE (6) o TRAPEZOIDAL (7). Observe que añadimos una T al nombre del atributo.
- **F1**, **F2**, **F3**, y **F4**: Los atributos cuyo nombre se forma añadiendo los números **1**, **2**, **3** y **4** al nombre del atributo almacenan la descripción de los parámetros que definen el dato y que depende del tipo de valor (**FT**) al que pertenezca:
- **UNKNOWN**, **UNDEFINED**, **NULL**: Estos 3 valores no necesitan ningún parámetro, por lo que todos ellos permanecen a NULL (entendiendo este valor como el NULL del SGBD anfitrión y no como el NULL del valor difuso, que no deben confundirse).
- **CRISP**: Un valor de tipo crisp, necesita tan sólo un parámetro, F1, en el cual se almacenará el valor crisp en cuestión.
- **LABEL**: Igualmente, un valor de tipo etiqueta sólo necesita un parámetro para almacenar el identificador asociado a dicha etiqueta (FUZZY ID). Ese indicador es útil para poder acceder a la FMB y obtener la descripción asociada a esta etiqueta.
- **INTERVALO**: Necesita los dos valores extremos del intervalo [n,m], que son almacenados en F1 y F4 respectivamente.
- **APROXIMADAMENTE**: Este valor sólo necesita un valor que se almacena en F1 y que es el valor central de la distribución de posibilidad triangular, d. Sin embargo, para reducir operaciones (tanto matemáticas como de acceso a datos), se aprovechan los atributos F2, F3 y F4 para almacenar los valores d-margen, d+margen y margen, respectivamente. El valor margen es un valor almacenado en la FMB para cada atributo difuso, y su valor depende del significado de dicho atributo. Esto nos permite la posibilidad de almacenar valores aproximados sin indicar el margen, usando el margen por defecto almacenado en la FMB, o también almacenar valores aproximados con un margen particular para ellos, distinto al de la FMB.
- **TRAPECIO**: Necesita forzosamente almacenar los 4 valores que identifican a un trapecio: $[\alpha,\beta,\gamma,\delta]$. En F2 y F3 se almacenan unas operaciones que simplifican las

ecuaciones cuando se opera con este tipo de dato.

En esta representación se ha primado los aspectos siguientes:

- Velocidad de ejecución frente a economía de almacenamiento. Para alguno de los tipos que puede recoger este atributo, se podría emplear una representación más compacta, sin embargo, esto ralentizaría la mayoría de las operaciones implicadas en una consulta (En realidad bastaría con los atributos FT y F1, almacenando los valores de cada valor trapecio en una tabla independiente, asociada a cada tabla que contenga atributos difusos Tipo 2).
- Uniformidad en la representación. Empleamos cinco atributos clásicos para representar cada atributo difuso de este tipo.
- Uso de los elementos del SGBDR anfitrión para representar la información respetando en dicha representación el esquema relacional. Este criterio, unido a otros adoptados a lo largo de esta exposición, posibilitará el reducir cualquier operación de naturaleza imprecisa a términos del modelo relacional clásico.

ATRIBUTOS DIFUSOS TIPO 3

Son atributos sobre "dominio discreto no ordenado con analogía". Estos atributos recogen datos escalares simples (SIMPLE) o distribuciones de posibilidad (DISTR. POS.) sobre dominios escalares, representados en la forma que se describe en el apartado 2.1.2.1. También aceptan datos de tipo UNKNOWN, UNDEFINED y NULL. Igual que en atributos difusos Tipo 2, para atributos de este tipo tendremos que almacenar en la base de datos el tipo del valor almacenado y los datos de este valor. La FMB (Base de Metaconocimiento Difuso) contabilizará cada atributo de este tipo que aparezca en la base de datos. También almacenará las "relaciones de semejanza" definidas sobre el dominio subyacente.

En la Tabla 2.2 mostramos el sistema utilizado para representar a los atributos difusos Tipo 3. Así, vemos que un atributo difuso Tipo 3, llamado por ejemplo **F**, está compuesto, de hecho, por un número variable de atributos clásicos:

- **FT**: El tipo de valor que corresponde al dato que queremos almacenar. Este puede ser: UNKNOWN (**0**), UNDEFINED (**1**), NULL (**2**), SIMPLE (**3**) o DISTRIBUCIÓN

de POSIBILIDAD (**4**).

- Lista de n parejas, con n≥1, del tipo (valor de posibilidad, etiqueta), **(FP,F1), ... (FPn,Fn)**. En estos atributos se almacenan los datos de la distribución de posibilidad que deseamos almacenar. En un valor de tipo SIMPLE sólo se usa la primera pareja y el valor de posibilidad debería ser 1 (para estar normalizada).

Tipos de Valores	Atributos de la BD para cada Tipo 3					
	FT	FP1	F1	…	FPn	Fn
UNKNOWN	0	NULL	NULL	…	NULL	NULL
UNDEFINED	1	NULL	NULL	…	NULL	NULL
NULL	2	NULL	NULL	…	NULL	NULL
SIMPLE	3	p	d	…	NULL	NULL
DISTR. POS.	4	p1	d1	…	pn	dn

Tabla 2.2. Representación interna de Atributos Difusos Tipo 3.

En la Tabla 2.2, vemos que el valor n es el máximo número de pares (grado posibilidad, valor) que puede representar el atributo instanciado. Este valor, que tiene que ser establecido en el momento en que se declara el atributo, acota la capacidad de este atributo para representar distribuciones de posibilidad y está almacenado en la FMB. En un valor de tipo DISTR .POS. se podrán almacenar hasta n parejas, donde en cada una de ellas el valor de posibilidad estará en el intervalo [0,1]. Se pueden usar menos de n parejas dejando el resto de campos a NULL. En la FMB se almacenan las etiquetas, su relación de semejanza y el valor de n.

2.1.3.2. FMB (Fuzzy Metaknowledge Base) Base de Metaconocimiento Difuso

Como hemos visto en el apartado anterior, existe cierto tipo de información sobre los atributos descritos, que precisa ser almacenada de una forma accesible por el sistema. La Base de Metaconocimiento Difuso, FMB, va a ser la encargada de organizar toda aquella información relacionada con la naturaleza imprecisa de estos atributos. En FIRST se contempla la Base de Metaconocimiento Difuso como una extensión del Catálogo del sistema, por ello, organiza la información mediante el uso de tablas o relaciones. Los elementos del tratamiento impreciso que se almacenan en la Base de Metaconocimiento son los siguientes:

- Atributos de la base de datos que reciben tratamiento impreciso.
- Clase de información imprecisa que recogen: De qué Tipo difuso son estos atributos (Tipo 1, 2 ó 3) y la longitud máxima de las distribuciones de posibilidad para los atributos Tipo 3.
- Objetos definidos en el ámbito de la base de datos, como por ejemplo cuantificadores difusos de consulta.
- Objetos difusos definidos sobre cada atributo:
  - ✓ Etiquetas lingüísticas (para atributos difusos Tipo 1, 2 y 3).
  - ✓ El margen para valores aproximados y la distancia mínima para que 2 valores sean considerados como muy separados (para atributos difusos Tipo 1 y 2).
  - ✓ Relaciones de semejanza (para atributos difusos 3).
- La descripción de esos objetos.

Tabla/Vista	Sinónimo
T.FUZZY_COL_LIST	FCL
T.FUZZY_OBJECT_LIST	FOL
T.FUZZY_LABEL_DEF	FLD
T.FUZZY_APPROX_MUCH	FAM
T.FUZZY_NEARNESS_DEF	FND
T.FUZZY_COMPATIBLE_COL	FCC
T.FUZZY_QUALIFIERS_DEF	FQD
V.LABELS_FOR_OBJCOL	LFOC
V.LABELS_OBJCOL_T	LOCT3
V.ALL_COMPATIBLES_T	ACT3

Tabla 2.3. Tablas y vistas de FIRST y sus sinónimos públicos.

Resumiendo, la FMB almacena una lista con los atributos que admiten tratamiento difuso y su Tipo de atributo difuso (1, 2 ó 3). Además, para cada atributo difuso se almacena distinta información dependiendo de su Tipo:

- **Atributos Difusos Tipo 1 o Tipo 2**: Para poder usar atributos Tipo 1 (crisp) y atributos Tipo 2 en condiciones difusas (consultas flexibles) basta con declararlos como tales (tabla FUZZY_COL_LIST de la FMB) y almacenar los siguientes datos en la FMB:
  - ✓ Etiquetas lingüísticas trapezoidales: Nombre de la etiqueta (tabla FUZZY_OBJECT_LIST) y los valores α,β,γ y δ (FUZZY_LABEL_DEF), como en la Figura 2.2.

- ✓ Valor para el margen de los valores aproximados (campo MARGEN de la tabla FUZZY_APROX_MUCH): Usada cuando se emplean valores aproximados en un atributo concreto (ver Figura 2.4 y Tabla 2.1 y Tabla 2.6).
- ✓ Distancia mínima para que dos valores sean considerados como muy separados (campo MUCH de la tabla FUZZY_APROX_MUCH): Usada por los comparadores MGT/NMGT y MLT/NMLT.
- ✓ Cualificadores y Cuantificadores (tablas FUZZY_QUALIFIERS_DEF y FUZZY_LABEL_DEF).

- **Atributos Difusos Tipo 3**: Además de declararlos como tales (tabla FUZZY_COL_LIST), en la FMB hay que almacenar también:
  - ✓ Longitud máxima de sus distribuciones de posibilidad (tabla FUZZY_COL_LIST).
  - ✓ Etiquetas lingüísticas (escalares): Su nombre (tabla FUZZY_OBJECT_LIST) y la relación de similitud entre ellas (tabla FUZZY_NEARNESS_DEF).
  - ✓ Compatibilidad entre atributos de este Tipo (tabla FUZZY_COMPATIBLE_COL).
  - ✓ Cualicadores y Cuanticadores (tablas FUZZY QUALIFIERS DEF y FUZZY LABEL DEF).

2.1.3.3. Vistas sobre la FMB

Sobre las tablas de FIRST existen unas vistas que son útiles para consultar distintos aspectos de una base de datos difusa de forma clara y sencilla.

- Vista **LABELS_FOR_OBJCOL**: Sirve para obtener o consultar fácilmente las etiquetas definidas sobre atributos difusos Tipo 1 ó 2 y los parámetros del trapecio posibilístico asociado.
- Vista **LABELS_OBJCOL_T3**: Sirve para obtener o consultar fácilmente las etiquetas definidas sobre atributos difusos Tipo 3, así como la relación de similitud entre ellos.
- Vista **ALL_COMPATIBLES_T3**: Sirve para obtener o consultar fácilmente los atributos difusos Tipo 3 que se han definido como compatibles a otros. De esta forma, podemos saber qué atributos difusos Tipo 3 no tienen etiquetas definidas sobre ellos y cuales sí las tienen. Además, también nos muestra la longitud máxima permitida para las distribuciones de posibilidad de esos atributos difusos Tipo 3.

2.2. Sintaxis y Semántica del Lenguaje FSQL

Los comandos básicos de este lenguaje se dividen en dos vertientes [16]:

- **DML** (Data Manipulation Language, Lenguaje de Manipulación de Datos): Las sentencias de este lenguaje permiten la consulta y la modificación de los datos almacenados en la base de datos. Ejemplos de comandos del DML SQL son: SELECT, INSERT, DELETE y UPDATE.
- **DDL** (Data Definition Language, Lenguaje de Definición de Datos): Las sentencias de este lenguaje permiten la creación y modificación de las estructuras en las que se almacenarán los datos. Ejemplos de comandos del DDL SQL son: CREATE (para crear objetos de la base de datos: tablas, vistas...), DROP (para borrar objetos), ALTER (para modificar objetos), y sentencias para controles de seguridad, índices y control del almacenamiento físico de los datos. Algunos autores distinguen entre el DDL y el DCL (Data Control Language, Lenguaje de Control de Datos), dejando para este último las tareas de control (de seguridad, almacenamiento...).

Cada sentencia FSQL que se desee ejecutar debe ser previamente analizada por el sistema para asegurar, por un lado que la sentencia está correctamente escrita, pertenece a nuestra gramática, y por otro, que tiene sentido efectuarla (por ejemplo, que usamos objetos válidos). Para ello usamos los típicos 3 analizadores siguientes:

- **Analizador Léxico**: Asegura que todos los elementos de la sentencia están permitidos, agrupando los caracteres en palabras (tokens). Igual que SQL, el lenguaje FSQL no es sensible a mayúsculas/minúsculas, i.e., las sentencias pueden escribirse independientemente en mayúsculas, minúsculas o ambas.
- **Analizador Sintáctico**: Asegura que los tokens están en un orden adecuado y que la construcción de la sentencia es correcta sintácticamente.
- **Analizador Semántico**: Asegura que el significado de la sentencia es correcto y que por tanto, tiene sentido efectuarla.

A continuación nos centraremos en revisar la sintaxis de los comandos más útiles e importantes, explicando las novedades que éstos incorporan en FSQL para que nos permitan manejar información difusa.

2.2.1. El DML de FSQL: SELECT

Dentro del DML la sentencia más usual y más compleja es la sentencia de consulta de datos, **SELECT**, aunque también son interesantes las sentencias INSERT, DELETE y UPDATE. Vamos a centrarnos únicamente en el SELECT, puesto que es la piedra angular sobre la que se basa el programa cliente del presente proyecto.

2.2.1.1. El SELECT Difuso

La sentencia SELECT es una sentencia tan potente como compleja y flexible, muy fácil de usar en consultas simples y no tan fácil de usar en consultas complejas debido a su potencia y versatilidad. Esta sentencia es tan potente que rara vez se llega a utilizar todo su poder expresivo para realizar una consulta, pues lo normal es efectuar consultas mucho más simples de lo que SELECT permite. A veces, para simplificar la escritura y el entendimiento de consultas complicadas se usan vistas intermedias que son creadas como subconsultas a la base de datos.

El lenguaje FSQL es una auténtica extensión de SQL. Esto significa que todas las sentencias válidas en SQL lo son también en FSQL. Además, FSQL incorpora algunas novedades para permitir el tratamiento de información imprecisa. Básicamente, las extensiones efectuadas a esta sentencia son las siguientes:

- **Etiquetas Lingüísticas**: Si un atributo es susceptible de tratamiento difuso entonces pueden definirse etiquetas sobre él. Estas etiquetas son precedidas, por convenio, por el símbolo $ para distinguirlas fácilmente de otros identificadores.

  Hay 2 tipos de etiquetas que serán usadas en diferentes tipos de atributos difusos.

- Etiquetas para atributos con un dominio subyacente ordenado: Cada etiqueta de este tipo tiene asociada, en la FMB, una distribución de posibilidad trapezoidal como la de la Figura 2.2. Así, por ejemplo, podemos definir las etiquetas $Muy_Bajo, $Bajo, $Normal, $Alto y $Muy_Alto sobre el atributo ***Altura*** de una persona. Por ejemplo, el atributo $Alto puede ser definido como una distribución de posibilidad con los siguientes valores (en centímetros) α=185, β=195, γ=200 y δ=210, como se muestra en la Figura 2.3.
- Etiquetas para atributos con un dominio subyacente no ordenado: Aquí, puede haber una relación de similitud definida entre cada dos etiquetas del dominio y almacenada en la FMB. El grado de similitud entre cada dos etiquetas será un valor del intervalo [0,1]. Por ejemplo, para un atributo que almacenara el color del pelo de una persona podríamos definir las etiquetas $Rubio y $Pelirrojo e indicar (en la relación de similitud) que ambos valores son similares en grado 0.6.

- **Comparadores Difusos**: Además de los comparadores clásicos típicos (=,>...), FSQL incluye los comparadores difusos de la Tabla 2.4. Como en SQL, los comparadores difusos pueden comparar una columna de una tabla con una constante o dos columnas del mismo tipo o de tipos compatibles. Los comparadores de posibilidad son más generales (menos restrictivos) que los de necesidad. Por tanto, los comparadores de necesidad recuperan menos tuplas y estas tuplas cumplirán necesariamente con las condiciones impuestas en la consulta.

	Comp. Difuso	Significado
Posibilidad	FEQ	Fuzzy Equal: Posiblemente Igual (sobre escalares o números difusos)
	FGT	Fuzzy Greater Than: Posiblemente Menor que
	FGEQ	Fuzzy Greater or Equal: Posiblemente Mayor o Igual que
	FLT	Fuzzy Less Than: Posiblemente Menor que
	FLEQ	Fuzzy Less or Equal: Posiblemente Menor o Igual que
	MGT	Much Greather Than: Posiblemente Mucho Mayor que
	MLT	Much Less Than: Posiblemente Mucho Menor que
Necesidad	NFEQ	Necessarily Fuzzy Equal: Necesariamente Igual que (o incluido en)
	NFGT	N. Fuzzy Greater Than: Necesariamente Mayor que
	NFGEQ	N. Fuzzy Greater or Equal: Necesariamente Mayor o Igual que
	NFLT	N. Fuzzy Less Than: Necesariamente Menor que
	NFLEQ	N. Fuzzy Less or Equal: Necesariamente Menor o Igual que
	NMGT	N. Much Greater Than: Necesariamente Mucho Mayor que
	NMLT	N. Much Less Than: Necesariamente Mucho Menor que

Tabla 2.4. Comparadores Difusos de FSQL (Fuzzy SQL).

En atributos con dominio subyacente no ordenado (difusos Tipo 3) sólo puede usarse el comparador difuso FEQ, puesto que carecen de orden.

El comparador difuso de "desigualdad" o "posiblemente distinto" no se ha considerado porque puede modelarse negando una comparación con FEQ:

```
NOT <Atributo_Difuso> FEQ <Atributo_o_Cte>
```

- **Umbral de Cumplimiento** (threshold, τ): Para cada condición simple se puede establecer un umbral de cumplimiento (por defecto será 1) con el formato siguiente:

```
<Condición simple> THOLD τ
```

indicando que la condición debe ser satisfecha con un grado mínimo de $\tau \in [0,1]$ para que la tupla en cuestión aparezca en la relación resultante. La palabra reservada THOLD es opcional y puede ser sustituida por cualquier comparador crisp tradicional (=,<...), modificando así, lógicamente, el significado de la consulta. La palabra THOLD es equivalente a usar el comparador crisp ≥.

En lugar de un número, como umbral puede ponerse un cualificador, que es un identificador o etiqueta que debería estar definida en la FMB.

<Condición> (con operadores lógicos)	CDEG(<Condición>)
<cond1> AND <cond2>	min(CDEG(<cond1>),CDEG(<cond2>))
<cond1> OR <cond2>	max(CDEG(<cond1>),CDEG(<cond2>))
NOT <cond1>	1 – CDEG(<cond1>)

Tabla 2.5. Operaciones por defecto para el cálculo de la función CDEG de FSQL con operadores lógicos.

Ejemplo: Dame todas las personas con pelo rubio (en grado mínimo 0.5) que son posiblemente más altas que la etiqueta **$Alta** (en grado mínimo 0.8):

```
SELECT * FROM Personas WHERE Pelo FEQ $Rubio THOLD 0.5 AND Altura FGT $Alta
THOLD 0.8
```

Si buscamos personas que son "necesariamente" más altas que la etiqueta $Alta, entonces debemos usar el comparador difuso NFGT en vez de FGT.

- **Función CDEG(<atributo>)**: Una llamada a la función CDEG (Compatibility DEGree) puede ser colocada en la lista de selección (expresiones tras la palabra reservada SELECT). Su argumento es un atributo y muestra una columna con el grado de compatibilidad o cumplimiento de la condición de la consulta para el atributo que se indica. Si aparecen operadores lógicos (NOT, AND y/o OR), el cálculo de este grado de cumplimiento es efectuado usando las funciones que se indican en la Tabla 2.5: Se usa la norma triangular del mínimo y del máximo, pero FSQL permite modificar las funciones a emplear. Para ello hay que modificar la vista FSQL_OPTIONS. En esta vista el usuario puede establecer las funciones a usar para cada uno de los 3 operadores lógicos. Obviamente, la función que se indique en esa vista debe estar implementada en el SGBD y el usuario debe tener acceso a la misma. Estas funciones pueden ser implementadas por un usuario particular para su uso personal. La función para el NOT tendría un único argumento numérico, mientras que las funciones para el AND y para el OR tendrán ambas dos argumentos numéricos. La precedencia de los operadores lógicos es la habitual, i.e., de mayor a menor precedencia están NOT, AND y OR.

  Si el argumento de la función CDEG es un atributo, entonces la función CDEG sólo usa las condiciones que incluyen a ese atributo. Si el atributo indicado como argumento de CDEG no aparece en la condición, entonces, esta función no es aplicable a dicho atributo, pero en vez de dar un error se procede a devolver grado 1 para todas las tuplas. También podemos usar un asterisco como argumento como se explica a continuación.

- **CDEG(*)**: Si esta función tiene un asterisco como argumento, entonces calcula y muestra el grado de cumplimiento de la condición para cada tupla. Este cálculo es efectuado como se explicó en el punto anterior pero teniendo en cuenta todos los atributos empleados en la condición, y no sólo uno de ellos.

Cte. Difusa	Significado
UNKNOWN	Valor desconocido pero el atributo es aplicable (tipo 8 de la tabla 2.4).
UNDEFINED	Atributo no aplicable o sin sentido (tipo 9 de la tabla 2.4).
NULL	Ignorancia total: No sabemos nada sobre eso (tipo 10 de la tabla 2.4).
$[α,β.γ,δ]	Distribución de posibilidad trapezoidal (α≤β≤γ≤δ): Ver figura 5.2.
$label	Etiqueta lingüística: Puede ser un trapecio o un escalar (definido en FMB).
[n,m]	Intervalo “Entre n y m” (α=β=n y γ=δ=m).
#n	Valor difuso “Aproximadamente n” (β=γ=n y n-α=δ-n=margen en FMB).

Tabla 2.6. Constantes difusas que pueden ser usadas en FSQL.

- **Carácter Comodín %**: El empleo de este carácter es similar al del utilísimo carácter comodín * de SQL, pero éste, además de incluir todas las columnas de las tablas indicadas en la parte FROM de la consulta, también incluye las columnas con el grado de cumplimiento de aquellos atributos relevantes. O sea, en el resultado también encontraremos columnas donde la función CDEG está aplicada a cada uno de los atributos que aparecen en la condición. Si, en el Ejemplo anteriormente expuesto, hubiéramos querido obtener dos columnas más con los grados de CDEG(Pelo) y CDEG(Altura), podríamos simplemente haber reemplazado el comodín * por %. Por supuesto, este carácter puede ser también usado con el formato [[scheme.]table.]%, como por ejemplo: Personas.%. Si un atributo difuso no aparece en la cláusula WHERE, su CDEG no es aplicable y por tanto no aparecerá su CDEG si se usa el comodín %.

- **Constantes Difusas**: En FSQL podemos usar diversos tipos de constantes difusas. Estos tipos son detallados en la Tabla 2.6. Como se ha dicho anteriormente las etiquetas están registradas en la FMB, al igual que el valor del margen para valores aproximados.

- **Condición con IS**: Otra clase de condición que podemos usar en FSQL tiene el siguiente formato:

$$< Atributo\_Difuso > IS\ [NOT] \begin{cases} UNKNOWN \\ UNDEFINED \\ NULL \end{cases} \tag{2.1}$$

Observaciones sobre la condición con IS:

- Este tipo de condición (sin NOT) será cierta si el valor del atributo difuso de la izquierda (<Atributo_Difuso>) es la constante situada a la derecha.
- Si el atributo no es difuso y la constante de la derecha es NULL, esta constante será entendida de la forma que lo haga el SGBD (si este lo permite). En particular, Oracle permite valores NULL como valor posible de los atributos, aunque esto puede ser evitado estableciendo la restricción NOT NULL en los comandos CREATE TABLE o ALTER TABLE.

- Si FEQ es usado en vez de IS el significado es distinto. Con FEQ se compara el grado de compatibilidad entre el atributo y la constante, y no simplemente si el atributo es igual a la constante.

- **Cuantificadores difusos**: Los cuantificadores difusos, tanto absolutos como relativos pueden establecerse de la siguiente forma, en sus dos modalidades, cuyos significados se muestran a continuación con unos ejemplos:

  $Cuantificador FUZZY[ρ] (condición_difusa) THOLD τ
  $Cuantificador FUZZY[ρ] (condición_difusa1) ARE (condición_difusa2) THOLD τ

  donde $Cuantificador es un cuantificador (absoluto o relativo). Está precedido del símbolo $ para distinguirlo de otros identificadores. Además, algunos cuantificadores relativos pueden llevar un argumento entre corchetes. La opción FUZZY[ρ] es opcional, al igual que su argumento ρ.

  Igual que en comparaciones difusas simples, τ es un umbral opcional, normalmente en [0,1], que debe cumplir el cuantificador para que la condición sea evaluada como cierta (por defecto 1). Aquí, también la palabra THOLD es opcional y también puede ser sustituida por cualquier comparador crisp tradicional, modificando el significado de la consulta. La palabra THOLD es equivalente a usar el comparador crisp ≥.

- **Comentarios**: FSQL permite incorporar comentarios en las sentencias, de forma que ellos no serán tenidos en cuenta al analizarla y ejecutarla. Los comentarios pueden ser de 3 tipos:

  1. Comentario hasta fin de línea: Con -- (dos guiones) señalamos que desde ese punto hasta el final de esa línea es un comentario. Este tipo de comentarios es también válido en SQL y PL/SQL.
  2. Comentario de un bloque: Todo lo que esté incluido entre las marcas /* y */ será considerado como un comentario. Este tipo de comentarios (estándar en el lenguaje C) es también válido en SQL y PL/SQL.
  3. Comentario hasta fin de sentencia: Con /* señalamos que desde ese punto hasta el final de la sentencia es un comentario. Es decir, si no se cierra el comentario se supone que el comentario termina al final, siendo ignorado totalmente el resto de

la sentencia. Este tipo de comentarios es propio de FSQL y generan un error si se usan en SQL o PL/SQL.

- **Evitar que una sentencia pase por el Servidor FSQL**: En determinadas circunstancias puede interesarnos que el Servidor FSQL no ejecute su función y se limite simplemente a operar como si no estuviera (de forma clásica). Para ello basta con incorporar el símbolo ! (admiración cerrada) como primer carácter de la sentencia enviada al Servidor. O sea, si el Servidor FSQL encuentra que el primer carácter de la sentencia es !, se comportará, con el resto de la sentencia, como si no estuviera. Esto es útil si el programa Cliente FSQL no admite la posibilidad de enviar una sentencia SQL estándar, de forma que con este sistema aceleramos el proceso para este tipo de sentencias, ya que el Servidor FSQL no se ejecuta completamente.

Las consultas difusas reducen el riesgo de obtener respuestas vacías, ya que permite utilizar una escala de discriminación más fina: El intervalo [0,1] en vez del conjunto {0,1}. Es decir, permite recuperar tuplas en consultas que en modo crisp no se obtiene ninguna respuesta. Sin embargo, a veces puede ocurrir que no existan elementos que satisfagan el resultado de una consulta. Para solucionar ese posible problema, las consultas FSQL se muestran especialmente flexibles, pues en cada condición simple, podemos actuar sobre los siguientes parámetros de consulta, para debilitar las condiciones:

- **Comparadores difusos** (Tabla 2.4): Existe una gran variedad y alternar el uso de comparadores entre posibilidad y necesidad puede resultar especialmente útil.
- **Umbrales**: Modificando este valor podemos decidir el grado de importancia de las tuplas que buscamos para recuperar sólo los elementos más importantes.
- **Usar comparadores clásicos en lugar de la palabra THOLD**: Con esto podemos conseguir modificar el significado de la consulta. Por ejemplo, podemos recuperar los elementos menos importantes usando el comparador $<$ (o $\leq$). También podemos recuperar justo los elementos que cumplen la condición con un determinado grado usando el comparador =.
- **Constantes difusas** (Tabla 2.6): Si la parte derecha de una condición simple es una constante difusa esta puede ser modificada para flexibilizar la consulta y conseguir mejor nuestro objetivo.

2.2.1.2. Comparadores Difusos de FSQL para Atributos Difusos Tipo 1 ó 2

Los comparadores difusos (Tabla 2.4) pueden usarse para comparar atributos difusos, entre sí, o con constantes (difusas o crisp).

Para atributos difusos **Tipo 3** sólo está permitido usar el comparador difuso FEQ, ya que no existe una relación de orden en el dominio de estos atributos. Por ejemplo, no podemos decir si el valor "Rubio" es mayor que el valor "Moreno", pero si podemos medir su similitud.

En atributos difusos **Tipo 2** podemos almacenar valores crisp -como en los **Tipo 1**-, pero además podemos almacenar valores como los expresados en la Tabla 2.6. El caso más general es el Trapecio Difuso, pues este tipo de valor engloba a todos los demás. Así, por ejemplo, un valor aproximado #n es una distribución de posibilidad triangular donde $\beta=\gamma=n$ y $n-\alpha=\delta-n=$margen y el valor para el margen es almacenado en la FMB (atributo MARGEN de la tabla FUZZY_APPROX_MUCH) para cada atributo de cada relación. Igualmente, un valor intervalo [n,m] puede verse como un trapecio en el que $\alpha=\beta=n$ y $\gamma=\delta=m$.

Por tanto, aquí nos centraremos en el estudio de los comparadores difusos cuando comparamos dos distribuciones de posibilidad trapezoidales, que es el caso más general. Como se verá más adelante, el Servidor FSQL tiene en cuenta los tipos de distribuciones de posibilidad que hay que comparar en cada momento para observar sus peculiaridades y aumentar la eficiencia de cada comparación. Así, si en una relación hay muchos valores crisp (o muchos intervalos...) las comparaciones serán más rápidas que si hay muchos trapecios.

Supondremos que deseamos comparar dos distribuciones de posibilidad cualesquiera que denotaremos por $A=\$[\alpha_A,\beta_A,\gamma_A,\delta_A]$ y $B=\$[\alpha_B,\beta_B,\gamma_B,\delta_B]$. Usaremos también la función CDEG, para expresar el grado de compatibilidad de una comparación difusa. El resultado de esta comparación dependerá, naturalmente, del comparador difuso (Tabla 2.4) empleado.

A continuación definimos cada uno de los comparadores difusos:

- **Comparador Difuso FEQ (Fuzzy Equal, Possibly Equal, Igual Difuso,**

Posiblemente Igual): Para comparar dos distribuciones de posibilidad A y B (trapezoidales) y obtener el Grado de Compatibilidad entre ellas, CDEG(A FEQ B), o, lo que es lo mismo, en qué medida son posiblemente similares, se usa la ecuación:

$$CDEG(A\ FEQ\ B) = \sup_{(p,p')\in UxU} \min(=(p,p'),A(p),B(p')) \quad (2.2)$$

$$= \sup_{d\in U} \min(A(d),B(d)) \quad (2.3)$$

donde U es el dominio subyacente a ambas distribuciones de posibilidad y A(d) es el grado de posibilidad para el valor d ∈ U en la distribución de posibilidad A. La función =, toma el valor 1 si sus argumentos son iguales y 0 si son diferentes.

- **Comparador Difuso NFEQ (Necessarily Fuzzy EQual, Necesariamente Igual Difuso)**: Este comparador mide el grado de necesidad (u obligatoriedad) que existe para que una distribución sea (o esté incluida en) otra. Este grado se calcula por la ecuación:

$$CDEG(A\ NFEQ\ B) = \inf_{d\in U} \max(1 - A(d), B(d)) \quad (2.4)$$

Con otros comparadores distintos de la igualdad difusa, se modifica la distribución de posibilidad de la parte derecha de la comparación. Además, en los comparadores difusos de necesidad se niega la distribución de posibilidad de la parte izquierda de la comparación, tal y como se hace en la ecuación 2.4.

- **Comparadores Difusos FGT/NFGT (possibly/Necessarily Fuzzy Greater Than, Posiblemente/Necesariamente Mayor Difuso)**: Para obtener el grado de compatibilidad con el que una distribución de posibilidad A es posiblemente/necesariamente mayor que otra B se utilizan las siguientes ecuaciones:

$$CDEG(A\ FGT\ B) = \begin{cases} 1 & si\ \gamma_A \geq \delta_B \\ \dfrac{\delta_A - \delta_B}{(\delta_B - \gamma_B) - (\gamma_A - \delta_A)} & si\ \gamma_A < \delta_B\ y\ \delta_A > \gamma_B \\ 0 & en\ otro\ caso\ (\delta_A \leq \gamma_B) \end{cases} \quad (2.5)$$

$$CDEG(A\ NFG\ T\ B) = \begin{cases} 1 & si\ \alpha_A \geq \delta_B \\ \dfrac{\beta_A - \gamma_B}{(\delta_B - \gamma_B) - (\alpha_A - \beta_A)} & si\ \alpha_A < \delta_B\ y\ \beta_A > \gamma_B \\ 0 & en\ otro\ caso\ (\beta_A \leq \gamma_B) \end{cases} \quad (2.6)$$

- **Comparadores Difusos FGEQ/NFGEQ (possibly/Necessarily Fuzzy Greater or EQual Than, Posiblemente/Necesariamente Mayor o Igual Difuso):**

$$CDEG(A\ FGEQ\ B) = \begin{cases} 1 & si\ \gamma_A \geq \beta_B \\ \dfrac{\delta_A - \alpha_B}{(\beta_B - \alpha_B) - (\gamma_A - \delta_A)} & si\ \gamma_A < \beta_B\ y\ \delta_A > \alpha_B \\ 0 & en\ otro\ caso\ (\delta_A \leq \alpha_B) \end{cases} \quad (2.7)$$

$$CDEG(A\ NFG\ EQ\ B) = \begin{cases} 1 & si\ \alpha_A \geq \beta_B \\ \dfrac{\beta_A - \alpha_B}{(\beta_B - \alpha_B) - (\alpha_A - \beta_A)} & si\ \alpha_A < \beta_B\ y\ \beta_A > \alpha_B \\ 0 & en\ otro\ caso\ (\beta_A \leq \alpha_B) \end{cases} \quad (2.8)$$

- **Comparadores Difusos FLT/NFLT (possibly/Necessarily Fuzzy Less Than, Posiblemente/Necesariamente Menor Difuso):**

$$CDEG(A\ FLT\ B) = \begin{cases} 1 & si\ \beta_A \leq \alpha_B \\ \dfrac{\alpha_A - \beta_B}{(\alpha_B - \beta_B) - (\beta_A - \alpha_A)} & si\ \beta_A > \alpha_B\ y\ \alpha_A < \beta_B \\ 0 & en\ otro\ caso\ (\alpha_A \geq \beta_B) \end{cases} \quad (2.9)$$

$$CDEG(A\ NFL\ T\ B) = \begin{cases} 1 & si\ \delta_A \leq \alpha_B \\ \dfrac{\gamma_A - \beta_B}{(\alpha_B - \beta_B) - (\delta_A - \gamma_A)} & si\ \delta_A > \alpha_B\ y\ \gamma_A < \beta_B \\ 0 & en\ otro\ caso\ (\gamma_A \geq \beta_B) \end{cases} \quad (2.10)$$

- **Comparadores Difusos FLEQ/NFLEQ (possibly/Necessarily Fuzzy Less or EQual Than, Posiblemente_Necesariamente Menor o Igual Difuso):**

$$CDEG(A\ FLEQ\ B) = \begin{cases} 1 & si\ \beta_A \leq \gamma_B \\ \dfrac{\delta_B - \alpha_A}{(\beta_A - \alpha_A) - (\gamma_B - \delta_B)} & si\ \beta_A > \gamma_B\ y\ \alpha_A < \delta_B \\ 0 & en\ otro\ caso\ (\alpha_A \geq \delta_B) \end{cases} \quad (2.11)$$

$$CDEG(A\ NFL\ EQ\ B) = \begin{cases} 1 & si\ \delta_A \le \gamma_B \\ \dfrac{\gamma_A - \delta_B}{(\gamma_B - \delta_B) - (\delta_A - \gamma_A)} & si\ \delta_A > \gamma_B\ y\ \gamma_A < \delta_B \\ 0 & en\ otro\ caso\ (\gamma_A \ge \delta_B) \end{cases} \quad (2.12)$$

En los comparadores que usan la expresión "Mucho" (mucho mayor/menor que), se usa la distancia μ (almacenada en el atributo MUCH de la tabla FUZZY_APPROX_MUCH de la FMB). También con estos comparadores se modifica la distribución de la parte derecha de la comparación.

- **Comparadores Difusos MGT/NMGT (possibly/Necessarily Much Greater Than, Posiblemente/Necesariamente Mucho Mayor difuso)**: Como ya hemos indicado, en la FMB se almacena un valor para cada atributo difuso Tipo 1 ó 2 que indica la distancia mínima para que 2 valores de ese atributo sean considerados como muy separados (atributo MUCH de la tabla FUZZY_APPROX_MUCH). Sea μ esta distancia, para un atributo con dos valores cualesquiera A y B. Entonces, para obtener el grado de compatibilidad con el que A es posiblemente/necesariamente mucho mayor que B, se utilizan las siguientes ecuaciones:

$$CDEG(A\ MGT\ B) = \begin{cases} 1 & si\ \gamma_A \ge \delta_B + \mu \\ \dfrac{\gamma_B + \mu - \delta_A}{(\gamma_A - \delta_A) - (\delta_B - \gamma_B)} & si\ \gamma_A < \delta_B + \mu\ y\ \delta_A > \gamma_B + \mu \\ 0 & en\ otro\ caso\ (\delta_A \le \gamma_B + \mu) \end{cases} \quad (2.13)$$

$$CDEG(A\ NMG\ T\ B) = \begin{cases} 1 & si\ \alpha_A \ge \delta_B + \mu \\ \dfrac{\gamma_B + \mu - \beta_A}{(\alpha_A - \beta_A) - (\delta_B - \gamma_B)} & si\ \alpha_A < \delta_B + \mu\ y\ \beta_A > \gamma_B + \mu \\ 0 & en\ otro\ caso\ (\beta_A \le \gamma_B + \mu) \end{cases} \quad (2.14)$$

- **Comparadores Difusos MLT/NMLT (possibly/Necessarily Much Less Than, Posiblemente/Necesariamente Mucho Menor difuso):**

$$CDEG(A\ MLT\ B) = \begin{cases} 1 & si\ \beta_A \le \alpha_B - \mu \\ \dfrac{\beta_B - \mu - \alpha_A}{(\beta_A - \alpha_A) - (\alpha_B - \beta_B)} & si\ \beta_A > \alpha_B - \mu\ y\ \alpha_A < \beta_B - \mu \\ 0 & en\ otro\ caso\ (\alpha_A \ge \beta_B - \mu) \end{cases} \quad (2.15)$$

$$CDEG(A\ NML\ T\ B) = \begin{cases} 1 & si\ \delta_A \le \alpha_B - \mathrm{M} \\ \dfrac{\beta_B - \mathrm{M} - \gamma_A}{(\delta_A - \gamma_A) - (\alpha_B - \beta_B)} & si\ \delta_A > \alpha_B - \mathrm{M}\ \ y\ \ \gamma_A < \beta_B - \mathrm{M} \\ 0 & en\ otro\ caso\ (\gamma_A \ge \beta_B - \mathrm{M}) \end{cases} \quad (2.16)$$

2.2.1.3. Restrictividad de los Comparadores Difusos

Hay comparadores cuyos resultados de comparación incluyen unos a otros. Por ejemplo, en modo crisp, el resultado del comparador ">=", incluye al de ">". Decimos entonces que el comparador ">" es más restrictivo que ">=". O sea, los comparadores más restrictivos recuperarán menor o igual cantidad de tuplas y las tuplas recuperadas nunca tendrán un grado de cumplimiento mayor que con los comparadores menos restrictivos.

En la Tabla 2.7 se puede ver el orden de restrictividad de los comparadores difusos, por familias. Observe que cualquier comparador de necesidad es más restrictivo que su correspondiente comparador de posibilidad.

Familia	Comparadores difusos, de mayor a menor restrictividad
Igual difuso	NFEQ > FEQ
Mayor difuso	NFGT > FGT > NFGEQ > FGEQ
Menor difuso	NFLT > FLT > NFLEQ > FLEQ
Mucho Mayor difuso	NMGT > MGT
Mucho Menor difuso	NMLT > MLT

Tabla 2.7. Restrictividad de los comparadores difusos por familias.

2.2.1.4. Definición del Comparador Difuso FEQ para Atributos Difusos Tipo 3

Como ya hemos indicado, para atributos difusos **Tipo 3** sólo está permitido usar el comparador difuso FEQ, ya que no existe una relación de orden en el dominio de estos atributos. Por ejemplo, no podemos decir si el valor "Rubio" es mayor que el valor "Moreno", pero si podemos medir su similitud. Así, cuando comparamos dos valores de tipo simple entre sí, el valor devuelto es el valor almacenado en su relación de similitud (tabla FUZZY NEARNESS DEF) suponiendo que ambos valores estén normalizados, i.e., tengan grado de posibilidad 1.

Lo normal (y deseable) es que tanto los valores simples como las distribuciones de posibilidad sobre atributos difusos Tipo 3 estén normalizadas, i.e., tengan grado de posibilidad 1 al menos en alguna componente. Sin embargo, si eso no ocurriera el Servidor FSQL lo tiene en cuenta en la comparación.

2.2.1.5. Tipos de Condiciones Difusas Elementales

Por todo lo visto, podemos concluir que existen tipos básicos de comparaciones o condiciones difusas elementales (sin expresiones aritméticas externas). Estas condiciones pueden ser enlazadas con otras condiciones (difusas o no) mediante los operadores lógicos (NOT, AND y OR) para formar condiciones difusas más complejas.

Sean <fcol> y <fcol2> dos atributos difusos expresados con el formato de SQL (con su esquema y tabla o no) y <fcomp> un comparador difuso de los definidos en la Tabla 2.4. Entonces, los tipos básicos de condiciones difusas son los siguientes:

- Comparación de un atributo difuso Tipo 1 ó 2 con una distribución de posibilidad constante de tipo aproximado #n (como la Figura 2.4):

  <fcol> <fcomp> #n

- Comparación de un atributo difuso Tipo 1 ó 2 con una distribución de posibilidad constante de tipo intervalo [n,m] (como la Figura 2.5):

  <fcol> <fcomp> [n,m]

- Comparación de un atributo difuso Tipo 1 ó 2 con una distribución de posibilidad constante de tipo trapecio $[$\alpha$,$\beta$,$\gamma$,$\delta$] (como la Figura 2.2):

  <fcol> <fcomp> $[$\alpha$,$\beta$,$\gamma$,$\delta$]

- Comparación de un atributo difuso de cualquier Tipo (1, 2 ó 3) con una distribución de

posibilidad constante de tipo etiqueta lingüística $label (definida en la FMB):

<fcol> <fcomp> $label

- Comparación de un atributo difuso Tipo 1 ó 2 con una expresión constante de tipo crisp <expr_crisp>:

  <fcol> <fcomp> <expr_crisp>

  En la expresión crisp también puede incluirse una columna crisp o difusa Tipo 1 de cualquier tabla.

- Comparación de un atributo difuso de cualquier Tipo con otro que sea compatible con el primero:

  <fcol> <fcomp> <fcol2>

  La única incompatibilidad que existe entre atributos difusos está entre atributos de Tipo 3 con atributos de otro tipo y con otros atributos de Tipo 3 que no sean declarados explícitamente como compatibles en la tabla FUZZY_COMPATIBLE_COL.
  Si el atributo de la derecha es difuso Tipo 2 ó 3 y tras éste aparecen operaciones aritméticas, estas serán consideradas como externas a la comparación difusa.

- Comparación de un atributo difuso con una constante difusa sin argumentos variables usando el comparador difuso FEQ:

$$< \text{fcol} > \text{FEQ} \begin{cases} \text{UNKNOWN} \\ \text{UNDEFINED} \\ \text{NULL} \end{cases}$$

  En este tipo de comparación no puede emplearse otro comparador difuso distinto de FEQ y la columna <fcol> no puede ser difusa Tipo 1.

- Comparación de un atributo difuso con una constante difusa sin argumentos variables

usando la comparación con IS (con o sin NOT):

$$< \text{fcol} > \text{IS} \ [\text{NOT}] \ \begin{cases} \text{UNKNOWN} \\ \text{UNDEFINED} \\ \text{NULL} \end{cases}$$

Si la columna <fcol> es difusa Tipo 1 sólo puede usarse la constante NULL, que tendrá el sentido que le da el SGBD y no es el NULL difuso.

A todas estas condiciones difusas, menos a la última, se le pueden poner un umbral de cumplimiento, de forma que la condición establecida sólo será válida si su grado de cumplimiento supera dicho umbral.

Las comparaciones difusas (excepto IS) pueden ir precedidas o sucedidas por expresiones aritméticas que operan con la comparación difusa, de forma que estas son consideradas como externas a la comparación difusa. El Servidor FSQL limita las expresiones aritméticas después de la comparación difusa de forma que éstas sólo pueden aparecer:

- Cuando queremos comparar una expresión crisp (caso 5): Entonces, toda la expresión debe ser crisp y situada a la derecha del comparador difuso. La comparación difusa se efectuará sobre toda la expresión. A la izquierda debe haber, lógicamente, un atributo difuso Tipo 1 ó 2.
- Cuando justo tras el comparador aparece un atributo difuso Tipo 2 ó 3: Entonces, las expresiones aritméticas que sucedan a ese atributo difuso serán consideradas como externas a la comparación, ya que el Servidor FSQL no admite operaciones aritméticas sobre atributos difusos Tipo 2 ó 3.

2.3. Arquitectura del Servidor FSQL

La Base de Datos Relacional Difusa (BDRD) y el Servidor FSQL [3, 4] han sido implementados, como ya se ha dicho, usando un SGBD ya existente, Oracle [16].

Básicamente, esto implica tres consecuencias:

- El sistema será, quizás, más lento que si se hubiera programado a bajo nivel.
- La tarea de implementación se hace más simple ya que no necesitamos programar el SGBD.
- Obtenemos todas las ventajas del SGBD anfitrión (seguridad, eficiencia...) sin que tengamos que tener en cuenta esos importantes detalles.

El SGBD elegido fue Oracle por su gran versatilidad, su popularidad (está implantado en multitud de empresas y entidades) y por la posibilidad que ofrece para crear paquetes (con procedimientos y funciones) internos al sistema. Estos paquetes son creados en un lenguaje propio, **PL/SQL** [16], que es muy eficiente en accesos a la base de datos. Por supuesto, esta arquitectura puede ser implementada en otros SGBD.

La arquitectura de la BDRD con el Servidor FSQL está compuesta por:

- **Datos**: Base de datos tradicional y Base de Metaconocimiento Difuso, FMB (Fuzzy Meta-knowledge Base).

- **Servidor FSQL** (versión 1.2): Encargado de ocultar el procesamiento difuso al usuario.

- **Cliente FSQL**: Encargado de hacer de interfaz entre el usuario y el Servidor FSQL. Es el objeto del presento proyecto.

A continuación explicamos cada uno de estos componentes:

2.3.1. Datos: Base de Datos Tradicional y FMB

Los datos pueden ser clasificados en dos categorías: La base de datos tradicional y la base de metaconocimiento difuso o FMB.

La base de datos tradicional está compuesta por los datos de nuestras relaciones con un formato especial para almacenar atributos difusos.

La base de metaconocimiento difuso, FMB (Fuzzy Meta-knowledge Base) almacena información sobre la BDRD en un formato relacional. En la FMB se almacena una lista con los atributos que admiten tratamiento difuso su Tipo de atributo difuso (1, 2 ó 3). Además, para cada atributo difuso se almacena distinta información dependiendo de su Tipo difuso.

2.3.2. Servidor FSQL

Como ya se ha dicho, ha sido programado íntegramente en PL/SQL, e incluye tres tipos de funciones:

- **Función de Traducción** (FSQL2SQL): Esta función, incluida en el paquete FSQL_PKG, efectúa un análisis léxico, sintáctico y semántico de la consulta FSQL. Si encuentra errores de cualquier naturaleza, generará una tabla con todos los errores encontrados. Si no hay errores, la consulta FSQL es traducida a una sentencia en SQL. La sentencia resultante en SQL podrá incluir llamadas a los siguientes tipos de funciones.
- **Funciones de Representación**: Estas funciones son utilizadas para mostrar los atributos difusos de manera que sean comprensibles por el usuario, evitando así, el críptico formato interno de estos atributos.
- **Funciones de Comparación Difusa**: Se utilizan para comparar atributos y valores difusos y para calcular los grados de compatibilidad que devuelve la función CDEG.

Lo que hace la función de Traducción, es reemplazar los atributos difusos del SELECT por llamadas a las funciones de Representación, las condiciones difusas por llamadas a las funciones de Comparación Difusa y reemplazar las llamadas a la función CDEG por llamadas a las funciones de Comparación Difusa y otras funciones si existen operadores lógicos involucrados en la condición del atributo argumento de CDEG. Las funciones por defecto para los distintos operadores lógicos se muestran en la Tabla 2.5.

2.3.3. Cliente FSQL

Es un programa independiente que sirve de interfaz entre el usuario y el Servidor FSQL. El usuario introduce la consulta FSQL (de forma directa o indirecta) y el programa Cliente se comunica con el Servidor y con la base de datos para obtener los resultados deseados.

La principal función que usará directamente el Cliente será la función de Traducción del Servidor FSQL. Sin embargo, en el paquete FSQL PKG existen otras funciones útiles que pueden ser también usadas por el Cliente FSQL.

La estructura de un programa Cliente FSQL y todo lo relacionado con él será explicado en detalle en el Capítulo 5.

2.3.4. Funcionamiento del Servidor FSQL

El proceso de llamada del Servidor FSQL está esquematizado en la Figura 2.8. Resumiendo, para una consulta FSQL se ejecutan los siguientes pasos:

1. El programa Cliente FSQL envía la consulta FSQL al Servidor FSQL.
2. El Servidor FSQL analiza la consulta y, si es correcta, genera una sentencia SQL a partir de la consulta original en FSQL. En este paso el Servidor FSQL usa la información de la FMB.
3. Una vez ha sido generada la consulta en SQL, el programa Cliente leerá dicha consulta. Si en el paso 2, el Servidor FSQL encontró errores, entonces, en este paso se leerán dichos errores, para ser mostrados al usuario.
4. El programa Cliente enviará la consulta SQL a cualquier base de datos que sea coherente con la FMB. Para la ejecución de esta consulta podrán usarse diversas funciones del Servidor FSQL (de Representación y Comparación Difusa), pero eso es trasparente al usuario.
5. Finalmente, el Cliente recibirá los datos resultantes y los mostrará al usuario. El formato de presentación dependerá, lógicamente, del programa Cliente FSQL empleado.

Los pasos 3 y 4 podrían haber sido eliminados para incrementar la eficiencia pero, de esta forma conseguimos una independencia entre la fase de Traducción (pasos 1, 2 y 3) y la fase de Consulta (pasos 4 y 5). Así, podemos usar una base de datos local con el Servidor FSQL y la FMB para traducir nuestras sentencias localmente y depurar los errores, y enviar las consultas traducidas a una base de datos remota, evitando así sobrecargar la red de comunicaciones con mensajes de error, consultas traducidas, etc... De esta forma, la base de datos remota podría no tener instalada la función de Traducción, aunque si requeriría tener instaladas las funciones de representación y comparación difusa.

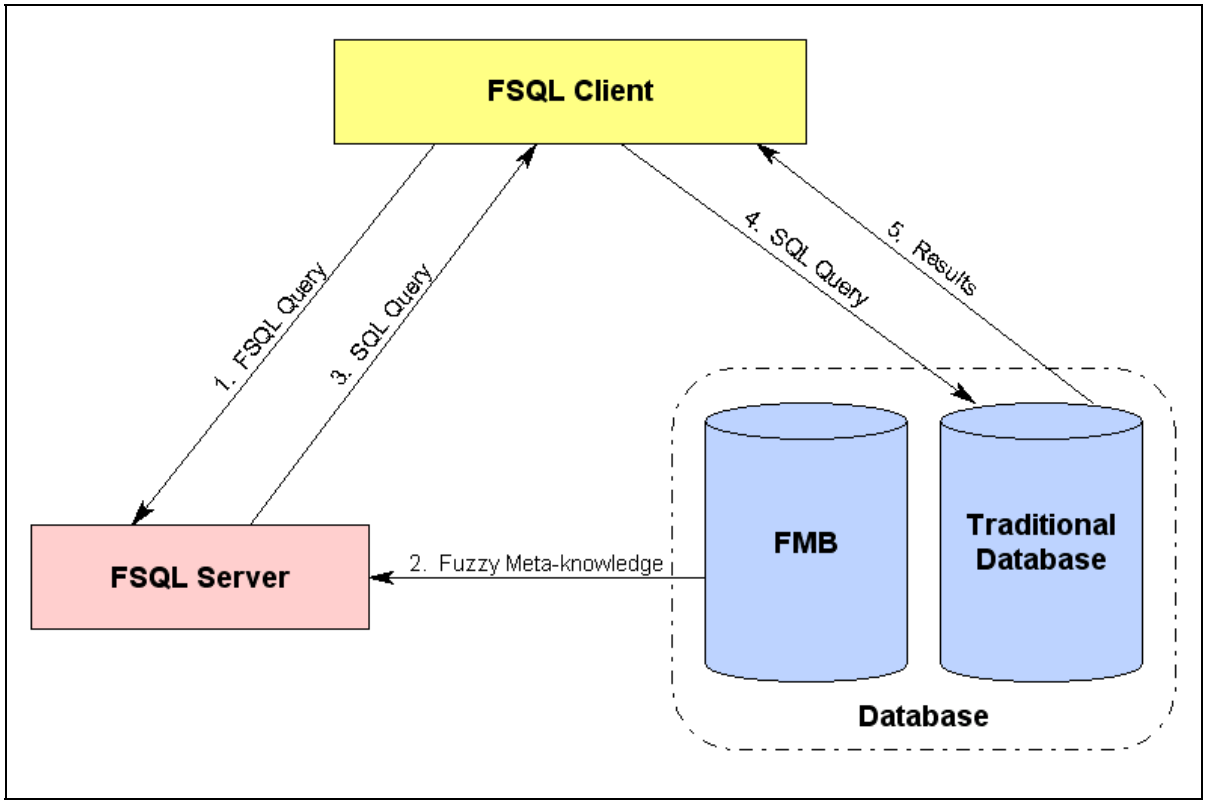


Figura 2.8. Arquitectura básica para la BDRD con el Servidor FSQL.

2.4. El Servidor FSQL

El objetivo del Servidor FSQL es conseguir traducir una sentencia FSQL a una sentencia en SQL, mediante llamadas a funciones del Servidor. Para ello, se han creado distintos paquetes PL/SQL de forma que cada parte del Servidor está en un paquete distinto. En la Tabla 2.8 puede verse una lista de los paquetes del Servidor FSQL y un resumen de su

contenido y utilidad. El paquete FSQL_PKG es el único que contiene funciones utilizables por el usuario o por el programa Cliente FSQL.

Nombre Paquete	Fichero	Contenido
FSQL_PKG	FSQL_PKG.sql	Analizador Léxico y Sintáctico y otras funciones útiles FSQL2SQL, FSQL_FIN...
FSQL_SEMANTIC	FSQL_SEM.sql	Analizador Semántico y conversor del Servidor FSQL
FSQL_AUX	FSQL_AUX.sql	Funciones útiles para el Servidor
FSQL_FUNCTIONS	FSQL_F.sql	Funciones de representación y comparación difusa
FSQL_FUNCTIONS2	FSQL_F2.sql	Más funciones de comparación difusa

Tabla 2.8. Paquetes PL/SQL del Servidor FSQL y resumen de su contenido.

Básicamente, desde un punto de vista conceptual el Servidor FSQL consta de los siguientes módulos:

1. **Analizador Léxico**: Se encarga de analizar si la sentencia FSQL es correcta léxicamente, generando una lista con los tokens (palabras) de la sentencia.
2. **Analizador Sintáctico**: Analiza si la sentencia es correcta sintácticamente. Para ello utiliza una gramática que genera sentencias que admiten las extensiones de FSQL.
3. **Analizador Semántico y Conversor**: Analiza si la sentencia es correcta desde el punto de vista semántico y, a la vez, va generando la sentencia SQL equivalente. La función encargada de lanzar el proceso global es la función FSQL_PKG.FSQL2SQL.
4. **Funciones de Representación y Comparación Difusa**: Estas funciones son utilizadas para mostrar los atributos difusos de manera que sean comprensibles por el usuario, para comparar atributos y valores difusos y para calcular los grados de compatibilidad que devuelve la función CDEG.

Para llevar a cabo todas estas tareas el Servidor FSQL utiliza unas tablas que son creadas al instalar el Servidor. Una lista de estas tablas con un resumen de su utilidad y de sus características puede verse en la Tabla 2.9.

Nombre Tabla/Vista	Contenido
T.RESERVADAS	Palabras Reservadas del lenguaje FSQL (Apéndice A)
T.T_TRANSI	Transiciones del Autómata del Analizador Léxico
T.TABLA_SINTAX	Transiciones de la Gramática LL(1) para el Analizador Sintáctico
T.PRODUCCIONES	Producciones de la Gramática para el Analizador Sintáctico
V.ACCESSIBLE_TABLES	Tablas y Vistas accesibles por el usuario
T.FSQL_ALL_ERRORS	Mensajes de Error FSQL de todos los usuarios
V.FSQL_ERRORS	Mensajes de Error FSQL del usuario particular
T.FSQL_ALL_QUERIES	Consultas FSQL y su traducción SQL de todos los usuarios
V.FSQL_QUERY	Consultas FSQL y su traducción SQL del usuario particular
T.FSQL_ALL_INFO	Información y opciones de configuración de todos los usuarios
V.FSQL_INFO	Información sobre el Servidor FSQL para todos los usuarios
V.FSQL_OPTIONS	Opciones de configuración modificables por cada usuario
T.FSQL_STATS	Control de accesos para estadísticas

Tabla 2.9. Tablas y Vistas del Servidor FSQL y resumen de su contenido.

Capítulo 3

Desarrollo de Aplicaciones Web

En los apartados de este capítulo, se explicarán los principales conceptos relacionados con la evolución de las técnicas y herramientas necesarias para la creación de un sitio web. Después se diferencian las propiedades y utilidades que presentan las páginas web estáticas, activas en el cliente y activas en el servidor, esto se explicará en el apartado 3.4 y siguientes, resaltando las ventajas e inconvenientes de cada uno de estos tipos de tecnologías.

También estudiaremos los servidores y componentes software utilizados para la administración de las aplicaciones web, se explicará en el apartado 3.7.

3.1. Evolución de Internet

Hoy en día, la cantidad de personas que utilizan Internet es enorme, y su número crece sin parar, circunstancia que propicia una verdadera revolución en el acceso a la información. Por otra parte, y desde un punto de vista técnico, este elevado crecimiento fomenta el desarrollo de nuevas tecnologías que satisfagan las necesidades que demandan los usuarios.

Actualmente nos encontramos inmersos en un ciclo evolutivo muy dinámico en el que conviene dominar las últimas herramientas software que nos invaden, especialmente si se orientan a la creación de páginas y aplicaciones web profesionales y están desarrolladas por empresas con el suficiente peso específico como para influir en el mercado. Este es el caso de las **ASP** (*Active Server Pages*, Páginas Activas de Servidor) [2] creadas por Microsoft y que se explicarán en el apartado 3.6.

3.2. Creación de Aplicaciones Web

Una vez que los sitios web basados en páginas estáticas se han extendido a través de Internet, se constata la necesidad de dotar de mecanismos de programación a las páginas **HTML** (*Hipertext Markup Language*, Lenguaje de Marcas de Hipertexto) [11] con el fin de conseguir dinamismo e interacción entre los usuarios y los servidores web. De esta manera, el desarrollo de aplicaciones web profesionales conlleva la utilización de tecnologías que permiten recoger información proveniente de clientes y de bases de datos, utilizando programas que funcionan en Internet e Intranet.

Veremos como vamos a diseñar y programar un sitio web profesional basado en la tecnología creada por Microsoft para el desarrollo de aplicaciones web: ASP. Con las páginas activas en el servidor vamos a poder crear páginas web con contenidos dinámicos.

Las páginas activas en el servidor presentan las siguientes ventajas:

- Son fáciles de utilizar
- Funcionan adecuadamente en todos los navegadores (Explorer, Netscape, etc.)
- Proporcionan un acceso muy simple a Bases de Datos (Oracle, SQL Server, Access, etc.)
- Su ejecución es muy eficiente (lo que permite utilizar servidores económicos)
- Simplifican el desarrollo y mantenimiento de las aplicaciones web
- Permiten la utilización e integración de productos y tecnologías de Microsoft como ActiveX, **ADO** (ActiveX Data Object, Objeto de Datos ActiveX), **FSO** (File System Object, Objeto Sistema de Ficheros), etc.
- ASP se encuentra integrado en el soporte estándar de comunicaciones de Windows 2000: **IIS** (Internet Information Services, Servicios de Información de Internet).

Con el objetivo de simplificar lo máximo posible las explicaciones y facilitar el proceso de desarrollo de las páginas web dinámicas, nos vamos a centrar en el uso de un entorno concreto de trabajo: Windows e IIS (*Internet Information Services*). Todas las herramientas y tecnologías que se explicarán pueden utilizarse desde Windows 98/ME y Windows 2000/NT/XP.

La base fundamental de este proyecto se dedica al acceso a bases de datos en aplicaciones web, de esta manera vamos a ver claramente como crear una aplicación web profesional que permita acceder con facilidad a una base de datos y como integrar esta información en las páginas web dinámicas diseñadas.

La estructura de este capítulo la vamos a presentar como seis bloques:

1. El primer bloque se centrará en un breve repaso de creación de páginas web estáticas mediante el uso de **HTML**.
2. El segundo proporciona una base para comprender conceptos importantes relacionados con las páginas web estáticas y dinámicas.
3. El tercer bloque explica al desarrollo de contenidos web dinámicos con tecnología **ASP** (Active Server Pages).
4. El cuarto bloque es una introducción a los servidores de contenidos web dinámicos, y en concreto se habla del soporte de servicios de comunicaciones **IIS** (Internet Information Services) de Microsoft.
5. En el quinto bloque se hace una referencia a los visualizadores o navegadores cliente para el acceso a contenidos web.
6. En el último bloque nos centraremos en una pequeña introducción a la Programación Internet-Intranet: Modelo cliente/servidor.

A pesar de la variedad de temas y conceptos que se tratan en este proyecto, se ha optado por un enfoque eminentemente práctico, se explicarán las tecnologías a utilizar en cada uno de los componentes del sistema con explicaciones detalladas.

3.3. El Lenguaje HTML

Bien, ya tenemos un servidor web preparado para que cualquier usuario Internet-Intranet pueda acceder a la información que deseemos mostrar y publicar. Pero claro, ¿cuál es el formato adecuado? ¿documentos Word, Excel, PDF...? Al fin y al cabo, los usuarios de Internet pueden acceder a nuestro web desde plataformas muy heterogéneas: Windows 98 y/o 2000, Macinstosh, sistemas Unix y X-Windows, Linux, etc... Afortunadamente esta incógnita está resuelta: la solución es HTML.

HTML *(HiperText Markup Language)* [11] es el lenguaje adoptado por la comunidad Internet como estándar para la creación y publicación de documentos en la World Wide web (WWW). La gran ventaja que nos proporciona HTML es que cualquier usuario de Internet, a través de cualquier navegador (y desde cualquier plataforma), puede recibir y visualizar los documentos creados con este lenguaje. Este tipo de documentos, como es sabido, son comúnmente denominados páginas HTML.

HTML nos permite editar documentos que contengan además de texto, elementos multimedia (imágenes, animación, sonido, etc.), controles de entrada de datos (texto, botones, listas...), soporte para lenguajes de script (lo que posibilita crear documentos con los que puede interactuar el usuario) y elementos de hiperenlace, que facilitan la navegación y el acceso a otras páginas HTML.

Pero, ¿cómo puede acceder un usuario de Internet a un determinado documento, imagen, vídeo o programa en nuestro servidor web? Todos los recursos en la web están identificados por un **URI** (*Universal Resource Identifier*, Identificador Universal de Recurso), un URI define o especifica una entidad (y puede ser de cualquier tipo: documento, dispositivo, vídeo, audio, imagen, etc...) y consta de tres partes:

- Nombre del mecanismo utilizado para acceder al recurso.
- Nombre del servidor que alberga el recurso.
- Nombre del recurso, incluido en camino (path) para llegar a él.

No debemos confundir URI con **URL** (*Universal Resource Locator,* Localizador Universal de Recurso), el cual es un subconjunto del esquema más general especificado por URI. Las URLs, o sea, lo que todos conocemos como la dirección de una página web (por ejemplo, **http://www.visualfsql.com**), es el tipo más familiar de URI. Una URL es una dirección única que nos permite visitar una página. Por ejemplo la dirección URI:

http://www.maquina.com/directorio/documento.extension

Se leería como sigue: Hay un documento disponible mediante protocolo HTTP residiendo en la máquina www.maquina.com, accesible en la ruta /directorio/documento.extensión.

Para profundizar un poco más vamos adentrarnos en la sintaxis del lenguaje HTML. En la Figura 3.1 mostramos el contenido de un documento HTML que, en un navegador Internet, muestra el texto "Hola Mundo".

```
<HTML>
<HEAD>
<TITLE>Mi primer documento</TITLE>
</HEAD>
<BODY>
<P>Hola Mundo
</BODY>
</HTML>
```

Figura 3.1. Documento HTML

HTML está construido alrededor de lo que se denominan tags (etiquetas). Podríamos decir que son instrucciones que formatean el texto incluido dentro del documento. De esta manera, indican al navegador el modo de representar el texto de la página, así como los gráficos, sonido y vídeo, por ejemplo. Los tags son fácilmente reconocibles porque en el documento aparecen delimitados por los caracteres menor que (<) y mayor que (>). La mayoría de ellos siempre aparecen por parejas envolviendo el texto que quieren formatear, un tag de apertura de la instrucción, el texto y por último el que cierra la instrucción, el cual es marcado con una barra (/).

Como vemos en la Figura 3.1, todo documento HTML debe comenzar por el tag <HTML> y finalizar con </HTML>, y está compuesto de dos secciones bien diferenciadas: la cabecera y el cuerpo. La cabecera, delimitada por el tag <HEAD> </HEAD> contiene el título del documento, el cual a su vez está delimitado por los tags <TITLE> </TITLE>. Podemos observar en la Figura 3.1 cómo este título aparece en la barra de título de la ventana del navegador. En el cuerpo, delimitado por <BODY> </BODY> aparece el contenido del documento. En el ejemplo mostrado por la Figura 3.1, el cuerpo sólo contiene un párrafo (marcado con <P>) con el texto "Hola Mundo".

A su vez, los tags están construidos alrededor de atributos, los cuales permiten indicar información adicional necesaria para completar la instrucción. Por ejemplo, el tag <FONT>, utilizado para asignar el tipo de letra del texto, tiene los atributos face que indica el aspecto del tipo de letra (Arial, Times New Roman, Courier, etc.) y size, para señalar el tamaño.

```
<font face="Arial" size="1" color="#FFFF00">Hola Mundo</font>
```

Existe también un carácter de control especial, el ampersand (&), que permite al navegador mostrar caracteres que no forman parte del código ASCII estándar. Por ejemplo, para mostrar el carácter "á", debemos usar "&aacute" en nuestro texto, esto se escribe así para guardar compatibilidad con el juego de caracteres de los primeros navegadores, aunque hoy en día la mayoría de éstos lo muestran correctamente si se escribe directamente el carácter "á".

No nos interesa en este momento profundizar demasiado en la sintaxis del lenguaje HTML, ya que existe abundante bibliografía al respecto y no es el objetivo del presente proyecto, aún así, la Tabla 3.1 proporciona una breve referencia a los tags más comúnmente utilizados en la creación de documentos HTML.

TAG	SIGNIFICADO	EJEMPLO
<B>	Texto en negrita	<B>Texto en Negrita</B>
<I>	Texto en cursiva	<I>Texto en Cursiva</I>
<U>	Texto subrayado	<U>Texto subrayado</U>
<P>	Nuevo Párrafo	<P>Párrafo 1
 	Salto de Línea	
<FONT>	Tipo de Letra	<FONT face="Arial" size="5">Texto grande</FONT>
<IMG>	Imagen o gráfico	<IMG src="gráfico1.gif">
<A>	Enlace a otro documento	<A href="pagina2.htm">Ir a Página 2</A>
<TABLE>	Definición de Tabla	<TABLE border="2">
<TR>	Una fila en una tabla	<TR><TD>Celda(1,1)</TD><TD>Celda(1,2)</TD></TR>
<TD>	Una columna en una tabla	<TR><TD>Celda(2,1)</TD><TD>Celda(2,2)</TD></TR>
		</TABLE>
<FORM>	Formulario entrada de datos	<FORM name="form1" method="POST" action="graba.asp">
<INPUT>	Campo de entrada de datos	Nombre:<INPUT type="text" name="nombre">
		<INPUT type="submit">
		</FORM>

Tabla 3.1. Resumen de etiquetas HTML.

Sólo voy a incidir en el hecho de que HTML está diseñado para crear documentos independientes de la plataforma y esto hace que cualquier navegador Internet sea capaz de interpretar y mostrar su contenido, cumpliendo con los requisitos de diseño con los que fue concebido por el autor de dichos documentos.

Ahora bien, por lo que estamos viendo, la creación de documentos HTML no parece, a priori, una tarea muy agradable, ya que conocer todos los tags con sus correspondientes atributos y su correcta aplicación exige un esfuerzo difícil de asumir, al menos de manera voluntaria. Tiene que ser más sencillo que todo esto.

También debemos mencionar una característica importante del lenguaje HTML y es que aunque se escriba algún tag de forma incorrecta, el Navegador nunca genera un error de HTML, sino que intenta mostrar la página lo mejor que puede.

Hoy en día existen herramientas para diseñar, componer y editar páginas HTML de forma visual (a golpe de ratón) y guardar los resultados automáticamente con la sintaxis HTML correcta. Entre muchas de estas herramientas, vamos a destacar tres: *Macromedia Dreamweaver MX, Microsoft Visual Interdev 6.0 y Microsoft FrontPage*.

3.4. Páginas Web Estáticas y Dinámicas

Paralelamente a la línea de avances mencionada, nos encontramos con una evolución en los servicios que la red ha ido ofreciendo, así como en las herramientas software que los sustentan. El uso más común que se le ha dado a Internet es el de publicación de información, y es precisamente en este aspecto donde mayores facilidades software se están proporcionando. Actualmente **FTP** (*File Transfer Protocol*, Protocolo de Transferencia de Ficheros) y WWW son los servicios más utilizados.

Las páginas web son textos ASCII escritos en el lenguaje HTML, que se transfieren entre los servidores de WWW y los exploradores de los clientes mediante el protocolo **HTTP** (*HyperText Transfer Protocol,* Protocolo de Transferencia de Hipertexto). La investigación y desarrollo en el campo de la creación y mantenimiento de páginas web es uno de los más

dinámicos en el ámbito de Internet, debido a sus consecuencias comerciales y de utilización de la red.

Inicialmente las páginas web era estáticas, en el sentido de que, a efectos de usuario, el único proceso realizado era el de la visualización de sus contenidos (escritos en lenguaje HTML) por parte del explorador del cliente. Las páginas estáticas se siguen utilizando ampliamente debido a que forman la base necesaria para la presentación de datos en muchos tipos de situaciones. También influye decisivamente la sencillez con que se pueden crear, instalar y mantener.

En el momento en que se requiere una interacción mayor entre los usuarios y el sistema que soporta las páginas web, surge la necesidad de reunir y procesar las peticiones del cliente con el fin de ofrecerle informaciones mejor dirigidas, escogidas y elaboradas, y en este punto hay que destacar la especial importancia de esta característica para el proyecto en cuestión.

Por ejemplo, en una universidad que proporcione las notas de los alumnos a través de las páginas web de cada departamento, un estudiante podría conseguir sus calificaciones navegando por cada departamento y asignatura entre una gran cantidad de páginas web estáticas relacionadas jerárquicamente. Una alternativa más elegante y sencilla para el alumno sería preguntarle sus datos en un formulario y ejecutar una aplicación en el servidor que seleccionara sus calificaciones entre las distintas asignaturas en las que se ha matriculado, para por fin, presentarle todas las calificaciones obtenidas a raíz de la petición sencilla y directa realizada por el usuario.

El ejemplo anterior ilustra la conveniencia de dotar de procesamiento al intercambio de información entre los usuarios y el servidor de páginas web, con el que introducimos el concepto de páginas web activas o dinámicas que el modelo *ASP* (*Active Server Pages*) soporta.

La capacidad de procesamiento que sustenta las páginas dinámicas se puede llevar a cabo siguiendo alguno de estos modelos:

- Procesamiento en el equipo del usuario (Páginas Activas en el cliente, Javascript y VBScript)

- Procesamiento en el equipo Servidor web (Páginas Activas en el Servidor, ASP y VBScript)
- Procesamiento Mixto (Páginas activas en el cliente y páginas activas en el Servidor)

En el caso particular de este proyecto se ha optado por la tercera opción, puesto que es la más completa y la que más potencia y funcionalidad proporciona.

La ventaja principal de las páginas activas en el cliente es la descarga de trabajo que le proporciona al equipo servidor la posibilidad de traspasar cómputos a los usuarios. Imaginemos una gran entidad bancaria que proporciona páginas web capaces de calcular amortizaciones de capital y evolución de intereses de préstamos según las condiciones particulares de consulta de cada cliente. ¿Qué tipo de ordenador sería necesario para soportar decenas de accesos simultáneos en paralelo a otras posibles aplicaciones? Desde luego supondría una gran descarga traspasar la ejecución del programa a los equipos clientes.

Otra importante ventaja se basa en el ahorro de comunicaciones (ancho de banda) que se puede experimentar en muchas aplicaciones que ejecutan procesos en el cliente, y evitan de esta manera realizar continuos traspasos de información con el servidor web. Un ejemplo muy sencillo de esta situación en la creación de una página web 'calculadora' que con una sola página activa en el cliente puede funcionar de forma autónoma (una vez cargada) en el equipo del usuario. De otra manera se necesitaría realizar un continuo traspaso de datos (pulsaciones) del equipo cliente al servidor, y de resultados (operaciones realizadas) de servidor al cliente. Esta situación refleja uno de los principios existentes en el campo de las comunicaciones, que establece una relación inversa entre capacidad de cómputo disponible y ancho de banda necesario.

A pesar de las innegables ventajas de la distribución del cómputo, existen diversas razones que nos llevan a crear páginas activas en el servidor. A continuación de detallan algunas de ellas.

Existe información de naturaleza centralizada. El ejemplo de las calificaciones de los alumnos en una universidad forma parte de los casos de este tipo. Aunque se pueden realizar diseños de páginas web que minimicen el ancho de banda necesario para las consultas o el procesamiento del servidor, los datos y una buena parte de los cómputos no pueden ser

eliminados del lugar 'central'. Distribuir la información de un servidor a varios servidores (por ejemplo uno por departamento) no significa en absoluto traspasar datos y procesamiento al cliente.

Las páginas activas en el cliente se basan actualmente en tecnologías muy dependientes del explorador y la plataforma del usuario, de esta manera, hoy en día, un explorador de Netscape no interpreta aplicaciones (*scripts*) de Visual Basic, solamente JavaScripts, mientras que el Explorer de Microsoft interpreta tanto VBScript como una versión de Javascript denominada JScript. Los ActiveX sólo funcionan con Windows, etc.

Aunque los usuarios cuenten con la plataforma y el explorador adecuados, no siempre están dispuestos a introducir componentes ejecutables (Applets y ActiveX fundamentalmente) en sus equipos, lo que disminuye considerablemente la potencia de las páginas activas en el cliente.

3.5. Modelos de Funcionamiento con Soporte Dinámico en el Servidor

Tradicionalmente, en los servidores web se ha utilizado el mecanismo **CGI** (*Common Gateway Interface,* Entorno de pasarela común) para implementar páginas web activas en el servidor. Los lenguajes PERL y C han sido muy empleados, aunque se podrían usar muchos otros con este propósito.

El funcionamiento básico de un programa CGI se basa en:

1. Lectura de datos provenientes de un formulario situado en una página web.
2. Procesamiento de la información, lo que puede llevar incluido el acceso a bases de datos.
3. Escritura de las páginas HTML de respuesta. Esto conlleva la introducción de numerosas sentencias de escritura de códigos HTML.

Aunque este método es conceptualmente sencillo, presenta graves inconvenientes. Sus principales desventajas son:

- Resulta difícil mantener las páginas web que se le devuelvan al cliente, debido a que las instrucciones HTML se encuentran insertadas en el propio código del programa CGI, mezclándose el código HTML dentro del código de servidor, lo cual es difícil de mantener cuando se requieran cambios en la página.

- La ejecución del programa CGI es muy ineficiente, debido al proceso de carga del código en memoria que se realiza cada vez que un usuario requiere su ejecución por medio de la página web que lo invoca. La existencia de múltiples clientes simultáneos supone múltiples copias del programa en memoria del servidor (con el coste añadido de todas estas cargas). Este funcionamiento resulta obsoleto y además compromete el funcionamiento de las máquinas que soportan servidores web muy utilizados.

- La respuesta tecnológica que se planteó a la ineficacia y desperdicio de recursos de las soluciones CGI fue la creación de aplicaciones **ISAPI** (Internet Server API, API de Servidor de Internet) y la versión de Netscape para sus servidores web: **NSAPI** (Netscape Server API, API de Servidor Netscape).

- Una aplicación ISAPI es una DLL de Windows que se ejecuta en el mismo espacio de direcciones que el servidor web. Estas aplicaciones cuyo código ejecutable es compartido, pueden soportar las peticiones simultáneas de diversos clientes con una sola imagen en memoria. En definitiva, la realización de aplicaciones ISAPI le confiere al servidor un alto grado de eficiencia, lo que aumenta la seguridad en la capacidad del sistema para dar respuesta a peticiones simultáneas de varios usuarios.

- Desgraciadamente, la creación de aplicaciones ISAPI es costosa debido a su complejidad técnica, al tiempo necesario para realizar las compilaciones necesarias con el servidor web, a la dificultad para hacer las pruebas en una máquina que esté dando servicios en red, etc.

- Con ambos métodos (CGI, ISAPI) de realizar aplicaciones que se ejecutan en el servidor, se pueden emplear las facilidades que proporciona **IDC** (Internet Database Conector, Conector de Bases de Datos de Internet) para el acceso a bases de datos con

IIS y **ODBC** (Open DataBase Connectivity, Conectividad para Bases de Datos Abiertas), que se explicará en el siguiente capítulo.

Llegados a este punto nos debemos preguntar: ¿qué preferimos, un método de programación ineficaz pero sencillo (CGI), o bien uno eficaz pero complejo (ISAPI)? En muchos casos, por desconocimiento de las posibilidades, sencillez, rapidez o economía, la pregunta no se plantea en toda su crudeza y se diseñan páginas web estáticas. También es habitual optar por la creación de aplicaciones CGI porque no se espera un uso intensivo del servidor web por parte de los clientes y se puede tolerar la ineficacia de este método.

La tecnología ASP ha sido diseñada por Microsoft para facilitar la creación de sitios web con una sencillez mayor que la empleada en la programación CGI y con una eficiencia igual a la que proporciona ISAPI (el núcleo de funcionamiento de ASP es una aplicación ISAPI).

3.6. Páginas Activas en el Servidor con ASP

Active Server Pages (ASP) [2] es una nueva tecnología creada por Microsoft, destinada a la creación de sitios *web*. No se trata de un lenguaje de programación en sí mismo (ya que los ASP se pueden programar en VBScript, JavaScript, PerlScript o en varios otros lenguajes), sino de un marco sobre el que construir aplicaciones basadas en Internet.

La tecnología ASP apareció por primera vez (versión 1.0) con el servidor *Internet Information Server 3.0* de Microsoft en Diciembre de 1996. La versión 4.0 de IIS (el *Option Pack* para Windows NT 4.0) incluye la versión 2.0 de ASP, y la versión 5.0 de IIS, distribuida con Windows 2000, incluye ASP 3.0.

Los predecesores de ASP incluyen CGI y Perl. Las tecnologías de Microsoft predecesoras de ASP incluyen IDC y webDB. Otras tecnologías que compiten con ASP son JSP (*Java Server Pages)* de "Sun Microsystems", PHP de libre distribución bajo "Open System" y ColdFusion de "Allaire".

3.6.1. Características de ASP

Las principales características que ofrece ASP son:

- ASP es totalmente gratuito para Microsoft Windows NT/2000/XP/2003 o Windows 95/98/ME.
- El código ASP se puede mezclar con el código HTML en la misma página (no es necesario compilarlo por separado).
- El código ASP se puede escribir con un simple editor de textos como el Bloc de notas de Windows.
- Como el código ASP se ejecuta en el servidor, y produce como salida código HTML puro, su resultado es entendible por todos los navegadores existentes.
- Mediante ASP se pueden manipular bases de datos (consultas, actualizaciones, borrados, etc.) de prácticamente cualquier plataforma, con tal de que proporcione un driver OLEDB u ODBC (ver capítulo siguiente).
- ASP permite usar componentes escritos en otros lenguajes (C++, Visual Basic, Delphi), que se pueden llamar desde los scripts ASP.
- Sin modificar la instalación, los scripts ASP se pueden programar en JScript o VBScript (este último es el más usado porque más programadores lo dominan), pero también existen otros lenguajes, como Perlscript y Rexx, que se pueden emplear para programar ASP.
- Se ha portado a la plataforma Linux por Chili!Soft y Halcyon Software, lo que permite que ASP sea usado en casi cualquier sistema operativo, aunque aún no esta muy extendido.

3.6.2. Ventajas

Las principales ventajas que ofrece ASP son:

- Permite acceder a bases de datos de una forma sencilla y rápida.
- Las páginas se generan dinámicamente mediante el código de scripts (guiones).
- El código de script se ejecuta en el servidor, y no se depende del navegador que se

emplee.

- Desde una página ASP se pueden ejecutar servidores OLE en el servidor web, lo que abre un abanico de nuevas posibilidades sólo accesibles previamente con CGI y filtros ISAPI: acceso a base de datos/ficheros, logging en el sistema, envío de correo, etc.
- En ASP, todas las páginas web pueden ser diseñadas con editores de HTML, puesto que las instrucciones ejecutables y el código HTML están suficientemente delimitados. Así mismo, pueden utilizarse diversos lenguajes para la programación de la funcionalidad de las páginas activas. Entre estos lenguajes se encuentran Visual Basic Script (VBScript) y Java Script (JScript).
- Los desarrollos realizados con ASP no necesitan compilaciones que retarden el proceso de producción, y los errores de programación no provocan la caída del servidor web como es habitual en la programación ISAPI.
- Desde ASP se pueden realizar accesos a componentes ActiveX que se ejecutan en el servidor. De esta manera, por ejemplo, se hace un uso muy simple de ODBC para el acceso a distintos tipos de bases de datos sin necesidad de utilizar IDC.
- ASP permite compatibilizar la creación de páginas web activas en el cliente y en el servidor, pudiéndose así balancear la carga de proceso y de comunicaciones según los deseos del diseñador. También resulta posible utilizar diversos tipos de lenguajes de programación de scripts en una misma página, aunque esta práctica no resulta recomendable, en algunas ocasiones puede resultar beneficiosa.
- Las páginas web que devuelve el servidor tras la ejecución de las instrucciones, están formadas por secuencias HTML visualizables por cualquier explorador.

3.6.3. Modelo de Programación con ASP

La extensión ISAPI Active Server Pages proporciona un modelo de objetos intrínsecos que pueden ser utilizados directamente (sin ningún tipo de declaración o referencia previa) desde el código script que contiene una página ASP.

Estos objetos nos permitirán interactuar, de una manera tremendamente sencilla, con los clientes, y dirigir al servidor web para construir, paso a paso, nuestra aplicación

cliente/servidor. Vamos a describir brevemente el papel que juegan cada uno de estos objetos en una aplicación web basada en páginas ASP, ver Figura 3.2.

- **Request**: Este objeto permite, dentro del script, el acceso a cualquier información enviada por el cliente a través de http. Esto incluye cookies, formularios (Forms), parámetros en URLs y cabeceras http.
- **Response**: Posibilita el envío de información al cliente a través de http.
- **Application**: Logra que varios clientes, haciendo uso de la misma aplicación, puedan compartir la misma información.
- **Session**: Para cada cliente que accede a una aplicación se crea un objeto Session asociado que mantiene información particular.
- **Server**: Proporciona métodos y propiedades sobre el servidor, muchos de ellos son utilidades.
- **ObjectContext**: Favorece la interacción con **MTS** (*Microsoft Transaction Server*) para añadir transaccionalidad a las operaciones realizadas desde páginas ASP.

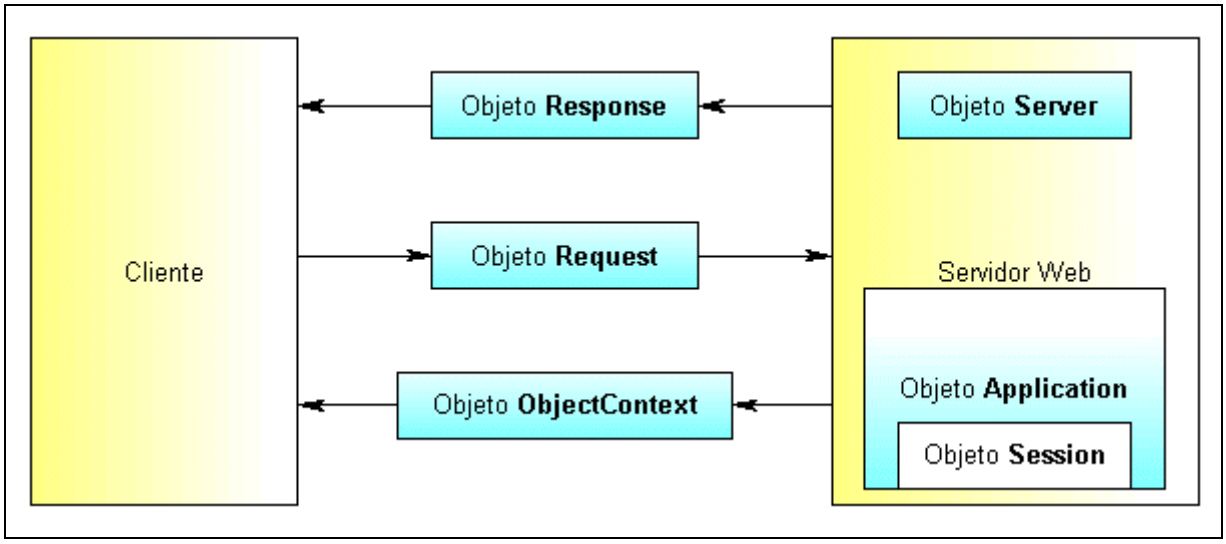


Figura 3.2. Diagrama de objetos de ASP.

Las páginas activas en el servidor (ASP) de Microsoft proporcionan un adecuado soporte de procesamiento en el servidor, a la vez que permiten su coexistencia con páginas activas en el cliente. El acceso a las bases de datos a través de ODBC ó de OLEDB es

sencillo y potente. Se sustenta en componentes ActiveX que en este caso se ejecutan únicamente en el servidor.

Con ASP, las páginas web devueltas al cliente se diseñan y mantienen de una forma sencilla (de forma centralizada), lo que supone una de las características más importantes que la diferencian de otro tipo de productos desarrollados hasta el momento. Por otra parte, el código que se manda al usuario es HTML estándar, con lo que se puede asegurar que la información se visualizará correctamente con independencia de la plataforma y el visualizador del cliente, siempre teniendo en cuenta que el código Javascript que lleve asociada la página HTML sea compatible para los Navegadores más utilizados (Microsoft Explorer y Netscape Navigator).

Las cualidades de las páginas ASP mencionadas en los párrafos anteriores confieren ventajas muy importantes en el desarrollo de aplicaciones web distribuidas, sin embargo, existe un factor de gran importancia que hay que aclarar: el producto ASP está ligado (en el servidor) al soporte de comunicaciones **IIS** de Windows 2000/NT/XP ó al **PWS** (*Personal Web Server*) ambos de Microsoft.

La ejecución de páginas web con ASP requiere que el host servidor sea Windows 2000/NT/XP ó 98/ME, lo que para muchos lejos de resultar un inconveniente supone una ventaja y una garantía de fiabilidad y de capacidad de comercialización. Pero también hay que tener en cuenta que las plataformas basadas en UNIX y las basadas en Windows compiten en el mercado, y aunque las cifras varían con rapidez a favor de Microsoft, en la actualidad existe un buen porcentaje de servidores en los que no se puede introducir ASP.

3.7. El Servidor Web, Internet Information Server (IIS)

Antes de empezar a diseñar y programar aplicaciones para el mundo de Internet, necesitamos crear y configurar un servidor con todo lo necesario para poder interactuar con dichos programas a través de un navegador. Ya explicamos anteriormente lo que tenemos que hacer para conectar un servidor a Internet, como solicitar una dirección IP y como asignar nombres de dominio en el servidor web.

Después de esto, necesitaremos algún tipo de software ejecutándose en nuestro servidor web que realice las tareas necesarias para atender a los usuarios que accedan a nuestra máquina desde Internet. Este software debe ser capaz de comunicarse con los clientes, como ya sabemos, a través de TCP/IP. El servicio TCP/IP utilizado para el acceso a la información y recursos de un servidor web es http, así que ya tenemos un dato más: nuestro software debe ser capaz de "entender" y "hablar" el protocolo http.

Llegado a este punto, si estamos decididos a crear un servidor web basado en Windows 2000, la solución a nuestro dilema es Internet Information Server. IIS 5.0 es un producto de Microsoft que proporciona a un servidor Windows 2000 los servicios más comúnmente utilizados en Internet: FTP, http y Gopher, convirtiéndolo por tanto en un servidor web. Es decir, IIS 5.0 está compuesto principalmente por un servidor http, uno FTP y otro Gopher.

El servidor http atiende peticiones de clientes web (navegadores) y envía los documentos apropiados desde el sistema de archivos, así como la salida generada por una página ASP, una aplicación ISAPI o una CGI. El servidor FTP atienda las peticiones de clientes FTP y realiza la transferencia (download) de los archivos solicitados por dichos clientes. Gopher es conceptualmente igual que http, con ciertas limitaciones. Podríamos decir que es el predecesor de http, así que prácticamente está en desuso. Además de estos servicios, imprescindibles para un servidor web, IIS 5.0 nos proporciona las siguientes funcionalidades:

- Creación y administración de diferentes sitios web.
- Integración con **MTS** (Microsoft Transaction Server – Servidor de transacciones de Microsoft) para añadir transaccionalidad a nuestros programas.
- Ejecución de programas CGI (Common Gateway Interface).
- Ejecución de extensiones ISAPI.
- Ejecución de páginas ASP.

Si no disponemos de una máquina con un sistema operativo modo servidor de tipo Windows 2000/NT/XP, podemos crear un entorno de pruebas en una máquina con Windows 98/ME mediante el PWS (*Personal web server*, servidor web personal) de Microsoft, el cual es una especie de versión Lite del IIS 5.0 para estaciones de trabajo.

IIS no es el único software que ofrece comunicaciones en Internet sobre Windows, y tampoco proporciona todos los servicios posibles; sin embargo, su importancia es enorme, puesto que se ha convertido en uno de los soportes de servidores web más utilizados, haciendo una fuerte competencia a los servidores basados en plataformas tipo UNIX-Linux como puede ser el "Apache Server".

Reuniendo las características expuestas y la evolución que existe en el mercado de servidores y servicios web, todo apunta hacia la idea de que el conjunto (Windows 2000/XP – IIS – Explorer) será utilizado de forma creciente para la publicación de datos en Internet/Intranet.

Microsoft Windows 2000 Server incluye una versión actualizada de Internet Information Services (IIS), llamadas IIS 5.0, la cual funciona como un servicio corporativo dentro de Windows 2000.

La opción más económica consiste en utilizar el IIS incluido en Windows 2000 Professional. En este caso, debemos tener en cuenta que las prestaciones que nos ofrece el conjunto no son lo suficientemente elevadas como para soportar un servidor web a gran escala. Pero como este no es el caso que nos ocupa, el uso de Windows 2000 Professional con IIS resulta muy adecuado como banco de aprendizaje y de pruebas en los desarrollos necesarios que implica el presente proyecto.

3.7.1. Administración del Servidor WEB

Nuestro objetivo será mostrar cómo administrar Internet Information Server 5 para configurar el servidor web en el que vamos a desarrollar la aplicación.

En primer lugar, si alguna vez nos hemos planteado la posibilidad de desarrollar una aplicación web nos habremos preguntado: ¿qué necesito para alojar una aplicación web?

Para tener un sitio propio, en el que la presencia sea individual y no dependa del nombre de un proveedor, deberemos obtener un nombre de dominio propio. Este es un identificador concedido por una autoridad y debe haberse asociado, mediante **DNS** (*Domain Name Service*,

Servicio de Nombres de Dominios), a una dirección IP que corresponderá con la máquina que alberga nuestro servidor. Una dirección IP fija, para que los usuarios que quieran navegar por nuestro sitio sepan dónde encontrarnos (a través de DNS; no directamente con los 4 números de la IP), aunque dentro de una Intranet y como concepto de aplicación distribuida quizás no sea tan necesario el nombre de dominio. La dirección IP sólo es posible conseguirla a través de un proveedor de servicios o telecomunicaciones, ya que ningún organismo internacional asigna actualmente direcciones IP a particulares o empresas que no sean proveedores de servicios Internet.

3.7.2. Hospedaje de Servidores

La premisa que hemos presentado hasta ahora, partía del requisito de que nuestro dominio debía residir en una máquina individual, razón que obligaba a tener una dirección IP propia y una línea de comunicaciones para nosotros solos. Sin embargo, en la actualidad y con IIS 5 esto nos es así. Un servidor web, que residirá en una máquina, puede hospedar varias sitios web, cada una de ellas asociada a un dominio distinto capaz incluso de compartir direcciones IP. Desde el punto de vista de los usuarios que acceden a los sitios web que comparten la máquina, esta situación es absolutamente indetectable, ya que ellos se limitan a escribir en su navegador el dominio al que desean acceder. DNS resolverá ese nombre y lo convertirá en la dirección IP de la máquina compartida.

3.7.3. Administración del Servidor

La utilización de **MMC** (*Microsoft Management Console*, Entorno de Gestión de Microsoft) va a permitirnos administrar desde una sola ventana todos los aspectos reseñables de nuestro servidor, tanto a nivel de los servicios Internet como para la gestión de usuarios y rendimiento. MMC (acrónimo del nombre de la consola) abre la posibilidad de modificar la configuración de cada uno de los servidores y directorios virtuales separadamente, utilizando una interfaz totalmente análoga a la del explorador de Windows. Así, seremos capaces de examinar y gestionar las propiedades de cada elemento, hasta el nivel del fichero.

En *IIS 5* se potencia extraordinariamente la posibilidad de administración remota a través de la web de cada uno de los servidores haciendo uso del navegador. Esto nos va a posibilitar, en primer lugar, resolver problemas que surjan en momentos en los que no estamos físicamente en contacto con la máquina.

3.7.4. Primeros Pasos

Entre las posibilidades básicas de los sitios web está la configuración de las características de las conexiones que van a establecerse con ellas. Podemos restringir el número límite, para protegernos ante un exceso de peticiones que afecte al rendimiento de nuestro servidor. También es configurable el tiempo de vida de las conexiones inactivas. En este sentido y para liberar recursos, tenemos la opción de especificar un valor en segundos del tiempo que se permitirá que una conexión esté inactiva antes de desconectarla.

En la pestaña **Rendimiento** (*Perfomance*) de la página de propiedades del sitio podemos modificar algunos parámetros que van a permitirnos ajustar el rendimiento del servidor, configurando el uso de la memoria y del ancho de banda del sitio. Estos son:

- **Número previsto de conexiones diarias** (*Perfomance Tuning*): Si llevamos a cabo una buena previsión, el servidor se preparará adecuadamente, en términos de memoria y mejora el rendimiento.

- **Limitación del ancho de banda de salida** (*Enable Bandwidth Throttling*): Si se indica un valor, limitaremos el ancho de banda disponible para los clientes, evitando que los usuarios colapsen el servidor repercutiendo en el rendimiento de otros.

- **Mantener conexiones http** (*http Sep-Alives Enabled*): Marcar esta opción permite a los clientes mantener la conexión http abierta con el servidor en lugar de reestablecerla con cada petición de página.

3.7.5. Monitorización del servidor

Internet Information Server proporciona mecanismos para monitorizar la actividad del servidor; es decir, para anotar las acciones que los usuarios van llevando a cabo en los sitios web. Este proceso es comúnmente conocido como *logging*.

Los mecanismos de monitorización se integran extraordinariamente con los aportados por Windows 2000, pero van bastante más allá. De hecho, no sólo todos los errores provocados por el servidor se anotan en el registro de sucesos del sistema operativo y pueden verse en el **Visor de sucesos** (*Event log*) de Windows 2000, sino que el proceso de *logging* conlleva anotar una gran cantidad de información acerca de la actividad de los usuarios.

Entre otras cosas, el proceso de logging anota las siguientes informaciones:

- Dinámica de visitas de nuestros sitios, incluyendo las direcciones IP de las máquinas desde las que se ha accedido al servidor.
- Que páginas se han visitado.
- La fecha en la que se han producido las visitas.

Esta información puede sernos de gran utilidad para, por ejemplo, conocer cuáles son las páginas que están teniendo más éxito, las horas de más tráfico o detectar posibles agujeros de seguridad y otros problemas que puedan producirse en la actividad del servidor. Además con la información de *logging*, el rendimiento de *Internet Information Server* puede valorarse con otra herramienta del sistema operativo: el **Monitor de Rendimiento** (*Perfomance Monitor*). Esta nos posibilita conocer los valores que en cada momento alcanzan un conjunto de contadores predefinidos o creados por nosotros mismos. Estos contadores reflejan estadísticas sobre ciertos parámetros tales como el número de conexiones activas o el porcentaje del procesador utilizado. El análisis de estas cifras estadísticas puede ser de gran valor en nuestra optimización de rendimiento.

3.8. Páginas Activas en el Cliente

Los lenguajes de Script están adquiriendo cada vez mayor importancia. Hoy en día no se usan solamente para añadir contenidos dinámicos y programáticos a las páginas de Internet. Desde hace tiempo están también disponibles como medios de manipulación y personalización de muchas aplicaciones (p.ej. Microsoft Office 2000), realización de procesos en servidores de Internet, administración del sistema (*Windows Scripting Host* en Windows), etc...

Es por todo esto que pocos programadores modernos, utilicen el lenguaje que utilicen, se pueden permitir el lujo de desconocer el funcionamiento de los lenguajes de Script.

Dentro de los lenguajes de Script los dos más conocidos son **Javascript** y **VBScript**. Este último es un subconjunto de Visual Basic adaptado para usarse como lenguaje interpretado. Si nos referimos a su uso en navegadores de Internet, sólo se puede emplear con Microsoft Internet Explorer (aunque existe un plug-in comercial que permite usarlo en Netscape Navigator). Javascript, sin embargo, está soportado por casi cualquier navegador gráfico que nos podamos encontrar. Ahí es donde radica precisamente su éxito y proliferación. Esta presencia casi ubicua en los navegadores, unida a sus potentes características, lo han llevado a ser el lenguaje de Script más extendido.

3.8.1. Navegadores de Internet y sus versiones

Actualmente es un hecho conocido que la mayor parte del mercado de navegadores de Internet está dominado por dos aplicaciones que se reparten el pastel: **Microsoft Internet Explorer** (IE) y **Netscape Navigator**. Siendo precavidos podemos decir que estos dos programas abarcan más del 95% del mercado. Es por este motivo que durante el presente proyecto se ha optado por el lenguaje Javascript común a ambos, para asegurar que el código escrito funcione sin problemas en la práctica totalidad de los navegadores que accedan a sus páginas.

Además existen otros muchos navegadores de Internet de terceras empresas como Opera, Netcaptor, Konqueror, etc... pero poco extendidos.

Microsoft y Netscape (con el apoyo de IBM, Sun, Oracle, etc...) se encuentran inmersos en una "guerra" comercial desde hace tiempo, cuyo objetivo es conseguir que los usuarios utilicen sus respectivos navegadores de Internet (Explorer y Netscape). A simple vista, parece tratarse de un competencia por conseguir el beneficio económico de la venta de los programas visualizadores en sí, pero nada más lejos de la realidad. De hecho, ambos son gratuitos.

Para entender las causas de esta feroz competencia debemos ser conscientes de que la tendencia apunta hacia una utilización masiva de los recursos de Internet por parte de los propietarios de equipos informáticos. Por otra parte, se tiende a homogeneizar la interfaz de acceso a los recursos locales de cada máquina (como el disco duro) con la interfaz de acceso a los recursos remotos (como una base de datos en otro equipo). Esta interfaz se pretende que sea la de los visualizadores de páginas web.

Si ambos exploradores funcionaran correctamente en distintas plataformas, no existiría ningún tipo de problema, pero la realidad actual es diferente. Debido especialmente al uso de páginas web activas en el cliente (aunque influyen otros muchos factores), nos podemos encontrar con que al acceder a un lugar web, las páginas no se pueden visualizar correctamente debido a que no disponemos de un intérprete del lenguaje adecuado (por ejemplo, Visual Basic Script con Netscape), o bien a que no podemos cargar algún control ActiveX (tecnología derivada de los OCX´s de Microsoft), etc. Ambos navegadores soportan a la perfección el lenguaje Javascript desde sus primeras versiones. A medida que iban saliendo nuevas versiones de los navegadores se iban añadiendo también nuevas características a Javascript.

Sin embargo, los cambios más significativos no se refieren al lenguaje propiamente dicho, sino más bien a la manera de acceder y tratar los contenidos de las páginas. A partir de las versiones 4.0 de ambos navegadores entra en escena lo que se conoce como HTML dinámico. Este es un hecho trascendental para los creadores de contenidos para la web, ya que teóricamente, permite una libertad casi absoluta para modificar contenidos tras las descarga de las páginas y añadir verdadera interactividad.

Lamentablemente todo lo bueno que esto supone se ha traducido en una verdadera pesadilla para los desarrolladores de páginas web. Las implementaciones de **DHTML** (*Dynamic HTML*, HTML dinámico) de Internet Explorer y Navigator difieren enormemente,

incluyendo en ambos casos etiquetas propias que sólo funcionan en uno de ellos. La manera de acceder desde Javascript a los elementos constituyentes de las páginas es diferente en ambos casos. Además, los atributos que se pueden modificar mediante código en cada elemento son más limitados en Navigator que en Internet Explorer.

Conclusión de todo esto: escribir un programa en Javascript con HTML dinámico que funcione en todos los navegadores y en todas sus versiones es poco menos que imposible. Esta coyuntura lleva a una cuestión realmente difícil de resolver para los desarrolladores de contenidos: ¿qué hago para que mis páginas puedan ser vistas por la mayor cantidad posible de personas que las visiten? Las respuestas las veremos a continuación.

1) Escribir código sólo para uno de los navegadores

Esto puede ser que no sea tan mala idea ya que, cada vez más, los usuarios se están decantando por Internet Explorer como su navegador de Internet predeterminado. Por supuesto en ello tiene mucho que ver el hecho de que Microsoft lo incluya "de serie" en sus sistemas operativos. Dejando a un lado disquisiciones de otro tipo (si es legal o no que hagan esto, etc...), Internet Explorer es el navegador más usado actualmente. Son los hechos.

Si nos decidimos por desarrollar para este navegador descubriremos que es mucho más flexible y agradable para escribir código de Script que Netscape Navigator. Sus posibilidades también son mayores. Basada en la experiencia puedo decir que conozco a fondo ambos programas y mi afirmación es objetiva.

En cualquier caso adoptando esta posibilidad estamos dejando fuera de su ámbito a unos cuantos millones de personas que aún emplean Netscape Navigator, que está disponible para más plataformas que Internet Explorer. Actualmente, este último está disponible para todas las plataformas Windows (95, 98, ME, NT, 2000 y XP), Macintosh y algunos UNIX (Sun Solaris y HP-UX). Netscape tiene versiones de su navegador para casi todas las plataformas informáticas existentes, y en el mundo UNIX es utilizado por la práctica totalidad de los usuarios.

Esto hace pensar que esta primera posibilidad de desarrollar sólo para uno de los dos navegadores no es muy adecuada, pues en cualquiera de los casos dejaríamos fuera a millones de potenciales visitantes de nuestra aplicación web.

2) Escribir dos veces el código adaptándolo a cada navegador

Bueno, desde luego es una posibilidad y utilizada en algunos desarrollos, pero evidentemente supone el doble de esfuerzo de recursos empleados en el desarrollo, lo cual no siempre compensa. Realmente es una opción sólo recomendable cuando el código a mantener es poco y muy especializado. En otro caso no compensa de ninguna de las maneras.

3) Normalizar el código

En este contexto normalizar significa llanamente "elegir el mínimo común denominador". Así, es: si queremos asegurarnos de que nuestro código va a funcionar en ambos navegadores y que además no vamos a tener que repetirlo para adaptarlo a cada uno de ellos, éste es el camino que debemos seguir.

Para el desarrollo del proyecto este es el criterio que se ha elegido. Todo el código está verificado para funcionar correctamente en ambos navegadores sin problemas. Esto significa autolimitar un poco la potencia de nuestros Scripts, pero por otro lado asegura ampliar el abanico de usuarios de nuestra aplicación web.

El hecho de normalizar el código Javascript significa, en la práctica, adaptarlo a las características de Netscape Navigator 4.0. Aunque la funcionalidad mínima común a todos los navegadores es la proporcionada por Navigator 2.0, es puede decir, sin miedo a equivocarnos, que más del 90% de las personas que emplean Navigator usan una versión 4.x o superior. Así, escogiendo esta versión como referencia nos aseguramos la compatibilidad en casi la totalidad de los casos.

Para el desarrollo del proyecto y debido a la gran sobrecarga de código cliente se ha optado por conservar la compatibilidad para los dos Navegadores más utilizados de Internet (Microsoft Explorer y Netscape Navigator).

3.8.2. JAVASCRIPT

JavaScript [1] es un lenguaje de programación que fue originalmente creado por la empresa Netscape para añadir interactividad a las páginas web vistas con su navegador de Internet. Actualmente JavaScript está integrado en otras aplicaciones y otros navegadores de Internet, y es uno de los lenguajes más utilizados en la red de redes para añadir interactividad a las páginas web.

No hay que confundir JavaScript con Java. Este último es un lenguaje de propósito más general creado por la empresa Sun Microsystems con unos objetivos muy diferentes. Javascript se diseñó teniendo Java en mente; pero, a pesar de que sus sintaxis es muy similar, son lenguajes muy distintos.

El código Javascript se embebe en el código HTML de las páginas web añadiendo cierta "inteligencia" e interactividad a las mismas. La mayor parte de las páginas web modernas incluyen algo de código Javascript, bien sea para obtener ciertos efectos estéticos (cambiar una imagen al pasarle por encima, mover un gráfico por la pantalla, etc...), bien para validar una entrada de datos, hacer cálculos, cargar dinámicamente valores en listas desplegables, etc...

Los programas en Javascript no generan ningún tipo de código compilado, sino que éste se interpreta en el navegador de Internet una vez se descarga la página que lo contiene. A este tipo de lenguajes de les denomina lenguajes interpretados.

Javascript es un lenguaje muy sencillo que enseguida nos permitirá desarrollar aplicaciones propias para Internet con poco esfuerzo, siempre dependiendo de la complejidad del proyecto.

No necesitamos ninguna herramienta especial para programar en Javascript. Simplemente usando el bloc de notas de Windows podemos escribir código dentro de una página web, aunque siempre es interesante poder usar alguna herramienta que nos facilite la escritura de código, como por ejemplo el entorno integrado de desarrollo (IDE) de *Visual Studio* de Microsoft ó el *Dreamweaver* de Macromedia.

3.8.2.1. Versiones de Javascript

Lamentablemente no existe una única versión de JavaScript sino varias distintas, que poseen, en algunos casos, características incompatibles entre sí.

Esto supone un problema para nosotros como programadores y desarrolladores de aplicaciones para Internet, puesto que evidentemente, nos interesa que nuestras páginas puedan ser vistas por el mayor número de usuarios posibles.

Microsoft tiene una implementación propia del lenguaje para su Internet Explorer llamada *JScript*. Esta no es totalmente compatible con la de los navegadores de Netscape, y por lo tanto, cuando escribamos un programa, debemos probarlo con ambos navegadores para asegurar que trabaja bien en los dos.

Además existen diferentes números de versión para el lenguaje que han ido apareciendo a medida que se modernizaban los navegadores de Internet. En la Tabla 3.2 se pueden observar las diferencias.

Navegador	Versión del Lenguaje
Navigator 2.0	JavaScript 1.0
Internet Explorer 3.0	JScript 1
Navigator 3.0	JavaScript 1.1
Internet Explorer 4.0	JScript 3
Navigator 4.0	JavaScript 1.2
Navigator 4.06 y 4.6	JavaScript 1.3
Internet Explorer 5.0	JScript 5
Navigator 6.0	JavaScript 1.4
Internet Explorer 5.5	Jscript 5.5

Tabla 3.2. Versiones de Javascript.

Cada versión sucesiva mantiene de manera adecuada la compatibilidad hacia atrás, es decir, si tenemos código escrito para una versión dada del navegador, éste deberá funcionar perfectamente en cualquier versión superior.

Debido a esto, y si no necesitamos usar las características más nuevas del lenguaje, emplearemos siempre las versiones más antiguas para conseguir que el código funcione para el mayor número de usuarios posibles.

La implementación de Microsoft, *Jscript*, permite el uso de muchas características adicionales del estándar ActiveX (creación y automatización de objetos ActiveX, objetos adicionales del navegador, etc...). Aunque nos da más flexibilidad también restringe la posible audiencia al no estar soportado por Netscape Navigator.

Ante esta disyuntiva se impone tomar una decisión respecto a cómo programar: ¿preferimos libertar absoluta para hacer cosas a costa de perder usuarios potenciales o mejor nos quedamos con unas posibilidades algo recortadas pero llegamos a todo el mundo? Personalmente me quedo con la segunda opción. Aunque las posibilidades son algo menores con las primeras versiones del lenguaje.

Jscript es un lenguaje que se adhiere muy bien al estándar *ECMAScript*, el cual trata de unificar los criterios entorno al lenguaje. Además, podemos obtener más funcionalidad en Internet Explorer que en Navigator gracias a ActiveX y a otras muchas características soportadas por el navegador. Incluso la sintaxis es bastante menos estricta en Explorer que en Navigator, facilitándonos así la escritura de código. De este modo parece mejor, a priori, programar para Internet Explorer que para Navigator. Sin embargo, no hay que olvidar que este último es todavía el navegador más empleado en algunas plataformas (como Linux o Macintosh); y aunque en Windows manda claramente Internet Explorer, aún queda un gran porcentaje de usuarios que emplean el navegador de Netscape. Además, si no nos importa programar solamente para Internet Explorer dejando de lado a los usuarios de Netscape, entonces es mejor usar directamente VBScript, otro lenguaje de los navegadores de Microsoft que es mucho más sencillo de usar y ofrece la misma potencia.

Así pues, tras esta digresión, se infiere que lo mejor es tratar de unificar el código de forma que sirva para ambas plataformas, quedándonos con la parte común a los dos navegadores.

3.8.2.2. Integración con HTML

Lo primero y más importante que debemos conocer de un lenguaje pensado para usar en páginas web es cómo podemos integrarlo dentro de una de ellas.

Esto se consigue mediante la utilización de un pareja de etiquetas HTML especiales llamadas <SCRIPT></SCRIPT>.

Todo lo que vaya escrito entre ellas es código de Script. Sin embargo, esto no es suficiente ya que no estamos indicando qué lenguaje vamos a utilizar, y si el navegador soporta varios (como es el caso de Internet Explorer) no sabría como interpretar nuestro código. Por ello debemos utilizar el modificador LANGUAGE dentro de la etiqueta <SCRIPT> que abre un fragmento de código.

Por ejemplo, consideremos el contenido HTML de la siguiente página:

```
<HTML>
<HEAD>
<TITLE>Ejemplo de código Javascript</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
alert("¡Hola Javascript!");
// -->
</SCRIPT>
</HEAD>
<BODY>
<P>
Proyecto Visual FSQL: primer ejemplo de Javascript
</P>
</BODY>
</HTML>
```

Esta página mostrará el texto "Proyecto Visual FSQL: primer ejemplo de Javascript", pero antes sacará un cuadro de diálogo con el mensaje "¡Hola Javascript!".

3.8.2.3. Sintaxis Básica del Lenguaje

Vamos a analizar lo comentado en la sección anterior.

Primeramente, tras el título de la página, se han colocado las etiquetas <SCRIPT></SCRIPT>, especificando con LANGUAGE el nombre del lenguaje que vamos a utilizar. En este caso se ha usado el genérico "JavaScript" aunque podíamos haber especificado la versión concreta, por ejemplo "JavaScript 1.3", o en el caso de Internet Explorer "JScript". Lo más habitual es poner simplemente "JavaScript", ya que pretendemos que se ejecute en el mayor número posible de navegadores.

A continuación se escribe el código Javascript. En el ejemplo consta de una única línea que se encarga de mostrar el cuadro de mensaje de la figura. Este código se interpreta en el mismo momento en que se carga en el navegador, a medida que éste analiza la página. Ello implica que en nuestro ejemplo se mostrará el mensaje, y hasta que no cerremos su ventana pulsando sobre Aceptar, no se seguirá interpretando el código de la página, y por lo tanto no aparecerá su contenido (ya que las etiquetas <BODY> están colocadas después).

Todas las líneas de código Javascript se deben terminar con un punto y coma (;). Ello nos permitirá escribir una misma sentencia que ocupe varias líneas (para poder verla mejor) ya que no se considerará terminada la sentencia hasta que se encuentre el punto y coma.

Hemos situado el código al principio del documento web, en la sección de cabecera (<HEAD>). Sin embargo, no existe un lugar obligatorio donde ubicarlo. Podemos escribirlo en cualquier parte del archivo, incluso con varias etiquetas <SCRIPT> se pueden distribuir por varios sitios diferentes. Lo recomendable, sobre todo para aplicaciones sencillas, es tenerlo todo junto para poder encontrarlo y corregirlo fácilmente, y el mejor sitio es al principio del archivo HTML. Sin embargo para el presente proyecto y debido a la complejidad de los códigos cliente (Javascript) y servidor (ASP) y la interacción entre ellos, se ha optado por distribuir muchos trozos de código Javascript a lo largo de las páginas HTML y ASP. Ver [1].

3.9.Programación Internet-Intranet

En este apartado veremos la creación del servidor web y nos iniciaremos en la programación de aplicaciones que residirán en dichos servidores y con las que podrá interactuar cualquier usuario a través de un navegador [21, 22].

3.9.1. El Modelo Cliente/Servidor

La programación de aplicaciones Internet-Intranet no introduce conceptos nuevos en el mundo del desarrollo de aplicaciones informáticas, ya que encaja a la perfección dentro del ya tradicional modelo de la programación cliente/servidor. Por tanto el desarrollo de soluciones informáticas cliente/servidor se compone de dos partes: un cliente, normalmente un modesto PC que transmite una petición y presenta la información al usuario final, y un servidor, normalmente una máquina de altas prestaciones que almacena, recoge, manipula y devuelve los datos que son objeto de dicha petición. Algo que puede verse reflejado en la Figura 3.3.

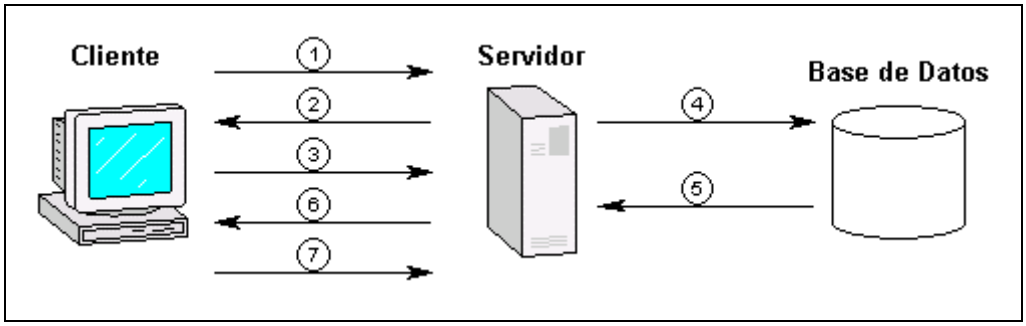


Figura 3.3. Modelo cliente/servidor.

Las acciones que conforman básicamente la realización de una tarea en un modelo de programación cliente/servidor son las siguientes:

1) Nos conectamos al servidor.
2) El servidor acepta la conexión.
3) Realizamos una petición de información al servidor.
4) El servidor recoge de la base de datos la información solicitada.
5) El servidor procesa y compone la información.

6) El servidor nos devuelve la información solicitada.
7) Cerramos la conexión con el servidor.

Si sustituimos en este modelo el cliente por una navegador Internet (Internet Explorer, Netscape Navigator u otro) y el servidor por un servidor web, ya tenemos el modelo cliente/servidor en Internet. Al fin y al cabo, para acceder a un sitio en Internet, lo que realmente hacemos es, a través de un navegador, conectarnos a un servidor web mediante una dirección Internet. El servidor web aceptará (o no) dicha conexión y nos dará la bienvenida. A partir de este momento, solicitaremos información que reside en dicho sitio y el servidor web la localizará, la preparará convenientemente y nos la devolverá en forma de páginas HTML, las cuales serán mostradas por nuestro navegador.

Dentro del modelo cliente/servidor, podemos diseñar soluciones basadas en diferentes tipos de arquitecturas, en función de las necesidades que se nos planteen. Vamos a describirlas [21, 22].

3.9.2. Arquitectura Cliente/Servidor de dos Capas

Es la más básica dentro del modelo cliente/servidor y, tal vez por esto, la más extendida en cuanto a su uso. Este tipo de arquitectura, como su propio nombre indica, está compuesta por dos capas: la primera constituida por ordenadores que actúan como clientes y la segunda por un servidor que mantiene y procesa toda la información y atiende las peticiones de todos los clientes.

Este modelo es muy común, por ejemplo en soluciones que requieren el acceso simultáneo y concurrente de usuarios a una base de datos remota. En la Figura 3.4 podemos ver un ejemplo de esta arquitectura.

En este ámbito, siendo muy estrictos, los clientes contienen toda la lógica de presentación de la información al usuario final y en el servidor se encuentra toda la lógica de almacenamiento, recuperación y tratamiento de la información solicitada por los clientes. Es lo que se denomina lógica o reglas de negocio.

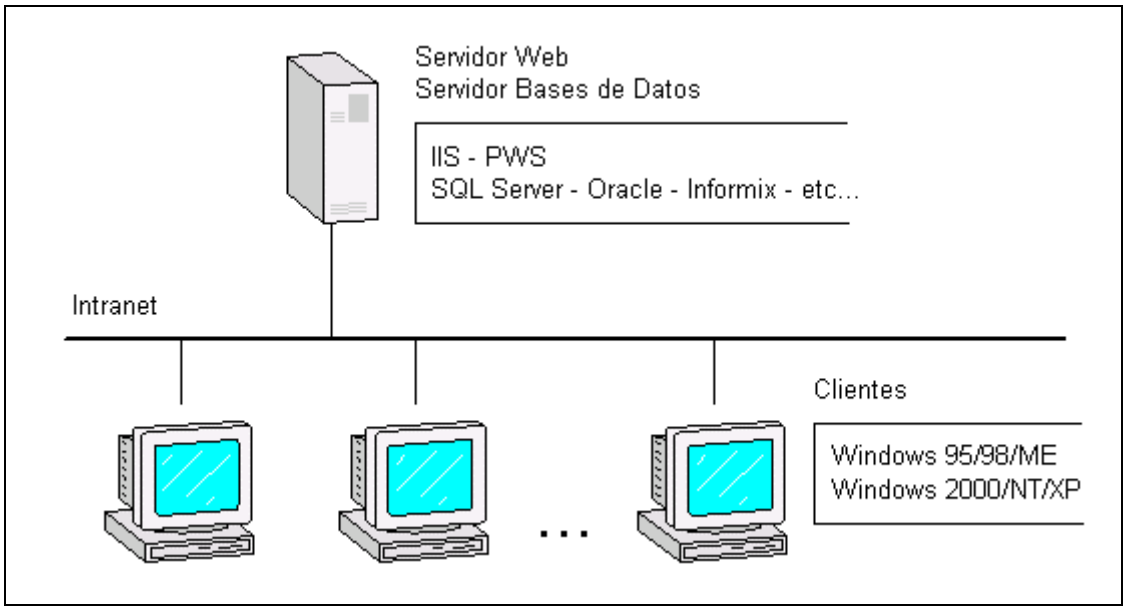


Figura 3.4. Arquitectura Cliente/Servidor de dos capas.

Para el presente proyecto se ha optado por esta arquitectura debido a su bajo-medio coste de desarrollo, y sobre todo por la adaptación al tipo de aplicación que se ha implementado, en el que se gestiona fundamentalmente el acceso a Bases de Datos, con gran funcionalidad de operaciones tanto en cliente como en servidor.

3.9.3. Arquitectura Cliente/Servidor de tres Capas

Ésta es algo más compleja, pero añade algunas mejoras con respecto a la anterior. La diferencia con la arquitectura de dos capas es que se incorpora una tercera, representada por un servidor, normalmente denominado servidor de aplicaciones, donde se encuentra toda la lógica de tratamiento y manipulación de la información solicitada por los clientes. Es decir, alberga las aplicaciones y componentes software encargados de proporcionar a las aplicaciones de usuario final las reglas de negocio o lógica de negocio. Para no extendernos demasiado en este punto, vamos a señalar únicamente dos grandes ventajas de esta arquitectura: independencia y escalabilidad. Así, si cambia la lógica de negocio, sólo se deben cambiar el o los componentes afectados en un solo sitio, el servidor de aplicaciones, manteniendo las aplicaciones de usuario final. Por ejemplo, supongamos que un banco aplica una comisión del 10% en determinadas operaciones. Si decide, a partir de un determinado

momento, aplicar una comisión del 15%, es más fácil implementar este cambio únicamente en el componente encargado de aplicar esta comisión de aplicaciones que modificar, probablemente, 20 aplicaciones de usuario final y además tener que instalar las nuevas versiones de los programas en 300 puestos clientes diferentes. Esto mismo es válido si, por ejemplo, se decide cambiar un motor de bases de datos (supongamos SQL Server) por otro (Oracle).

Las aplicaciones de usuario final no deberían verse afectadas. Por otro lado, ¿qué sucedería si nuestra solución, inicialmente diseñada para atender a 10 clientes, tuviese la necesidad de atender de repente a 200 clientes? La respuesta parece obvia, ¿verdad? Simplemente cambiamos los servidores por otros con más capacidad de procesamiento y más recursos, y solucionado el problema. Bien, esto es una solución, pero tal vez sea muy cara, ya que para obtener los rendimientos deseados, una máquina de las características requeridas podría suponer un coste demasiado elevado, o aún peor, es posible que ni siquiera encontremos una máquina de estas características.

A todo esto, ¿no sería mejor añadir un nuevo servidor de aplicaciones y/o base de datos a nuestra solución? Bien, así es, el diseño de soluciones informáticas basadas en la arquitectura de tres capas facilita su escalabilidad, en función de las necesidades de cada momento, ya que podemos distribuir con mayor facilidad la lógica de nuestro negocio en diferentes maquinas, sin por ello afectar a los usuarios finales (clientes). Todo esto nos llevaría a convertir nuestra solución en una arquitectura cliente/servidor multicapa.

3.9.4. Arquitectura Cliente/Servidor Multicapa

Esta arquitectura, conceptualmente, es exactamente igual a la de tres capas, con la diferencia de que se pueden añadir tantas capas como sean necesarias en la solución, es decir, pueden existir tantos servidores como se requieran.

Es bastante común en el mundo de Internet que las soluciones estén diseñadas basándose en esta arquitectura.

Capítulo 4

Nuevas Tecnologías de Acceso a Bases de Datos

En este capitulo vamos a obtener una visión general de las nuevas tecnologías de acceso a bases de datos existentes actualmente para entornos Microsoft en el desarrollo de aplicaciones distribuidas con acceso a Bases de Datos. No todas estas técnicas son estrictamente una novedad, ya que algunas estaban disponibles en anteriores versiones de las plataformas de desarrollo de Microsoft.

En la situación actual de desarrollo de aplicaciones cliente/servidor (sección 3.9.1) no basta con definir adecuadamente el almacenamiento de la información en bases de datos relacionales, sino que es preciso definir también un conjunto de tecnologías que nos permitan acceder a los datos desde nuestras aplicaciones. Estas aplicaciones incorporan mecanismos para posibilitar un acceso uniforme e integrado a gestores heterogéneos y facilitan a los desarrolladores la tarea de crear aplicaciones mediante la definición de modelos de programación, algunos orientados a objetos y otros siguiendo un modelo similar al del API de Windows, utilizables desde los lenguajes de programación más comunes.

Además se va a profundizar un poco más en la arquitectura *ADO* que junto al modelo *OLE DB Provider* serán las técnicas utilizadas para el acceso a los datos de Oracle en el presente proyecto [2].

4.1. UDA (Acceso Universal a los Datos)

UDA (*Universal Data Access*, Acceso Universal a Datos) es una nueva estrategia de Microsoft para el acceso a información que puede modelarse para ser presentada como un

conjunto de registros o *recordsets*, independientemente de si la información reside en una base de datos relacional o no. La idea es acceder, por ejemplo, a un sistema de ficheros o de mensajería y obtener información mediante consultas, análogas a las SQL que se envían a un gestor de bases de datos relacionales a tal efecto.

En el típico modelo de empresa actual existe una buena cantidad de información crítica que no se encuentra en bases de datos relaciones, sino en sistemas de ficheros, hojas de cálculo, bases de datos no relacionales y correo electrónico. Para procesar y obtener datos de manera integrada y eficiente es necesario migrar esta información a bases de datos relacionales; con el consiguiente coste y redundancia. Lo que UDA soluciona es el acceso a los datos independientemente de dónde estén.

Por lo tanto, UDA no está ligada a ningún gestor y permite acceder a todas las informaciones que necesitemos sin necesidad de duplicarlas, sino que las obtiene desde donde se encuentren almacenadas. Por otro lado, es un estándar abierto que funciona con un importante número de productos de bases de datos. En la Figura 4.1 podemos ver gráficamente la arquitectura de UDA.

4.2. COM (Component Object Model)

4.2.1. Origen

COM tiene sus raíces en OLE versión 1 (Object Linking and Embedding, Vinculación e incrustación de objetos) creada en 1991 como marco de gestión e integración de documentos para la familia de programas Microsoft Office. Más adelante, la compañía se dio cuenta que la integración de documentos no es sino un caso especial de integración de componentes, y a partir de la versión 2 de OLE, presentada en 1995 y denominada desde entonces como COM, proporciona un mecanismo de propósito general para la integración de componentes en plataformas Windows. Aunque ya esta versión inicial de COM incorpora ciertas nociones de componentes distribuidos, sólo con la presentación de la especificación **DCOM** (*Distributed Component Object Model*, Modelo Distribuido de Objetos Componentes) y su implementación para los entornos Windows 95 y NT en 1996 alcanza la distribución un

auténtico soporte. Al poco tiempo aparecen versiones provisionales de DCOM para las plataformas Mac, Solaris y otros sistemas operativos.

4.2.2. Campo de Aplicación y Alcance

Con el término *COM* se conoce tanto la especificación como la implementación de Microsoft que proporciona un marco para la integración de componentes. Este marco soporta la *interoperabilidad* y la *reutilización* de componentes distribuidos, con lo que los desarrolladores pueden construir sistemas a base de ensamblar componentes de diferentes proveedores, que se comunican unos con otros vía *COM*. Otros beneficios que se pretenden alcanzar con este esquema de construcción de sistemas es la *adaptabilidad* y la *mantenibilidad.*

COM define un interfaz de programación (*API*) para la creación de componentes que van a usarse para la construcción de aplicaciones a medida o para permitir que diversos componentes interactúen. Sin embargo, para que esta interacción sea posible, los componentes deben adoptar una estructura binaria especificada por Microsoft. Si lo hacen, aunque los componentes estén escritos en diferentes lenguajes, pueden interoperar.

DCOM es una extensión de *COM* que permite la interacción de componentes en una red. Mientras que los procesos *COM* pueden funcionar en la misma máquina aunque en diferentes espacios de direcciones, con *DCOM* estos procesos pueden estar dispersos en diferentes lugares de una red. Con *DCOM* es posible la interacción de componentes disponibles en diferentes plataformas, siempre que exista *DCOM* para esos entornos.

Resulta más conveniente considerar *COM* y *DCOM* como una sola tecnología que proporciona una serie de servicios para la interacción de componentes, desde la integración de objetos en una sola plataforma hasta la interactividad de componentes a través de redes heterogéneas. De hecho, las extensiones *COM* y *DCOM* están mezcladas en un solo 'runtime' que proporciona tanto acceso local como remoto.

4.2.3. Conexión con Otras Normas

COM y *DCOM* son tecnologías de "bajo nivel" que permiten interactuar a los componentes, mientras que *OLE* y *ActiveX* representan servicios de aplicación de alto nivel construidos sobre aquéllas. *OLE* proporciona servicios de *linking* y *embedding* (enlace e incrustación) de objetos para la creación de documentos compuestos (documentos creados con herramientas de orígenes múltiples). *ActiveX*, por su parte, extiende las capacidades básicas para poder incrustar los componentes en instalaciones Web.

4.2.4. Utilización y Acceso

COM y *DCOM* están especialmente maduras para los entornos propiedad de Microsoft. Existe un amplio mercado de componentes conformes con las especificaciones *COM* para Windows. Una iniciativa de importancia fuera del campo Windows es el desarrollo *EntireX* de la compañía "*Software AG"*, que extiende las posibilidades de *DCOM* a diversos sistemas Unix (Solaris, Digital Unix, HP-UX, AIX, Linux) así como al entorno de grandes ordenadores OS/390.

El paradigma de las aplicaciones distribuidas está moviéndose muy rápidamente, debido en parte a la relativa inmadurez de la tecnología y a los avances recientes de la informática basada en la web. La industria que desarrolla esta informática ha empezado a alinearse en dos campos tecnológicos rivales, uno de ellos centrado alrededor de las tecnologías de Microsoft, COM/DCOM, Internet Explorer, y ActiveX, y el otro en soluciones de Netscape, CORBA y Java, sin que todavía pueda decidirse que haya un claro vencedor. Afortunadamente ambos campos trabajan en diversos mecanismos para interconectar las dos bases tecnológicas.

4.3. Que es OLE DB

UDA es una filosofía de actuación, plasmada en la práctica por OLE DB (*Object Linking and Embedding DataBase,* Enlace e incrustación de objetos de Bases de Datos). La idea es, mediante mecanismos totalmente integrados con el modelo *COM*, extender el concepto de controlador *ODBC* a fuentes de datos no relacionales. De esta forma, se supera

la naturaleza del *API ODBC* (escrito en C y pensado para ser utilizado desde este lenguaje de programación) y se propone un estándar independiente del lenguaje que sigue las directrices de *COM*. *OLE DB* obtiene su funcionalidad de la existencia de un servidor *COM* (un servidor *OLE*, si se prefiere; de ahí su nombre) que se comunica tanto con los clientes que quieren acceder a los datos como con las propias fuentes en las que éstos se almacenan.

En la Figura 4.1 se puede observar la integración de OLE DB dentro de UDA.

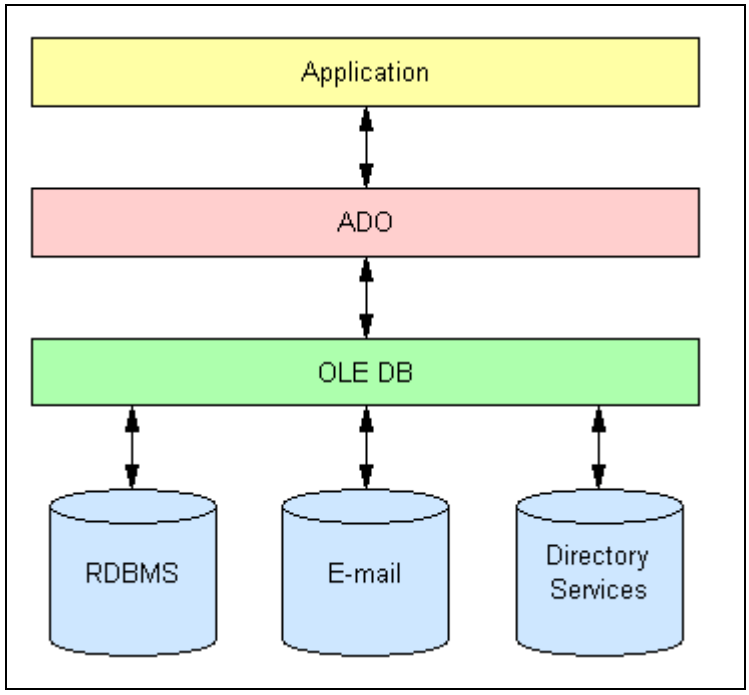


Figura 4.1. Arquitectura de UDA.

Lo que denominábamos *"Controlador o driver ODBC"*, pasa a llamarse *"OLE DB Provider"*. De hecho, *OLE DB* se fundamenta en una abstracción que considera tres entidades diferenciadas en el acceso a datos: los consumidores de datos (*Data Consumers*), los proveedores de los mismos (*Data Providers*) y los componentes proveedores de servicio (*Service Providers*), que los procesan y transportan (por ejemplo, los elementos que implementan el mecanismo de cursores).

Supongamos que tenemos una base de datos en un gestor de bases que contiene una tabla con la información relativa a los usuarios de un proveedor de servicios Internet. Este ha

enviado a los usuarios mensajes de correo electrónico en los que les comunica cambios en la configuración del servidor POP. El administrador desea comprobar qué usuarios han enviado un acuse de recibo de la comunicación de cambios por el mismo medio de comunicación. Normalmente, esta tarea supondría examinar exhaustivamente la bandeja de entrada hasta encontrar los mensajes de correo de cada uno de los usuarios y la inspección para determinar cuáles aún no han contestado.

Este procedimiento es costoso e implica un componente manual claramente mejorable. Resultaría ideal si pudiésemos lanzar una consulta SQL contra la base de datos de usuarios y el sistema de mensajería que nos devolviese directamente los resultados deseados. Con *OLE DB* esto es posible ya que existe un proveedor de datos *OLE DB* que permite el acceso al sistema de mensajería.

4.3.1. Arquitectura de OLE DB

Como ya se ha comentado, *OLE DB* está basado en el modelo *COM*, lo que supone que su uso a bajo nivel tiene una complejidad considerable. Está definido como un conjunto de objetos autocontenidos, que pueden compartirse por varios consumidores y que exponen un conjunto de interfaces que encapsulan porciones disjuntas de las funcionalidades que requeriríamos de un gestor de bases de datos. Estas interfaces son ampliables en función de los servicios que los proveedores deseen proporcionar.

- El **Proveedor de Datos** (*Data Provider)* es un componente *OLE DB* que presenta datos desde una fuente relacional o no (correo, sistema de ficheros, etc.) de una manera similar a como lo haría un gestor de bases de datos relacional, es decir, como un conjunto de registros. Esta exposición de los datos, evidentemente, se lleva a cabo mediante la presentación de un conjunto de interfaces. La interfaz mínima, por tanto, que puede exponerse sería aquella que presentase un conjunto de datos como una tabla.

- Un componente **Proveedor de Servicios** (*Service Provider*) no está vinculado a ningún dato en particular, sino que encapsula uno o varios servicios que pueden

añadirse a la funcionalidad de los proveedores de datos; y por lo tanto, proporcionan valor añadido a los consumidores.

- Un **Consumidor de Datos** (*Data Consumer*), por su parte, es una aplicación o componente que recibe servicios de un proveedor *OLE DB* a través de sus interfaces.

En la Figura 4.2 se puede ver gráficamente lo comentado anteriormente.

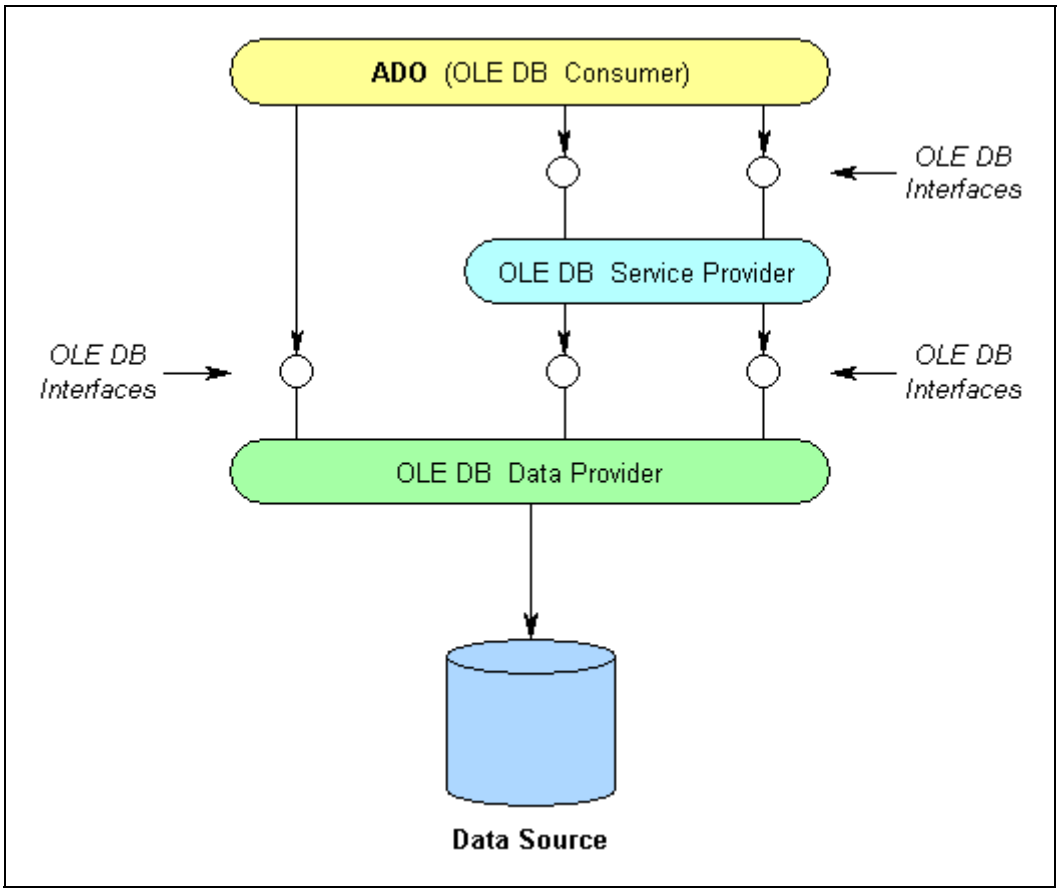


Figura 4.2. Arquitectura de OLE DB

4.3.2. Componentes

A modo ilustrativo, se mostrarán algunos de los objetos definidos en *OLE DB*:

- **Enumeradores** (*Enumerators*): Es muy común que existan múltiples fuentes de datos expuestas mediante *OLE DB* en una máquina. Los objetos enumeradores hacen

posible obtener en un conjunto de registros todas las fuentes disponibles en la máquina (p.ej: Base de Datos, Mensajería, Servicio de Directorios, etc...).

- **Objetos de Fuentes de Datos** (*Data Source objects*): Son los que permiten obtener los datos propiamente dichos. Para acceder a un cierto proveedor *OLE DB*, el consumidor debe crear en primer lugar una instancia de un objeto de fuente de datos para este proveedor, proporcionando el identificador único (CLSID) del mismo y llamando a la función "CoCreateInstance". Cada objeto de fuente de datos expone la interfaz "IDBProperties", a través de la cual el consumidor podrá comunicarse con el proveedor para proporcionar y reclamar información previa a la conexión (como por ejemplo, la autentificación de usuario). La interfaz "IDBInitialize", por otra parte, se utiliza para establecer finalmente la conexión.

- **Sesiones** (*Sessions*): Las sesiones se generan, una vez disponemos de la fuente de datos, mediante la interfaz "IDBCreateSession". En el ámbito de la sesión es donde tendrán lugar las transacciones, la ejecución de consultas y comandos, es decir, las operaciones realmente productivas. Esta interfaz contiene el método "CreateSession", que permite la creación de la sesión. Una vez ha sido creada, los datos pueden obtenerse mediante la generación de objetos comando, que veremos a continuación, o directamente con la utilización de la interfaz "IopenRowset".

- **Conjunto de registros** (*Rowset*): Al final, la mayoría de las operaciones acaban en la obtención de un conjunto de registros. Para ello, *OLE DB* proporciona el objeto "Rowset", que permite a los proveedores exponer los datos en una tabla. Un "Rowset" es la abstracción universal para datos tabulares, que son los objetos de la utilización de *OLE DB*. Normalmente se logran como resultados de consultas.

- **Objetos comando** (*Command*): Un objeto comando es el mecanismo que se utiliza para enviar una sentencia SQL al gestor o al proveedor de datos que simula su existencia. La situación más normal es que la sentencia SQL contenga una consulta que devuelva un conjunto de registros que se almacenarán en un objeto "Rowset". Sin embargo, un comando no sólo está limitado a la ejecución de una consulta, sino que puede contener una sentencia SQL que o bien actualice los registros de una o varias

tablas (*Data Manipulation Language* o **DML**), o bien creen nuevos objetos de la base de datos, tales como tablas (*Data Definition Language* o **DDL**). Estos tipos de comandos son denominados en la terminología de *OLE DB text commands*, no devuelven *Rowsets* y la sintaxis SQL de los mismos debe someterse al dialecto SQL admisible por cada proveedor de datos.

- **Objetos Transacción** (*Transaction*): Los objetos transacción permiten la gestión de transacciones complejas. De esta forma podemos asegurar la consistencia de un conjunto de operaciones utilizando transacciones de una forma global confirmándolas ó abortándolas.

4.4. ODBC

ODBC (*Open DataBase Connectivity*, Conectividad de Bases de Datos Abiertas) define un método de conexión a fuentes de datos abiertas de manera que cualquier desarrollador pueda alcanzar esta información desde todo tipo de plataforma que disponga de esta funcionalidad. Del lado de la base de datos, esta conectividad abierta se obtiene mediante el uso de *drivers* contenidos en DLLs, los cuales transforman las funciones *ODBC API* en llamadas a funciones que soporta la fuente de datos que se usa. De la misma forma, dichos controladores transforman la sintaxis SQL de *ODBC* en aquella que acepta la fuente de datos. De esta manera, se puede acceder a diferentes fuentes simplemente cargando un *driver* distinto.

Uno de los beneficios del concepto *ODBC* estriba en la capacidad de tener accesibilidad a diversos formatos de ficheros simples, como si se trataran de bases de datos (aunque sólo sean ficheros de texto). Así, los *drivers ODBC* pueden conectarse a servidores de bases de datos reales como DB/2, SQL Server, Oracle e Informix (por nombrar unos pocos). En estos casos, el controlador *ODBC* transforma las llamadas a funciones *ODBC* en llamadas a funciones nativas del servidor, las cuales se pasan a este último. En el caso de fuentes locales basadas en ficheros (como ficheros de texto, dBASE, Excel XLS, Access MDB...), el controlador *ODBC* no utiliza la aplicación nativa que los creó, sino que actúa él mismo como servidor.

UDA (Universal Data Access) se apoya en *ODBC*, en el sentido de que aprovecha la formidable experiencia adquirida con el uso de esta metodología en la integración de sistemas de bases de datos. Sin embargo, *ODBC* ha sido en ocasiones denostada como tecnología por la dificultad de su administración y gestión, y por lo limitado de su rendimiento en ciertas plataformas.

ODBC sigue estando soportada en *UDA*, pero paulatinamente, y gracias a las mejoras que *OLE DB* incorpora, se irá reemplazando. Existe un puente transparente entre *OLE DB* y *ODBC*, por lo que a efectos prácticos podemos considerar *OLE DB* como un superconjunto de *ODBC*. Microsoft incluye con "*Visual Studio 6"* proveedores (*providers*) *OLE DB* para un buen número de gestores de bases de datos comerciales y para fuentes de datos no relaciones, siguiendo las directrices de *UDA*. De hecho, *SQL Server 7* ya implementa *OLE DB* como su interfaz de comunicación nativa.

De este modo, se pueden desarrollar aplicaciones utilizando los mecanismos de programación que se incorporan con *OLE DB* y *ADO*. El proveedor *OLE DB* para *ODBC* se conectará a la fuente de datos utilizando *ODBC* de manera totalmente transparente para el programador. Por otra parte, y como hemos comentado anteriormente, *OLE DB* incorpora además una serie de servicios añadidos proporcionados por componentes que extienden la funcionalidad del *Data Provider*. Estos servicios también son accesibles cuando se utiliza *OLE DB* y *UDA* para entrar en fuentes de datos *ODBC*, con lo que se puede extender la potencia de los *drivers ODBC*.

A pesar de que, como estamos viendo, *ODBC* sigue permitiendo el uso de *UDA*, las mejoras de rendimiento introducidas en *OLE DB* hacen recomendable la migración gradual y progresiva a la nueva tecnología. Cada *driver* implementa sus propios mecanismos para proporcionar servicios, como por ejemplo los cursores, mientras que en *OLE DB* esto se consigue mediante servicios separados. El segundo acercamiento permite una reutilización más eficiente de los recursos, ya que componentes reutilizables dan servicio a un conjunto de proveedores de datos, reduciendo el número de módulos de software necesarios en la máquina cliente. Consecuentemente, se logra una tangible mejora en la rapidez de ejecución y de aprovechamiento del espacio.

4.5. Modelos de Programación de Acceso a Datos

Estas interfaces y tecnologías de acceso a datos que hemos presentado son utilizables directamente a la hora de desarrollar aplicaciones, pero su naturaleza las hace poco adecuadas para lenguajes orientados a objetos, ya que tanto *ODBC* como *OLE DB* son APIs de distinta naturaleza. Precisamente por ello se han desarrollado unos modelos de programación que se apoyan en estas u otras tecnologías exponiendo y definiendo una jerarquía de objetos que los desarrolladores pueden utilizar de una forma más sencilla y natural. Podríamos decir que se trata de "agradables" envoltorios que permiten acceder a los datos sin tener que preocuparnos del API que hay que utilizar.

Ejemplos de este tipo de modelos son *RDO*, *DAO* y *ADO*. Asimismo, podríamos considerar JDBC como un modelo de programación, pero este último también es una tecnología propia y completa de acceso a datos. En cualquier caso, y para hacer más sencilla la comparación entre modos de acceso, describiremos brevemente esta tecnología en este punto.

- **DAO** (*Data Access Objects*, Objetos de Acceso a Datos): es un modelo de programación orientado a objetos desarrollado para la programación con el motor de bases de datos *Microsoft Jet*, que es el que gestiona las bases de datos de Access, aunque se puede utilizar independientemente. De esta forma, *Jet* ha sido incorporado como motor de base de datos a varios entornos de desarrollo *Microsoft*. Con la combinación *DAO* y *ODBC* se puede utilizar el motor *Jet* para acceder a bases de datos *ODBC*.

- **RDO** (*Remote Data Objects*, Objetos de Datos Remotos): este método implementa un conjunto de objetos para conseguir requerimientos especiales de acceso a bases de datos. *RDO* se constituye como una delgada capa sobre el API de *ODBC* (y el administrador de *drivers ODBC*) que facilita el establecimiento de conexiones, la creación de conjuntos de resultados (*ResultSets*) y la ejecución de procedimientos complejos. Otras funcionalidades de *RDO* son la flexibilidad en el uso de cursores, la ejecución de procedimientos almacenados que opcionalmente devuelven parámetros de salida o valores de retorno. También es posible delimitar el número de filas devueltas y monitorizar los mensajes de error generados por la fuente de datos sin

comprometer la ejecución de la consulta actual. *RDO* permite además la ejecución asíncrona, con lo que las aplicaciones no han de permanecer bloqueadas mientras esperan respuesta del servidor.

- **ADO** (*Active Data Objects*, Objetos de Datos Activos): es propuesta de *Microsoft* para sustituir y ampliar el modelo de interconectividad que hasta ahora se había venido utilizando en las plataformas de desarrollo de la casa. La firma ha desarrollado este nuevo modelo de acceso a bases de datos orientado a objetos por encima de *OLE DB* con el objetivo de implementar *UDA*. *ADO* ha sido creado para ser utilizado no sólo en las páginas web sino para quedarse, facilitando el proceso con los modelos de bases de datos actuales, aportando simplicidad y buenos rendimientos, coherencia en la notación e integración en las soluciones. Por otra parte, *ADO* ha sido diseñado conjuntamente con *OLE DB* y facilitará, por tanto, la migración al nuevo modelo. *ADO* está disponible desde *Visual C++ 5.0* y *Visual InterDev 1.0* mientras que *Visual Basic 5.0*, por imposibilidad temporal en el momento de la publicación de este producto, no lo soportaba. *Visual J++ 1.1*, por su parte, y tal u como el producto fue publicado, daba soporte para los ya tradicionales *RDO* (*ODBC*) y *DAO*, que se desarrollaron para las otras plataformas de *Microsoft*. *Visual Studio 6* incorpora *ADO 2.0* como el modelo de programación de preferencia en todas sus plataformas. De hecho, y a modo de ejemplo del soporte a *ADO* que *Microsoft* ofrece, en la versión 6.0 de *Visual J++* se incorpora un conjunto de clases, agrupadas bajo el nombre de *J/ADO*, que encapsulan y dan acceso a *ADO* al desarrollar en Java.

Y debido a las ventajas que nos puede proporcionar *ADO*, se ha optado por este modelo de Acceso a Datos para el desarrollo del presente proyecto, vamos a ver de forma más detallada las principales características de este modelo.

4.6. Arquitectura de ADO

ADO es un entorno orientado a objetos que facilita el acceso a bases de datos a las que se puede llegar bien desde el *OLE DB* u *ODBC*.

ADO proporciona un acceso consistente y de alto rendimiento a datos, a la vez que permite crear un cliente de base de datos o un objeto de negocio utilizando una aplicación, herramienta, lenguaje o incluso un navegador de Internet. *ADO* es el único interfaz de datos que los desarrolladores necesitan conocer para el desarrollo de soluciones cliente/servidor de N niveles basadas en datos y Web. *ADO* ha sido diseñado como un interfaz a nivel de aplicación fácil de usar al último paradigma de acceso a datos más potente de *Microsoft*, *OLE DB*.

ADO se constituye como una capa entre la base de datos y la aplicación cliente. El lenguaje nativo que utilizan los objetos *ADO* para comunicarse con las bases de datos es, no obstante, *OLE DB*, es decir, *ADO* se comunica directamente con los *OLE DB providers*. Cuando la base de datos deba ser accedida mediante *ODBC*, las peticiones de datos serán traducidas por las DLLs **MSDASQL.DLL** (SQL Server) o **MSDAORA.DLL** (Oracle). Ver Figura 4.3.

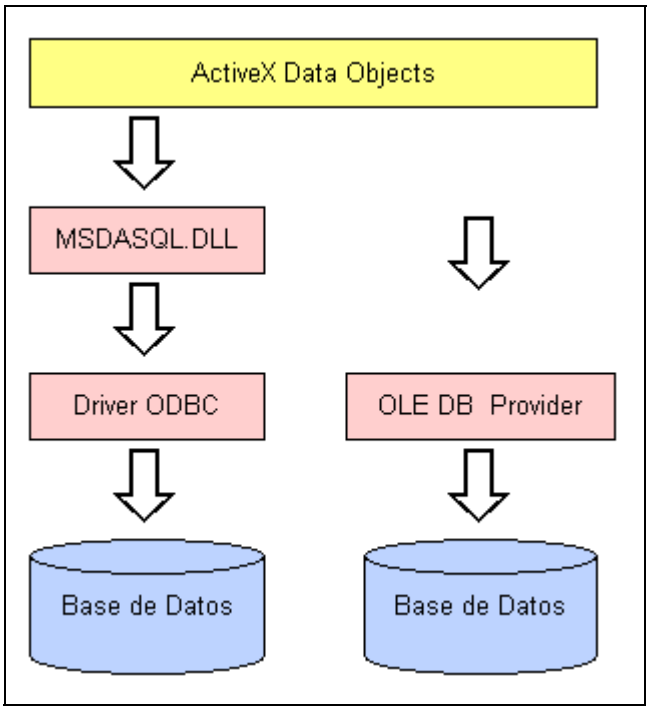


Figura 4.3. Arquitectura del modelo ADO.

Algunas de las características más importantes de *ADO* son:

- **Sencillez**: *ADO* presenta una jerarquía con un pequeño número de objetos, lo que hace a esta tecnología fácil de aprender y de utilizar.
- **Buen rendimiento**: La necesidad de crear pocos ejemplares de objeto para acceder a los datos provoca una estupenda funcionalidad en términos de potencia.
- **Gestión completa de eventos**: *ADO* proporciona un completo conjunto de eventos para los desarrolladores.
- **Potencia**: Soporta cualquier operación de datos, incluyendo cursores complejos, ejecución de procedimientos almacenados e incluso actualizaciones por bloques mediante el uso de cursores en cliente.

4.6.1. La Jerarquía de Objetos ADO

Todo el sistema *ADO* se base en sólo siete objetos: tres principales y cuatro secundarios. Como puede verse, *ADO* tiene un modelo de objetos extremadamente sencillo, que sin embargo, posee una gran potencia. Cualquier operación que desee realizarse con *OLE DB*, está soportada por estos siete elementos.

Como valores de muchas de las propiedades de los objetos *ADO* se utilizan constantes, como *adOpenForwardOnly*, *adOpenKeySet*, etc. Todas estas constantes empiezan por ad y están definidas en un archivo llamado *adovbs.inc* para Visual Basic Script y *adojavas.inc* para Javascript. Estos archivos los podrá encontrar en su instalación local de *ADO*. Para incluirlos en las páginas ASP es necesario utilizar un código de este tipo al comienzo de cada página ASP que use las constantes:

```
<!--#INCLUDE FILE="adovbs.inc"-->
```

Para que esta inclusión funcione el archivo *adovbs.inc* debe estar en el mismo directorio que la página ASP que lo incluye.

En la Tabla 4.1 podemos ver un resumen de los objetos *ADO* más importantes:

OBJETO	PROPIEDADES	MÉTODOS
Connection	ConnectionString	Open Close
Recordset	ActiveConnection Source LockType CursorType	Recordcount BOF EOF MoveFirst MoveLast MoveNext MovePrevious Move(n) AddNew Update Delete
Command	ActiveConnection CommandType CommandText	Execute
Error		
Field		
Parameter		
Property		

Tabla 4.1. Resumen de objetos ADO.

4.6.1.1. Connection

Representa una conexión a una base de datos. Este es el primer objeto que debemos crear para poder conectar con la base de datos. Tanto el objeto *ADO* que nos permite acceder a los datos (*Recordset*) como el que nos permite realizar consultas (*Command*) disponen de una propiedad llamada *ActiveConnection* que es una referencia al objeto *connection* que enlaza con la base de datos a la que queremos atacar. De ahí que el primer paso sea crear la conexión.

Propiedades y métodos más relevantes:

- ***ConnectionString***: Es una cadena de caracteres con la información necesaria para establecer una conexión con la fuente de datos. Por tanto, es la propiedad básica de este objeto. Aunque hay hasta 7 argumentos distintos que se pueden suministrar en esta cadena, los básicos son el DSN que identifica al archivo de base de datos y el login y password si existen. Los argumentos se separan con punto y coma, por ejemplo:

  ```
  "DSN=midsn; UID=milogin; PWD=micontraseña"
  ```

- ***Open***: Abre la conexión con la base de datos. Si antes hemos asignado la propiedad *ConnectionString*, este método no necesita parámetros.

- ***Close***: Cierra la conexión con la base de datos, por ejemplo:

  ```
  <%
  Set miconexion = Server.CreateObject("ADODB.Connection")
  miconexion.ConnectionString = "DSN=midsn"
  miconexion.Open
  ' .....
  miconexion.Close
  %>
  ```

4.6.1.2. Recordset

Este es el objeto *ADO* más importante ya que es con el que accederemos directamente a los datos de las tablas, tanto para leerlos como para modificarlos.

Un objeto *Recordset* representa una tabla, que puede ser una tabla física de la base de datos o bien una obtenida mediante una operación como un filtrado o sentencia SQL. En cualquier caso, el objeto representa a la tabla con todos sus registros, aunque sólo uno de ellos es el activo. El registro activo es en el que podemos leer o modificar los valores de los campos. También se le llama cursor.

Para una mejor compresión y puesto que son numerosos, hemos dividido sus propiedades por categorías de utilidad.

1) Propiedades que hacen referencia al origen de los datos:

Estas propiedades deben ser asignadas a todo *Recordset*, pues le dicen la base de datos y la tabla de la que obtener sus datos.

- ***ActiveConnection***: Como se ha comentado antes a esta propiedad se le debe asignar un objeto *connection* que se haya creado previamente. Indicará al *Recordset* la base de datos en la que buscar su tabla.

- ***Source***: Indica al objeto *Recordset* la tabla a la que representará. A la propiedad *Source* se le asigna normalmente una cadena de caracteres con el nombre de la tabla. Sin embargo, también es posible asignarle una sentencia SQL y entonces el objeto *Recordset* referenciará al resultado de aplicar dicha sentencia.

- ***LockType***: Indica el tipo de bloqueo que se realiza sobre la base de datos. Por defecto toma el valor *adLockReadOnly* que como su nombre indica sólo sirve para leer datos. Si deseamos editar o añadir registros, tendremos que cambiar esta propiedad por otro valor. El más usado es *adLockOptimistic* que permite editar la base de datos o añadir registros realizando un bloqueo optimista (sólo cuando sea estrictamente necesario).

- ***CursorType***: El tipo de cursor que se utiliza para recorrer los registros de un *recordset*. Por defecto toma el valor *adOpenForwardOnly* que como su nombre indica sólo permite moverse hacia adelante. Por ello si queremos utilizar libremente todos los métodos de movimiento de un *recordset* (*MoveFirst*, *MoveTo*, *MoveNext*, *MovePrevious*, etc) tendremos que cambiar el cursor por uno más potente. Esto puede hacerse asignando a esta propiedad el valor *adOpenKeyset*

Vamos a ver un ejemplo en el que crearemos en primer lugar un objeto *Connection* y luego un *Recordset* al que se asigna dicho objeto. Este ejemplo es una especie de esqueleto para los demás que sigan: Se supone que debe hacerse algo parecido a esto para abrir el *Recordset*. Es decir, en los restantes ejemplos escribimos sólo el código que iría en el lugar de los puntos suspensivos de este script (NOTA: Recuerde que si hay que añadir registros es necesario cambiar el *LockType* antes de inicializar el *recordset*).

```
<%
Set miconexion = Server.CreateObject("ADODB.Connection")
miconexion.ConnectionString = "DSN=midsn"
miconexion.Open
Set mirecordset = Server.CreateObject("ADODB.Recordset")
mirecordset.ActiveConnection = miconexion
mirecordset.Source = "Clientes"
mirecordset.Open
..... ' Operaciones con los datos de la tabla Clientes.
mirecordset.Close
miconexion.Close
%>
```

2) Propiedades que hacen referencia al número de registros:

- ***RecordCount***: Número de registros de la tabla a la que representa el objeto *recordset*, por ejemplo:

```
<h3>Tenemos <%= rstClientes.RecordCount%> clientes en nuestra base de datos</h3>
```

- ***EOF***: Acrónimo de *End Of File*. Vale TRUE si estamos en el último registro y FALSE si no. Se usa mucho como condición en bucles *while*, los cuales se ejecutan hasta llegar al último registro.

- ***BOF***: Acrónimo de *Begin Of File*. Vale TRUE si estamos en el primer registro y FALSE si no.

3) Métodos para mover el cursor (registro activo):

Estos métodos pueden funcionar todos o no dependiendo del tipo de cursor creado. El tipo de cursor se asigna en la propiedad *CursorType*. Por ejemplo, para asignar un cursor *adOpenKeySet* que permita moverse hacia adelante o hacia atrás:

```
<%
Set miconexion = Server.CreateObject("ADODB.Connection")
miconexion.ConnectionString = "DSN=midsn"
miconexion.Open
Set mirecordset = Server.CreateObject("ADODB.Recordset")
mirecordset.ActiveConnection = miconexion
mirecordset.Source = "Clientes"
mirecordset.CursorType = adOpenKeySet
mirecordset.Open
..... ' Operaciones con los datos de la tabla Clientes.
mirecordset.Close
miconexion.Close
%>
```

Nótese que la asignación de propiedades al *recordset* debe hacerse antes de invocar el método *open* que es el que lo inicializa con registros.

- ***MoveFirst***: Mueve el cursor al primer registro de la tabla.
- ***MoveLast***: Mueve el cursor al último registro de la tabla.
- ***MoveNext***: Mueve el cursor al siguiente registro.
- ***MovePrevious***: Mueve el cursor al registro anterior.
- ***Move(n)***: Mueve el cursor al registro con índice n.

Ejemplo de un bucle que recorre todos los registros de una tabla:

```
<%
mirecordset.MoveFirst
do while not mirecordset.EOF
    ..... ' Tratamiento de datos
    mirecordset.MoveNext
loop
%>
```

4) Lectura y modificación de los campos del registro activo.

La sintaxis para acceder a los datos de un campo del registro activo de un *recordset* es:

```
mirecordset ("Domicilio")
```

Esto se usa tanto para leer como asignar valores. En nuestro ejemplo, el objeto *mirecordset* representa a una tabla uno de cuyos campos tiene el nombre "Domicilio". Así, con la expresión

```
dom = mirecordset("Domicilio")
```

Leemos el valor del campo domicilio del registro activo y se lo asignamos a la variable *dom*. Con la expresión:

```
mirecordset("Domicilio") = "Avda. Andalucía 19, 1º C"
mirecordset.Update
```

Asignamos un valor al campo "Domicilio" del registro activo. Tras la edición del registro, es necesario llamar al método ***Update***. El motivo es que los cambios en el registro activo se realizan sobre un *buffer* (espacio de almacenamiento intermedio) y no sobre el registro propiamente dicho.

El *recordset rstClientes* representa a nuestra tabla de clientes que, entre otros, tiene los campos Provincia e IVA. El siguiente bucle recorre todos los clientes, comprueba su provincia y le asigna el IVA correspondiente: 0 para Canarias y 16 para los demás:

```
<%
rstClientes.MoveFirst
do while not rstClientes.EOF
    if rstClientes("Provincia") = "Las Palmas" or rstClientes("Provincia") = "Tenerife" Then
       rstClientes("IVA") = 0
    else
        rstClientes("IVA") = 16
    end if
    rstClientes.Update
    rstClientes.MoveNext
loop
%>
```

Métodos para agregar o eliminar registros de la tabla

- ***AddNew*** y ***Update***: Crear un nuevo registro involucra dos métodos: Primero *AddNew* crea un nuevo registro en blanco. Después asignamos valores a los distintos campos del registro. Por último invocamos el método *Update* para que se haga efectiva la incorporación del nuevo registro con los valores asignados.

En este ejemplo incorporamos un nuevo cliente a nuestra base de datos:

```
<%
rstClientes.AddNew
rstClientes("Nombre") = "Nombre1 Apellidos1"
rstClientes("Direccion") = "C/ Direccion1"
rstClientes("Localidad") = "Localidad1"
rstClientes("Profesion") = "Profesion1"
rsClientes.Update
%>
```

Recuerde que para que esto funcione hay que asignar un tipo de bloqueo que permita la edición de la base de datos. Esto se hace asignando a la propiedad *LockType* del *recordset* uno de estos valores: *adLockOptimistic*, *adLockPessimistic* o *adLockBatchOptimistic*

- ***Delete***: Eliminar el registro activo es muy fácil. Basta con invocar este método. Por ejemplo, este bucle elimina todos los clientes morosos de la base de datos:

```
<%
rstClientes.MoveFirst
do while not rstClientes.EOF
    if rstClientes("Deuda") > 0 Then
        rstClientes.Delete
    end if
    rstClientes.MoveNext
loop
%>
```

4.6.1.3. Command

Este objeto es la representación de un comando que se envía a la base de datos. El objeto *Command* es altamente dependiente del driver *ODBC* o del *OLE DB provider*, ya que describe un comando a enviar a la fuente de datos, por tanto no es recomendable utilizarlo cuando se pretenda desarrollar aplicaciones genéricas e independientes del modelo de base de datos.

Cada gestor de base de datos tendrá su propio conjunto de comandos admisibles, con su sintaxis propia, con lo que el objeto comando lleva aparejada una inherente heterogeneidad.

El objeto *Command*, contiene la definición de un comando SQL cualquiera que pueda, por tanto, realizar tareas como realizar una consulta, actualizar o insertar datos o ejecutar procedimientos almacenados en el servidor. Sus propiedades y métodos dependerán de la naturaleza de la fuente de datos. El comando puede ser de estos 3 tipos:

- Llamada a un procedimiento almacenado en la base de datos.
- El texto de una sentencia SQL.
- El nombre de una tabla.

Propiedades y métodos más relevantes:

- ***ActiveConnection***: Es una referencia al objeto *connection* que enlaza con la base de datos. Es imprescindible asignar esta propiedad antes de invocar el método *command.execute* para ejecutar el comando.

- ***CommandType***: Este es el tipo del comando a ejecutar. Para una consulta SQL utilizamos la constante *adCmdText*, para llamar a un procedimiento almacenado adCmdStoredProc, etc...

- ***CommandText***: Este es el texto del comando. Si se trata de una consulta SQL (lo habitual), esta propiedad es simplemente una cadena con el texto de la consulta.

- ***Execute***: El método que ejecuta la consulta. El resultado de la ejecución del comando normalmente será un *recordset*.

En el ejemplo, obtenemos un *recordset* con un filtrado de la tabla *Clientes* en el que se toman sólo aquellos clientes cuya provincia es "Málaga".

```
<%
Set miconexion = Server.CreateObject("ADODB.Connection")
miconexion.ConnectionString = "DSN=midsn"
miconexion.Open
Set SqlCommand = Server.CreateObject("ADODB.Command")
SqlCommand.ActiveConnection = miconexion
SqlCommand.CommandText = "SELECT * FROM Clientes WHERE Provincia = 'Málaga'"
Set rstNavarros = SqlCommand.execute
'.....
miconexion.Close
%>
```

4.6.1.4. Error

Permite examinar los errores que puedan producirse al realizar una conexión con la fuente de datos, es decir, con el *Data Provider*. Cada vez que se produce un error, y para cada uno de ellos, se crea uno de estos objetos que se almacena en la colección *Errors* de *Connection*.

4.6.1.5. Field

Cada *recordse*t contiene una colección de *Fields* que representa las columnas del conjunto de registros que integran el resultado de la consulta. Cada uno de los elementos de la colección es un objeto *Field*.

4.6.1.6. Parameter

Representa a un parámetro para un objeto *Command*, sea para una consulta parametrizada o para la ejecución de un procedimiento almacenado que los incluya. Un ejemplo de consulta parametrizada sería la siguiente:

```
SELECT * from ?
```

Donde el signo interrogación representa un parámetro que podrá ser sustituido por un valor en tiempo de ejecución. Cada uno de los parámetros presentes en un comando es representado por un objeto *Parameter*.

4.6.1.7. Property

Representa una propiedad de un objeto *ADO* definida por el proveedor de datos. La colección *Properties* proporciona la información sobre las características de los diferentes objetos *Connection*, *Command*, *Recordset* y *Field*. Cada objeto de *Property* es accesible a través de la colección *Properties* de cada uno de estos objetos. El objeto colección *Properties*

puede contener 0 ó más objetos *Property*. La razón por la que tenemos la colección *Properties* en *ADO* es debido a que se utiliza con diferentes fuentes de datos, cada una de las cuales pueden tener distintas propiedades, de ahí el carácter dinámico de este objeto. Por ejemplo si utilizamos el proveedor de *OLE DB* para *Jet* nos permitirá acceder a las propiedades específicas de seguridad de Jet, mientras que utilizando directamente *ADO* con el proveedor *OLE DB* la colección *Properties* se rellenará con los valores por defecto del proveedor.

4.6.2. Acceso a Datos con ADO

En este apartado vamos a presentar los procedimientos básicos para acceder a datos utilizando *ADO*. Para dar una idea más clara de que *ADO* es utilizable desde cualquiera de las plataformas de desarrollo de *Microsoft*, vamos a presentar estos ejemplos con sintaxis común de *Visual Basic* y *VBScript* para ASP, que es la utilizada en este proyecto.

En primer lugar, antes de realizar cualquier modificación de los datos, es necesario establecer la conexión con la fuente que los representa.

El primer paso es crear la conexión, y para ello es preciso crear un objeto *Connection* al que llamaremos *miconexion* mediante el Método *CreateObject* del objeto *Server*, la sintaxis es la siguiente:

```
Set miconexion = Server.CreateObject(“ADODB.Connection”)
```

Para establecer la conexión, una vez creado el objeto, debemos llamar a su método *Open* según la siguiente sintaxis:

```
Connection.Open Cadena_de_conexión, usuario, password
```

A continuación vemos un ejemplo de cuál sería el aspecto de cada uno de estos argumentos para la apertura de la conexión:

```
<%
Set miconexion = Server.CreateObject("ADODB.Connection")
Set rs = Server.CreateObject("ADODB.Recordset")
miconexion.ConnectionTimeOut = 10
miconexion.CommandTimeOut = 20
miconexion.Open "DSN=DBOracle2000","scott","tiger"
rs = miconexion.Execute ("Select * from personal")
…
rs.Close
Set rs = Nothing
Set miconexion = Nothing
%>
```

En el ejemplo vemos que una vez creados los objetos *Connection* asociado a la variable "miconexion" y *Recordset* asociado a la variable "rs", hemos asignado algunos parámetros al objeto conexión. El *ConnectionTimeOut* especifica el máximo tiempo en segundos que vamos a esperar a establecer la conexión antes de generar un error, mientras que *CommandTimeOut* es equivalente al anterior pero cuando ejecutemos un comando sobre la conexión. A continuación se establece la conexión con el Método *Open* del objeto *Connection* y por último se ejecuta una consulta sobre la conexión y el resultado se asigna a un objeto *Recordset* asociado a la variable "rs". Finalmente debemos cerrar el *recordset* y liberar los objetos creados asignándolos a *Nothing*. Existen tres modos para crear un recordset:

- Utilizar el método "Execute" del objeto "Connection": Ya visto en el ejemplo anterior.
- Método "CreateObject" del objeto "Server": haremos uso de él para generar el objeto "Recordset" y, posteriormente, utilizaremos el método "Open".

```
<%
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open "Select * from personal","DSN=DBOracle2000","scott","tiger"
…
rs.Close
Set rs = Nothing
%>
```

- Utilizar el método “Execute” del objeto “Command”.

```
<%
Set miconexion = Server.CreateObject(“ADODB.Connection”)
Set micomando = Server.CreateObject(“ADODB.Command”)
miconexion.Open “DSN=DBOracle2000”,”scott”,”tiger”
micomando.ActiveConnection = miconexion
micomando.CommandType = adCmdText
micomando.CommandText = “Select * from personal”
Set rs = micomando.Execute
…
rs.Close
Set rs = Nothing
Set micomando = Nothing
Set miconexion = Nothing
%>
```

En este ejemplo, creamos los objetos que vamos a necesitar “miconexion” y “micomando”, establecemos la conexión, a continuación asignamos algunas propiedades al objeto comando, establecemos la conexión activa que utilizaremos cuando ejecutemos el comando con *ActiveConnection*, especificamos el tipo de comando a ejecutar *CommandType*, en este caso es una sentencia SQL aunque también podemos hacer llamadas a *Procedimientos Almacenados*, a continuación asignamos el comando a ejecutar *CommandText* y por último llamamos al método *Execute* encargado de ejecutar el comando y asignar el resultado a un *Recordset* al que hemos llamado “rs”. Finalmente debemos cerrar el *Recordset* y liberar los objetos creados asignándolos a *Nothing*.

Capítulo 5

El Cliente FSQL: VisualFSQL

Ya se indicó en capítulos anteriores que el programa Cliente FSQL es el encargado de hacer de interfaz gráfica y visual entre el usuario y el Servidor FSQL. Este programa puede ser muy simple, pero es muy importante, pues va a permitir al usuario ejecutar sentencias difusas en FSQL.

Además, para el caso concreto de este proyecto se ha desarrollado un programa cliente muy avanzado, llamado "**Visual FSQL**", para realizar consultas complejas de forma totalmente asistida y con un entorno visual muy accesible para cualquier usuario con unos mínimos conocimientos de bases de datos y lógica difusa. En el siguiente capítulo se incluye un *manual de usuario* de este programa, y en este pretendemos mostrar el desarrollo.

5.1. Objetivo y Funcionamiento

El programa Cliente FSQL es el encargado de guiarnos visualmente a través de varias pantallas y construir de forma asistida cualquier sentencia de selección. Posteriormente se envía la sentencia en lenguaje FSQL al Servidor FSQL y obtenemos de éste los errores que contenga dicha sentencia (usando la función *FSQL2SQL*). Si la sentencia no contiene errores, entonces el Cliente debe obtener del Servidor dicha sentencia FSQL traducida a una sentencia SQL equivalente. Hecho esto, enviará la sentencia SQL al SGBD y, en su caso, obtendrá la respuesta y efectuará las acciones oportunas con los resultados. Además, estas sentencias FSQL generadas por el asistente o tecleadas manualmente las podremos almacenar en un fichero de sentencia Visual FSQL (.vfq) ó FSQL(.fq) dependiendo del caso.

La relaciones del programa Cliente *Visual FSQL* con el Servidor FSQL son efectuadas de forma transparente al usuario: el usuario no tiene que conocer nada acerca del Servidor FSQL. Vamos a repasar las operaciones principales que debe realizar el Cliente FSQL:

1. **Conexión al SGBD**: En esta operación el programa cliente debe solicitar del usuario el nombre del servicio de la Base de Datos (DNS), su nombre de conexión (login) y su clave de acceso (password). Tras esto, intenta abrir una sesión en el SGBD y, si se consigue, asigna un número de sesión a la conexión efectuada. Una vez realizada la validación se acepta al usuario a nivel de aplicación y cada vez que este usuario intente acceder o realizar una consulta a la BD se establecerá una conexión independiente para cada consulta lanzada, es decir, se realizan conexiones temporales para no saturar el servidor con una conexión permanente. También, es posible el acceso multiusuario o incluso multisesión por parte de un mismo usuario desde distintas ubicaciones, o desde distintas instancias del programa Visual FSQL en el mismo terminal.
2. **Edición manual de la sentencia en FSQL (ó SQL)**: En este apartado podemos escribir en modo texto sentencias SQL ó FSQL. En este caso el usuario deberá conocer el lenguaje SQL y su extensión a FSQL. Además, es posible almacenar estas sentencias en un fichero para posteriormente poderlas recuperar y reutilizar.
3. **Asistente Visual para construir sentencias en FSQL**: Este es el apartado más atractivo y sobre el que se fundamenta el desarrollo de este proyecto. Consiste en una serie de pantallas que nos van guiando visualmente en la construcción de cualquier sentencia FSQL de una forma cómoda para el usuario, y sin necesidad de conocer en profundidad el lenguaje FSQL. Este asistente consta de 9 pantallas (Selección, Relaciones entre tablas, Condiciones, Organizar condiciones, Agrupación, Organiza grupos, Ordenación, Resultados y Editar Consulta) por las que iremos avanzando en la construcción de la sentencia. También tendremos la posibilidad de almacenar la sentencia visual construida en un archivo con un formato específico, para así poderla recuperar en otro momento como una sentencia Visual del asistente.
4. **Obtener y mostrar los errores** (si los hubiera): En caso de producirse un error a la hora de traducir la sentencia de FSQL a SQL ó al lanzar la sentencia SQL definitiva, se muestra en pantalla el código de error y la descripción del mismo.

5. **Obtención de Resultados**: Una vez ejecutada la sentencia satisfactoriamente, se mostrarían los resultados de los registros obtenidos. Igualmente, se nos mostrará en pantalla el tiempo empleado en ejecutar la sentencia.
6. **Desconexión del SGBD**: Esta operación supone que no se van a efectuar más operaciones con el SGBD. Realmente esta operación no es estrictamente necesaria y en general, bastará con cerrar el programa Cliente. Pero por seguridad y para que nadie pueda utilizar nuestra sesión abierta es recomendable desconectarse del sistema. Además, el control de la desconexión es totalmente a nivel de aplicación en cuanto a validación del usuario conectado, puesto que las sentencias se lanzan al servidor FSQL y una vez obtenidos los resultados nos desconectamos del SGBD, es decir siempre trabajamos de forma desconectada y el control de la conexión se realiza a nivel de aplicación no del SGBD.

Adicionalmente se ha incluido también un pequeño módulo de "*Gestión online*" a través del cual podemos consultar y modificar registros de nuestra base de datos de una forma totalmente simplificada. Hay que indicar que este módulo es operativo parcialmente y que únicamente constituye una ayuda complementaria a nuestro verdadero objetivo que es el Asistente Visual de Consultas Difusas.

5.2. Programación del Cliente: Visual FSQL

Ya hemos visto en el apartado anterior y a grandes rasgos los módulos básicos que se han desarrollado en la construcción del Cliente *Visual FSQL*. Vamos a profundizar un poco más en la estructura y programación de la aplicación, para ello es importante conocer los capítulos anteriores donde se explicaron las tecnologías utilizadas para desarrollar este programa Cliente.

A grandes rasgos vamos a indicar como se realiza el proceso de ejecución de una sentencia FSQL:

1. Llamamos a la función *FSQL2SQL* del paquete *FSQL_PKG*, para descubrir el número de errores de la sentencia: Num_Errores = *FSQL_PKG.FSQL2SQL*(sentencia);
2. Si encontramos errores (Num_Errores>0) los mensajes de error correspondientes se

encuentran en el campo Msg_error de la vista *FSQL_ERRORS*, uno tras otro ordenados por el campo Indice en orden de aparición.

3. En caso de que no existan errores, la sentencia SQL resultante de la traducción estará almacenada en la vista *FSQL_QUERY* y será obtenida concatenando el campo *Atributo* en el orden que indique el campo Indice.
4. Después de obtener la sentencia SQL resultante, el Cliente FSQL deberá enviar esta sentencia al SGBD como si se tratara de una sentencia normal en SQL. Si la sentencia es una consulta, se deberán recuperar los resultados de la misma.
5. Cuando se ha terminado de ejecutar sentencias FSQL se debe llamar al procedimiento *FSQL_FIN*, borrando los datos temporales que ya no son útiles. Este procedimiento no tiene argumentos ni devuelve ningún dato. Esta operación no hay que efectuarla tras cada sentencia FSQL sino una única vez, al final.

En nuestro programa *Visual FSQL* también se ha incorporado la opción de consultar la sentencia SQL resultante de traducir la sentencia FSQL original.

5.2.1. Ubicación de archivos y carpetas de la Aplicación

Tal y como hemos comentado, el programa cliente se ha desarrollado en un lenguaje para su ejecución a través de un entorno web y por tanto ser accesible desde Internet con cualquier navegador. Las ventajas ya se han comentado en capítulos anteriores y ahora vamos a hacer referencia a la estructura de algunas partes del código empleado en el desarrollo. Se ha optado por mantener una estructura simple de carpetas y archivos ordenada jerárquicamente para facilitar la actualización y el mantenimiento de los elementos del programa, ver ubicación de archivos en el Servidor Web (Figura 5.1).

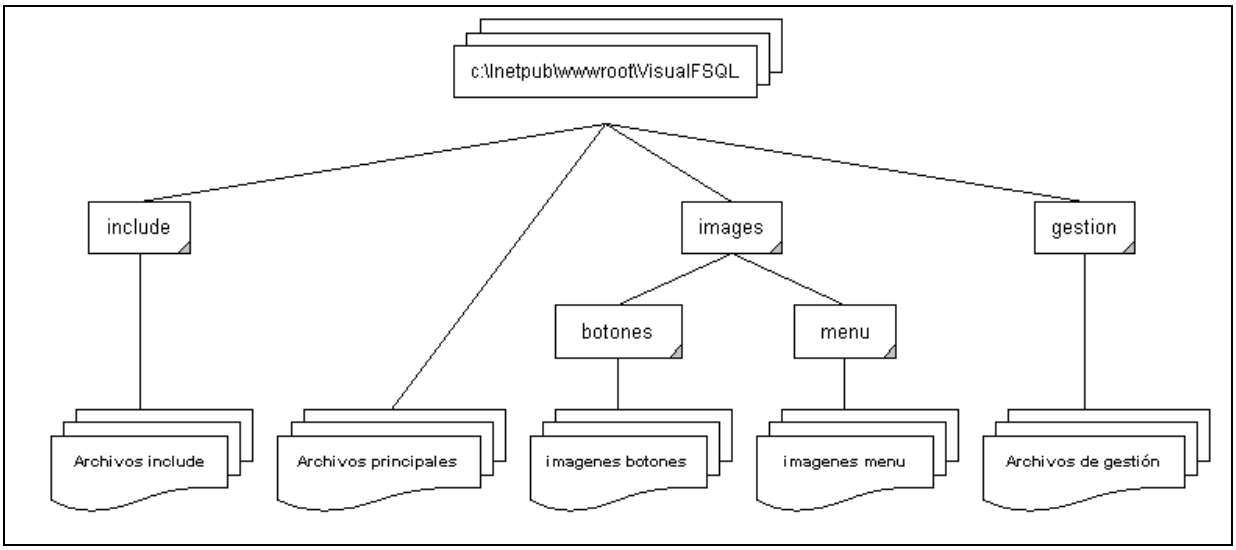


Figura 5.1. Esquema de ubicación de los archivos de la aplicación VisualFSQL.

Tenemos una carpeta principal o raíz donde están los principales archivos de código del programa "Visual FSQL", y su ubicación es la siguiente:

- "C:/Inetpub/wwwroot/VisualFSQL/": Archivos principales de la aplicación (.asp). A partir de esta estructura cuelgan las siguientes carpetas.
- "/include": Archivos que se incluyen en los principales y sirven como plantillas o código genérico y reutilizable en distintos módulos del programa (.asp y .inc).
- "/images/botones": Archivos gráficos de imágenes de los botones (.gif).
- "/images/menu": Archivos gráficos de imágenes de las barras de menús (.gif).
- "/gestion": Archivos utilizados para la *gestión online* de nuestra base de datos (.asp).

La estructura genérica que se ha seguido es utilizar una página ASP (.asp) por cada pantalla de la aplicación: Así tendremos varias páginas .asp equivalentes a las pantallas que representan.

En la Tabla 5.1 podemos ver estas páginas.

PÁGINA ASP	PANTALLA QUE REPRESENTA
conexion.asp	Verificación e información de la conexión del SGBD
desconexion.asp	Desconexión del SGBD
terminalsql.asp	Consultas SQL/FSQL en modo texto
terminalsql.asp	Resultados de las consultas escritas en modo texto
consultavisual_select.asp	Consulta Visual: Parte de la Selección
consultavisual_relaciones.asp	Consulta Visual: Establecemos las relaciones entre tablas
consultavisual_condiciones.asp	Consulta Visual: Condiciones de la consulta
consultavisual_organizacondiciones.asp	Consulta Visual: Organiza las condiciones e la consulta
consultavisual_agrupa.asp	Consulta Visual: Agrupa los resultados de la consulta
consultavisual_organizaagrupaciones.asp	Consulta Visual: Organiza los grupos resultados de la consulta
consultavisual_ordena.asp	Consulta Visual: Ordena los resultados de la consulta
consultavisual_finalizar.asp	Consulta Visual: Resultados de ejecutar la sentencia
consultavisual_editarconsulta.asp	Consulta Visual: Podemos editar manualmente la sentencia generada con el asistente visual.
fso_leerconsultas.asp	Ficheros: Lee una sentencia modo texto de un fichero (.fq)
fso_grabarconsultas.asp	Ficheros: Graba una sentencia modo texto a un fichero (.fq)
fso_leerconsultasvisual.asp	Ficheros: Lee una sentencia visual de un fichero (.vfq)
fso_grabarconsultasvisual.asp	Ficheros: Graba una sentencia visual a un fichero (.vfq)

Tabla 5.1. Resumen de las principales páginas ASP de VisualFSQL.

Se puede observar la nomenclatura seguida para nombrar las páginas ASP. Por ejemplo, para las páginas relacionadas con las pantallas del Asistente Visual se ha optado por nombrarlas como "*consultavisual_*" seguida de un nombre identificativo de la pantalla que representa. Igualmente para las pantallas relacionadas con la lectura/grabación de ficheros con las sentencias, en las que se utiliza "*fso_*" (file system object: *Objeto Sistema de Ficheros*, que es intrínseco del lenguaje ASP) seguida de la operación que realiza. Para el resto de carpetas y archivos la nomenclatura utilizada es muy aclarativa de la pantalla o función que representa cada página ASP.

5.2.2. Ejemplos de código

En este punto queremos hacer una pequeña exposición de algunos ejemplos de código de las partes o trozos de módulos más significativas de la aplicación. De cualquier manera no se va a detallar todo el código de una forma exhaustiva, sino que queremos aclarar como se han programado las instrucciones más específicas y principales del Cliente Visual FSQL.

- **Conexión** (conexión.asp)

```
...
<!--#include file="include/settings_ini.asp"-->
```

```
<%
‘Abrimos la conexion
Set conexion = Server.CreateObject(“ADODB.Connection”)
conexion.Open cnxVisualFSQL

‘Verificamos si se ha establecido conexion
session(“acceso”)=”0”
if (conexion.State) then
        session(“acceso”)=”1”
end if
...
%>
```

Se ha incluido el archivo “*settings_ini.asp”* de la carpeta “*include*”, a través del cual se construye la cadena de conexión *cnxVisualFSQL* con los parámetros recogidos en el formulario de conexión (USER: usuario y PASS: contraseña). Posteriormente se crea el objeto conexión (*ADODB.Connection*) y se establece un intento de abrir la conexión (*Open*) especificada en la variable *cnxVisualFSQL.*

Por último, mediante la propiedad *State* del objeto *conexion*, comprobamos si se ha establecido correctamente y, en ese caso, establecemos nuestra variable de session (*acceso*) a 1. En caso contrario la dejamos a 0. Con este pequeño trozo de código se puede ver el control realizado para validar el acceso al SGBD. La construcción de la cadena de conexión se realiza en *settings_ini.asp*.

```
<%
‘ ****************************************
‘ Archivo settings_ini.asp
‘ Datos de la Conexion a la Base de Datos
‘ (DSN,Usuario,Password)
‘ ****************************************
cnxVisualFSQL = “Provider=” & Trim(session(“SGBD”)) & “;Data Source=” &
Trim(session(“SERVER”)) &”;User ID=” & Trim(session(“USER”)) & “;Password=” &
Trim(session(“PASS”))
...
%>
```

Consulta Visual: Selección (consultavisual_select.asp)

```
<%
‘ ****************************************
‘ Archivo consultavisual_select.asp
‘ ****************************************
%>
<!--#include file=”include/settings.asp”-->
<%On Error resume Next%>
<!--#include file=”include/obtieneTablas_Oracle.asp”-->
<!--#include file=”include/pseudocolumnas.inc”-->
<html>
<head>
<title>Consulta Visual - SELECCIÓN</title>
<meta http-equiv=”Content-Type” content=”text/html; charset=iso-8859-1”>
<link rel=”stylesheet” href=”estilos_VisualFSQL.css” type=”text/css”>

<SCRIPT language=”Javascript”>
function seleccionCampo(lado){
var indexcampo = eval(“document.form.l_campos”+lado+”.selectedIndex”);
pseudocol = document.form.tipocampo[1].checked;
if (indexcampo<0)
        indexcampo=”0”;
var idcampo = eval(“document.form.l_campos”+lado+”.options[”+indexcampo+”].value”);


//Si no selecciono ninguna tabla ó he seleccionado pseudocolumnas no obtengo los tipos
if (idcampo!=”” && !pseudocol)
        cambiaTipoAtributo(lado);
else
        eval(“document.form.t_tipoatributo”+lado+”.value=””);
cambiaComentarioAtributo(lado);
}
…
%>
```

En este ejemplo vemos que se incluye el archivo “*settings.asp*” donde tenemos una serie de funciones y, además, se construye la cadena de conexión *cnxVisualFSQL*.

Posteriormente incluimos el archivo "*obtieneTablas_Oracle.asp*" donde realizamos la carga de todas las tablas del sistema y las vamos escribiendo como arrays de código Javascript, para posteriormente poder acceder a estos arrays y trabajar con la lista de tablas y campos asociados.

También hemos puesto un ejemplo de una función Javascript que funciona en modo cliente en el Navegador web. En este caso la función *seleccionCampo( )* se activa cada vez que seleccionamos un campo de la pantalla, y lo que hace es llamar a otras dos funciones que se encargan de mostrar información del tipo y comentario del atributo seleccionado, para ello cambia los valores de las cajas (*TipoAtributo*) y (*ComentarioAtributo*) en función del campo seleccionado.

Vamos a comentar el contenido de "*obtieneTablas_Oracle.asp*":

Dependiendo de las tablas (Usuario, Sistemas, Vistas, etc...) que vayamos a mostrar construimos la sentencia SQL. A continuación vamos construyendo los arrays de Javascript con los códigos y nombres de los elementos recuperados:

```
<SCRIPT language="Javascript">
<%
On Error Resume Next
'Abrimos la conexion
Set conexion = Server.CreateObject("ADODB.Connection")
conexion.Open cnxVisualFSQL
set rstablas=server.createobject("ADODB.recordset")

'Definimos los Arrays de los tipos de tabla
response.write "var tiptab = new Array(); " & Vbcr
response.write "tiptab[""ttab1""] = new Array(); " & Vbcr
response.write "tiptab[""ttab1""][tiptab[""ttab1""].length] = new opt(""0"",""(Seleccione una)""); " & Vbcr
...

select case session("tipotabla"&t)
        'Para las Tablas de Usuario
        case "0":
        sqltablas="select * from all_objects where (object_type = 'TABLE' AND owner = '" &
UCase(session("USER")) & "') Order by OBJECT_NAME"
```

```
        'Para Tablas de Sistema
        case "1":
        sqltablas="select * from all_objects where (object_type = 'TABLE' AND owner <> '" &
UCase(session("USER")) & "') Order by OBJECT_NAME"
        'Para Vistas de Usuario
        case "2":
        sqltablas="select * from all_objects where (object_type = 'VIEW' AND owner = '" &
UCase(session("USER")) & "') Order by OBJECT_NAME"
        'Para Vistas de Sistema
        case "3":
        sqltablas="select * from all_objects where (object_type = 'VIEW' AND owner <> '" &
UCase(session("USER")) & "') Order by OBJECT_NAME"
        'Por defecto(al iniciar)
        case else:
        sqltablas="select * from all_objects where (object_type = 'TABLE' AND owner = '" &
UCase(session("USER")) & "') Order by OBJECT_NAME"
end select
rstablas.open sqltablas,conexion,1,3
if not(rstablas.eof) then
        Do While not(rstablas.eof)
                nombretabla = rstablas("OBJECT_NAME")
                nombretabla_asterisco = nombretabla
                If (InStr(session("TABLAS"),nombretabla&" ") <> 0) then
                        nombretabla_asterisco = nombretabla & " (*)"
                end if
                idtabla = rstablas("OBJECT_ID")
                if (CLng(idtabla)=CLng("0" & session("l_tablas"&t))) then
                        nombretablasel = nombretabla
                end if
                response.write "tiptab[""ttab" & t & """][tiptab[""ttab" & t & """].length] = new opt("""" &
        idtabla & """,""" & nombretabla_asterisco & """); " & Vbcr
                rstablas.movenext
        Loop
end if
rstablas.close
…
%>
</SCRIPT>
```

Obtener las etiquetas de los atributos difusos:

```
<%
...
set rsetiquetas=server.createobject(“ADODB.recordset”)
‘sqletiquetas = “Select * From FOL where OBJ# in (“ & session(“l_tablas”&t) & “) Order By OBJ#, COL#, FUZZY_ID”
sqletiquetas = “SELECT FOL.*, FLD.* FROM FOL, FLD WHERE FOL.OBJ# in (“ & session(“l_tablas”&t) & “) AND FOL.OBJ#=FLD.OBJ# AND FOL.COL#=FLD.COL# AND FOL.FUZZY_ID=FLD.FUZZY_ID Order by FOL.OBJ#, FOL.COL#, FOL.FUZZY_ID”
rsetiquetas.open sqletiquetas,conexion,1,3
tabla_actual = 0
campo_actual = 0
if not(rsetiquetas.eof) then
  Do While not(rsetiquetas.eof)
    idtabla = CLng(rsetiquetas(“OBJ#”))
    idcampo = CLng(rsetiquetas(“COL#”))
    idetiqueta = rsetiquetas(“FUZZY_NAME”)
        etiqueta = rsetiquetas(“FUZZY_NAME”) & “ = $[” & rsetiquetas(“ALFA”) & “,” & rsetiquetas(“BETA”) & “,” & rsetiquetas(“GAMMA”) & “,” & rsetiquetas(“DELTA”) & “]”
        if (idtabla<>tabla_actual OR idcampo<>campo_actual) then
            response.write “etiq[””t”&idtabla&”c”&idcampo&”””] = new Array(); “ & Vbcr
        end if
        response.write “etiq[””t”&idtabla&”c”&idcampo&”””][etiq[””t”&idtabla&”c”& idcampo&”””].length] = new opt(””” & idetiqueta & “””,””” & etiqueta & “”””); “ & Vbcr
        rsetiquetas.movenext
    tabla_actual = idtabla
    campo_actual = idcampo
  Loop
end if
rsetiquetas.close
set rsetiquetas = Nothing
…
%>
```

Por último, creamos los arrays de los campos de las tablas seleccionadas, de los tipos de los atributos, de los comentarios de los atributos, y de los valores MARGEN, MUCH.

```
<%
...
‘ ********** Recuperamos los campos de las tablas seleccionadas ******************
idtabla = session(“l_tablas”&t)
sqlcampos = “SELECT ALL_TAB_COLUMNS.TABLE_NAME,
ALL_TAB_COLUMNS.COLUMN_NAME, ALL_TAB_COLUMNS.COLUMN_ID,
ALL_TAB_COLUMNS.DATA_TYPE,ALL_COL_COMMENTS.COMMENTS ,FCL.OBJ#, F_TYPE,
FCL.COL#, FAM.MARGEN, FAM.MUCH FROM ALL_TAB_COLUMNS, ALL_COL_COMMENTS,
FCL, FAM where ALL_TAB_COLUMNS.TABLE_NAME=’” & nombretablasel & “’ AND FCL.OBJ# (+) =
“ & idtabla & “ AND FCL.COL# (+) = ALL_TAB_COLUMNS.column_id AND
ALL_TAB_COLUMNS.TABLE_NAME = ALL_COL_COMMENTS.TABLE_NAME AND
ALL_TAB_COLUMNS.COLUMN_NAME = ALL_COL_COMMENTS.COLUMN_NAME AND
FCL.OBJ#=FAM.OBJ# (+) AND FCL.COL#=FAM.COL# (+) ORDER BY
ALL_TAB_COLUMNS.COLUMN_ID”
set rscampos=server.createobject(“ADODB.recordset”)
rscampos.open sqlcampos,conexion,1,3
response.write “tab[””t”&idtabla&”””] = new Array(); “ & Vbcr
response.write “colcom[””t”&idtabla&”””] = new Array(); “ & Vbcr
response.write “margen[””t”&idtabla&”””] = new Array(); “ & Vbcr
response.write “much[””t”&idtabla&”””] = new Array(); “ & Vbcr
if not(rscampos.eof) then
  Do While not(rscampos.eof)
  if (rscampos(“F_TYPE”)=”2” OR rscampos(“F_TYPE”)=”3”) then
    campo = rscampos(“COLUMN_NAME”)
    campoformateado = Left(campo,Len(campo)-1)
    response.write “tab[””t”&idtabla&”””][tab[””t”&idtabla&”””].length] = new opt(“”” &
rscampos(“COLUMN_ID”) & “””,””” & campoformateado & “”””); “ & Vbcr
  else
    response.write “tab[””t”&idtabla&”””][tab[””t”&idtabla&”””].length] = new opt(“”” &
rscampos(“COLUMN_ID”) & “””,””” & rscampos(“COLUMN_NAME”) & “”””); “ & Vbcr
  end if
  ‘**************** Creamos el ARRAY DE COMENTARIOS *********************
  comentario = Replace(rscampos(“COMMENTS”), Chr(10), “ “)
  response.write “colcom[””t”&idtabla&”””][colcom[””t”&idtabla&”””].length] = new opt(“”” &
rscampos(“COLUMN_ID”) & “””,””” & comentario & “”””); “ & Vbcr
  ‘****************** Creamos el ARRAY DE VALORES “MARGEN” ******************
  valormargen = rscampos(“MARGEN”)
  response.write “margen[””t”&idtabla&”””][margen[””t”&idtabla&”””].length] = new opt(“”” &
rscampos(“COLUMN_ID”) & “””,””” & valormargen & “”””); “ & Vbcr
  ‘******************** Creamos el ARRAY DE VALORES “MUCH” ******************
  valormuch = rscampos(“MUCH”)
  response.write “much[””t”&idtabla&”””][much[””t”&idtabla&”””].length] = new opt(“”” &
rscampos(“COLUMN_ID”) & “””,””” & valormuch & “”””); “ & Vbcr
  ‘**************** Creamos el ARRAY DE TIPOS DE LOS ATRIBUTOS ***************
  response.write “ftypes[””t”&idtabla&”c”&rscampos(“COLUMN_ID”)&”””] = new Array(); “ & Vbcr
```

```
 if (rscampos("F_TYPE")="1" OR rscampos("F_TYPE")="2" OR rscampos("F_TYPE")="3") then
   response.write "ftypes[""t"&idtabla&"c"&rscampos("COLUMN_ID")&"""][ftypes[""t"&
idtabla&"c"&rscampos("COLUMN_ID")&"""].length] = new opt("" & CLng("0" & rscampos("F_TYPE")) &
"","" & rscampos("COLUMN_NAME") & ""); " & Vbcr
 else
   select case rscampos("DATA_TYPE")
      case "CHAR","VARCHAR","VARCHAR2","NVARCHAR": response.write
"ftypes[""t"&idtabla&"c"&rscampos("COLUMN_ID")&"""][ftypes[""t"&idtabla&"c"&rscampos("COLUMN_I
D")&"""].length] = new opt(""4"","" & rscampos("COLUMN_NAME") & ""); " & Vbcr
      case "NUMBER","LONG","INTEGER","SHORT": response.write
"ftypes[""t"&idtabla&"c"&rscampos("COLUMN_ID")&"""][ftypes[""t"&idtabla&"c"&rscampos("COLUMN_I
D")&"""].length] = new opt(""5"","" & rscampos("COLUMN_NAME") & ""); " & Vbcr
     case "DATE": response.write
"ftypes[""t"&idtabla&"c"&rscampos("COLUMN_ID")&"""][ftypes[""t"&idtabla&
"c"&rscampos("COLUMN_ID")&"""].length] = new opt(""7"","" & rscampos("COLUMN_NAME") & ""); "
& Vbcr
      case else: response.write
"ftypes[""t"&idtabla&"c"&rscampos("COLUMN_ID")&"""][ftypes[""t"&idtabla&"c"&rscampos("COLUMN_I
D")&"""].length] = new opt(""0"","" & rscampos("COLUMN_NAME") & ""); " & Vbcr
   end select
 end if
…%>
```

- **Consulta Visual: Ejecutar** (consultavisual_finalizar.asp)

En este código construimos la sentencia FSQL resultante del Asistente de Consulta Visual, y entonces llamamos a la función *convierteFSQL2SQL* definida en el archivo "*FSQL2SQL.asp*" y si todo ha ido bien nos devuelve la sentencia SQL resultado de la traducción. Por último ejecutamos esta sentencia SQL para obtener los resultados.

```
<!--#include file="include/settings.asp"-->
<%
On Error Resume Next
time_ini = time

'Abrimos la conexion
Set conexion = Server.CreateObject("ADODB.Connection")
conexion.Open cnxVisualFSQL
set rsconsulta=server.createobject("ADODB.recordset")
rsconsulta.CursorLocation = 3
```

```
‘Creamos la sentencia SQL
...
%>

<!--#include file=”include/FSQL2SQL.asp”-->
<%
consultafsql = sqlconsulta
sqlconsulta = convierteFSQL2SQL(consultafsql,cnxVisualFSQL)
...
%>
rsconsulta.open sqlconsulta,conexion,1,2
If not (rsconsulta.EOF) then
        num_campos = rsconsulta.Fields.Count
i=0

‘Mostramos los registros encontrados
...
%>
```

A continuación vamos a mostrar el código de “*FSQL2SQL.asp*”, donde se define una de las funciones más importantes de la aplicación:

```
<%
function convierteFSQL2SQL(sentenciaFSQL,cnxVisualFSQL)
set rssentenciafsql=server.createobject(“ADODB.recordset”)
rssentenciafsql.CursorLocation = 3
‘Abrimos la conexion
Set conexionfsql = Server.CreateObject(“ADODB.Connection”)
With conexionfsql
   .Open cnxVisualFSQL
End With

QSQL = “SYS.PFSQL2SQL”  ‘llamamos al procedimiento almacenado en SYS.PFSQL2SQL
Set FSQLCom = Server.CreateObject(“ADODB.Command”)
With FSQLCom
   .ActiveConnection = conexionfsql
   .CommandText = QSQL
   .CommandType = 4 ‘adCmdStoredProc
```

```
    .Parameters.Append .CreateParameter(, 201, 1, 1000)
    .Parameters.Append .CreateParameter(, 5, 2)
End With
‘Establecemos el parámetro 0(el primero) del comando FSQLCom
FSQLCom(0) = sentenciaFSQL
FSQLCom.Execute
Num_Errores = FSQLCom(1)

If Num_Errores <> 0 Then
    convierteFSQL2SQL = Num_errores
Else
    mselect = “SELECT ATRIBUTO FROM FSQL_QUERY WHERE ATRIBUTO IS NOT NULL ORDER
BY INDICE”
    rssentenciafsql.open mselect,conexionfsql,1,2
    ‘Cogemos primer valor del Recorset que nos devuelve.
    ConsultaSQL = rssentenciafsql.Fields(“atributo”).Value
    ‘Nos movemos al siguiente registro del recordset para recoger otra parte de la consulta SQL.
    rssentenciafsql.MoveNext
    ‘ No meter espacios ni antes ni después de determinados caracteres
    anterior_sin_esp = False
    Do While rssentenciafsql.EOF <> True
        valor = rssentenciafsql.Fields(“atributo”).Value
        ch1 = Left(valor, 1)
        If anterior_sin_esp Or _
        (Len(valor) = 1 And _
        (ch1 = “,” Or ch1 = “.” Or ch1 = “=” Or _
        ch1 = “;” Or ch1 = “-“ Or ch1 = “+” Or _
        ch1 = “(“ Or ch1 = “)” Or _
        ch1 = “/” Or ch1 = “*” Or _
        ch1 = “>” Or ch1 = “<”)) Then
            ConsultaSQL = ConsultaSQL & valor
            anterior_sin_esp = Not anterior_sin_esp
        Else
            ConsultaSQL = ConsultaSQL & “ “ & valor
        End If
        rssentenciafsql.MoveNext
    Loop
End If
```

```
'Limpiamos la memoria.
rssentenciafsql.close
set rssentenciafsql = Nothing
conexionfsql.Close
Set conexionfsql = Nothing
Set FSQLCom = Nothing

convierteFSQL2SQL = ConsultaSQL
end function
...
%>
```

Esta es la función que nos va a convertir una sentencia FSQL a SQL estándar. Como parámetros vamos a pasar la consulta FSQL y la cadena de conexión, una vez establecida la conexión vamos a crear un objeto ASP de tipo *Command* para llamar al procedimiento almacenado *PFSQL2SQL* del usuario SYS del SGBD. Hay que aclarar que, el objeto *Command* de ADO no se puede comunicar con funciones de Oracle, entonces para acceder a la función *FSQL2SQL* del Servidor FSQL hemos tenido que crear un procedimiento almacenado intermedio llamado PFSQL2SQL, el proceso para crear este procedimiento se puede ver en el capítulo siguiente (Instalación del Sistema).

Para continuar con la descripción del código, lo siguiente es verificar si no se han producido errores, y ese caso construimos la consulta SQL traducida mediante la tabla *FSQL_QUERY* del servidor.

Capítulo 6

Instalación del Sistema

Vamos a explicar los pasos necesarios para instalar el sistema completo, y que hemos dividido en 3 apartados. Es importante seguir el orden establecido puesto que no se trata únicamente de instalar una aplicación cliente, sino de un sistema completo. Vamos a detallar todo el proceso que se divide en: Instalación/configuración de Oracle, instalación del Servidor FSQL y por último instalación/configuración del Servidor WEB IIS (Internet Information Server).

6.1. Instalación de ORACLE

De Oracle Release 3 (8.1.7) for Microsoft Windows NT/2000/XP tenemos varias versiones para instalar, "*Enterprise Edition*", "*Standard Edition*", "*Personal Edition*" y "*Lite Edition*".

- *Enterprise Edition*: Es la más completa y profesional, incluye múltiples herramientas de ayuda y gestión.
- *Standard Edition*: Es la siguiente versión en cuanto a componentes y módulos incluidos.
- *Personal Edition*: Es una versión más pequeña pero totalmente funcional del Servidor de Base de Datos. Será la que instalaremos para el desarrollo del proyecto.
- *Lite Edition*: Es una miniversión de Oracle utilizada para entornos de prueba y en equipos portátiles.

Hay que indicar que todas estas versiones incluyen el Cliente de Oracle, que será necesario instalar en el Equipo que actúe como servidor Web, puesto que este actuará a su vez como cliente del Servidor de BD Oracle.

En el caso que nos ocupa se ha optado por instalar ambos Servidores (Web y BD-Oracle) en una misma máquina, puesto que el sistema del presente proyecto no va a tener una carga masiva de accesos al tratarse de una versión de desarrollo y no de explotación.

6.1.1. Oracle Release 3 (8.1.7) for Microsoft Windows NT/2000/XP

Para su instalación hay que seguir los siguientes pasos:

1) En primer lugar introducimos el CD de instalación y seleccionamos "Install/Deinstall Products".

2) Nos aparecerá una pantalla de Bienvenida en la que tendremos la posibilidad de revisar los productos de Oracle instalados, en nuestro caso y al tratarse de una primera instalación pulsamos en "Siguiente".

3) Ahora debemos seleccionar la UBICACIÓN DE LOS FICHEROS: "Origen" de la instalación y el "Destino" donde se instalará el Servidor Oracle (ver Figura 6.1).

Figura 6.1. Instalación de Oracle: Ubicación de los ficheros.

Como vemos en la Figura 6.1, tenemos seleccionado por defecto el "Origen" de la unidad lectora de CD (E:\) que normalmente no hay que modificar. En el "Destino" también dejamos por defecto lo seleccionado por el instalador, que en este caso hace referencia a una unidad de nuestro disco duro (D:\).

Pulsamos en "Siguiente" para continuar el proceso.

4) Ahora nos aparecerán los PRODUCTOS DISPONIBLES que podemos instalar tal y como vemos en la Figura 6.2. Entonces seleccionamos la primera opción "Oracle 8i Personal Edition 8.1.7.0.0".

Figura 6.2. Instalación de Oracle: Productos Disponibles.

5) Tenemos nuevamente 3 opciones de TIPOS DE INSTALACION, seleccionamos la "Típica" y pulsamos en "Siguiente" (ver Figura 6.3).

Con este tipo de instalación nos será suficiente para la Base de Datos que necesitamos: Instala una base de datos inicial preconfigurada, las opciones del producto, las herramientas de administración, los servicios de red, las utilidades y el software básico del cliente para un servidor de bases de datos Oracle.

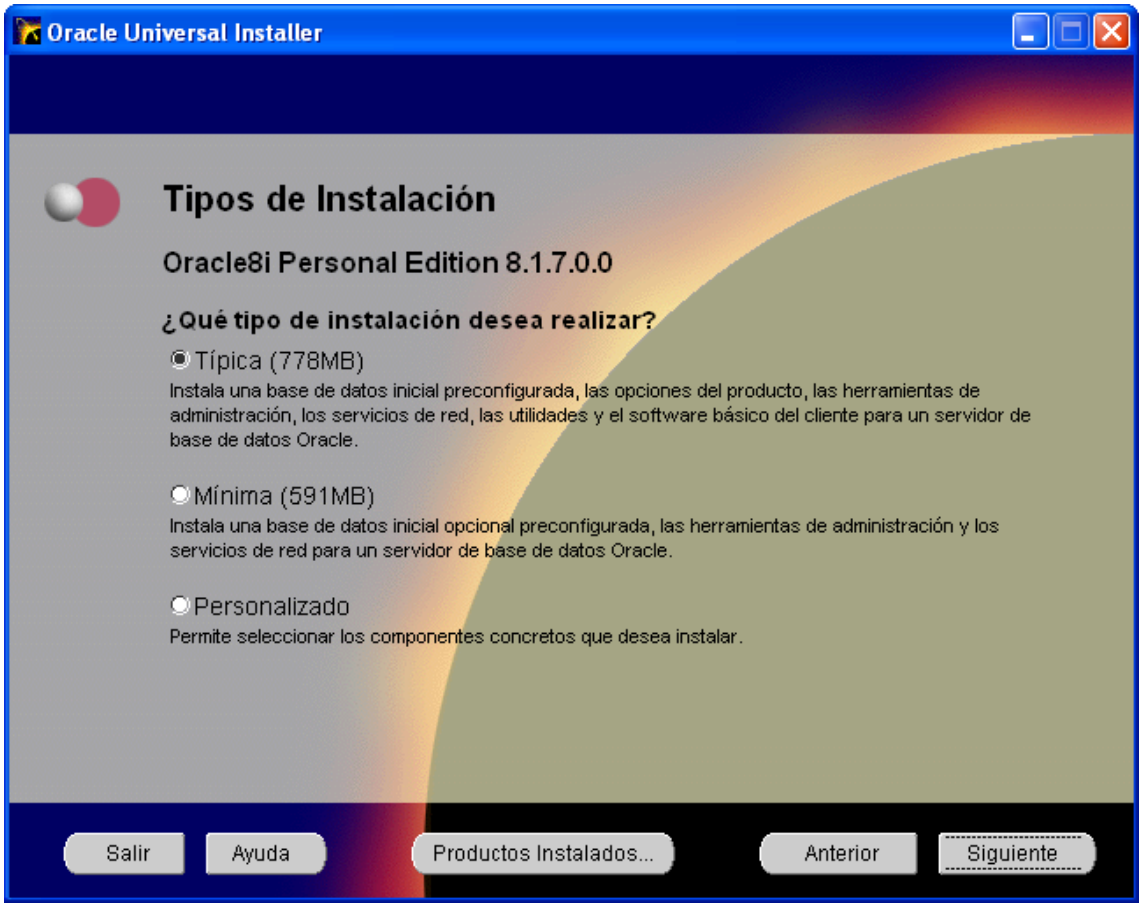


Figura 6.3. Instalación de Oracle: Tipos de Instalación.

6) IDENTIFICACIÓN DE BASE DE DATOS (Figura 6.4). En esta pantalla indicamos el "Nombre de la Base de Datos Global", por ejemplo ORAFSQL (de ORAcle y de FSQL), y como SID (identificador único de Base de Datos) p.ej. ORCL. Pulsamos en "Siguiente".

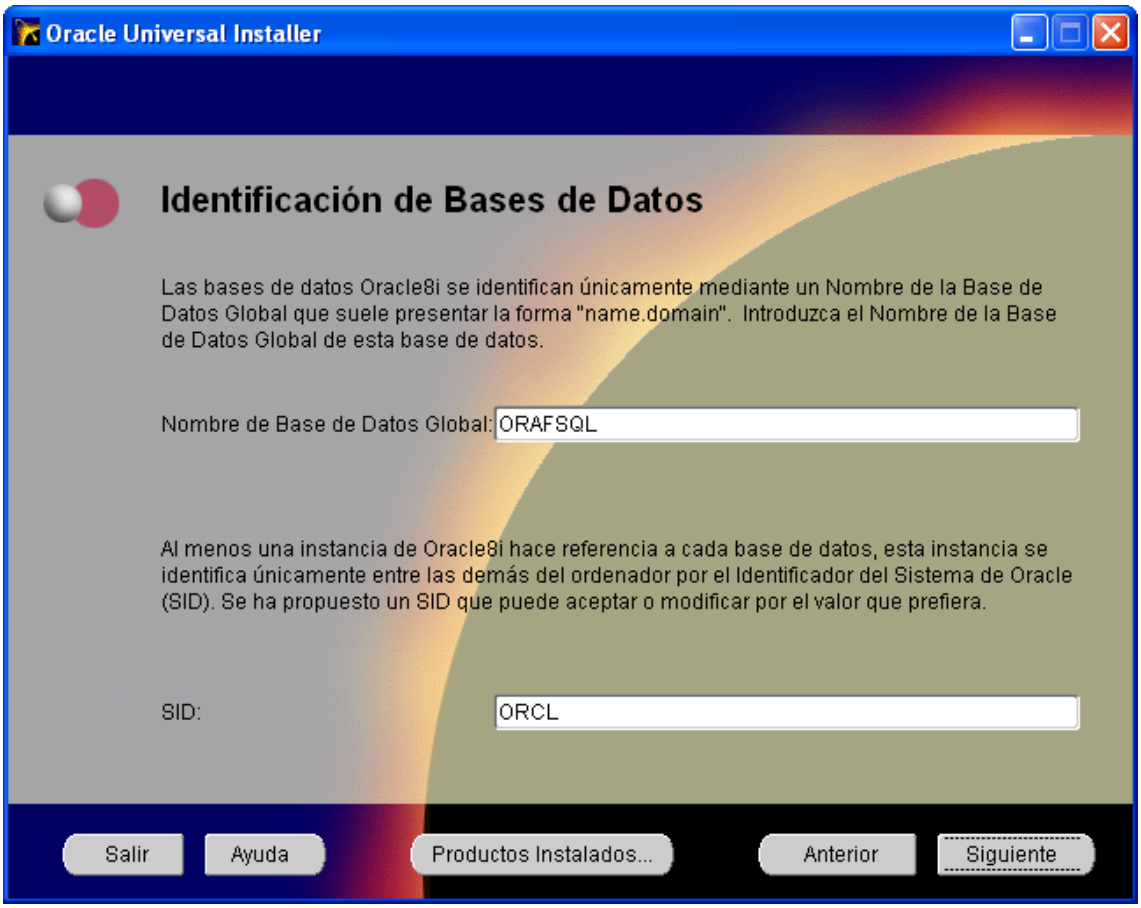


Figura 6.4. Instalación de Oracle: Identificación de Bases de Datos.

7) Por último, nos aparece una ventana “RESUMEN” (Figura 6.5) de los productos que se instalarán. Para confirmar pulsaremos en “Instalar”.

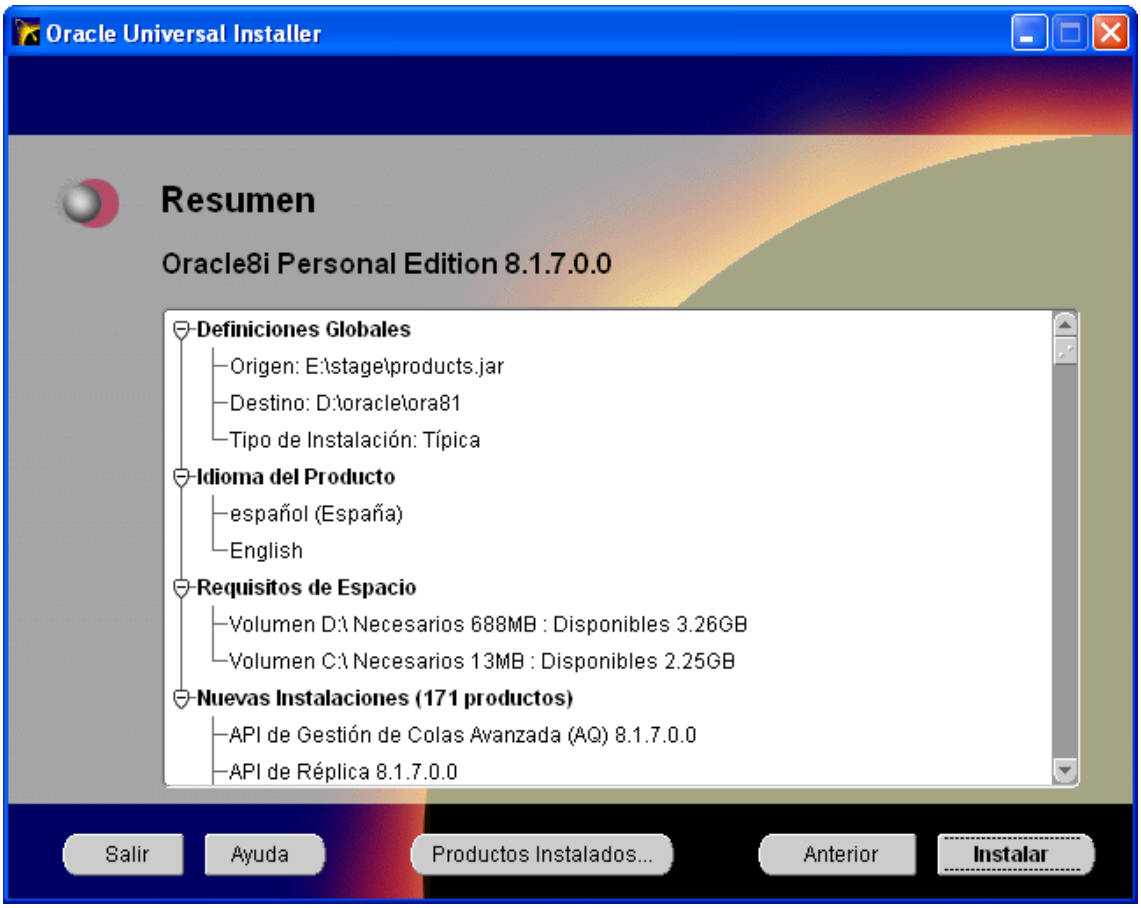


Figura 6.5. Instalación de Oracle: Resumen.

En este momento comenzará a instalación, que puede tardar varios minutos (Figura 6.6).

Una vez finalizada la instalación se ejecutan automáticamente unos asistentes de configuración del LISTENER y de la instancia de la BD.

Figura 6.6. Instalación de Oracle: Progreso de Creación de la Base de Datos.

Si todo ha ido bien, nos aparecerá un esperado mensaje de "Fin de la Instalación". Para finalizar pulsaremos en "Salir".

6.1.2. Configuración del "Net8Assistant"

Ahora tenemos que configurar el Servicio del Cliente Oracle, para que nos podamos conectar al Servidor de Base de Datos Oracle.

Tenemos dos opciones: Bien configurando manualmente el fichero de servicios "D:\oracle\ora81\network\admin\tnsnames.ora" ó bien de forma visual mediante el entorno Net8Assistant incluido en la instalación de Oracle. Nos decantamos por está última opción.

Vamos a: Inicio -> Programas –> Oracle-OraHome81 -> Network Administration -> Net8 Assistant

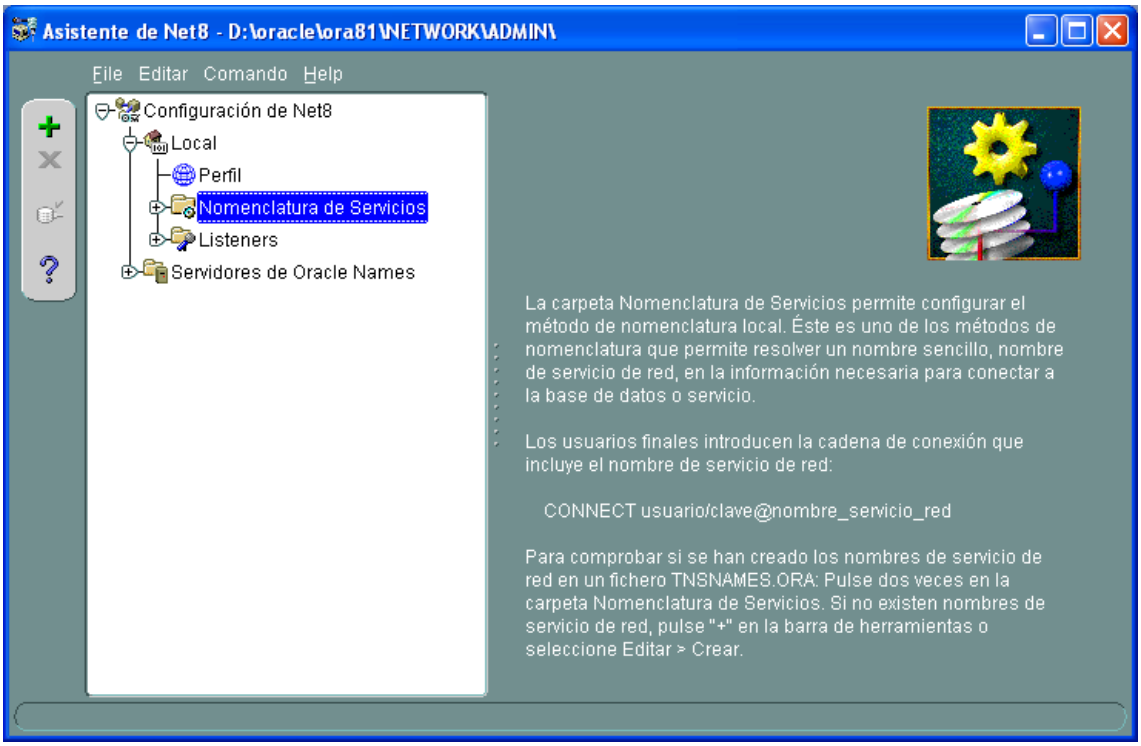


Figura 6.7. Asistente de Net8 (1).

Para añadir una nuevo Servicio, debemos seleccionar “Nomenclatura de Servicios” y a continuación pulsar el símbolo “+” (de color verde), ver Figura 6.7.

En nuestro caso, al tener instalado Oracle en la misma máquina que el Servidor Web, ponemos la dirección IP local (127.0.0.1), en caso de que tuviéramos el Servidor Oracle instalado en otra máquina pondríamos la IP de ese Servidor.

Introducimos los siguientes valores (Figura 6.8):

- Nombre de Servicio de Red: “FSQLSVC” (podemos dar un nombre cualquiera)
- Protocolo “TCP/IP (Protocolo Internet)”
- Nombre del Host: 127.0.0.1
- El puerto dejamos por defecto el “1521”.
- Nombre del Servicio (Oracle 8): “ORAFSQL” (nombre de la instancia Oracle, asignada durante la instalación, ver Figura 6.4).

Por último, y una vez finalizado el proceso tendremos creado el nombre de servicio "fsqlsvc" que hace referencia a la instancia de Oracle "ORAFSQL" de la máquina "127.0.0.1" a través del puerto "1521".

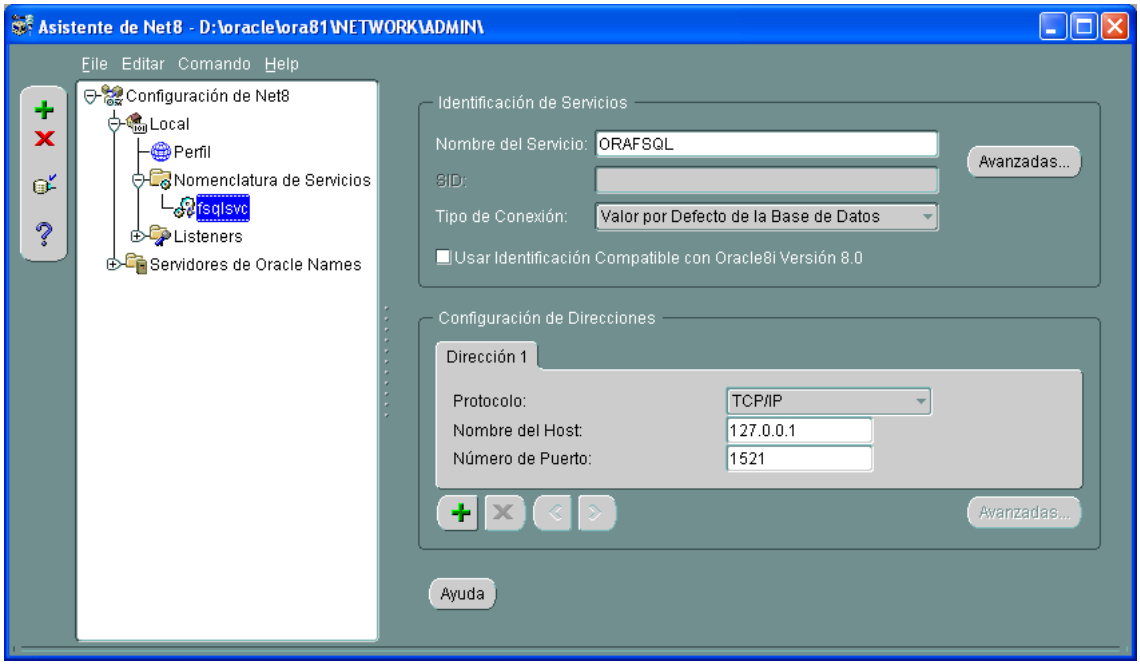


Figura 6.8. Asistente de Net8 (2).

Antes de salir del asistente grabamos los cambios realizados: File -> Guardar Configuración de Red.

6.2. Instalación del Servidor FSQL

Para el funcionamiento de nuestra aplicación necesitaremos tener instalado el Servidor de Consultas Difusas FSQL, que ha sido desarrollado bajo entorno Oracle.

Para proceder a la instalación vamos a ejecutar una serie de *Scripts* que crearán la estructura de tablas y procedimientos almacenados que conforman el Servidor FSQL para la gestión interna del Sistema Difuso.

6.2.1. Preparación del Entorno

En primer lugar, para tener un entorno más limpio, vamos a borrar las tablas de ejemplo que instala Oracle: ACCOUNT, BONUS, DEPT, EMP, RECEIPT y SALGRADE. Esto no es necesario, pero en nuestro caso hemos creído conveniente hacerlo para tener nuestro sistema inicialmente sin tablas de usuario.

Para ellos, nos vamos al SQL-Plus (programa cliente que trae Oracle por defecto), y entramos como (ver Figura 6.9):

Usuario: scott
Password: tiger
Cadena de conexión: fsqlsvc

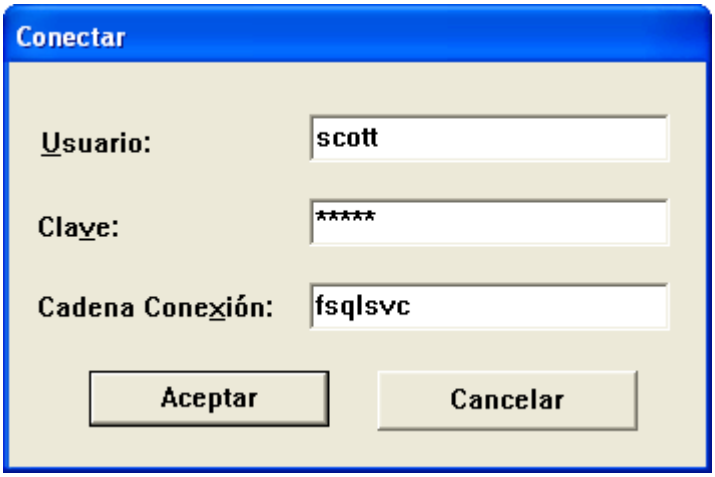


Figura 6.9. Ventana de conexión a SQL-Plus.

El usuario scott lo crea por defecto Oracle en la instalación y la cadena de conexión es el servicio que dimos de alta en el Net8Assistant durante la configuración de Oracle.

Para borrar estas tablas ponemos (en este caso vamos a borrar la tabla de ejemplo *account*):

```
Drop table account;
```

6.2.2. Instalación de FIRST, FSQL y Base de Datos de ejemplo

Vamos a conectarnos en el SQL-Plus como usuario "SYS" ya que tenemos que instalar el sistema como DBA "Data Base Administrator" (SYS es un usuario DBA que se crea por defecto en la instalación de Oracle, su contraseña por defecto es "change_on_install"). Opcionalmente podemos instalar una Base de Datos de ejemplo para empezar a probar el sistema, esta instalación la hacemos conectándonos como un Usuario normal de Oracle, p.ej. "scott". Vamos a detallar este proceso:

Si ya estamos conectados podemos desconectarnos y volvernos a conectar de nuevo:

```
Disconnect
Connect

Usuario: sys
Password: change_on_install
Cadena de conexión:  fsqlsvc
```

En la Tabla 6.1 vemos los archivos que vamos a necesitar.

Archivo	Usuario	Descripción
FIRSTins.sql	SYS	Script de instalación de FIRST (tablas, permisos...)
FIRSTdes.sql	SYS	Script de desinstalación de FIRST
SFSQLins.sql	SYS	Script de instalación del Servidor FSQL FSQL_PKG.sql: Paquete FSQL_PKG, servidor FSQL y funciones útiles para el Cliente FSQL FSQL_SEM.sql: Paquete FSQL_SEMANTIC, analizador semántico del Servidor FSQL_AUX.sql: Paquete FSQL_AUX, funciones auxiliares usadas por los demás paquetes FSQL_F.sql: Paquete FSQL_FUNCTIONS: Funciones de comparación difusa FSQL_F2.sql: Paquete FSQL_FUNCTIONS2: Funciones de comparación difusa
SFSQLdes.sql	SYS	Script de desinstalación del Servidor FSQL
DBVisualFSQLins.sql	Scott	Script de instalación de un ejemplo de Base de Datos Difusa
DBVisualFSQLdes.sql	Scott	Script de desinstalación de un ejemplo de Base de Datos Difusa

Tabla 6.1. Archivos de instalación de FIRST, FSQL.

Ahora ejecutamos los archivos con los *Scripts* de instalación en el siguiente orden, teniendo especial cuidado con el usuario que instala cada archivo (tal y como podemos ver en la Tabla 6.1):

- FIRSTins.sql : Instalación de FIRST (tablas, permisos, etc...)
- SFSQLins.sql : Instalación del Servidor FSQL
- DBVisualFSQLins.sql: Instala nuestra base de Datos Difusa de ejemplo

Los *scripts* se pueden ejecutar en “SQL-Plus” poniendo el símbolo '@' delante del nombre de fichero. Por ejemplo:

```
SQL> @FIRSTins
```

NOTAS:

Se puede incluir o establecer el PATH por defecto, seleccionándolo en la opción "Open" del menú "File" y pulsando el botón "Cancel" una vez seleccionado el directorio deseado.

Antes de instalar algo, lo desinstala previamente, por lo que en las instalaciones puede generarse algún error, al intentar borrar una tabla, paquete... que no exista. Así, se pueden efectuar re-instalaciones sin desinstalar antes.

Aunque improbable, es posible que algún paquete se instale mal. Por eso es bueno comprobar los errores que salen a lo largo de la instalación para asegurarse que no da más errores que los de 'Imposible borrar algo que no existe' (ORA-00942, ORA-04043...).

Para verificar que los objetos se han creado correctamente podemos mirar que el campo STATUS de la tabla DBA_OBJECTS valga 'VALID'. En caso negativo lo mejor es reinstalar todo o sólo la parte que dió errores. Si siguen produciéndose errores es que tiene una copia mala de este programa o que falla el SGBD.

Para asegurarnos de que no se producen más errores que los indicados arriba, se puede instalar cada parte 2 veces, de forma que en la segunda se eliminan los errores del tipo 'Imposible borrar algo que no existe' (ORA-00942, ORA-04043...).

6.2.3. Desinstalación

También tenemos unos scripts de desinstalación del sistema, que hay que ejecutarlos en el siguiente orden:

- SFSQLdes.sql : Desinstalación del servidor FSQL
- DBVisualFSQLdes.sql : Desinstala nuestra Base de Datos Difusa de ejemplo
- FIRSTdes.sql : Desinstalación del FIRST

6.2.4. PFSQL2SQL

PFSQL2SQL (Procedure FSQL2SQL) es un "procedimiento almacenado" que tenemos que crear para poder acceder mediante ADO a la "función" *FSQL2SQL* del Paquete FSQL_PKG que se instala con el Servidor FSQL. Esto está motivado porque desde ADO no se ha podido acceder a las "funciones" de Oracle y entonces hemos tenido que construir un "procedimiento almacenado" intermedio.

Para crear este procedimiento, tenemos que ejecutar el Script "PFSQL2SQL.sql" como usuario "SYS". Si nos fijamos en el código fuente vemos que tenemos que darle permiso de ejecución a Public para que todos los usuarios puedan ejecutar este procedimiento.

```
-- Fichero PFSQL2SQL.sql
-----------------------------------------------------------------
-- CREACIÓN del Procedimiento Almacenado
-- PFSQL2SQL.
-----------------------------------------------------------------
CREATE OR REPLACE PROCEDURE PFSQL2SQL ( consulta IN VARCHAR, errores OUT
NUMBER)
IS
BEGIN
        errores := FSQL_PKG.FSQL2SQL(consulta);
END PFSQL2SQL;

GRANT EXECUTE ON PFSQL2SQL TO PUBLIC;
--Ejecutar como SYS.
```

6.2.5. DBVisualFSQLins.sql

Este es el fichero de instalación de la Base de Datos Difusa de ejemplo.

-- Fichero DBVisualFSQLins.sql

```
-------------------------------------------------------------------------------------------
-- INSTALACIÓN de una Base de Datos Difusa (FDB, Fuzzy Database)
-- CREACIÓN de las TABLAS inserción de algunas tuplas en ellas,
-- así como creación de elementos en la FMB de esta Bd
-- (etiquetas...).
-- Para la instalación no hace falta desinstalarla previamente.
-- Debe ejecutarse por el usuario que será propietario de las
-- tablas de la Bd (no necesariamente un DBA).
-------------------------------------------------------------------------------------------
-- Para que salgan los mensajes (de put_line) al instalar desde FSQL*Plus:
set serveroutput on
-- Para que salgan (o no) las ordenes a ejecutar: set echo on
set echo off
-- Para que no salgan los mensajes cuando todo va bien (los errores si salen):
set feedback off

exec dbms_output.put_line('>>>>');
exec dbms_output.put_line('>>>> ****************************************** <<<<');
exec dbms_output.put_line('>>>> **** INSTALACIÓN de una Base de Datos **** <<<<');
exec dbms_output.put_line('>>>> **** Difusa de ejemplo.              **** <<<<');
exec dbms_output.put_line('>>>> ****************************************** <<<<');
exec dbms_output.put_line('>>>>');
exec dbms_output.put_line('>>>> NOTAS:');
exec dbms_output.put_line('>>>>   - Es requisito indispensable que esté instalado FIRST (tablas de la
FMB...).');
exec dbms_output.put_line('>>>>   - No importa si se producen errores al intentar borrar');
exec dbms_output.put_line('>>>>     TABLAS que no existan (ORA-00942).');
exec dbms_output.put_line('>>>>   - Cualquier otro error podrá impedir la correcta consulta a esta
Bd.');
exec dbms_output.put_line('>>>>     En ese caso revise el código del error y el lugar donde ha
ocurrido.');
exec dbms_output.put_line('>>>>');
```

-- TABLA EMPLEADOS

```
exec dbms_output.put_line('>>>> Borrando y Creando tabla EMPLEADOS...');
DROP   TABLE EMPLEADOS CASCADE CONSTRAINTS;
CREATE TABLE EMPLEADOS(
  EMP# NUMBER(4) NOT NULL,
  NOMBRE CHAR(20) NOT NULL,
  APELLIDOS CHAR(40) NOT NULL,
  SEXO CHAR(1) NOT NULL,
  EDADT NUMBER(1) DEFAULT 0 NOT NULL
     CHECK (EDADT BETWEEN 0 AND 7)
     CONSTRAINT NULL_INVALIDO_EDAD      CHECK (EDADT<>2)
     CONSTRAINT UNDEFINED_INVALIDO_EDAD CHECK (EDADT<>1),
  EDAD1 NUMBER(3),
  EDAD2 NUMBER(3),
  EDAD3 NUMBER(3),
  EDAD4 NUMBER(3),
  TELEFONO CHAR(12) NOT NULL,
  PRIMARY KEY (EMP#));
```

-- TABLA APTITUDES

```
exec dbms_output.put_line('>>>> Borrando y Creando tabla APTITUDES...');
DROP   TABLE APTITUDES CASCADE CONSTRAINTS;
CREATE TABLE APTITUDES(
  EMP# NUMBER(4) NOT NULL,
  ESTUDIOST NUMBER(1) DEFAULT 0 NOT NULL
     CHECK (ESTUDIOST BETWEEN 0 AND 7), -- LIMITAMOS A 8(0-7), LOS TIPOS DE TRIBUTOS
TIPO2
  ESTUDIOS1 NUMBER(2),
  ESTUDIOS2 NUMBER(2),
  ESTUDIOS3 NUMBER(2),
  ESTUDIOS4 NUMBER(2),
  PROFESIONT NUMBER(1) DEFAULT 0 NOT NULL
     CHECK (PROFESIONT BETWEEN 0 AND 4),       -- LIMITAMOS A 5(0-4), LOS TIPOS DE
TRIBUTOS TIPO3
  PROFESIONP1 NUMBER(3,2),
  PROFESION1 NUMBER(3) NOT NULL, -- FUZZY_OBJECT_LIST.FUZZY_ID%TYPE
```

```
  EXPERIENCIAT NUMBER(1) DEFAULT 0 NOT NULL
     CHECK (EXPERIENCIAT BETWEEN 0 AND 7), -- LIMITAMOS A 8(0-7), LOS TIPOS DE
TRIBUTOS TIPO2
  EXPERIENCIA1 NUMBER(2),
  EXPERIENCIA2 NUMBER(2),
  EXPERIENCIA3 NUMBER(2),
  EXPERIENCIA4 NUMBER(2),
  PRIMARY KEY (EMP#),
  FOREIGN KEY (EMP#) REFERENCES EMPLEADOS ON DELETE CASCADE);
```

-- TABLA DEPARTAMENTOS

```
exec dbms_output.put_line('>>>> Borrando y Creando tabla DEPARTAMENTOS...');
DROP   TABLE DEPARTAMENTOS CASCADE CONSTRAINTS;
CREATE TABLE DEPARTAMENTOS(
  DPTO#   NUMBER(4)  NOT NULL,
  NOMBRE  CHAR(20) NOT NULL,
  BENEFICIOST NUMBER(1) DEFAULT 0 NOT NULL
     CHECK (BENEFICIOST BETWEEN 0 AND 7), -- LIMITAMOS A 8(0-7), LOS TIPOS DE
TRIBUTOS TIPO2
  BENEFICIOS1 NUMBER(8),
  BENEFICIOS2 NUMBER(8),
  BENEFICIOS3 NUMBER(8),
  BENEFICIOS4 NUMBER(8),
  PRIMARY KEY (DPTO#));
```

-- TABLA PUESTOS

```
exec dbms_output.put_line('>>>> Borrando y Creando tabla PUESTOS...');
DROP   TABLE PUESTOS CASCADE CONSTRAINTS;
CREATE TABLE PUESTOS(
  PUESTO# NUMBER(4) NOT NULL,
  NOMBRE CHAR(20) NOT NULL,
  ESTUDIOST NUMBER(1) DEFAULT 0 NOT NULL
     CHECK (ESTUDIOST BETWEEN 0 AND 7), -- LIMITAMOS A 8(0-7), LOS TIPOS DE TRIBUTOS
TIPO2
  ESTUDIOS1 NUMBER(2),
  ESTUDIOS2 NUMBER(2),
```

```
  ESTUDIOS3 NUMBER(2),
  ESTUDIOS4 NUMBER(2),
  PROFESIONT NUMBER(1) NOT NULL,
     CHECK (PROFESIONT BETWEEN 0 AND 4),
  PROFESIONP1 NUMBER(3,2),
  PROFESION1 NUMBER(3) NOT NULL,
  EXPERIENCIAT NUMBER(1) NOT NULL,
     CHECK (EXPERIENCIAT BETWEEN 0 AND 7),
  EXPERIENCIA1 NUMBER(2),
  EXPERIENCIA2 NUMBER(2),
  EXPERIENCIA3 NUMBER(2),
  EXPERIENCIA4 NUMBER(2),
  PRIMARY KEY (PUESTO#));
```

-- TABLA EMPLEOS

```
exec dbms_output.put_line('>>>> Borrando y Creando tabla EMPLEOS...');
DROP   TABLE EMPLEOS;
CREATE TABLE EMPLEOS(
  EMP# NUMBER(4) NOT NULL,
  DPTO# NUMBER(4) NOT NULL,
  PUESTO# NUMBER(4) NOT NULL,
  SUELDO NUMBER(7) NOT NULL,
  COMISIONT NUMBER(1) NOT NULL,
     CHECK (COMISIONT BETWEEN 0 AND 7),
  COMISION1 NUMBER(7),
  COMISION2 NUMBER(7),
  COMISION3 NUMBER(7),
  COMISION4 NUMBER(7),
  RENDIMIENTOT NUMBER(1) NOT NULL,
     CHECK (RENDIMIENTOT BETWEEN 0 AND 4),
  RENDIMIENTOP1 NUMBER(3,2),
  RENDIMIENTO1 NUMBER(3) NOT NULL,
  PRIMARY KEY (EMP#,DPTO#,PUESTO#),
  FOREIGN KEY (EMP#) REFERENCES EMPLEADOS ON DELETE CASCADE,
  FOREIGN KEY (EMP#) REFERENCES APTITUDES ON DELETE CASCADE,
  FOREIGN KEY (DPTO#) REFERENCES DEPARTAMENTOS ON DELETE CASCADE,
  FOREIGN KEY (PUESTO#) REFERENCES PUESTOS ON DELETE CASCADE);
```

```
DECLARE
  t_APTITUDES           NUMBER;
  t_EMPLEOS             NUMBER;
  t_PUESTOS             NUMBER;
  t_DEPARTAMENTOS NUMBER;
  t_EMPLEADOS           NUMBER;
  c_estudios      NUMBER;
  c_experiencia NUMBER;
  c_profesion     NUMBER;
  c_comision    NUMBER;
  c_rendimiento NUMBER;
  c_sueldo      NUMBER;
  c_estudios1   NUMBER;
  c_experiencia1  NUMBER;
  c_profesion1  NUMBER;
  c_beneficios    NUMBER;
  c_edad        NUMBER;
BEGIN
  dbms_output.put_line('>>>> Insertando valores en la FMB sobre esta Bd Difusa...');


  -- Calcular OBJ# para las tablas con columnas difusas:
  SELECT object_id into t_APTITUDES FROM user_objects WHERE object_name='APTITUDES';
  SELECT object_id into t_EMPLEOS FROM user_objects WHERE object_name='EMPLEOS';
  SELECT object_id into t_PUESTOS FROM user_objects WHERE object_name='PUESTOS';
  SELECT object_id into t_DEPARTAMENTOS       FROM user_objects WHERE
object_name='DEPARTAMENTOS';
  SELECT object_id into t_EMPLEADOS   FROM user_objects WHERE object_name='EMPLEADOS';


  -- Calcular COL# de las columnas difusas:
  SELECT column_id into c_estudios
    FROM user_tab_columns WHERE table_name='APTITUDES' AND column_name='ESTUDIOST';
  SELECT column_id into c_experiencia
    FROM user_tab_columns WHERE table_name='APTITUDES' AND
column_name='EXPERIENCIAT';
  SELECT column_id into c_profesion
    FROM user_tab_columns WHERE table_name='APTITUDES' AND column_name='PROFESIONT';
  SELECT column_id into c_comision
    FROM user_tab_columns WHERE table_name='EMPLEOS' AND column_name='COMISIONT';
  SELECT column_id into c_rendimiento
    FROM user_tab_columns WHERE table_name='EMPLEOS' AND
column_name='RENDIMIENTOT';
```

```
SELECT column_id into c_sueldo
  FROM user_tab_columns WHERE table_name='EMPLEOS' AND column_name='SUELDO';
SELECT column_id into c_estudios1
  FROM user_tab_columns WHERE table_name='PUESTOS' AND column_name='ESTUDIOST';
SELECT column_id into c_experiencia1
  FROM user_tab_columns WHERE table_name='PUESTOS' AND column_name='EXPERIENCIAT';
SELECT column_id into c_profesion1
  FROM user_tab_columns WHERE table_name='PUESTOS' AND column_name='PROFESIONT';
SELECT column_id into c_beneficios
  FROM user_tab_columns WHERE table_name='DEPARTAMENTOS' AND
column_name='BENEFICIOST';
SELECT column_id into c_edad
  FROM user_tab_columns WHERE table_name='EMPLEADOS' AND column_name='EDADT';

-- Atributos con tratamiento difuso: Tipos 1, 2 ó 3
INSERT into FCL values (t_APTITUDES,c_estudios,2,1,USER||'.APTITUDES.ESTUDIOS');
INSERT into FCL values (t_APTITUDES,c_experiencia,2,1,USER||'.APTITUDES.EXPERIENCIA');
INSERT into FCL values (t_APTITUDES,c_profesion,3,1,USER||'.APTITUDES.PROFESION');
INSERT into FCL values (t_EMPLEOS,c_comision,2,1,USER||'.EMPLEOS.COMISION');
INSERT into FCL values (t_EMPLEOS,c_rendimiento,3,1,USER||'.EMPLEOS.RENDIMIENTO');
INSERT into FCL values (t_EMPLEOS,c_sueldo,1,1,USER||'.EMPLEOS.SUELDO');
INSERT into FCL values (t_PUESTOS,c_estudios1,2,1,USER||'.PUESTOS.ESTUDIOS');
INSERT into FCL values (t_PUESTOS,c_experiencia1,2,1,USER||'.PUESTOS.EXPERIENCIA');
INSERT into FCL values (t_PUESTOS,c_profesion1,3,1,USER||'.PUESTOS.PROFESION');
INSERT into FCL values
(t_DEPARTAMENTOS,c_beneficios,2,1,USER||'.DEPARTAMENTOS.BENEFICIOS');
INSERT into FCL values (t_EMPLEADOS,c_edad,2,1,USER||'.EMPLEADOS.EDAD');

-- COMPATIBILIDAD de atributos tipo 3:
-- Campo PUESTOS.profesion es compatible con APTITUDES.profesion
INSERT into FCC values (t_PUESTOS,c_profesion1,t_APTITUDES,c_profesion);
```

-- Objetos para la tabla EMPLEOS:

```
INSERT into FOL values (t_EMPLEOS,c_sueldo,0,'MUY_BAJO',0);
INSERT into FOL values (t_EMPLEOS,c_sueldo,1,'BAJO',0);
INSERT into FOL values (t_EMPLEOS,c_sueldo,2,'MEDIO',0);
INSERT into FOL values (t_EMPLEOS,c_sueldo,3,'ALTO',0);
INSERT into FOL values (t_EMPLEOS,c_sueldo,4,'MUY_ALTO',0);
INSERT into FOL values (t_EMPLEOS,c_sueldo,5,'ALTISIMO',0);
INSERT into FOL values (t_EMPLEOS,c_comision,0,'REDUCIDA',0);
```

```
INSERT into FOL values (t_EMPLEOS,c_comision,1,'BAJA',0);
INSERT into FOL values (t_EMPLEOS,c_comision,2,'NORMAL',0);
INSERT into FOL values (t_EMPLEOS,c_comision,3,'ALTA',0);
INSERT into FOL values (t_EMPLEOS,c_comision,4,'ELEVADA',0);
INSERT into FOL values (t_EMPLEOS,c_rendimiento,0,'MALO',1);
INSERT into FOL values (t_EMPLEOS,c_rendimiento,1,'REGULAR',1);
INSERT into FOL values (t_EMPLEOS,c_rendimiento,2,'BUENO',1);
INSERT into FOL values (t_EMPLEOS,c_rendimiento,3,'MUY_BUENO',1);
INSERT into FOL values (t_EMPLEOS,c_rendimiento,4,'EXCELENTE',1);
```

-- Objetos para la tabla APTITUDES:

```
INSERT into FOL values(t_APTITUDES,c_estudios,0,'PRIMARIA',0);
INSERT into FOL values(t_APTITUDES,c_estudios,1,'SECUNDARIA',0);
INSERT into FOL values(t_APTITUDES,c_estudios,2,'DIPLOMADO',0);
INSERT into FOL values(t_APTITUDES,c_estudios,3,'LICENCIADO',0);
INSERT into FOL values(t_APTITUDES,c_estudios,4,'DOCTOR',0);
INSERT into FOL values(t_APTITUDES,c_profesion,0,'ECONOMISTA',1);
INSERT into FOL values(t_APTITUDES,c_profesion,1,'INFORMATICO',1);
INSERT into FOL values(t_APTITUDES,c_profesion,2,'TELECOMUNICACIONES',1);
INSERT into FOL values(t_APTITUDES,c_profesion,3,'CREATIVO',1);
INSERT into FOL values(t_APTITUDES,c_profesion,4,'ADMINISTRATIVO',1);
INSERT into FOL values(t_APTITUDES,c_profesion,5,'COMERCIAL',1);
INSERT into FOL values(t_APTITUDES,c_experiencia,0,'POCA',0);
INSERT into FOL values(t_APTITUDES,c_experiencia,1,'ALGUNA',0);
INSERT into FOL values(t_APTITUDES,c_experiencia,2,'MUCHA',0);
INSERT into FOL values(t_APTITUDES,c_experiencia,3,'BASTANTE',0);
INSERT into FOL values(t_APTITUDES,c_experiencia,4,'GRANDE',0);
```

-- Objetos para la tabla PUESTOS:

```
INSERT into FOL values(t_PUESTOS,c_estudios1,0,'PRIMARIA',0);
INSERT into FOL values(t_PUESTOS,c_estudios1,1,'SECUNDARIA',0);
INSERT into FOL values(t_PUESTOS,c_estudios1,2,'DIPLOMADO',0);
INSERT into FOL values(t_PUESTOS,c_estudios1,3,'LICENCIADO',0);
INSERT into FOL values(t_PUESTOS,c_estudios1,4,'DOCTOR',0);
INSERT into FOL values(t_PUESTOS,c_experiencia1,0,'POCA',0);
INSERT into FOL values(t_PUESTOS,c_experiencia1,1,'ALGUNA',0);
INSERT into FOL values(t_PUESTOS,c_experiencia1,2,'MUCHA',0);
INSERT into FOL values(t_PUESTOS,c_experiencia1,3,'BASTANTE',0);
INSERT into FOL values(t_PUESTOS,c_experiencia1,4,'GRANDE',0);
```

-- Objetos para la tabla DEPARTAMENTOS:

INSERT into FOL values(t_DEPARTAMENTOS,c_beneficios,0,'ESCASOS',0);
INSERT into FOL values(t_DEPARTAMENTOS,c_beneficios,1,'POCOS',0);
INSERT into FOL values(t_DEPARTAMENTOS,c_beneficios,2,'MODERADOS',0);
INSERT into FOL values(t_DEPARTAMENTOS,c_beneficios,3,'ELEVADOS',0);

-- Objetos para la tabla EMPLEADOS:

INSERT into FOL values(t_EMPLEADOS,c_edad,0,'MUY_JOVEN',0);
INSERT into FOL values(t_EMPLEADOS,c_edad,1,'JOVEN',0);
INSERT into FOL values(t_EMPLEADOS,c_edad,2,'MADURO',0);
INSERT into FOL values(t_EMPLEADOS,c_edad,3,'MAYOR',0);
INSERT into FOL values(t_EMPLEADOS,c_edad,4,'MUY_MAYOR',0);

-- Definición de las etiquetas lingüísticas:
INSERT into FLD values(t_EMPLEADOS,c_edad,0,16,16,20,26);
INSERT into FLD values(t_EMPLEADOS,c_edad,1,18,22,30,35);
INSERT into FLD values(t_EMPLEADOS,c_edad,2,25,32,45,50);
INSERT into FLD values(t_EMPLEADOS,c_edad,3,40,45,55,60);
INSERT into FLD values(t_EMPLEADOS,c_edad,4,50,55,62,70);

INSERT into FLD values(t_APTITUDES,c_experiencia,0,0,0,1,2);
INSERT into FLD values(t_APTITUDES,c_experiencia,1,1,2,4,5);
INSERT into FLD values(t_APTITUDES,c_experiencia,2,4,5,8,10);
INSERT into FLD values(t_APTITUDES,c_experiencia,3,7,8,15,20);
INSERT into FLD values(t_APTITUDES,c_experiencia,4,12,15,50,50);
INSERT into FLD values(t_APTITUDES,c_estudios,0,0,0,1,1);
INSERT into FLD values(t_APTITUDES,c_estudios,1,1,1,2,2);
INSERT into FLD values(t_APTITUDES,c_estudios,2,2,2,3,3);
INSERT into FLD values(t_APTITUDES,c_estudios,3,3,3,4,4);
INSERT into FLD values(t_APTITUDES,c_estudios,4,4,4,5,5);

INSERT into FLD values(t_PUESTOS,c_experiencia1,0,0,0,1,2);
INSERT into FLD values(t_PUESTOS,c_experiencia1,1,1,2,4,5);
INSERT into FLD values(t_PUESTOS,c_experiencia1,2,4,5,8,10);
INSERT into FLD values(t_PUESTOS,c_experiencia1,3,7,8,15,20);
INSERT into FLD values(t_PUESTOS,c_experiencia1,4,12,15,50,50);
INSERT into FLD values(t_PUESTOS,c_estudios1,0,0,0,1,1);
INSERT into FLD values(t_PUESTOS,c_estudios1,1,1,1,2,2);
INSERT into FLD values(t_PUESTOS,c_estudios1,2,2,2,3,3);

```
INSERT into FLD values(t_PUESTOS,c_estudios1,3,3,3,4,4);
INSERT into FLD values(t_PUESTOS,c_estudios1,4,4,4,5,5);

INSERT into FLD values(t_DEPARTAMENTOS,c_beneficios,0,0,1000000,2000000,3000000);
INSERT into FLD
values(t_DEPARTAMENTOS,c_beneficios,1,1500000,3000000,5000000,7000000);
INSERT into FLD
values(t_DEPARTAMENTOS,c_beneficios,2,3000000,4500000,10000000,14000000);
INSERT into FLD
values(t_DEPARTAMENTOS,c_beneficios,3,7000000,10000000,100000000,100000000);

INSERT into FLD values(t_EMPLEOS,c_sueldo,0,50000,55000,80000,100000);
INSERT into FLD values(t_EMPLEOS,c_sueldo,1,75000,90000,140000,155000);
INSERT into FLD values(t_EMPLEOS,c_sueldo,2,125000,155000,195000,205000);
INSERT into FLD values(t_EMPLEOS,c_sueldo,3,195000,215000,285000,310000);
INSERT into FLD values(t_EMPLEOS,c_sueldo,4,300000,320000,400000,420000);
INSERT into FLD values(t_EMPLEOS,c_sueldo,5,410000,430000,1000000,1000000);

INSERT into FLD values(t_EMPLEOS,c_comision,0,0,5000,10000,15000);
INSERT into FLD values(t_EMPLEOS,c_comision,1,11000,15000,20000,30000);
INSERT into FLD values(t_EMPLEOS,c_comision,2,15000,17000,30000,40000);
INSERT into FLD values(t_EMPLEOS,c_comision,3,20000,32000,50000,60000);
INSERT into FLD values(t_EMPLEOS,c_comision,4,40000,50000,200000,300000);

-- Definición de las relaciones de semejanza:
INSERT into FND values(t_APTITUDES,c_profesion,0,1,.6);
INSERT into FND values(t_APTITUDES,c_profesion,0,2,.6);
INSERT into FND values(t_APTITUDES,c_profesion,0,3,.3);
INSERT into FND values(t_APTITUDES,c_profesion,0,4,.8);
INSERT into FND values(t_APTITUDES,c_profesion,0,5,.7);
INSERT into FND values(t_APTITUDES,c_profesion,1,2,.8);
INSERT into FND values(t_APTITUDES,c_profesion,1,3,.6);
INSERT into FND values(t_APTITUDES,c_profesion,1,4,.3);
INSERT into FND values(t_APTITUDES,c_profesion,1,5,.4);
INSERT into FND values(t_APTITUDES,c_profesion,2,3,.4);
INSERT into FND values(t_APTITUDES,c_profesion,2,4,.3);
INSERT into FND values(t_APTITUDES,c_profesion,2,5,.5);
INSERT into FND values(t_APTITUDES,c_profesion,3,4,.2);
INSERT into FND values(t_APTITUDES,c_profesion,3,5,.4);
INSERT into FND values(t_APTITUDES,c_profesion,4,5,.7);
```

```
INSERT into FND values(t_EMPLEOS,c_rendimiento,0,1,.8);
INSERT into FND values(t_EMPLEOS,c_rendimiento,0,2,.4);
INSERT into FND values(t_EMPLEOS,c_rendimiento,0,3,.2);
INSERT into FND values(t_EMPLEOS,c_rendimiento,0,4,.1);
INSERT into FND values(t_EMPLEOS,c_rendimiento,1,2,.7);
INSERT into FND values(t_EMPLEOS,c_rendimiento,1,3,.3);
INSERT into FND values(t_EMPLEOS,c_rendimiento,1,4,.2);
INSERT into FND values(t_EMPLEOS,c_rendimiento,2,3,.8);
INSERT into FND values(t_EMPLEOS,c_rendimiento,2,4,.5);
INSERT into FND values(t_EMPLEOS,c_rendimiento,3,4,.7);

-- Definición de:
-- 1) Márgenes para valores APROX en cada columna y
-- 2) Distancia Mínima para afirmar en los comparadores MGT y MLT.
INSERT into FAM values(t_EMPLEADOS,c_edad,5,10);
-- En EDAD:
-- 1) Si decimos "Aproximadamente 15 años", entendemos 15±5
-- 2) Para que dos edades sean MUY diferentes (MGT ó MLT), tendrán que diferir en 5 años
INSERT into FAM values(t_APTITUDES,c_estudios,.0,1);
INSERT into FAM values(t_APTITUDES,c_experiencia,2,5);
INSERT into FAM values(t_DEPARTAMENTOS,c_beneficios,2000000,4000000);
INSERT into FAM values(t_PUESTOS,c_estudios1,.0,1);
INSERT into FAM values(t_PUESTOS,c_experiencia1,2,5);
INSERT into FAM values(t_EMPLEOS,c_sueldo,10000,50000);
INSERT into FAM values(t_EMPLEOS,c_comision,5000,15000);

-- INSERTAMOS REGISTROS EN LAS TABLAS DEFINIDAS
dbms_output.put_line('>>>> Insertando valores en las tablas de esta Bd Difusa...');
-- *****  INSERCIONES de los valores de las tablas ***** --
INSERT into EMPLEADOS values (0 ,'VICTOR'   ,'GOMEZ
ROBLES','V',4,0,NULL,NULL,NULL,'837483732');
INSERT into EMPLEADOS values (1 ,'JAVIER'   ,'PEREZ
GARCIA','V',3,17,NULL,NULL,NULL,'934758356');
INSERT into EMPLEADOS values (2 ,'OLGA'     ,'DE LA TORRE
MORENO','H',3,27,NULL,NULL,NULL,'836746372');
INSERT into EMPLEADOS values (3 ,'JOSE'     ,'LASA NARANJO','V',5,25,0,0,30,'952834758');
INSERT into EMPLEADOS values (4 ,'SALVADOR' ,'DE MIGUEL MARTOS','V',7,40,5,-
7,55,'937463037');
INSERT into EMPLEADOS values (5 ,'SANDRA'   ,'ROMAN ROJAS','H',6,30,25,35,5,'937461048');
INSERT into EMPLEADOS values (6 ,'ANA'      ,'LOPEZ
DOBLAS','H',0,23,NULL,NULL,NULL,'952837473');
```

```
INSERT into EMPLEADOS values (7 ,'IGNACIO'  ,'ROBLEDO
LAIZ','V',3,45,NULL,NULL,NULL,'988762100');

INSERT into EMPLEADOS values (8 ,'LUIS'     ,'DE TOMAS
RUIZ','V',3,18,NULL,NULL,NULL,'983746211');

INSERT into EMPLEADOS values (9 ,'ALBERTO'  ,'GUTIERREZ
DENIA','V',3,23,NULL,NULL,NULL,'95263482');

INSERT into EMPLEADOS values (10,'ALVARO'   ,'CID
LAFUENTE','V',4,1,NULL,NULL,NULL,'952847384');

INSERT into EMPLEADOS values (11,'LAURA'    ,'CABELLO
RUBIO','H',4,3,NULL,NULL,NULL,'98473622');

INSERT into EMPLEADOS values (12,'CARLOS'   ,'TURIEL
ARGANDA','V',4,4,NULL,NULL,NULL,'916273633');

INSERT into EMPLEADOS values (13,'CRISTINA' ,'ANTEQUERA
GARCIA','H',4,2,NULL,NULL,NULL,'952003422');

INSERT into EMPLEADOS values (14,'INES'     ,'SANZ
RATO','H',4,1,NULL,NULL,NULL,'926374826');

INSERT into EMPLEADOS values (15,'JULIAN'   ,'BLANCO
SANCHEZ','V',3,31,NULL,NULL,NULL,'928374423');

INSERT into EMPLEADOS values (16,'ANTONIO'  ,'MARTIN
GONZALEZ','V',3,55,NULL,NULL,NULL,'988273100');

INSERT into EMPLEADOS values (17,'SILVIA'   ,'VALENCIA
MARMOLEJO','H',6,35,30,40,5,'983273441');

INSERT into EMPLEADOS values (18,'PALOMA'   ,'SAN MARTIN PICARD','H',7,28,4,-
3,42,'916273300');

INSERT into EMPLEADOS values (19,'DAVID'    ,'MOLINA
FERNANDEZ','V',4,1,NULL,NULL,NULL,'934333232');

INSERT into EMPLEADOS values (20,'CARMEN'   ,'VALDIVIESO
FUENTES','H',3,60,NULL,NULL,NULL,'923844091');

INSERT into EMPLEADOS values (21,'JUAN'     ,'ZAMORA
JIMENEZ','V',3,48,NULL,NULL,NULL,'928732278');

INSERT into EMPLEADOS values (22,'RAFA'     ,'VAZQUEZ
SORIA','V',3,29,NULL,NULL,NULL,'952272288');

INSERT into EMPLEADOS values (23,'JOSE LUIS','RICO
HERAS','V',3,34,NULL,NULL,NULL,'98433011');

INSERT into EMPLEADOS values (24,'PATRICIA' ,'RODRIGUEZ
SILVA','H',3,27,NULL,NULL,NULL,'951772122');

INSERT into EMPLEADOS values (25,'DOLORES'  ,'NARVAEZ
SEGOVIA','H',3,34,NULL,NULL,NULL,'951233037');


INSERT into APTITUDES values (0 ,4,0,NULL,NULL,NULL,3,1,5,4,0,NULL,NULL,NULL);

INSERT into APTITUDES values (1 ,4,0,NULL,NULL,NULL,3,1,3,4,0,NULL,NULL,NULL);

INSERT into APTITUDES values (2 ,4,2,NULL,NULL,NULL,3,1,1,4,1,NULL,NULL,NULL);

INSERT into APTITUDES values (3 ,4,1,NULL,NULL,NULL,3,1,3,3,3,NULL,NULL,NULL);

INSERT into APTITUDES values (4 ,4,1,NULL,NULL,NULL,3,1,5,3,20,NULL,NULL,NULL);

INSERT into APTITUDES values (5 ,4,4,NULL,NULL,NULL,3,1,0,5,15,NULL,NULL,20);

INSERT into APTITUDES values (6 ,4,0,NULL,NULL,NULL,3,1,4,0,23,NULL,NULL,NULL);
```

```
INSERT into APTITUDES values (7 ,4,3,NULL,NULL,NULL,3,1,2,4,3,NULL,NULL,NULL);
INSERT into APTITUDES values (8 ,4,1,NULL,NULL,NULL,3,1,5,3,1,NULL,NULL,NULL);
INSERT into APTITUDES values (9 ,4,1,NULL,NULL,NULL,3,1,3,6,2,0,4,2);
INSERT into APTITUDES values (10,4,1,NULL,NULL,NULL,3,1,4,4,1,NULL,NULL,NULL);
INSERT into APTITUDES values (11,4,0,NULL,NULL,NULL,3,1,0,4,3,NULL,NULL,NULL);
INSERT into APTITUDES values (12,4,2,NULL,NULL,NULL,3,1,1,3,20,NULL,NULL,NULL);
INSERT into APTITUDES values (13,4,3,NULL,NULL,NULL,3,1,3,3,15,NULL,NULL,NULL);
INSERT into APTITUDES values (14,4,1,NULL,NULL,NULL,3,1,4,6,3,1,5,2);
INSERT into APTITUDES values (15,4,2,NULL,NULL,NULL,3,1,1,7,2,4,-4,17);
INSERT into APTITUDES values (16,4,1,NULL,NULL,NULL,3,1,3,4,3,NULL,NULL,NULL);
INSERT into APTITUDES values (17,4,2,NULL,NULL,NULL,3,1,1,3,7,NULL,NULL,NULL);
INSERT into APTITUDES values (18,4,2,NULL,NULL,NULL,3,1,2,3,4,NULL,NULL,NULL);
INSERT into APTITUDES values (19,4,1,NULL,NULL,NULL,3,1,5,4,1,NULL,NULL,NULL);
INSERT into APTITUDES values (20,4,2,NULL,NULL,NULL,3,1,2,4,4,NULL,NULL,NULL);
INSERT into APTITUDES values (21,4,1,NULL,NULL,NULL,3,1,2,4,3,NULL,NULL,NULL);
INSERT into APTITUDES values (22,4,3,NULL,NULL,NULL,3,1,1,4,1,NULL,NULL,NULL);
INSERT into APTITUDES values (23,4,4,NULL,NULL,NULL,3,1,1,4,2,NULL,NULL,NULL);
INSERT into APTITUDES values (24,4,0,NULL,NULL,NULL,3,1,5,6,15,13,17,2);
INSERT into APTITUDES values (25,4,0,NULL,NULL,NULL,3,1,4,4,0,NULL,NULL,NULL);

INSERT into DEPARTAMENTOS values (0,'DIRECCION',6,3000000,1000000,5000000,2000000);
INSERT into DEPARTAMENTOS values (1,'PROGRAMACION',4,2,NULL,NULL,NULL);
INSERT into DEPARTAMENTOS values (2,'DISEÑO',5,5000000,NULL,NULL,8000000);
INSERT into DEPARTAMENTOS values (3,'SISTEMAS',7,2000000,500000,-500000,6000000);
INSERT into DEPARTAMENTOS values (4,'ADMINISTRACION',3,1500000,NULL,NULL,NULL);
INSERT into DEPARTAMENTOS values (5,'COMERCIAL',4,0,NULL,NULL,NULL);

INSERT into PUESTOS values (0 ,'DIRECTOR'
        ,4,3,NULL,NULL,NULL,3,1,0,4,4,NULL,NULL,NULL);
INSERT into PUESTOS values (1 ,'JEFE DPTO'
        ,4,3,NULL,NULL,NULL,3,1,1,4,2,NULL,NULL,NULL);
INSERT into PUESTOS values (2 ,'ANALISTA'
        ,4,2,NULL,NULL,NULL,3,1,1,4,2,NULL,NULL,NULL);
INSERT into PUESTOS values (3 ,'PROGRAMADOR'
        ,4,2,NULL,NULL,NULL,3,1,1,4,1,NULL,NULL,NULL);
INSERT into PUESTOS values (4 ,'TECNICO'
        ,4,1,NULL,NULL,NULL,3,1,2,4,0,NULL,NULL,NULL);
INSERT into PUESTOS values (5 ,'DISEÑADOR'
        ,4,1,NULL,NULL,NULL,3,1,2,4,1,NULL,NULL,NULL);
INSERT into PUESTOS values (6 ,'ADMINISTRATIVO'
        ,4,1,NULL,NULL,NULL,3,1,3,4,1,NULL,NULL,NULL);
INSERT into PUESTOS values (7 ,'ADMINISTRATIVO AUX'
        ,4,1,NULL,NULL,NULL,3,1,3,4,0,NULL,NULL,NULL);
```

```
  INSERT into PUESTOS values (8 ,'SECRETARIO'
        ,4,0,NULL,NULL,NULL,3,1,4,4,1,NULL,NULL,NULL);
  INSERT into PUESTOS values (9 ,'COMERCIAL'
        ,4,0,NULL,NULL,NULL,3,1,5,4,2,NULL,NULL,NULL);
  INSERT into PUESTOS values (10,'COMERCIAL AUX1'
        ,4,0,NULL,NULL,NULL,3,1,5,3,0,NULL,NULL,NULL);
  INSERT into PUESTOS values (11,'COMERCIAL AUX2'
        ,4,0,NULL,NULL,NULL,3,1,5,0,NULL,NULL,NULL,NULL);
  INSERT into PUESTOS values (12,'COMERCIAL AUX3'
        ,4,0,NULL,NULL,NULL,3,1,5,1,NULL,NULL,NULL,NULL);

  INSERT into EMPLEOS values (0 ,5,11, 60000,3,10000,NULL,NULL,NULL,3,1,1);
  INSERT into EMPLEOS values (1 ,5,10, 80000,3,5000,NULL,NULL,NULL,3,1,3);
  INSERT into EMPLEOS values (2 ,1,1 ,230000,4,2,NULL,NULL,NULL,3,1,1);
  INSERT into EMPLEOS values (3 ,1,3 ,120000,3,10000,NULL,NULL,NULL,3,1,2);
  INSERT into EMPLEOS values (4 ,5,1 ,160000,4,3,NULL,NULL,NULL,3,1,3);
  INSERT into EMPLEOS values (5 ,0,0 ,420000,3,30000,NULL,NULL,NULL,3,1,2);
  INSERT into EMPLEOS values (6 ,2,4 ,120000,4,0,NULL,NULL,NULL,3,1,1);
  INSERT into EMPLEOS values (7 ,0,0 ,500000,4,4,NULL,NULL,NULL,3,1,3);
  INSERT into EMPLEOS values (8 ,2,5 , 90000,3,35000,NULL,NULL,NULL,3,1,2);
  INSERT into EMPLEOS values (9 ,1,3 ,120000,6,5000,0,10000,5000,3,1,2);
  INSERT into EMPLEOS values (10,3,4 ,130000,4,1,NULL,NULL,NULL,3,1,0);
  INSERT into EMPLEOS values (11,1,2 ,450000,4,4,NULL,NULL,NULL,3,1,4);
  INSERT into EMPLEOS values (12,3,1 ,215000,3,40000,NULL,NULL,NULL,3,1,2);
  INSERT into EMPLEOS values (13,4,1 ,375000,3,50000,NULL,NULL,NULL,3,1,0);
  INSERT into EMPLEOS values (14,3,4 ,150000,4,1,NULL,NULL,NULL,3,1,2);
  INSERT into EMPLEOS values (15,2,1 ,230000,7,50000,10000,-15000,100000,3,1,2);
  INSERT into EMPLEOS values (16,1,3 ,170000,4,2,NULL,NULL,NULL,3,1,3);
  INSERT into EMPLEOS values (17,4,6 ,135000,4,0,NULL,NULL,NULL,3,1,0);
  INSERT into EMPLEOS values (18,2,5 ,160000,3,20000,NULL,NULL,NULL,3,1,2);
  INSERT into EMPLEOS values (19,4,7 ,120000,4,0,NULL,NULL,NULL,3,1,3);
  INSERT into EMPLEOS values (20,0,2 ,255000,4,3,NULL,NULL,NULL,3,1,2);
  INSERT into EMPLEOS values (21,3,2 ,205000,6,25000,20000,30000,5000,3,1,2);
  INSERT into EMPLEOS values (22,1,2 ,200000,4,1,NULL,NULL,NULL,3,1,4);
  INSERT into EMPLEOS values (23,5,9 , 85000,4,2,NULL,NULL,NULL,3,1,1);
  INSERT into EMPLEOS values (24,5,12, 85000,4,1,NULL,NULL,NULL,3,1,0);
  INSERT into EMPLEOS values (25,0,8 , 70000,3,0,NULL,NULL,NULL,3,1,1);
  COMMIT;
END;
/

-- Mensaje de FIN:
```

```
exec dbms_output.put_line('>>>>');
exec dbms_output.put_line('>>>> FIN de la INSTALACIÓN: Si existieron errores, revise su texto.');
exec dbms_output.put_line('>>>>');

-- Reestablecer los valores por defecto:
set serveroutput off
set feedback on
```

6.3. Instalación del Servidor Web "Internet Information Server" (IIS)

Dependiendo del Sistema Operativo que tengamos en el servidor vamos a instalar el Internet Information Server (en adelante IIS). La instalación del IIS proporciona a nuestro Sistema Operativo los Servicios necesarios para funcionar como Servidor web con soporte de páginas ASP.

6.3.1. Instalación

Insertaremos el CD de instalación de Windows 2000 (ó de cualquier sistema operativo con soporte de IIS, p.ej. Windows Server 2003, Windows XP, Windows NT, etc...).

- Nos vamos a: Inicio -> Panel de Control -> Agregar o quitar programas -> Agregar o quitar componentes de Windows (aparecerá la ventana de la Figura 6.10).
- En esta pantalla marcamos "Servicios de Internet Information Server (IIS)" y pulsamos en "Siguiente".

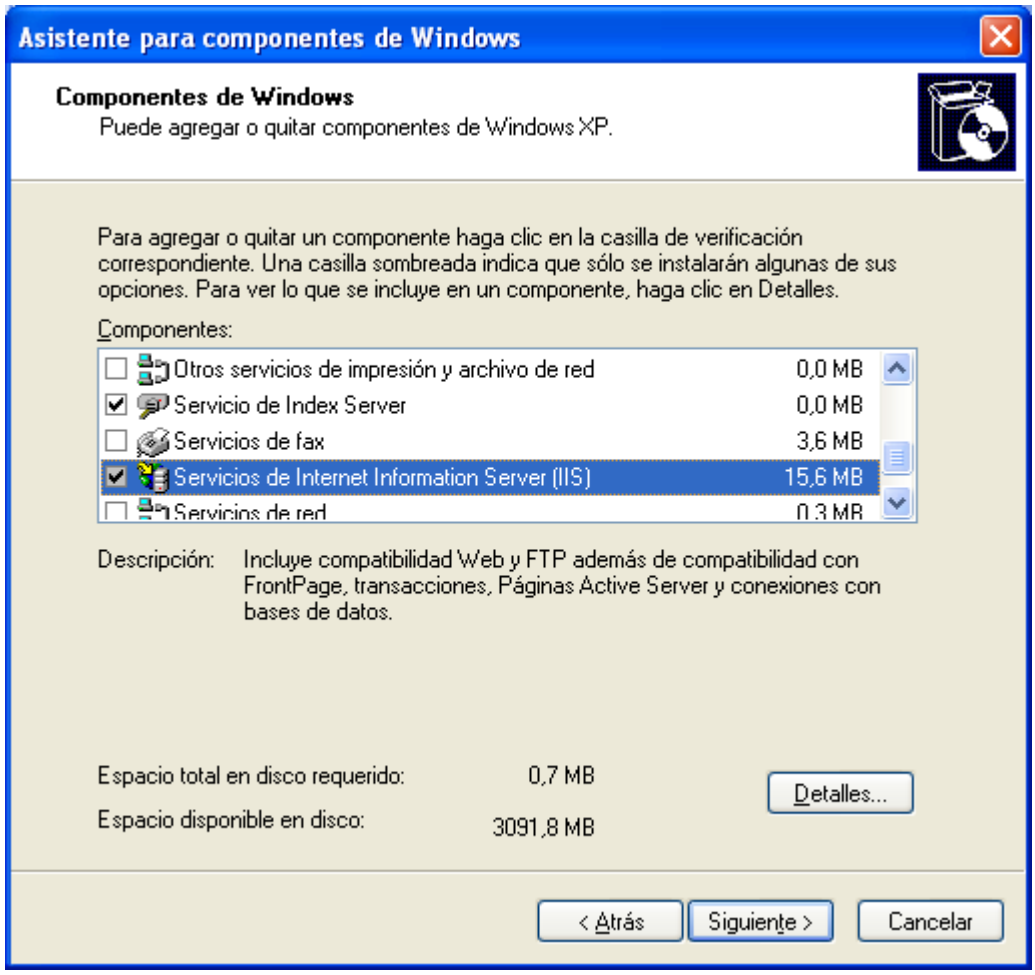


Figura 6.10. Ventana de instalación de componentes de Windows.

Una vez finalizada la instalación ya podemos configurar el IIS.

6.3.2. Configuración del IIS

La configuración del IIS consiste en crear una carpeta donde tendremos alojada nuestra aplicación web, a continuación definiremos una serie de parámetros que serán fundamentales para el buen funcionamiento del sitio web.

Nos vamos a: Inicio -> Panel de Control -> Rendimiento y Mantenimiento -> Herramientas Administrativas -> Servicios de Internet Information Server (ver Figura 6.11).

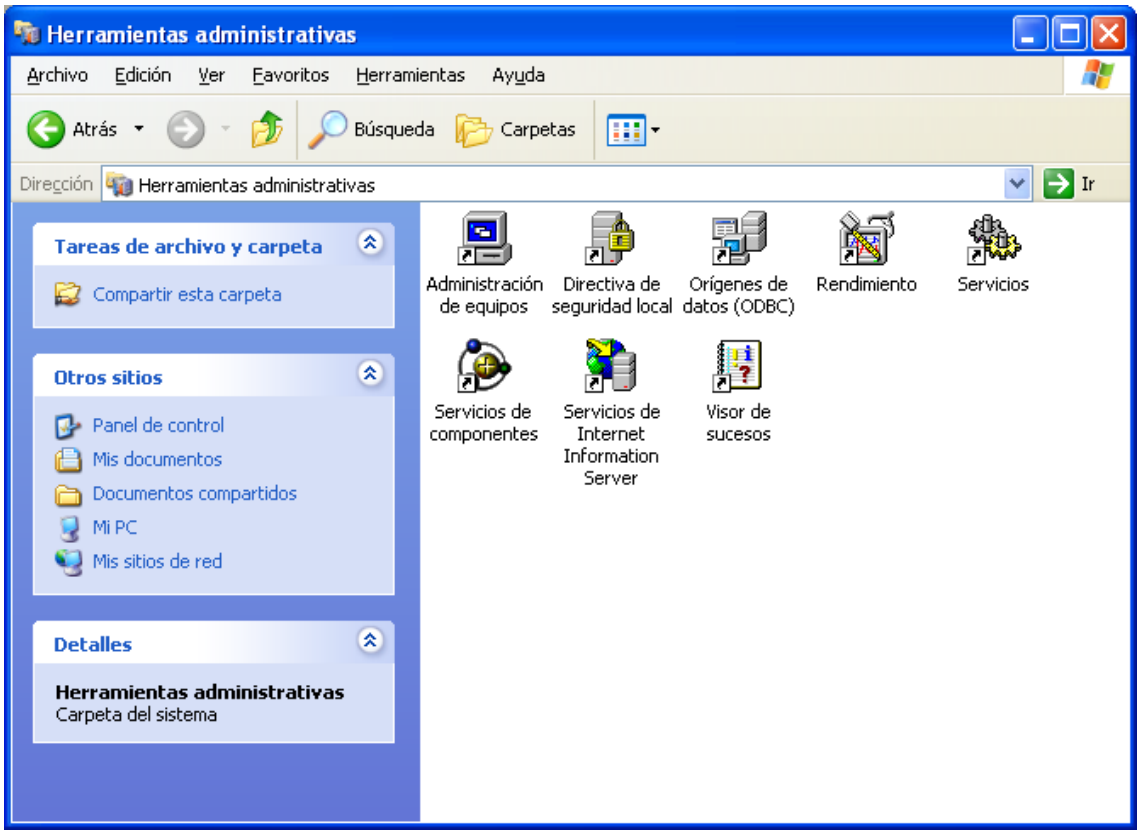


Figura 6.11. Ventana Herramientas Administrativas de Windows.

Vemos que tenemos un equipo asignado a los Servicios del IIS. A continuación vamos desplegando el arbol hasta ver "Sitio Web Predeterminado". También podemos observar en la parte derecha la versión del IIS instalada, en este caso la 5.1 (ver Figura 6.12).

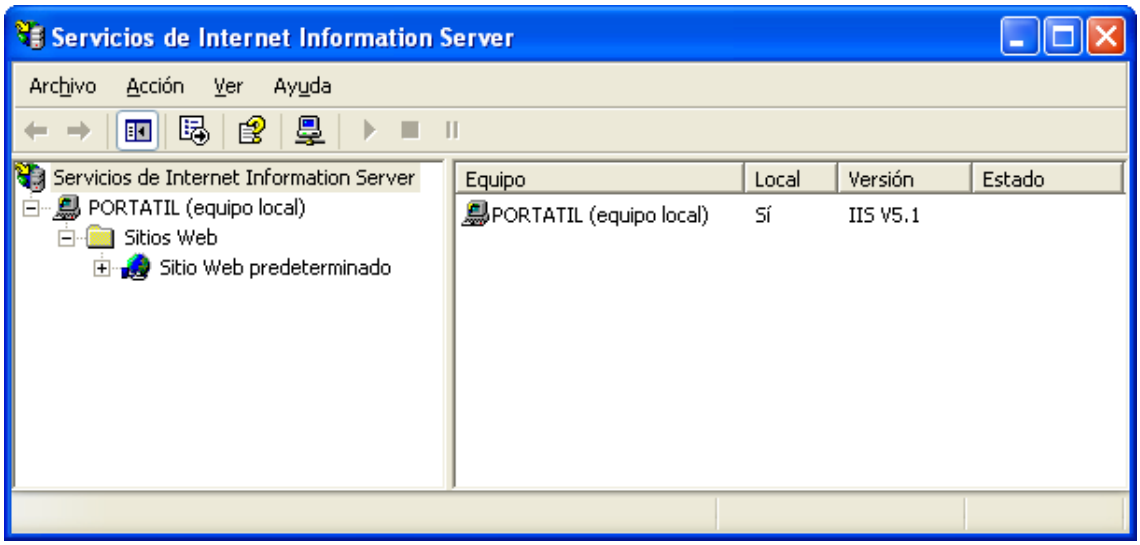


Figura 6.12. Ventana de Servicios de Internet Information Server (IIS) 1/2.

Este es el sitio web que se crea por defecto al instalar el IIS, y a partir de aquí podremos ir añadiendo directorios virtuales donde alojaremos nuestras aplicaciones web. En nuestro caso vamos a crear un nuevo directorio virtual para nuestra aplicación llamado "VisualFSQL".

Pinchamos con el botón derecho sobre "Sitio Web Predeterminado" y "Nuevo Directorio Virtual". Entonces se nos mostrará un Asistente en el que indicaremos lo siguiente:

- Alias: El nombre de nuestro sitio cuando vayamos a acceder "VisualFSQL".
- Directorio: Nos vamos por defecto al raíz del IIS ("C:\Inetpub\wwwroot\") y a partir de aquí creamos una carpeta con nombre "VisualFSQL".
- En los "permisos de acceso" dejamos por defecto las 2 primera opciones marcadas (Leer y Ejecutar secuencias de comandos: ASP).
- Finalizamos el Asistente.

En la pantalla principal del IIS vemos que ya tenemos creado el sitio web para alojar nuestra aplicación (ver Figura 6.13).

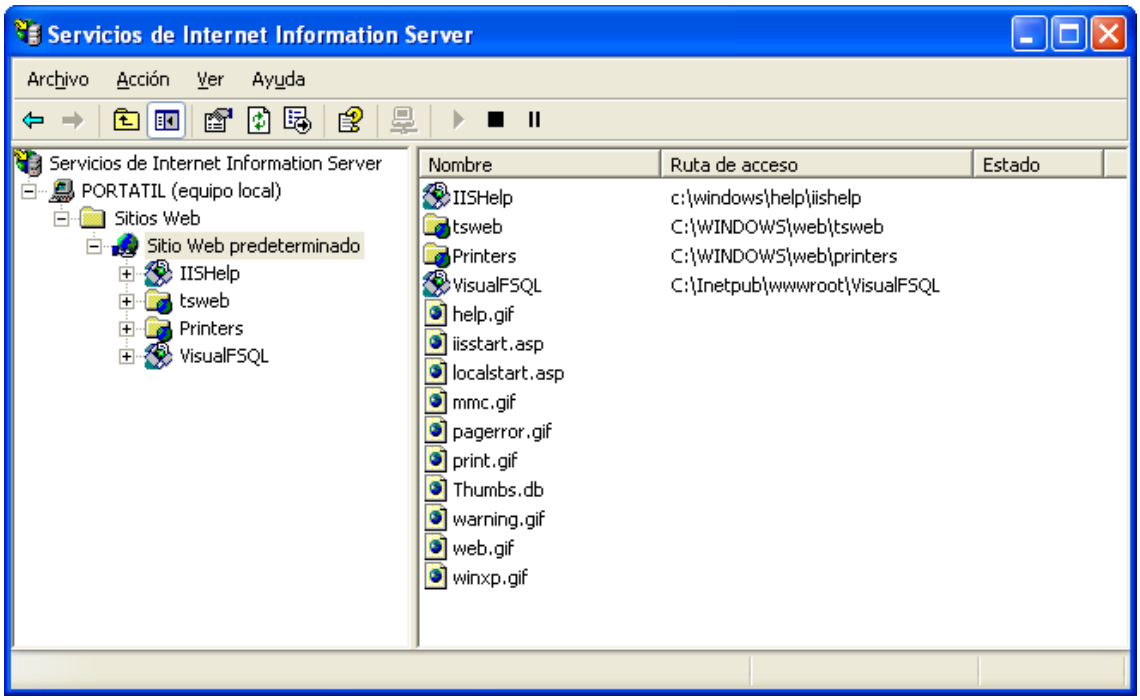


Figura 6.13. Ventana de Servicios de Internet Information Server (IIS) 2/2.

Ahora ya podemos editar las propiedades de nuestro Sitio: Botón derecho -> Propiedades. Y ponemos las opciones tal y como aparecen en la Figura 6.14.

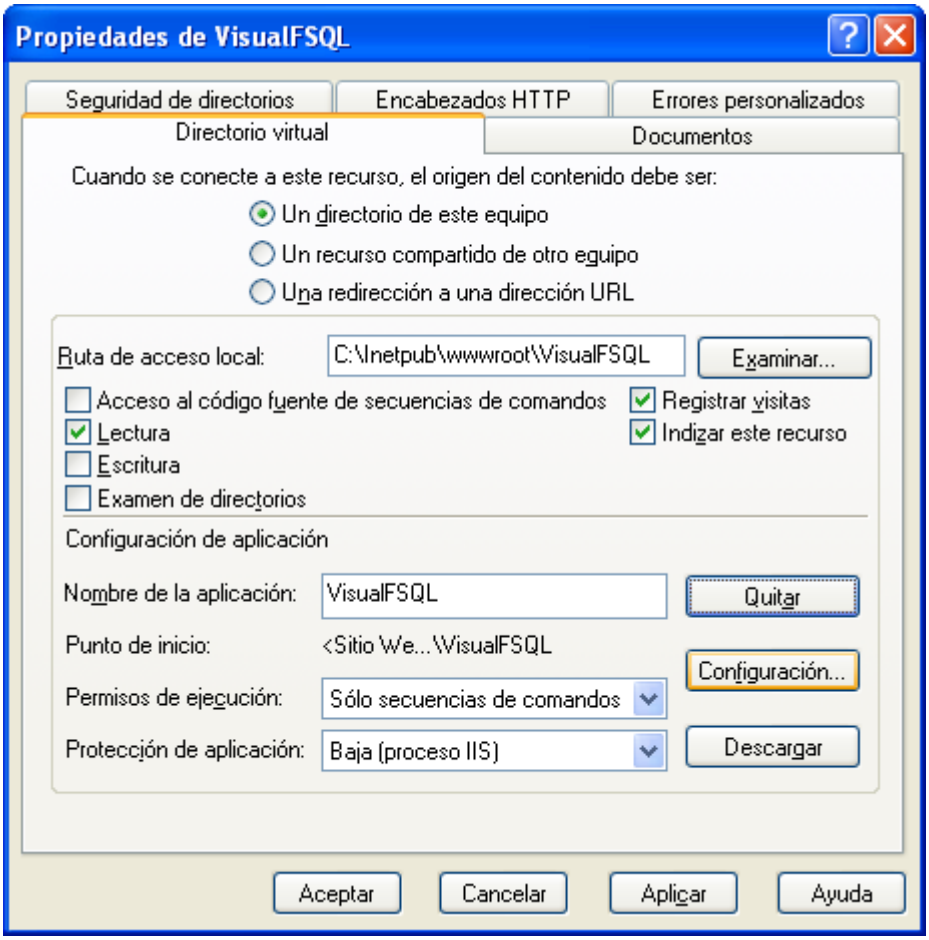


Figura 6.14. Ventana de propiedades de sitio web "VisualFSQL" (1/2).

A continuación nos vamos a la pestaña "Documentos" y agregamos "index.asp" (ver Figura 6.15). Estos son los documentos y el orden en que buscará por defecto el IIS cuando en el Internet Explorer sólo especifiquemos la dirección web sin indicar el archivo que queremos visualizar.

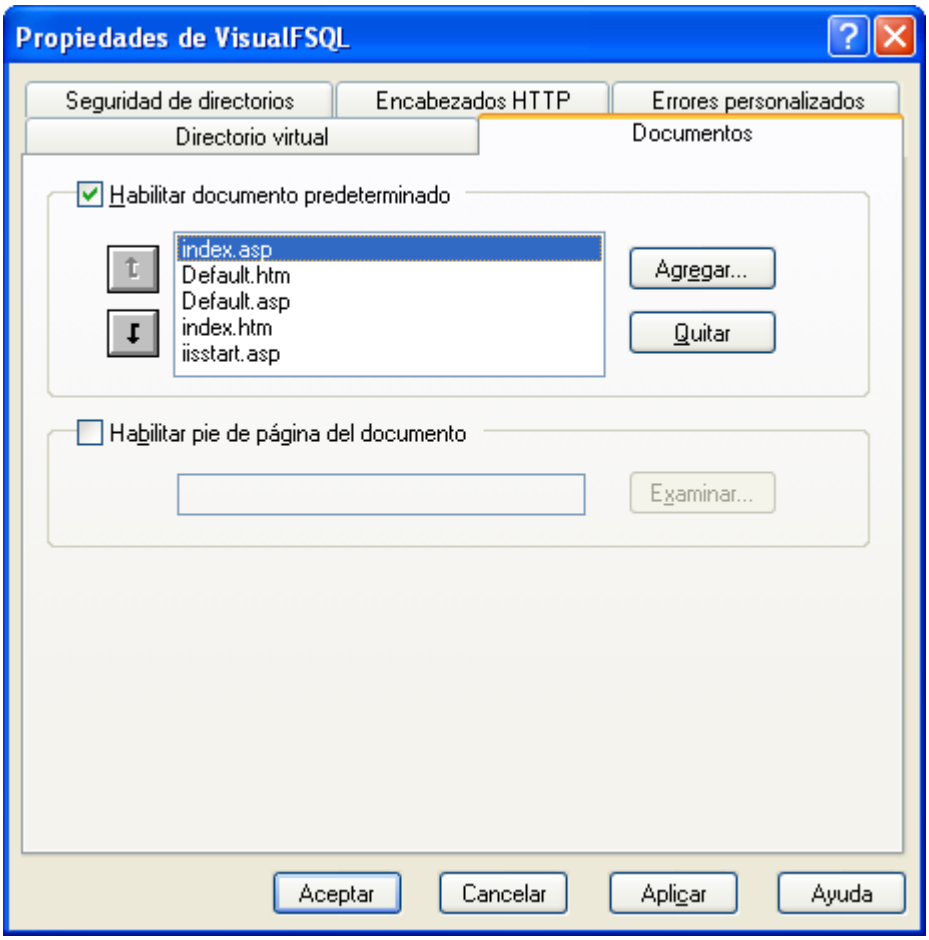


Figura 6.15. Ventana de propiedades de sitio web "VisualFSQL" (2/2).

Para terminar sólo nos quedaría grabar en el directorio "C:\Inetpub\wwwroot\VisualFSQL" todos los archivos y carpetas de nuestra aplicación web. Para ello copiamos toda la estructura de carpetas y archivos ASP de nuestra aplicación y lo pegamos en el directorio indicado anteriormente.

Con esto podemos dar por concluido el proceso de instalación de nuestro sistema.

Capítulo 7

Manual de Usuario de Visual FSQL

Visual FSQL versión 1.0, es un cliente FSQL concebido para efectuar consultas difusas en lenguaje FSQL mediante un asistente visual que nos facilitará mucho la tarea de construir sentencias más complejas. También es posible realizar consultas clásicas en SQL mediante una opción del programa.

El objetivo no es consultar sin saber nada de SQL o FSQL, sino sin que no haya que saber la sintaxis exacta y teniendo casi todo al alcance del ratón (sin casi usar el teclado). Visual FSQL ayuda a aprender la sintaxis de SQL y FSQL.

Esta aplicación tiene un doble enfoque, uno como cliente del sistema Servidor Difuso FSQL y otro como un sistema servidor de navegadores web accediendo como clientes.

Las principales características de Visual FSQL son:

- Es un programa de plataforma abierta, por lo que tenemos un elevado potencial de clientes que se pueden conectar y acceder al sistema. No tenemos la limitación de una plataforma concreta, puesto que cualquier navegador web con soporte Javascript es totalmente operativo.
- Disponibilidad universal a través de Internet y sin necesidad de instalación de ningún cliente ni componente.
- Accesibilidad multiusuario y concurrencia soportada.
- Muy flexible en cuanto a escalabilidad y mantenimiento, puesto que cualquier modificación se realiza de forma centralizada en el servidor de la aplicación.
- Interfaz de usuario sencilla y guiada a través de asistentes y ayudas para la construcción de consultas difusas ó clásicas.

- Sistema de almacenamiento y recuperación de consultas generadas en el programa.
- Mínimos requisitos de hardware, únicamente supeditados a las necesidades de Oracle si se instala éste en el mismo servidor. Para el VisualFSQL son necesarios un mínimo de 5 Mb. de espacio en disco y 128 Mb. de RAM, basado fundamentalmente en los requisitos recomendados para el Servidor Web IIS "Internet Information Server" de Microsoft, que es el encargado de dar servicio a la Aplicación.
- También existe la posibilidad de balancear la carga del sistema separándolo en 2 máquinas, por un lado el servidor web IIS y la Aplicación VisualFSQL y por otro el Servidor de Bases de Datos Oracle y el sistema servidor FSQL.

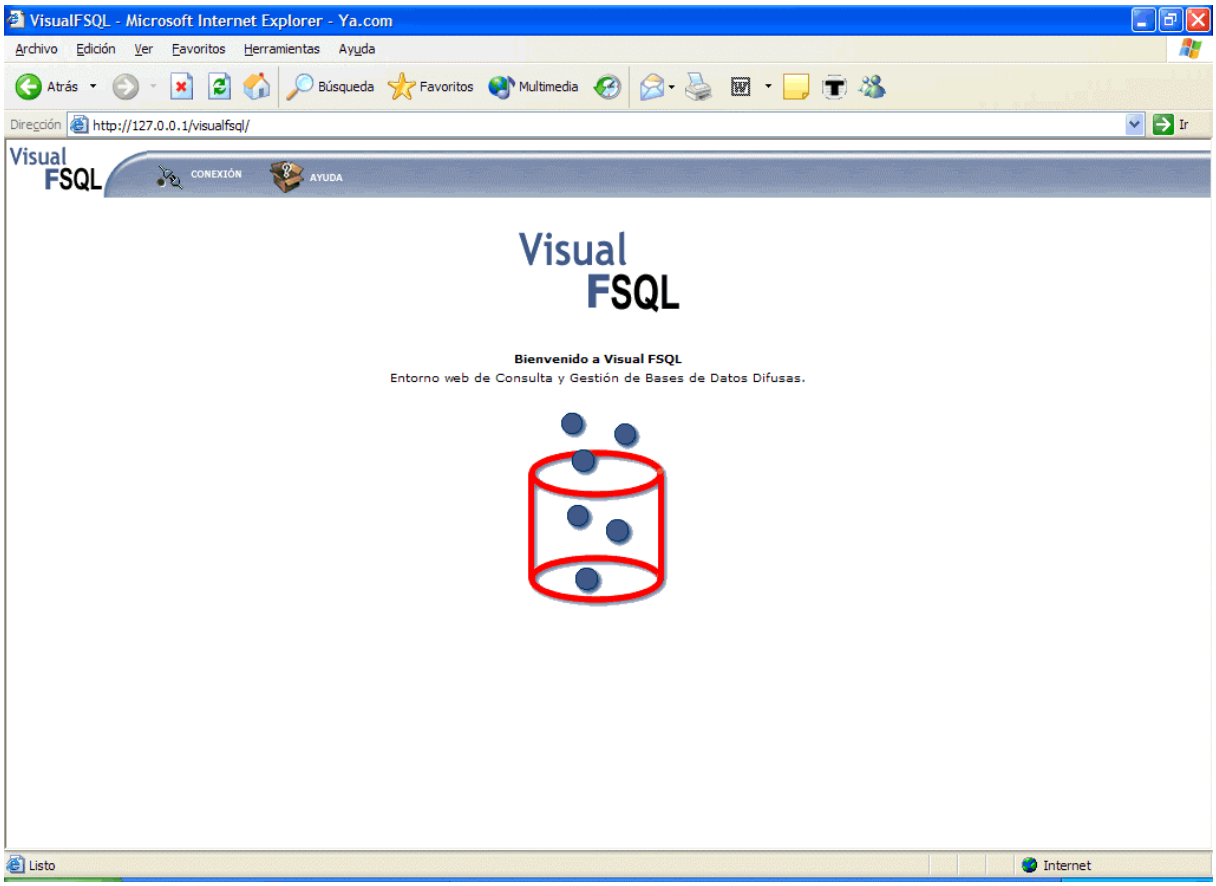


Figura 7.1. Pantalla de bienvenida al Sistema Visual FSQL.

En la Figura 7.1, podemos ver la pantalla inicial de la aplicación, donde tenemos la parte central de navegación (en el mensaje de bienvenida) y la Barra de Menús, que inicialmente sólo nos permite Acceder a Conexión y a la Ayuda y que una vez conectados al servidor se completaría con más opciones. Si pulsamos en el Logo de la aplicación "Visual

FSQL" en la parte superior izquierda,, se abrirá una ventana con los créditos del sistema: Autor, Director del Proyecto, etc...

7.1. CONEXIÓN

Si pulsamos en el botón "Conexión" del Menú nos iremos a la pantalla de conexión, donde vamos a disponer de todas las opciones necesarias para establecer la comunicación con el sistema y tendremos la posibilidad de visualizar información de la conexión establecida.

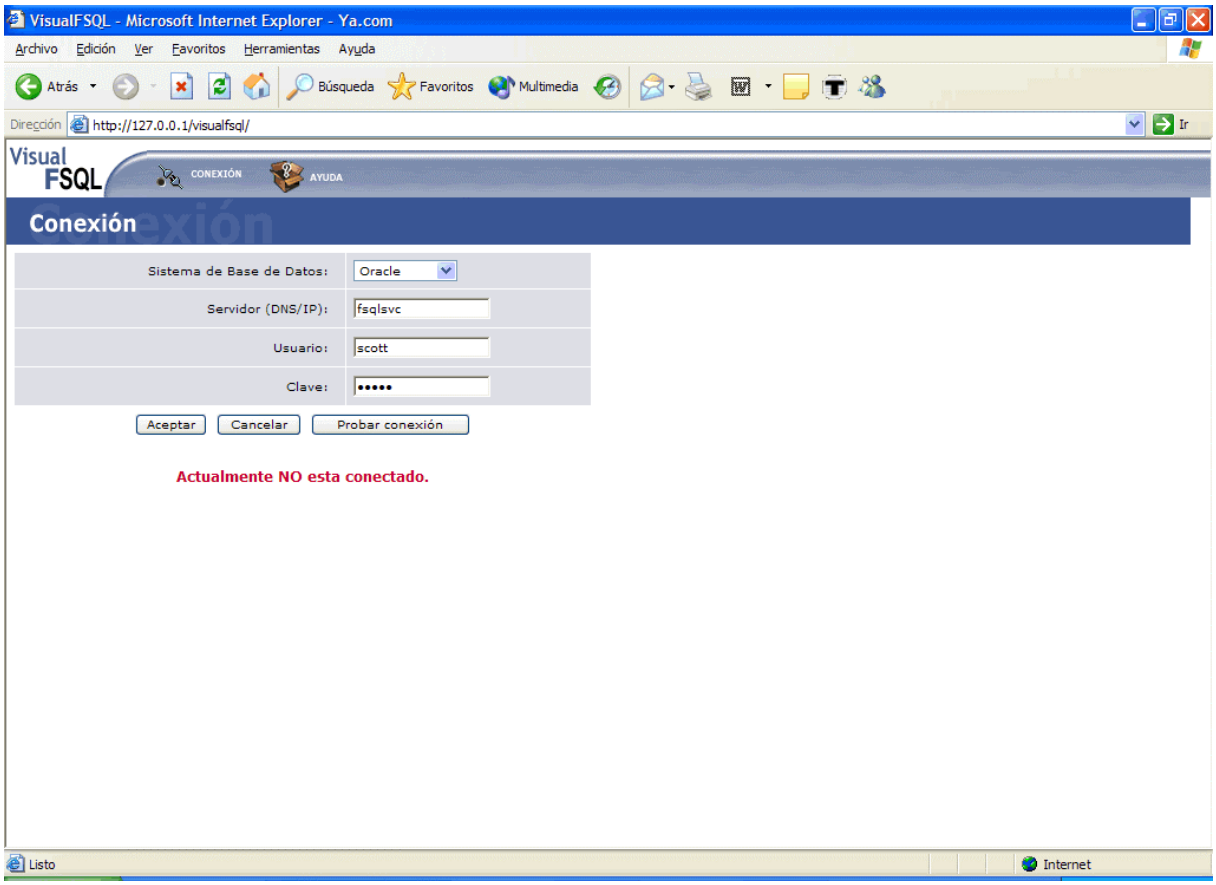


Figura 7.2. Pantalla de Conexión.

En la Figura 7.2 podemos ver la pantalla inicial de conexión, nos indica que aún no estamos conectados al sistema, para ello debemos completar los siguientes campos:

- Sistema de Base de Datos: Siempre estará seleccionado Oracle que de momento es la

única opción disponible, aunque en un futuro será posible trabajar con otros sistemas de Bases de Datos, como SQL-Server o Access.

- Servidor (Nombre de Servicio Oracle): Es el servicio que configuramos en Oracle y hace referencia a la Base de Datos del Sistema.
- Usuario: El nombre de usuario, se puede utilizar por ejemplo SCOTT, que es un usuario que se crea automáticamente al instalar Oracle.
- Password: La contraseña de acceso al sistema, va relacionada con el usuario, por ejemplo para el usuario SCOTT la contraseña es TIGER.

Podemos pulsar en el botón "Probar conexión" para comprobar si los valores introducidos son correctos. Si se produce un error al intentar conectarnos al sistema, nos habría aparecido una pantalla como la de la Figura 7.3, en la que se mostraría el código de error y una simple descripción del mismo.

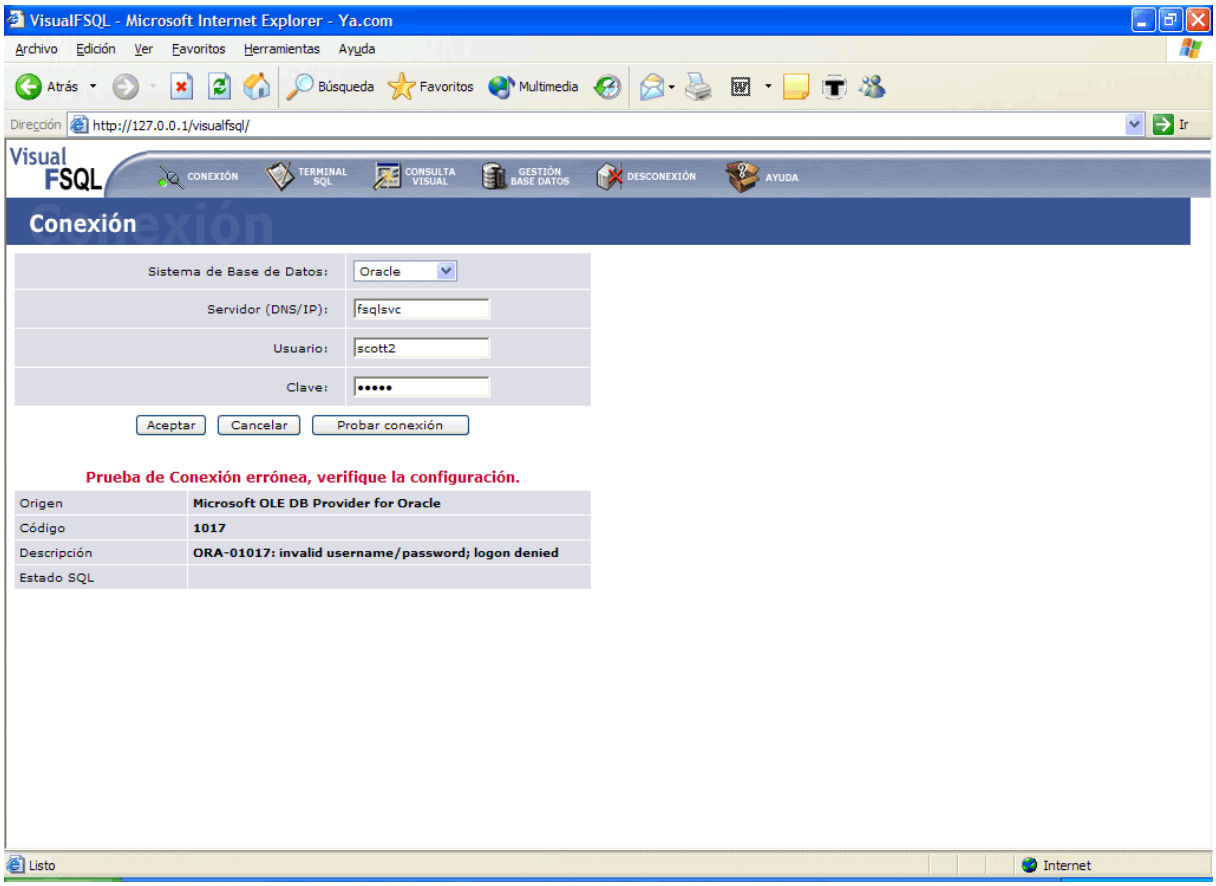


Figura 7.3. Pantalla de Error en la Conexión.

Una vez probada la conexión y si ésta se ha establecido correctamente, aparece a la derecha de la pantalla la cabecera "INFORMACIÓN DE LA CONEXIÓN" (ver Figura 7.4). Si pulsamos en el signo "[+]" veremos Información de la conexión y del SGBD Oracle al que nos hemos conectado (ver Figura 7.5).

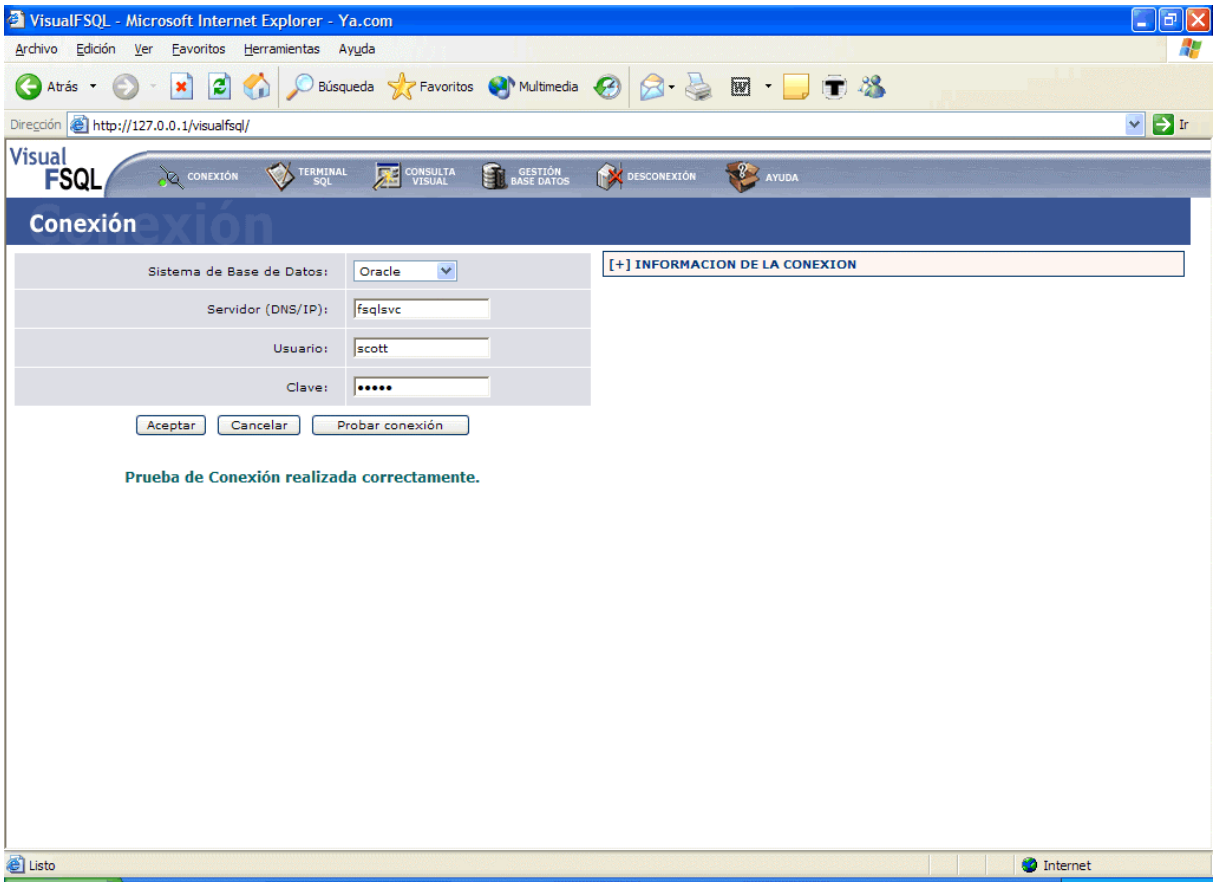


Figura 7.4. Pantalla de Conexión establecida correctamente.

Se puede observar en la Figura 7.4, que una vez conectados nos aparece en la Barra de Menús todas las opciones de la Aplicación que serán explicadas más adelante.

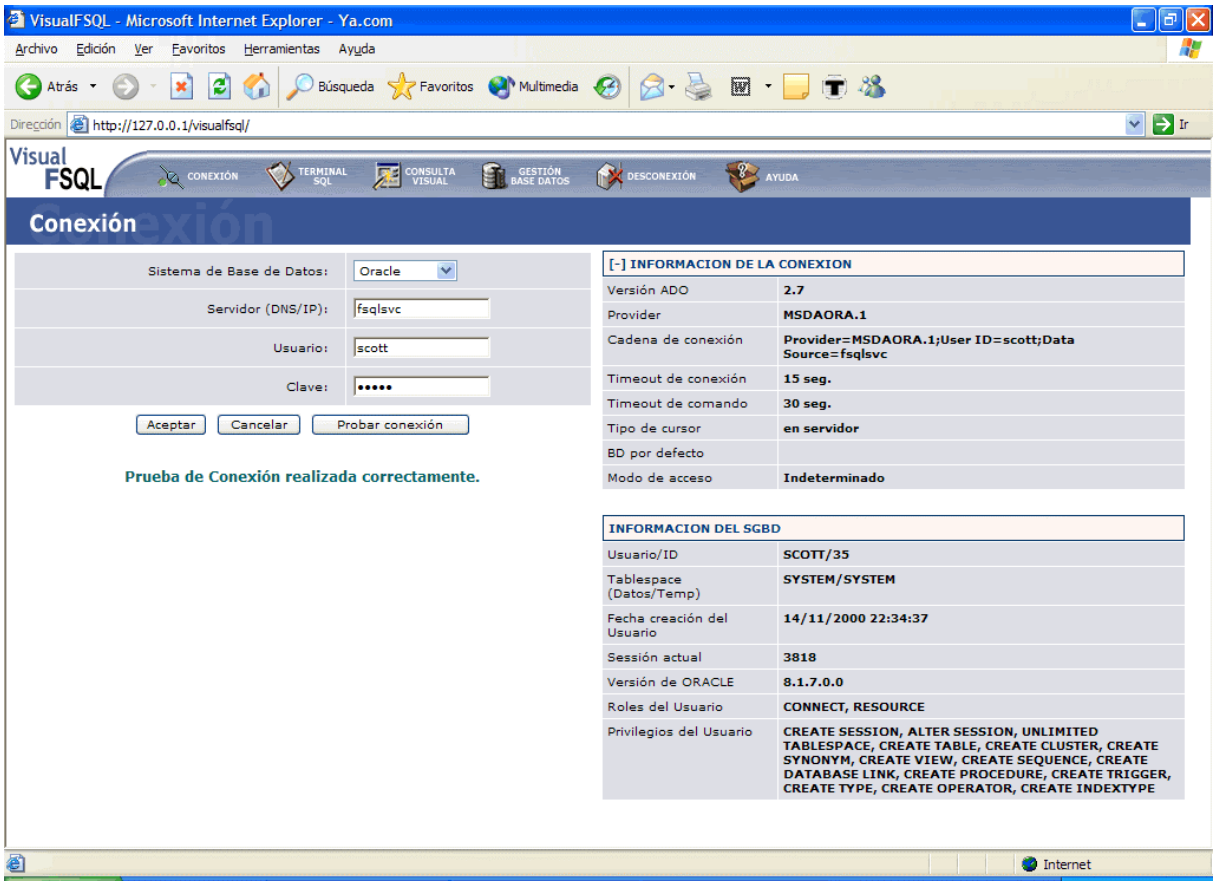


Figura 7.5. Pantalla Información de la Conexión y del SGBD.

Para finalizar el proceso de conexión y empezar a trabajar con el sistema, pulsamos en el botón Aceptar que será el que nos reenviará a la pantalla del Asistente Visual de Consultas FSQL, que se verá más adelante (Apartado 7.4).

También podemos conectarnos directamente pulsando en "Aceptar" sin necesidad de probar previamente la conexión.

7.2. DESCONEXIÓN

Siempre que estemos conectados al sistema podemos finalizar la sesión de conexión pulsando en la opción "Desconexión" de la Barra de Menús superior (se puede ver en la Figura 7.4 ó Figura 7.5).

7.3. TERMINAL SQL

Esta es la segunda opción del Menú, y en esta pantalla vamos a poder construir y editar las sentencias SQL ó FSQL manualmente, sin asistente, además tendremos la posibilidad de grabar las consultas construidas o leer las almacenadas anteriormente para volverlas a utilizar.

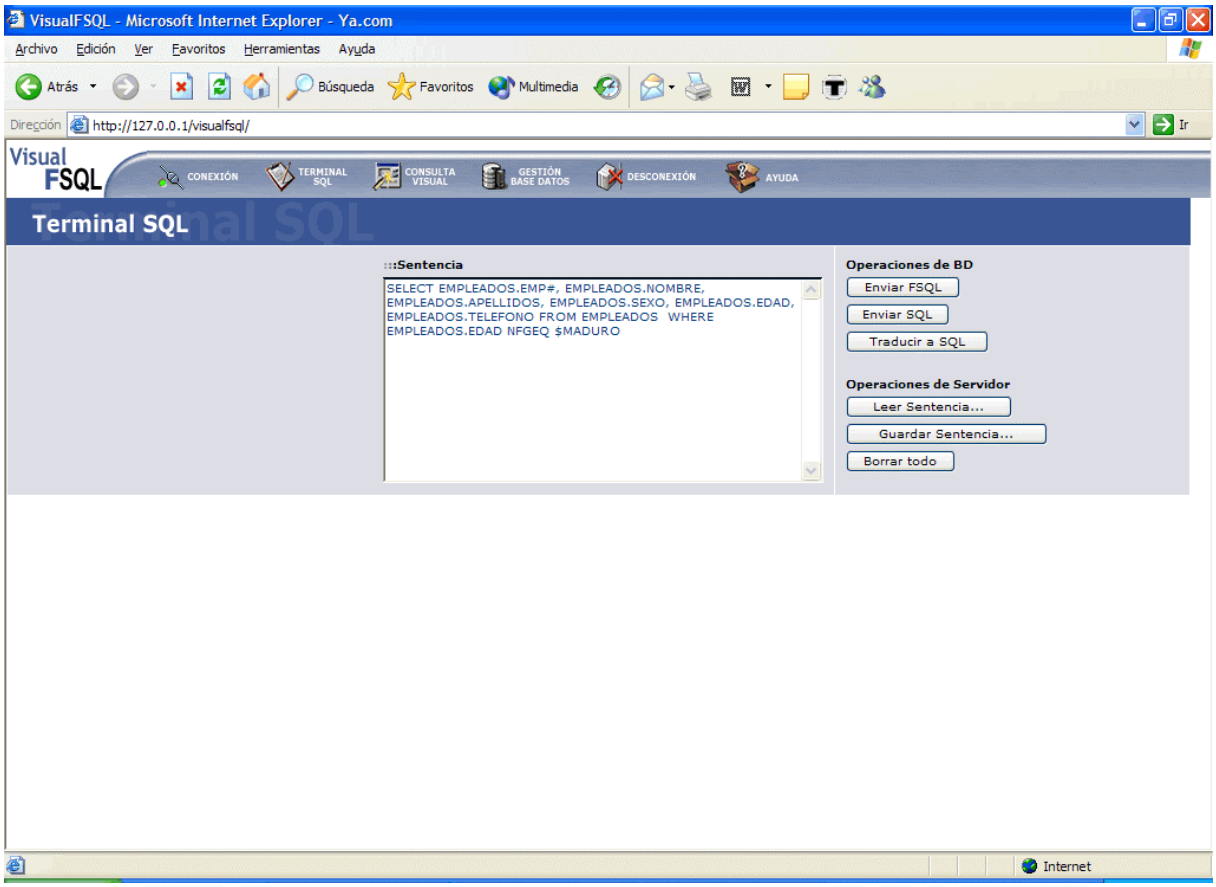


Figura 7.6. Pantalla de Terminal SQL.

En el campo “Sentencia” es donde introducimos la consulta SQL ó FSQL. A la derecha hay 6 botones clasificados en dos grupos:

- OPERACIONES DE BD
- OPERACIONES DE SERVIDOR

OPERACIONES DE BD

- Enviar FSQL: Ejecuta la consulta FSQL y nos devuelve los resultados (Figura 7.7).
- Enviar SQL: Ejecuta la consulta SQL y nos devuelve los resultados.
- Traducir a SQL: La consulta FSQL se envía al sistema, se realizará el proceso de traducción y se nos mostrará otro campo de texto con la consulta SQL equivalente una vez traducida (Figura 7.8).

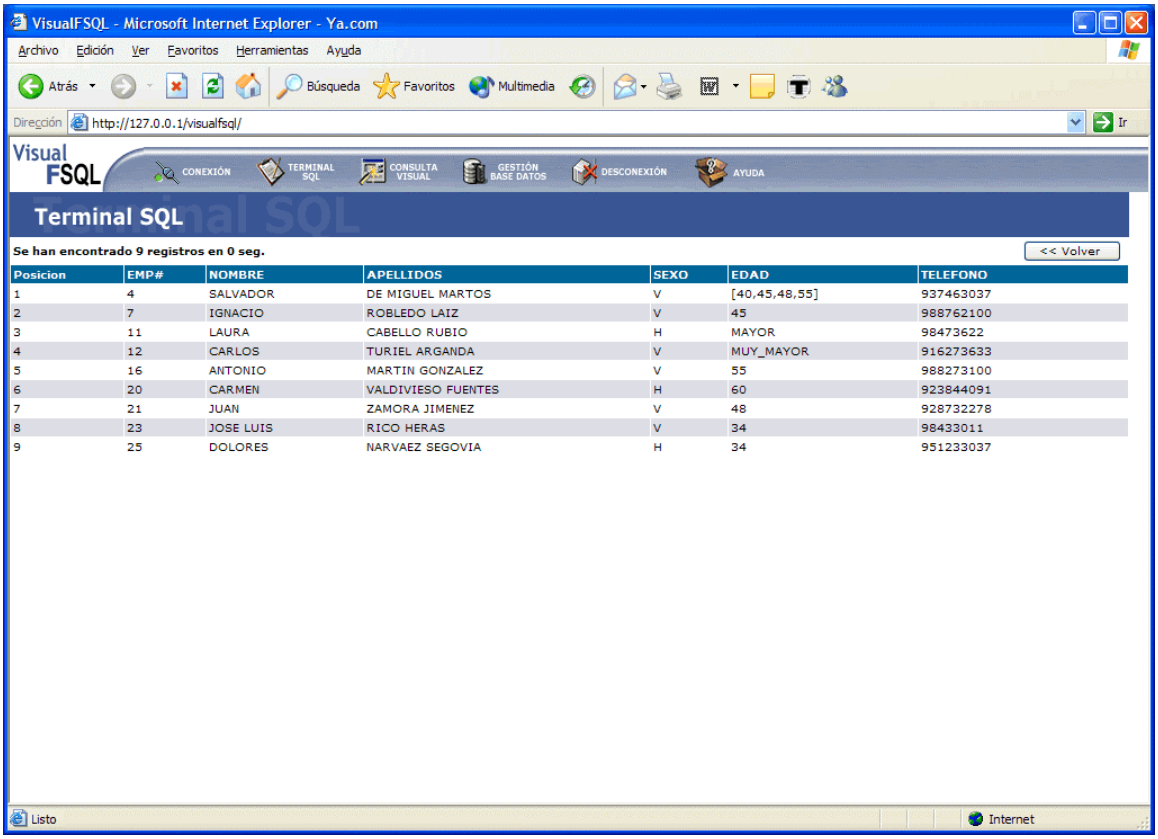


Figura 7.7. Pantalla de Resultados de ejecución de la sentencia.

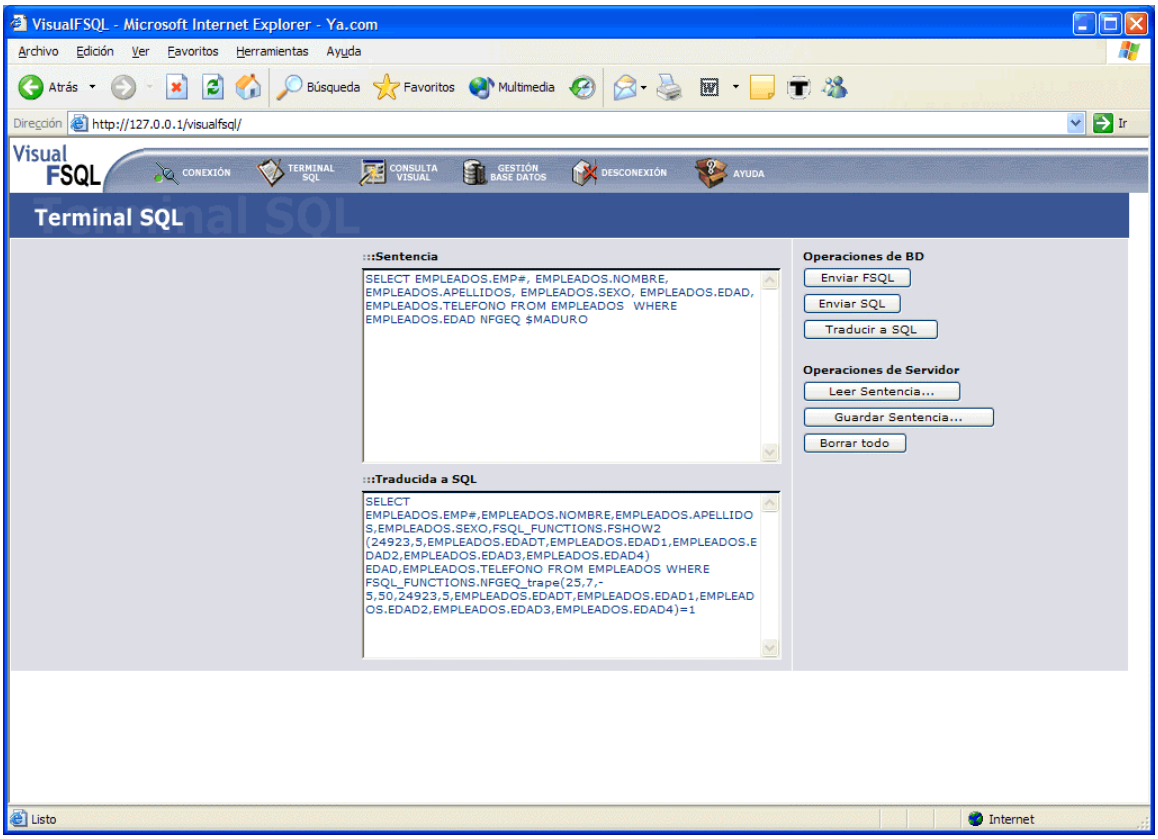


Figura 7.8. Pantalla de Traducción de sentencia FSQL a SQL.

OPERACIONES DE SERVIDOR

- Leer Sentencia...: Se abre una ventana (ver Figura 7.9) con la lista de sentencias almacenadas previamente (Nombre y Fecha de modificación), al pinchar en el nombre se cargará la sentencia en nuestra pantalla de Terminal SQL. Desde esta ventana también podemos borrar una sentencia pinchando en la marca roja de la derecha, se nos pedirá confirmación por seguridad.

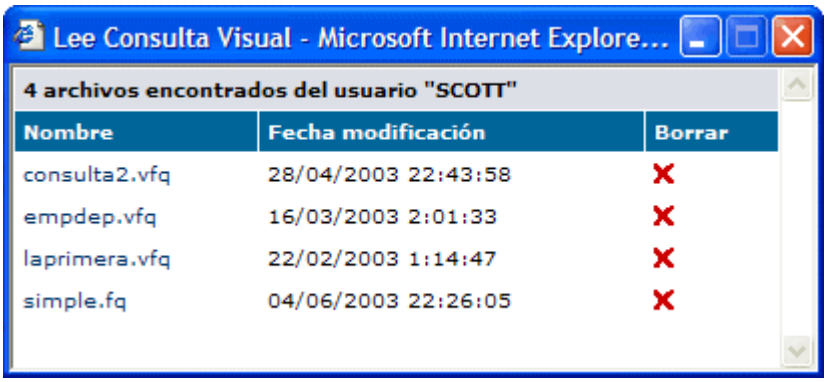


Figura 7.9. Pantalla de Leer Consulta.

- Guardar Sentencia...: Se abre una ventana (Figura 7.10) y en el cuadro de texto superior que aparece pondremos el nombre del archivo donde queremos almacenar la sentencia. También se muestran las consultas previamente almacenadas y además con la posibilidad de borrar como en el punto anterior.

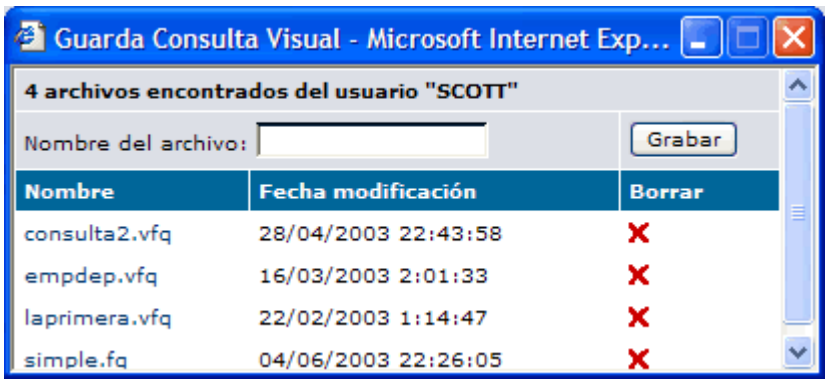


Figura 7.10. Pantalla de Guardar Consulta.

Podemos ver que existen dos tipos de extensiones para los archivos: "**.fq**" (Fuzzy Query: Consulta Difusa) y "**.vfq**" (Visual Fuzzy Query: Consulta Difusa Visual).

- Borrar todo: Limpia el campo "Sentencia" y lo deja en blanco para empezar a escribir una nueva consulta.

Hay que aclarar que estas ventanas de gestión de archivos son distintas en función del usuario con el que nos hayamos conectado al Sistema (en este caso para "SCOTT").

7.4. CONSULTA VISUAL

Esta es la parte principal y más extensa de la aplicación. A través de una serie de sucesivas pantallas consecutivas vamos a ir construyendo mediante asistentes la consulta difusa en FSQL.

En todas las pantallas, para identificar en qué paso nos encontramos, podemos ver un texto en grande y amarillo en la parte superior derecha, indicando el número de paso. Por ejemplo 1/9 (estamos en el primer paso de nueve) y el nombre de la pantalla actual "SELECCIÓN" en este ejemplo (ver Figura 7.11).

También existe una barra de navegación en la parte superior y otra exactamente igual en la inferior, esto es así por el hecho de que existen algunas pantallas muy extensas que necesitan scroll vertical para seguir completando los campos. Entonces para no tenernos que mover de abajo hacia arriba a la hora de navegar a otra pantalla, se ha optado por poner otra barra de navegación en la parte inferior. Los botones que componen esta barra de navegación son (ver Figura 7.11):

- Ayuda: Nos muestra una descripción de la pantalla actual en la que nos encontramos.
- Leer consulta: Se abrirá una ventana como la comentada en el apartado 7.3 (ver Figura 7.9), podremos leer cualquier archivo de consulta visual (con extensión .vfq).
- Cancelar: Cancelamos el proceso de construcción de la consulta Visual y volvemos a empezar desde el principio (previamente nos pedirá confirmación para cancelar el proceso).
- Finalizar: Lanzamos/ejecutamos la consulta Visual en el estado actual tal y como la llevemos construida hasta ese momento, con esto nos ahorramos tener que ir avanzando por todos los pasos.
- Desplegable de pantallas: Seleccionando uno de los pasos y pulsando el botón "Ir a" nos vamos directamente a la pantalla seleccionada.
- Siguiente: Para ir avanzando secuencialmente paso a paso.
- Anterior: Para retroceder a un paso anterior.

A continuación, vamos a detallar cada uno de los pasos de la Consulta Visual.

7.4.1. SELECCIÓN (1/9)

En esta primera pantalla es donde vamos a seleccionar los elementos que queremos mostrar de la consulta (Figura 7.11).

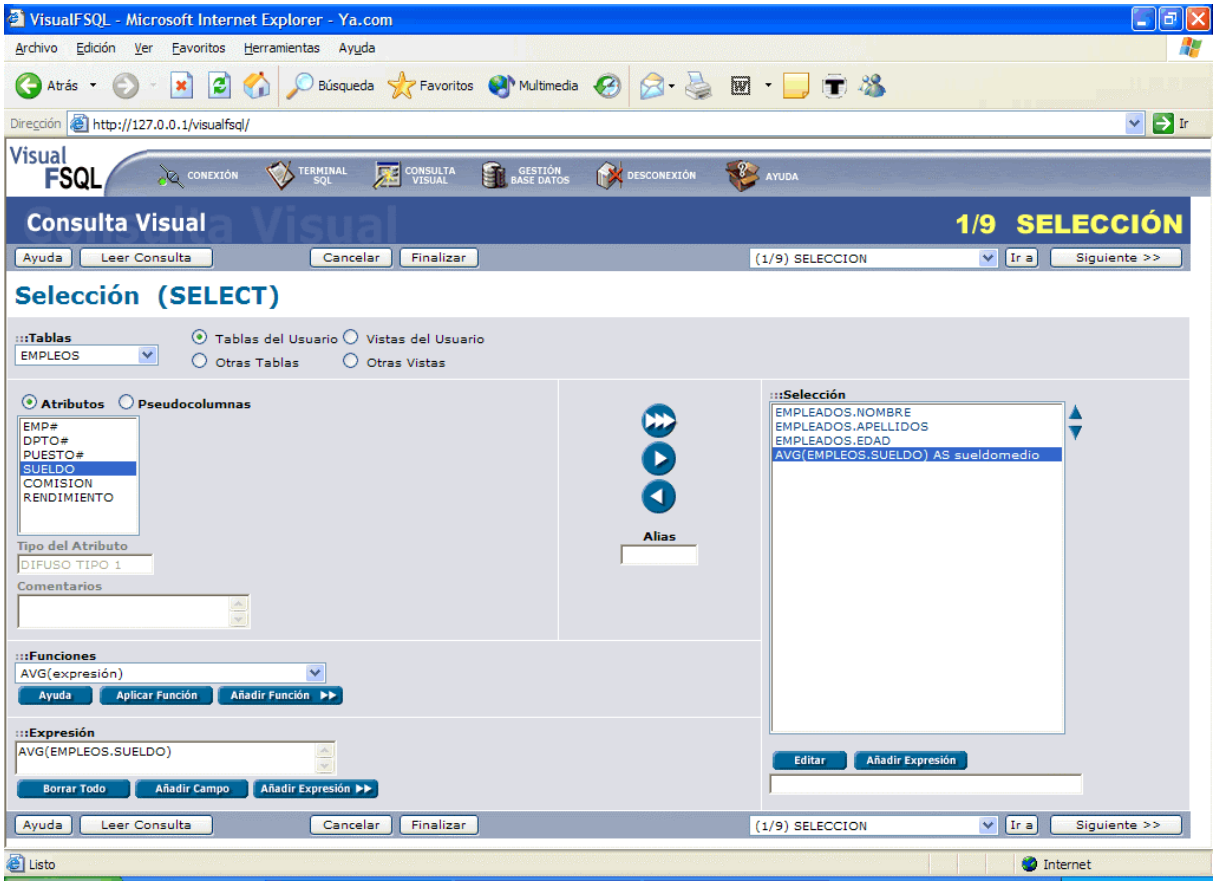


Figura 7.11. Pantalla SELECCIÓN.

Tablas

En la parte superior tenemos una lista desplegable con las Tablas accesibles en función del usuario con el que nos hayamos conectado. Estas tablas las podemos filtrar en función de 4 puntos que tenemos a la derecha de esta lista:

- Tablas del Usuario: Aquellas tablas en las que el usuario es el propietario.
- Vistas del Usuario: Aquellas vistas en las que el usuario es el propietario
- Otras Tablas: Otras tablas accesibles por el usuario.
- Otras Vistas: Otras vistas accesibles por el usuario.

Cada vez que pinchemos en una de estas 4 opciones se cambiarán los elementos de la lista desplegable de Tablas. Hay que indicar que esta carga puede durar varios segundos, puesto que el sistema se conecta al servidor de bases de datos para obtener los nuevos elementos de la lista de "Tablas". Nos aparecerá un mensaje en pantalla cada vez que se dé esta circunstancia.

El siguiente paso consistiría en seleccionar un elemento de la lista de "Tablas", entonces, y tras una breve espera, veremos una lista de atributos de la tabla seleccionada. En la Figura 7.11 se puede ver que se ha seleccionado la tabla EMPLEOS.

Atributos y Pseudocolumnas

Ahora tendremos dos opciones, trabajar con la lista de atributos de la tabla seleccionada (vemos que por defecto aparece esta opción marcada en "Atributos"), o bien, pinchar en la opción "Pseudocolumnas", que cargará una serie de constantes ó pseudocolumnas que podemos utilizar para nuestras consultas.

Con los atributos, al seleccionar alguno de ellos, se nos muestra más abajo el "Tipo del Atributo" y "Comentarios" que es una descripción del atributo añadida por el diseñador de la base de datos y que se encuentra en el diccionario de Datos de Oracle. En la Figura 7.11 se puede observar que el atributo seleccionado SUELDO es Difuso de tipo 3. Los atributos pueden ser de alguno de los siguientes tipos:

- Numérico
- Carácter
- Lógico
- Fecha
- Difuso tipo 1
- Difuso tipo 2
- Difuso tipo 3
- Sin determinar.

Con las pseudocolumnas nos aparece una pequeña ayuda o descripción en la caja "Comentarios" del elemento seleccionado.

A partir de ahora ya podemos ir añadiendo elementos a nuestra caja de "Selección" de la derecha. Esta caja contendrá todos los elementos y expresiones que vamos a agregar a nuestro SELECT.

Para esto debemos seleccionar los elementos de la lista anteriormente comentada y pulsar alguno de los botones circulares que tenemos para agregar y que están situados a la izquierda de la lista "Selección" (para obtener ayuda acerca de cada uno de los botones solo es necesario posicionar el puntero del ratón sobre el botón deseado, entonces se mostrará un texto con una breve descripción de su funcionalidad).

Vamos a detallar el funcionamiento de los tres botones y de la caja de "Alias", situada bajo éstos:

- El primer botón se utiliza para agregar TODOS los elementos de la tabla seleccionada, en este caso se añadiría el elemento *nombretabla.** que hace referencia a todos los atributos de la tabla *nombretabla*.
- El segundo botón sólo agrega los elementos que estén seleccionados en ese momento.
- El tercer botón sirve para quitar de "Selección" los elementos que tengamos marcados en esa lista, puede hacerse individualmente o en grupo.
- Por último la caja "Alias" se utiliza para asignar Alias a los elementos o expresiones seleccionadas, sólo debemos introducir un nombre en la caja de texto y utilizar los botones anteriormente descritos.

Hay que recordar que con con "*ctrl.+ clic de ratón*" ó "*shift + clic de ratón*" es posible marcar/seleccionar varios elementos de cualquier lista.

Funciones

En la parte inferior izquierda tenemos una lista con una gran cantidad de funciones agrupadas por categorías, y que vamos a poder aplicar para construir expresiones más complejas y añadirlas a nuestra consulta SELECT.

Las categorías por las que están agrupadas son:

- Numéricas
- De caracteres
- De fechas
- De conversión
- De agrupación
- Otras
- Lógica Difusa: Incluye las funciones difusas CDEG(atributo) y CDEG(*) para mostrar el grado de cumplimiento para el atributo asignado y para cada tupla.

Vamos a ver el significado de los botones de Funciones:

- "Ayuda": Si seleccionamos alguna función y pulsamos en este botón se nos mostrará una pequeña ventana con información de su funcionalidad.
- "Aplicar Función": Aplicará la Función seleccionada al atributo o elemento marcado y esto lo agregará a la caja inferior de "Expresión" que será donde podemos construir una expresión más compleja para nuestra consulta SELECT. Si la función tuviera varios parámetros utilizaría el atributo seleccionado como primer parámetro, y dejaría a continuación una coma para poder seguir construyendo la expresión con más parámetros.
- "Añadir Función": Coge la función y el atributo seleccionados y los agrega a la parte de "Selección", hay que tener cuidado en elegir una función de un solo parámetro porque el sistema por defecto sólo utiliza el atributo seleccionado como parámetro de la función.

Expresión

En esta caja es donde vamos a poder modificar las expresiones que estemos construyendo, a parte de los botones que tenemos vamos a poder editar y escribir libremente en esta caja.

Vamos a ver los botones implicados en esta parte:

- "Borrar Todo": Borra la caja de expresión y elimina lo que estuviéramos editando.

- "Añadir Campo": Coge el "Atributo" ó "Pseudocolumna" marcado y lo añade a la expresión que estuviéramos editando.
- "Añadir Expresión" pasa la expresión que se está editando a la parte de "Selección" de la derecha.

En la Figura 7.11 se observa que se ha seleccionado el atributo SUELDO y la función AVG, al pulsar en "Aplicar Función" se pasa esta combinación a la caja de "Expresión" donde podemos modificar la expresión a construir. Además en el ejemplo se le ha aplicado un alias "sueldomedio" a esta expresión, que puede verse a la derecha en la caja "Selección".

Selección

Esta es la lista de los elementos y expresiones que se van a incluir en el SELECT, y como hemos visto, está en la parte derecha.

Disponemos de dos botones, con forma de fecha triangulares, situados a la derecha de esta lista y que nos van a servir para subir o bajar el orden de los elementos cuando se vaya a mostrar los resultados de la consulta. Para ello, seleccionaremos un elemento y pulsaremos en el botón superior para posicionar el elemento más al principio o bien el botón inferior para posicionar el elemento más al final.

Además, en caso de que queramos modificar alguno de estos elementos tenemos en la parte inferior de la caja "Selección" un botón "Editar", que cargará en la caja de texto el elemento de la lista que tuviéramos seleccionado con la posibilidad de editarlo en esta caja libremente, y como el elemento original se quita de la "Selección", posteriormente tendremos que añadir la expresión nuevamente a "Selección" mediante el botón "Añadir Expresión".

Una vez concluida la parte del "SELECT" podemos avanzar a la siguiente pantalla pulsando en el botón "Siguiente >>" del Menú de Navegación, o bien seleccionando la pantalla que deseemos en la "lista de pantallas" de la barra de navegación, y pulsando el botón "Ir a".

7.4.2. RELACIONES ENTRE TABLAS (2/9)

En esta segunda pantalla vamos a relacionar las tablas utilizadas en la consulta mediante sus atributos relacionados. Se puede observar en la Figura 7.12 que se dispone de dos partes con la misma funcionalidad, el lado izquierdo y el derecho de la relación.

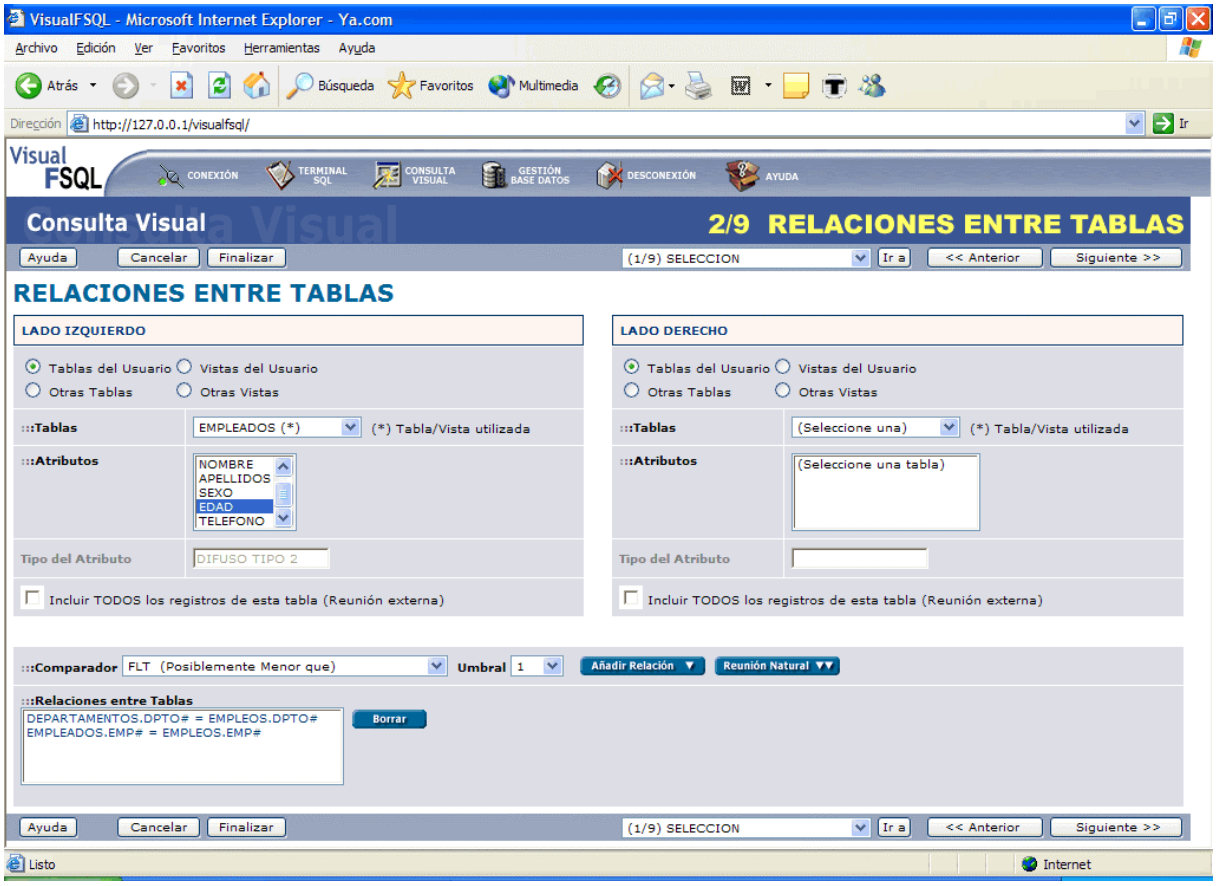


Figura 7.12. Pantalla RELACIONES ENTRE TABLAS.

Lado izquierdo y Lado derecho

La funcionalidad y forma de proceder es exactamente igual para ambos casos. Se puede observar que, al igual que en la pantalla anterior de SELECT, aquí también tenemos 4 posibles opciones para cargar la lista de Tablas: Tablas del Usuario, Vistas del Usuario, Otras Tablas y Otras Vistas. Al pinchar en alguna de estas opciones se cargarán las tablas correspondientes y ahora las ya utilizadas en el SELECT aparecen marcadas con un (*), para así facilitar la búsqueda de aquellas tablas que supuestamente tendremos que relacionar. El

siguiente paso sería marcar alguno de los atributos de la tabla seleccionada, entonces se nos mostrará el Tipo de atributo elegido.

También podemos ver que debajo tenemos una opción para marcar que dice "Incluir TODOS los registros de esta tabla (reunión externa)", esto mostrará todos los registros de la tabla del lado seleccionado aunque dichas filas no se relacionen con ninguna de la tabla del otro lado, esto se conoce comúnmente como "*reunión externa*".

Una vez que hayamos seleccionado un atributo a cada lado, ya podemos marcar el "Comparador" que se le va a aplicar a la relación. Esta lista aparece más abajo. Es importante saber que esta lista de Comparadores no es fija, sino que cambia en función del Tipo de Atributo seleccionado en el lado izquierdo de la relación, puesto que habrá distintos comparadores para distintos tipos de atributos: numérico, carácter, difuso tipo 2, etc... Además, si seleccionamos un atributo de tipo difuso, se activará la lista "Umbral" para seleccionar el THOLD de la relación.

El siguiente paso sería pulsar en el botón "Añadir Relación" y la relación construida se agregaría a la lista de abajo "Relaciones entre Tablas".

Tenemos otra opción muy útil y cómoda que es la del botón "Reunión Natural". En este caso el sistema agrega a la lista de "Relaciones entre Tablas", y con el comparador seleccionado, todas las relaciones formadas entre atributos con el mismo nombre de las tablas seleccionadas a ambos lados.

Existe un último botón "Borrar" que nos permite eliminar de la lista aquella relación que no deseemos incluir.

Una vez concluida la parte de "RELACIONES ENTRE TABLAS" podemos avanzar a la siguiente pantalla pulsando en el botón "Siguiente >>" del Menú de Navegación.

7.4.3. CONDICIONES SIMPLES (3/9)

Esta es una de las pantallas más completas y complejas de todo el asistente de creación de consultas visuales. En ella vamos a establecer las condiciones que debe cumplir la sentencia o consulta que estemos construyendo. Por tanto, es de vital importancia comprender correctamente el funcionamiento de todos los elementos y componentes de esta pantalla para obtener los resultados deseados en nuestra consulta.

Debido a que esta pantalla es bastante extensa se ha dividido en tres Figuras y la vamos a explicar por partes (Figura 7.13, Figura 7.14 y Figura 7.15).

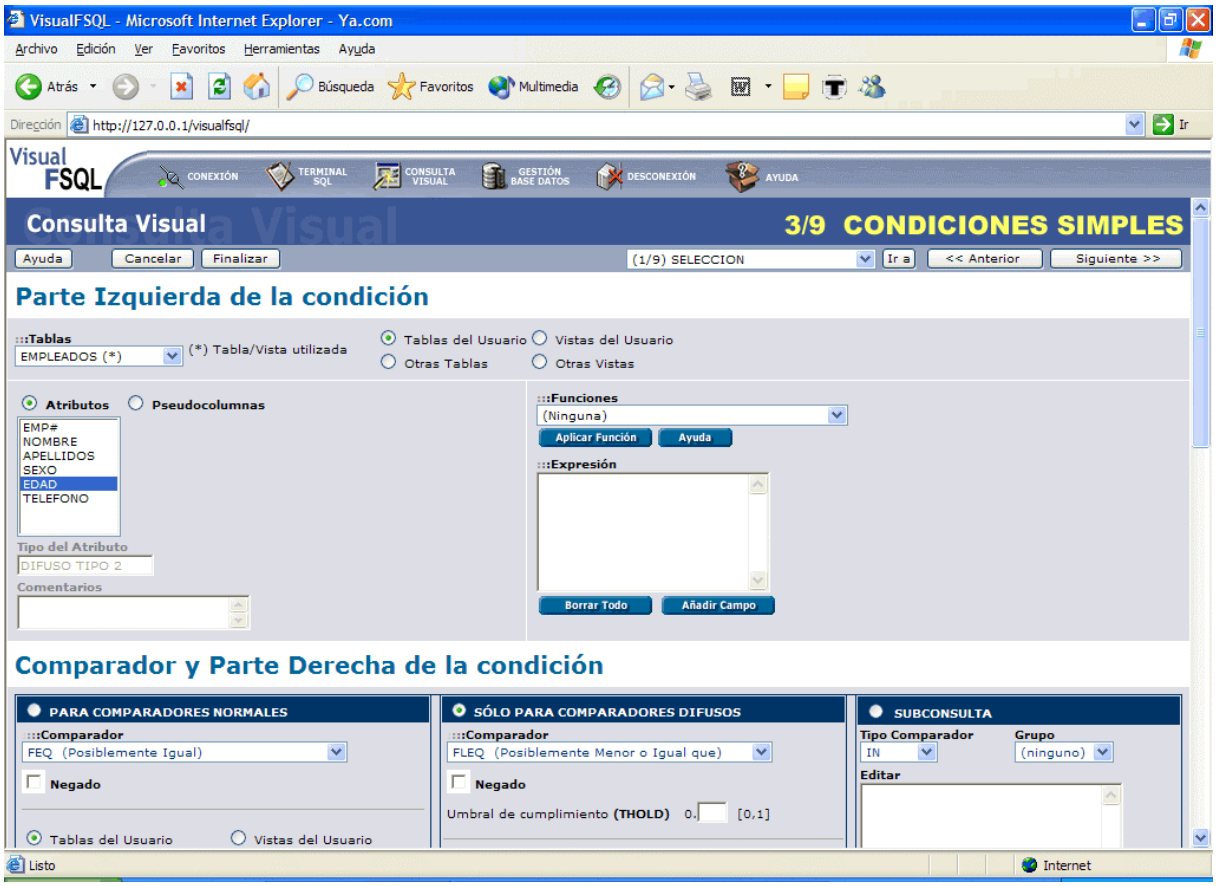


Figura 7.13. Pantalla de CONDICIONES SIMPLES (Parte Izquierda).

Parte Izquierda de la condición

Aquí vamos a tener una serie de elementos (ver Figura 7.13) y cajas de texto muy similares a las anteriormente comentadas en la pantalla de SELECT.

Disponemos de cuatro opciones para cargar una lista de Tablas, y una vez seleccionada una de las Tablas se mostrarán los atributos que contiene, entonces si pinchamos un atributo podemos ver de qué "Tipo" es y los "Comentarios" asociados (si los tiene).

La lista de "Funciones" nos va a servir cuando queramos aplicar una función al atributo seleccionado. Ya vimos como se realiza en la pantalla del SELECT: pulsaremos en "Aplicar Función" y entonces se agregará a la caja de texto "Expresión", donde podremos editar libremente la expresión que estemos construyendo. Si pulsamos en el botón "Ayuda" podemos ver la ayuda en una ventana emergente para la función seleccionada.

Debajo de la "Expresión" tenemos dos botones, el primero "Borrar Todo" elimina la expresión que estemos editando en ese momento, y el botón "Añadir Campo" agrega directamente a la caja de "Expresión" el "Atributo" que tuviéramos seleccionado.

Comparador y Parte Derecha de la condición

Una vez formada la parte izquierda de la condición, en esta zona disponemos de los elementos necesarios para construir la parte derecha (Figura 7.14).

Esta zona se divide en tres bloques:

- Para comparadores normales (parte izquierda).
- Sólo para comparadores difusos (parte central).
- Subconsulta (parte derecha).

En función del atributo marcado en la parte izquierda de la condición (en la lista "Atributos") podremos construir la sentencia de comparación mediante "Comparador y Parte Derecha de la Condición" utilizando alguno de los tres bloques mencionados. Así por

ejemplo cuando seleccionemos un atributo normal (no difuso) podemos utilizar el primer bloque (para comparadores normales) y el tercer bloque (Subconsulta) para construir la comparación. Sin embargo, si seleccionamos un atributo difuso tendremos acceso al primer bloque (para comparadores normales) y al segundo bloque (sólo para comparadores difusos). En la Tabla 7.1 se resumen qué bloques vamos a poder utilizar en cada caso:

Atributo Seleccionado de la parte izquierda	BLOQUES		
		Sólo para comparadores difusos	Subconsulta
(Ninguno)	SI	NO	SI
No Difuso	SI	NO	SI
Difuso	SI	SI	NO

Tabla 7.1. Opciones para la parte derecha de la Condición en función del tipo de atributo.

Si intentamos seleccionar algún bloque no asociado al tipo de atributo seleccionado, el sistema nos mostrará una ventana de advertencia.

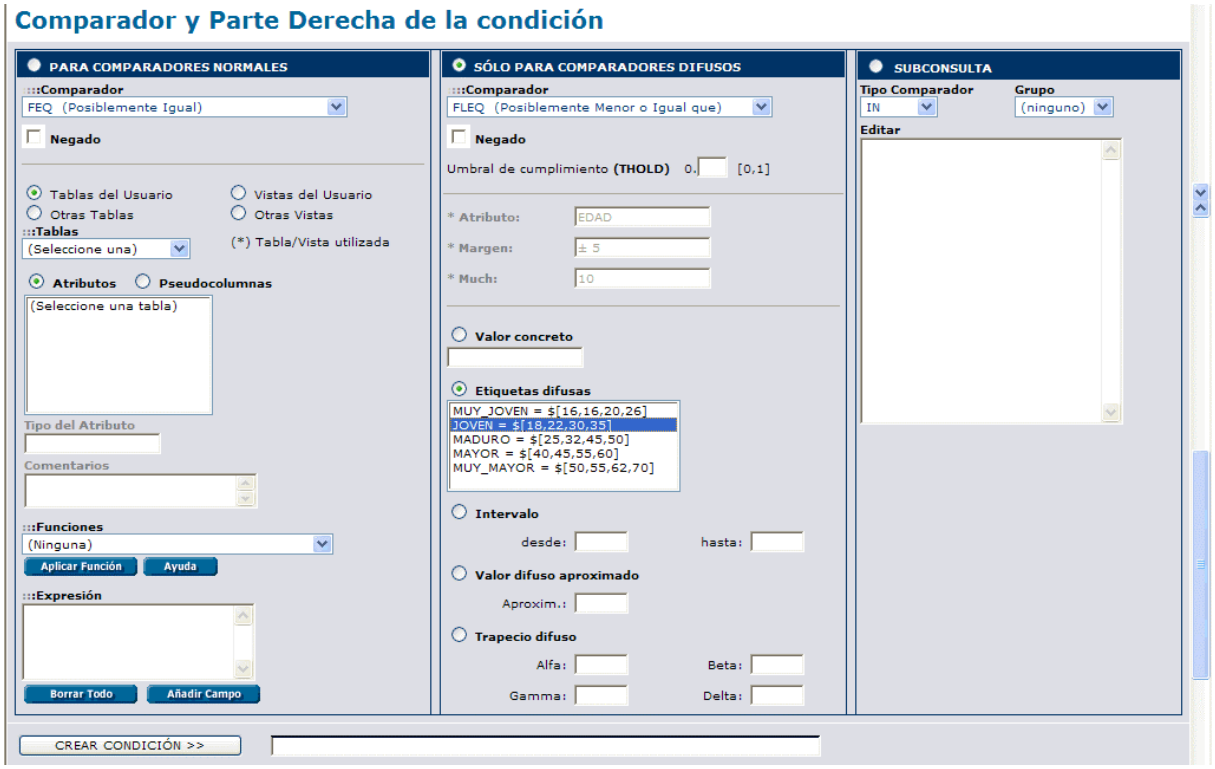


Figura 7.14. Pantalla de CONDICIONES SIMPLES (Comparador y Parte Derecha).

En la Figura 7.14 podemos observar que tenemos seleccionado el segundo bloque (Sólo para comparadores difusos) y tenemos acceso a él puesto que hemos seleccionado el atributo Difuso “EDAD” en la parte izquierda de la comparación.

A continuación vamos a describir la funcionalidad de cada bloque para construir las expresiones de comparación.

PARA COMPARADORES NORMALES:

Tal y como podemos apreciar en la parte izquierda de la Figura 7.14, el funcionamiento de este bloque es exactamente igual que el ya visto en la pantalla del SELECT. Por tanto nos remitimos a esta pantalla para aprender su operatividad. Sólo hay un nuevo elemento, una marca "Negado" para negar (NOT) la condición a construir. Además hay que mencionar que en la lista "Comparador" van cambiando los comparadores en función del tipo de atributo seleccionado en la parte izquierda.

SÓLO PARA COMPARADORES DIFUSOS:

En la parte central de la Figura 7.14 podemos ver una lista con los posibles comparadores difusos que podemos utilizar además de la marca de negado comentada anteriormente en el primer bloque. También existe una caja de texto para introducir el Umbral de cumplimiento (THOLD) de la condición, que está en el rango [0,1].

A continuación vemos información del tributo seleccionado en la parte izquierda, y que nos puede servir como referencia para asignar los valores difusos de comparación. La información que se muestra es: Nombre del Atributo, valor de "Margen" y valor de "Much".

Más abajo observamos las posibles opciones para comparar nuestro atributo difuso, podemos utilizar uno de los siguientes elementos:

- Valor concreto: Introducimos un valor ó constante.
- Etiqueta difusa: En función del atributo seleccionado vemos que se cargan las etiquetas lingüísticas que tuviera definidas en el sistema. En el ejemplo de la la Figura 7.14, para el atributo EDAD se muestran sus etiquetas asociadas (MUY_JOVEN, JOVEN, MADURO, MAYOR, MUY_MAYOR). Además detrás de cada etiqueta se muestran los 4 valores del trapecio que la definen.

- Intervalo: Se utiliza el tipo intervalo de valores, “desde” y “hasta”.
- Valor difuso aproximado: Para comparaciones de aproximación.
- Trapecio Difuso: Se introducen los cuatro valores que definen el trapecio de la distribución de posibilidad trapezoidal (Alfa, Beta, Gamma y Delta).

SUBCONSULTA:

En este caso podemos construir una subconsulta SQL ó FSQL que se utilizará para comparar con nuestro atributo. Los elementos de que consta son los siguientes:

- Tipo de Comparador: Las opciones son IN, NOT IN, =, !=, ^=, <>, >, <, >= y <=.
- Grupo: Las opciones para comparar con un grupo de registros de la subconsulta son ANY/SOME y ALL.
- Editar: Es la caja de texto donde escribiremos nuestra subconsulta.

Una vez que hemos seleccionado la Parte Izquierda, el Comparador y la Parte derecha de la condición tendremos que pulsar en el botón “CREAR CONDICIÓN >>” para construir la sentencia de comparación. Entonces se mostrará la sentencia en la caja de texto de la derecha del botón.

Lista de Condiciones

En esta zona es donde se van a ir agregando las condiciones de nuestra consulta (ver Figura 7.15). Una vez que hemos verificado que la condición construida es correcta (se puede ver en el campo de texto que hay a la derecha del botón “Crear Condición >>”) pulsaremos el botón “Añadir Condición” para agregarla a nuestra lista “Condiciones”.

A la derecha de esta lista “Condiciones” hay dos botones en forma de flecha hacia arriba y flecha hacia abajo, si pulsamos respectivamente en uno o en otro una vez seleccionada una condición, modificaremos su prioridad sobre las otras subiéndola o bajándola. El botón de “Borrar” eliminará aquellas condiciones que tengamos seleccionadas en la lista.

Por último, podemos editar libremente una condición de la lista. Aquella que esté seleccionada la podemos pasar a la caja de texto que hay que la parte inferior pulsando el botón "Editar". Entonces éste se pondrá en color azul claro para indicar que estamos editando el elemento, y cuando queramos volver a añadir la condición a la lista pulsaremos en el botón "Actualizar".

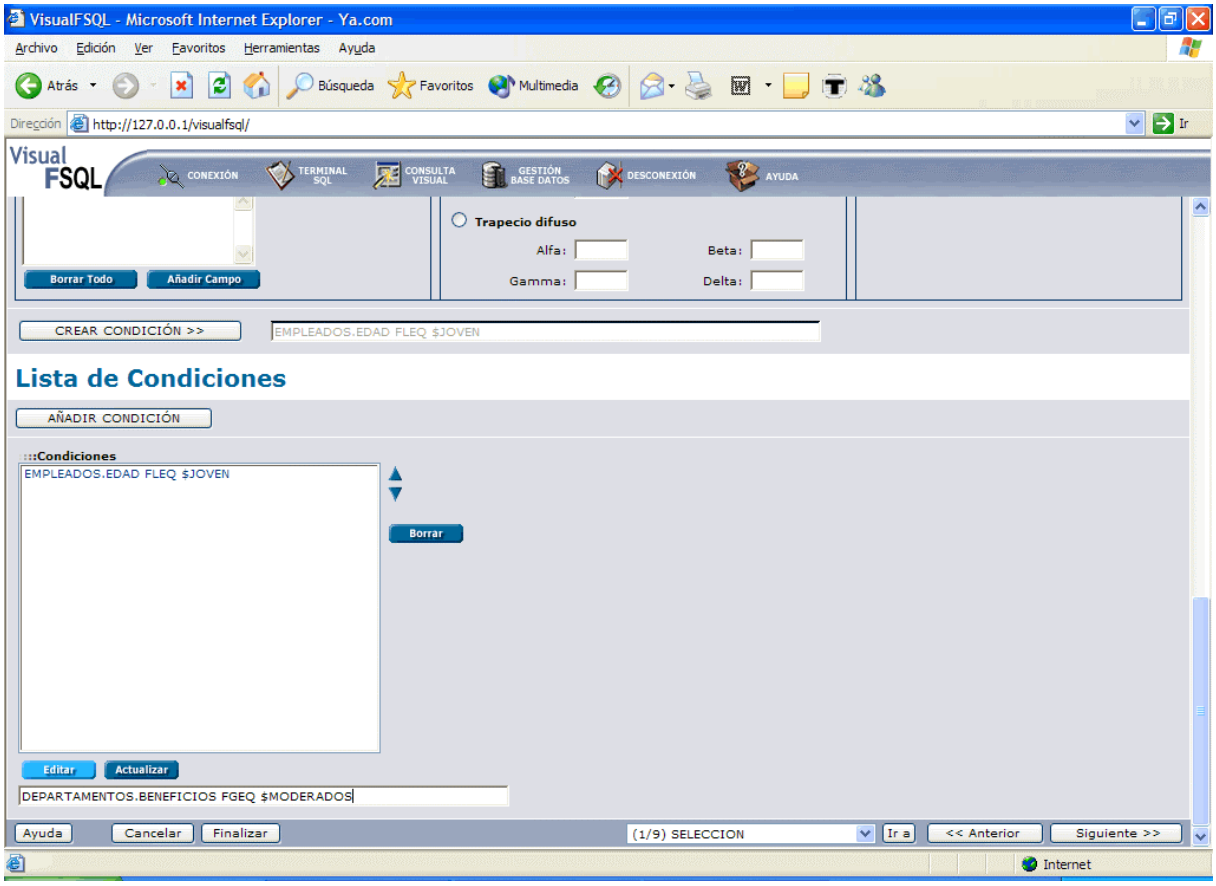


Figura 7.15. Pantalla de CONDICIONES SIMPLES (Lista de Condiciones).

Una vez concluida la parte de "CONDICIONES SIMPLES" podemos avanzar a la siguiente pantalla pulsando en el botón "Siguiente >>" del Menú de Navegación.

7.4.4. ORGANIZA CONDICIONES (4/9)

Ahora vamos a organizar y agrupar las condiciones construidas en la pantalla anterior. La concatenación de las condiciones la realizaremos con los operadores AND y OR.

Las condiciones las podemos agrupar utilizando paréntesis que seleccionaremos de los desplegables laterales, también vemos a la izquierda la lista para elegir el operador "OR" o "AND" y así ir uniendo las condiciones para formar una única sentencia con todas las condiciones agrupadas, para ello pulsaremos en el botón "ORGANIZAR CONDICIONES >>" y entonces se construirá en la caja "Condiciones Organizadas".

En la Figura 7.16 se puede observar un ejemplo con dos condiciones y el operador "AND".

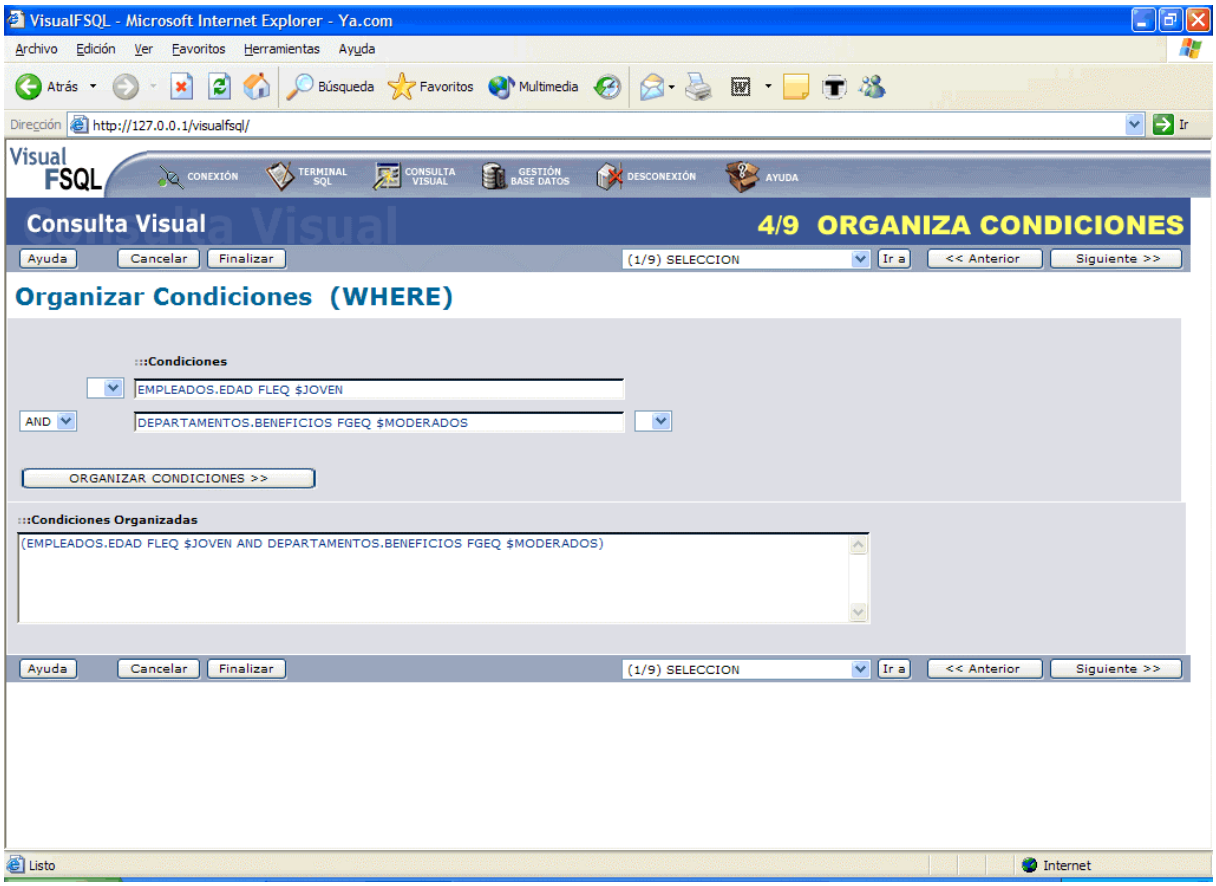


Figura 7.16. Pantalla ORGANIZA CONDICIONES.

7.4.5. AGRUPA RESULTADOS (5/9)

Si queremos agrupar los resultados de la consulta podemos utilizar esta pantalla para construir la sentencia de agrupación, se corresponde con las cláusulas "GROUP BY" y "HAVING" de SQL (ver Figura 7.17).

Crear Agrupaciones (GROUP BY)

En la parte superior existe una lista "Selección" con los atributos y expresiones que elegimos en la primera pantalla del SELECT. A la derecha tenemos otra lista con los elementos por los que queremos agrupar, llamada "Elementos de Agrupación". La forma de proceder es seleccionando los elementos deseados de la lista "Selección", pulsando el botón con la flecha a la derecha, y entonces esos elementos pasarán a formar parte de los elementos de agrupación. Si queremos quitar alguno de estos elementos lo seleccionaremos en la lista y pulsaremos el botón con flecha hacia la izquierda.

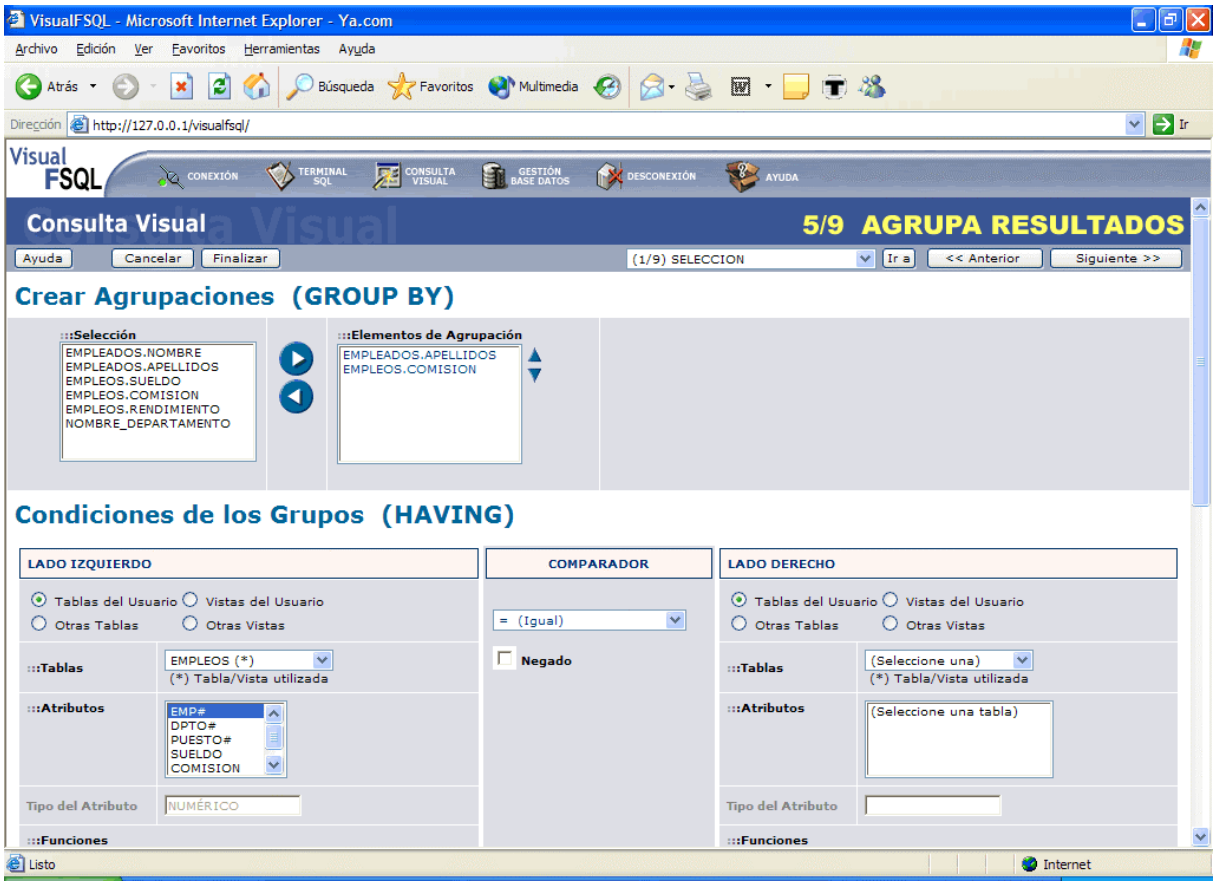


Figura 7.17. Pantalla AGRUPA RESULTADOS (1/3).

Condiciones de los grupos (HAVING)

Podemos igualmente crear condiciones para las agrupaciones formadas con el "GROUP BY". Para ello podemos utilizar la zona "Lista de Condiciones (Having)". Como se puede

apreciar en la Figura 7.17 y en la Figura 7.18, el esquema es básicamente el mismo que el de otras pantallas y por tanto su funcionalidad ya debe ser totalmente intuitiva. Seleccionaremos a ambos lados de la comparación los atributos o expresiones (formadas con funciones o escritas libremente) y le aplicaremos un comparador de la lista central. Se puede negar la comparación marcando la casilla "Negado". Posteriormente pulsaremos el botón "CREAR CONDICIÓN >>" para construir la sentencia de comparación.

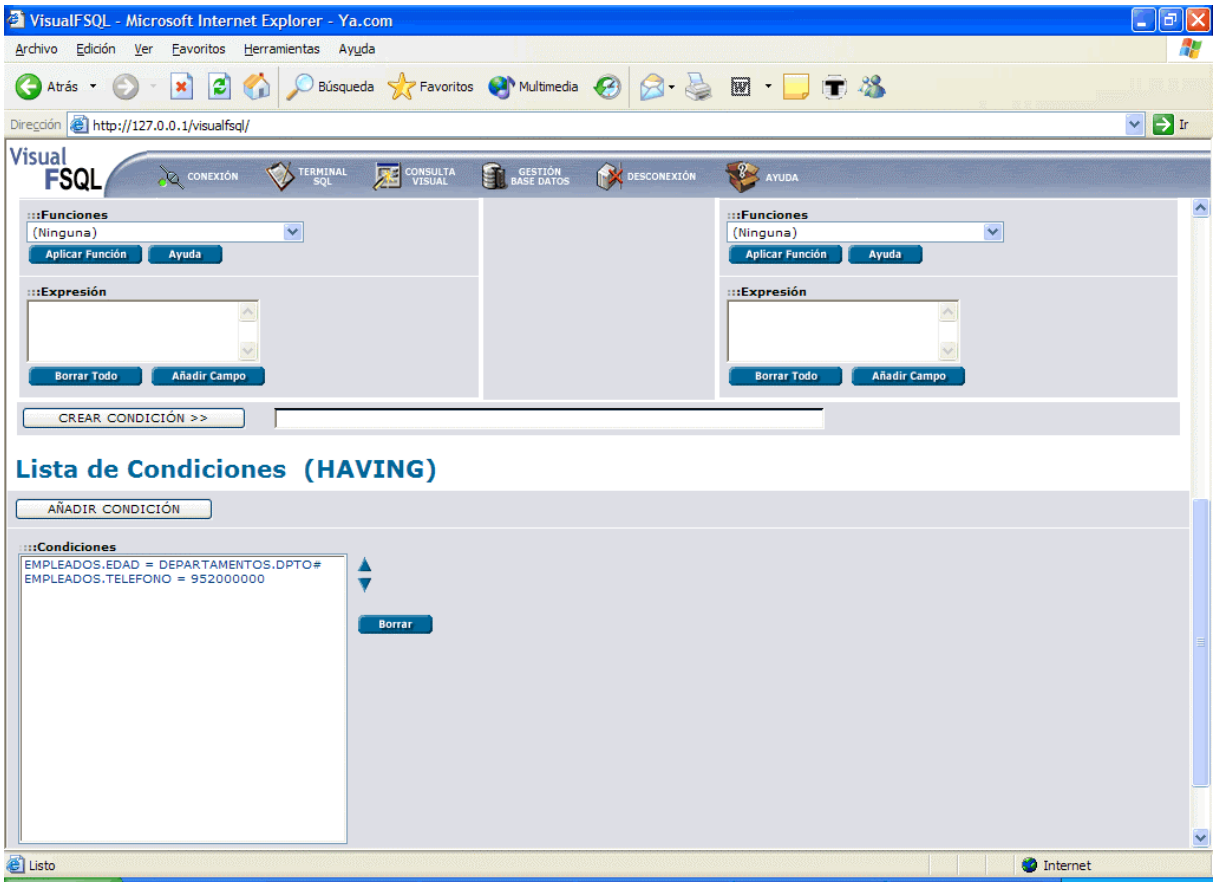


Figura 7.18. Pantalla AGRUPA RESULTADOS (2/3).

Lista de Condiciones (Having)

Al final de la pantalla tendremos la lista de Condiciones construidas, para pasar una condición a esta lista pulsaremos en el botón "AÑADIR CONDICIÓN" (ver Figura 7.18). La forma de gestionar es igual que las otras listas ya comentadas anteriormente, es decir tenemos el botón para ordenar los elementos, el botón para borrar y más abajo vemos los botones para editar y actualizar cualquier condición de la lista (ver Figura 7.19).

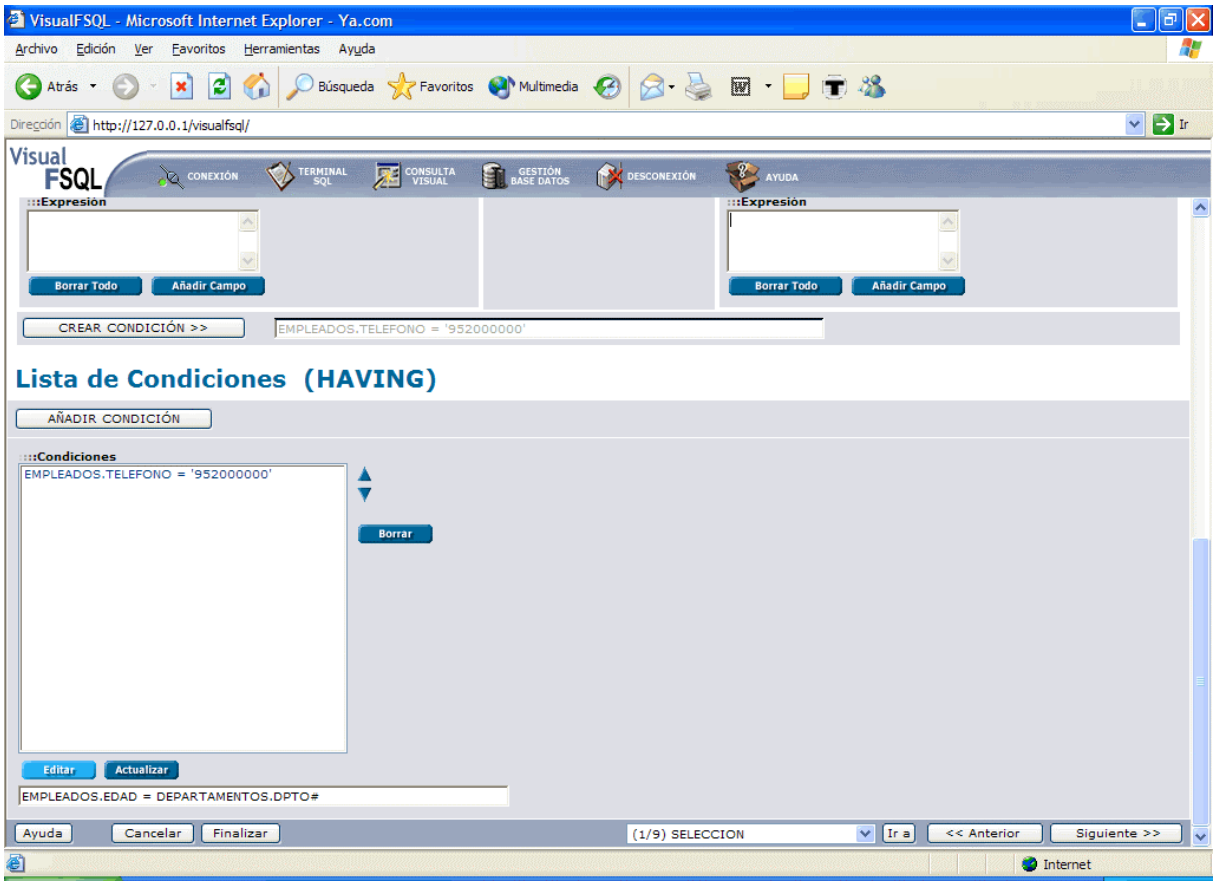


Figura 7.19. Pantalla AGRUPA RESULTADOS (3/3).

7.4.6. ORGANIZA AGRUPACIONES (6/9)

El objetivo de esta pantalla (Figura 7.20), es organizar y agrupar las condiciones del HAVING construidas en la pantalla anterior. La funcionalidad es exactamente igual que la ya comentada para la pantalla "ORGANIZA CONDICIONES" (ver apartado 7.4.4).

Podemos concatenar las condiciones mediante los operadores "AND" y "OR" y además tendremos los paréntesis para establecer prioridades de agrupación. Para construir la sentencia resultante pulsaremos en el botón "ORGANIZAR AGRUPACIONES >>" y entonces veremos el resultado.

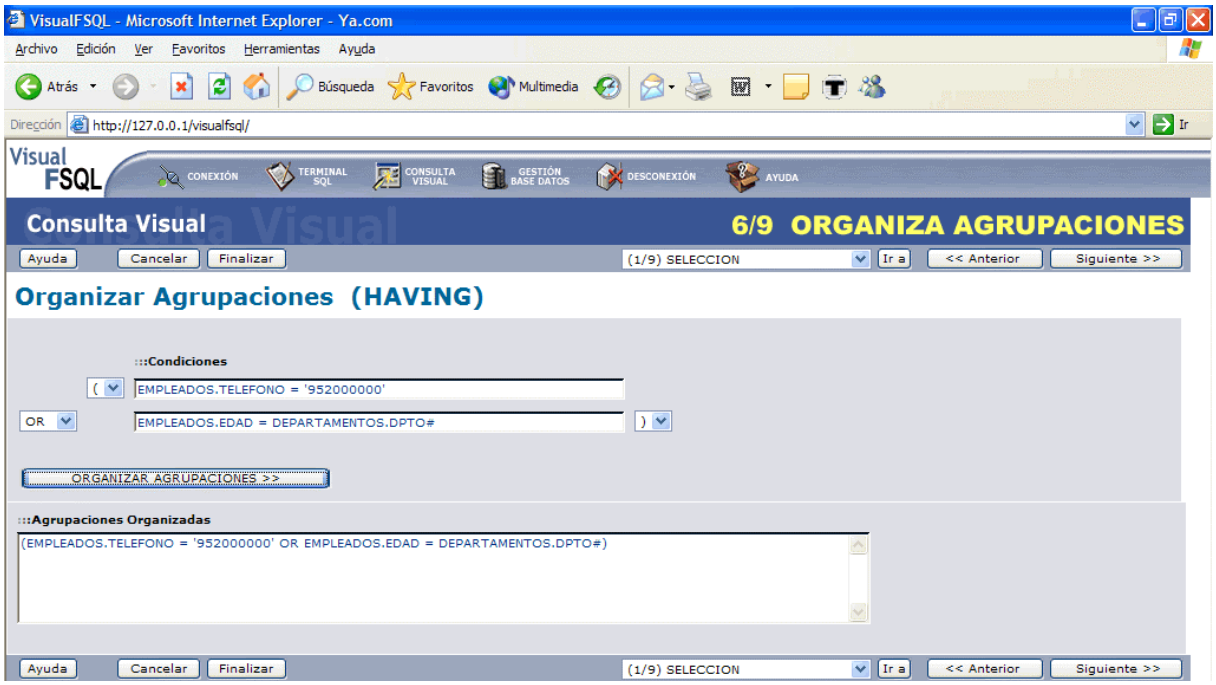


Figura 7.20. Pantalla ORGANIZA AGRUPACIONES.

7.4.7. ORDENA RESULTADOS 7/9

Esta es la última pantalla propiamente dicha del asistente visual para construir la consulta difusa. Aún nos quedarán dos pantallas, pero son adicionales al asistente.

Aquí vamos a poder establecer los criterios de ordenación de los resultados de la ejecución de la consulta, que se corresponde con la cláusula "ORDER BY".

Podemos observar en la Figura 7.21, que disponemos de tres zonas, a la izquierda tenemos la lista "Selección" con los atributos y expresiones seleccionados en el "SELECT", en la parte central los botones circulares para agregar o quitar elementos de la ordenación,, y una opción para establecer el tipo de ordenación (ascendente o descendente). En la parte derecha están los "Elementos de Ordenación" que se van a agregar. Para ello seleccionaremos el elemento de la parte izquierda, pulsaremos en el botón con una flecha hacia la derecha y estableceremos el tipo de ordenación (Ascendente o Descendente) y entonces veremos que dicho elemento pasa a formar parte de la lista "Elementos de Ordenación". Para quitar un elemento de esta lista solo debemos seleccionarlo y pulsar en el botón con la flecha hacia la izquierda.

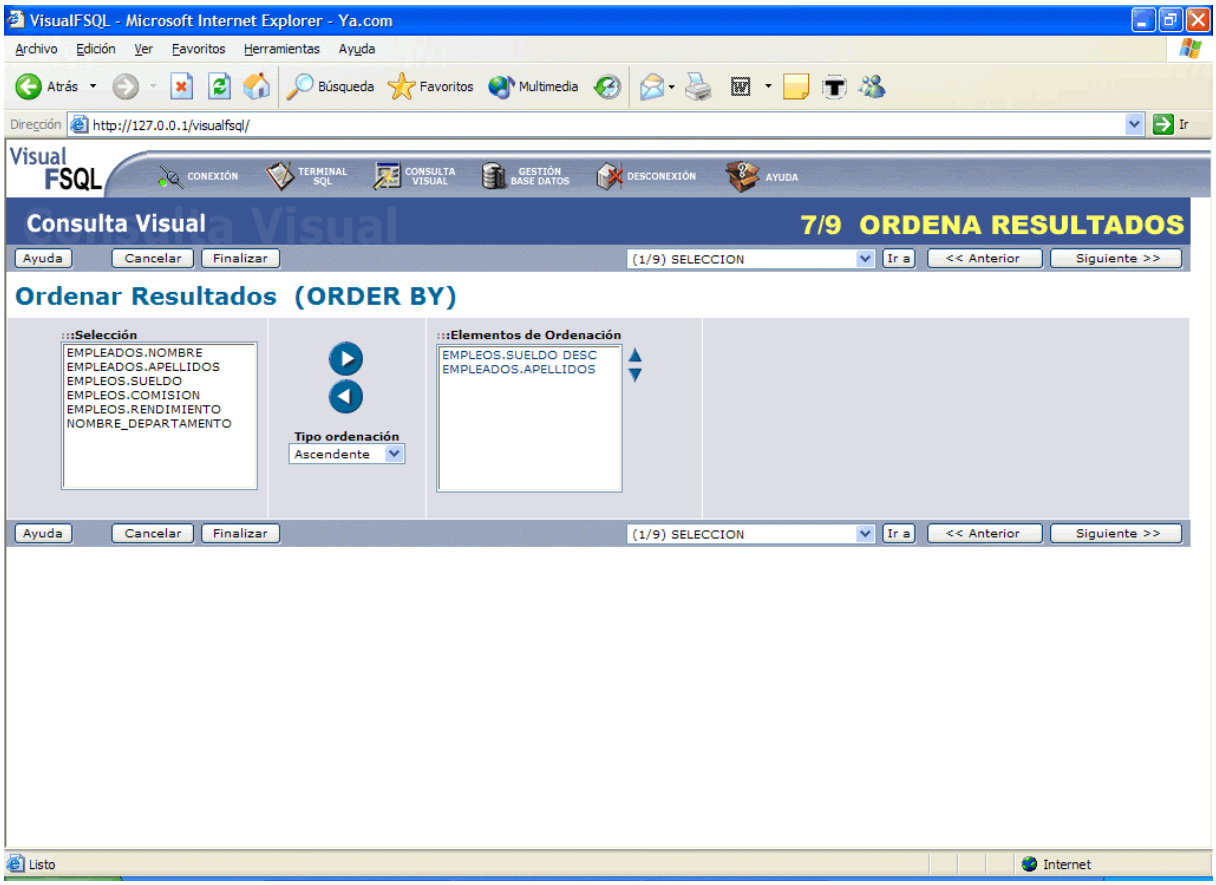


Figura 7.21. Pantalla ORDENA RESULTADOS.

7.4.8. FINALIZAR (8/9)

Esta es la pantalla (Figura 7.22) donde se muestran los resultados de ejecutar la consulta construida. Podemos ejecutar la consulta que estemos construyendo en cualquier momento. En la "Barra de Navegación" disponemos del botón "Finalizar" que ejecutará la sentencia y nos mostrará los resultados en esta pantalla.

En la parte superior, debajo de la "Barra de Navegación" se muestra el número de registros o tuplas obtenidas, y el tiempo empleado en ejecutar la consulta (en segundos). Más abajo vemos una cabecera con el nombre de los campos, y la lista de registros. También existe una columna inicial que nos indica el número de cada fila.

Si la lista de campos es muy extensa o el número de registros muy grande podemos desplazarnos mediante las barras de scroll.

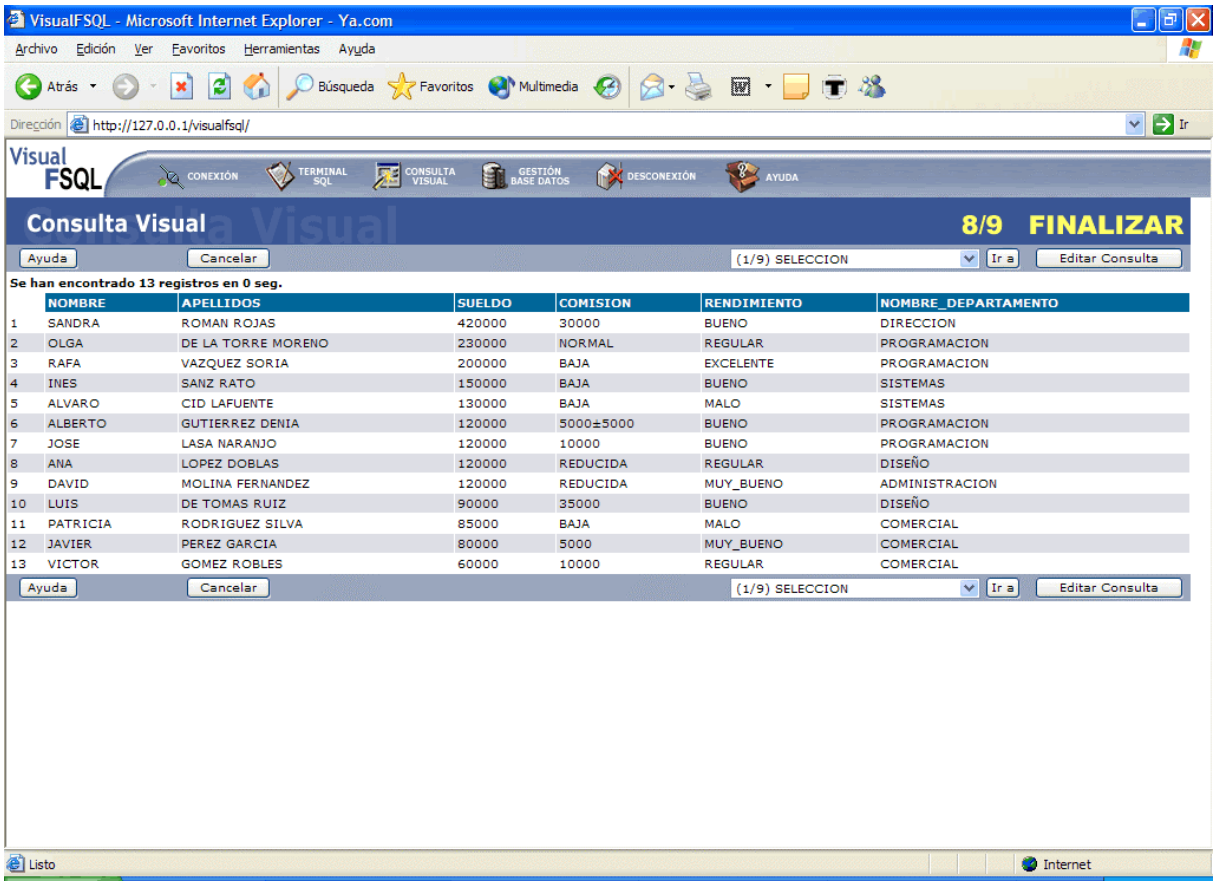


Figura 7.22. Pantalla FINALIZAR.

En esta pantalla podemos observar que la "Barra de Navegación" ha cambiado, ahora no aparece el botón "Finalizar" puesto que no tiene sentido aquí, tampoco están los botones de avanzar y retroceder ("Siguiente >>" y "<< Anterior"), pero si se ha incorporado un nuevo botón "Editar Consulta" que nos llevará a la siguiente y última pantalla de la Consulta Visual.

Si por cualquier circunstancia la ejecución de la consulta diera un error en el SGBD, se mostraría el código de Error y su descripción, estos valores se capturan directamente del Sistema Oracle o del Servidor FSQL.

7.4.9. EDITA CONSULTA (9/9)

Esta pantalla (Figura 7.23) es muy similar a la comentada en el apartado 7.3 "Terminal SQL", por tanto nos remitimos a esta sección para comprender su funcionamiento.

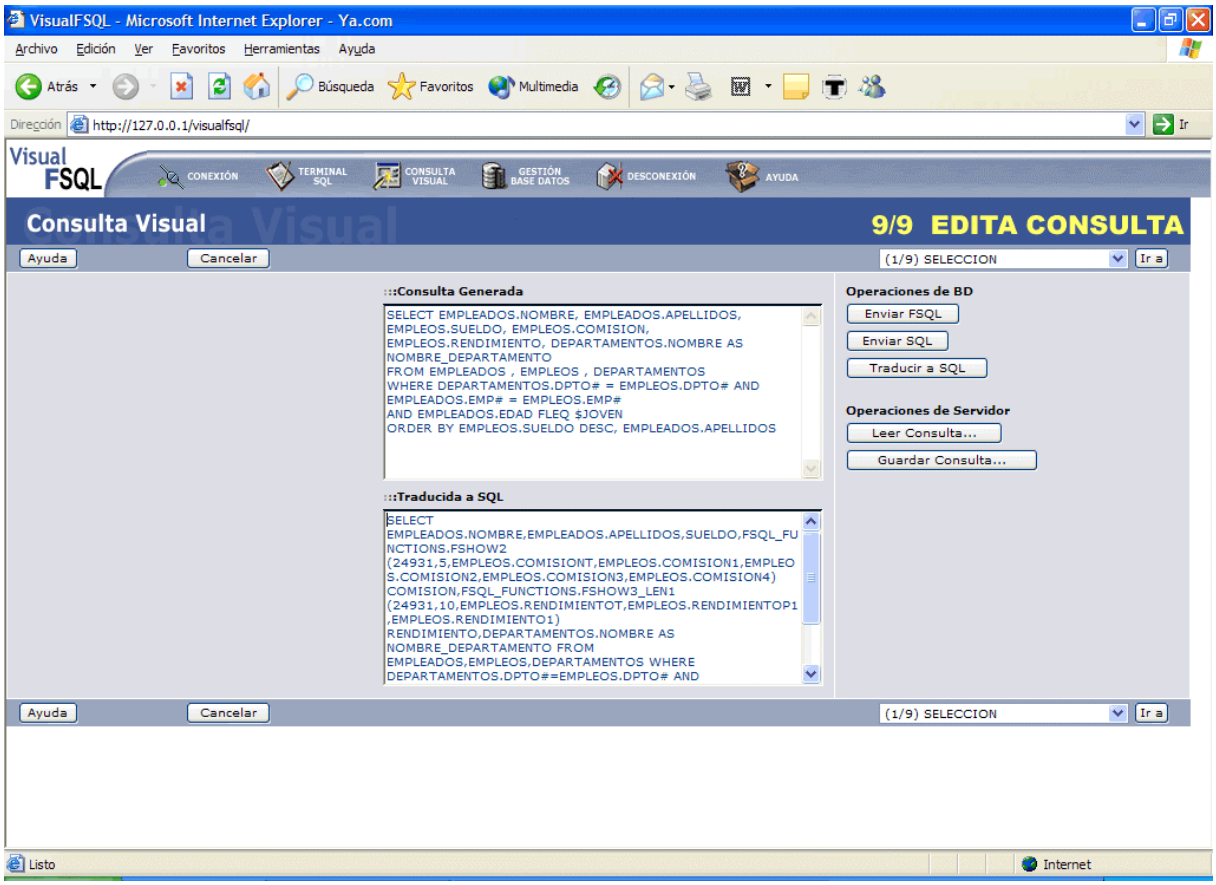


Figura 7.23. Pantalla EDITA CONSULTA.

7.5. GESTIÓN BASE DE DATOS

Podemos acceder a esta gestión a través de la opción que tenemos en la Barra de Menús superior, a la derecha de “Consulta Visual”.

Con esta gestión online vamos a poder administrar los registros de las tablas existentes en nuestra base de datos, las operaciones disponibles son: dar de alta, consultar, modificar y borrar. Es una gestión muy simplificada y sencilla y únicamente se ha incorporado como un módulo adicional al desarrollo del presente Proyecto.

Vamos a comentar breve y resumidamente su funcionamiento.

Selección de Tabla

Es la primera pantalla que aparece (Figura 7.24), aquí seleccionaremos la tabla con la que queremos trabajar y pulsaremos en alguna de las dos opciones disponibles:

- Consultar: Para consultar, modificar o borrar registros.
- Insertar: Para añadir un nuevo registro.

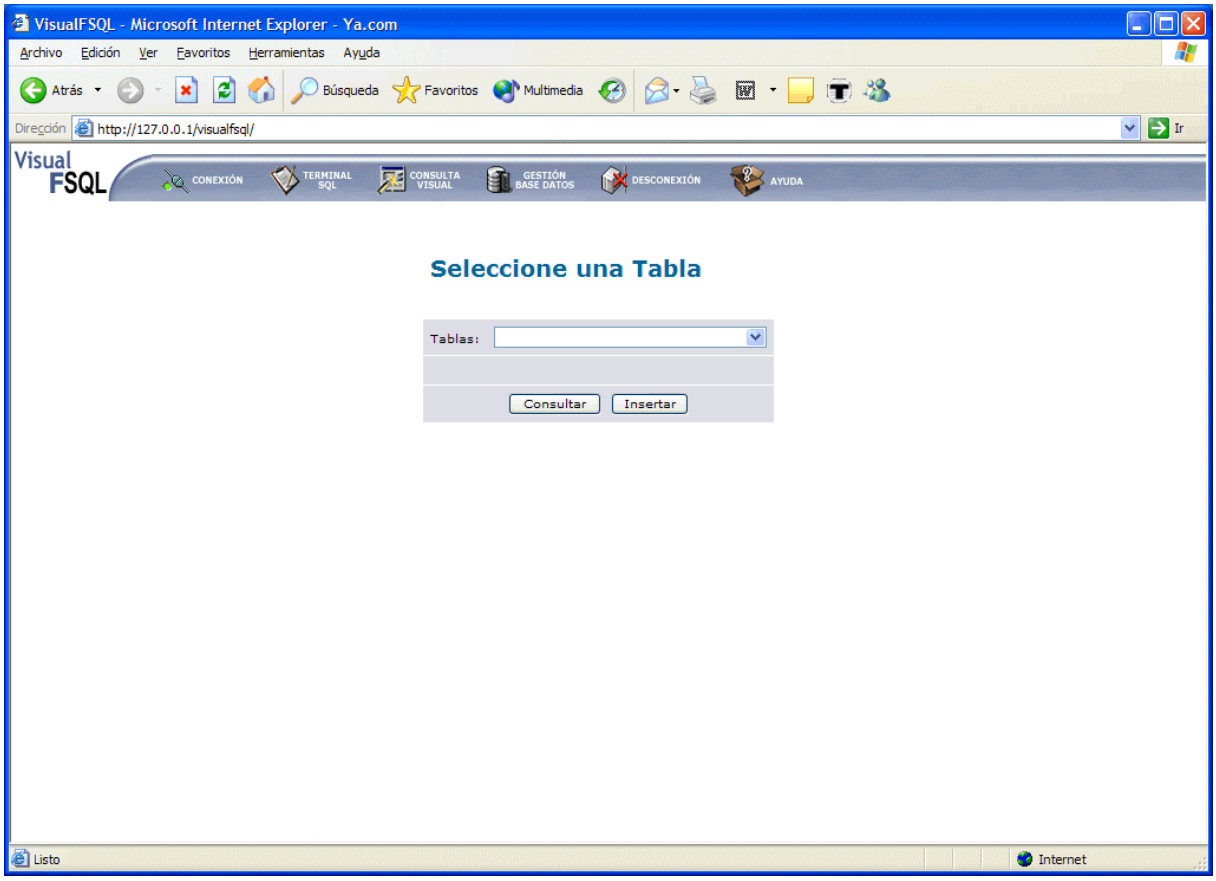


Figura 7.24. Pantalla Selección de Tabla.

Búsqueda

Si pulsamos en Consultar nos aparecerá una pantalla de búsqueda de registros de la tabla seleccionada. Esta pantalla (Figura 7.25) mostrará tantas cajas de texto como campos tenga la tabla seleccionada, entonces rellenaremos aquellos campos por los que deseamos buscar los registros o bien dejamos todo en blanco para obtener la lista completa de registros.

Con el botón “Buscar” se procede a realizar la búsqueda, “Limpiar” borrará el contenido de todas las cajas de texto y con “Volver” retrocederemos a la pantalla anterior.

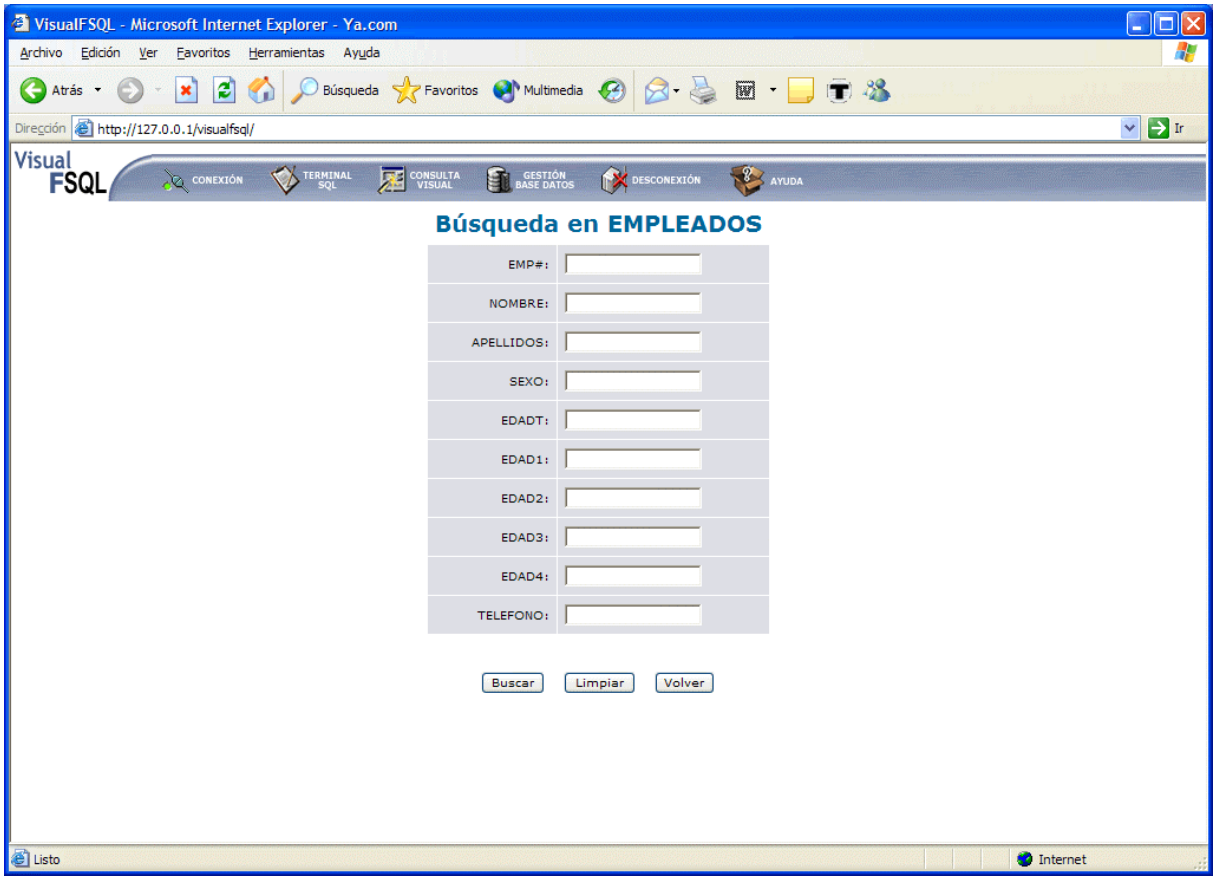


Figura 7.25. Pantalla Búsqueda en la Tabla.

Resultados de la Búsqueda

En esta pantalla (Figura 7.26), se nos mostrarán todos los registros encontrados con los parámetros de búsqueda seleccionados en la pantalla anterior.

Tendremos en la parte superior el nombre de la tabla (por ejemplo EMPLEADOS), a continuación el número de registros encontrados y lo siguiente serían las cabeceras de columnas con los nombres de los campos para dar paso a la lista de registros. A la derecha tenemos la columna “Borrar”, que es un campo de marca para seleccionar aquellos registros que deseemos borrar, una vez marcados, pulsaremos el botón inferior de “Borrar”, y entonces se eliminarán de la tabla todos los registros que hubiéramos seleccionado.

El siguiente paso sería la consulta de los registros, para ello únicamente nos posicionaremos con el ratón encima de cualquier valor de un registro y pulsaremos. Esto nos llevará a la pantalla de Consulta/Edición.

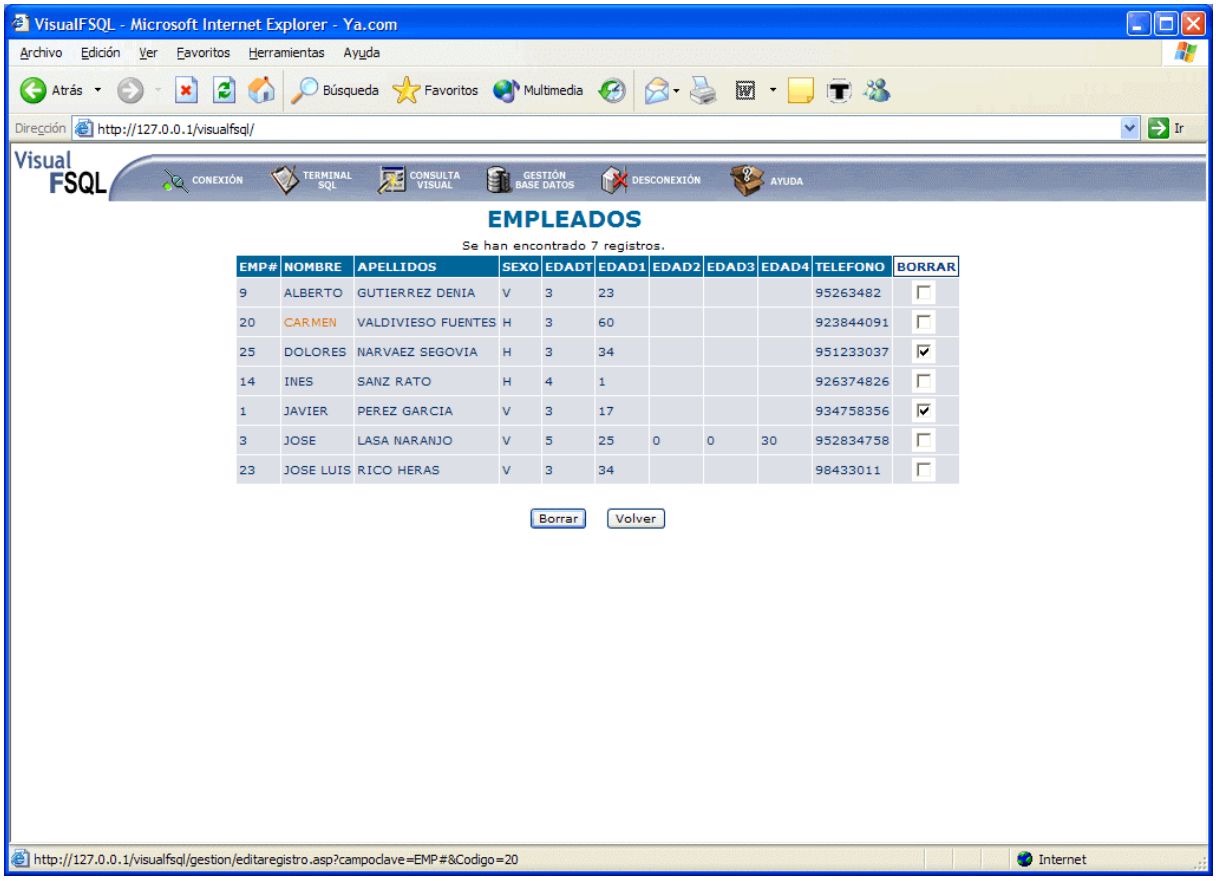


Figura 7.26. Pantalla Resultados de búsqueda en la Tabla.

Consulta/Edición

En esta pantalla (Figura 7.27), se mostrarán los campos del registro seleccionado y podremos consultar o modificar el valor de cualquier campo.

Una vez cambiado cualquier valor pulsaremos el botón “Grabar” y las modificaciones se almacenarán en la Base de Datos, el botón “Limpiar” sirve para restablecer los valores originales de los campos y con “Volver” retrocederíamos a la pantalla anterior.

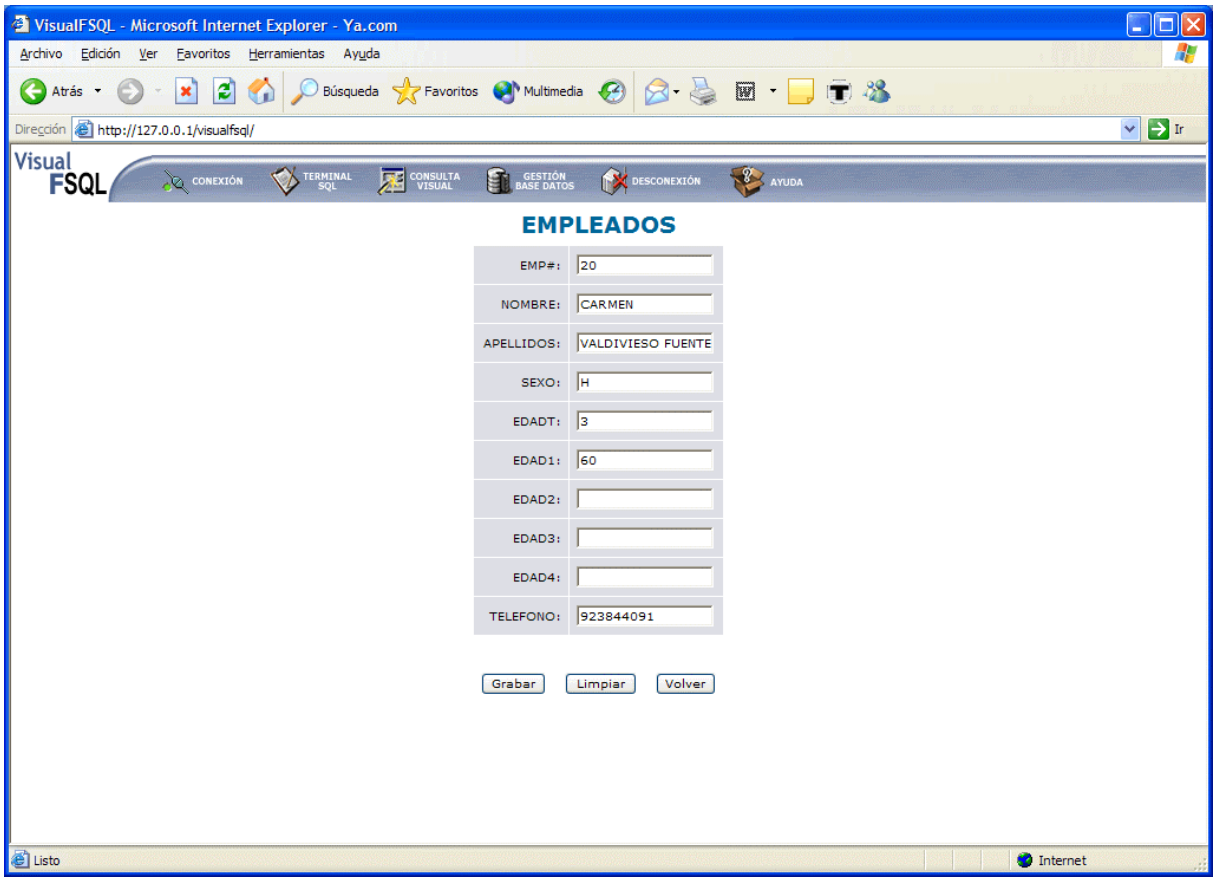


Figura 7.27. Pantalla Consulta/Edición de Registro.

Nuevo Registro

Por último nos quedaría comentar como dar de alta un nuevo registro. Para ello en la parte de "Selección de Tabla" (Figura 7.24) seleccionamos la Tabla y a continuación pulsamos en el botón "Insertar". Entonces nos aparece una pantalla (Figura 7.28) con los campos de la tabla para introducir los valores.

Una vez completados los campos pulsamos en "Grabar" y se insertaría el nuevo registro en la Base de Datos. El botón "Limpiar" restablece los campos a blanco y con "Volver" retrocederíamos a la pantalla anterior.

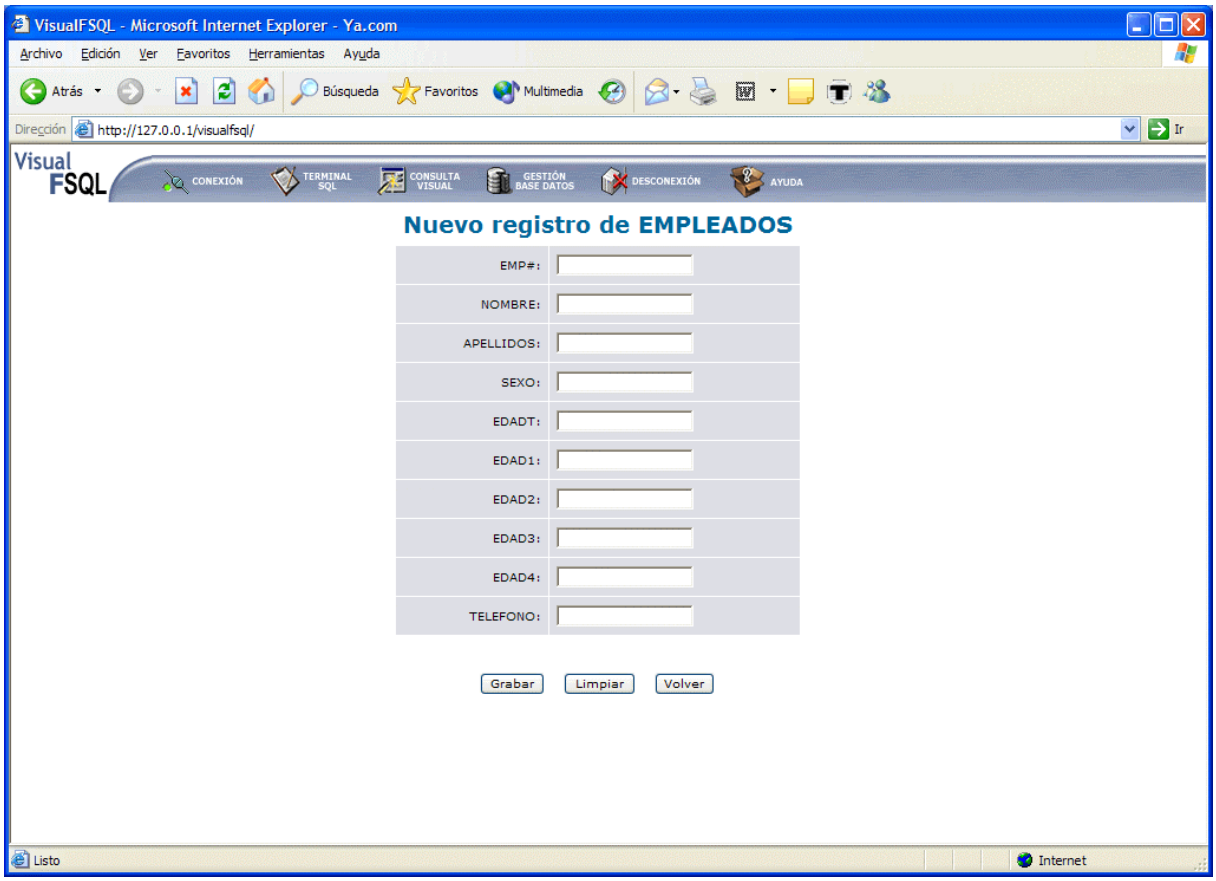


Figura 7.28. Pantalla Alta de Nuevo Registro.

7.6. AYUDA

Por último hay que mencionar el botón de "Ayuda" que se encuentra en "Barra de Menús" de la parte superior derecha. Al pulsar aquí se abrirá una ventana de ayuda con el capitulo actual "Manual de Usuario".

APÉNDICE A. Mensajes de Error en la conexión de ORACLE

Estos son los principales mensajes con los que nos podemos encontrar al intentar establecer la conexión con Oracle.

- "ORA-01017 Nombre de usuario/password erróneo; Conexión denegada"
  El username o el password son erróneos y no existen, por lo que no tenemos acceso al servidor. Debemos corregir estos valores.
- "ORA-06105 No se puede encontrar el Host Remoto; Conexión denegada"
  Lo escrito como Servidor es erróneo y no existe, por lo que debemos cambiarlo si queremos acceso de conexión.
- "ORA-06108 Fallo de conexión al Host"
  Existe un error en la conexión, posiblemente porque estemos haciendo uso de una red que no está en funcionamiento. Debemos hablar con el administrador de la red para que subsane el problema.
- "ORA-06116 Servidor de Oracle no disponible"
  El servidor de SQL*Net TCP/IP no pudo crear el proceso servidor de Oracle. Contactar con el administrador del Sistema para comprobar que se esté ejecutando este proceso (ORASRV).
- "ORA-12154 TNS: No se pudo resolver el nombre del servicio"
  Significa que Net8 no ha podido localizar el nombre del servicio en el archivo de configuración TNSNAMES.ORA.
  Debemos verificar que el nombre del servicio introducido en la aplicación "Servidor (DNI/IP)" sea el correcto, o bien hay que configurar correctamente el servicio en Net8 ó editar el archivo TNSNAMES.ORA.

En caso de que nos aparezcan otros mensajes de error distintos a los aquí expuestos, nos remitiremos al Manual de ORACLE.

Conclusiones y Líneas Futuras

Con este proyecto se ha aportado una gran funcionalidad en cuanto a la gestión y consulta de Bases de Datos Difusas basadas en FSQL. El asistente visual de creación de consultas difusas facilita el acceso a este tipo de Bases de Datos a un mayor número de usuarios potenciales, por lo que el abanico de aplicaciones prácticas se incrementa enormemente.

También es de destacar el tipo de desarrollo de este sistema, al que puede acceder cualquier usuario desde Internet, únicamente con las propias limitaciones de seguridad que queramos establecer. Además la aplicación está totalmente centralizada en un Servidor web y no es necesaria la instalación en los clientes de ningún programa ni componente adicional, únicamente con un Explorador de Internet por parte de los clientes es suficiente. Esto facilita muchísimo la escalabilidad y mantenimiento del sistema, puesto que las modificaciones, actualizaciones ó ampliaciones sólo serían necesarias en el servidor.

Debido a la propia funcionalidad del sistema, que es muy interactivo, se ha optimizado el rendimiento para que el tiempo de respuesta sea el mínimo posible (teniendo en cuenta el tipo de conexión a Internet). Para ello se ha analizado la cantidad de información que el servidor debe enviar en cada una de las peticiones que los clientes realizan a través de Internet, y así intentar reducir el trafico de datos para agilizar el tiempo de respuesta.

También se ha incluido un sistema de Gestión online de la Base de Datos, que nos permite insertar, modificar y borrar datos de tipo crisp en todas las tablas. Igualmente es de destacar que disponemos de Ayuda online en todas las pantallas de la aplicación.

En cuanto a instalación del sistema, el servidor de Bases de Datos lo podemos tener en la misma máquina que el servidor Web de Internet ó bien utilizar otro equipo dedicado para ello, con lo cual se gana en escalabilidad y balanceo de carga. El sistema es flexible a cualquier arquitectura de servidores.

Por otro lado, en la memoria del presente Proyecto se han aportado unos primeros Capítulos con una Introducción a la teoría de la Lógica Difusa, se ha comentado la Arquitectura de la Base de Datos Relacional Difusa utilizada y el Servidor FSQL. A continuación se ha proseguido con un repaso a las técnicas y metodologías de Desarrollo de Aplicaciones Web, materia muy importante en la que se basa la aplicación de este Proyecto. También se han comentado las tecnologías más actuales de Acceso a Datos en el desarrollo de aplicaciones. Los últimos Capítulos se han dedicado al explicar la estructura y desarrollo de Visual FSQL, la instalación completa del sistema y el Manual de Usuario.

A pesar de la enorme potencia, practicidad y aplicabilidad del sistema implementado en este proyecto, se han pensado algunas posibles ampliaciones futuras, entre las que podemos destacar las siguientes:

- Extender la Aplicación para poder trabajar con otros SGBDR distintos de ORACLE (Microsoft SQL-Server 2000, Informix, Sybase, Interbase, etc...), aún cuando se accedan a Bases de Datos Relaciones no Difusas. Sin embargo esto sería algo muy complejo de desarrollar ya que el Servidor FSQL habría que reprogramarlo de nuevo.
- Gestión online de la Base de Datos Difusa. Es decir tener un sistema que nos permita gestionar tablas y registros en la Base de Datos Difusa (operaciones de inserción, actualización y borrado de filas), realizando operaciones automáticas con registros en las Tablas necesarias del Servidor FSQL para mantener la funcionalidad del sistema difuso.
- Posibilidad de usar el Sistema para generar resultados de consultas reutilizables en otras aplicaciones, permitiendo así, enlazar el resultado de una consulta en FSQL con procesadores de texto, hojas de cálculo y otras bases de datos. También sería interesante poder exportar lo resultados a un archivo con alguno de los formatos más usuales de exportación de datos (texto, csv, html, xml, etc...).
- Visualización de las etiquetas de los atributos difusos mediante elementos más visuales (gráficos).
- Gestión más visual de las SubConsultas en la pantalla “Condiciones Simples” de la Consulta Visual.
- Adaptación de la Aplicación para poderla traducir a otros idiomas.
- Reprogramación completa de la Aplicación en otros lenguajes (p.ej. PHP) para

poderla instalar en Servidores Web Linux, ampliando de esta manera el ámbito de operatividad del Sistema.

- Incluir la extensión dmFSQL para Data Mining basado en FSQL [4].
- Además de muchas otras que se puedan estimar convenientes (optimizar el rendimiento de carga y ejecución de las pantallas, cambiar la estructura visual, etc...).

BIBLIOGRAFÍA

[1] J.M. Alarcón. "Programación en Javascript". Anaya Multimedia (2000).

[2] R. Anderson, D. Denault, B. Francis, M. Gibbs, M. Gregorini, A. Homer, C. McQueen, S. Robinson, J.Schenken and K. Williams. "ASP 3.0 Programmer´s Reference". Wrox (2000).

[3] J.C. Bezdek. "Pattern Recognition with Fuzzy Objetive Function Algorithms". Plenum Press. New York (1981).

[4] R.A. Carrasco. Tesis Doctoral : "Lenguajes e Interfaces de Alto Nivel para Data Mining con Aplicación Práctica en Entornos Financieros". Universidad de Granada (2003).

[5] D. Dubois and H. Prade. "Fuzzy Sets and Systems: Theory and Applications". Academic Press. New York (1980).

[6] J. Galindo Gómez. Tesis Doctoral: "Tratamiento de la Imprecisión en Bases de Datos Relacionales: Extensión del modelo y adaptación de los SGBD actuales". Universidad de Granada (1999). www.lcc.uma.es.

[7] J. Galindo Gómez, J.M. Medina Rodríguez, O. Pons y J.C. Cubero. "A Server for Fuzzy SQL Queries". En "Flexible Query Answering Systems", eds. T. Andreasen, H. Christiansen and H.L. Larsen, Lecture Notes in Artificial Intelligence (LNAI) 1495, pp. 164-174. Ed. Springer (1998).

[8] J. Galindo Gómez, J.M. Medina Rodríguez, M.A. Vila y O. Pons. "Fuzzy Comparators for Flexible Queries to Databases". Sixth Iberoamerican Conference on Artificial Intelligence, IBERAMIA'98, pp. 29-41, Lisbon, Portugal, October (1998).

[9] J. Galindo Gómez y J.M. Medina Rodríguez. "FSQL: Consultas Difusas a Bases de Datos Tradicionales ó Difusas". Revista Cuore (Círculo de Usuarios de Oracle España), 24, suplemento Vivat Academia 6, pp. XI-XIX, Octubre (2003).

[10] J. Galindo Gómez, A. Urrutia y M. Piattini. "Fuzzy Databases: Modeling, Design and Implementation". To publish by Idea Group Publishing Hershey, USA (2004/2005).

[11] A. Homer and C. Ullman. "Instant Dynamic HTML". Wrox (1997).

[12] J.M. Medina Rodríguez. Tesis Doctoral: "Bases de Datos Relacionales Difusas: Modelo teórico y aspectos de su implementación". Universidad de Granada (1991).

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

[14] Microsoft. "MASTERING: Distributed Application Design". Official Microsoft Curriculum (1998).

[15] Microsoft. "MASTERING: Enterprise Development". Official Microsoft Curriculum (1998).

[16] Oracle. "ORACLE RDBMS. SQL Language Reference Manual".

[17] F.E. Petry. "Fuzzy Databases: Principles and Applications" (with contribution by Patrick Bosc). International Series in Intelligent Technologies. Ed. H.-J. Zimmermann. Kluwer Academic Publishers (KAP) (1996).

[18] H. Prade and C. Testemale. "Generalizing Database Relational Algebra for the Treatment of Incomplete/Uncertain Information and Vage Queries". Information Sciencies 34, pp.115-143, (1984).

[19] B. Schweizer and A. Sklar. "Probbabilistic Metric Spaces". North-Holland (1983).

[20] J.L. Torralbo Barragán, R. Vela Garrido y D. Rashid Jiménez. Proyecto Fin de Carrera: "Consuldi v.1.0: Aplicación para Consultas difusas en SGBDR Oracle". Universidad de Granada (1995).

[21] E. Trillas. "Sobre Funciones de Negación en la Teoría de Conjuntos Difusos". Stochastica, Vol.3, Nº1, pp. 47-59, (1979).

[22] R.R. Yager. "Fuzzy Sets amd Applications: Selected pappers by L.A. Zadeh". Wiley Intersc. (1987).

[23] L.A. Zadeh. "The Concept of a Linguistic Variable and Its Application to Approximate Reasoning". Information Sci., 8, pp. 199-248, pp. 301-357; 9, pp. 43-80, (1975).

[24] L.A. Zadeh. "Fuzzy Sets". Information Control, 8, pp. 338-353, (1965).

[25] H.J. Zimmermann. "Fuzzy Set Theoria and Its applications. Second Edition". Ed. Kluwer Academy Publisher (1991).

Índice de Figuras

Índice de Tablas