ESCUELA TÉCNICA SUPERIOR
DE
INGENIERÍA INFORMÁTICA

MEDUSA:
UN CLIENTE AVANZADO DE ORACLE

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

MEDUSA:
UN CLIENTE AVANZADO DE ORACLE

Realizado por

ALBERT PHILIPPE DE LA FUENTE VIGLIOTTI

Dirigido por

JOSÉ GALINDO GÓMEZ

Departamento

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA

Málaga, Septiembre de 2003

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

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:

MEDUSA:

UN CLIENTE AVANZADO DE ORACLE

Realizado por D. Albert Philippe De La Fuente Vigliotti

Dirigido por D. 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. _____

A Dios por su inmensa bondad.

A mis padres que siempre están cuando los necesito.

A todas las personas que siempre confiaron en mí.

Y a mi tutor que con mucho esmero me ha prestado su valiosa ayuda en este trabajo.

Muchas gracias.

Capítulo

1

1 Introducción

Este es el comienzo de un recorrido por el apasionante y maravilloso mundo de la programación y de las bases de datos. Medusa, nace de la necesidad de tener un entorno integrado de desarrollo de programación PL/SQL y de consultas y sentencias SQL. Para conseguir este objetivo, primero necesitamos examinar minuciosamente otros desarrollos similares.

Empezamos...

1.1 Preámbulo

La programación orientada a objetos está ganando adeptos durante los últimos años. Cada vez es mayor el número de personas que se sienten intrigadas, en principio, y atraídas, posteriormente, por el mundo de la programación. Las posibilidades ofrecidas son inmensas, ya que no se nos limita a las funciones de una aplicación estándar, más o menos preparada para lo que se espera de ella, sino que se pueden desarrollar aplicaciones propias, a nuestra medida, con todas nuestras preferencias. Éste, por supuesto, será sólo el punto de partida, ya que la mayoría de los programadores no crean programas para sí mismos, sino para terceros.

Nunca ha sido tan fácil como ahora hacer programas, de ahí que se incremente constantemente el número de personas interesadas. Hace sólo unos años, la creación de una aplicación para el entorno Windows requería un profundo conocimiento de programación, así como de la API (*Aplication Programming Interface,* Interfaz de Programación de Aplicaciones) o conjunto de funciones de Windows. Partiendo de esta base y empleando muchas horas, se podía llegar a crear algo funcional. En la actualidad, por el contrario, existen herramientas que nos permiten crear aplicaciones Windows con menor esfuerzo, sin apenas conocimiento del funcionamiento interno de Windows e invirtiendo relativamente poco tiempo.

Por otro lado, en la sociedad de la información en la que estamos inmersos, las bases de datos son imprescindibles. Muchas instituciones requieren almacenar grandes cantidades de información de forma simple y ordenada, con el objeto de poder acceder fácilmente a la información allí almacenada, así como poder modificar y estructurar convenientemente todos los datos ya incluidos. Por ello, la utilización de un Sistema Gestor de Bases de Datos (SGBD) se hace indispensable y facilita la tarea del diseñador a la hora de manejar la inmensa cantidad de datos con los que se trabaja. Cuando la información incluida en ella se encuentra repartida entre distintos emplazamientos se dice que la base de datos está distribuida. Ello supone ventajas considerables en cuanto a fiabilidad, aceleración del proceso de consulta..., aunque también tiene inconvenientes serios, como el mayor coste de desarrollo de los programas, una mayor posibilidad de fallos y sobre todo la coordinación entre los emplazamientos.

1.2 Objetivos del proyecto

En síntesis el principal objetivo de este proyecto consiste en desarrollar una aplicación que intente mejorar algunos de los aspectos del programa de comunicación SQL*Plus de Oracle, como por ejemplo, no trabajar con un *buffer* de ordenes, sino en forma de edición directa, de forma que se aumente la comodidad a la hora de trabajar.

Otro objetivo también muy importante, es tener todo integrado en un mismo programa, la parte de edición y la parte de ejecución de comandos de sentencias SQL (*Structured Query Language*). Esto hace posible ver los resultados devueltos por el SGBD, dentro del mismo entorno del programa, llamado *Medusa*.

Los programadores de SQL*Plus suelen utilizar un editor externo puesto que el interfaz de este programa es un *buffer* que sólo ofrece la posibilidad de edición del último comando y en un editor de líneas. A la hora de trabajar esto no es cómodo, y este es uno de los principales objetivos: que no sea necesario conmutar entre aplicaciones a la hora de realizar algún trabajo, lo cual permite al administrador, programador o usuario realizar múltiples tareas desde una sola aplicación. Esto permitirá mejorar la comodidad, facilidad y rapidez en la ejecución de las órdenes.

Puesto que se desea hacer de este programa, un sistema completo, también se incluirá la posibilidad de crear paquetes, procedimientos, funciones, disparadores (*triggers*) y bloques PL/SQL en general, junto con la posibilidad de ejecutarlos y ver en caso de error la información de éste. También se ha intentado facilitar, de alguna forma, la creación de estos objetos.

Algunos objetivos más que podemos destacar para facilitar la tarea del programador, son por ejemplo, el realzado de sintaxis, la exportación a otros formatos, como por ejemplo HTML (*HyperText Markup Language*), de los códigos fuente de los *scripts* o consultas, para la integración en la red. Además de esto, también es un completo editor de texto con sus funcionalidades inherentes como son abrir, guardar, copiar, cortar, pegar, imprimir, previsualizar, etc. También incluiremos un menú de consultas favoritas de usuario, de forma que con un solo click se puedan realizar multitud de tareas. También se incorporarán consultas predefinidas al diccionario de datos para facilitar

el acceso a información sobre tablas, vistas de usuario, vistas accesibles, llaves primarias de cada tabla, otras restricciones y roles entre otras.

Los beneficios esperados, son principalmente una mayor facilidad y comodidad a la hora de programar, efectuar consultas u otro tipo de sentencias. También se desea integrar en *Medusa* un interfaz MDI (Interfaz de Múltiples Documentos), de forma que se puedan comparar resultados, o depurar programas de forma más fácil.

1.3 Trabajos relacionados

En esta sección nos dedicamos a ver las características principales de otros programas que interactúan con Oracle. Los llamados entornos de desarrollo integrados (IDE, *Integrated Desktop Environment*) poseen muchas características muy atrayentes, entre las principales, encontramos que se puede hacer prácticamente todo, desde estos entornos. Veamos algunos de ellos.

1.3.1 SQL*Plus Worksheet

Es un cliente sencillo de Oracle basado en SQL*Plus. Funciona bajo línea de comandos, aunque se puede hacer de todo, es un programa muy poco intuitivo. La característica más importante que se incluye en SQL*Plus Worksheet es la posibilidad de guardar las sentencias en un buffer con un tamaño de hasta 50 sentencias, donde se pueden ejecutar cualquiera de ellas de forma sencilla. También se incluyen facilidades para guardar las sentencias en ficheros de texto.

1.3.2 SQL Navigator (actualmente Toad) de Quest Software

SQL Navigator es un entorno avanzado de desarrollo para Oracle server. Posee numerosas características de mejoras de productividad que simplifican las tareas complejas y reducen el trabajo de los desarrolladores y los administradores de bases de datos. Su web está disponible en **[**W3QUE**]**.

1.3.2.1 Algunas características

1. **Navegador de bases de datos:**

- Vista jerárquica de los objetos de la base de datos
- Sesiones ilimitadas concurrentes

2. **Edición visual de objetos de la base de datos:**

- Restricciones
- Enlaces
- Índices
- Procedimientos, funciones y paquetes
- Roles
- Secuencias
- Sinónimos
- Tablas
- Disparadores (*triggers*)
- Usuarios

3. **Ventana de ejecución de comandos SQL:**

- Resaltado de la sintaxis de SQL y resaltado de errores
- Procesos largos se pueden cancelar
- Capacidad de ejecutar sólo el texto seleccionado

4. **Tratamiento de *Scripts* SQL:**

- Capacidad de edición y ejecución de ficheros con múltiples comandos SQL
- Capacidad de ejecutar, ejecutar paso a paso, saltar... comandos SQL

5. **Editor de texto:**

- Drag and Drop
- Configuración de los colores del realzado de sintaxis
- Impresión con realzado de sintaxis
- Deshacer/Rehacer ilimitado
- Accesos de teclado programables
- Marcadores visuales

6. **Herramientas de codificación:**

- Asistentes SQL para `SELECT`, `INSERT`, `UPDATE` y `DELETE`

7. **Planificación gráfica:**

- Jerarquía de colores en el código del plan SQL

8. **Herramientas estadísticas:**

- Posibilidad del análisis de tablas, índices y clusters
- Capacidad de ver estadísticas

1.3.3 OraTools de CoreLab

OraTools es una herramienta poderosa y versátil suplementaria con las herramientas de depuración y administración que posee Oracle. Su web está disponible en **[**W3CRL**]**. Algunas características:

- Interfaz de usuario amigable y consistente (se explica detalladamente a continuación).
- Conexión única a la base de datos.
- Utilidades para transacciones.

1.3.3.1 Algunas utilidades de OraTools

OraDesigner. Permite escribir y ejecutar consultas SQL, así como un conjunto de éstas envueltas en bloques PL/SQL, procedimientos almacenados y consultas parametrizadas.

Las Características de OraDesigner son:

- Realzado de sintaxis de PL/SQL
- Rejilla de datos avanzada, especial para datos, con posibilidades de filtrado, búsqueda y controles de navegación para facilitar la interpretación
- Cargar/guardar ficheros de *scripts*
- Página de plan de ejecución

OraExplorer. Es un explorador sencillo de usar para objetos del esquema y meta-datos (información sobre datos) disponibles. Los nombres de los objetos están organizados jerárquicamente en un árbol.

PL/SQL Debugger. Es un depurador de programas PL/SQL, para poder ver el hilo de ejecución de los comandos, al igual de cómo son ejecutados por Oracle. La ejecución paso a paso permite obtener información sobre qué es lo que ocurre en el programa y su entorno. Están disponibles todos los comandos usuales de depuración: paso dentro, paso sobre, paso fuera, puntos de ruptura (añadir o quitar), etc. Se pueden añadir variables a la lista "*watch list*" para inspeccionar de forma interactiva o incluso modificar los contenidos.

1.3.4 PL/SQL Developer de Allround Automations

PL/SQL Developer es un entorno de desarrollo integrado (IDE, *Integrated Desktop Environment*) para el desarrollo de programas para una base de datos Oracle. Al utilizar PL/SQL Developer, se puede crear la parte del servidor para las aplicaciones de los clientes. PL/SQL Developer, se puede descargar de **[**W3ARA**]**.

Antes, en el peor de los casos, era necesario seguir aproximadamente estos pasos para poder desarrollar un programa:

1. Utilizando un editor de texto, escribíamos el programa (procedimientos, *triggers*...).
2. Utilizábamos SQL*Plus para compilar los fuentes.
3. Si hay un error de compilación, era necesario localizar donde se encontraba en el código fuente, corregirlo y nuevamente conmutar a SQL*Plus para recompilar, sólo para buscar el siguiente error.
4. Utilizábamos SQL*Plus o el programa cliente, para probar el programa.
5. En el caso de un error de ejecución, nuevamente era necesario localizar el error y corregirlo.
6. Finalmente, para ver o modificar cualquier objeto de la base de datos, usábamos SQL*Plus u otra herramienta.

Las tareas de edición, compilación, corrección, prueba, depuración, se pueden hacer todas desde PL/SQL Developer. Además, PL/SQL Developer proporciona muchas otras herramientas que pueden ser útiles durante el proceso de desarrollo de programas PL/SQL.

Algunas características de PL/SQL Builder son:

- Potente editor de múltiples documentos (MDI) con realzado de sintaxis y ayuda contextual de comandos SQL y PL/SQL.
- Compilación y corrección de errores, con su localización exacta en el código fuente.
- Potente depurador de bloques PL/SQL.
- Optimización integrada, usando la utilidad de planificación de Oracle.
- Entorno multi-consulta. Las consultas se almacenan en un buffer, de forma que se puedan recuperar rápidamente. Cada consulta nueva tiene asociada su propia ventana y rejilla para poder visualizar los resultados.
- Ejecución de *scripts* PL/SQL y comandos similares a SQL*Plus
- Fácil modificación de objetos de la base de datos, mediante interfaz gráfico.
- Generador de informes basado en HTML.

1.4 Estructura de la memoria

A continuación describiremos brevemente la estructura y contenido de cada uno de los capítulos de esta memoria.

Capítulo 2. Oracle

En este capítulo vemos las características generales de los sistemas gestores de bases de datos, en particular de Oracle, un gestor de bases de datos basado en el lenguaje de consulta *SQL*. También trataremos en este capítulo, la instalación paso a paso de Oracle.

Capítulo 3. Delphi

Hablaremos en este capítulo de Delphi, las características generales del lenguaje de programación, así como la instalación de su compilador, justificando el porqué hemos elegido este lenguaje de programación.

Capítulo 4. Componentes de programación: DOA y SynEdit

En este capítulo conoceremos los componentes de programación empleados: el interfaz de acceso a Oracle, que proporciona la librería DOA, y el reconocimiento sintáctico que nos brinda SynEdit.

Capítulo 5. En las profundidades de *Medusa*

Explicaremos *Medusa* en profundidad, al igual que detallaremos toda la teoría relacionada con MDI (*Multiple Document Interface,* Interfaz Múltiple de Documentos), así como también se indicarán, los problemas que se han planteado durante su desarrollo, cómo se han resuelto y la estructura interna del programa.

Capítulo 6. Detalles finales: Generación de la ayuda y de la instalación

En este capítulo, explicamos cómo hacer y como compilar los ficheros de ayuda para el usuario, que se incluyen en *Medusa*, junto con la realización de la instalación.

Capítulo 7. Medusa: Manual de usuario

Explicamos detalladamente el manual de usuario de este programa desarrollado, incluyendo cada uno de los menús y opciones que nos encontraremos dentro de *Medusa*, junto con algunos comentarios y trucos.

Conclusiones y líneas futuras

Finalizamos indicando las aportaciones realizadas en este proyecto, así como un conjunto de ideas que podrían añadirse o mejorarse para sucesivas versiones de *Medusa*.

Referencias bibliográficas

Incluimos libros, manuales y los enlaces de Internet consultados y a los que se hace referencia a lo largo de esta memoria.

Capítulo

2

2 Oracle

Oracle fue la primera base de datos relacional del mercado. El objetivo de una base de datos relacional consiste en construir una estructura en la cual las modificaciones requeridas no la afecten a ella, sino únicamente a los datos, es decir, se minimicen las modificaciones a las aplicaciones, se termine con la redundancia de los datos y se garantice la sincronización de los cambios hechos a los mismos.

2.1 ¿Qué es Oracle?

En este capítulo, expondremos una descripción de Oracle, actualmente uno de los SGBDR (Sistemas Gestores de Bases de Datos Relacional. Aunque actualmente también es objeto-relacional) más ampliamente usados. Un *servidor Oracle* consta de una *base de datos Oracle* (el conjunto de datos almacenados que incluye el diario (log) y los ficheros de control) y la *instancia Oracle* (los procesos, que incluyen los procesos del sistema Oracle y los procesos de usuario tomados en conjunto, creados para una instancia específica de la operación de base de datos). El servidor Oracle soporta SQL para definir y manipular los datos. Además, tiene un lenguaje procedimental, llamado PL/SQL, para controlar el flujo de SQL, para usar variables y para proporcionar procedimientos y funciones de manejo de errores u otras operaciones en la base de datos. A Oracle también se puede acceder desde lenguajes de programación de carácter general tales como C, Java o Delphi. Se puede obtener más información de Oracle, en [W3ORA].

La base de datos Oracle tiene dos estructuras primarias: (1) *una estructura física*, que hace referencia a los datos realmente almacenados, y (2) *una estructura lógica*, correspondiente a una representación abstracta de los datos almacenados, que se corresponde con el esquema conceptual de la base de datos.

Un sistema gestor de bases de datos (SGBD, también conocido como DBMS, DataBase Management System) es una colección de programas que permiten a los usuarios crear y mantener una base de datos. El SGBD es por tanto un *sistema software de propósito general* que facilita los procesos de *definición, construcción y manipulación* de bases de datos para distintas aplicaciones. La *definición* de una base de datos consiste en especificar los tipos de datos, las estructuras y restricciones para los datos que se van a almacenar en dicha base. La *construcción* de la base de datos es el proceso de almacenar los datos concretos sobre algún medio de almacenamiento controlado por el SGBD. La manipulación de la base de datos incluye funciones tales como consultar la base de datos para recuperar unos datos específicos, actualizar la base de datos para reflejar los cambios ocurridos y generar informes a partir de datos.

No es preciso utilizar software de SGBD de propósito general para implementar una base de datos informatizada. Podríamos codificar nuestro propio conjunto de programas para crear y

mantener la base de datos, es decir, crear nuestro propio software de SGBD de *propósito específico*. En cualquier caso, usemos o no un SGBD de propósito general, normalmente tendremos que emplear gran cantidad de software para manipular la base de datos. Denominaremos *sistema de base de datos* al conjunto formado por la base de datos más el SGBD.

La arquitectura de los paquetes de SGBD ha evolucionado desde los primeros sistemas monolíticos, donde todo el paquete de software del SGBD era un sistema completamente integrado, a los actuales paquetes de SGBD que tienen un diseño modular, según la arquitectura cliente-servidor. Esta evolución refleja la tendencia en computación, donde los grandes computadores centralizados están siendo reemplazados por cientos de estaciones de trabajo y computadores personales distribuidos y conectados mediante redes de comunicación. En una arquitectura cliente-servidor básica la funcionalidad del sistema se distribuye entre dos tipos de módulos. El *modulo cliente* se suele ejecutar sobre una estación de trabajo o un computador personal del usuario. Normalmente, los programas de aplicación y las interfaces de usuario que acceden a la base de datos se ejecutan en el módulo cliente. Así, el módulo cliente maneja la interacción del usuario y proporciona interfaces amigables como GUI (*Graphics User Intefrace,* interfaces gráficas de usuario) basadas en formularios o menús. El otro tipo de módulo, denominado *módulo servidor*, normalmente maneja el almacenamiento, acceso, búsqueda de datos, y otras funciones.

2.2 Características de los SGBD

Hay varias características que distinguen el enfoque de bases de datos del enfoque tradicional de programación con ficheros. En el *procesamiento de ficheros* tradicional, cada usuario define e implementa los ficheros necesarios para una aplicación específica como parte de la programación. Para una información más detallada, puede consultar **[**ELM02**]**.

2.2.1 Naturaleza autodescriptiva de los sistemas de base de datos

Una característica fundamental del enfoque de base de datos es que el sistema de base de datos no sólo contiene la base de datos propiamente dicha sino también una definición o descripción completa de la estructura de la base de datos y sus restricciones. Esta definición se almacena en el *catálogo* del sistema. La información almacenada en el catálogo se denomina *meta-datos*, y describe la estructura de la base de datos.

El catálogo es utilizado por el software del SGBD y también por los usuarios que precisan información sobre la estructura de la base de datos. Un paquete de software de un SGBD de propósito general no se crea para una aplicación de base de datos específica y por tanto es preciso consultar el catálogo para averiguar la estructura de los ficheros de una base de datos específica, como el tipo de formato de los datos a los que se va a acceder. El software del SGBD ha de trabajar igual de bien con *cualquier número de aplicaciones de base de datos* (por ejemplo, una base de datos universitaria, una base de datos de un banco o una base de datos de una empresa) siempre que la definición de la base de datos esté almacenada en el catálogo.

2.2.2 Separación entre los programas y los datos: abstracción de datos

En el procesamiento de ficheros tradicionales, la estructura de los ficheros de datos viene integrada en los programas de acceso, así que cualquier modificación de la estructura de un fichero puede requerir la *modificación de todos los programas* que acceden a dicho fichero. Por el contrario, los programas de acceso del SGBD no requieren dichas modificaciones. La estructura de los ficheros de datos se almacena en el catálogo del SGBD separadamente de los programas de acceso. A esta propiedad se le denomina *independencia entre programas y datos*.

La característica que permite la independencia de programas y datos se llama *abstracción de datos*. Un SGBD ofrece a los usuarios una *representación conceptual* de los datos que no incluye muchos detalles sobre el almacenamiento de los mismos. En términos informales, un *modelo de datos* es un tipo de abstracción de datos que se utiliza para proporcionar esta representación conceptual.

2.2.3 Soporte de múltiples vistas de los datos

Una base de datos suele tener muchos usuarios, cada uno de los cuales puede requerir una perspectiva o *vista* diferente de la base de datos. Una vista puede ser un subconjunto de la base de datos o puede contener *datos virtuales* derivados de los ficheros de la base de datos pero que no están explícitamente almacenados. Es posible que algunos usuarios no necesiten saber si los datos a los que hacen referencia están almacenados o son derivados. Un SGBD multiusuario cuyos usuarios tengan diversas aplicaciones debe proporcionar mecanismos para definir múltiples vistas.

2.2.4 Compartimiento de datos y procesamiento de transacciones multiusuario

Todo SGBD multiusuario, como su nombre indica, debe permitir a varios usuarios tener acceso simultáneo a la base de datos. Esto es indispensable si los datos de múltiples aplicaciones se deben integrar y mantener en una sola base de datos. El SGBD debe incluir software de *control de concurrencia* para asegurar que cuando varios usuarios intenten actualizar los mismos datos lo hagan de manera controlada para que el resultado de las actualizaciones sea correcto. Para ello, lo ideal, sería disponer de *aislamiento entre las transacciones,* lo cual es generalmente deseable, pero puede comprometer el rendimiento de muchas aplicaciones. Hay diversos tipos de errores relacionados con el control de concurrencia, entre los que encontramos los siguientes:

- Lecturas erróneas (*dirty reads*): Una transacción lee datos que han sido escritos por otra transacción que aún no ha sido confirmada.
- Doble lectura (*non-repeatable reads*): Una transacción lee datos que ya había leído, encontrándose que, entre las dos lecturas, los datos han sido modificados o borrados por una transacción que ya ha sido confirmada.
- Lectura fantasma (*phantom read*): Una transacción reejecuta una consulta encontrando que el conjunto de filas resultantes ha sido ampliado por otra transacción que insertó nuevas filas y que ya ha sido confirmada.

2.2.5 Control de redundancia

La redundancia, es el almacenamiento de los mismos datos varias veces, lo cual provoca varios problemas. En primer lugar, es necesario realizar una misma actualización lógica (como introducir los datos de un nuevo alumno) varias veces: una vez por cada localización en el que se registren los datos. Esto implica una *duplicación del trabajo*. En segundo lugar, se *desperdicia espacio de almacenamiento* al guardar los mismos datos en varios sitios, y este problema puede ser grave si las bases de datos son grandes. En tercer lugar, es posible que los ficheros que representan los mismos datos, se vuelvan *inconsistentes*. Esto puede suceder porque una actualización se haya aplicado a ciertos datos pero no a otros. Incluso si la actualización (por ejemplo, la modificación del nombre de un alumno) se aplica a todas sus localizaciones, persiste la posibilidad de que los datos (los relacionados con el alumno) sean *inconsistentes* porque cada grupo de usuarios aplica las actualizaciones de manera independiente.

Con el enfoque de bases de datos, las vistas de los diferentes grupos de usuarios se integran durante el diseño de la base de datos. Para conservar la consistencia, debe crearse un diseño que almacene cada dato lógico en un solo lugar de la base de datos. Ello evita la inconsistencia y ahorra espacio de almacenamiento.

2.2.6 Restricción de los accesos no autorizados

Cuando muchos usuarios comparten una misma base de datos, es probable que no todos tengan la autorización para acceder a toda la información que contiene. Por ejemplo, es habitual considerar que los datos financieros son confidenciales y que sólo ciertas personas pueden tener autorización para acceder a los mismos. Además, es posible que sólo algunos usuarios tengan permiso para recuperar datos, mientras que a otros se les permita obtenerlos y actualizarlos.

Por tanto, también es preciso controlar el tipo de operaciones de acceso (recuperación o actualización). Normalmente, a los usuarios o grupos de usuarios se les asignan números de cuenta protegidos con contraseñas, que sirven para tener acceso a la base de datos. El SGBD debe contar con un *subsistema de seguridad y autorización* que permita al administrador crear cuentas y especificar restricciones para ellas. El SGBD deberá entonces garantizar automáticamente el cumplimiento de dichas restricciones.

2.2.7 Garantizar el cumplimiento de las restricciones de integridad

La mayoría de las aplicaciones de la base de datos tienen ciertas *restricciones de integridad* que deben cumplir los datos. El SGBD debe ofrecer recursos para definir tales restricciones y garantizar que se cumplan. La restricción de integridad más simple consiste en especificar un tipo de datos para cada elemento de datos. Otro tipo de restricción más compleja que encontramos a menudo, es la que implica especificar que un registro de un fichero debe relacionarse con registros de otros ficheros. Estas restricciones se derivan del significado o *semántica* de los datos y del mundo que representan. Es responsabilidad de los diseñadores de la base de datos identificar las restricciones de integridad durante el diseño. Algunas restricciones se pueden especificar en el SGBD, el cual garantizará automáticamente que se cumplan; otras pueden requerir verificación mediante programas de actualización o en el momento en que se introducen los datos (como los disparadores).

2.2.8 Suministro de copias de seguridad y recuperación

Todo SGBD debe contar con recursos para recuperarse de fallos de hardware o de software. El *subsistema de copias de seguridad (backup) y recuperación* del SGBD es el responsable de llevar a cabo dicha recuperación. Por ejemplo, si el sistema falla mientras se está ejecutando un complejo programa de actualización, el subsistema de recuperación se encargará de asegurarse de que la base de datos se restaure al estado en el que estaba antes de que comenzara la ejecución de dicho programa. Como alternativa, el subsistema de recuperación puede asegurarse de que el programa reanude su ejecución en el punto en que fue interrumpido, de modo que su efecto completo se registre en la base de datos.

2.3 Elementos de una base de datos Oracle

Una *base de datos* es un conjunto de datos relacionados. Oracle proporciona la capacidad de almacenarlos y de acceder a ellos de una forma coherente con un modelo definido y conocido como el modelo relacional. Debido a esto, Oracle constituye lo que se conoce como un sistema de gestión de bases de datos relacionales (RDBMS, *Relational Database Management System*, Sistema de Manejo de Bases de Datos Relacionales). Cuando se utiliza el término de *base de datos* no solo nos estamos refiriendo a los datos físicos, sino también a la combinación de objetos físicos, de memoria y de procesos.

Los datos de una *base de datos relacional* se almacenan en *tablas* (o relaciones). Las tablas relacionales están definidas por sus *columnas* y se les asigna un nombre. Los datos se almacenan como *filas* de la tabla. Las tablas pueden estar relacionadas entre sí y la base de datos puede utilizarse para hacer que se apliquen esas relaciones. En la Figura 2.1 se muestra un ejemplo de la estructura de una tabla.

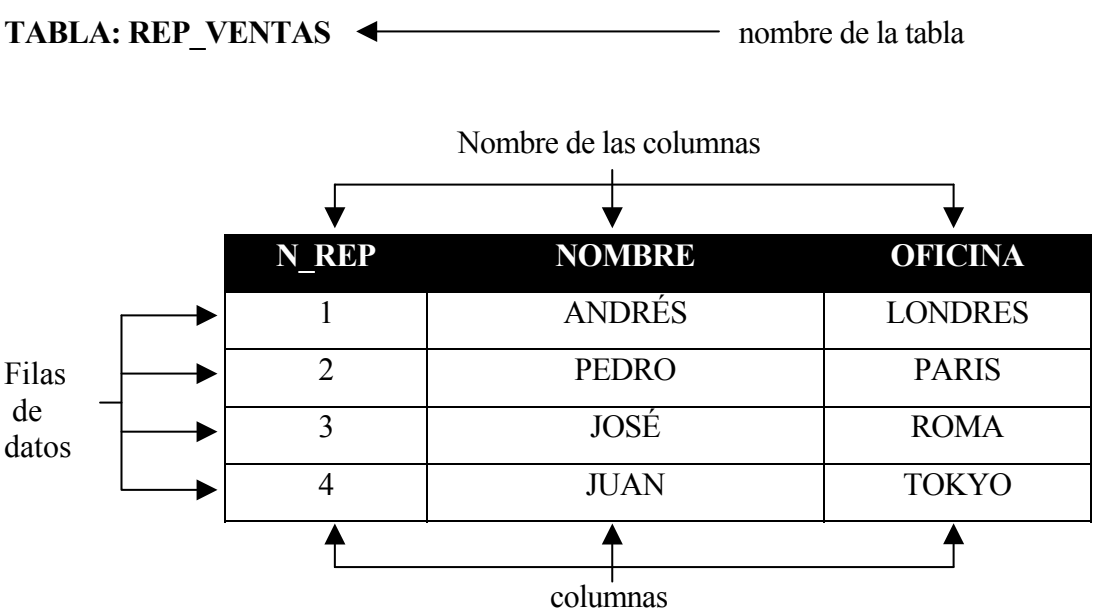

Figura 2.1: Ejemplo de la estructura de una tabla

Además de almacenar datos en formato relacional, Oracle (a partir de Oracle8) admite estructuras *orientadas a objetos (OO)*, como métodos y tipos abstractos de datos. Los objetos pueden estar relacionados con otros objetos y contener otros objetos. También se pueden usar vistas de objetos para definir interfaces orientadas a objetos para los datos sin hacer ninguna modificación en las tablas.

Tanto si se utilizan estructuras relacionales, como si se utilizan estructuras orientadas de tipo OO, la base de datos Oracle almacena los datos en archivos. Internamente, existen estructuras en la base de datos que se encargan de la correspondencia lógica entre los datos y los archivos, lo que permite almacenar por separado diferentes tipos de datos. Estas divisiones lógicas se denominan espacios de tabla (*tablespace*).

Ahora veamos un poco más en detalle los elementos propiamente dichos de una base de datos *relacional Oracle*. Después de tener unas nociones básicas de los principales elementos, daremos un paseo, con un poco más de detenimiento por uno de los elementos más importantes de Oracle, que es el *diccionario de datos* (Sección 2.4).

Sobre una base de datos, se pueden crear estructuras internas que den soporte a las aplicaciones. Entre los elementos internos de la base de datos se incluyen los siguientes:

- Tablas, columnas, restricciones y tipos de datos.
- Usuarios y esquemas.
- Índices, clusters y clusters hash.
- Vistas.
- Secuencias.
- Procedimientos, funciones, paquetes y disparadores.
- Sinónimos.
- Privilegios y roles.
- Instantáneas o vistas materializadas.

Ahora, veamos un poco mas en detalle, estas estructuras internas.

2.3.1 Tablas

Las tablas son el mecanismo de almacenamiento de los datos en una base de datos Oracle. Como se mostró anteriormente en la Figura 2.1, contienen un conjunto fijo de columnas. Las columnas de una tabla describen los atributos de la entidad que se representa con la tabla. Cada columna tiene un nombre y unas características específicas.

Una columna tiene un *tipo de datos* y una *longitud*. En las columnas que utilizan el tipo de datos `NUMBER`, pueden especificarse las características de precisión y escala. La *precisión* determina el número de cifras significativas de un valor numérico. La *escala* determina la posición de la coma decimal por ejemplo, la especificación `NUMBER(9,2)` en una columna, indica que tiene un total de nueve cifras, de las cuales dos son decimales. La precisión predeterminada es de 38 cifras, que además coincide con la precisión máxima. Hay muchos más tipos de datos, para saber más sobre éstos, se puede consultar **[**LON02**]** y el manual de Oracle **[**ORA02**]**.

El usuario *SYS* posee unas tablas que se denominan *tablas del diccionario de datos* y proporcionan un catálogo del sistema que la base de datos utiliza para gestionarse a sí misma. El diccionario de datos lo crea un conjunto de programas de catálogo incluido en Oracle. Cada vez que se instala o actualiza una base de datos, es necesario ejecutar esos programas que creen o que modifiquen las tablas del diccionario de datos. El usuario *SYSTEM* posee vistas en las tablas del diccionario de datos.

Las tablas se relacionan entre sí mediante las columnas que tienen en común. La base de datos puede utilizarse para asegurar que estas relaciones estén correctamente establecidas por medio de la *integridad referencial*. La integridad referencial se impone en el nivel de base de datos por medio de restricciones.

2.3.2 Restricciones

Se pueden crear restricciones en las columnas de una tabla, cuando esto ocurre, todas las filas de la tabla deben cumplir las condiciones que se especifiquen en la definición de la restricción.

La *clave primaria* de una tabla es la columna o conjunto de columnas que hacen que cada fila de la tabla sea única. Una columna de clave primaria se definirá dentro de la base de datos como `NOT NULL`, lo que significa que toda fila de dicha tabla deberá tener un valor para esa columna, no pudiendo dejarse en blanco (`NULL`). La restricción `NOT NULL` puede aplicarse a cualquier columna de la tabla.

Una columna puede tener una restricción `DEFAULT`. Este tipo de restricción se utiliza para generar un valor en una columna cuando se inserta una fila en una tabla sin especificarse un valor para dicha columna.

La restricción `CHECK` se utiliza para asegurarse de que los valores de una determinada columna cumpla un cierto criterio o condición. Una restricción `CHECK` no puede hacer referencia a una tabla diferente. La base de datos trata una restricción `NOT NULL` como si se tratara de una `CHECK`.

Otra restricción, `UNIQUE`, garantiza la unicidad de una o varias columnas que deben ser unívocas, pero que no forman la clave primaria. Una característica de esta restricción, es que sí aceptan `NULL` en las columnas de la llave candidata.

Para especificar la naturaleza de la relación entre tablas, se utiliza una restricción de *clave externa*. Una clave externa de una tabla hace referencia a una clave primaria que se haya definido previamente en cualquier otro lugar de la base de datos.

Las restricciones de la base de datos ayudan a garantizar la integridad de los datos. De esta forma, uno puede estar seguro de que todas las referencias de la base de datos son válidas y de que se cumplen todas las restricciones.

2.3.3 Usuarios

Una cuenta de usuario no es una estructura física de la base de datos, pero sí que tiene importantes relaciones con los objetos de la base de datos: *los usuarios son propietarios de los objetos de la base de datos*. El usuario SYS es propietario de las tablas del diccionario de datos, en las

que se almacena información sobre el resto de las estructuras de la base de datos. El usuario SYSTEM posee las vistas que permiten acceder a estas tablas del diccionario de datos, para que las utilicen los restantes usuarios de la base de datos.

Cuando se crean objetos en la base de datos, se crean bajo cuentas de usuario. Cada una de estas cuentas se puede personalizar para que utilice un espacio de tablas específico de manera predeterminada.

Las cuentas de la base de datos se pueden asociar a cuentas del sistema operativo, permitiendo así a los usuarios el acceso a la base de datos desde el sistema operativo, sin tener que introducir una contraseña para el sistema operativo y luego otra para la base de datos. Los usuarios pueden acceder a los objetos que poseen o a aquellos a los que se les ha concedido acceso.

2.3.4 Esquemas

El conjunto de objetos que posee una cuenta de usuario se denomina *esquema* (*scheme*) del usuario. Se pueden crear usuarios que no tengan la capacidad de iniciar una sesión en la base de datos. Tales cuentas de usuario proporcionan un esquema que puede utilizarse para guardar un conjunto de objetos de base de datos separados de otros esquemas de usuario.

2.3.5 Índices

Para que sea posible encontrar los datos, todas las filas de todas las tablas se etiquetan con un identificador llamado *RowID (identificador de fila)*. Este identificador de fila indica a la base de datos la ubicación exacta de la fila (archivo, bloque del archivo y fila del bloque).

Existen diversos tipos de índices entre los cuales destacan tres tipos: índices de cluster, índices de tabla e índices de mapa de bits. Los índices de cluster almacenan los valores de clave de cluster en clusters. Un índice de tabla almacena los valores de las filas de una tabla junto con su ubicación física de la fila, es decir, su RowID. Un índice de mapa de bits es un tipo especial de índice de tabla diseñado para dar soporte a consultas de tablas de gran tamaño con columnas que contengan pocos valores distintos.

Los índices constan de un valor clave y de un identificador de fila (RowID). Se puede indexar una sola columna o un conjunto de columnas. Oracle almacena los elementos de los índices utilizando un mecanismo de árbol binario que garantiza una ruta de acceso corta hasta el valor clave. Cuando una consulta accede a una tabla con índice, busca las entradas del índice que coincidan con los criterios de consulta. El valor RowID que coincida con la consulta proporciona a Oracle la ubicación física de la fila asociada, reduciendo así el tiempo necesaria para localizar los datos.

Los índices se utilizan tanto para mejorar el rendimiento como para garantizar (opcionalmente) la unicidad de una columna. Oracle crea un índice, de forma automática, siempre que se especifique una cláusula de restricción `UNIQUE` o `PRIMARY KEY` en un comando `CREATE TABLE` (*crear tabla)*. El comando `CREATE INDEX` (*crear índice*) sirve para crear índices de forma manual.

A partir de Oracle7.3 pueden crearse *índices de mapas de bits*. Este tipo resulta muy útil cuando los datos no son muy diversos (cuando hay muy pocos valores distintos en una columna). Los índices de mapas de bits aceleran las búsquedas en las que se utilizan esas columnas como limitaciones para las consultas. Los índices de mapas de bits no son muy efectivos con los datos muy estáticos.

En Oracle8 pueden crearse índices que *invierten* el orden de los bytes de los datos antes de almacenarlos. La inversión del orden antes de la indexación ayuda en ocasiones a mantener los datos mejor distribuidos dentro del índice. Como estos índices invierten los valores de los datos, sólo se usan para realizar operaciones sobre un dato concreto. Sobre un conjunto de datos (un rango de valores impuesto por una condición “>” o “<”), los índices de orden inverso no satisfacen eficazmente las necesidades porque los valores consecutivos no se almacenarán uno al lado del otro. Puesto que las filas consecutivas se almacenarán en ubicaciones separadas, las consultas basadas en rangos no pueden utilizar los índices de clave inversa.

En Oracle8 se puede crear una tabla organizada mediante índice. En este tipo de tabla, toda la tabla se almacena dentro de una estructura de índice, con los datos ordenados por la clave primaria de la tabla. Para crear una tabla organizada mediante un índice, hay que especificar una restricción de clave primaria para la tabla. La tabla no tendrá identificadores de fila (RowID) para las filas, Oracle utiliza el valor de clave primaria como un valor de identificador de fila lógico.

Se pueden crear índices basados en funciones utilizando funciones de Oracle o funciones definidas por los usuarios. Por ejemplo, podemos crear un índice basado en `UPPER(Name)`, en lugar de utilizar simplemente la columna `Name` si sabemos que las cláusulas `WHERE` de las consultas utilizan con frecuencia la función `UPPER` sobre esa columna.

Para una información complementaria sobre índices, se puede consultar **[**ABB02**]** y el manual de Oracle **[**ORA02**]**.

2.3.6 Clusters

Las tablas a las que se suele acceder conjuntamente pueden almacenarse físicamente juntas. Para almacenarlas juntas, se crea un `CLUSTER` que contenga las tablas. Los datos de las tablas se almacenan juntos con el fin de minimizar el número de operaciones de E/S que deben realizarse y mejorar así el rendimiento.

Las columnas relacionadas de las tablas (mediante una clave externa) se denominan *clave de cluster*. Esta clave de cluster se indexa utilizando un *índice de cluster*, y su valor se almacena una única vez para las diversas tablas del cluster.

Los clusters pueden resultar ventajosos en las tablas que se consultan con frecuencia. Dentro del cluster, filas de tablas diferentes se almacenan en los mismos bloques, de forma que las consultas que combinen estas tablas no necesitan llevar a cabo tantas operaciones de E/S como si las tablas se almacenaran en ubicaciones diferentes. Sin embargo, el rendimiento de las operaciones de inserción, actualización y eliminación de tablas agrupadas puede ser notablemente inferior que las mismas operaciones realizadas en tablas no agrupadas.

2.3.7 Clusters hash

Existe un segundo tipo de cluster: los `CLUSTER HASH`. Estos clusters utilizan *funciones de tipo hash* sobre la clave de cluster de la fila para determinar la ubicación física en la que debe almacenarse una fila. Así se obtiene el mayor rendimiento en las consultas de equivalencia (coinci-

dencia exacta), aunque la eficiencia no suele ser la misma si, en la cláusula `WHERE`, se especifica un rango de valores. El uso de una función hash reduce el número de operaciones E/S necesario para devolver las filas en clases de equivalencia.

2.3.8 Vistas

Una *vista* es una porción personalizada de los datos almacenados en una o más tablas base. A su vez, las tablas base pueden ser tablas o también vistas. A diferencia de una tabla, una vista no contiene datos, sino, simplemente, una instrucción SQL almacenada. Cuando un usuario ejecuta una consulta que accede a la vista, la base de datos se dirige al diccionario de datos, recupera la instrucción SQL almacenada y ejecuta la instrucción. Los datos recuperados de esta consulta se presentan en forma de tabla.

De hecho, si no se crea una vista, se pensaría que se está trabajando con una tabla. Como ocurre con una tabla, se puede insertar, actualizar, eliminar y seleccionar datos en la vista. Todos los cambios introducidos en la vista serán trasladados a las tablas base, aunque hay algunas excepciones y restricciones para que esto se pueda llevar a cabo.

Una vista se utiliza para ocultar la complejidad de los datos. Se puede ofrecer a los usuarios acceso a una vista cuando la instrucción SQL que creó la vista es una compleja unión de múltiples tablas. De esta forma, los usuarios no tendrán que enfrentarse a la complejidad de una base de datos relacional. También se pueden utilizar vistas por motivos de seguridad.

2.3.9 Secuencias

Las definiciones de *secuencias* también se almacenan en el diccionario de datos. Las secuencias proporcionan una lista consecutiva de números unívocos que sirve para simplificar las tareas de programación.

La primera vez que una consulta llama a una secuencia, se devuelve un valor predeterminado. En cada consulta siguiente, se obtendrá un valor incrementado según el incremento especifi-

cado. Las secuencias pueden ser cíclicas o seguir creciendo hasta alcanzar un valor máximo especificado.

Cuando se utiliza una secuencia, no se ofrecen garantías de que se pueda generar una cadena de valores que no esté rota. Por ejemplo, si en una sesión busca el valor siguiente de una secuencia para utilizarlo en una sentencia `INSERT`, la suya es la única sesión que puede utilizar ese valor de secuencia. Si no confirma su transacción, el valor de secuencia no se insertará en la tabla y las inserciones posteriores utilizarán los valores siguientes de la secuencia.

2.3.10 Procedimientos

Un *procedimiento* es un bloque de instrucciones en lenguaje PL/SQL (*Procedural Language*) que se almacena en el diccionario de datos y al que pueden llamar las aplicaciones y usuarios. En la Sección 2.5 se explicará brevemente el lenguaje PL/SQL. Los procedimientos permiten almacenar dentro de la base de datos la lógica de las aplicaciones que se emplea con más frecuencia. Cuando se ejecuta el procedimiento, sus instrucciones se ejecutan como una unidad. Los procedimientos no devuelve ningún valor al programa que los llama.

Se pueden utilizar procedimientos que ayuden a forzar la seguridad de los datos. En lugar de conceder a los usuarios acceso directamente a las tablas de una aplicación, se les puede conceder la capacidad de ejecutar un procedimiento. Al ejecutarse un procedimiento, lo hará con los privilegios de su propietario, independientemente de quien lo llame (si tiene permiso para ejecutarlo). Los usuarios que usan el procedimiento, puede ser que no puedan acceder a las tablas, si no es por medio del procedimiento.

2.3.11 Funciones

Las *funciones*, al igual que los procedimientos, son bloques de código PL/SQL que se almacenan en la base de datos. A diferencia de éstos, las funciones pueden devolver valores al programa que las llama. Se pueden crear funciones e invocarlas desde las instrucciones SQL, de igual forma que se ejecutan las funciones que proporciona Oracle. Sólo se puede utilizar una función

definida por el usuario en una instrucción SQL si la función no modifica ninguna fila de la base de datos.

2.3.12 Paquetes

Los *paquetes* se pueden utilizar para organizar los procedimientos y las funciones en agrupaciones lógicas. Las especificaciones y el contenido de los paquetes se almacenan en el diccionario de datos. Los paquetes son muy útiles en las labores administrativas necesarias para gestionar los procedimientos y las funciones. También permiten la declaración de variables globales a una sesión, lo cual puede ser muy útil para algunas operaciones (como para solucionar el problema de las *tablas mutantes en los disparadores*).

2.3.13 Disparadores

Los *disparadores (o triggers)* son procedimientos que se ejecutan cuando se produce un suceso en la base de datos, sobre una tabla determinada. Pueden utilizarse para aumentar la integridad, imponer requisitos de seguridad adicionales o mejorar las opciones de auditoría disponibles.

Existen dos tipos de disparadores:

- Disparadores a nivel de instrucción: Se activan una vez por cada instrucción de disparo.
- Disparadores a nivel de fila: Se activan una vez por cada fila de una tabla afectada por la instrucción de disparo.

Por ejemplo, un disparador a nivel de instrucción se activa una sola vez para una sentencia `DELETE` que elimina 10.000 filas. Un disparador a nivel de fila se activaría 10.000 veces para esa misma sentencia.

Entre los sucesos de disparo se encuentran las operaciones `INSERT`, `UPDATE` y `DELETE`. Para cada tipo de disparador, puede crearse un disparador `BEFORE` *(antes)* y otro `AFTER` *(después)* para cada tipo de suceso de disparo.

Los disparadores a nivel de instrucción son útiles si el código del disparador no depende de los datos afectados. Por ejemplo, se puede crear un disparador de instrucción `BEFORE INSERT` en una tabla, para impedir que se efectúen operaciones de inserción en dicha tabla excepto durante determinados períodos de tiempo.

Los disparadores a nivel de fila son útiles si la acción del disparador depende de los datos afectados por la transacción. Por ejemplo, puede crearse un disparador de fila `AFTER INSERT` que introduzca filas nuevas en una tabla de auditoría.

A partir de Oracle8, se pueden crear disparadores `INSTEAD OF` sobre vistas. Un disparador `INSTEAD OF` se ejecuta en lugar de la acción que hizo que se iniciase. Es decir, si se creara un disparador `INSTEAD OF INSERT` en una vista objeto, el código del disparador se ejecutaría y la operación de inserción que hizo que se ejecutara el disparador no se produciría nunca. Si una vista combina varias tablas en una consulta, un disparador `INSTEAD OF` puede redirigir las acciones de Oracle en caso de que un usuario trate de actualizar las filas utilizando directamente la vista.

2.3.14 Sinónimos

Un sinónimo es un nombre alternativo para una tabla, una vista, una secuencia o una unidad de programa. Normalmente se utilizan los sinónimos por una serie de razones diferentes:

- Ocultar el nombre real del propietario del objeto de la base de datos.
- Ocultar la verdadera ubicación del objeto de la base de datos.
- Proporcionar un nombre para un objeto que sea menos complicado o más fácil de escribir.

Un sinónimo puede ser *privado* o *público*. Un sinónimo privado sólo se encuentra disponible para el usuario que lo ha creado, mientras que un sinónimo público se encuentra disponible en toda la base de datos.

2.3.15 Privilegios y roles

Antes, cuando se quería conceder a un usuario acceso a una aplicación, había que hacerlo tabla por tabla. Cada aplicación tenía un determinado conjunto de permisos dependiendo del usuario. Pronto, este método resultó ser un caos. Oracle creó el objeto de base de datos llamado *rol.*

Los Privilegios pueden ser de Objetos (para `INSERT`, `SELECT`, `UPDATE`, `DELETE`, `EXECUTE`...) o del Sistema (para crear tablas, vistas...). Los roles son conjuntos de privilegios que pueden concederse simultáneamente a un usuario.

Para otorgar roles y permisos (o privilegios) del sistema o de objetos a usuarios, se utiliza el comando `GRANT`. Los privilegios se retiran con el comando `REVOKE`.

Una forma alternativa de conceder privilegios, es mediante roles, se crea un rol de base de datos y se concede privilegios al rol y, finalmente, se asigna el rol al usuario.

2.3.16 Instantáneas o vistas materializadas

Las *vistas materializadas* se pueden utilizar para proporcionar copias locales de datos remotos a los usuarios o para almacenar datos duplicados en la misma base de datos. Una vista materializada se basa en una consulta que puede utilizar un enlace de base de datos para seleccionar datos desde una base de datos remota. Las vistas materializadas se pueden implementar para que sean de sólo lectura o actualizables. Para mejorar el rendimiento, se puede indexar la tabla que utiliza la vista materializada.

Dependiendo de la complejidad de la consulta base de la vista materializada, podría utilizar un *registro de vistas materializadas* (`LOG`) con el fin de mejorar el rendimiento de las operaciones de duplicación. Este tipo de operaciones se pueden llevar a cabo automáticamente, según la programación temporal que se especifique para cada vista materializada. Por defecto, las *vistas materializadas*, no se actualizan al modificar los objetos (tablas...) de los que se extrajo la información.

2.4 El diccionario de datos

El diccionario de datos de Oracle es un conjunto de tablas de sólo lectura que mantiene los meta-datos de una base de datos, es decir, la descripción de su esquema. Está compuesto por tablas de base que contienen los datos cifrados y almacenados por el sistema. Hay vistas accesibles por el usuario en el diccionario que descodifican, resumen y visualizan convenientemente la información a los usuarios. Los usuarios muy pocas veces tienen acceso a dichas tablas de base. Los prefijos especiales USER, ALL y DBA se usan respectivamente para hacer referencia a la vista del usuario (objetos del esquema que posee el usuario), vistas de objetos de otros usuarios expandidas (objetos para los que el usuario tiene autorización para acceder) y un conjunto completo de información (para uso del administrador de bases de datos). El diccionario de Oracle, que es un catálogo del sistema tiene el siguiente tipo de información:

- Nombres de los usuarios.
- Información de seguridad (privilegios y roles) sobre qué usuarios tienen acceso a qué información.
- Información sobre los objetos del esquema
- Restricciones de integridad
- Asignación de espacio y uso de los objetos de la base de datos.
- Estadísticas sobre los atributos, tablas y predicados.
- Información de auditoría sobre los accesos.

Es posible consultar el diccionario de datos usando *SQL*. Por ejemplo, la consulta:

```
SELECT object_name, object_type FROM user_objects;
```

Devuelve la información sobre los objetos del esquema que el usuario posee (nombre del objeto y el tipo del objeto, de cada objeto del usuario).

Además de la información anterior sobre el diccionario, Oracle constantemente está supervisando la actividad de la base de datos y la escribe en unas tablas llamadas *tablas dinámicas de rendimiento*. El administrador de bases de datos tiene acceso a esas tablas para supervisar el rendi-

miento del sistema y puede conceder permiso de acceso a algunos usuarios sobre las vistas de estas tablas.

También podemos encontrarnos en el catálogo del sistema información acerca de los tres niveles de esquemas de bases de datos: *externo* (definiciones de vistas), *conceptual* (tablas de base), e *interno* (almacenamiento y descripciones de índice).

2.5 Programación en PL/SQL

PL/SQL es el lenguaje de programación de Oracle, que es una extensión de SQL. PL/SQL ofrece a los programadores características de la ingeniería del software como el encapsulamiento de datos. La ocultación de la información, la sobrecarga y el manejo de excepciones.

PL/SQL es un lenguaje estructurado en bloques (basado en el lenguaje Ada). Es decir, las unidades básicas (procedimientos, funciones, bloques anónimos y bloques nominados) que constituyen un programa en PL/SQL son bloques lógicos que pueden contener cualquier número de sub-bloques anidados. Un bloque o sub-bloque agrupa declaraciones y sentencias relacionadas lógicamente. Las declaraciones son locales a los bloques y dejan de existir cuando el bloque termina. Según se puede ver más abajo, un bloque PL/SQL tiene tres partes: (1) una *parte declarativa* donde se declaran las variables y los objetos, (2) una *parte ejecutiva* donde se manipulan estas variables y (3) una *parte de excepciones* donde se procesan las excepciones y errores que aparezcan durante la ejecución.

```
[ DECLARE
     -- declaraciones ]
  BEGIN
     -- sentencias
[ EXCEPTION
     -- manipuladores ]
  END;
```

En la parte declarativa, que es opcional, se declaran las variables. Las variables pueden contener cualquier tipo de datos de SQL así como tipos de datos adicionales de PL/SQL. En esta sección también se les puede asignar valores a las variables. Los objetos se procesan en la parte ejecutable, que es la única que es obligatoria. Aquí los datos se procesan utilizando sentencias condicionales, iterativas o sentencias de control de flujo. La parte de excepciones maneja cualquier condición de error que aparezca en la parte ejecutiva. Las excepciones pueden ser cualquier error definido por el usuario o bien errores o excepciones de la base de datos. Cuando ocurre un error o una excepción, se detiene la ejecución normal, y se transfiere el control a la parte del control de excepciones del bloque o subprograma PL/SQL.

2.5.1 Cursores en PL/SQL

El conjunto de filas devueltas por una consulta puede estar compuesto por cero, una o varias filas, dependiendo de cuántas filas cumplan los criterios de búsqueda utilizados. Cuando una consulta devuelve varias filas, es necesario declarar explícitamente un cursor para procesar las filas. Un cursor es similar a una *variable de fichero* o *puntero de fichero*, que señala a una única fila (tupla) del resultado de la consulta. Hay que declarar los cursores en la parte declarativa y tenemos tres instrucciones para controlarlos: `OPEN`, `FETCH` y `CLOSE`. El cursor se inicia con una sentencia `OPEN`, que ejecuta la consulta, obtiene el conjunto de filas resultantes y sitúa el cursor en una posición antes de la primera fila resultado de la consulta. Esto convierte a esa fila en la fila en curso para el cursor. La sentencia `FETCH`, cuando se ejecuta por primera vez, introduce la primera fila en

las variables del programa y hace que el cursor apunte a esa fila. Las sucesivas ejecuciones de `FETCH` adelantan el cursor a la siguiente fila del conjunto resultado, situando dicha fila en las variables de programa. Esto es similar al método tradicional de procesar la información registro a registro. Cuando se haya procesado la última fila, se libera el cursor con la sentencia `CLOSE`.

Para obtener información complementaria sobre PL/SQL en general y cursores, puede consultar [LON02], [ELM02], [OWE96], [URM98].

2.6 Los distintos perfiles de usuarios

En una pequeña base de datos personal, como la lista de direcciones, lo normal es que una sola persona defina, construya y manipule la base de datos. En cambio, muchas personas participan en el diseño, utilización y mantenimiento de una base de datos grande (con algunos cientos de usuarios). En esta sección identificaremos a las personas cuyo trabajo requiere el empleo de una base de datos grande.

2.6.1 Administradores de bases de datos

En cualquier organización en la que muchas personas utilicen los mismos recursos se requiere un administrador jefe que supervise y gestione dichos recursos. En un entorno de bases de datos, el recurso primario es la propia base de datos, y el secundario es el SGBD y el software relacionado con él. La administración de estos recursos es responsabilidad del *administrador de bases de datos (ABD)*. El ABD se encarga de autorizar el acceso a la base de datos, de coordinar y vigilar su utilización y de adquirir los recursos de software y hardware que sean necesarios. El ABD es la persona responsable cuando surgen problemas como violaciones de la seguridad o una respuesta lenta del sistema. En las organizaciones grandes, el ABD cuenta con la ayuda de personal para poder desempeñar estas funciones.

2.6.2 Diseñadores de bases de datos

Los *diseñadores de bases de datos* se encargan de identificar los datos que se almacenarán en la base de datos y de elegir las estructuras apropiadas para almacenar dichos datos. Por lo general, estas tareas se realizan antes de que se implemente la base de datos y se carguen los datos. Los diseñadores tienen la responsabilidad de comunicarse con todos los futuros usuarios de la base de datos con el fin de comprender sus necesidades, y de presentar un diseño que satisfaga esos requerimientos. En muchos casos los diseñadores forman parte del personal del ABD y tal vez asuman otras responsabilidades una vez terminado el diseño de la base de datos. Casi siempre, los diseñadores interactúan con cada uno de los grupos de usuarios potenciales y desarrollan una vista de la base de datos que satisfaga los requerimientos de datos y de procesamiento de cada grupo. Después, se analizan las vistas y se integran con las de otros grupos de usuarios. El diseño final debe ser capaz de satisfacer las necesidades de todos los grupos.

2.6.3 Usuarios finales

Los usuarios finales son las personas cuyo trabajo requiere acceder a la base de datos para consultarla, actualizarla y generar informes; la base de datos existe principalmente para que ellos la utilicen. Hay varias categorías de usuarios finales:

- Los *usuarios finales ocasionales* acceden de vez en cuando a la base de datos, pero es posible que requieran información diferente en cada ocasión. Utilizan un lenguaje de consulta de base de datos avanzado (como *SQL*) para especificar sus solicitudes y suelen ser gerentes de nivel medio o alto u otras personas que examinan la base de datos ocasionalmente.
- Los *usuarios finales simples o paramétricos.* La función principal de su trabajo gira en torno a consultas y actualizaciones constantes de la base de datos, utilizando tipos estándar de consultas y actualizaciones, llamadas *transacciones programadas*, que se han programado y probado con mucho cuidado.
- Los *usuarios finales avanzados* pueden ser los ingenieros, científicos, analistas de negocios y otros, que están suficientemente familiarizados con los recursos del SGBD como para implementar sus aplicaciones de forma que cumplan sus complejos requerimientos.

- Los *usuarios autónomos* mantienen bases de datos personales mediante la utilización de paquetes de programas comerciales que cuentan con interfaces de fácil uso, basados en menús o en gráficos.

Normalmente los SGBD proporcionan múltiples recursos para acceder a la base de datos. Los usuarios finales simples necesitan aprender pocas cosas sobre los recursos proporcionados por el SGBD; solo necesitan entender los tipos de las transacciones estándar diseñadas e implementadas para que ellos las usen. Los usuarios ocasionales aprenden únicamente unos pocos recursos que pueden utilizar de forma repetida. Los usuarios avanzados intentan conocer la mayoría de los recursos del SGBD para satisfacer sus complejos requerimientos. Los usuarios autónomos normalmente adquieren gran habilidad para utilizar un paquete de software específico.

2.6.4 Analistas de sistemas y programadores de aplicaciones

Los *analistas de sistemas* determinan los requerimientos de los usuarios finales, sobre todo los de los simples o paramétricos y desarrollan especificaciones para transacciones programadas que satisfagan dichos requerimientos. Los *programadores de aplicaciones* implementan esas especificaciones en forma de propagandas, y luego prueban, depuran, documentan y mantienen estas transacciones programadas. Para realizar dichas tareas, los analistas y programadores (actualmente denominados *ingenieros de software*) deben conocer a la perfección toda la gama de capacidades del SGBD.

2.7 La arquitectura cliente-servidor y su relación con bases de datos distribuidas

En la actualidad, aún no se ha establecido exactamente cómo dividir la funcionalidad del SGBD entre el cliente y el servidor. Se han propuesto diferentes enfoques. Una posibilidad es incluir la funcionalidad de un SGBD centralizado en el nivel del servidor. Varios productos de SGBD relacionales han tomado este enfoque, en el que se proporciona a los clientes un *servidor SQL*. Entonces cada cliente debe formular las consultas SQL apropiadas y proporcionar una interfaz de usuario y funciones de interfaz del lenguaje de programación. Mientras que SQL es un estándar relacional, algunos servidores SQL, posiblemente proporcionados por diferentes vendedores, pueden aceptar instrucciones SQL. El cliente suele también referirse al diccionario de datos que incluye información de la distribución de los datos entre varios servidores SQL, así como módulos para descomponer una consulta global en varias consultas locales que pueden ejecutarse en varios sitios. La interacción entre el cliente y el servidor durante el procesamiento de una consulta SQL podría proceder de la siguiente manera:

1. El cliente analiza una consulta de usuario y la descompone en varias consultas a sitios independientes. Cada consulta se envía al correspondiente sitio de cada servidor.
2. Cada servidor procesa la consulta local y envía la relación resultante al cliente.
3. El cliente combina los resultados de las subconsultas para producir el resultado de la consulta original realizada.

En este enfoque, al servidor SQL también se le denomina *servidor de transacciones* (o máquina final también conocida como *back-end machine*), mientras que al cliente se le denomina *procesador de aplicaciones* (o máquina de la parte visible al usuario también conocida como *front-end machine*). La interacción entre el cliente y el servidor puede especificarla el usuario en el nivel cliente o mediante un módulo especializado del cliente del SGBD que es parte del paquete del SGBD.

En un SGBD típico, es usual dividir los módulos software en tres niveles:

1. El software del *servidor* es el responsable de la gestión de los datos locales en un sitio, al igual que el software del SGBD centralizado.
2. El software del *cliente* es el responsable de la mayoría de las funciones de distribución; accede a la información de la distribución de los datos que está en el catálogo del SGBD y procesa todas las peticiones que requieren acceso a más de un sitio. También maneja todas las interfaces de usuario.
3. El *software de comunicaciones* proporciona las primitivas de comunicación que utiliza el cliente para transmitir instrucciones y datos entre sitios necesarios. Esta parte proporciona servicios y primitivas esenciales de comunicación.

2.7.1 La arquitectura cliente-servidor

En la arquitectura cliente-servidor, el sistema de base de datos Oracle se divide en dos partes: (1) la máquina de la parte visible al usuario (máquina *front-end*) como parte cliente, y (2) la máquina del sistema final (máquina *back-end*) como parte servidor. La parte cliente es la aplicación de base de datos que interactúa con el usuario. El cliente no tiene la responsabilidad de acceso a datos y maneja meramente la petición, procesamiento y presentación de los datos gestionados por el servidor. La parte servidor ejecuta Oracle y maneja las funciones relativas al acceso compartido concurrente. Acepta sentencias SQL y PL/SQL originadas por aplicaciones del cliente, las procesa, y devuelve los resultados al cliente. Las aplicaciones cliente-servidor Oracle proporcionan transparencia de localización haciendo la localización de los datos transparente al usuario; algunas características como vistas, sinónimos y procedimientos contribuyen a proporcionar esa transparencia.

2.8 Instalación paso a paso de Oracle

El formato de distribución de Oracle (en este caso concreto la versión 8.1.5) es el CD puesto que ocupa un volumen considerable de espacio. La ocupación real en nuestro disco no será tan elevada, ya que hay elementos que no se tienen que instalar, sino, que pueden quedar en el CD, para poder consultarse, como por ejemplo la ayuda.

2.8.1 Requerimientos de Oracle

La ocupación en disco oscilará entre los 350 megabytes, para una instalación mínima, y los alrededor de 664 megabytes para una instalación típica. También tenemos la posibilidad de realizar una instalación personalizada, seleccionando los elementos a copiar a nuestro disco y controlando la ocupación final.

Los requerimientos de memoria de Oracle son los propios de una herramienta de este tipo, siendo el mínimo de 64 megabytes. En la práctica, 96 megabytes de memoria es una configuración adecuada, aunque lo recomendado son al menos 128 megabytes.

El software necesario puede ser Windows 95/98. Si usamos NT 4.0/2000 debemos adquirir la versión para esas plataformas, puesto que no hay compatibilidad entre distintas plataformas. Es muy recomendable tener instalados los últimos *Service Packs* en sistemas NT/2000.Oracle, también está disponible para plataformas Linux y UNIX. Para obtener más información sobre Oracle, puede visitar la web de Oracle[1].

2.8.2 Comenzando la instalación

Conociendo ya los aspectos básicos tratados en los puntos anteriores, vamos a iniciar la instalación de Oracle en nuestro sistema. Para ello bastará con insertar el CD-ROM en nuestro

[1] La página web de Oracle está disponible en www.oracle.com

lector, lo que causará la auto ejecución del programa de instalación. En caso de tener desactivada esta característica de Windows, habrá que ejecutar el programa *SETUP* que se encuentra en el directorio raíz del CD, lo cual se puede hacer cómodamente desde el *Explorador de Windows*.

Figura 2.2: Ventana del menú de inicio de instalación de la versión 8.1.5

Dependiendo de la versión que estemos instalando aparecerá una ventana similar a la que aparece en la Figura 2.2.

Ahora pulsamos el botón correspondiente a la instalación. Tras unos instantes, se verá aparecer una pantalla de bienvenida. Tras pulsar el botón *Siguiente*, podemos cambiar el directorio donde se instalará Oracle. Pulsamos de nuevo el botón *Siguiente*, lo que nos lleva donde se selecciona el tipo de instalación, en la que podrá elegir entre una instalación típica, completa o a medida.

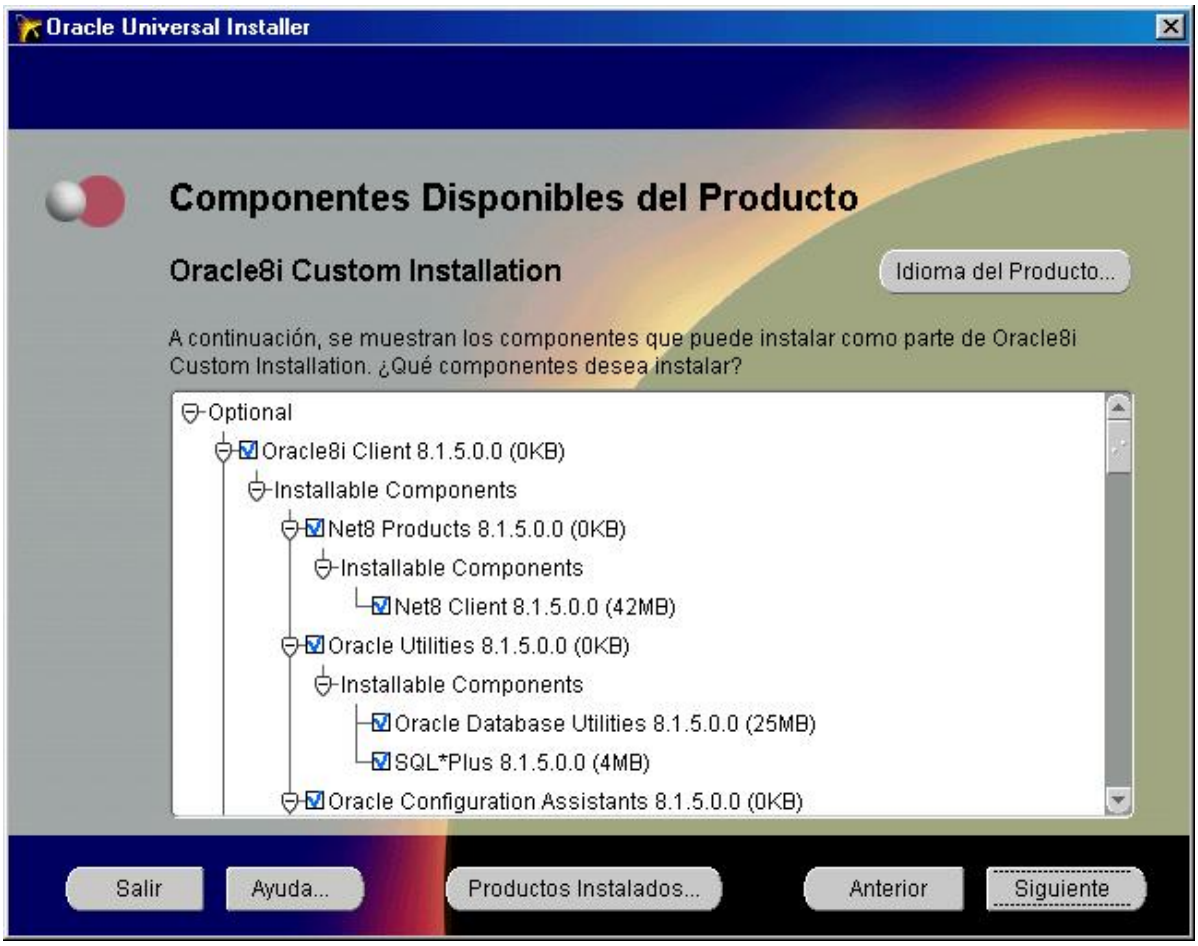

Figura 2.3: Selección de los elementos a instalar

La primera opción, instalación típica, copiará en nuestro sistema todos los archivos que necesita Oracle y las herramientas más habituales. La instalación compacta copiará en nuestro disco lo mínimo para poder funcionar. Por último, tenemos la instalación a medida, que nos permite seleccionar los elementos a instalar. Al seleccionar este tipo de instalación aparecerá una ventana con unos apartados principales. En la Figura 2.3 se puede ver cómo se ha abierto la ventana de componentes a instalar habiéndose seleccionado para instalación los archivos de programa principales, SQL*Plus, etc., dejando en el CD el resto de elementos.

Haciendo unas pulsaciones más en el botón *Siguiente* llegamos a una ventana en la que podremos ver una lista resumen con todas las opciones que se instalarán. Para iniciar el proceso de instalación, pulsamos el botón *Instalar*.

Cuando la instalación haya llegado a su fin, se verá aparecer en pantalla una ventana en la que se indica que es necesario reiniciar el ordenador. Debemos hacerlo antes de poder trabajar con Oracle, ya que de lo contrario no todos los componentes estarán adecuadamente configurados.

Capítulo

3

3 Delphi

Delphi se encuentra sin lugar a dudas entre los mejores entornos de desarrollo rápido de aplicaciones (RAD), con su potentísimo lenguaje Object Pascal, un compilador rapidísimo que nos permite crear ejecutables con una velocidad cercana al C++, y con múltiples posibilidades: bases de datos, multimedia, web, etc. No cabe duda que es un lenguaje del siglo XXI, con el permiso, claro está, de Java.

3.1 ¿Qué es Delphi?

Delphi es, por naturaleza, un compilador de *Pascal*. Es el siguiente paso en la evolución del mismo compilador de *Pascal* que *Borland* ha ido desarrollando desde que Anders Hejlsberg escribiera el primer compilador de *Turbo Pascal* hace más de quince años. A lo largo de los años, los programadores de *Pascal* han disfrutado de la estabilidad, la gracia, y, por supuesto, de la velocidad de compilación ofrecida por *Turbo Pascal*. Delphi no es una excepción: su compilador es la síntesis de más de una década de experiencia en compilación y es un compilador optimizado de 32 bits moderno. Aunque las posibilidades del compilador han aumentado considerablemente a lo largo de los años, la velocidad del compilador únicamente se ha visto ligeramente reducida. Además, la estabilidad del compilador de Delphi continúa siendo la regla por la que otros se miden.

Ahora haremos un breve recorrido en el tiempo, mientras repasamos cada una de las versiones de Delphi y vemos un poco del contexto histórico de cada una de ellas.

3.1.1 Delphi 1

En los primeros días del *DOS (Disk Operating System*, Sistema Operativo de Disco*)*, los programadores tenían que elegir entre un *BASIC (Beginner's All purpose Symbolic Instruction Code,* Código de Instrucción Simbólica Multipropósito para Principiantes*)* productivo pero lento, y un lenguaje ensamblador complejo pero eficiente. *Turbo Pascal*, que ofrecía la simplicidad de un lenguaje estructurado y el rendimiento de un verdadero compilador, acortó las diferencias. Los programadores de Windows 3.1 se encontraron con una disyuntiva similar: la elección entre un lenguaje potente pero pesado como *C++* y uno fácil de utilizar pero limitado como *Visual Basic.* Delphi 1 respondió a esa llamada ofreciendo un método totalmente diferente para el desarrollo Windows: desarrollo visual, ejecutables compilados, DLL, bases de datos, y todo lo demás; un entorno visual sin límites. Delphi 1 fue la primera herramienta de desarrollo Windows en combinar un entorno de desarrollo visual, un compilador de código nativo optimizado, y motor de acceso a la base de datos escalable, lo que definió la frase "desarrollo rápido de aplicaciones" (R*apid Application Development, RAD*).

La combinación del compilador, la herramienta *RAD*, y un acceso rápido a la base de datos eran razones demasiado convincentes para un montón de desarrolladores en *VB*, y Delphi ganó muchos adeptos. Además, muchos desarrolladores de *Pascal* redirigieron sus carreras y se pasaron a esta nueva herramienta. Se corrió la voz de que *Object Pascal* no era el mismo lenguaje que nos hacía sentir una mano atada a la espalda, y muchos mas desarrolladores se apuntaron a Delphi para aprovecharse de los sólidos métodos de diseño impuestos por el lenguaje y por la herramienta. El equipo de *Visual Basic* en *Microsoft*, que carecía de una competencia antes de Delphi, estaba totalmente desprevenido. Más lento, *Visual Basic 3* no era rival para Delphi 1.

3.1.2 Delphi 2

Un año más tarde, Delphi 2 proporcionaba los mismos beneficios bajo los modernos sistemas operativos de 32 bits Windows 95 y Windows NT. Además, Delphi 2 mejoraba la productividad con características adicionales y funcionalidad que no se encontraban en la primera versión, tales como un compilador de 32 bits que generaba aplicaciones rápidas, una biblioteca de objetos mejorada y ampliada, soporte mejorado de la base de datos, mejora en el manejo de cadenas de caracteres, soporte *OLE (Object Linking and Embedding, E*nlace e Incrustación de Objetos*), VFI (Visual Form Inheritance,* Herencia Visual de Formularios), y compatibilidad con proyectos Delphi de 16 bits. Delphi 2 se convirtió en el modelo con el que se comparaban el resto de herramientas *RAD*.

Esto ocurría en 1996, y el lanzamiento más importante para la plataforma Windows desde 3.0, Windows 95 de 32 bits, había tenido lugar a finales del año anterior. *Borland* estaba ansiosa por hacer que Delphi fuera la herramienta de desarrollo predominante para esa plataforma.

Microsoft intentó contraatacar con *Visual Basic 4*, pero se vió limitado por un bajo rendimiento, por su falta de portabilidad de 16 a 32 bits, y por fallos graves de diseño. Pese a ello hay un gran número de desarrolladores que por alguna razón continuaron utilizando *Visual Basic*. *Borland* también quería ver cómo Delphi penetraba en el mercado *cliente-servidor* de alto nivel, ocupado por herramientas tales como *PowerBuilder*, pero esta versión todavía no tenía la potencia necesaria para desbancar a esos productos de sus puestos.

3.1.3 Delphi 3

Durante el desarrollo de Delphi 1, al equipo de desarrollo Delphi sólo le importaba la creación de una herramienta de desarrollo revolucionaria. Para Delphi 2, el equipo de desarrollo estaba ocupado en las tareas de pasar de 32 bits (mientras se mantenía una casi completa compatibilidad hacia atrás) y se añadían nuevas características de base de datos y *cliente-servidor* requeridas por las empresas de Tecnología de la Información (*IT)*. Mientras se creaba Delphi 3, el equipo de desarrollo tuvo la oportunidad de ampliar el conjunto de herramientas para proporcionar un nivel extraordinario de extensión y profundidad de las soluciones a algunos de los problemas más complicados a los que se enfrentaban los desarrolladores Windows. En particular, Delphi 3 facilitó la utilización de tecnologías complejas como *COM* y *ActiveX*, el desarrollo de aplicaciones *World Wide Web*, aplicaciones *thin client* y arquitecturas de bases de datos *multitier (bases de datos distribuidas y reglas de negocio hacia los clientes)*. *CodeInsight* de Delphi 3 facilitó que el proceso de escritura del código fuera un poco más sencillo, aunque la metodología básica, seguía siendo la misma.

Estábamos en 1997 y la competencia también estaba haciendo algunos progresos interesantes. A bajo nivel, *Microsoft* finalmente comenzó a hacer algo bien con *Visual Basic 5*, el cual incluía un compilador para solucionar algunos problemas de rendimiento que venían de lejos, un bien soporte *COM / ActiveX*, y algunas nuevas características de plataforma claves.

3.1.4 Delphi 4

Delphi 4 se centró en hacer más fácil el desarrollo en Delphi. *Module Explorer* se introdujo en Delphi, permitiendo ver y editar unidades desde un interfaz gráfica cómoda. Una nueva navegación de código y características de finalización de clases permitían centrarse en el desarrollo de aplicaciones minimizando las tareas engorrosas. Se rediseño el *IDE (Integrated Development Environment,* Entorno de Desarrollo Integrado*)* con barras de herramientas y ventanas anclables para que el desarrollo fuera más cómodo, y se mejoró mucho el depurador. Delphi 4 extendió el alcance del producto a la empresa.

Esto ocurría en 1998, y Delphi había asegurado eficazmente su posición frente a la competencia. La vanguardia se había estabilizado de alguna forma, aunque Delphi siguió lentamente ga-

nado cuota de mercado. *CORBA* era el tema de moda en la industria, y Delphi lo incluía y la competencia no. También Delphi 4 tenía su parte negativa: tras disfrutar de varios años siendo la herramienta de desarrollo más estable del mercado, Delphi 4 se había ganado una reputación entre los usuarios de no estar a la altura de ofrecer una ingeniería y estabilidad sólidas.

3.1.5 Delphi 5

Delphi 5 supone un avance en algunos frentes: en primer lugar, Delphi 5 continúa lo que empezó Delphi 4, añadiendo muchas características para que las tareas que normalmente tardan mucho tiempo sean más fáciles, permitiendo concentrarse en lo que se desea escribir y tener que pensar menos en cómo escribirlo. Estas nuevas características de productividad incluyen mejoras en el *IDE* y en el depurador, software de desarrollo de equipos *TeamSource*, y herramientas de traducción. En segundo lugar Delphi 5 contiene un conjunto de características nuevas destinadas a facilitar el desarrollo en Internet, convirtiéndolo en una plataforma de datos muy versátil. Para terminar, la característica mas importante de Delphi 5 es la estabilidad.

3.1.6 Delphi 6

Delphi 6 al igual que en las anteriores versiones, sigue siendo una herramienta estupenda para el desarrollo de aplicaciones Windows. Una de las características más importantes de esta nueva versión es la portabilidad a otros sistemas operativos como *Linux* haciendo posible un desarrollo multiplataforma, con mayor posibilidad de reutilización de componentes. También se ha mejorado el *IDE* añadiendo mas funcionalidades y multitud de nuevos componentes. Igualmente se han añadido características interesantes de diseño de diagramas orientado a objetos, diseño de tablas de bases de datos, consultas y otros muchos elementos.

3.2 Características

Delphi, está disponible en diversas variedades destinadas, a cubrir distintas necesidades: Delphi *Standard*, Delphi *Professional* y Delphi *Enterprise*. Cada una de estas versiones está destinada a un tipo de desarrollador diferente.

Delphi *Standard* es la versión para principiantes. Proporciona todo lo necesario para comenzar a escribir aplicaciones en Delphi, y es ideal para aficionados y estudiantes que deseen comenzar a programar en Delphi de forma económica. Esta versión incluye las siguientes características:

- Compilador *Object Pascal* con optimización para 32 bits.
- *Visual Component Library (VCL)*, que incluye alrededor de 85 componentes estándar en la Paleta de componentes.
- Soporte de Paquetes, permitiéndole crear pequeños ejecutables y bibliotecas de componentes.
- Un *IDE* que incluye un editor, depurador, diseño de formularios y un conjunto de características de productividad. El diseñador de formularios soporta la herencia y enlazado de formularios visuales.
- Delphi 1, incluido para desarrollo de aplicaciones Windows de 16 bits.
- Soporte completo para el *API Win32*, incluyendo *COM, GDI, DirectX, multithread*, y varios kits de desarrollo de *Microsoft* y otros kits de desarrollo de otras empresas (*SDK*).

Delphi *Profesional* está destinado a los desarrolladores profesionales que no precisan de las características *cliente-servidor*. También es el indicado para la creación y distribución de aplicaciones o componentes Delphi, este producto es el indicado para usted. La edición *Profesional* incluye lo mencionado anteriormente para la edición *Stantard*, además de lo siguiente:

- Más de 150 componentes *VCL* en la Paleta de componentes.
- Soporte de base de datos, incluyendo controles relativos a *VCL*, *Borland Database Engine (BDE)*, controladores *BDE* para las tablas locales, una arquitectura de conjuntos de datos virtuales que le permite incorporar otros motores de base de datos en *VCL*, la herramienta

Database Explorer, un repositorio de datos, soporte *ODBC* y componentes nativos *InterBase Express*.

- Asistentes para la creación de componentes *COM*, tales como controles *ActiveX*, *ActiveForms*, servidores *Automation* y páginas de propiedad.
- La herramienta de generación de informes *QuickReports* para la integración de informes a medida en sus aplicaciones.
- Los componentes de gráficos y diagramas de *TeeChart* para la representación de datos.
- Un *Local InterBase Server (LIBS)* monousuario, que le permite realizar desarrollos *cliente-servidor* basados en *SQL* sin necesidad de estar conectado a una red.
- La característica *Web Deployment* para facilitar la distribución de contenido *ActiveX* a través de la Web.
- La herramienta para la instalación de aplicaciones *InstallShield Express.*
- La *API OpenTools* para el desarrollo de componentes fuertemente integrados en el entorno Delphi, así como una interfaz para el control de versiones *PVCS*.
- Asistentes *WebBroker* y *FastNet* y componentes para el desarrollo de aplicaciones para Internet.
- Código fuente de la *Visual Component Library (VCL)*, biblioteca en tiempo de ejecución (*RTL*), y editores de propiedades.
- La herramienta *WinSight32* para información mediante ventanas y mensajes.

Delphi *Enterprise* está dirigido a desarrolladores de alto nivel y de aplicaciones *cliente-servidor* corporativas. Es la versión indicada para desarrollar aplicaciones que se comunican con servidores de bases de datos *SQL*, esta edición contiene todas las herramientas necesarias para guiar al programador a lo largo del ciclo de desarrollo de aplicaciones *cliente-servidor*. La versión *Enterprise* incluye todo lo descrito anteriormente para las otras dos ediciones de Delphi, así como lo siguiente:

- Alrededor de 200 componentes *VCL* en la Paleta de componentes.
- Soporte y licencia de desarrollo *MIDAS (Multitier Distributed Applicaction Services)*, que proporciona un nivel de sencillez sin precedentes para el desarrollo de aplicaciones *multi-tier.*
- Soporte *CORBA*, incluyendo *ORB VisiBroker.*
- Componentes *XML InternetExpress.*

- Software de control de fuente *TeamSource*, que permite el desarrollo en equipo y soporta varios motores de versiones (incluidos *ZIP* y *PVCS*).
- Soporte nativo de *Microsoft SQL Server 7*.
- Soporte avanzado para *Oracle8*, incluyendo campos de tipo abstracto de datos.
- Soporte directo de *ADO (ActiveX Data Objects,* Objetos de datos ActiveX).
- Componentes *DecisionCube*, que proporcionan un análisis visual y multidimensional de los datos (se incluye el código fuente).
- Controladores *SQL Links BDE* para *InterBase, Oracle, Microsoft SQL Server, Sybase, Informix* y servidores de base de datos *DB2*, así como una licencia para la redistribución ilimitada de dichos controladores.
- *SQL Database Explorer*, que le permite visualizar y editar metadatos específicos del servidor.
- Herramienta de construcción gráfica de consultas *SQL Builder*.
- *SQL Monitor*, que le permite ver comunicaciones *SQL* desde y hacia el servidor, de forma que pueda depurar y ajustar el rendimiento de su aplicación *SQL*.
- *Data Pump Expert* para la modificación rápida de tamaño.
- Licencia para cinco usuarios de *InterBase* para *Windows NT*.

3.3 ¿Por qué Delphi?

La razón más importante por la que hemos escogido Delphi como entorno de desarrollo, es la *productividad*. Las razones combinadas que hacen que Delphi sea tan productivo, las podríamos condensar como sigue:

- La calidad del entorno del entorno de desarrollo visual.
- La velocidad del compilador frente a la eficiencia del código compilado.
- La potencia del lenguaje de programación frente a su complejidad.

- La flexibilidad y la escalabilidad de la arquitectura de la base de datos que incorpora Delphi, comúnmente conocida como BDE (Borland Database Engine, Motor de Bases de datos de Borland).
- Los métodos de diseño y de utilización recomendados por el entorno.
- Prototipos turbo.

Aunque probablemente habrá otros muchos factores, como temas de utilización, documentación o soporte de otros productos, etc. hemos descubierto que este modelo simplista explica con bastante precisión porqué elegimos Delphi. Algunas de estas categorías también incluyen algo de subjetividad, pero esa es la cuestión. Así que ahora vamos a estudiar un poco más en profundidad las características anteriores.

3.3.1 La claridad del entorno de desarrollo visual

El entorno de desarrollo visual puede dividirse generalmente en tres componentes: el editor, el depurador y el diseñador de formularios. Al igual que la mayoría de las herramientas de desarrollo rápido de aplicaciones (*RAD*, Rapid Application Development), estos tres componentes funcionan en armonía mientras diseña una aplicación. Mientras trabaja con el diseñador de formularios, Delphi está generando código en segundo plano para los componentes que arrastramos y manipulamos en los formularios. Podemos añadir código adicional en el editor para definir el comportamiento de la aplicación, y se puede depurar la aplicación desde el propio editor añadiendo puntos de ruptura, temporizadores u otros mecanismos.

El editor Delphi generalmente es equiparable a los de las demás herramientas. La tecnología *CodeInsight*, la cual ahorra gran parte de la escritura, es probablemente la mejor. Se basa en la información del compilador, en lugar de información del tipo biblioteca como *Visual Basic*, y, por tanto, es capaz de ayudar en un mayor número de situaciones. Aunque el editor de Delphi tiene buenas opciones de configuración, el editor de *Visual Studio (suite de programación de Microsoft)* es más configurable.

El depurador de Delphi finalmente se ha puesto a la altura del depurador presentado en *Visual Studio*, con propiedades avanzadas como depuración remota, adherencia a procesos, depu-

ración de DLL y de paquetes, temporizadores locales automáticos, y una ventana de CPU. Delphi también tiene un soporte IDE agradable para la depuración, al permitir que las ventanas se sitúen y se anclen donde se quiera durante la depuración y que el estado se guarde como una configuración de escritorio. Una muy buena característica del depurador, habitual en los entornos interpretados como *Visual Basic* y algunas herramientas *Java*, es la posibilidad de cambiar el código para modificar el comportamiento de las aplicaciones mientras se depura la aplicación. Por desgracia, este tipo de característica es mucho más difícil de lograr cuando se compila en código nativo y, por tanto, no está soportado por Delphi.

Un diseñador de formularios es por lo general una característica única de las herramientas *RAD*, como Delphi, *Visual Basic*, *C++ Builder* y *PowerBuilder*. Entornos de desarrollo más clásicos, como *Visual C++* y *Borland C++*, generalmente proporcionan editores de diálogo, pero aquellos tienden a no estar integrados en el flujo de trabajo de diseño como los diseñadores de formularios. La falta de un diseñador de formularios realmente tiene un efecto negativo en la productividad general de la herramienta para el desarrollo de aplicaciones. A lo largo de los años, Delphi y Visual Basic se han enfrentado en un tira y afloja de características de diseño de formularios, con cada nueva versión, sobrepasando al otro en funcionalidad. Una propiedad del diseñador de formularios de Delphi que lo diferencia del resto es el hecho de que Delphi está construido encima de un verdadero entorno orientado a objetos. Debido a ello, los cambios que efectúa a las clases base se propagarán hasta cualquier clase ancestro. Una característica clave que mejora esto es la *VFI* (Visual Form Inheritance, Herencia Visual de Formularios). La *VFI* le permite descender dinámicamente desde cualquiera de los restantes formularios de su proyecto o del repositorio (formularios en uso del proyecto). Más aún, los cambios efectuados en el formulario base del que desciende se propagarán en cascada y se reflejarán en sus descendientes.

3.3.2 La velocidad del compilador frente a la eficiencia del código compilado

Una compilación rápida le permite desarrollar software de forma incremental, posibilitando que la realización de cambios frecuentes en su código fuente, recompilar, probar, cambiar, recompilar, probar de nuevo, sea un ciclo de desarrollo muy eficiente. Cuando la velocidad de compilación es menor, los desarrolladores se ven obligados a introducir cambios en el código en bloques, haciendo modificaciones múltiples antes de compilar y adaptándose a un ciclo de desarrollo menos

eficiente. La ventaja de la eficiencia del tiempo de ejecución es evidente; *siempre es mejor que los tiempos de ejecución sean más rápidos y los ejecutables más pequeños*.

Quizá la característica más conocida del compilador de *Pascal* en el que se basa Delphi es su velocidad. De hecho, probablemente sea el compilador de código nativo del lenguaje de alto nivel más rápido para Windows. *C++*, que tradicionalmente ha ido a paso de tortuga en cuanto a la velocidad de compilación se refiere, ha hecho grandes esfuerzos en los últimos años utilizando el enlace incremental y varias estrategias de caché que pueden encontrarse en *Visual C++* y en *C++ Builder* en particular. Pese a ello, incluso estos compiladores C++ son típicamente varias veces más lentos que el compilador de Delphi.

¿Significa toda esta velocidad en el tiempo de compilación una reducción de la eficiencia en el tiempo de ejecución? La respuesta es un no rotundo. Delphi comparte el compilador *back end* con el compilador de *C++ Builder*, por lo que la eficiencia del código generado está a la par de la de un muy buen compilador de *C++*. En las últimas comparativas fiables, *Visual C++* en realidad quedó en primer lugar en velocidad y eficiencia del código en muchos casos, gracias a algunas muy buenas optimizaciones. Aunque estas pequeñas ventajas pueden pasar desapercibidas para el desarrollo general de aplicaciones, pueden suponer realmente una diferencia si escribe código con mucha capacidad de computación.

Visual Basic es algo singular respecto a la tecnología de compilación. Durante el desarrollo, *VB* trabaja en modo interpretado y responde bastante bien. Cuando se desea hacer la distribución, puede llamar al compilador de *VB* para generar el EXE. Este compilador es realmente lento y su eficiencia en cuanto a velocidad se sitúa bastante por debajo de las herramientas Delphi y *C++*.

Java es otro caso interesante. Las herramientas basadas en *Java* de alto nivel, como *JBuilder* y *Visual J++*, presumen de tener tiempos de compilación cercanos a los de Delphi. La eficiencia de velocidad en tiempo de ejecución, sin embargo, deja algo que desear, porque *Java* es un lenguaje interpretado (en cuanto a su maquina virtual se refiere). Aunque *Java* sigue mejorando a buen ritmo, la velocidad en tiempo de ejecución en la mayoría de los casos está muy por debajo de la de Delphi y *C++*.

3.3.3 La potencia del lenguaje de programación frente a su complejidad

La potencia y la complejidad dependen del cristal con que se mira, y esta categoría en particular ha servido como pretexto para muchas discusiones en Internet. Lo que le resulta fácil a una persona le parecerá difícil a otra, y lo que es una limitación para una puede ser considerado algo elegante por otra. Por tanto, lo que viene a continuación se basa en la experiencia del autor y en sus preferencias personales.

El *ensamblador* es el lenguaje definitivo en cuanto a potencia. Hay pocas cosas que no se puedan hacer en *ensamblador*. Sin embargo, la escritura de incluso la más sencilla de las aplicaciones para Windows en *ensamblador* es una aventura ardua y sujeta a fallos. Y no sólo eso, sino que a veces es prácticamente imposible mantener una base de diseño en código *ensamblador* en un equipo al cabo de un tiempo. Cuando el código pasa de una persona a la siguiente, las ideas y las posibilidades de diseño se vuelven cada vez más turbias, hasta que el código se torna poco comprensible. Por tanto, daremos una baja puntuación al *ensamblador* en esta categoría, ya que, aunque potente, el lenguaje *ensamblador* es demasiado complejo para casi cualquier tarea de desarrollo de aplicaciones.

C++ es otro lenguaje extremadamente potente. Con la ayuda de características realmente potentes como macros de preprocesador, plantillas, sobrecarga de operadores y demás, es casi posible diseñar su propio lenguaje con *C++*. Si utiliza la enorme cantidad de características que se encuentran a su disposición, puede desarrollar código claro y de fácil mantenimiento. El problema, sin embargo, es que muchos desarrolladores no pueden resistirse a abusar de estas funciones, y es muy fácil crear un código realmente horrible. De hecho, es más fácil escribir mal código en *C++* que buen código, debido a que el lenguaje no se presta por sí mismo a hacer un buen diseño: depende mucho del desarrollador.

Hay dos lenguajes que creemos que son muy parecidos en cuanto al equilibrio que mantienen entre complejidad y potencia. Estos lenguajes son *Object Pascal* y *Java*. Ambos siguen el método de limitar las características disponibles con el fin de que el diseñador siga un diseño lógico. Por ejemplo, ambos evitan una noción realmente orientada a objeto pero de la que es muy fácil abusar, que es la herencia múltiple, mientras permiten que una clase implemente interfaces múltiples. Ambos carecen de la ingeniosa, pero no por ello exenta de peligro, función de la sobrecarga

de operadores. Además, ambos hacen que los archivos fuente sean excelentes en el lenguaje, en lugar de ser un detalle del que debe ocuparse el enlazador. Lo que es más, ambos lenguajes se aprovechan de potentes características que dan más por menos, como la *manipulación de excepciones*, *RTTI* (Runtime Type Information, Información de Tipos en Tiempo de ejecución), y *cadenas de caracteres nativas gestionados por la memoria*. No es una casualidad que ambos lenguajes no hayan sido escritos por comités, sino que han sido criados por un individuo o por un pequeño grupo en una única organización, con un entendimiento común de cómo debería ser el lenguaje.

Visual Basic dio sus primeros pasos como un lenguaje diseñado para ser lo suficientemente sencillo para que los recién llegados a la programación se pusieran en marcha rápidamente (de aquí su nombre BASIC, Begginner's All-Purpose Symbolic Instruction Code, Código de Instrucción Simbólica Multipropósito para Principiantes). Sin embargo, al ir añadiendo características de lenguaje a lo largo del tiempo para mejorar sus deficiencias, *Visual Basic* se ha hecho cada vez más complejo. En un esfuerzo de ocultar los detalles a los desarrolladores, *Visual Basic* todavía mantiene algunos obstáculos que hay que evitar a la hora de construir proyectos complejos.

3.3.4 La flexibilidad y la escalabilidad de la arquitectura de la base de datos

Debido a la falta de una agenda de base de datos de Borland, Delphi mantiene lo que en nuestra opinión es una de las arquitecturas de base de datos más flexibles de cualquier herramienta. Nada más sacarlo de la caja, el *BDE* funciona estupendamente y da un buen rendimiento para casi todas las aplicaciones en una amplia gama de plataformas locales, de cliente-servidor y plataformas de bases de datos *ODBC*. También se pueden utilizar los nuevos componentes nativos *ADO* en lugar del *BDE*. Si *ADO* no es de su agrado, puede escribir su propia clase de acceso a datos elevando la arquitectura del conjunto abstracto de datos, o adquirir una solución de conjunto de datos de otra empresa. Además, *MIDAS* le facilita realizar una división física o lógica, en múltiples *tier*, para el acceso a cualquiera de estas fuentes de datos.

Las herramientas de *Microsoft* lógicamente tienden a centrarse en las propias bases de datos y en las soluciones de acceso a los datos de *Microsoft*, tanto si son *ODBC, OLE DB* u otras.

3.3.5 Los métodos de utilización y diseño recomendados por el marco de trabajo

Éste es el punto, que se echa de menos en otras herramientas. Habiendo igualdad en el resto, *VCL* es la parte más importante de Delphi. La posibilidad de manipular componentes durante el diseño, diseñar componentes, y heredar el comportamiento de otros componentes mediante técnicas orientadas a objeto (OO), es un ingrediente fundamental para el nivel de productividad de Delphi. Cuando se escriben componentes *VCL*, no es posible evitar el empleo de metodologías sólidas de diseño OO en muchos casos. Por el contrario, otros entornos basados en componentes son a menudo demasiado rígidos o demasiado complicados. Los controles *Actives*, por ejemplo, proporcionan muchos beneficios análogos a los controles *VCL*, pero no hay ninguna forma de heredar desde un control *Actives* para crear una nueva clase con algunos comportamientos diferentes. Los entornos de clases tradicionales, como *OWL* y *MFC*, generalmente requieren tener un buen conocimiento del marco interno para ser productivos, y están limitados por una falta de soporte en tiempo de diseño de herramientas del tipo *RAD.* Una herramienta que coincide en las características con *VCL* es *Windows Foundation Classes (WFC)* de *Visual J++*. Sin embargo, el futuro de *Visual J++* es todavía incierto debido a una demanda pendiente iniciada por *Sun Microsystems* debido a temas de *Java*.

3.3.6 Prototipos turbo

Tras utilizar Delphi durante algún tiempo, hemos observado que la curva de aprendizaje es especialmente suave. De hecho al escribir el primer proyecto en Delphi, se observan resultados inmediatos por medio de un ciclo de desarrollo corto y una aplicación robusta. Delphi destaca en la única faceta del desarrollo de aplicaciones que ha sido la pesadilla de muchos programadores *Windows*: el diseño de la *Interfaz de usuario (GUI, Graphical User Interface,* Interfaz de Usuario Gráfica*).*

A veces, el diseño de la *IU* y el desarrollo general de un programa se denomina *prototipado*. En un entorno no visual, el *prototipado* de una aplicación a menudo lleva más tiempo que escribir la implementación de la aplicación, o lo que se denomina el *back end*. Por supuesto, el *back end* de una aplicación es el objetivo fundamental del programa, por otra parte, una interfaz de usua-

rio intuitiva y agradable a la vista es una buena parte de la aplicación, pero, aun más es importante la funcionalidad.

Delphi permite utilizar sus controles a medida para montar unas interfaces de usuario estupendas en poco tiempo. De hecho, verá que cuando se encuentre a gusto con los formularios de Delphi, los controles y los métodos de respuesta a eventos, ahorrará un montón de tiempo de desarrollo de prototipos de aplicación. También verá que las interfaces de usuario que desarrolle en Delphi tendrán un aspecto tan bueno, si no mejor, que el de las aplicaciones desarrolladas con herramientas tradicionales. A menudo, lo que se "simula" en Delphi resulta ser el producto final.

Si desea saber más sobre las características de Delphi, puede consultar **[**TEI00**]**, **[**MAR97**]**, **[**W3MAR**]**, **[**W3CAN**]**.

3.4 Instalación paso a paso

El formato de distribución de Delphi (en este caso concreto la versión 6) es el CD, en el cual se incluye *Delphi 6*, *InterBase Server*, *InstallShield*, *TeamSource*, los archivos imagen que permiten ejecutar todo ello desde el propio CD y múltiples archivos con información diversa. La ocupación real en nuestro disco no será tan elevada, ya que no tenemos por qué instalar todas las opciones, podemos dejar las partes que nos interesen en el CD, sin instalarlas en nuestro sistema.

3.4.1 Requerimientos de Delphi

Como se ha comentado anteriormente, el contenido del CD en el que se distribuye Delphi ocupa un volumen considerable de espacio, pero es cierto también que algunos elementos no tienen porqué ser copiados a nuestro sistema, como los manuales, que pueden visualizarse desde el propio CD, o el código fuente de la VCL y los ejemplos.

La ocupación en disco oscilará entre los 100 megabytes, para una instalación típica de la versión *Standard*, y los alrededor de 300 megabytes para una instalación de la versión *Enterprise*.

La instalación mínima, que copiará al disco tan sólo lo necesario dejando el resto de archivos en el CD, tiene el inconveniente de que cada vez que necesitemos una de las herramientas no instaladas, el CD deberá encontrarse en la unidad. También tenemos la posibilidad de realizar una instalación personalizada, seleccionando los elementos a copiar a nuestro disco y controlando la ocupación final.

Los requerimientos de memoria de Delphi son los propios de una herramienta de este tipo, siendo el mínimo de 32 megabytes. Este mínimo, sin embargo es poco realista, ya que con esa cantidad de memoria los accesos a disco durante procesos como la compilación serán continuos y el funcionamiento de Delphi resultará algo lento. En la práctica, 64 megabytes de memoria es una configuración adecuada, aunque lo recomendado son al menos 128 megabytes.

El software necesario será Windows 95/98 o NT 4.0/2000. Si usamos NT 4.0, deberemos tener instalados los últimos *Service Packs.*

3.4.2 Comenzando la instalación

Conociendo ya los aspectos básicos tratados en los puntos anteriores, vamos a iniciar la instalación de Delphi en nuestro sistema. Para ello bastará con insertar el CD-ROM en nuestro lector, lo que causará la auto ejecución del programa de instalación. En caso de tener desactivada esta característica de Windows, habrá que ejecutar el programa *INSTALL* que se encuentra en el directorio raíz del CD, lo cual se puede hacer cómodamente desde el *Explorador de Windows*.

Dependiendo de la versión que estemos instalando aparecerá una ventana con más o menos opciones. En la Figura 3.1, se puede ver el aspecto que muestra la ventana de instalación de la versión *Enterprise.*

Figura 3.1: Ventana inicial del programa de instalación

Ahora pulsamos el botón correspondiente a la instalación de Delphi. Tras unos instantes, se verá aparecer una pantalla de bienvenida. Tras pulsar el botón *Next*, tendremos que introducir el número de serie y la clave de activación del producto, datos que encontrará en la propia funda del CD.

Una nueva pulsación del botón *Next* y se visualizará el acuerdo de licencia del producto, que habrá que aceptar pulsando el botón *Yes*. A continuación aparecerá el archivo con notas informativas de última hora. Pulsamos de nuevo el botón *Next*, lo que nos lleva donde se selecciona el tipo de instalación, en la que podrá elegir entre una instalación típica, completa o a medida.

La primera opción, instalación típica, copiará en nuestro sistema todos los archivos que necesita Delphi y las herramientas más habituales que le acompañan para funcionar en una forma independiente, sin necesidad de disponer del CD. La instalación compacta copiará en nuestro disco lo mínimo para poder funcionar. Por último tenemos la instalación a medida, que nos permite seleccionar los elementos a instalar. Al seleccionar este tipo de instalación aparecerá una ventana con unos apartados principales. En la Figura 3.2 se puede ver cómo se ha abierto la ventana de componentes *Program Files* habiéndose seleccionado para instalación los archivos de programa principales, el editor de imágenes, etc., dejando en el CD los programas de ejemplo

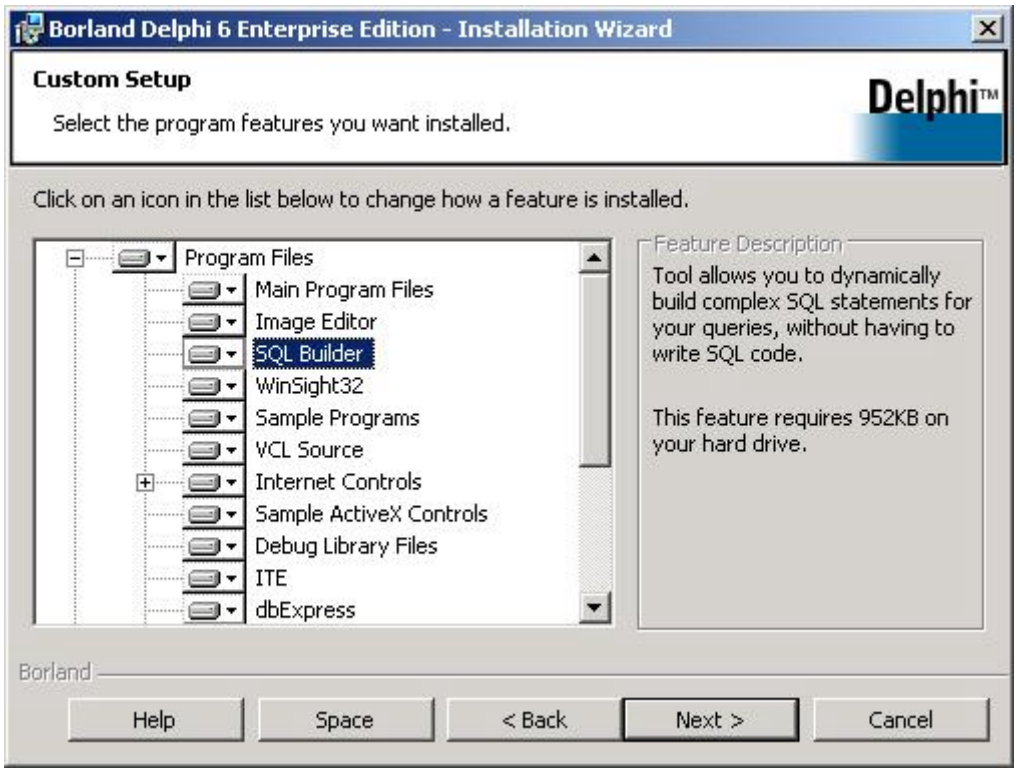

Figura 3.2 Selección de los elementos a instalar

Independientemente del método de instalación que se haya elegido en la siguiente ventana que aparece, tras pulsar una vez más el botón *Next*, indicaremos si queremos o no instalar el cliente *InterBase*, que nos servirá para hacer pruebas locales de desarrollo con bases de datos *InterBase*.

Después de seleccionar las carpetas y los grupos de inicio, haciendo una pulsación más del botón *Next* llegamos a una ventana en la que podremos ver una lista con todas las opciones de configuración. Para iniciar el proceso de instalación, pulsamos el botón *Install*.

Cuando la instalación haya llegado a su fin, se verá aparecer en pantalla una ventana en la que se indica que es necesario reiniciar el ordenador. Debemos hacerlo antes de intentar ejecutar Delphi, ya que de lo contrario no todos los componentes estarán adecuadamente configurados.

Capítulo

4

4 Componentes de programación

Los componentes son la piedra angular de la programación en Delphi. Aunque la mayoría de los componentes representan partes visibles de la interfaz de usuario, existen también componentes no visuales.

Los componentes hacen fácil la programación en Delphi. En vez de tener que operar a nivel de unidades, el usuario simplemente tiene que pinchar en un componente y situarlo en la posición deseada de su formulario. Eso es todo: Delphi se encarga de lo demás.

4.1 Un poco de teoría básica

La *VCL* (*Visual Component Library*, Biblioteca de Componentes Visuales), está diseñada específicamente para trabajar en el entorno visual de Delphi. En lugar de crear una ventana o un cuadro de diálogo, y programar su comportamiento mediante código, se modifican las características de comportamiento o el aspecto visual del componente, a medida que se diseña el programa visualmente.

El grado de conocimiento de la VCL necesario depende, en realidad, de cómo se utilice. Hay que puntualizar que existen dos tipos de programadores en Delphi: programadores de aplicaciones y diseñadores de componentes visuales. Los programadores de aplicaciones desarrollan aplicaciones concretas interactuando con el entorno visual de Delphi. Por otro lado, los diseñadores de componentes, implementan bloques funcionales que se utilizan para construir aplicaciones.

4.1.1 ¿Qué es un componente?

Los componentes son bloques constructivos que utilizan los programadores para diseñar la interfaz de usuario y proporcionar algunas capacidades no visuales a sus aplicaciones. Por lo que se refiere a programadores de aplicaciones, un componente es algo que el programador toma de la paleta de componentes para colocarlo en sus formularios. A partir de ahí, pueden manipular las distintas propiedades y añadir manejadores de eventos para dar al componente una apariencia o un comportamiento específicos. Desde el punto de vista de un diseñador de componentes, los componentes son objetos en código *Object Pascal*. Estos objetos pueden encapsular el comportamiento de elementos proporcionados por el sistema (tales como los controles estándar de Windows 95/98). Otros objetos pueden introducir elementos, tanto visuales como no visuales, completamente nuevos, en cuyo caso un código de componente define todo el comportamiento del mismo.

La complejidad de los componentes varía notablemente. Algunos componentes son sencillos; otros encapsulan tareas complicadas. No hay limitación a lo que un componente puede hacer o de qué puede estar compuesto. Puede haber un componente sencillo como una `TLabel`, que

simplemente muestra un texto en un formulario o uno mucho más complicado que encapsule toda la funcionalidad de una hoja de cálculo.

4.1.2 Tipos de componentes

Existen cuatro tipos básicos de componentes que se utilizan y/o crean en controles (en ocasiones se utiliza el término *"control"* como sinónimo de *"componente visual"*) estándar Delphi: controles estándar, controles personalizados, controles gráficos, y componentes no visuales.

4.1.2.1 Componentes estándar

Delphi suministra componentes estándar que encapsulan el comportamiento de los controles de Windows 95/98, tales como `TRichEdit` (editor de textos con formato enriquecido), `TTrackBar` (barra de desplazamiento para seleccionar valores) y `TListView` (lista con elementos en forma de iconos grandes, pequeños, detalle, etc.), por nombrar algunos. Estos componentes existen en una página de la paleta de componentes. Estos componentes son en realidad envoltorios de Object Pascal para los controles comunes de Windows 95/98. Si se dispone del código fuente de VCL, se puede ver el método de Borland para envolver estos controles, en el fichero `ComCtrls.pas`.

4.1.2.2 Componentes personalizados

Componentes personalizados es un término general que se refiere a componentes que no forman parte de la biblioteca de componentes estándar de Delphi. En otras palabras, son componentes que los programadores crean y añaden al conjunto de componentes existente.

Para una información más detallada y en profundidad sobre la manipulación de componentes, se puede hacer referencia a **[**TEI00**]** o bien **[**MAR97**]**

4.1.2.3 Componentes gráficos

Los componentes gráficos permiten tener o crear controles visuales que no reciben el enfoque de entrada del usuario (foco). Estos componentes son útiles cuando se quiere mostrar algo al usuario pero no se desea que el componente utilice recursos Windows, como hacen los componentes estándar y personalizados. Los componentes gráficos no utilizan recursos Windows porque no necesitan manejador de ventana, que es, a su vez, la razón por la que no pueden recibir el enfoque. Estos componentes no sirven tampoco como componentes contenedores; esto es, no pueden tener otros componentes situados sobre ellos. Algunos ejemplos de componentes gráficos son `TImage` (una imagen), `TBevel` (para realzar componentes), `TLabel` (etiqueta de texto en un formulario) y `TShape` (formas de figuras simples)

4.1.2.4 Manejadores

Los manejadores son números de 32 bits generados por Win32 que se refieren a instancias de objeto. El término objetos aquí se refiere a objetos Win32, no a objetos Delphi. Existen diferentes tipos de objetos en Win32: Objetos *Kernel*, objetos de usuario y objetos GDI. La denominación *Kernel* se aplica a elementos tales como eventos, objetos de posicionamiento de ficheros y procesos. Objetos de usuario se refiere a objetos de ventana como controles de edición, cuadros de lista y botones. Objetos GDI son mapas de bits, pinceles, fuentes, etc.

En el entorno Win32, cada ventana tiene un solo manejador. Muchas funciones Windows API necesitan un manejador para saber sobre qué ventana deben realizar la operación. Delphi encapsula gran parte de Win32 API y realiza la gestión de los manejadores. Si se quiere utilizar una función API de Win32 que necesite un manejador de ventana, deben usarse descendientes de `TWinControl` y `TCustomControl` que tienen ambos una propiedad *Handle*.

4.1.2.5 Componentes no visuales

Como su nombre indica, los componentes no visuales no tienen una característica visual. Estos componentes proporcionan la capacidad para encapsular la funcionalidad de una entidad dentro de un objeto, y permiten modificar ciertas características de ese componente a través del Inspector de objetos durante el diseño, mediante la modificación de sus propiedades y suministrando manejadores de eventos para sus eventos.

Ejemplos de estos componentes son `TOpenDialog` (diálogo de apertura de ficheros) `TTable` (tabla de una base de datos) y `TTimer` (para trabajar con eventos periódicos en el tiempo).

4.1.3 Anatomía de un componente. Propiedades, métodos y eventos

Como ya se ha mencionado un componente es un objeto, y como tal, consta de código y datos. Pero al referirnos a éstos no utilizaremos estos términos, sino que hablaremos de propiedades y métodos, así como de eventos.

4.1.3.1 Propiedades

Las propiedades proporcionan al usuario del componente un fácil acceso al mismo. Al mismo tiempo, permite al programador del componente "esconder" la estructura de datos subyacente. Entre las ventajas de utilizar propiedades para acceder al componente se pueden citar:

- Las propiedades están disponibles en tiempo de diseño (programación). De este modo el usuario del componente puede inicializar los valores de las propiedades sin necesidad de escribir una sola línea de código.
- Las propiedades permiten la validación de los datos al mismo tiempo de ser introducidas. Así se pueden prevenir errores causados por valores inválidos.

- Nos aseguran que desde el primer momento nuestras propiedades tendrán un valor válido, evitando el error común de hacer referencia a una variable que no ha sido convenientemente inicializada.

4.1.3.2 Eventos

Los eventos son las conexiones existentes entre un determinado suceso y el código escrito por el programador de componentes. Así por ejemplo, ante el suceso (evento) click del ratón se podría mostrar un mensaje en pantalla. Al código que se ejecuta cuando se produce un determinado evento se le denomina manejador de eventos (*event handler*) y normalmente es el propio usuario del componente quién lo escribe. Los eventos más comunes ya forman parte de los propios componentes de Delphi (acciones del ratón, pulsaciones de teclado...), pero es también posible definir nuevos eventos (como al cargar la configuración de un componente de un fichero, al guardarla...).

4.1.3.3 Métodos

Los métodos son los procedimientos y/o funciones que forman parte del componente. El usuario del componente los utiliza para realizar una determinada acción o para obtener un valor determinado al que no se puede acceder por medio de una propiedad. Ya que requieren ejecución de código, los métodos sólo están disponibles en tiempo de ejecución.

4.2 La familia de componentes DOA: Direct Oracle Access (Acceso Directo a Oracle)

DOA son una serie de componentes y objetos que facilitan el acceso a bases de datos Oracle, directamente, evitando pasar a través del Borland Database Engine (BDE). Esto acarrea numerosas ventajas, como accesos más rápidos y la posibilidad de usar muchas características específicas del sistema gestor de bases de datos Oracle. DOA se puede obtener en [W3ARA].

4.2.1 Características

Algunas de las características más importantes son:

- Ya no es necesario distribuir, instalar y configurar el BDE. Se puede usar Delphi o C++Builder para desarrollar aplicaciones cliente-servidor.
- Mayor velocidad de acceso a la base de datos (hasta 5 veces más rápido), por no tener que pasar a través del BDE.
- Configuración automática de maestro-detalle (relación entre dos tablas donde uno o varios registros de la tabla detalle, hacen referencia a un registro de la tabla maestra).
- El cliente cumple automáticamente las restricciones del servidor.
- Posibilidad de uso de objetos del servidor (como procedimientos y funciones por ejemplo).
- Uso de bloques PL/SQL para una lógica de servidor en las aplicaciones.
- Incrementar el nivel de rendimiento con arrays DML (conjunto de sentencias DML).
- Posibilidad de ejecutar scripts SQL, similar a SQL*Plus a través del componente `TOracleScript`.
- Monitorizar el acceso a la información de la base de datos con la utilidad de monitorización de Oracle
- Acceso a los paquetes almacenados mediante el componente `TOraclePackage`.
- Encapsulación de los paquetes estándar de Oracle (`dbms_alert`, `dbms_job`...).

- Multitud de características especificas de Oracle son soportadas (`Savepoints`, `Set-Transaction`...).
- Compatibilidad con SQL*Net1 hasta Net8, y con Personal Oracle Lite hasta Oracle8i Enterprise Server.
- Posibilidad de usar características de Oracle8 como LOB's y Objetos.

4.2.2 Los componentes de DOA

Direct Oracle Access contiene el siguiente conjunto de componentes:

- `TOracleSession`. Se utiliza para conectar a una base de datos Oracle y controlar las transacciones. Se pueden usar varias sesiones simultáneamente, accediendo a distintas bases de datos.
- `TOracleLogon`. Este componente permite al usuario especificar el nombre de usuario y la contraseña para una sesión, mediante el diálogo (formulario) estándar de entrada.
- `TOracleQuery`. Se puede utilizar este componente para ejecutar una sentencia SQL o un bloque PL/SQL en una sesión.
- `TOraclePackage`. Proporciona un interfaz de acceso a funciones, procedimientos, variables y constantes almacenados en un paquete.
- `TOracleEvent`. Este componente permite a una aplicación reaccionar a las señales de `dbms_alert` y a los mensajes de `dbms_pipe` en una ejecución de un *thread* en segundo plano.
- `TOracleDataSet`. Es la fuente de datos de todos los componentes. Se utiliza internamente un `TOracleQuery` para leer y actualizar la base de datos.
- `TOracleDirectPathLoader`. Permite cargar datos a la velocidad máxima posible usando el entorno Oracle Direct Load.
- `TOracleScript`. Ofrece la posibilidad de ejecutar scripts SQL.
- `TOracleNavigator`. Es un componente similar al estándar `TDBNavigator`.
- `TOracleProvider`. Este componente es similar al estándar `TProvider` y puede ser utilizado para crear aplicaciones multi-tier que utilicen los componentes de DOA.

4.2.3 Instalación paso a paso

Para instalar los componentes de Direct Oracle Access en el entorno de Delphi, primero debemos ejecutar el programa incluido "setup.exe". Esto instalará los *paquetes de diseño*, unidades y el fichero de la ayuda on-line. Dependiendo del compilador que se utilice, será necesario llevar a cabo algunos pasos más.

4.2.3.1 Instalando el componente `TOraclewwDataSet`

Opcionalmente podemos instalar el componente `TOraclewwDataSet` (puesto que no se instala por defecto), para permitir a los controles de InfoPower, usar Direct Oracle Access. Para ello, desde el menú "Component" de Delphi, seleccionamos "Install component". Ahora, buscamos el fichero `OraclewwData.pas` en el directorio lib de Delphi. Puede que sea necesario instalar este componente en el paquete de Direct Oracle Access Desde (`doa.dpk`), el cual está localizado en el mismo directorio.

4.2.3.2 Instalando el componente `TOracleProvider`

En nuestro caso, por usar la versión cliente-servidor, podemos adicionalmente instalar el componente `TOracleProvider`, si deseamos crear aplicaciones multi-tier con Direct Oracle Access (ver Capítulo 3 y **[**MAR97**]**, donde se dedica todo un capítulo a este tema). Para ello, desde el menú "Component" de Delphi, seleccionamos "Install component". Ahora, seleccionamos el fichero OracleProviderReg.pas en el directorio lib de Delphi.

4.2.4 Programas de ejemplo

En el directorio Demos, podremos encontrar los siguientes directorios. Cada directorio contiene un proyecto de demostración de la utilización de los componentes.

- LongRaw. Lee y escribe columnas de datos long raw.

- ThreadDemo. Una aplicación multi-thread.
- ObjectGrid. Ejemplo de manipulación de objetos persistentes (solo Oracle8).
- DeptEmp. Un ejemplo de una actualización en cascada, usando los componentes data-aware.
- PictureDemo. Un ejemplo de utilización de campos Blob.
- 3Tier. Solo para la versión cliente-servidor. Un ejemplo de una aplicación de 3-tier.
- PkgApply. Un ejemplo de cómo utilizar un paquete almacenado, para seleccionar registros y aplicar cambios a un conjunto de datos.

4.3 La familia de componentes SynEdit: (Syntax highlighting Edit control)

SynEdit son una serie de componentes y objetos, que permiten el realzar la sintaxis de un determinado lenguaje. SynEdit incorpora multitud de unidades. Cada unidad se encarga de un lenguaje en concreto. SynEdit también nos ofrece la posibilidad de crear nuestro propio resaltador de sintaxis y poder emplearlo asociado con en el control `TSynEdit`. SynEdit, se puede descargar desde **[**W3SYN**]**.

4.3.1 Características

Algunas de las características más importantes son:

- No es necesario que el programador controle manualmente la sintaxis del texto, para cambiar los colores.
- Alta velocidad al parsear el texto.
- Posibilidad de múltiples resaltadores en un solo control.
- Gran número de resaltadores ya integrados en el conjunto de componentes (más de 40).
- Incluye un componente para ver código fuente almacenado en bases de datos.

- Incluye un componente capaz de exportar a formatos HTML y RTF.
- Incluye un componente para auto-completado de código fuente.
- Ofrece la posibilidad de grabar macros de usuario.
- Incluye componente para la impresión y previsualización del código fuente, completamente configurable.
- Incluye componente para la corrección del código fuente.
- Ofrece multitud de características de un editor de programación, como gutter (margen izquierdo del texto, donde se pueden numerar las líneas, mostrar indicadores...), margen (derecho), personalización de colores, etc.

4.3.2 Los componentes

SynEdit, contiene el siguiente conjunto de componentes:

- `TSynEdit`. Se utiliza como control base que interactúa con el usuario. También se encuentra disponible `TSynMemo`, que es similar a `TSynEdit`, pero con un conjunto de características un poco más reducido.
- `TDBSynEdit`. Similar a `TSynEdit`, posee la capacidad de vincularse a un conjunto de datos. Es decir, se utiliza para ver código fuente almacenado en una base de datos.
- `TSynExporterHTML`. Se utiliza para exportar el código fuente a formato HTML, con el realzado de sintaxis incorporado.
- `TSynExporterRTF`. Similar a `TSynExporterHTML`, se utiliza para exportar a formato RTF.
- `TSynCompletionProposal` y `TSynAutoComplete`. Se utilizan para implementar un motor de auto completado, similar al que trae el editor de Delphi (*CodeInsight*).
- `TSynMacroRecorder`. Ofrece la posibilidad de definir macros de usuario como accesos a teclado y combinaciones de tecla para efectuar tareas.
- `TSynEditPrint`. Este componente encapsula la impresora, pudiendo así imprimir código, además de darle un formato uniforme.
- `TSynPrintPreview`. Nos da la posibilidad de previsualizar el documento antes de imprimirlo y hacer zoom en el documento.

- `TSynAutoCorrect`. Se utiliza para implementar un motor de auto corrección.
- En la paleta SynEdit HightLighters, encontraremos multitud de resaltadores, que abarcan un gran abanico de lenguajes, como por ejemplo: C++, Fortran, Java, Pascal, Basic, HTML, JavaScript, PHP, VBScript, Perl, SQL (incluido PL/SQL), ASM, y un largo etc.

4.3.3 Instalación paso a paso

Para llevar a cabo la instalación de SynEdit, tenemos que seguir los siguientes pasos.

1. Descomprimir el fichero zip, en un directorio (por ejemplo *c:\UsrLib*)
2. Abrimos el fichero SynEdit_D6.dpk que se encuentra ubicado dentro del directorio *Packages* donde hayamos descomprimido anteriormente el fichero *zip*.
3. Ahora compilamos el paquete. Para ello como muestra la Figura 4.1, debemos pulsar compile.
4. Una vez que hayamos compilado, instalamos el paquete. Para ello, debemos pulsar *install*.
5. Con este proceso, hemos instalado el paquete. Ahora tenemos que informarle a Delphi del nuevo *path* de los componentes SynEdit. Para ello debemos añadir el directorio donde hemos instalado los componentes, de la siguiente forma.
    a. En el menú *Project*, hacemos click sobre *Options*.
    b. Ahora vamos a la pestaña *Directories / Conditionals*
    c. En *Search path*, añadimos el directorio donde estén los fuentes de los componentes SynEdit (por ejemplo *c:\UsrLib\source*)

Esto también se podría haber hecho en *Tools* | *Environment Options...* y en la pestaña *Library*, haber añadido el *path* a *Library Path*, tal y como se ilustra en la Figura 4.2.

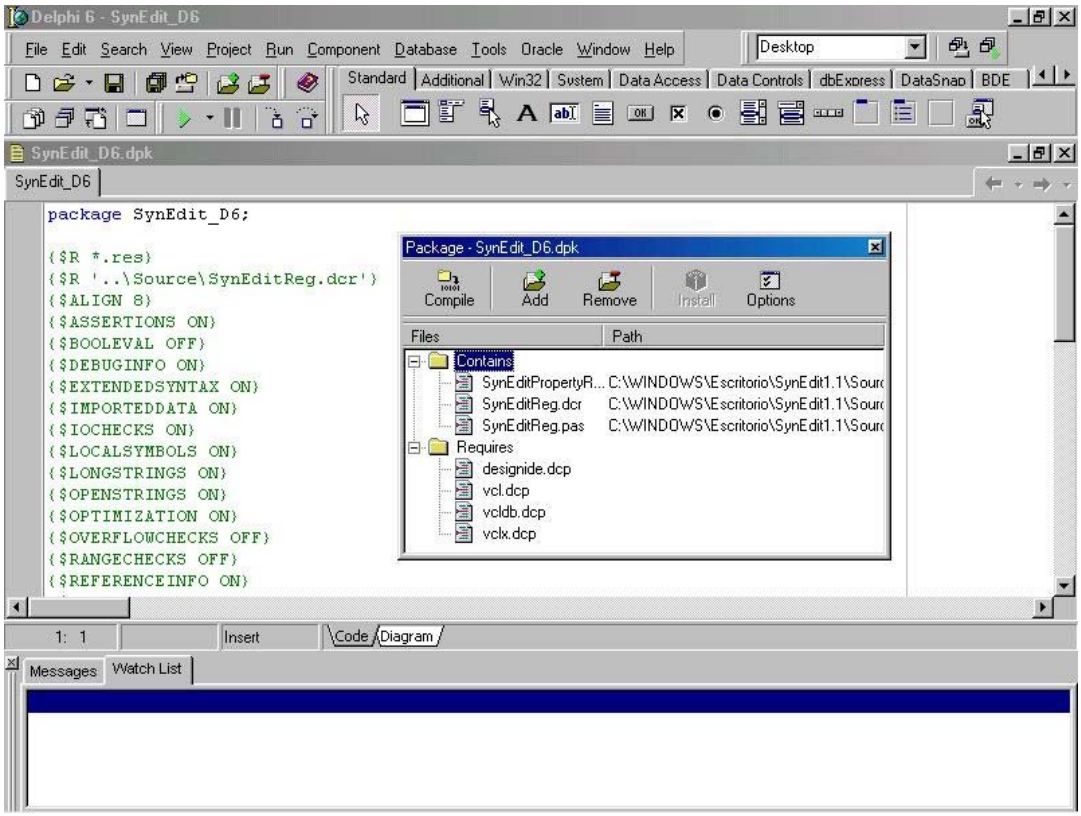

Figura 4.1 Instalación del paquete `SynEdit_D6.dpk`

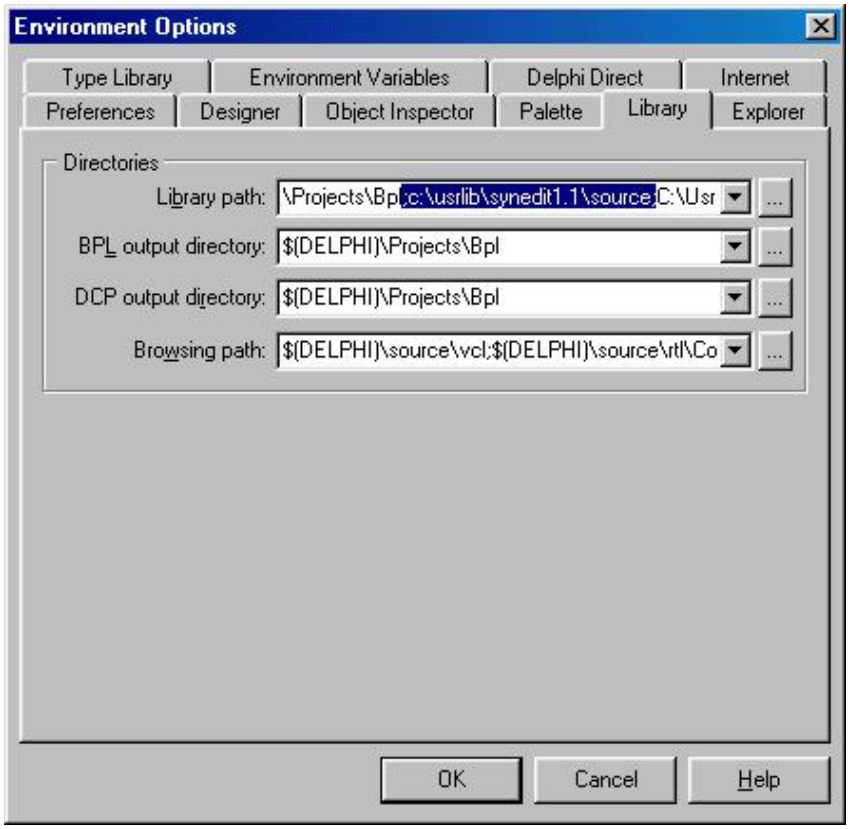

Figura 4.2 Añadiendo el *path* de búsqueda a los componentes.

4.3.4 Programas de ejemplo

Al igual que DOA y la mayoría de componentes, SynEdit, también dispone de algunos programas de ejemplo de utilización de los componentes.

- D4Demo. Muestra muchas de las capacidades de SynEdit y todos los resaltadores de sintaxis. Se necesita al menos Delphi 4.
- DBSynEdit. Muestra como utilizar el componente TDBSynEdit.
- ExportDemo. Muestra como exportar ficheros a formato HTML o al portapapeles, usando los componentes `TSynExporterHTML` y `TSynExporterRTF`
- HighLighterDemo. Muestra como crear un nuevo resaltador, usando un fichero de gramáticas y la utilidad SynGen.
- PrintDemoNew. Muestra como imprimir y hacer previsualizaciones de impresión.

Capítulo

5

5 En las profundidades de Medusa

En este capítulo trataremos el programa en todo detalle. Un poco de teoría MDI para comenzar, seguido de la estructura general del programa (organización, módulos...) y el desarrollo paso a paso. Para poder seguir bien este capítulo, es necesario conocer Delphi.

5.1 Un poco de teoría

Antes de comenzar con la implementación del programa, es necesario que sepamos algunos conceptos previos, sobre todo de MDI (*Multiple Document Interface)*, puesto que la forma de pensar y de programar difiere un poco de las aplicaciones normales SDI (*Single Document Interface*).

5.1.1 MDI (Multiple Document Interface)

La Interfaz de Documentos Múltiples, también conocida como MDI, se introdujo en el sistema operativo Windows 2.0, en el programa de hoja de cálculo Microsoft Excel. MDI les dio a los usuarios de Excel la posibilidad de trabajar en más de una hoja de cálculo a la vez. Otras opciones de utilización de MDI incluían los programas Administrador de programas (*Program Manager*) de Windows 3.1 y Administrador de archivos (*File Manager*).

Puede parecer difícil manipular eventos simultáneamente entre múltiples formularios. En la programación tradicional de Windows, hay que tener conocimiento de la clase de Windows llamada MDICLIENT, estructuras de datos MDI y las funciones adicionales y mensajes específicos de MDI.

5.1.1.1 Cómo crear una aplicación MDI

Para crear aplicaciones MDI es necesario familiarizarse con los dos estilos de formulario `fsMDIForm` y `fsMDIChild` (padre e hijo) y con un poco de la metodología de la programación MDI. Las secciones siguientes presentan algunos conceptos básicos relacionados con MDI y muestran cómo funciona con formularios hijo MDI especiales.

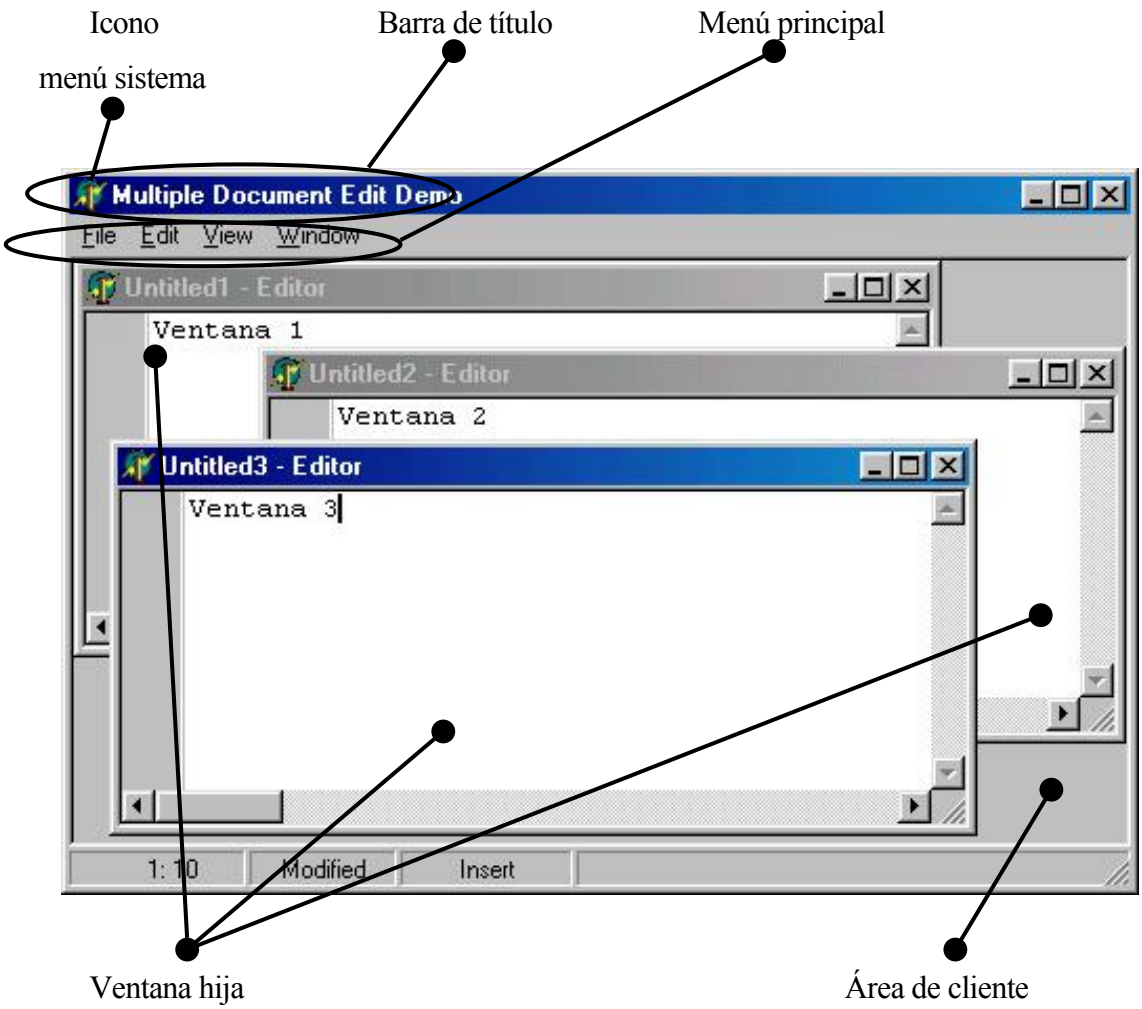

Figura 5.1: Estructura de una aplicación MDI.

Las ventanas incluidas en una aplicación MDI son, como se ve en la Figura 5.1, las siguientes:

- Ventana marco (*Frame window*). Es la ventana principal de la aplicación. Tiene un título, un menú principal y un icono de menú de sistema (opcionalmente, puede tener una barra de herramientas). Hay botones de minimizar, maximizar y cierre en la esquina superior derecha. El espacio en blanco que hay dentro de la ventana se conoce como su área cliente y es, en realidad, la ventana del cliente.
- Ventana cliente (*Client window*). Es el gestor de las aplicaciones MDI. La ventana cliente manipula todos los comandos específicos de MDI y las ventanas hijo que residen en su superficie (incluyendo su dibujo). Cuando creamos una ventana marco, la Biblioteca de Componentes Visuales (*Visual Component Library, VCL*) crea automáticamente el cliente.

- Ventanas hijo (*Child windows*). Las ventanas hijo de MDI son sus documentos activos (archivos de texto, hojas de cálculo, imágenes y otros tipos de documentos). Como las ventanas marco, tienen un título, un menú de sistema, botones para maximizar, minimizar y cerrar y, posiblemente, un menú y una barra de herramientas. El menú de estas ventanas está combinado con el de la ventana marco. Las ventanas hijo nunca se mueven fuera del área de cliente.

5.1.1.2 La ventana hija

Al programar en Delphi no necesitamos conocer los mensajes especiales de las ventanas MDI. La ventana cliente es la responsable de la manipulación de la funcionalidad MDI, como la representación en cascada y repetición de las ventanas hijas. Por ejemplo, para presentar en cascada las ventanas hijas utilizando el método tradicional, tenemos que utilizar la función API de Windows `SendMessage()` para enviar un mensaje `WM_MDICASCADE` a la ventana cliente, como se muestra a continuación en el Listado 5.1.

```
procedure TFrmPrincipal.CascadeClick(Sender: TObject);
begin
  SendMessage(ClientHandle, WM_MDICASCADE, 0, 0);
end;
```

Listado 5.1: Método de mensajes mediante llamadas a la API

En Delphi, tenemos la posibilidad de, en lugar de hacer la llamada directamente a la API, hacerlo directamente usando el método **`Cascade()`**, como muestra el Listado 5.2

```
procedure TFrmPrincipal.CascadeClick(Sender: TObject);
begin
  Cascade;
end;
```

Listado 5.2: Método de mensajes mediante el método `Cascade()`

También existe la posibilidad de asociar acciones estándar de Delphi. De esta forma se mejora la forma de programación, puesto que se puede reutilizar más el código, hay que escribir menos y, por tanto, habrá menos errores.

Puesto que todos los hijos MDI necesitan tener la misma funcionalidad (o similar), podría tener sentido encapsular esta funcionalidad en una clase base de la que descenderán todos estos formularios hijos. De esta forma, los formularios hijo MDI no tienen que definir los mismos métodos. Heredarán el menú principal, así como los componentes de la barra de herramientas que puedan tener. En nuestro caso como sólo hay un tipo de formulario hijo MDI, se le han asociado directamente los métodos a éste.

Hay que mencionar que en el evento `FormClose()`, se debe establecer el parámetro **`Action`** como **`caFree`** para asegurar que se destruye la instancia cuando se cierra. Esto se hace porque no se cierran automáticamente los formularios hijo MDI cuando se llama al método `Close()`. En el manejador de eventos `OnClose` se debe especificar lo que deseamos que le ocurra al formulario hijo cuando se llame a su método `Close()`. El manejador de eventos `OnClose` del formulario hijo pasa en una variable **`Action`**, de tipo `TCloseAction` (que es un tipo enumerado), a la que debemos asignar uno de estos cuatro posibles valores:

- `caNone`. No hace nada.
- `caHide`. Oculta el formulario pero no lo destruye.
- `caFree`. Libera el formulario.
- `caMinimize`. Minimiza el formulario (es el valor por defecto).

Cuando se activa un formulario, se llama a su manejador de eventos `OnActivate`. Siempre que se llama, activa un formulario hijo hay que realizar alguna lógica específica. Así pues, en el manejador de eventos `FormActivate()`, se observa que se aplican las configuraciones del editor, sintaxis, etc. y se actualiza el estado de los botones de la barra de herramientas del formulario principal. El manejador de eventos `OnDeactivate` simplemente se utiliza en el caso contrario, cuando se desactiva el formulario hijo. Para terminar, el evento `OnClose` libera la memoria consumida por los elementos que forman parte del formulario, por ejemplo, la pestaña asociada a ese formulario, y el *buffer*; y finalmente actualiza el estado de los botones de la barra de herramientas del formulario principal.

Hay una forma muy interesante para tratar el tema de la actualización de menús en Delphi, que se trata en profundidad en **[**TEI00**]**. La técnica consiste en que se puedan *asignar* menús de ventanas hijo al formulario principal, lo cual permite tener distintos menús, uno para cada tipo de ventana hija y mostrar el menú y la barra de herramientas en el formulario principal en función de la ventana hija activa.

5.1.1.3 La ventana padre

El formulario principal es aquél con el que el usuario trabaja inicialmente para crear el formulario hijo. Este formulario se llama `FrmPrincipal`. `FrmPrincipal` sirve como padre para el formulario hijo MDI del cliente avanzado de Oracle llamado `FrmEditor`. El aspecto de `FrmPrincipal` en ejecución es el mostrado en la Figura 5.2.

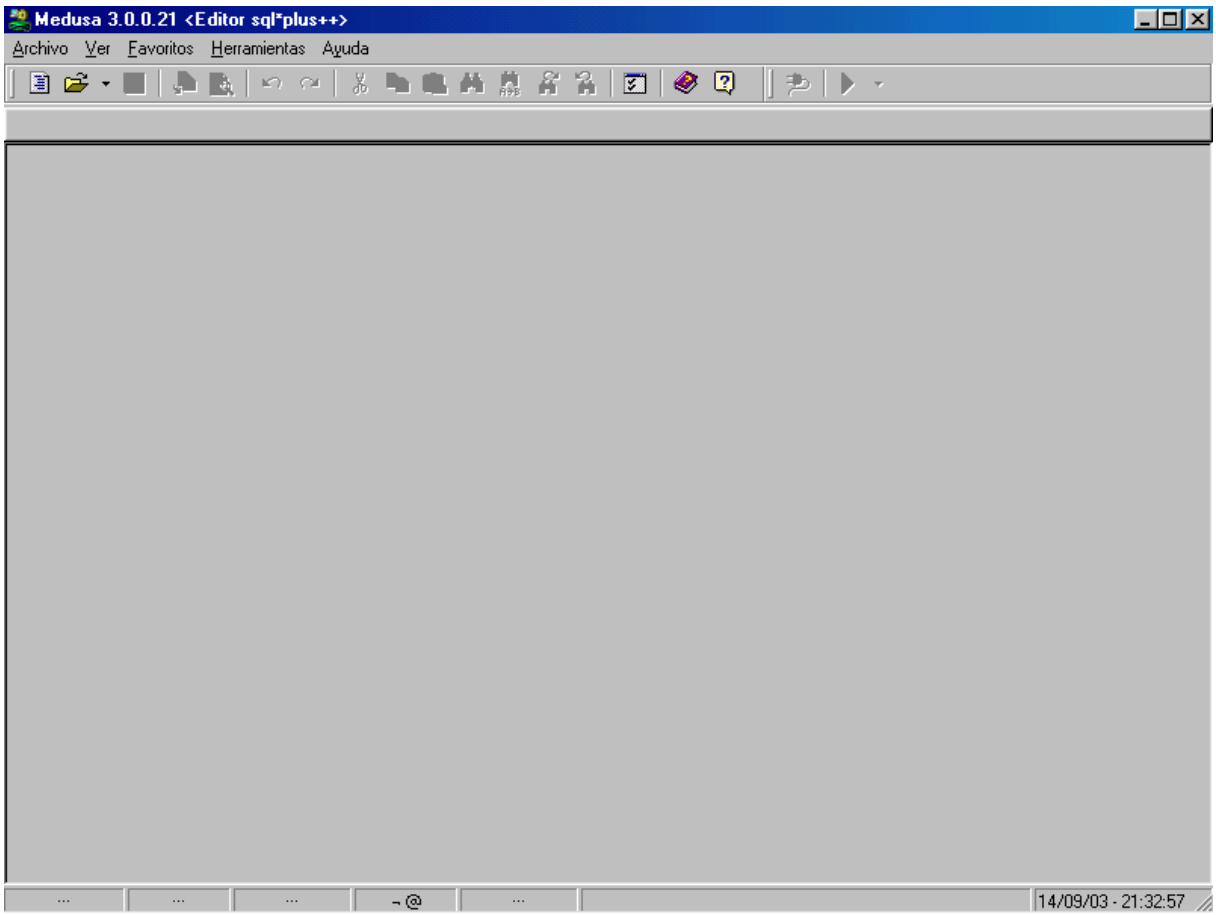

Figura 5.2: Aspecto del formulario principal de *Medusa*

`FrmPrincipal` tiene la propiedad `FormStyle` asignada a `fsMDIForm`, en cambio, la ventana del cliente avanzado (`FrmEditor`), tiene la propiedad `FormStyle` asignada a `fsMDIChild`. Esta propiedad permite distinguir entre las ventanas, para saber cual es la ventana padre y cual es la ventana hijo.

5.1.1.4 Fusión de menús en aplicaciones MDI

Esta sección muestra cómo una aplicación MDI permite que sus formularios hijos compartan la misma barra de menú utilizando un método llamado *"fusión de menús"*.

Si observamos el `TMainMenu` de la clase `TFrmEditor`, veremos que haciendo doble click en el icono `TMainMenu`, se obtiene el editor de menú.

El menú de `TFrmEditor` contiene varios elementos de menú en la barra de menú. Estos elementos son *Archivo*, *Edición*, *Diccionario de datos*, *Oracle y Ventanas*. Cada uno de estos elementos de menú tiene una propiedad `GroupIndex` que se muestra en el inspector de objetos si se hace click en un elemento del menú en el editor de menú. El elemento de menú *Archivo* tiene un valor `GroupIndex` de 0, el elemento de menú *Edición* tiene un valor de `GroupIndex` de 1, el elemento del menú *Diccionario de datos* tiene un valor `GroupIndex` de 3, el elemento de menú *Oracle*, tiene un valor `GroupIndex` de 6 y por último, el elemento de menú *Ventanas* tiene un valor `GroupIndex` de 7. El aspecto del menú de este formulario es el que muestra la Figura 5.3

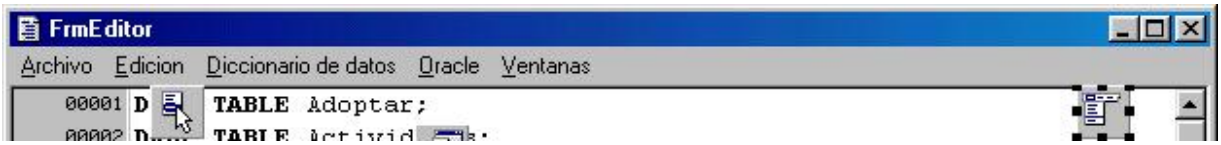

Figura 5.3: Aspecto del menú del formulario `FrmEditor` en tiempo de diseño

En el menú de la clase del formulario principal `TFrmPrincipal`, se encuentran cinco elementos de menú en la barra de menú. Estos elementos son *Archivo*, *Ver*, *Herramientas*, *Favoritos* y *Ayuda*. Cada uno de estos elementos de menú, al igual de como ocurría con el formulario hijo, tiene una propiedad `GroupIndex` que se muestra en el inspector de objetos si se hace click

en un elemento del menú en el editor de menú. El elemento de menú *Archivo* tiene un valor `GroupIndex` de 0, el elemento de menú *Ver* tiene un valor de `GroupIndex` de 2, el elemento del menú *Favoritos* tiene un valor `GroupIndex` de 5, el elemento del menú *Herramientas* tiene un valor `GroupIndex` de 5, y por último, el elemento de menú *Ayuda* tiene un valor `GroupIndex` de 8. El aspecto del menú de este formulario es el que muestra la Figura 5.4

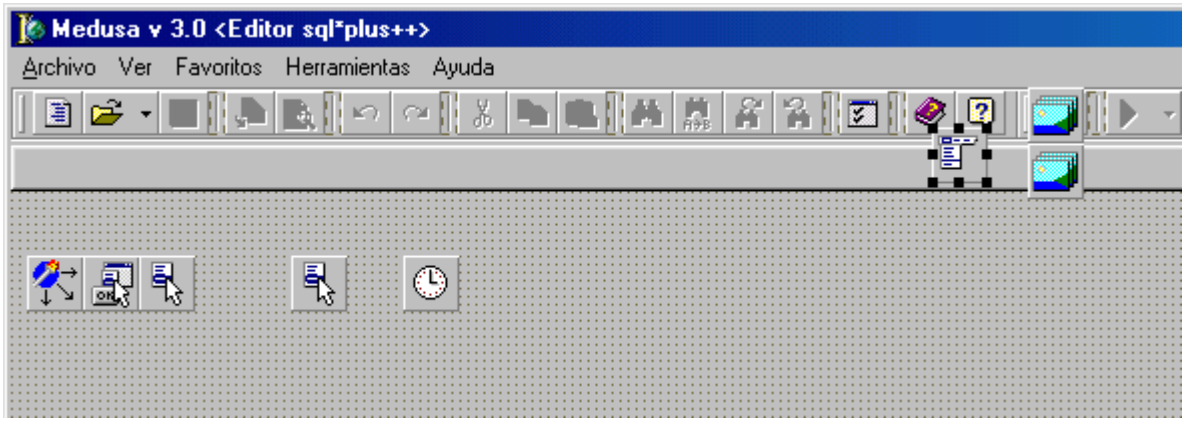

Figura 5.4: Aspecto del menú del formulario `FrmPrincipal` en tiempo de diseño

Nótese que el menú *Archivo* de `TFrmPrincipal` y el menú *Archivo* de `TFrmEditor`, tienen elementos de submenú diferentes. El menú *Archivo* de `TFrmEditor` tiene más elementos de submenú que el menú *Archivo* de `TFrmPrincipal`. Ambos elementos de menú tienen el mismo valor de la propiedad `GroupIndex`, lo cual permite que al crear una ventana hija, el elemento de menú de la ventana del cliente avanzado (`FrmEditor`) se fusione con el del formulario principal.

La propiedad `GroupIndex` es importante ya que permite que los menús de los formularios se fusionen. Esto significa que cuando el formulario principal genera un formulario hijo, el menú del formulario hijo se fusiona con el menú del formulario principal. La propiedad `GroupIndex` determina cómo se ordenan los menús y qué menús del formulario principal se reemplazan por menús del formulario hijo. La fusión de menús sólo es aplicable a elementos de menú de la barra de menú de un componente `TMainMenu` y no a submenús. Cuando una propiedad `GroupIndex` de un elemento de menú de un formulario hijo tenga el mismo valor que la propiedad `GroupIndex` del elemento de menú del formulario principal, el elemento de menú del formulario hijo sustituye al elemento de menú del formulario principal. Los menús restantes se distribuyen en la barra la barra de menú en el orden especificado por las propiedades

`GroupIndex` de todos los elementos de menú combinados. Cuando `FrmEditor` es el formulario activo del proyecto, los elementos del menú que aparecen en la barra de menú del formulario principal son *Archivo*, *Edición*, *Ver*, *Diccionario de datos*, *Favoritos*, *Herramientas*, *Oracle*, *Ventanas* y *Ayuda*, en ese orden. El menú *Archivo* es el menú *Archivo* de `TFrmEditor`, ya que ambos menús *Archivo* tienen valores de la propiedad `GroupIndex` de 0. Por tanto, el menú *Archivo* de `TFrmEditor` sustituye al menú *Archivo* de `TFrmPrincipal`. El orden de estos menús refleja directamente el orden de las propiedades `GroupIndex` de cada elemento de menú en la barra del menú, tal y como se muestra en la Figura 5.5.

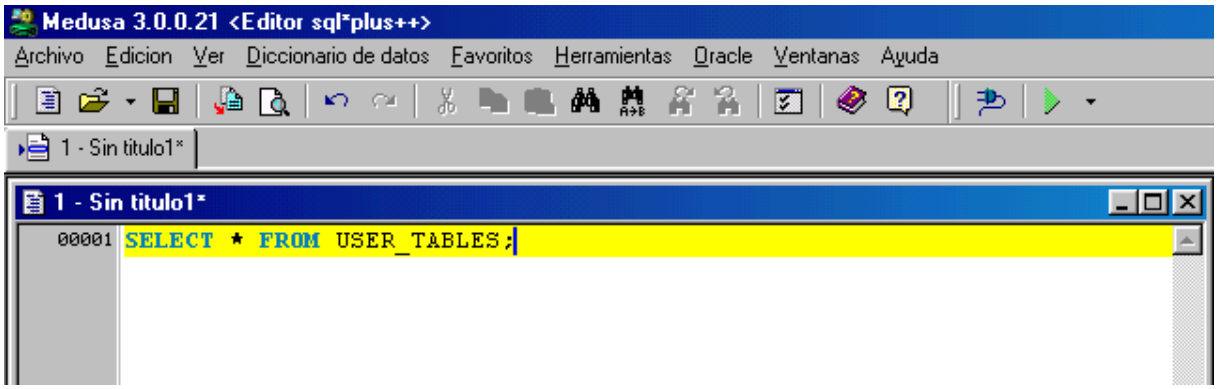

Figura 5.5: Aspecto del menú una vez fusionado con el menú de la ventana hija.

La fusión de menús de las aplicaciones MDI es automática. Mientras los valores de la propiedad `GroupIndex` de los elementos de menú estén establecidos en el orden que especifiquemos, los menús se fusionan correctamente cuando se invocan formularios hijos MDI.

5.2 Estructura de Medusa

Medusa contiene multitud de formularios. Unos son utilizados únicamente para aspectos relacionados con la edición, mientras que otros se utilizan para aspectos relacionados con la reconstrucción y obtención de datos de la base de datos. No todos los formularios, se utilizan de forma exclusiva en estas categorías. Básicamente, podemos decir que *Medusa* se estructura en seis partes, que son:

- Ventana principal.
- Ventana hija.
- Ventanas de configuración.
- Ventanas relacionadas con la edición.
- Ventanas relacionadas con la base de datos.
- Contenedor global de objetos.

5.2.1 Componentes de programación empleados

En esta sección, se describen brevemente los componentes empleados en los formularios más importantes de *Medusa*. Se han empleado multitud de componentes, mayoritariamente de las paletas *Standard*, *Additional* y *Win32* de Delphi y los componentes de DOA ubicados en la paleta *Data Access*, junto con los componentes de visualización, ubicados en la paleta *Data Controls* y por último, también se han utilizado los componentes de SynEdit, ubicados en las paletas *SynEdit* y *SynEdit Highlighters*.

5.2.1.1 La clase `TFrmPrincipal`

Entre los componentes de `TFrmPrincipal`, podemos destacar los siguientes componentes:

- Un componente `TMainMenu`. Se utiliza como menú principal de la aplicación.
- Un componente `TCoolBar`. Se utiliza como contenedor de las dos barras de herramientas accesibles al usuario.
- Un componente `TStatusBar`. Se utiliza como barra de estado, para mostrar diversa información como la línea y columna del cursor, el usuario, etc.
- Un componente `TPopUpMenu`. Se utiliza como menú de ficheros recientemente abiertos (LRU, *Last Recently Used*).
- Dos componentes `TImageList`. Se emplean para las barras de herramientas.

- Un componente `TApplicationEvents`. Se emplea para mostrar los *hints* (mini ayuda) de los controles, cuando el usuario pasa el cursor del ratón por encima.

5.2.1.2 La clase `TFrmEditor`

Entre los componentes de `TFrmEditor`, podemos destacar los siguientes componentes:

- Un componente `TSynEdit`. Se emplea como editor de textos, para escribir consultas, *scripts* o cualquier texto en general.
- Un componente `TPageControl`. Se emplea como contenedor para las pestañas de *resultados*, *mensajes* y *errores*.
- Un componente `TMainMenu`. Se emplea como menú principal, que se fusionará con el menú del formulario principal.
- Varios componentes `TActionList`. Se utilizan como contenedores de acciones, hay uno para cada menú.
- Dos componentes `TPopUpMenu`. El primero, es el menú del botón derecho del ratón asociado al editor; el segundo, es el menú de los modos de ejecución del documento (consulta o *script*).
- Un componente `TOracleDataSet`. Se emplea para lanzar las consultas a Oracle, tiene asociado un componente de la clase `TDBGrid` para la visualización de los datos.
- Un componente `TOracleScript`. Se emplea para la ejecución de los *scripts* en Oracle (unos capítulos más adelante, se verá la diferencia entre ambos modos de ejecución).

5.2.1.3 La clase `TDMGlobal`

`TDMGlobal`, es un contenedor (*Data Module*, Módulo de Datos) de componentes no visuales, asociados a varios componentes o controles de la aplicación en diversos puntos. Por ello se

ha factorizado la utilización común de éstos en el módulo de datos. Entre los componentes de `TDMGlobal`, podemos destacar los siguientes componentes:

- Un componente `TSynSQLSyn`. Se emplea como resaltador de sintaxis de SQL y PL/SQL de las ventanas de edición.
- Un componente `TSynExporterHTML`. Se emplea como exportador a formato HTML de la consulta, para añadir también la tabla con los datos obtenidos, el usuario, fecha, etc. Se han empleados Streams (flujos) de datos.
- Un componente `TSynAutocompletionProposal`. Se emplea como motor de autocompletado (lista visible de autocompletado). El otro tipo de autocompletado (combinación de teclas), se hace con un componente `TSynAutoComplete`, de la unidad `SynEditAutoComplete`. Es muy importante tener en cuenta la unidad, puesto que hay dos clases con el mismo nombre y son completamente distintas, y una pequeña confusión puede provocar graves errores en tiempo de ejecución.

5.2.2 Estructuras de datos empleadas

Las estructuras empleadas en *Medusa* son escasas y poco complejas, podemos destacar la estructura de datos empleada para la implementación del *buffer* circular de texto, expresada en el Listado 5.3.

```
PDatosBuffer = ^TDatosBuffer;
TDatosBuffer =  record
                  Hora  : TTime;
                  Texto : Pointer;
                  Error : Boolean;
                end;
```

Listado 5.3: Estructura de datos para la implementación del *buffer* circular

También hay que comentar que se han añadido varios atributos a los formularios, especialmente, al formulario `FrmEditor`. Los atributos añadidos son, por ejemplo, para implementar

el *buffer* circular, para lo cual necesitamos una lista y un entero que represente el índice actual; la pestaña de la ventana, algunas variables para la búsqueda, etc. Al igual que se han añadido atributos, también se han añadido numerosos métodos, por ejemplo, para la búsqueda, para abrir un fichero, para consultar el diccionario de datos, etc.

5.2.3 Organización

Todo proyecto en Delphi, está formado por un módulo principal (*Medusa*.dpr, en nuestro caso), y opcionalmente algunos formularios o módulos de datos. Cada formulario, a su vez, está ligado a una unidad, por ejemplo el formulario `FrmPrincipal`, que viene definido en el fichero `UPrincipal.dfm` (delphi form) y está asociado a la unidad `UPrincipal.pas`, que contiene el código de los manejadores de eventos así como otras rutinas, tipos de datos, etc. También pueden existir unidades solas sin formularios, que contienen rutinas, tipos, clases, etc.

5.2.3.1 Módulos de trabajo

En esta sección explicaremos brevemente los formularios más importantes de *Medusa* y para qué son empleados:

- `DMGlobal` (`UDMGlobal.dfm`, `UDMGlobal.pas`). Módulo de datos global a la aplicación, contiene los componentes comunes que se utilizan en varios puntos, que han sido factorizados para no duplicar código.
- `FrmAbout` (`UAbout.dfm`, `UAbout.pas`). Formulario que muestra los datos del autor, tutor y el programa.
- `FrmBuffer` (`UBuffer.dfm`, `UBuffer.pas`). Formulario que permite el acceso al *buffer* circular de los textos de las órdenes.
- `FrmBuscarTexto` (`UBuscar.dfm`, `UBuscar.pas`). Formulario que permite la búsqueda de texto de forma circular en el documento.
- `FrmCabeceraPlantilla` (`UCabecerasPlantillas.dfm`, `UCabecerasPlantillas.pas`). Formulario que permite añadir y modificar las cabeceras de las plantillas de teclado.

- `FrmComentarios` (`UComentarios.dfm`, `UComentarios.pas`). Formulario que permite la fácil manipulación de comentarios sobre tablas y columnas.
- `FrmConfiguracion` (`UConfiguracion.dfm`, `UConfiguracion.pas`). Formulario que permite personalizar el programa, accediendo a multitud de opciones.
- `FrmConfirmarReemplazo` (`UConfirmarReemplazar.dfm`, `UConfirmarReemplazar.pas`). Formulario que se muestra cuando se está reemplazando un texto, para confirmar el reemplazo.
- `FrmEditor` (`UEditor.dfm`, `UEditor.pas`). El formulario más importante de *Medusa*, es el formulario de edición, que permite al mismo tiempo ejecutar en modo consulta o ejecutar en modo script.
- `FrmEstructuraBD` (`UEstructuraBD.dfm`, `UEstructuraBD.pas`). Formulario que permite tener una visión global de la base de datos, recuperando a partir de un conjunto de tablas, sus atributos y relaciones, en un documento de texto o en pantalla.
- `FrmFormatoFechas` (`UFormatoFechas.dfm`, `UFormatoFechas.pas`). Formulario que permite la fácil construcción de formatos de fechas, para ayudar al usuario con esta tarea.
- `FrmInformación` (`UInformacion.dfm`, `UInformacion.pas`). Formulario que muestra multitud de datos sobre el usuario conectado, como pueden ser: los privilegios, roles, limitaciones, etc. y de la sesión actual
- `FrmPermisos` (`UPermisos.dfm`, `UPermisos.pas`). Formulario que permite la gestión de permisos de forma visual, para facilitar esta labor.
- `FrmPreview` (`UPreview.dfm`, `UPreview.pas`). Formulario que permite la pre-visualización del documento antes de ser impreso.
- `FrmPrincipal` (`UPrincipal.dfm`, `UPrincipal.pas`). Formulario que permite el acceso a todas las opciones iniciales del programa. El punto de partida del usuario.
- `FrmReconstruirCreateTable` (`UReconstruirCreateTable.dfm`, `UReconstruirCreateTable.pas`). Formulario que permite

la reconstrucción de una tabla, obtener todos sus atributos, restricciones, referencias, etc.

- `FrmReemplazarTexto` (`UReemplazar.dfm`, `UReemplazar.pas`). Formulario que permite el reemplazo de texto de forma iterativa y circular.
- `FrmSplash` (`USplash.dfm`, `USplash.pas`). Formulario que muestra la ventana de presentación del programa.
- `FrmVisualIndexes` (`UVisualIndexes.dfm`, `UVisualIndexes.pas`). Formulario que facilita la creación de índices de forma visual.
- `FrmVisualJava` (`UVisualJava.dfm`, `UVisualJava.pas`). Formulario que se utiliza para facilitar la creación de bloques Java que se utilizarán en la Base de Datos.
- `FrmVisualPackages` (`UvisualPackages.dfm`, `UvisualPackages.pas`). Formulario que permite crear la definición y el cuerpo de paquetes de una forma sencilla.
- `FrmVisualProcFuncs` (`UVisualProcFuncs.dfm`, `UVisualProcFuncs.pas`). Formulario que facilita la construcción de procedimientos y funciones, para que se pueda llevar a cabo de forma visual.
- `FrmVisualProfiles` (`UVisualProfiles.dfm`, `UVisualProfiles.pas`). Formulario que facilita la construcción de perfiles de forma visual.
- `FrmVisualSeqs` (`UVisualSeqs.dfm`, `UVisualSeqs.pas`). Formulario que permite generar de forma sencilla el código de una secuencia, de forma gráfica.
- `FrmVisualSynonyms` (`UVisualSynonyms.dfm`, `UVisualSynonyms.pas`). Formulario que permite generar sinónimos de forma visual, para los objetos de la Base de Datos.
- `FrmVisualTablespaces` (`UVisualTablespaces.dfm`, `Uvisual-Tablespaces.pas`). Formulario que permite crear *tablespaces* de forma visual, para almacenar los objetos de la Base de Datos.
- `FrmVisualTriggers` (`UVisualTriggers.dfm`, `UVisualTriggers.pas`). Formulario que facilita la construcción de disparadores, para que se pueda llevar a cabo de forma visual.

- ℘ `FrmVisualUsers` (`UVisualUsers.dfm`, `UVisualUsers.pas`). Formulario que facilita la creación de usuarios, para que se pueda llevar a cabo de forma visual.
- ℘ `FrmVisualViews` (`UVisualViews.dfm`, `UVisualViews.pas`). Formulario que facilita la construcción de vistas, para que se pueda llevar a cabo de forma visual.
- ℘ `UGlobal.pas`. Unidad global, que contiene rutinas útiles empleadas en varios puntos del programa, por ejemplo para guardar y cargar el estado de un formulario, manipulación de *tokens*, obtener parámetros de la aplicación, etc.

5.2.3.2 Ficheros de datos

En esta sección trataremos los ficheros de datos, su ubicación, su formato y su contenido.

5.2.3.2.1 Consultas al diccionario de datos

Todos los ficheros de consultas de datos, están en el directorio <*raiz*>*ConsultasDD*, el contenido de los ficheros son consultas sobre tablas del diccionario de datos, con todas sus columnas, por líneas comentadas, con lo que representa cada atributo. El nombre del fichero coincide con el nombre de la tabla. Por ejemplo, en el Listado 5.4 se ve un ejemplo de la consulta a la vista `ALL_TAB_COMMENTS` del diccionario de datos de Oracle.

Cada uno de estos ficheros se carga desde *Medusa*, al hacer click en cualquier elemento del menú *Diccionario de datos*.

```
-- Vista que muestra información de los comentarios
-- de todas las tablas
SELECT
  OWNER,       -- Propietario
  TABLE_NAME,  -- Nombre de la tabla
  TABLE_TYPE,  -- Tipo
  COMMENTS     -- Comentario
FROM ALL_TAB_COMMENTS
```

Listado 5.4: Ejemplo del formato de la consulta a la vista `ALL_TAB_COMMENTS`

5.2.3.2.2 Plantillas. Autocompletado por código

El formato de estas plantillas es el mismo que el de los formatos DCI que Borland emplea en Delphi. La idea es autocompletar el texto a partir de una cadena inicial que representa el índice del texto que se emplea para autocompletar. Por ejemplo, el texto del Listado 5.5.

```
[putline | Añadir una línea]
DBMS_OUTPUT.PUT_LINE('|') ;
```

Listado 5.5: Item de autocompletado

En el ejemplo, si se teclea en el editor *"*`putline`*"* y se teclea la combinación de teclas `CTRL + J`, se sustituye el texto *"*`putline`*"* que se acababa de teclear por el texto de autocompletado, en este caso es `DBMS_OUTPUT.PUT_LINE('')`. La barra vertical ("|") indica donde se dejará el cursor después de realizar la sustitución.

De forma general, la sintaxis del fichero para cada entrada, sería la mostrada en la Listado 5.6

```
[nombre | descripción]
Líneas de autocompletado
```

Listado 5.6: Sintaxis de un elemento de la lista de autocompletado

5.2.3.2.3 Plantillas. Listas de autocompletado

El formato de estas plantillas es muy simple, consiste simplemente en el nombre de la funcion (procedimiento o paquete), seguido de los argumentos. La idea es autocompletar el texto a partir de una cadena inicial que representa el índice y filtro del texto que se emplea para autocompletar.

```
FUNCTION ABS(N: NUMBER): NUMBER;
```

Listado 5.7: Item de autocompletado

Por ejemplo, si se teclea en el editor *"a"* y se teclea la combinación de teclas `CTRL + Espacio`, se muestra una lista con todas las funciones, procedimientos, variables y paquetes que comiencen por *"a"* (en nuestro caso `ABS` y `ADD_MONTHS`) donde se puede seleccionar cualquier línea y simplemente pulsando <ENTER> se sustituye el texto *"a"* que se acababa de teclear por el texto de autocompletado, en este caso por ejemplo `ABS()`. Si fuese un paquete, además hay que definir otro fichero con todas las funciones de ese paquete, el nombre del fichero debe ser el del paquete. Un ejemplo de la lista de autocompletado, se puede observar en la Figura 5.6

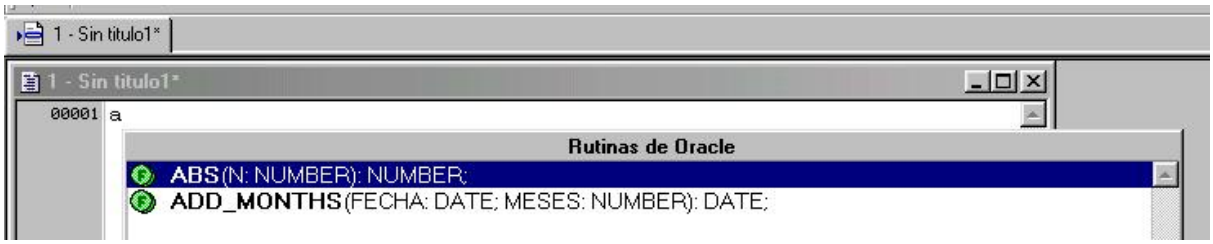

Figura 5.6: Lista de autocompletado visual

5.2.3.2.4 Favoritos

El formato de las sentencias favoritas es similar a los elementos favoritos de Microsoft Explorer. El formato en cuestión, consiste en un fichero *INI* (fichero con estructura, formado por varias secciones, donde cada sección, contiene elementos con datos asociados), formado por las siguientes secciones:

- ℘ `[INFO]`: Contiene los datos de información de la sentencia, por ejemplo, destacamos `Titulo`, que contiene el título de la consulta y `Líneas`, que contiene el número de líneas para poder iterar sobre la sentencia.
- ℘ `[SQL]`: Contiene la sentencia. Cada elemento, se identifica por un número, el número de línea de la sentencia.

```
[INFO]
Titulo=Sur
Lineas=4

[SQL]
0=select *
1=from regiones
2=where ubicacion = 'AMERICA'
3=  and zona = 'SUR'
```

Listado 5.8: Ejemplo de una consulta favorita

Los favoritos, pueden estar organizados en carpetas y subcarpetas, y cada una de éstas, contiene las sentencias, guardadas en los ficheros. En el Listado 5.8 encontramos un ejemplo de una consulta favorita del usuario.

5.3 Desarrollo en general

Medusa en sus comienzos, hace ya un poco de tiempo de eso, nació siendo un cliente de Oracle muy simple, donde simplemente había un editor, una rejilla de resultados y un botón para conectarse. Después poco a poco fue evolucionando hasta convertirse en lo que es ahora, cierto es que es un proyecto muy ambicioso, que podría crecer mucho más pero sobre ese tema, se hablará en la sección de tendencias futuras, un poco más adelante.

Cuando *Medusa* nació, utilizaba otros componentes, en concreto, utilizaba la librería ODAC (*Oracle Direct Access Components*), que aunque es muy similar a DOA, encontré más problemas a la hora de implementar los *scripts*.

La idea de ambas librerías es evitar pasar por el BDE (*Borland Database Environment*) para acceder a los datos, para acelerar el acceso a los datos.

En el esquema de las conexiones necesarias para acceder a los datos mediante el BDE, la gran diferencia es que habría que utilizar los módulos de BDE y SQL Links, como se muestra a continuación.

[Servidor Oracle]↔[SQL*Net]↔[OCI]↔[SQL Links]↔[BDE]↔[Aplicación cliente]

En cambio, con las librerías de interfaz de acceso, se evita pasar por el BDE:

[Servidor Oracle]↔[SQL*Net]↔[OCI]↔[Aplicación cliente]

En aquel entonces, la aplicación era SDI (Single Document Interface), y no soportaba realce sintáctico. Después se añadió precisamente el realce sintáctico, con la ayuda de los componentes SynEdit, y finalmente ya estaba lista la primera versión de *Medusa*. A partir de aquí fue cambiando continuamente, se cambiaron las librerías de ODAC a DOA, se cambió el interfaz de SDI a MDI, se añadieron varias opciones de configuración, y se empezaron a crear formularios que facilitan el acceso a los datos, y así continuó hasta convertirse en lo que actualmente es.

5.3.1 Implementación muy reducida de un cliente de Oracle

Ya que hemos visto un poco la evolución de *Medusa* en el tiempo, es hora de concretar un poco más cómo se ha hecho. Para ello vamos a explicar cómo hacer un cliente de Oracle reducido. Si se desea profundizar más, se puede acceder al código fuente de *Medusa* para ver como se ha hecho exactamente.

1. Lo primero que haremos será crear un nuevo proyecto en Delphi, mediante *New | Aplication*.
2. Agregamos un componente `TOracleSession` (`OracleSession`) y un componente `TOracleLogon` (`OracleLogon`). Ahora asociamos a la propiedad

`Session` del componente `TOracleLogon` el componente de la sesión (`OracleSession`).

3. Agregamos un componente `TOracleDataSet` (`OracleDS`) y un componente `TDataSource` (`DSConsulta`). Ahora asociamos a la propiedad `DataSet` del componente `TDataSource` el componente que representa el conjunto de datos (`OracleDS`). Asociamos también la propiedad `Session` al componente que es la sesión (`OracleSesion`).
4. Agregamos un componente `TDBGrid` (`DBGrdDatos`) y asociamos a la propiedad `DataSource`, la fuente de datos (`DSConsulta`).
5. Agregamos un componente `TMemo (MmoConsulta)`, que será el control de edición de texto.
6. Agregamos un componente `TButton` para iniciar una sesión (`BtnConectar`). En el evento `OnClick` del botón, asociamos el código del Listado 5.9

```
procedure TFrmPrincipal.BtnConectarClick(Sender: TObject);
begin
  OracleLogon.Execute;
end;
```

Listado 5.9: Manejador del evento `OnClick` del botón conectar

7. Agregamos un componente TButton para ejecutar la consulta (BtnConsultar). En el evento OnClick del botón asociamos el código del Listado 5.10.

```
procedure TFrmPrincipal.BtnConsultarClick(Sender: TObject);
begin
  OracleDS.SQL.Assign(MmoConsulta.Lines);
  OracleDS.Open;
end;
```

Listado 5.10: Manejador del evento `OnClick` del botón consultar

Bien, con esto ya tenemos una versión sumamente primitiva de un cliente de Oracle, que puede ser útil para entender las relaciones de los componentes, la forma de iniciar una sesión y la forma de lanzar consultas al servidor. Si el servidor fuese remoto, la única diferencia, es que ten-

dríamos que escribir la cadena de conexión adecuada que nos enlace al servidor, la cual debe estar previamente configurada.

5.3.2 Implementación de un editor simple

Se puede obtener una implementación simple de un editor de texto MDI en **[**W3TIN**]**. No hay mucha diferencia entre un editor de texto SDI y uno MDI, la única diferencia es la complejidad que aporta MDI, que no es muy elevada. En lo que se refiere a la funcionalidad, hay que implementar lo típico, funciones de abrir, guardar, copiar, pegar, etc. Todas estas funciones están ya predefinidas en las acciones que proporciona Delphi.

5.3.3 Solución de algunos de los problemas planteados a lo largo del desarrollo de *Medusa*

Ya hemos visto cómo hacer una versión muy simple de un cliente de Oracle, que es capaz de mostrar las tablas resultantes, utilizando un dataset. Ahora, se puede plantear la pregunta, ¿pero, cómo haría para ejecutar un *script*?, y la respuesta es, utilizar un componente `TOracleScript`, que es capaz de interpretar varias sentencias, e irlas ejecutando una a una de forma ordenada. Claro habría que programar también la lógica para el control general, pero sería similar al caso de las consultas, salvo que los resultados de la ejecución del script no se pueden ver en un `TDBGrid`, sino que son en forma de texto plano, y deben visualizarse en un componente por ejemplo `TMemo`, `TRichEdit` o similares.

A continuación veremos varias preguntas que se podrían realizar cuando se está implementando un cliente de Oracle, junto con sus respuestas, y algunas recomendaciones.

5.3.3.1 Cómo cambiar cadenas en la conexión

En el componente `TOracleLogon`, existen varias constantes de tipo cadena de caracteres, que se utilizan como las etiquetas del texto que aparece en el cuadro de diálogo de conexión a

la base de datos. Las constantes, se pueden conseguir en el fichero OracleLogon.pas y su contenido, es el que muestra el Listado 5.11. Para obtener más detalles, se puede consultar la ayuda de DOA, que viene en formato *doc*.

```
const // Allow translation of the Logon dialog
  // Title of Logon dialog
  ltLogonTitle:     string = 'Oracle Logon';
  // Title of Change password dialog
  ltPasswordTitle:  string = 'Change password';
  // Title of password confirmation dialog
  ltConfirmTitle:   string = 'Confirm';
  ltUsername:       string = 'Username';
  ltPassword:       string = 'Password';
  ltDatabase:       string = 'Database';
  ltConnectAs:      string = 'Connect as';
  ltNewPassword:    string = 'New password';
  ltOldPassword:    string = 'Old password';
  ltVerify:         string = 'Verification';
  ltVerifyFail:     string = 'Verification failed';
  ltChangePassword: string = 'Do you wish to change your' +
                             'password now?';
  ltExpired:        string = 'Your password has expired';
  ltOKButton:       string = 'OK';
  ltCancelButton:   string = 'Cancel';
  ltHistoryHint:    string = 'Logon history';
```

Listado 5.11: Constantes para personalizar el diálogo de conexión

Si queremos cambiar las cadenas de caracteres asociadas al diálogo, tendríamos que hacer algo similar a lo mostrado en el Listado 5.12.

```
    ltLogonTitle   := 'Medusa - Login a la BD';
    ltUsername     := 'Usuario';
    ltPassword     := 'Contraseña';
    ltDatabase     := 'BD';
    ltOKButton     := 'Ok';
    ltCancelButton := 'Cancelar';
    ltConnectAs    := 'Conectar como';
    ltHistoryHint  := 'historial de conexiones';
```

Listado 5.12: Asignación de cadenas de caracteres al diálogo de conexión

5.3.3.2 Destrucción de formularios MDI

La forma de destrucción de todos los formularios MDI hijos creados durante la ejecución del programa, desde el manejador del evento `OnCloseQuery` del formulario principal está descrita en la ayuda de Delphi. Es importante recalcar que la destrucción debe llevarse a cabo desde el índice superior de los formularios MDI creados hasta el índice inferior de los formularios MDI creados, puesto que si se hiciese de la forma contraria, habría errores por referencias inválidas. En el Listado 5.13 se muestra un posible código de destrucción de los formularios hijos creados en tiempo de ejecución.

```
for i := MDIChildCount -1 downto 0 do
begin
  with TFrmEditor(MDIChildren[i]) do
  begin
    // cerramos las conexiones activas
    OracleDataSet.Close;
    OracleSession.LogOff;
  end;
  MDIChildren[i].Close;
end;
```

Listado 5.13: Bucle de destrucción de los formularios creados

5.3.3.3 Los saltos de línea y los puntos y coma en las consultas

Es muy importante comentar, que ocurre un fenómeno un tanto curioso en el componente `TOracleDataSet` y es que éste, no quita los saltos de línea ni los puntos y coma que aparezcan en las consultas que se introduzcan en el editor de textos. Además, si no se eliminan estos caracteres, provoca un error y no funciona la consulta, por tanto la solución es quitar todos estos caracteres antes de asignar el texto de la consulta a la propiedad `SQL` del conjunto de datos. Este error no ocurre en los *scripts* porque es normal que en un script haya varias sentencias una detrás de otra, separadas por punto y coma y además, pueden haber bloques PL/SQL (procedimientos, funciones, *triggers* o bloques no nominados), que en su cuerpo, separan las sentencias también por punto y coma.

Capítulo

6

6 Detalles finales: Generación de la ayuda y de la instalación

Los toques mágicos de un programa, son la ayuda y la instalación. Una buena ayuda o manual de usuario, puede sacar a éste de más de un problema o duda. Y una instalación aunque no es obligatoria, es muy recomendable para facilitar la puesta en marcha y asegurar la integridad de los ficheros.

6.1 Shalom Help Maker

Shalom Help Maker ha sido diseñado, para aquellos programadores, que nunca se aclaran con el proceso de crear ficheros de ayuda, porque sea muy problemático o demasiado costoso adquirir una utilidad de este tipo, o bien demasiado difícil de aprender, etc. Shalom Help Maker, ha intentado eliminar muchos de los dolores de cabeza, del proceso de creación de ficheros de ayuda. Shalom Help Maker se puede descargar desde **[**W3SHA**]**.

Shalom Help Maker, también puede ser utilizado para agrupar imágenes en un fichero que pueda ser abierto en todos los sistemas operativos de Microsoft (Windows 95/98 / ME / NT4 / 2000 / XP). Se puede utilizar para escribir un diario, cartas, manuales, libros en formato electrónico etc. Un fichero de ayuda compilado se puede fácilmente dividir en varias partes, modificar e integrar de nuevo.

Shalom Help Maker es una aplicación completa e integrada, solo necesita el compilador de ayudas de Microsoft, para poder producir los ficheros de ayuda (hlp). Puesto que Windows es un sistema propiedad de Microsoft, solo hay un compilador disponible, y ya no está siendo mantenido. Shalom Help Maker no es compatible con los nuevos formatos de ayuda HTML, basados en ficheros chm.

Al contrario que otros programas para crear ficheros de ayuda Shalom Help Maker funciona como un editor ordinario y por tanto, no hay prácticamente curva de aprendizaje. No es necesario saber nada sobre el proceso de creación de ficheros de ayuda. Solo hay que escribir, insertar imágenes y escribir las cabeceras de las páginas. Se puede ver una previsualización de la página y posteriormente compilar todo el proyecto.

En la Figura 6.1 se puede ver la apariencia que presenta el programa al iniciarlo. También se puede apreciar como, realmente, tiene una apariencia sencilla como la de un editor.

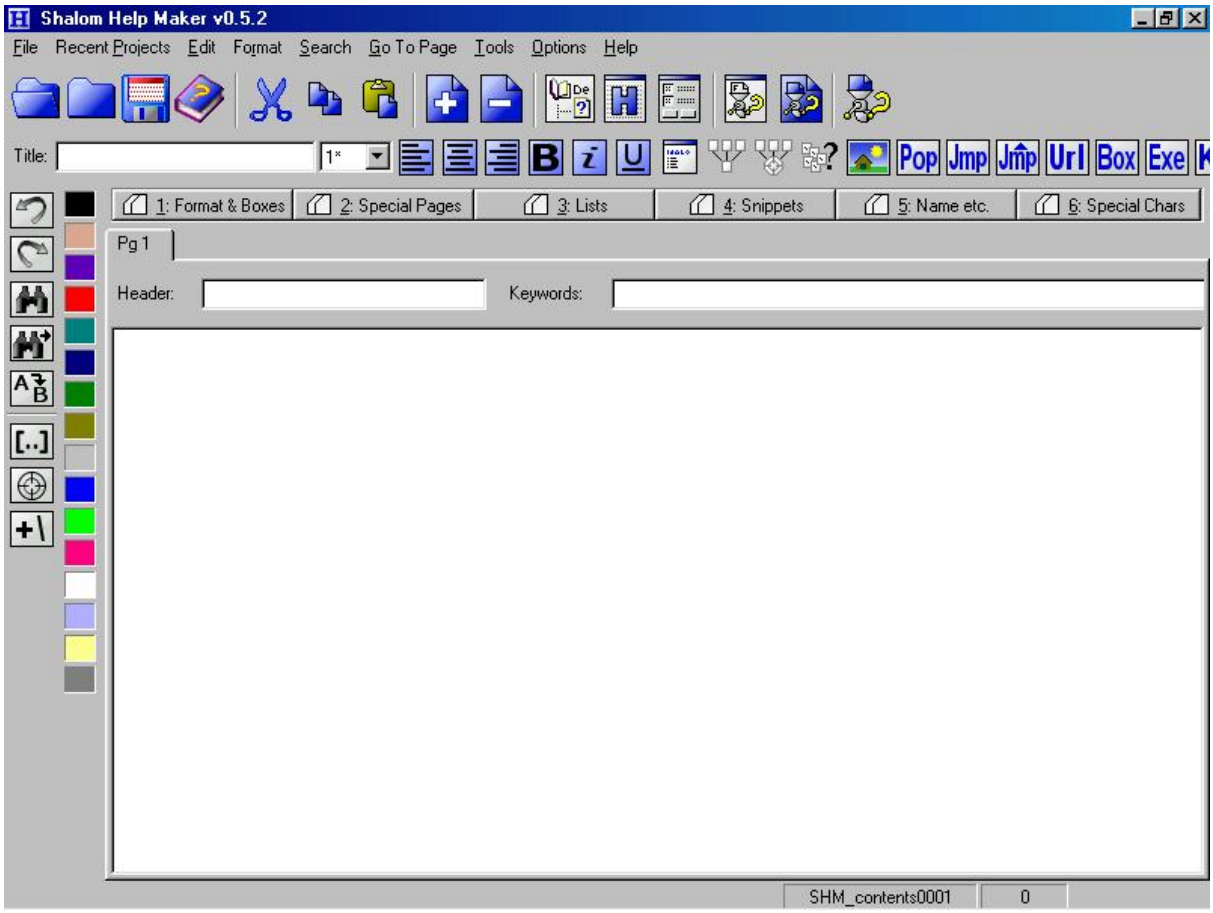

Figura 6.1: Pantalla inicial de Shalom Help Maker

Para obtener más información y una guía de la utilización de Shalom Help Maker se puede consultar el manual **[**SHA03**]**, en el que viene muy bien explicado

6.1.1 Características de Shalom Help Maker

- Formato. Se puede personalizar completamente, pudiendo escoger el tipo de fuente, el tamaño, color e incluso el color de fondo. También se puede personalizar la alineación del texto.
- Enlaces. Se pueden incluir enlaces de todo tipo, enlaces a paginas nuevas (páginas popup), saltos (enlaces) a otras páginas, a páginas web y direcciones de correo electrónico. También se puede ejecutar un fichero externo, simplemente haciendo un click en un enlace, además se pueden insertar macros de ejecución.

- Imágenes. Según el formato de ayuda de Microsoft, solo son reconocidas por el compilador imágenes en formato bmp (y otros formatos menos conocidos como dib, wmf y mrb). Los formatos jpg y png (utilizados en páginas web) no son reconocidos por el compilador.
- Contenidos e índices. En la zona de palabras clave de cada página, se pueden escribir las palabras clave relacionadas, separadas por punto y coma (;). Estas palabras clave se podrán visualizar posteriormente en el índice del fichero de ayuda después de compilar. Las cabeceras (títulos) de páginas, también se incluyen automáticamente como palabras clave.
- Identificadores de tópicos. Cada página tiene un identificador de contexto (*ID number*). Cada página también tiene un identificador de tópico, que se puede utilizar como referencia en los enlaces de tipo popup.
- Tamaño configurable de la ventana. En las opciones del programa, se puede elegir la posición y el tamaño que tendrá la ventana de ayuda. El tamaño más pequeño es de 640 pixels en longitud por 480 pixels en altura. La mayoría se configuran por defecto a 800 x 600. También hay muchos que se configuran a 1024 x 768 pixels, e incluso hay algunos que son mayores. No se debe escoger un tamaño de ventana demasiado grande.
- Compresión. Se puede también seleccionar el nivel de compresión, en la lista desplegable de la barra de herramientas. Existen 7 niveles de compresión aunque se recomienda utilizar en nivel 1, que es el que permite al compilador encontrar el óptimo nivel de compresión.

6.1.2 Restricciones de Shalom Help Maker

En este punto, podemos ver las restricciones del programa, hay que puntualizar, que éstas restricciones vienen impuestas por el compilador de Microsoft.

- Tamaño máximo del fichero de ayuda: 2 Gigabytes.
- Tópicos (páginas) por fichero de ayuda: No hay limitación.
- Longitud de tópicos (páginas): 16.383 caracteres.
- Longitud máxima de las palabras clave: 255 caracteres.
- Titulo de la ayuda: 127 caracteres.
- Titulo de tópico (en cabecera de página): 127 caracteres.
- Imágenes: 65.535 imágenes por fichero de ayuda.

- Nombre de fichero: 259 caracteres (Windows XP puede tener problemas con nombres de ayuda excesivamente largos).
- Contenidos de cabeceras en la organización de las páginas (*Page Organizer*): 9 niveles de tabulación.

Hay que hacer una pequeña nota, y es que el exceder el tamaño de página (16.383 caracteres), produce un fichero de ayuda corrupto y un error por falta de memoria. Esto hay que tenerlo en cuenta para no hacer páginas muy largas.

No hay limites en el número de páginas (tópicos). En el manual se comenta que se han llegado a hacer ficheros de ayuda de más de 200 páginas.

6.1.3 Generando y probando la ayuda

En este apartado veremos como compilar la ayuda y la fase de integración en Delphi.

6.1.3.1 Formas de organizar el proyecto de ayuda

Existen 2 formas de organizar el proyecto de la ayuda: Una es con números de identificación de contexto y otra es con nombres constantes.

6.1.3.1.1 Generando la ayuda con números de identificación de contexto.

Cada página, tiene un número identificador de contexto (*ContextID*). La página inicial, tiene asignada el identificador 9999 (si existe página principal). Cada número identificador de página, puede se puede cambiar en las opciones del proyecto.

La primera página, tiene el identificador 1, la siguiente el 2 y así sucesivamente. Sólo hay que fijarse en la pestaña del editor. Si la pestaña dice *Pg 12*, entonces, el número de identificación de contexto es el 12.

Estos números de identificación de contexto se utilizan en la aplicación para abrir el fichero de ayuda en una cierta página.

6.1.3.1.2 Generando la ayuda con nombres constantes

Este otro método es mucho más recomendable, es preferible usar constantes alfanuméricas que no cambian, cuando se muevan o cuando se añadan páginas. Por el contrario, los números identificadores de tópicos y los números identificadores de contexto si cambian. Por eso es mejor utilizar nombres de página constantes si tenemos la necesidad de hacer referencia a páginas especificas. Shalom Help Maker tiene una opción para generar estas constantes automáticamente.

6.1.3.2 Cómo utilizar la ayuda generada en Delphi

En esta sección, veremos cómo incorporar la ayuda al programa, para poder hacer referencia a ésta, con simplemente pulsar una tecla.

6.1.3.2.1 Indicándole a la aplicación donde se encuentra el fichero de ayuda

Se puede indicar al programa, de dos formas que la ayuda se encuentra en un path determinado. Lo podemos hacer en el inspector de objetos (*Object inspector*) o bien, mediante código. En el se muestra Listado 6.1 un ejemplo de cómo se haría mediante código.

```
procedure TFrmPrincipal.FormCreate(Sender: TObject);
var
  Path : String;
begin
  Path := ExtractFilePath(ParamStr(0)) + 'Ayuda.hlp';
  if FileExists(Path) then
    Application.HelpFile := Path
  else begin
    // Puede ser útil para informar al usuario, en caso de
    // que el fichero de ayuda no existe y poder deshabilitar
    // los menús de ayuda
  end;
end;
```

Listado 6.1: Inicialización del atributo HelpFile

6.1.3.2.2 Abriendo el fichero de ayuda cuando el usuario hace click en un menú

Aquí se muestran tres formas de llamar al programa encargado de mostrar la ayuda, cuyo código se detalla en el Listado 6.2:

- La primera, muestra la página por *defecto* (la página principal).
- La segunda, muestra la ventana de *contenido* del fichero de ayuda.
- La tercera, muestra la pestaña *índice* del fichero de ayuda.

```
procedure TFrmPrincipal.MIAyudaAyudaClick(Sender: TObject);
begin
  // Para mostrar la página por defecto (usamos F1 como
  // tecla rápida, para este elemento de menú):
  Application.HelpCommand(HELP_CONTENTS, 0);
  // La página por defecto, se define en el fichero hpj
  // como 'CONTENTS=SHM_contentsXXXX' (sea la página
  // inicial, o la página 1).
  // Microsoft dice, que este método es antiguo y nos
  // recomienda usar HELP_FINDER en su lugar (ver
  // siguiente). De ese modo, mostramos la ventana de
  // contenido.
end;

procedure TFrmPrincipal.MIAyudaContextoClick(Sender:
                                                TObject);
begin
  Application.HelpCommand(HELP_FINDER, 0);
  // Muestra la ventana de contenido
end;

procedure TFrmPrincipal.MIAyudaIndiceClick(Sender:
                                              TObject);
begin
  Application.HelpCommand(15, -2);
  // Muestra la pestaña índice
end;
```

Listado 6.2: Tres distintas formas de invocar a la ayuda

6.1.3.2.3 Abriendo una página, utilizando su identificador

En este caso se muestran dos métodos, cuyo código está en el Listado 6.3:

- El primero utiliza números de identificación de contexto.

- El segundo, utiliza constantes alfanuméricas. He aquí la ventaja de utilizar las constantes. Además de darnos mayor seguridad en caso de que cambien los identificadores, no tenemos que recordar los identificadores de las páginas.

```
procedure TFrmPrincipal.MIAyudaTeclaRapidaClick(Sender:
                                                    TObject);
begin
  Application.Helpcontext(5); // Muestra la página 5
end;

procedure TFrmPrincipal.MIAyudaComienzoRapidoClick(Sender:
                                                   TObject);
begin
  // Utilizando una constante de página (recomendado)
  Application.Helpcontext(hlp_QuickStartPleaseRead);
  // Muestra la página QuickStartPleaseRead (sea cual sea
  // su número de identificación)
end;
```

Listado 6.3: Dos métodos para abrir una página determinada

6.1.3.2.4 Finalizar la ayuda cuando finaliza el programa

Finalmente en el Listado 6.4 vemos como cerrar el fichero de ayuda en caso de que esté abierto y el usuario cierre el programa.

```
procedure TFrmPrincipal.FormClose(Sender: TObject;
var Action: TCloseAction);
begin
  Application.HelpCommand(HELP_QUIT, 0);
  // Cierra el fichero de ayuda si está abierto
end;
```

Listado 6.4: Cerrando el fichero de ayuda, cuando finaliza el programa

6.2 InstallShield Express – Edición limitada para Borland

El toque final del desarrollo de una aplicación es la generación de un programa para su instalación. Para poder distribuir aplicaciones con determinados componentes en Delphi, suele ser necesario en ocasiones, distribuir también las librerías compiladas u otros ficheros. También puede ser interesante hacer una instalación para distribuir la estructura de una base de datos, o por ejemplo, si la aplicación trabaja con múltiples ficheros (como pueden ser ficheros de ayuda y la documentación en línea) no tener que distribuirlos por separado.

A partir de la versión 3 de InstallShield también se hace necesario distribuir los paquetes, para las aplicaciones que hacen uso de ellos, lo que añade más complejidad a una instalación típica. Borland/Inprise incluye con Delphi una versión reducida del programa de creación de instalaciones InstallShield. En la Figura 6.2 podemos ver la pantalla inicial del programa.

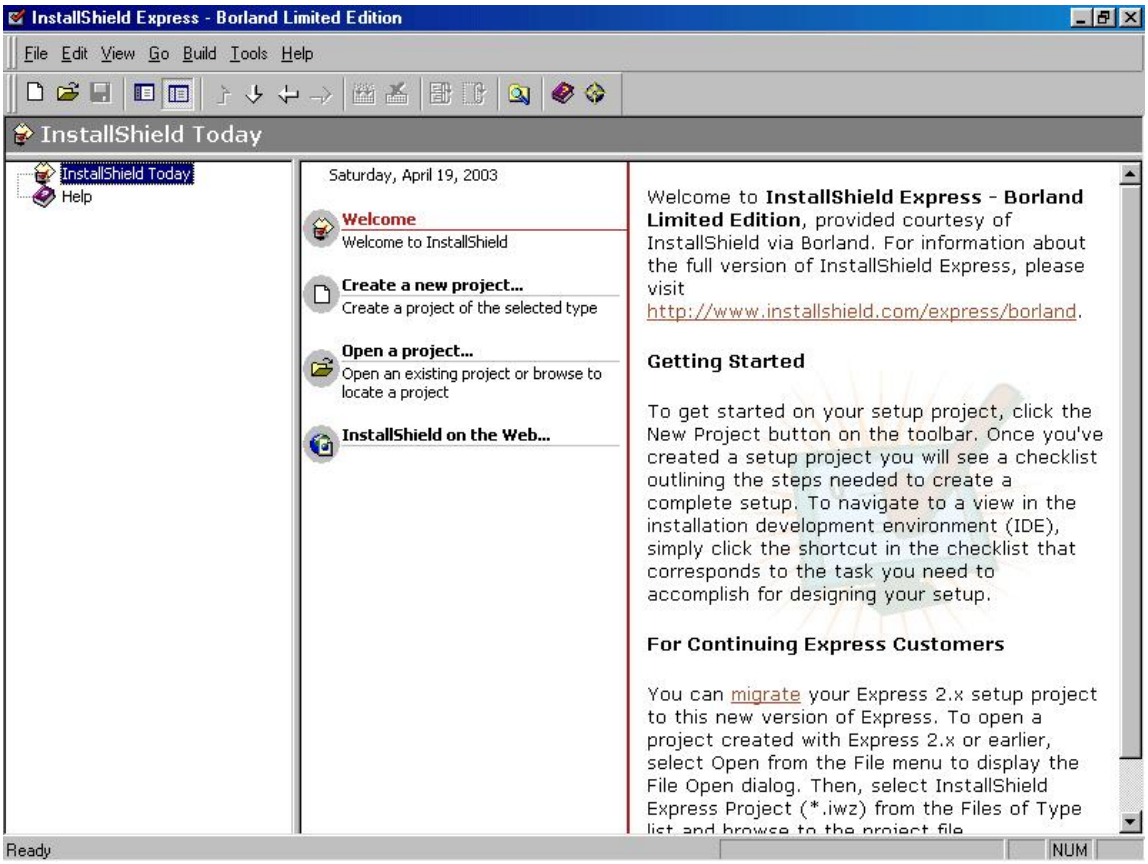

Figura 6.2: Pantalla inicial de InstallShield Express – Edición limitada para Borland

Para obtener más información de la utilización de InstallShield Express – edición limitada para Borland, se puede consultar **[**INS01**]**.

6.2.1 Los proyectos de InstallShield Express

InstallShield no se instala automáticamente junto con Delphi, y se debe ejecutar la opción correspondiente del programa de instalación global, que aparece al insertar el CD-ROM en el ordenador. Una vez que el programa se ha copiado en el ordenador, podemos acceder al mismo por medio de un acceso directo que se coloca directamente en el menú de inicio de Windows. InstallShield Express nos permite generar las instalaciones mediante una serie de diálogos de configuración en los cuales debemos indicar qué ficheros debemos copiar, qué parámetros de Windows debemos cambiar y que interacción debe tener el programa de instalación con el usuario. Hay que aclarar este punto, porque existen versiones de InstallShield en las cuales las instalaciones se crean compilando un *script* desarrollado en un lenguaje de programación al estilo C. Evidentemente, el trabajo con estas versiones es mucho más complicado.

Para comenzar a trabajar con InstallShield necesitamos crear un `proyecto`, que contendrá los datos que se suministrarán al generador. Los proyectos se almacenan en ficheros de extensión iwz. Cuando se crea un proyecto nuevo, hay que indicar el nombre y el directorio donde queremos situar este fichero. Si el directorio no existe, podemos crearlo tecleando su nombre en el cuadro de edición *New subdirectory*. También podemos abrir un proyecto existente, o seleccionarlo de la lista de los últimos proyectos cargados.

En el cuadro de diálogo de creación de proyectos hay una casilla importante con el título *Include a custom setup type*. Debemos marcar esta casilla si queremos que el usuario de la instalación pueda seleccionar los componentes de la aplicación que quiere instalar y los que no. Si no incluimos esta posibilidad al crear el proyecto se puede indicar más adelante, pero nos costará más trabajo hacerlo.

6.2.2 Características de InstallShield Express

Algunas de las características que podremos encontrar en InstallShield, son:

- Realizar la transferencia de ficheros. La transferencia de ficheros, involucra el copiado de ficheros desde un medio fuente, como un CD o un disquete, al disco duro local de la máquina del usuario final. Dependiendo de la configuración que el usuario seleccione, todos o solo algunos se transferirán al disco local. Durante la transferencia de ficheros, una instalación, es capaz de mostrar anuncios que proporcionen información sobre el producto como nuevas características o consejos de uso. También se suele mostrar un indicador de estado, para mostrar el progreso del proceso de la transferencia.
- Gestión del interfaz de usuario. La interfaz de usuario de una instalación, proporciona información y la posibilidad de la elección de las opciones de configuración al usuario final. Mediante la interfaz de usuario, el usuario final, puede elegir el instalar solo una parte de la aplicación, dejar ciertos ficheros en el medio fuente, ver la licencia del programa, o proporcionar cierta información a la instalación que puede ser necesaria para asegurar la configuración adecuada de la instalación. El interfaz de usuario, puede ser personalizado para satisfacer cualquier necesidad que se pueda tener. Por ejemplo, se puede solicitar al usuario un serial (código o número de serie), antes de iniciar la instalación, si se desea proteger el software contra el uso ilegal.
- Creación de accesos directos. Los accesos directos, son enlaces a ficheros y aplicaciones que se pueden crear en la máquina del usuario final durante la instalación. Los accesos directos, se pueden ubicar en el escritorio o en el menú de inicio de la máquina, para garantizar un acceso rápido al programa o a los ficheros.
- Registrar asociaciones de ficheros. Si el programa usa distintos tipos de ficheros, será necesario registrarlos en el sistema. Por ejemplo, Notepad crea un fichero con una extensión txt. Con la finalidad de que se reconozca por el sistema, será necesario almacenarlo en el registro de Windows. El proceso de registro de un tipo de fichero, se puede gestionar desde la instalación.
- Registrar ficheros COM, COM+ y DCOM. Los servidores COM (como ActiveX, COM y COM+) requieren un registro especial, de forma que las aplicaciones puedan acceder a los ficheros del interfaz. Tradicionalmente, estos ejecutables y librerías, contienen una función de auto-registrado que puede ser invocada para registrar los ficheros durante la instalación.

No obstante, confiar en el auto-registrado, puede causar algunos problemas, como que el usuario no pueda estar seguro de que información está siendo registrada o de que el registro sea limpiado cuando los ficheros sean desinstalados. La solución que ofrecen las instalaciones, es guardar sólo lo necesario en el registro durante la instalación y borrarlo después cuando el elemento sea desinstalado. Este método ayuda a asegurar que todos los servidores COM serán registrados de forma adecuada.

- Registrar la desinstalación del producto. Con la finalidad de desinstalar el producto, el sistema operativo, debe saber que el producto está instalado. Para ello, la instalación registra el producto en el sistema operativo, de forma que esa labor se pueda hacer de forma sencilla.

6.2.3 Restricciones de esta edición de InstallShield Express

InstallShield Express – Edición limitada para Borland, no tiene todas las posibilidades de la versión completa de InstallShield Express. Algunas de las características que no están disponibles en la versión de edición limitada para Borland son:

- Anuncios. Los anuncios están deshabilitados en la versión de edición limitada. Con la versión completa de InstallShield Express, se pueden incluir anuncios en los proyectos de instalación, para informar o entretener al usuario durante el proceso de instalación.
- Acciones personalizadas. La versión limitada, no soporta acciones personalizadas, por ejemplo como llamar una función de una DLL o ejecutar un programa.
- Enlazado de ficheros dinámico. La posibilidad de añadir ficheros a la instalación de forma dinámica, tampoco está soportada por la versión limitada. En la versión completa de InstallShield Express, se puede añadir el contenido de un directorio completo en el proyecto de instalación, mediante el enlazado dinámico de ficheros. El directorio de vinculación es revisado cada vez que se compile el proyecto de instalación y los ficheros se incluyen de forma automática.
- Variables de entorno. En la versión limitada, no existe la posibilidad de ver, crear, modificar y eliminar variables de entorno en el ordenador destino, mediante el programa de instalación.

- Globalización. Cuando se crea un proyecto, la edición limitada, nos limita solo al ingles. Se puede importar y exportar tablas de cadenas de caracteres con las traducciones en la versión completa de InstallShield Express.
- Asistentes de búsqueda. El asistente de búsqueda estático y dinámico, no están disponibles en la edición limitada. En la versión completa, se pueden utilizar estos asistentes para buscar en los proyectos, dependencias y añadir los ficheros necesarios al proyecto.
- Caminos de actualización. Los caminos de actualización están deshabilitados en la versión de edición limitada. En la versión completa de InstallShield Express, se puede identificar información de actualización del producto.

6.2.4 La versión completa de InstallShield Express

InstallShield Express, la versión completa del producto que se puede comprar por separado, ofrece características interesantes para el programador. Estas son algunas de ellas:

- Se incluyen módulos adicionales de lenguajes, para cambiar el lenguaje de la instalación.
- Se pueden desarrollar instalaciones para plataformas de 16 y 32 bits.
- Los sistemas de desarrollo soportados son varios: Delphi, Borland C++ y C++ Builder, Visual Basic, Optima++, etc.
- Es posible ejecutar *extensiones* de InstallShield desde la instalación. Estas son DLLs o ejecutables desarrollados por el programador que realizan acciones imposibles de efectuar por InstallShield.
- Se pueden modificar los ficheros *autoexec.bat* y *config.sys*. Esto es indispensable en instalaciones para 16 bits.
- Se pueden mezclar ficheros de extensión reg en el Registro. Esta técnica permite registrar fácilmente los controles ActiveX.
- Los accesos directos y entradas en el menú que se crean para las aplicaciones tienen más opciones de configuración. Podemos, por ejemplo, indicar que cierta aplicación se ejecute minimizada o maximizada, y colocar una aplicación en otra carpeta, como la de aplicaciones de inicio.

- Se pueden indicar ficheros temporales a la instalación, que se eliminan una vez finalizada la misma, dejar espacio en el primer disco para ficheros que se instalan sin descomprimir, y crear instalaciones auto-ejecutables desde un solo fichero ejecutable.

6.2.5 Generando y probando la instalación

Cuando todos los parámetros de la instalación han sido suministrados, es hora de crear los discos de la instalación, utilizando la opción *Disk Builder*. Podemos, antes de generar la instalación, indicar el formato en que queremos los discos. InstallShield permite utilizar diversos formatos de discos.

En cualquier caso, la escritura no se efectúa directamente sobre el medio seleccionado. El generador de instalaciones crea un subdirectorio bajo el directorio del proyecto, utilizando el nombre del formato. Por ejemplo, si seleccionamos discos de doble densidad de 3.5 pulgadas, el subdirectorio se llamará *144mb*. Bajo este subdirectorio, a su vez, se crearán dinámicamente otros subdirectorios llamados *Disk1*, *Disk2*, etc., con el contenido de cada uno de los discos de instalación. Una vez elegido el formato de los discos, pulsamos el botón *Build* para generar sus contenidos. Se puede utilizar este comando varias veces, con formatos diferentes, si necesitamos distribuir la aplicación en soportes de diferente tamaño. Finalmente, se puede probar la instalación desde la opción *Test Run*. Hay que tener en cuenta que el programa se instalará realmente en el ordenador, pues no se trata de simular la instalación.

Capítulo

7

7 *Medusa*: Manual de usuario

Un aspecto muy importante que nunca se debe descuidar, es el manual de usuario y la ayuda. Una buena ayuda o manual de usuario, puede proporcionar al usuario una fácil aproximación a la aplicación, ayudándole a comprender cómo utilizarla, junto con los riesgos que puede conllevar la mala utilización del programa, así como brindarle pequeños consejos que le hagan más amena la interacción con la aplicación.

7.1 Dando nuestros primeros pasos

Medusa es un completo cliente de Oracle. Como es lógico, posee un completo editor de texto, con muchas posibilidades tanto básicas (copiar, cortar, pegar, seleccionar todo...), como avanzadas (*bookmarks*, realzado de sintaxis, marcas de *offset*, numeración de líneas...). Para facilitar el trabajo por parte del usuario, se ha incluido dentro de la misma ventana, donde se encuentra el editor, una rejilla donde se obtendrán los resultados, en caso de que se ejecute en modo ***consulta*** (este modo sólo se emplea cuando se deseen realizar consultas), o una consola donde se observarán los resultados en caso de que se ejecute en modo ***script*** (este modo se emplea para el resto de casos, como son sentencias DDL, inserciones de datos, actualizaciones, eliminación de datos, bloques PL/SQL, opciones de administración como por ejemplo concesión de permisos, etc). Éstos elementos, se encuentran en la parte inferior de la ventana de edición, organizados en pestañas, para poder cambiar fácilmente entre ellos. Igualmente, se dispone de una pestaña que nos informa de los posibles errores que puedan haber en las sentencias (tabla inexistente, falta de una coma...), marcándolos en caso de producirse en el *gutter* (margen izquierdo del editor, que también se utiliza para numerar las líneas, poner *bookmarks*... y que puede verse en la Figura 7.5).

7.2 El entorno de Medusa, una primera aproximación

Primero, debemos instalar la aplicación. Para instalar la aplicación, será necesario hacer doble click sobre el ejecutable "Setup.exe", dentro de la carpeta de instalación. El proceso de instalación es similar al de cualquier otra aplicación del sistema operativo Windows, donde se pedirán diversos datos como por ejemplo, el directorio donde se instalará la aplicación.

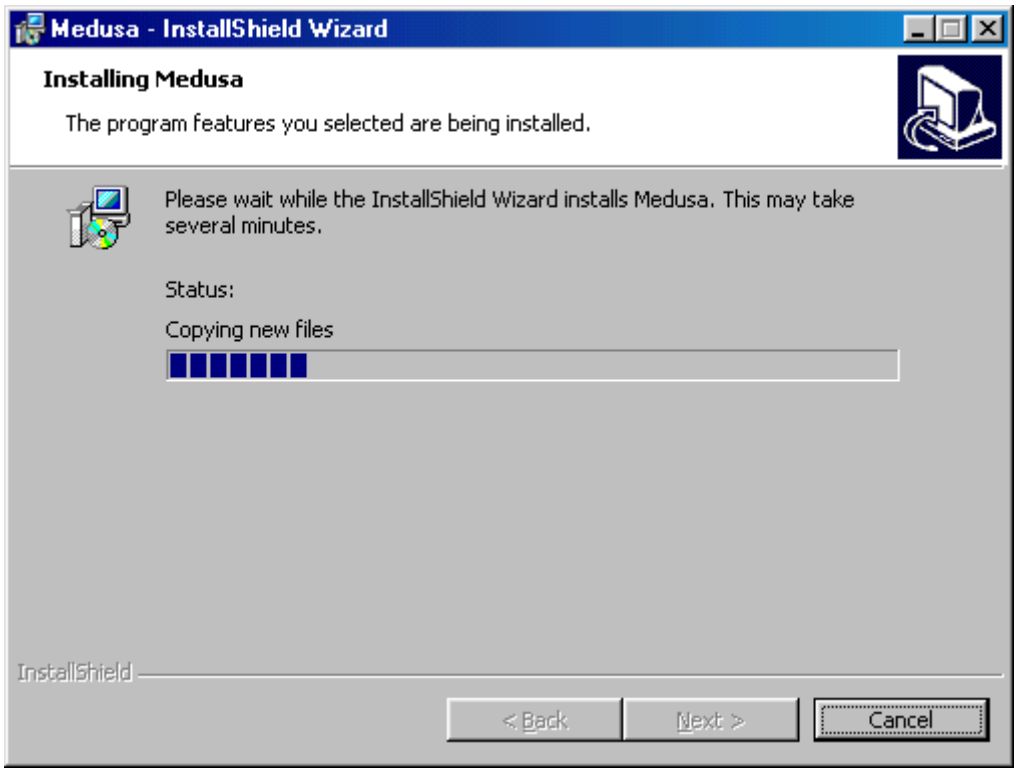

Figura 7.1: Aspecto de la ventana de instalación de Medusa

El proceso de desinstalación, se puede hacer de dos formas. La primera es igual que el proceso de instalación, pero seleccionando borrar en el formulario que muestra las opciones de instalación. La segunda es desde el panel de control, en la opción *Agregar o quitar programas*.

Una vez que esté instalada la aplicación, debemos iniciarla, lo cual lo podemos hacer, realizando click en el acceso directo que encontraremos en el menú de inicio. Una vez que hayamos hecho esto observaremos que sale una ventana de carga del programa, que se puede observar en la Figura 7.2. El propósito de este formulario, es simplemente mostrar el estado de la carga de la aplicación, junto con la versión del programa. Principalmente tiene un interés meramente estético. Una vez que esté cargada la aplicación, tendremos acceso a la ventana principal de *Medusa*.

Figura 7.2: Aspecto del formulario de *carga*

La ventana principal del programa, presenta un aspecto similar al que se observa en la Figura 7.3. La ventana principal, es una ventana muy simple, con pocas opciones a priori, puesto que nos servirá simplemente de interfaz para crear nuevas ventanas de edición, ya que es una aplicación con una interfaz MDI (*Multiple Document Interface*, Interfaz de Múltiples Documentos).

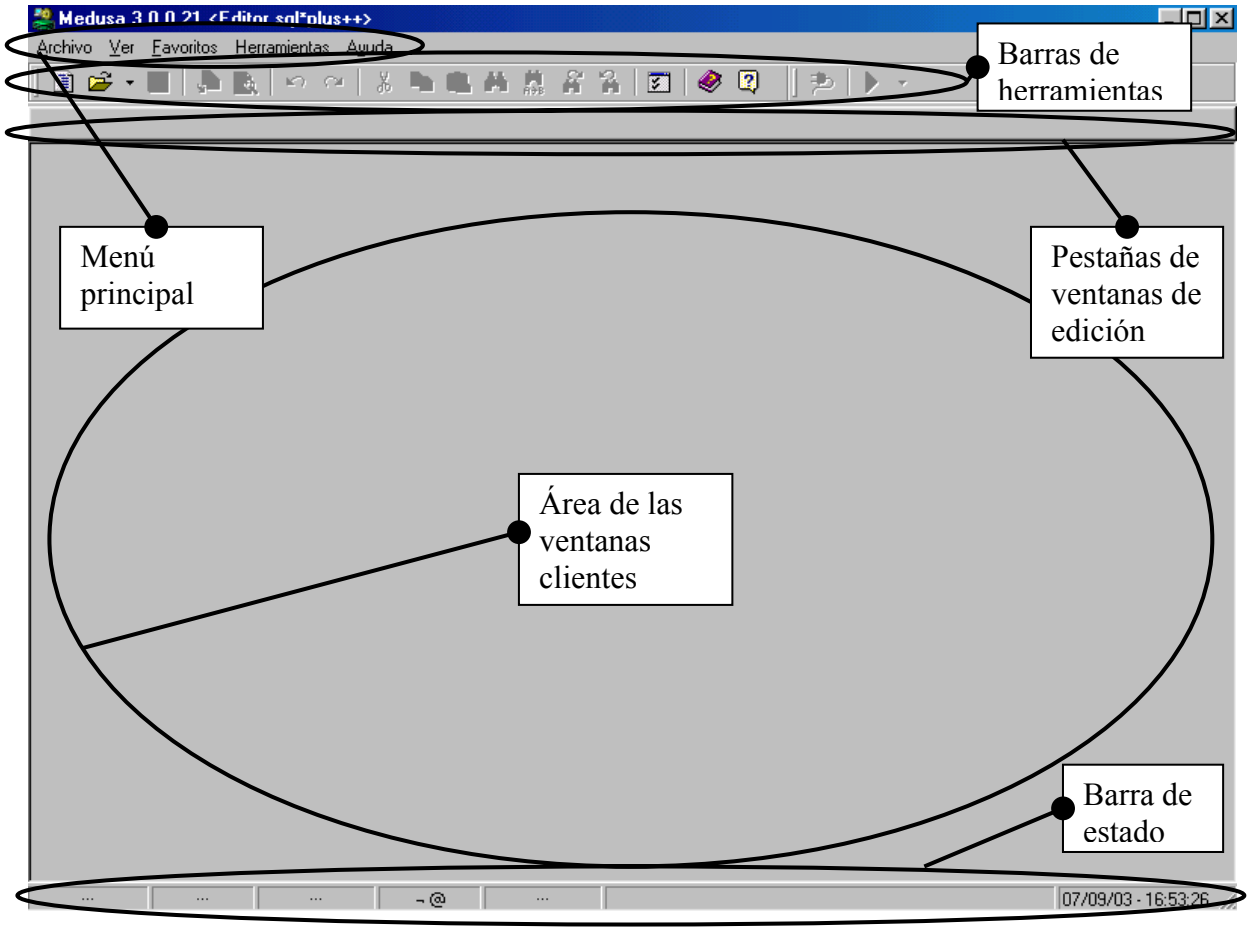

Figura 7.3: Aspecto de la ventana principal de *Medusa*

La ventana principal, está compuesta por cinco zonas bien diferenciadas, tal y como se ilustra en la Figura 7.3.

- Menú principal. Se descompone a su vez en cinco sub-menús. Todos los menús, se explicarán con más detalle, un poco más adelante.
  - Menú *Archivo*
  - Menú *Ver*
  - Menú *Favoritos*
  - Menú *Herramientas*
  - Menú *Ayuda*

- Barras de herramientas. Son accesos directos a las opciones de menú. Con la idea de facilitar la utilización del programa, se han incluido estos accesos a las opciones de los menús más frecuentemente utilizadas.
- Pestañas de ventanas de edición. Son pestañas vinculadas a cada una de las ventanas cliente, para poder acceder de forma sencilla a una ventana en caso de haber varias ventanas superpuestas. En la Figura 7.3 no hay ninguna pestaña debido a que no hay ninguna ventana activa en ese momento. Cuando se dispone de una ventana de edición con su pestaña asociada, la pestaña refleja en todo momento el estado del editor, es decir, si ha sido modificado, o no, si ha sido guardado, etc. Más adelante se explicará detalladamente. Se puede cambiar de ventana mediante las combinaciones de teclas CTRL + TAB para ir a la siguiente ventana o SHIFT + CTRL + TAB para ir a la ventana anterior.
- Área de las ventanas clientes. Es el espacio de la aplicación reservado para las ventanas clientes, donde podremos tener varias ventanas a la vez y estar trabajando sobre la ventana activa.
- Barra de estado. Se descompone a su vez en siete paneles, de izquierda a derecha:
  - Panel 1. Muestra información sobre número de línea y el número de columna en la que se encuentra el cursor en ese momento. En la Figura 7.3 no se observa nada porque no hay ninguna ventana de edición activa.
  - Panel 2. Muestra información sobre el *offset* del cursor. El usuario puede al hacer click sobre el área del *gutter* correspondiente a una línea para, colocar una marca de *offset*, la cual se tendrá en cuenta para mostrar la numeración relativa a esta línea marcada. Esta funcionalidad es útil, sobre todo para poder depurar los errores en procedimientos y funciones, puesto que Oracle proporciona una numeración de estos elementos, de forma relativa y no absoluta a todo el documento. Sólo puede haber una marca en todo el documento.
  - Panel 3. Muestra el modo de edición. Los modos de edición disponibles son actualmente tres: Inserción (se puede insertar texto en cualquier parte). Reescritura (se puede sobre escribir el texto borrando lo que hubiera escrito). Solo lectura (el fichero no se puede modificar).
  - Panel 4. Muestra el nombre del usuario conectado a la base de datos, junto con el enlace de la base de datos al que está conectado. El formato es el siguiente.

    Usuario@Enlace

- Panel 5. Muestra el número de filas obtenidas al realizar la consulta a la base de datos. Si además estamos navegando a través de los datos (moviéndonos a través de la rejilla), también se mostrará el índice de la fila actual.
- Panel 6. Muestra una breve ayuda sobre la función de un determinado elemento de la aplicación. Por ejemplo, en el caso de los menús, se observa una breve descripción de lo que realiza el menú sobre el que el ratón esté situado.
- Panel 7. Muestra la fecha y la hora actual del sistema.

7.3 Breve introducción

Para poder comenzar a trabajar realmente con toda las opciones que ofrece *Medusa*, debemos crear una ventana de edición, sea creando un nuevo fichero (*Archivo* | *Nuevo*) o abriendo un fichero existente (*Archivo* | *Abrir* | *Abrir*) o abriendo un documento reciente (*Archivo* | *Abrir* y seleccionando un elemento del menú de la parte inferior). Una vez que creemos una nueva ventana de edición, descubriremos que se habilitan multitud de opciones nuevas en los menús. Antes de ver las nuevas opciones, veremos primero el formulario de conexión a la base de datos detalladamente, el cual aparecerá automáticamente en función del protocolo de conexión (se explica detalladamente en la sección 7.11.4) o si pulsamos el botón de conexión a la base de datos.

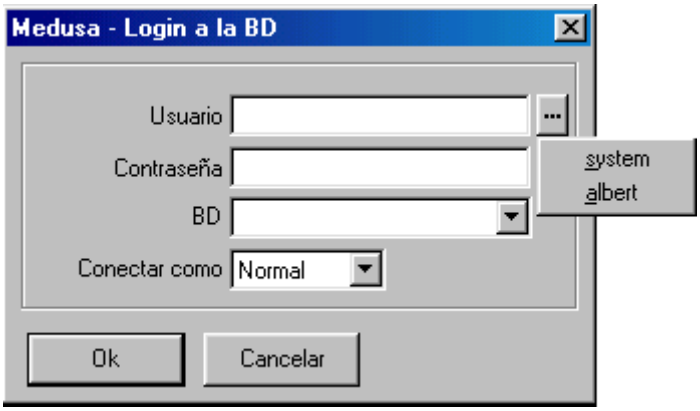

Figura 7.4: Diálogo de conexión a la base de datos

En la Figura 7.4 se observa, que hay un campo para ingresar el nombre de usuario, y otro para la contraseña, que son los campos más típicos para la autentificación de usuarios. También se ha incluido una lista desplegable para poder seleccionar un posible enlace a una base de datos, en caso de existir. Esta lista se carga automáticamente con todos los enlaces a bases de datos existentes en la máquina. Igualmente se dispone al lado del campo del nombre de usuario, de un botón, que al pulsarlo nos proporciona, una lista con los usuarios que suelen utilizar el sistema (se va actualizando automáticamente), de forma que se facilite el inicio de sesión. Por último, también disponemos de una lista desplegable con los posibles privilegios de administración (para poder emplearlos, se deben poseer, al igual que cualquier otro privilegio normal del sistema). Estos privilegios son: `SYSDBA` (contiene todos los privilegios del sistema `WITH ADMIN OPTION` y permite usar `CREATE DATABASE`), `SYSOPER` (puede hacer casi cualquier tarea de administración, excepto crear una base de datos, conceder la administración a otros y poco más). También existe la opción *Normal*, que permite la conexión por defecto.

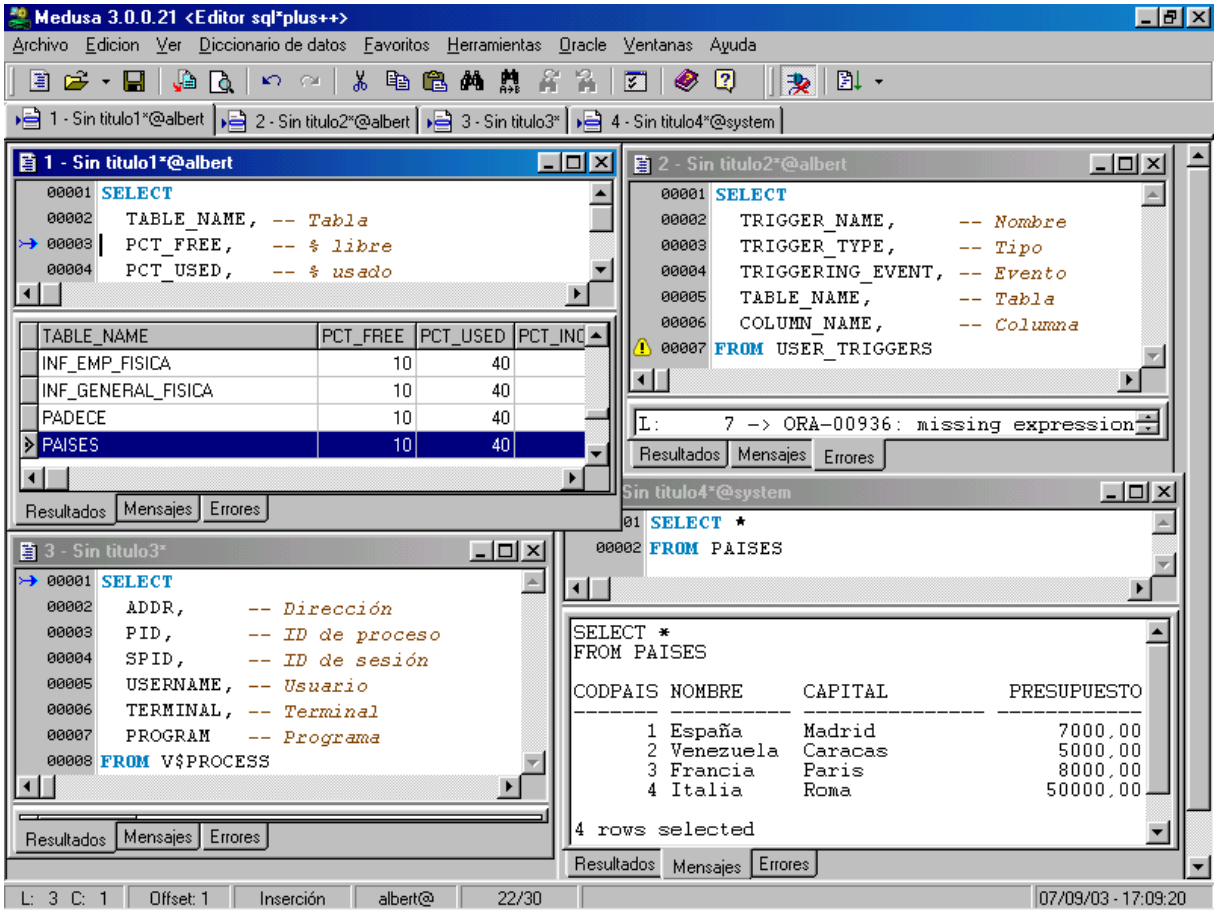

Figura 7.5: Aspecto de *Medusa* con tres ventanas de edición

En la Figura 7.5 se muestra el aspecto de *Medusa* trabajando con cuatro ventanas de edición. Entre los aspectos a destacar de ésta, encontramos que han aparecido en el menú principal nuevas opciones que antes no eran visibles. Igualmente, se han habilitado nuevos botones en la barra de herramientas. Las pestañas de las ventanas muestran el nombre del fichero que se está editando, junto con el nombre del usuario conectado, así como el estado de edición (modificado: el cual se representa con un asterisco, no modificado: no se muestra el asterisco). También se puede ver el realzado de sintaxis (distinto color y atributo para las palabras reservadas de SQL, como por ejemplo `SELECT`, `FROM`...).

En la Figura 7.5 está activa la ventana número uno y en ella se puede apreciar el funcionamiento de la marca de *offset* (pequeña flecha de color azul situada en el *gutter* del editor), cuyo valor relativo, se indica en la barra de estado (concretamente en el segundo panel, como se ha explicado con anterioridad). También se puede ver el resultado de la consulta (rejilla en la pestaña *resultados*), el cual está situado en la parte inferior del área de edición (igualmente se puede ver el número de fila seleccionada y el número total de filas en la barra de estado, tal y como se ha explicado anteriormente).

En la ventana número dos, se puede ver que se ha producido un error (se muestra en la pestaña *errores*, junto con su descripción). La línea del error producido, se marca con el símbolo de exclamación en un triangulo amarillo en el *gutter*. En la ventana número tres, se puede ver que no se ha iniciado aun la sesión y por tanto se está utilizando exclusivamente como editor (lógicamente, la rejilla de resultados está en blanco). Observe que el título de esa ventana no incluye el nombre del usuario conectado. Finalmente la ventana número cuatro, contiene una consulta que se ha ejecutado en modo *script*. Los resultados, junto con los mensajes de Oracle, se pueden observar en la pestaña *mensajes* en modo texto sin utilizar la rejilla.

Todas las ventanas de opciones y diálogos, a excepción de las ventanas de edición tienen un formato muy similar. Todas incluyen un botón *Cancelar* para cancelar la operación y un botón *Aceptar* para confirmar y proseguir con la acción. Se dispone de un acceso directo al botón *Cancelar* para todos los formularios, que es la tecla escape (ESC).

7.4 Introducción a los menús de la aplicación

Tal y como se ha comentado anteriormente, al crear una nueva ventana de edición, tendremos accesibles numerosas opciones nuevas que antes no podíamos utilizar. Entre estas nuevas opciones, podemos destacar:

7.4.1 Menú Archivo

Está compuesto por diversos elementos:

- Nuevo. Crea un documento nuevo.
- Abrir. Permite abrir un documento del disco (se incluye una lista con los documentos recientes).
- Cerrar. Cierra la ventana activa.
- Guardar. Guarda el documento en disco
- Guardar como. Permite guardar el documento con otro nombre
- Exportar. Exporta el documento a otros formatos (como texto o HTML). Se pueden exportar tanto consultas como *scripts*. En el caso de las consultas, si se ha ejecutado la consulta que se desea exportar, también se exportará la tabla resultante. La exportación a formato HTML, se explica con más detalle en la sección 7.5.1.
- Previsualizar. Permite previsualizar el documento antes de imprimir (ver sección 7.5.2).
- Configurar la impresora. Nos permite configurar los parámetros de la impresora antes de imprimir.
- Imprimir. Imprime el documento de la ventana activa.

En la sección 7.5 se explica con detenimiento la exportación a otros formatos, como por ejemplo HTML y la previsualización de impresión de un documento.

7.4.2 Menú Edición

Este menú, al que antes no teníamos acceso, permite interactuar con las acciones más habituales en la edición de texto, como son: deshacer la última acción, rehacer la última acción, copiar, cortar, pegar, seleccionar todo, buscar texto, reemplazar texto y finalmente, también nos permite el acceso al *buffer* de textos (se explica de forma detallada, un poco más adelante en la sección 7.6.4). Las opciones de deshacer y rehacer pueden ser configuradas en cuanto al tamaño de entradas se refiere. Si desea saber cómo puede configurarlas, puede consultar la sección 7.11.1 que hace referencia a la configuración del editor.

7.4.3 Menú *Ver*

Nos permite interactuar con la configuración del aspecto visual de la aplicación. Podemos mostrar u ocultar las barras de herramientas, mostrar u ocultar la barra de estado y también mostrar u ocultar el *gutter* del editor.

7.4.4 Menú Diccionario de datos

Nos ofrece un fácil acceso a algunas de las consultas más usuales al diccionario de datos. El formato de las consultas es detallado, informando de todas las columnas, su significado y el significado de la vista en cuestión. Permite que esas consultas puedan modificarse. Esto se explica con más detalle en el apartado 7.7.

7.4.5 Menú Favoritos

Nos proporciona un fácil acceso a las sentencias favoritas del usuario. De una forma sencilla, podremos cargar una consulta o un *script* para poder editarlo como si fuese un fichero base o ejecutarlo, cuando se deseen realizar tareas repetitivas y se puede configurar, aunque se verá un poco más adelante en la sección 7.11.6.

7.4.6 Menú Herramientas

Nos permite de forma sencilla, el acceso a la configuración de la aplicación, así como a la lista configurable de los programas de usuario. La configuración de *Medusa*, se trata con detenimiento en la sección 7.11.

7.4.7 Menú Oracle

Sin duda el menú más importante de *Medusa*. En este menú, encontraremos las siguientes opciones: (las cuales, se detallan una a una a lo largo de este capítulo).

- Conexión a la base de datos. Permite asociar una nueva sesión a la ventana activa (ver sección 7.3).
- Ejecución de consultas / *scripts*. Permite enviar el documento al Sistema Gestor de Bases de Datos (SGBD) de Oracle para recibir las respuestas. Existen distintas formas de creación de sesiones, pero estas se tratarán en detalle más adelante, concretamente en la sección 7.11.4.
- Obtención información de la sesión (debemos estar conectados). Se muestran varios datos importantes, como la versión de Oracle instalada, el identificador de sesión, el identificador de usuario, los tablespaces (por defecto y temporal) y alguna información adicional, como los roles que posee el usuario, las limitaciones y los privilegios (ver sección 7.8.1).
- Obtención la estructura de la base de datos (debemos estar conectados). A partir de un conjunto de tablas de entrada, se puede obtener un esquema de sus relaciones, así como la estructura global de la base de datos. Se puede encontrar una descripción más detallada en la sección 7.8.2.
- Creación objetos de la base de datos. Se han incluido dentro de *Medusa*, formularios para facilitar la creación de diversos objetos de forma visual tales como procedimientos, funciones, disparadores, paquetes (definición y cuerpo), bloques java, secuencias y sinónimos todas estas opciones, pueden ser de utilidad para los usuarios menos experimentados. Todas estas funciones se tratan en la sección 7.8.3.

- Reconstrucción de la sentencia `CREATE TABLE`. Una opción muy interesante es la reconstrucción de una tabla, que consiste en que a partir del nombre de una tabla, se obtengan todos sus atributos, tipos, restricciones, comentarios, etc. (ver sección 7.8.4).
- Gestión visual de permisos. Nos permite conceder y revocar permisos fácilmente, de forma visual. Se pueden encontrar varios tipos de permisos, como son, permisos del sistema, roles y privilegios sobre diversos tipos de objetos (como tablas, procedimientos, funciones...). Se explica detalladamente esta opción en la sección 7.8.5.
- Gestión visual de comentarios. Nos permite crear comentarios sobre tablas y sobre columnas, de forma visual. Esta opción se ha incluido puesto que los comentarios generalmente son una de las opciones de las cuales casi nunca se saca todo el provecho que se podría. Se explica de forma detallada un poco más adelante en la sección 7.8.7.
- Generación de formatos de fechas de forma gráfica. Nos permite de forma gráfica, la creación de formatos personalizados de fechas. Se pensó que sería muy útil esta opción, puesto que los formatos de fechas suelen ser fáciles de olvidar (ver sección 7.8.6).

7.4.8 Menú Ventanas

En este menú encontraremos todas las opciones usuales de gestión de ventanas de una aplicación MDI, como pueden ser cerrar una ventana, arreglo en cascada, arreglo horizontal, arreglo vertical, minimización de todas las ventanas y arreglo automático. Por último, también tendremos la posibilidad de movernos a través de las ventanas, mediante menús que nos llevarán a la siguiente ventana y a la ventana anterior (disponen de teclas de acceso rápido, tal y como se han explicado anteriormente).

7.4.9 Menú *Ayuda*

Se trata del típico menú de acceso al fichero de ayuda y a la información del programa. El fichero de ayuda, nos brinda de forma similar todo el contenido de este capítulo, en formato de ayuda de Windows (.hlp) de forma que pueda ser integrado dentro de *Medusa* para consultarlo de forma fácil, simplemente pulsando la tecla `F1`. De igual forma también está disponible la ayuda de Oracle, y finalmente el formulario *Acerca de.*

7.4.10 Barras de herramientas

Como se ha comentado anteriormente, básicamente, son accesos directos a las opciones de menú.

Figura 7.6: Aspecto de la barra de herramientas relacionada con la edición totalmente habilitada

Tal y como se muestra en la Figura 7.6, las opciones de la barra de herramientas, son (de izquierda a derecha) 1 - Nuevo, 2 - Abrir (con lista de últimos documentos abiertos), 3 - guardar (también se comporta como la opción *"Guardar como"*, si no se ha guardado el documento), 4 - Exportar a formato HTML (ver sección 7.5.1), 5 - Previsualizar (ver sección 7.5.2), 6 - Deshacer, 7 - Rehacer, 8 - Cortar, 9 - Copiar, 10 - Pegar, 11 - Buscar (ver sección 7.6.1), 12 - Reemplazar (ver sección 7.6.2), 13 - Continuar búsqueda hacia delante (ver sección 7.6.1), 14 - Continuar búsqueda hacia atrás (ver sección 7.6.1), 15 - Configuración (ver sección 7.11), 16 - Ayuda (ver sección 7.4.9).

Figura 7.7: Aspecto de la barra de herramientas relacionada con Oracle

Tal y como se muestra en la Figura 7.7, las opciones de la barra de herramientas, son (de izquierda a derecha) 1 - Conectar (inicia una nueva sesión en la ventana activa. Si desea consultar los protocolos de conexión disponibles, puede ver la sección 7.11.4), 2 - Ejecutar (por defecto, se ejecuta en modo consulta, pero dispone de una lista desplegable para seleccionar el modo de ejecución, siendo capaz de recordar el modo para cada ventana de edición).

7.5 Menú Archivo

En esta sección veremos las opciones más importantes de este menú. Tal y como se detalló en la sección 7.4.1, este menú está compuesto por las opciones más habituales de trabajo con ficheros. Entre estas opciones detallamos a continuación las siguientes:

7.5.1 Exportando un fichero a HTML

Para exportar a formato HTML, así como para exportar a los otros formatos disponibles de *Medusa* (actualmente dos: HTML y texto plano), simplemente tenemos que acceder al menú *Archivo* | *Exportar* y después seleccionar el formato deseado, en nuestro caso HTML.

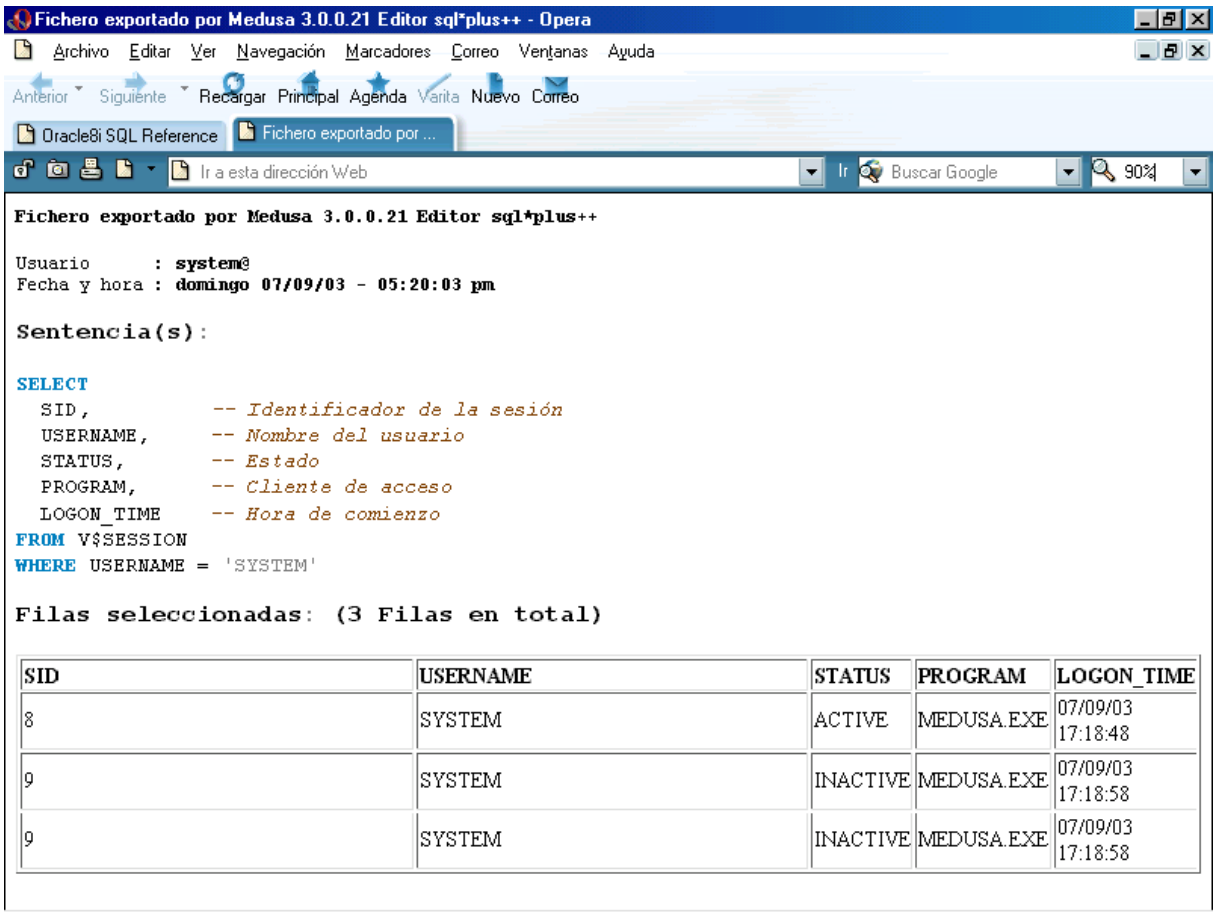

Figura 7.8: Documento HTML generado automáticamente por *Medusa*

El resultado será similar al mostrado en la Figura 7.8. En el documento exportado, se muestra la versión de *Medusa* con que fue exportado el fichero y la fecha de exportación del documento. Sólo cabe destacar que si no se dispone de sesión, no se mostrarán los resultados de la rejilla, ni el usuario conectado, aunque si se dispone de ésta, se mostrarán también el usuario, junto con los resultados obtenidos por la consulta, al igual que el número de filas seleccionadas. El realzado de sintaxis se conserva también y con los mismos colores en el documento generado al exportar.

Pueden ocurrir varios casos distintos:

- El documento es un script. En este caso, el documento se exportará sin los resultados, es decir sólo se exportará el documento, puesto que los resultados de un script pueden ser simplemente confirmaciones de que se ha creado una tabla, se ha insertado una fila, etc.
- El documento es una consulta. En este caso se exportarán o no los resultados dependiendo de si se ha lanzado la consulta o no (tal y como se ha explicado anteriormente). Hay que tener cuidado puesto que si se lanza una consulta, luego se modifica el texto y posteriormente se decide exportar, el programa no será capaz de darse cuenta que los resultados no coinciden con la consulta SQL.

7.5.2 Previsualización de impresión

De forma similar, si queremos previsualizar el documento, basta con hacer click en el menú *Archivo* | *Previsualizar*. El resultado será la previsualización en una nueva ventana del documento seleccionado, teniendo múltiples opciones como por ejemplo navegar a través de las páginas (ir una página hacia delante, ir una página hacia atrás, ir al comienzo de la previsualización e ir al fin de la previsualización), hacer zoom hacia dentro, hacer zoom hacia fuera, configurar la impresora, imprimir, etc.

El documento a la hora de imprimir, está formado por tres partes:

- Cabecera de documento. Muestra una información fija (nombre del programa, junto con la versión, nombre del fichero y usuario conectado), y una parte variable (nombre del usuario del programa) que se puede configurar en el diálogo de configuración (se puede ver más detalladamente en la sección 7.11.3).
- Cuerpo del documento. Es el documento en sí que ha escrito el usuario.
- Pie de página. Simplemente muestra una línea horizontal de división y debajo de ésta, el número de página actual y el número total de páginas (esta sección es opcional y configurable en el diálogo de configuración).

En la Figura 7.9 no se alcanza a ver el pie de página, aunque en ese momento si está activa la opción.

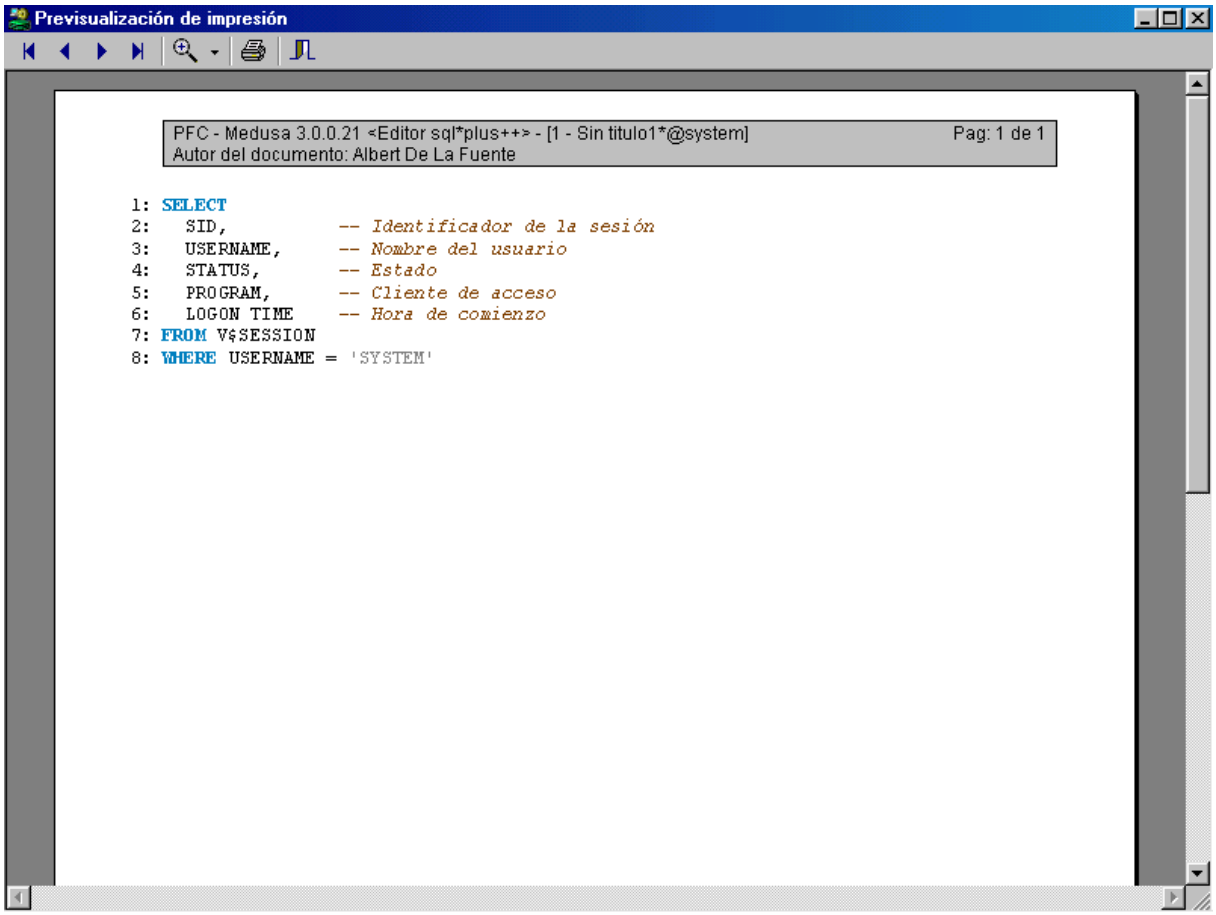

Figura 7.9: Previsualizando un documento

7.6 Menú Edición

En esta sección veremos las opciones más importantes de este menú. Tal y como se detalló en la sección 7.4.2, este menú está compuesto por las opciones más habituales de edición del texto. Entre estas opciones detallamos a continuación las siguientes:

7.6.1 Búsqueda de texto

Este formulario es muy simple, y se emplea para buscar un texto determinado en una ventana de edición. Este formulario tiene un acceso directo, cuya combinación de teclas es CTRL + F. El aspecto de este formulario, se puede ver en la Figura 7.10. Las opciones principales de las que dispone, son: el campo de texto a buscar (*Buscar texto*), un conjunto de opciones, entre los que encontramos:

- *Sensibilidad a mayúsculas y minúsculas*. Se emplea para tener en cuenta si el texto a buscar está en mayúsculas o en minúsculas que por defecto está desactivada.
- *Palabra completa*. Se emplea para buscar sólo la palabra completa e ignorar las palabras que contengan a la palabra que se desee buscar.
- *Búsqueda desde el cursor*. Se emplea para buscar en el documento a partir del cursor.
- *Búsqueda sólo en texto seleccionado*. Se emplea para buscar únicamente en el texto seleccionado.

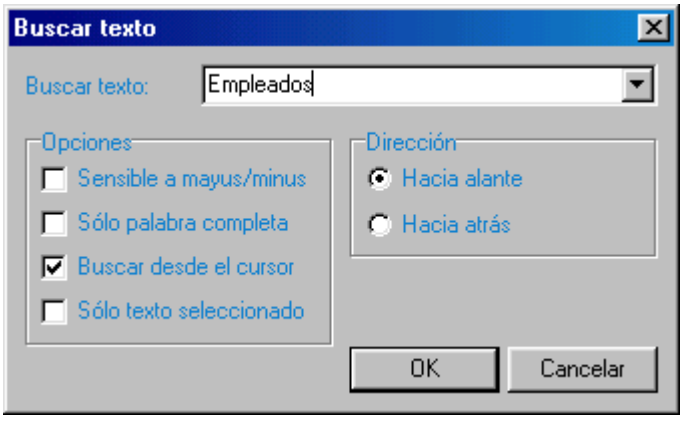

Figura 7.10: Aspecto del formulario de búsqueda de texto

También disponemos de opciones para la dirección de búsqueda, las cuales pueden ser hacia delante (que buscará el texto antes del cursor, de forma cíclica) o hacia atrás (que buscará el texto después del cursor, de forma cíclica).

Se disponen de tres accesos directos en la barra de herramientas. El primero es el de búsqueda, el cual mostrará primero este formulario, el segundo es de búsqueda hacia delante y el tercero es de búsqueda hacia atrás. Estos dos últimos, no muestran el formulario, sino que buscan de forma automática, pero previamente se tiene que haber activado alguna búsqueda. En caso contrario, estarán desactivados.

7.6.2 Reemplazo de texto

Este formulario es muy parecido al formulario de búsqueda anteriormente visto. La única diferencia, es que incluye un campo para escribir el texto por el cual se reemplazará la ocurrencia. Este formulario al igual que el de búsqueda, también dispone de un acceso desde teclado, que es CTRL + R, e igualmente funciona de forma cíclica. También se dispone de esta función en la barra de herramientas. El aspecto de este formulario, se puede observar en la Figura 7.11.

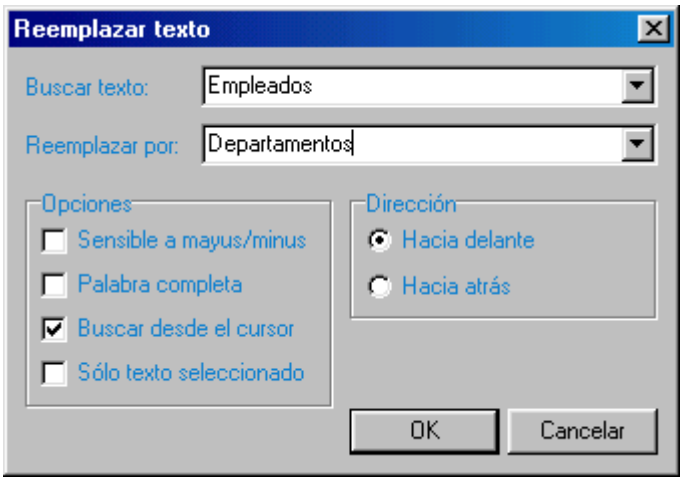

Figura 7.11: Aspecto del formulario de *reemplazado de texto*

7.6.3 Confirmación de reemplazo de texto

El aspecto de este formulario, se puede observar en la Figura 7.12. Se puede observar, que es un formulario muy simple, con únicamente cuatro botones.

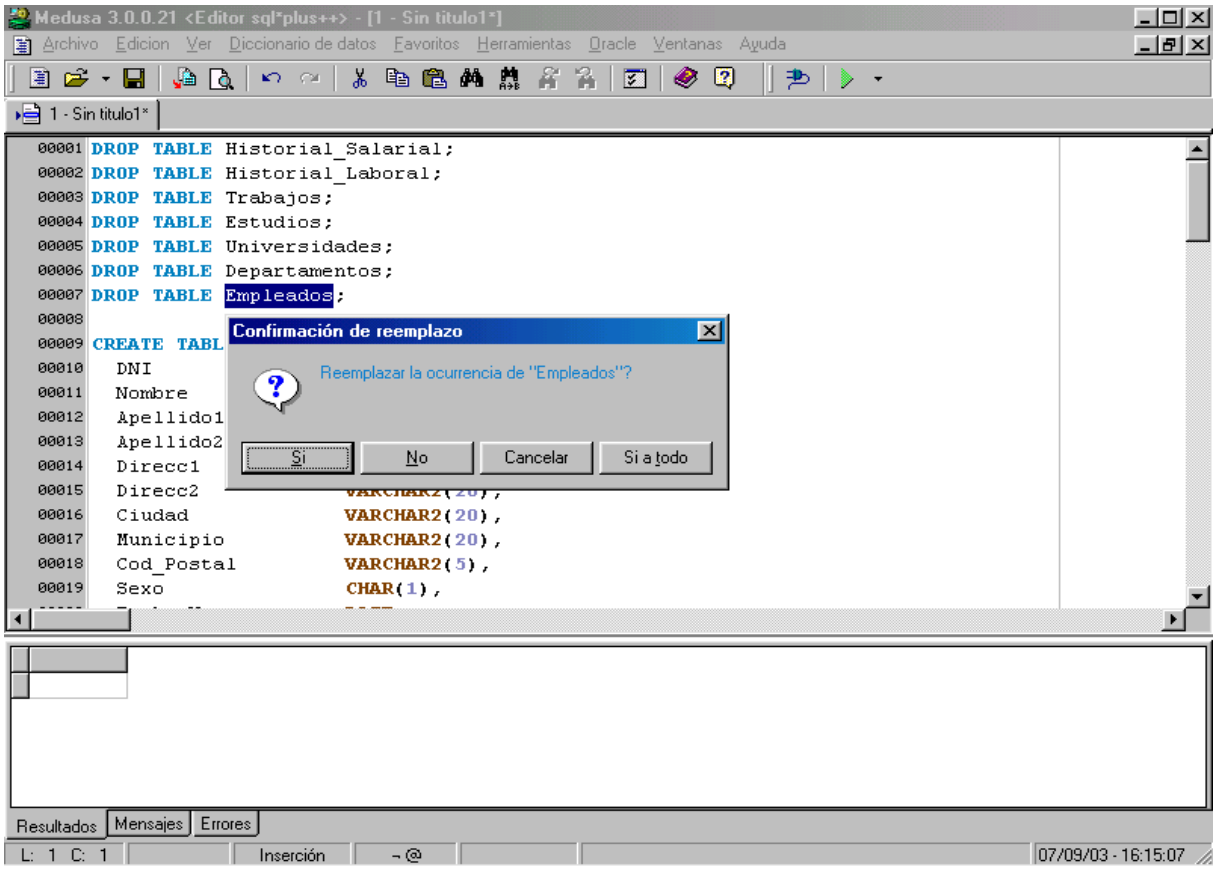

Figura 7.12: Aspecto del formulario de *confirmación de reemplazo*

El primer botón *Si*, simplemente acepta el reemplazo de esa ocurrencia y continúa la búsqueda. El segundo botón *No*, cancela ese reemplazo y continúa la búsqueda. El tercer botón *Cancelar*, aborta la búsqueda. Finalmente, el cuarto botón *Si a todo*, acepta todos los posibles reemplazos que haya en el documento, reemplazándolos de forma automática. Una peculiaridad de este formulario, es que al estar buscando, éste se situará, cerca de la ocurrencia y resaltará seleccionando el texto de la ocurrencia para que se pueda observar fácilmente.

7.6.4 Utilización del buffer configurable de textos

En cada ventana de edición, disponemos de un *buffer* de textos de tamaño configurable circular, que va almacenando las sentencias escritas, cada vez que se ejecuta. La idea de este *buffer* es poder recuperar en todo momento cualquier sentencia que se haya efectuado. *Medusa*, incluye un interfaz que facilita el acceso al *buffer* de la ventana activa, para poder ver la lista de sentencias almacenadas en el *buffer* mientras se trabaja (al cual se accede mediante el menú *Edición* | *Ver estado del buffer*, o bien, mediante su acceso directo mediante teclado, CTRL + B). Se puede ver el texto asociado a cada elemento del *buffer* (ranura, o *slot*), así como información de interés sobre los elementos del *buffer*, un índice de elemento, si ha habido algún error al ejecutar esa(s) sentencia(s), y la fecha y hora de ejecución.

En la Figura 7.13 se observa el aspecto del interfaz de acceso al *buffer*. El tamaño del *buffer* es por defecto de veinte slots, pero se puede configurar por ejemplo a 50 para conseguir una funcionalidad similar al programa SQL*Plus Worksheet.

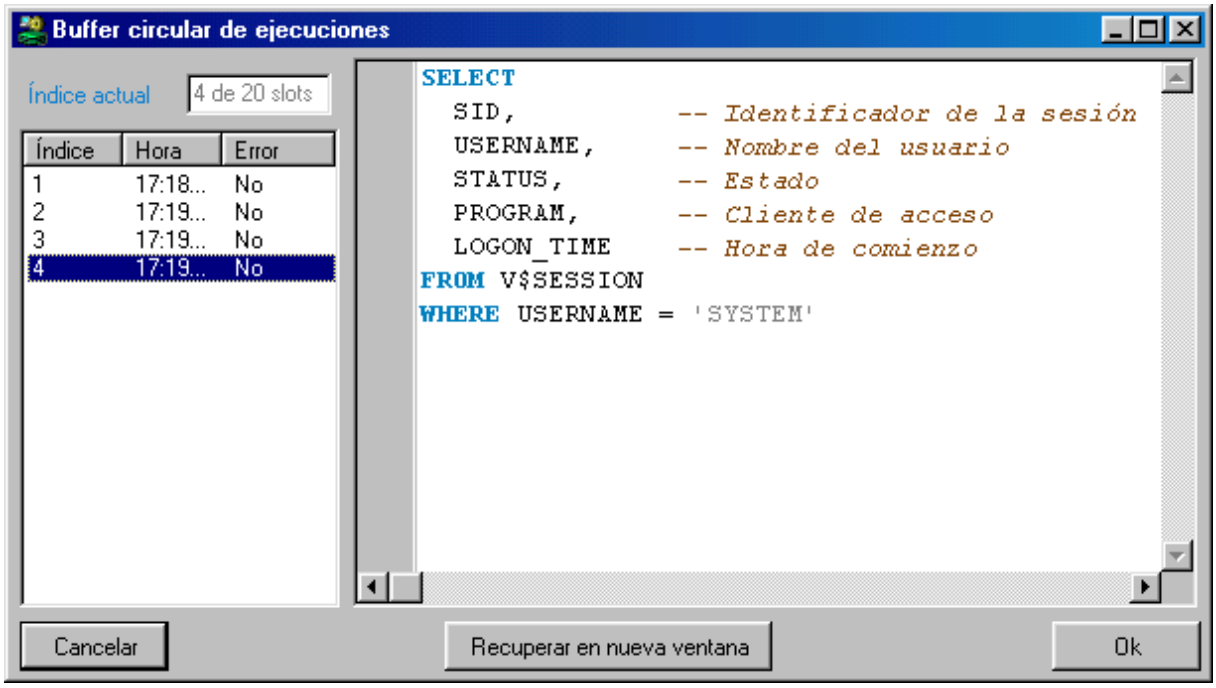

Figura 7.13: Aspecto del *buffer* de textos circular configurable

Se puede recuperar el texto del *buffer* seleccionando un slot o elemento de la lista de la izquierda y pulsando el botón *Ok*, entonces, el texto será copiado a la ventana de trabajo. Es importante resaltar que el texto que se haya podido modificar en la ventana de trabajo, se pierde si no se

ha ejecutado recientemente. Para evitar esto se ha incluido el botón *Recuperar en nueva ventana*, que sirve para recuperar el texto de ese slot o elemento de la lista en una nueva ventana, conservando el texto que hubiese en la ventana activa. En el cuadro superior se muestra el índice mas reciente, que en este caso es el número cuatro, pero no necesariamente es el último de la lista, por la forma circular del buffer. Así por ejemplo, el buffer puede estar lleno y el índice puede indicar el slot número seis, lo cual significa que ya se ha comenzado de nuevo por el principio y actualmente la última sentencia se encuentra en el índice número seis.

7.6.5 Autocompletado por código

Medusa dispone de una opción muy interesante, que es el autocompletado por código. El autocompletado por código es una lista de entradas de autocompletado. Una entrada de autocompletado, consiste en una palabra clave y un texto asociado. Por ejemplo la clave ***putline*** asociada a `DBMS_OUTPUT.PUT_LINE('');`. Podremos tener acceso a esta opción en *Edición* | *Autocompletado por código*, aunque se recomienda utilizar su acceso de teclado (se explica a continuación).

El funcionamiento de esta función es el siguiente: el usuario, por ejemplo, está editando y escribe por ejemplo la palabra clave ***putline*** y a continuación teclea la combinación de teclas CTRL + J. Entonces la palabra clave que acababa de teclear, se sustituye por el texto asociado a esa palabra clave: `DBMS_OUTPUT.PUT_LINE('');`. Esta funcionalidad puede ser útil para incluir trozos de código, llamadas a rutinas, etc. Además, esta lista de autocompletado por código es completamente configurable a través del menú *Herramientas* en *Configuración*. Se puede ver el aspecto del formulario de configuración de autocompletado por código en la sección 7.11.7, donde también se detalla cómo personalizar la lista.

7.6.6 Listas de autocompletado

Las listas de autocompletado aunque parezca que tiene relación con la sección anterior, no tienen nada que ver. Esta lista se muestra de forma visual al usuario cuando ocurre un suceso de disparo.

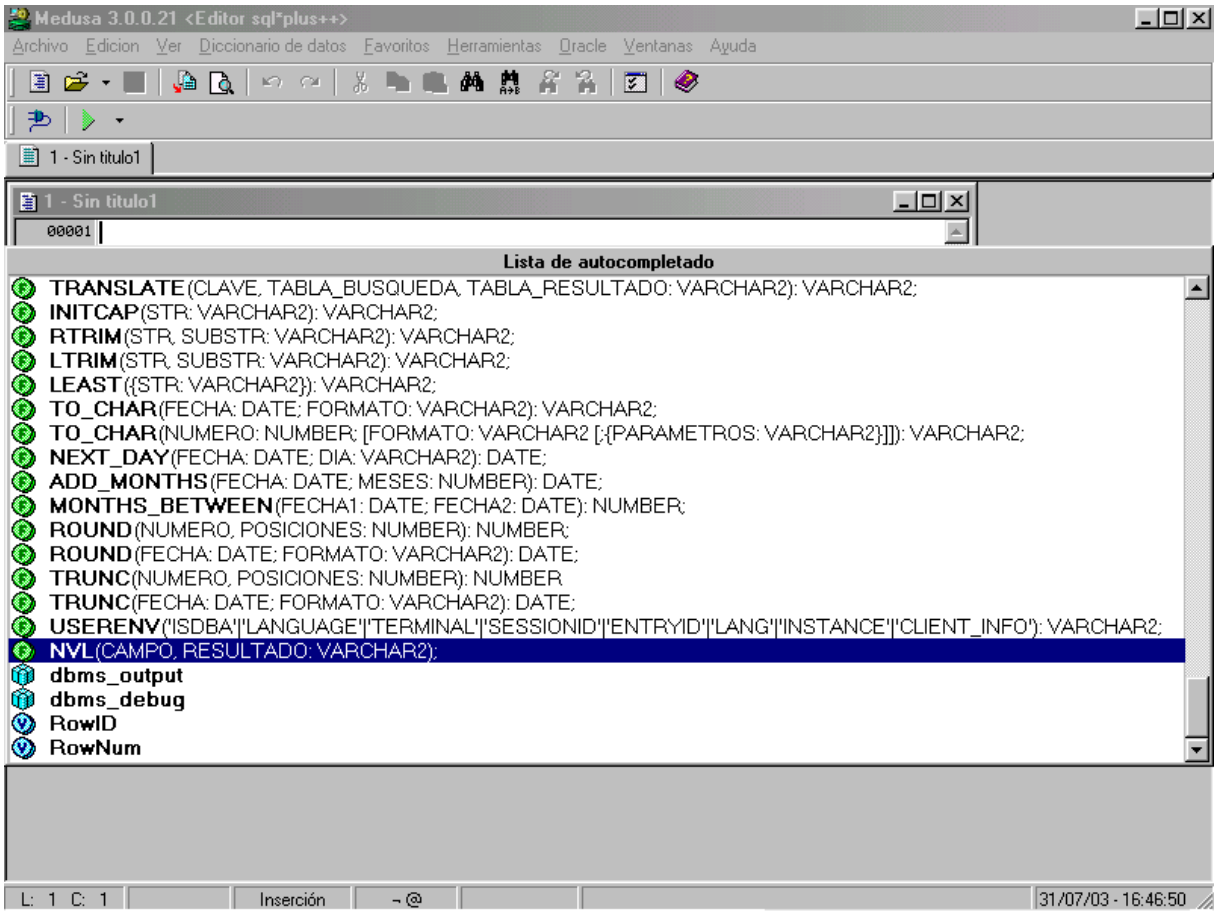

Figura 7.14: Lista de autocompletado

Hay tres sucesos de disparo:

- Mediante el menú *Edición* | *Listas de autocompletado* (aunque se recomienda el acceso a esta opción mediante los métodos explicados a continuación).
- Teclear un punto (por ejemplo, al acceder a un elemento de un paquete).
- Teclear la combinación de teclas CTRL + Espacio.

Las listas de autocompletado están formadas por nombres de rutinas (funciones o procedimientos) junto con sus parámetros, nombres de paquetes o nombres de variables. En el caso de los paquetes, también tiene que existir otra lista de autocompletado llamada como el paquete en cuestión.

Una vez que ha aparecido la lista de autocompletado, el usuario puede seleccionar de la lista la rutina a la que desea invocar o el paquete al cual desea acceder y entonces se agrega de forma automática el texto necesario para llevar a cabo la acción. Si es un paquete, se añadirá el nombre

del paquete, en cambio si es una rutina lo que se desea invocar, se agregará la llamada a la rutina. Aunque se muestren los parámetros de la rutina en la lista de autocompletado, éstos no se añadirán en el editor. En la Figura 7.14 se observa el aspecto de la lista de autocompletado, donde el elemento seleccionado es la rutina `NVL`.

Los ficheros que almacenan los valores de estas listas, tienen extensión *"lat"*. Existe un fichero principal, que representa el índice de la lista que es el fichero *"Oracle.lat"*. Dentro de este fichero, se almacenan según un formato determinado todas las entradas de la lista, es decir, procedimientos, funciones, variables y paquetes. En el caso de los paquetes, debe existir también un fichero llamado con el nombre del paquete y extensión *"lat"*, de forma que cuando se invoque la lista en el caso de un paquete determinado, se abra automáticamente su fichero correspondiente. No es recomendable editar estos ficheros a menos que se el usuario tenga conocimiento de las acciones que está llevando a cabo. Si el usuario desea incluir más opciones, como por ejemplo, una expresión que utilice de forma habitual, puede hacerlo mediante la opción de *autocompletado por código*.

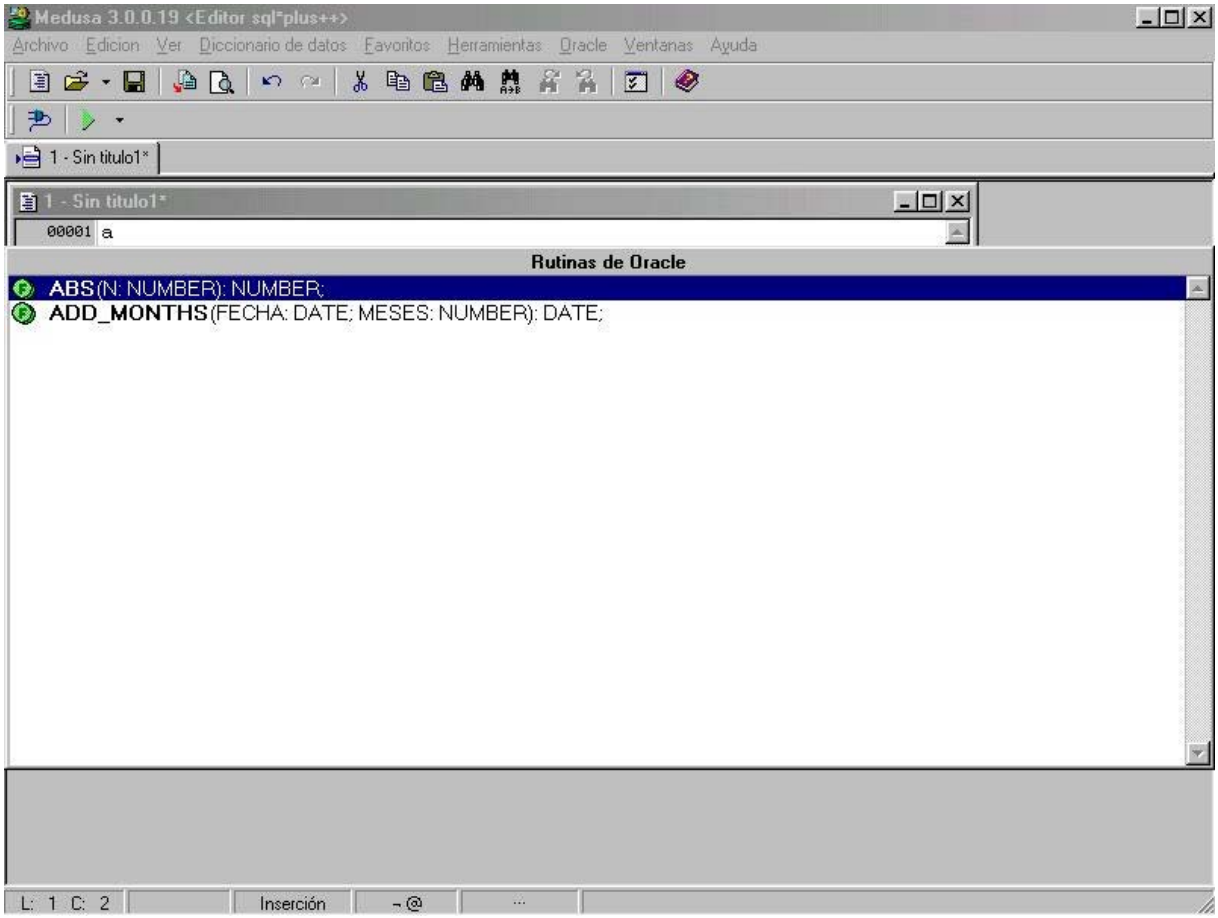

Figura 7.15: Aspecto de la lista de autocompletado filtrada por la letra *"a"*

Las listas de autocompletado, además de mostrar un índice con las rutinas y paquetes disponibles, también filtran, es decir, si el usuario ha tecleado algo, sólo se mostrarán los elementos de la lista que empiecen por el texto tecleado por el usuario. Ese es el caso de la Figura 7.15, donde se el usuario ha tecleado la letra *"a"* y se ha filtrado la lista de autocompletado a los elementos que comiencen por la letra *"a"*.

En la Tabla 7.1 se puede ver la representación de los tipos de entradas, junto con su significado que se pueden almacenar en las listas de autocompletado.

Icono asociado	Significado
	Representación de un procedimiento
	Representación de una función
	Representación de una variable (por ejemplo la variable "*RowID*" o "*RowNum*")
	Representación de un paquete (por ejemplo `dbms_output` o `dbms_debug`)

Tabla 7.1: Representación de los iconos de las listas de autocompletado

7.7 Menú Consultas al diccionario de datos

Medusa dispone de un extenso menú con algunas de las consultas al diccionario de datos más importantes. El menú se organiza básicamente en cuatro secciones:

- Sección `USER_`. En esta sección encontramos las consultas más usuales a las vistas cuyo prefijo es `USER_`, es decir, las consultas que hacen referencia a los objetos del usuario.
- Sección `ALL_`. En esta sección encontramos las consultas más usuales a las vistas cuyo prefijo es `ALL_`, es decir, las consultas que hacen referencia a todos los objetos de la base de datos accesibles por el usuario activo.

- Sección `DBA_`. En esta sección encontraremos las consultas más usuales a las vistas cuyo prefijo es `DBA_`, es decir, las consultas que hacen referencia a todos los objetos de la base de datos.
- Sección `V$_`. En esta sección encontramos las consultas más usuales a las vistas cuyo prefijo es `V$_`, es decir, las consultas que hacen referencia a los objetos que se actualizan dinámicamente y que gestiona Oracle.

Entre las consultas que podemos encontrar en la sección `USER_`, `ALL_` y `DBA_`, tenemos las siguientes: tablas, disparadores, vistas, usuarios, objetos, trabajos, sinónimos, índices, privilegios sobre tablas cedidos y recibidos, privilegios sobre columnas cedidos y recibidos, restricciones, restricciones sobre columnas, comentarios sobre tablas y finalmente comentarios sobre columnas.

Entre las consultas que podemos encontrar en la sección `V$_`, tenemos las siguientes: base de datos, ficheros de datos, tablas y vistas, sesiones, procesos y finalmente ficheros de control.

Las consultas al diccionario de datos que se encuentran en *Medusa*, son consultas con comentarios, donde se comenta brevemente la utilidad de esa vista y el significado de cada columna, de forma similar a como se muestra en la Figura 7.13, salvo que en ese caso se eliminaron algunos comentarios, algunas columnas y se ha añadido una cláusula `WHERE`.

7.8 Menú Oracle

En esta sección veremos las opciones más importantes de este menú. Tal y como se detalló en la sección 7.4.7, este menú está compuesto por las opciones más habituales para la manipulación de objetos de la base de datos, junto con algunas otras opciones. Entre estas opciones detallamos a continuación las siguientes:

7.8.1 Información de la sesión

Podremos tener acceso a esta opción haciendo click en el menú *Oracle* y después haciendo click en *Información de la sesión*. Se nos mostrará un formulario con diversa información, no sólo de la sesión, sino además de la versión de Oracle instalada, información del usuario (roles y limitaciones) y privilegios de la sesión. En la Figura 7.16 se muestra toda la información que recopila el formulario. Sólo se puede tener acceso a esta opción si está conectado.

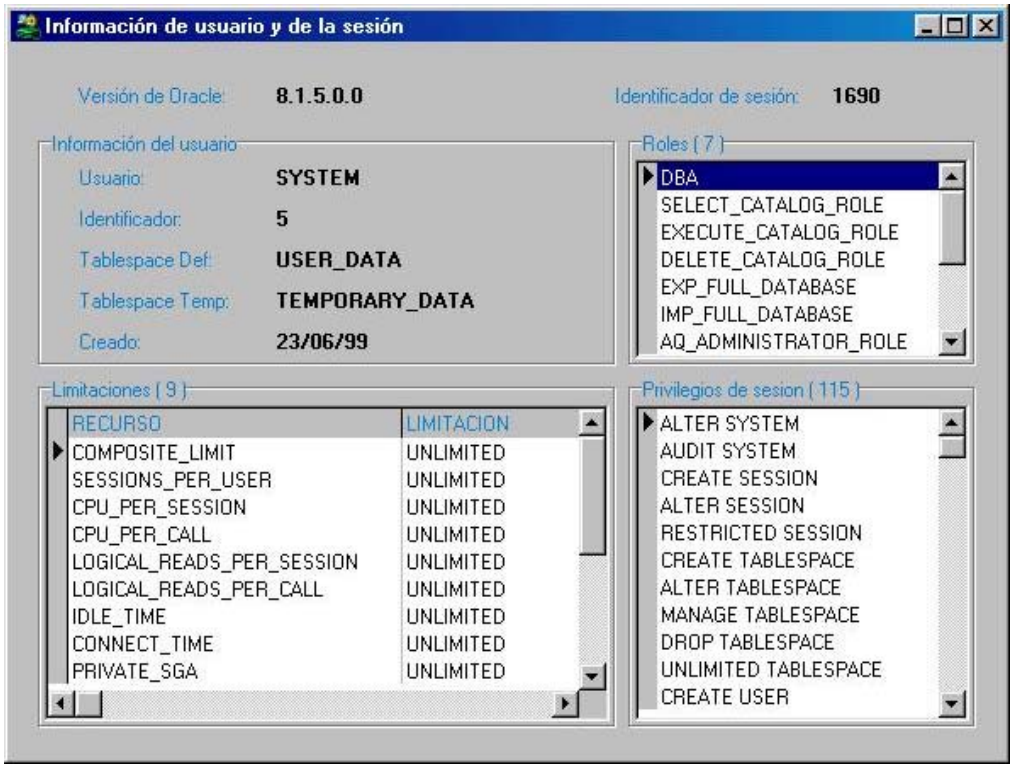

Figura 7.16: Información disponible de la sesión

Entre la información que nos muestra, podemos ver los roles, limitaciones y privilegios de la sesión organizados en forma de tablas. En estas tablas, en el borde superior, se muestra un número entre paréntesis que indica la cantidad de elementos que contiene la tabla, así por ejemplo en la Figura 7.16, se puede ver como en la tabla de roles, encontramos siete roles que posee el usuario `SYSTEM`.

7.8.2 Estructura de la Base de Datos

Una opción muy interesante que está disponible en *Medusa*, es la posibilidad de obtener un informe de los atributos y relaciones de un conjunto de tablas. En la Figura 7.17, se puede observar el aspecto de este formulario. Es un formulario muy simple, donde en la parte superior encontraremos una lista de todas las tablas disponibles en la vista del diccionario de datos `ALL_TABLES`, que se cargarán de forma automática al seleccionar la opción *Estructura de la BD* del menú *Oracle* en el formulario principal. El mecanismo de funcionamiento de este formulario, es que primero se selecciona la tabla deseada de la lista y después se pulsa el botón *Agregar*, con lo cual se agregará la tabla a la lista de tablas de las que se desea obtener la información.

Tal y como se observa en el formulario, existe la casilla *"Incluir referencias"*, que se puede marcar opcionalmente. La función de esta casilla es incluir las tablas automáticamente a las que alguna de las tablas del conjunto de partida haga referencia, de forma que se pueda ampliar el informe si se desconocen las referencias.

En la Figura 7.18 se puede observar el resultado del informe generado, con todas las columnas de las tablas y la información de sus referencias (lo cual se representa con una flecha. Igualmente, se agrega la tabla referenciada y atributo(s) referenciado(s) por la tabla). El asterisco que está a la izquierda del (los) atributo(s), significa que forman parte de la clave primaria.

Figura 7.17: Aspecto del formulario de Estructura de la base de datos

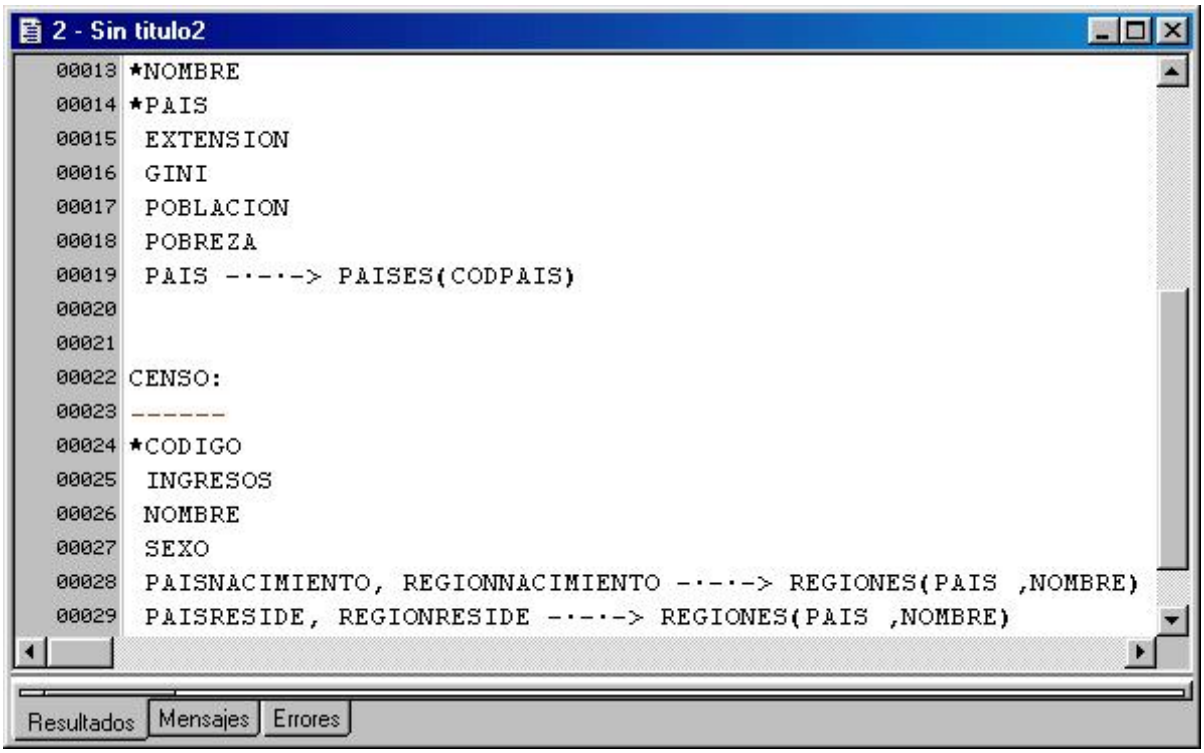

Figura 7.18: Informe generado automáticamente

7.8.3 Construcción de objetos

En esta sección veremos en detalle los interfaces gráficos diseñados para facilitar la tarea de construcción de diversos tipos de objetos que se han incluido en *Medusa*. El aspecto de los formularios es similar, todos disponen de un botón *Cancelar* y un botón *Aceptar*.

7.8.3.1 Construcción de procedimientos y funciones

Esta opción incluida en *Medusa*, facilita la creación de procedimientos y funciones de forma visual. Sobre todo puede ayudar a los usuarios menos experimentados. Podremos acceder a esta opción haciendo click en el menú *Oracle* y posteriormente en *Construir procedimiento / función*, o bien mediante la combinación ALT + R. En la Figura 7.19 se puede ver el aspecto de este formulario. En la Figura 7.20 se puede ver el resultado de la rutina generada automáticamente.

El funcionamiento es muy sencillo, primero rellenamos el nombre y seleccionamos el tipo de rutina (procedimiento o función), en caso de ser una función, habrá que indicar el tipo devuelto por ésta. Opcionalmente podemos agregar un comentario aclaratorio sobre la rutina. Ahora sólo queda añadir los parámetros, para lo cual pulsaremos primero el botón *Añadir* y nos saldrá en la rejilla de la izquierda una línea en blanco que se selecciona automáticamente y después agregaremos los datos referentes a ese parámetro como son su nombre, si es de entrada o de salida y su tipo. En la lista de tipos, también se puede escribir, en caso de que el tipo de ese parámetro sea un tipo creado por el usuario.

Si se desea eliminar un parámetro, simplemente bastará con seleccionar el parámetro deseado de la lista y pulsar el botón *Eliminar*.

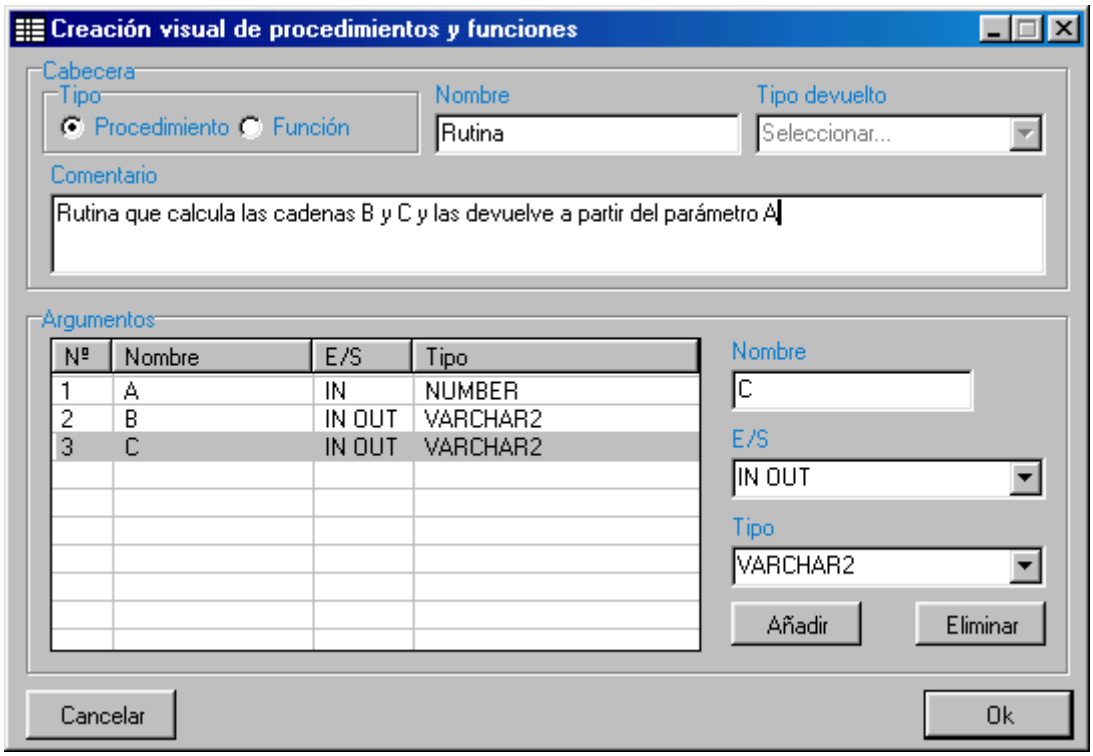

Figura 7.19: Creación visual de procedimientos y funciones

El control de errores de este formulario al igual que del resto de la aplicación es simplemente informar al usuario en caso de producirse un error y la solución en el momento en que el usuario pulse el botón de *Ok*, para que pueda arreglarlo. Así por ejemplo si se omite el nombre de la rutina, el programa seguirá funcionando sin problemas hasta que el usuario pulse el botón *Ok*, momento en el cual *Medusa* notificará al usuario que se le ha olvidado rellenar el campo nombre

de la rutina. Si el usuario desea cancelar la operación sólo tendrá que pulsar el botón *Cancelar* o cerrar el formulario y se abortará la construcción de la rutina.

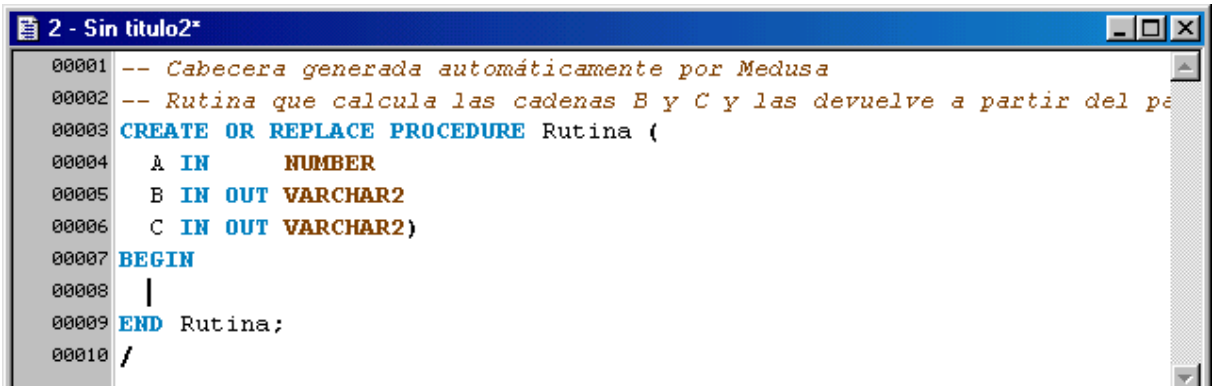

Figura 7.20: Rutina creada utilizando el formulario de creación visual de rutinas con los datos de la Figura 7.19

7.8.3.2 Construcción de disparadores

Medusa también dispone de un formulario para crear de forma visual disparadores (*triggers*). Podremos acceder a esta opción, haciendo click en el menú *Oracle* y posteriormente en *Construir disparador*, o bien, mediante la combinación ALT + D.

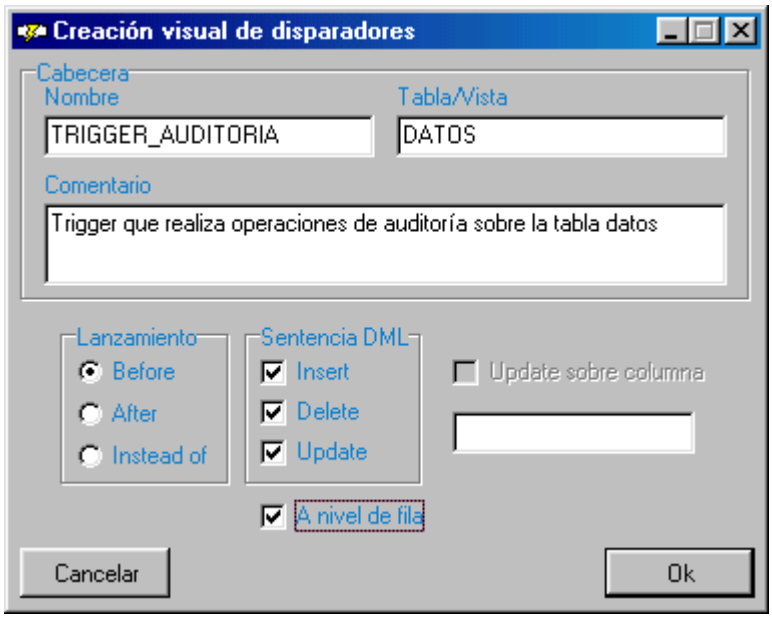

Figura 7.21: Aspecto del formulario de creación de disparadores

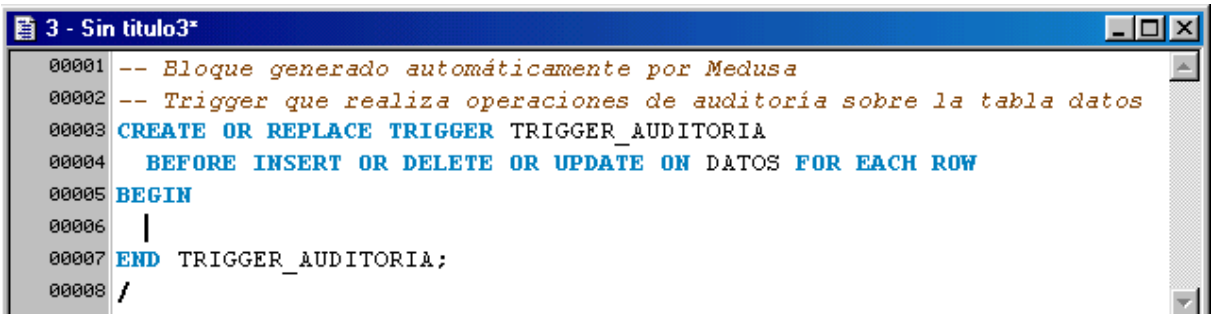

Figura 7.22: Código de un disparador generado con los datos de la Figura 7.21

En la Figura 7.21 se muestra el aspecto de este formulario. Como se puede observar es un formulario sencillo, donde simplemente se deben rellenar los datos del disparador (nombre, tabla, evento de lanzamiento, sentencia y comentario). En la Figura 7.22, se muestra el código generado por la aplicación.

Igualmente que en el caso anterior, si el usuario olvidase de complementar correctamente los datos, el programa le advertirá que no se puede seguir hasta que no rellene los datos faltantes.

7.8.3.3 Construcción de paquetes

Medusa también incorpora un formulario para la construcción de forma visual de paquetes (definición y cuerpo). Accederemos a esta opción en el menú *Oracle* | *Construir paquete*, o bien mediante la combinación de teclas ALT + P. La definición del paquete, junto con el cuerpo del paquete, se generan en una nueva ventana de edición. Ambas partes estarán separadas por la barra de compilación *"/"* de bloques de código de Oracle.

En la Figura 7.23, se puede observar el aspecto visual de este formulario. El campo propósito, es opcional, sólo se incluye como una breve descripción del paquete.

El código generado, incluye el usuario del programa, la fecha de creación y el comentario (no se alcanza a ver en la Figura 7.24). La estructura del código es de un código completamente general, y sirve más que nada para recordar la sintaxis de los paquetes en PL/SQL.

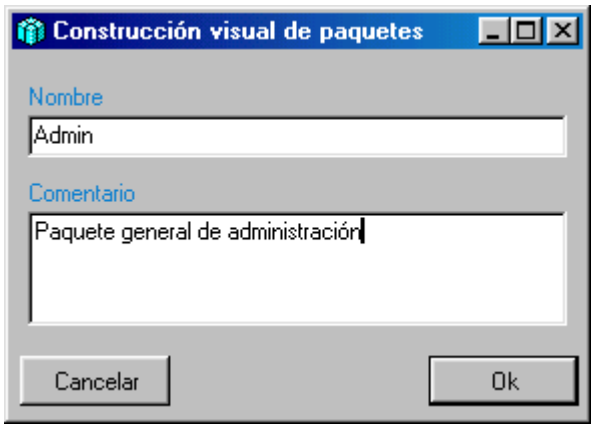

Figura 7.23: Aspecto del formulario de construcción visual de paquetes

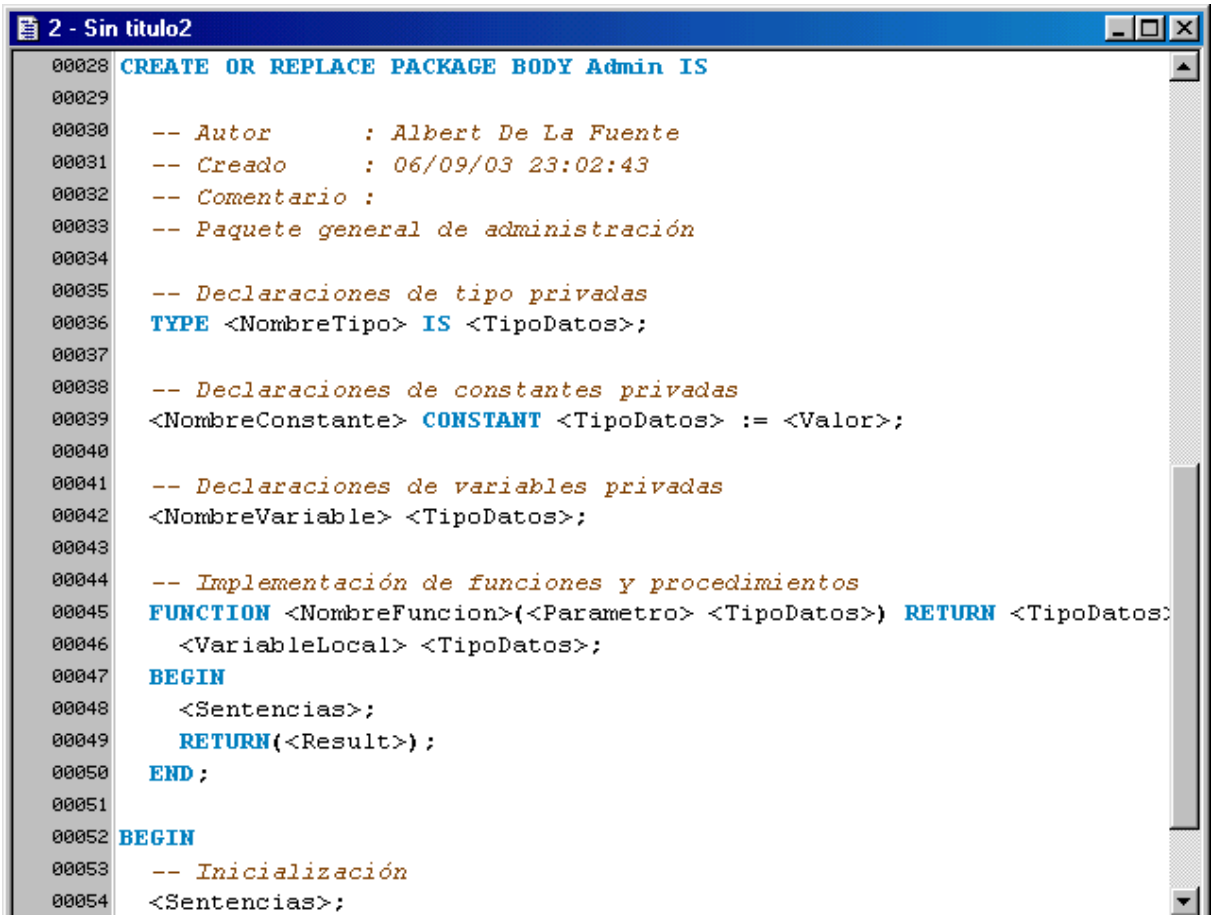

Figura 7.24: Código generado por la opción de generación de paquetes con los datos de la Figura 7.23

Hay que matizar que el código generado del paquete es demasiado grande para poder ser visualizado en una sola figura. Concretamente salen cincuenta y nueve líneas de código, de las cuales sólo podemos observar unas cuantas en la Figura 7.24.

El control de errores al igual que en el resto de opciones se produce cuando el usuario pulse el botón *Ok*. La aplicación comprueba si todos los campos están cumplimentados de forma correcta y en caso de no ser así, se muestra un mensaje con el error pertinente.

7.8.3.4 Construcción de bloques Java

De forma similar a los paquetes, *Medusa* también ofrece la posibilidad de generar el código básico (esqueleto de código) de un bloque Java que se puede posteriormente almacenar en la base de datos. Esta opción la encontraremos en *Oracle* | *Construir bloque Java*. El aspecto de este formulario, lo podemos observar en la Figura 7.25, que como se puede observar, basta con indicar el nombre del bloque, el nombre de la clase y opcionalmente un comentario. Tal y como se puede observar en la Figura 7.26, *Medusa* conmuta automáticamente la opción del realzado sintáctico a modo Java.

El control de errores es similar a todos los casos anteriores, se informará mediante un mensaje al usuario en caso de ocurrir alguno y se esperará hasta que el usuario lo solvente.

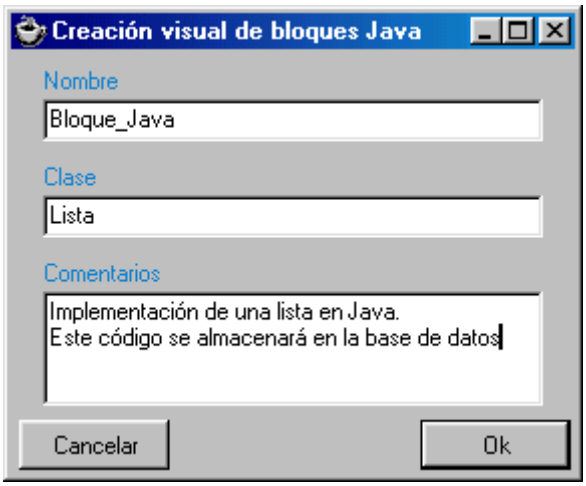

Figura 7.25: Aspecto del formulario de generación de bloques Java

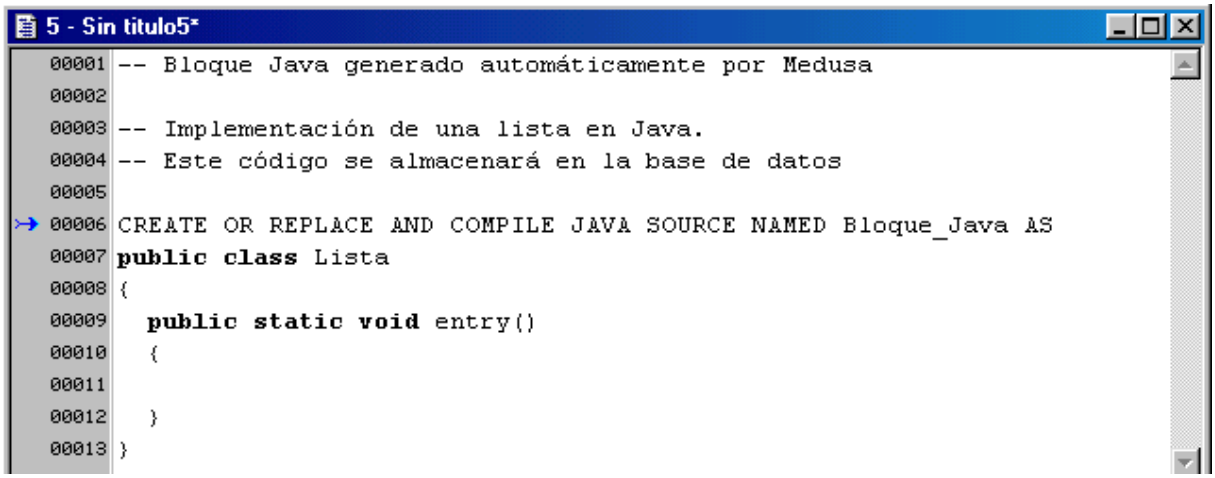

Figura 7.26: Código del bloque de Java generado por la opción de generación de bloques Java, con los datos de la Figura 7.25

7.8.3.5 Construcción de secuencias

De forma similar a los casos anteriores, *Medusa* también incorpora un interfaz para facilitar la construcción de secuencias de una forma visual, lo cual puede ser bastante útil sobre todo para los usuarios menos experimentados. Se puede tener acceso a esta opción, haciendo click en el menú *Oracle* y posteriormente haciendo click en el menú *Construir Secuencia*. Igual que en todos los casos anteriores, también se dispone de una combinación de acceso desde teclado, que es ALT + C.

En la Figura 7.27, se puede observar el aspecto del formulario de generación de secuencias. El campo del propietario es opcional (por defecto está vacío), aunque se puede incluir en caso de desear crear la secuencia en el esquema de otro usuario. El campo *Nombre*, es el nombre que recibirá la secuencia generada. El campo *Valor mínimo*, es el valor mínimo de la secuencia. Este valor puede ser –1, en cuyo caso se tomará la constante de Oracle `NOMINVALUE` (especifica el valor 1 y -10^{26} para secuencias ascendentes y descendentes respectivamente). El campo *valor máximo*, es el valor máximo de la secuencia, el cual, al igual que en el caso anterior, puede ser –1, en cuyo caso se tomará la constante de Oracle NOMAXVALUE (especifica el valor –1 y 10^{27} para secuencias ascendentes y descendentes respectivamente). El valor de *Comenzar desde*, es el valor de inicio de la secuencia. El valor de *Incremento*, es el incremento entre los elementos. El tamaño de la *Cache* es opcional. Si posee un valor de -1, no se incluirá en el código generado de la secuencia.

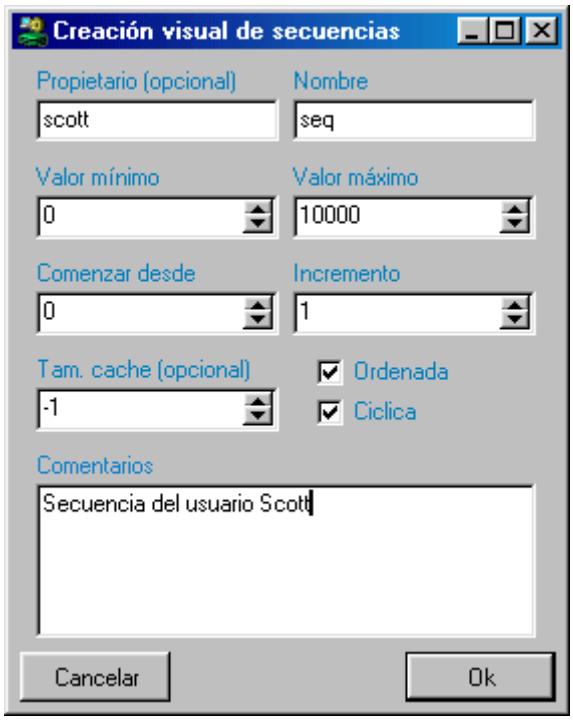

Figura 7.27: Aspecto del formulario de generación de secuencias

En la Figura 7.28, se puede ver el código generado a partir del formulario mostrado en la Figura 7.27, donde se crea una secuencia llamada *seq*, en el esquema de *scott*.

El control de errores al igual que en el resto de la aplicación consiste en mostrar un mensaje de error en el caso de que el usuario olvide algún campo o cualquier otro, como por ejemplo, un valor fuera de rango, etc.

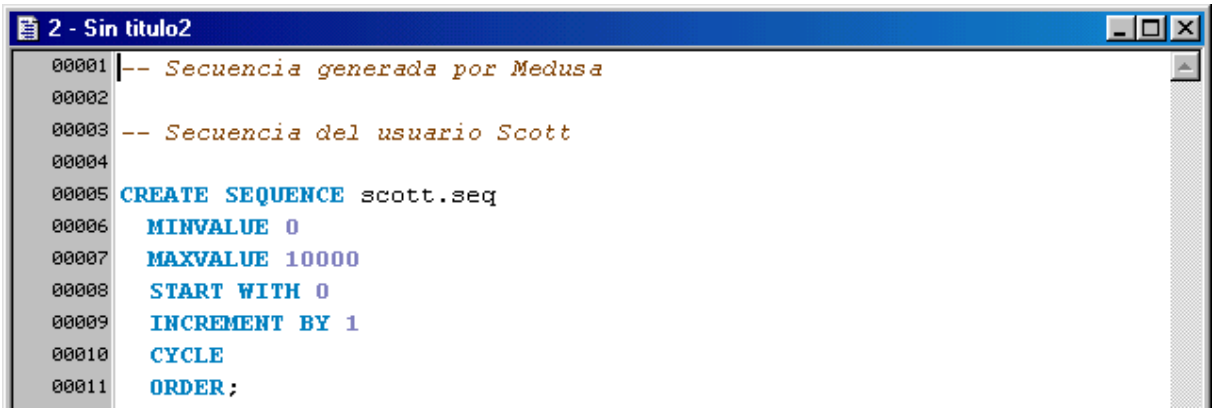

Figura 7.28: Código generado por la opción de generación de secuencias con los datos de la Figura 7.27

7.8.3.6 Construcción de sinónimos

Medusa incluye también una opción para la construcción de sinónimos de forma fácil. Se puede tener acceso a esta opción haciendo click en el menú *Oracle* y posteriormente en *Construir sinónimos* o bien mediante ALT + O. Tal y como se observa en la Figura 7.29, el formulario de generación de sinónimos se compone de varios campos, algunos de los cuales no son obligatorios. En el caso campo del propietario, se puede utilizar para generar la secuencia en el esquema de otro usuario. Si por el contrario, se marca la casilla de verificación, el sinónimo será público. En caso de estar marcada, aparecerá el texto *"PUBLIC"*, como propietario del sinónimo en el formulario. El nombre, es el nombre que recibirá el sinónimo. El propietario del objeto, es opcional, y es el propietario del objeto al que se le creará el sinónimo apuntando a éste. El objeto, es el nombre del objeto en sí. Finalmente, Hay otro campo también interesante que representa la posibilidad de la creación de sinónimos en objetos sobre bases de datos remotas, que es el denominado *"Enlace DB"*.

En la Figura 7.30 se puede observar el sinónimo *público* creado por el usuario, sobre el objeto *emp* (tabla en nuestro caso), del usuario *scott*.

Por último sólo queda comentar, que al igual que en el resto de formularios y opciones, se puede salir utilizando la tecla *escape* (ESC).

El control de errores sigue la misma línea que los casos anteriores. Generalmente los principales errores que se suelen producir son la falta de algún dato.

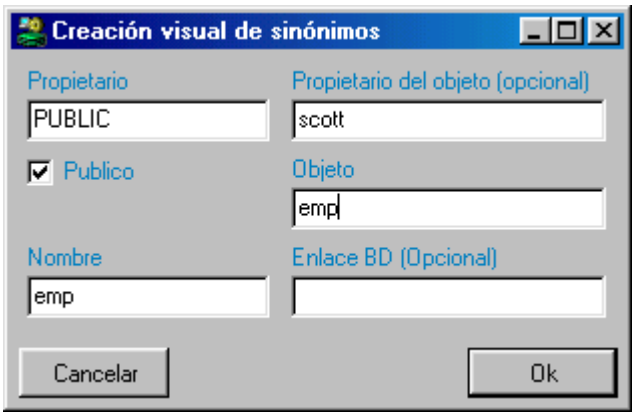

Figura 7.29: Aspecto visual del formulario de generación de sinónimos

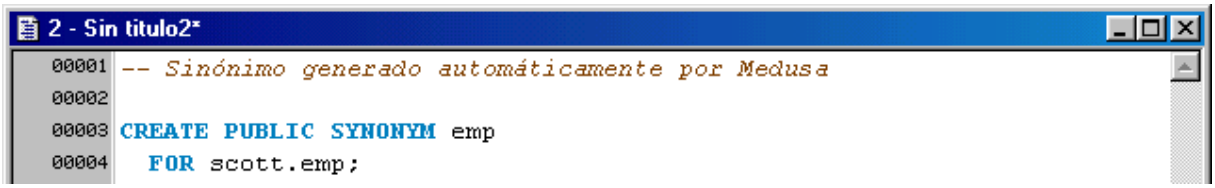

Figura 7.30: Código generado del sinónimo *público* sobre la tabla *emp* perteneciente al esquema del usuario *scott*, según los datos de la Figura 7.29

7.8.3.7 Construcción de vistas

De forma similar a los casos anteriores, *Medusa* también incorpora un interfaz para facilitar la construcción de vistas y vistas materializadas de una forma visual, lo cual puede ser bastante útil. Se puede tener acceso a esta opción haciendo click en el menú *Oracle* y posteriormente en *Construir vista*, o bien, mediante su acceso de teclado correspondiente ALT + V.

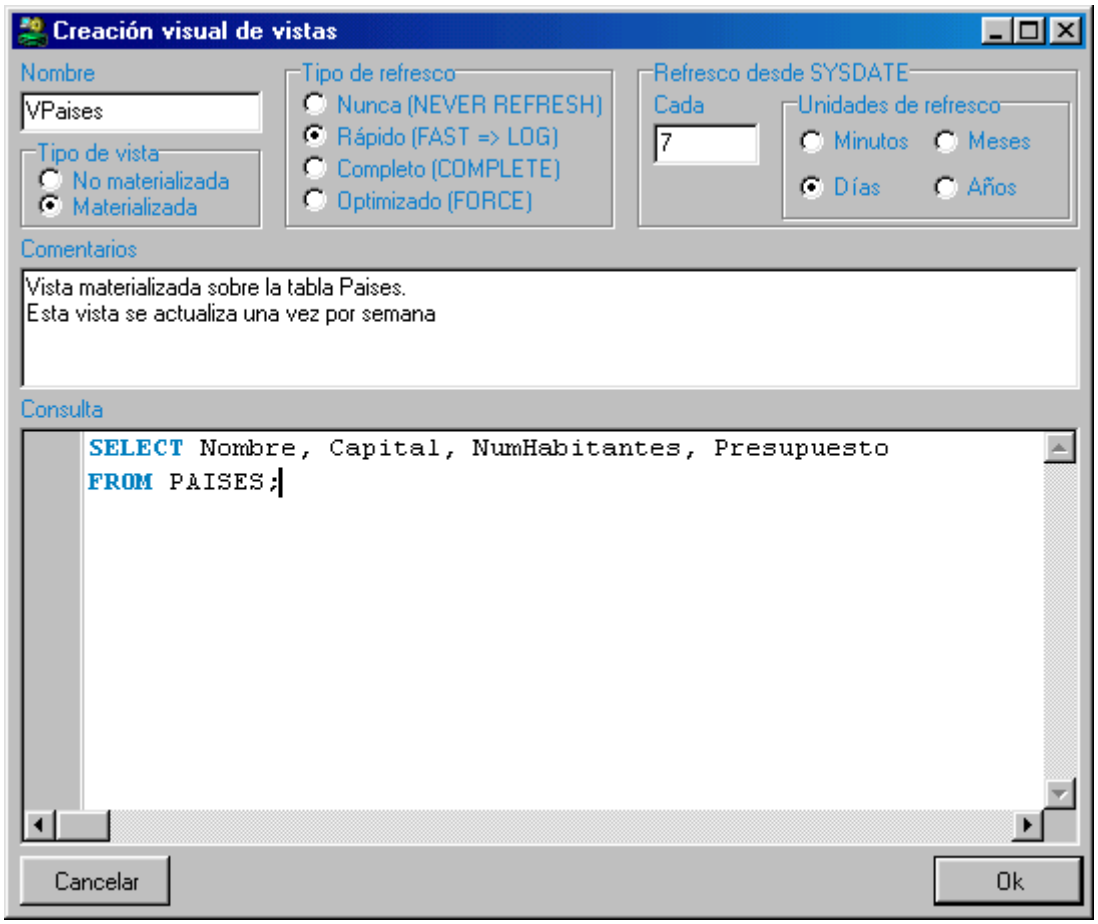

Figura 7.31: Aspecto visual del formulario de creación de vistas

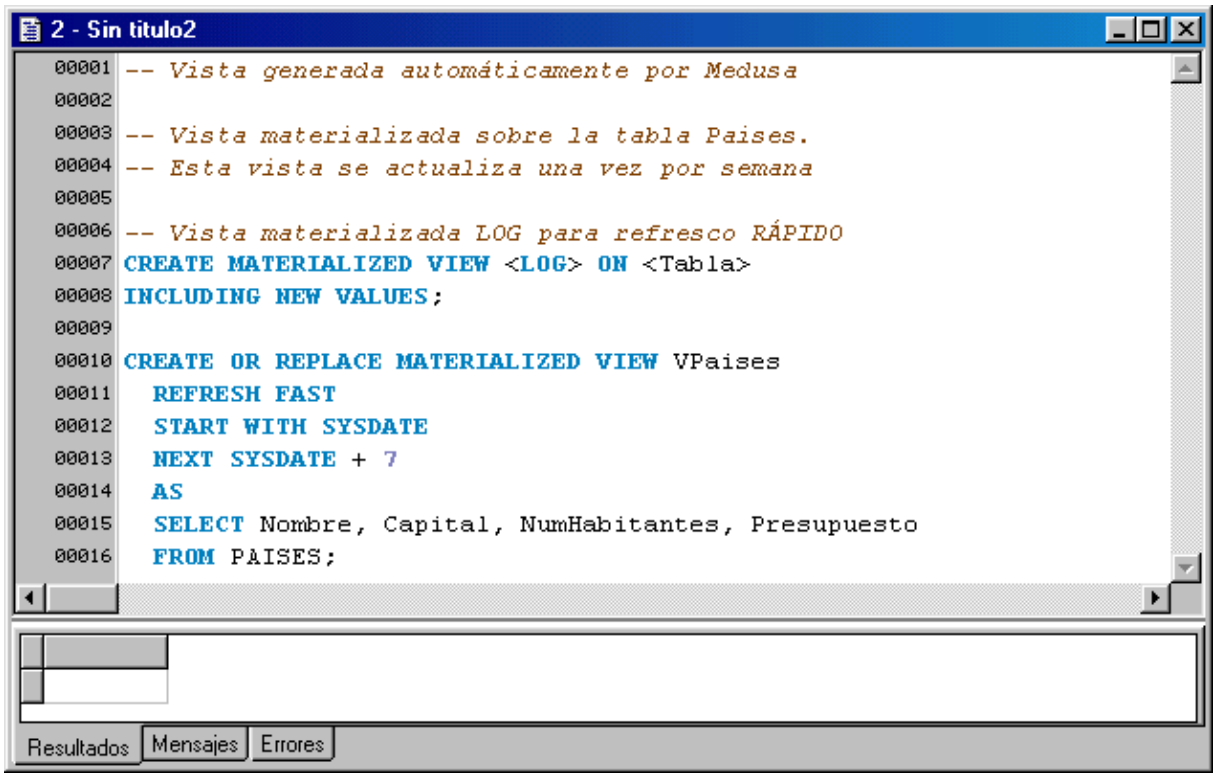

Figura 7.32: Código generado por la opción de construcción de vistas, con los datos de la Figura 7.31

En la Figura 7.31, se puede observar el aspecto del formulario de generación de vistas. El campo nombre, es el que contendrá el nombre de la vista o vista materializada que deseemos crear. Tendremos la opción de seleccionar el tipo de vista que deseemos crear, entre los cuales disponemos de vistas normales (no materializadas) y vistas materializadas. Según se seleccione uno u otro, veremos que las opciones estarán inactivas o activas respectivamente. En caso de querer crear una vista materializada, podremos seleccionar el tipo de refresco, entre los cuatro tipos disponibles, encontraremos las opciones: no refrescar (la vista nunca se refrescará), rápido (requiere de una tabla especial que contiene los cambios hechos sobre las tablas base, también llamado LOG), completo (se rehace la consulta cada vez) y finalmente optimizado (se intentará ejecutar un refresco rápido cuando se pueda, y en caso de no poder, se ejecutará un refresco completo).

Por último, aunque no menos importante, encontraremos el intervalo de refresco, en el campo denominado *"cada"*. Ahí ingresaremos el intervalo de refresco. Justo a su derecha, tendremos las posibles unidades de refresco. Como unidades de refresco tenemos: minutos (la vista se actualiza en el intervalo indicado en minutos), días (la vista se actualiza en el intervalo indicado en días), meses (la vista se actualiza en el intervalo indicado en meses, hay que tener en cuenta que es

una aproximación, puesto que no todos los meses tienen los mismos días. Se toma como base treinta días) y finalmente, también disponemos de años (la vista se actualiza en el intervalo indicado en años, e igualmente que en el caso de los meses, es una aproximación, puesto que se toman doce meses de treinta días cada uno).

En la Figura 7.32, podemos observar como habiendo escogido la opción de refresco rápido, se ha generado automáticamente, el esqueleto de la vista materializada LOG. Puesto que para una actualización rápida, es necesario esta vista materializada, se decidió proporcionar al usuario la posibilidad de que al efectuar esta elección, se cree de una vez su esqueleto.

El usuario únicamente debe indicar el nombre de la vista materializada LOG, junto con la tabla base a la que refleja, los cuales no se han podido incluir puesto que dependen de la consulta que el usuario desea asociar a la vista materializada.

7.8.3.8 Construcción de *tablespaces*

Medusa ofrece también la posibilidad de crear *tablespaces* (espacios de tablas, donde se almacenan los datos de forma lógica que físicamente se almacenan en ficheros de datos o *datafiles*). Se puede tener acceso a esta opción en el menú *Oracle* y después, haciendo click en *Construir tablespace*. Se ofrece un acceso directo a esta opción desde la aplicación, el cual es ALT + T. Para ello ofrece al usuario un formulario compuesto por los principales tipos de *tablespaces* y algunas de sus opciones habituales, las cuales podremos configurar fácilmente. El aspecto del formulario de creación de *tablespaces* se puede ver en la Figura 7.33.

El formulario está compuesto de tres zonas básicas. La primera zona (arriba a la izquierda), nos ofrece la posibilidad de escoger el tipo de *tablespace* que deseamos construir. Para comprender bien los tipos de *tablespaces* disponibles, junto con las opciones y sintaxis en general de construcción de *tablespaces*, se recomienda consultar **[**ORA02**]**. La segunda zona (arriba a la derecha), tiene dos campos de edición, que son el nombre del *tablespace* (*Nombre*) y el fichero donde se almacenará físicamente (*Fichero*). Estos dos campos siempre estarán disponibles para cualquier tipo de *tablespace* escogido.

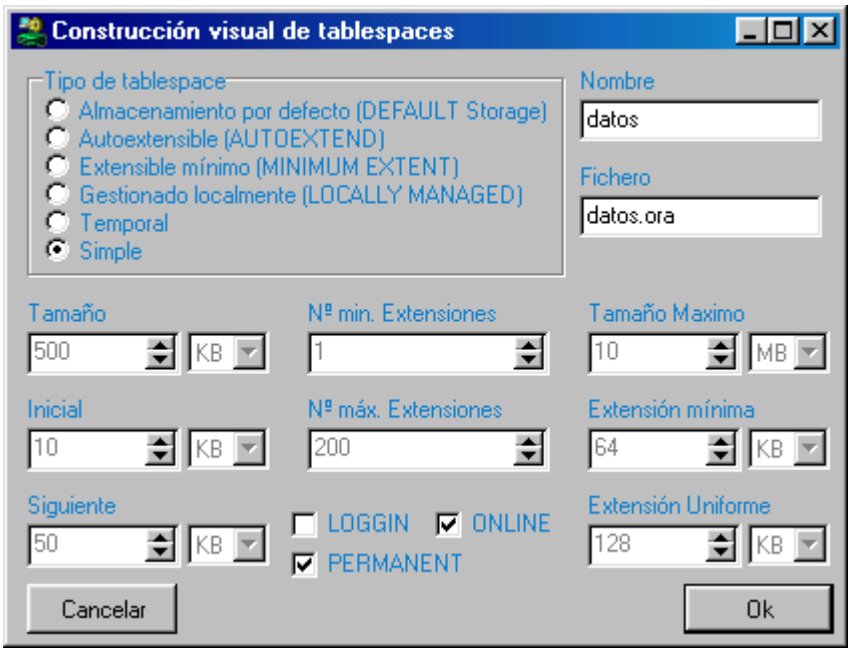

Figura 7.33: Aspecto del formulario de construcción visual de *tablespaces*

La tercera zona, (es la que encontramos a partir de la mitad del formulario hasta abajo). Esta zona está compuesta por numerosos campos, que se activarán o desactivarán en función del tipo de *tablespace* seleccionado (en la zona uno). A la derecha de algunos campos (si representa un tamaño) encontraremos una lista desplegable con las unidades de tamaños disponibles (actualmente dos KB: Kbytes y MB: Megabytes).

A continuación se describirá la correspondencia entre los campos del formulario y la cláusula de la sentencia `CREATE TABLESPACE`, por si se desea consultar el manual de Oracle **[**ORA02**]**. El campo *Tamaño* indica el tamaño total del *tablespace*. Se corresponde con la cláusula `SIZE`. El campo *Inicial* indica el tamaño de la extensión inicial. Se corresponde con la cláusula `INITIAL`. El campo *Siguiente* indica el tamaño de los siguientes ficheros de datos cuando se requieran más extensiones. Se corresponde con la cláusula `NEXT`. Los campos *Nº mín. extensiones* y *Nº máx. extensiones*, representan el número mínimo y máximo de extensiones. Se corresponden con las cláusulas `MINEXTENTS` y `MAXEXTENTS` respectivamente. El campo *Tamaño máximo* representa el tamaño máximo de los ficheros. Se corresponde con la cláusula `MAXSIZE`. El campo *Extensión mínima* se utiliza cuando es autogestionada. Se corresponde con la cláusula `MINIMUM EXTENT`. El campo *Extensión uniforme* se utiliza en las gestionadas localmente. Se corresponde con la cláusula `UNIFORM`.

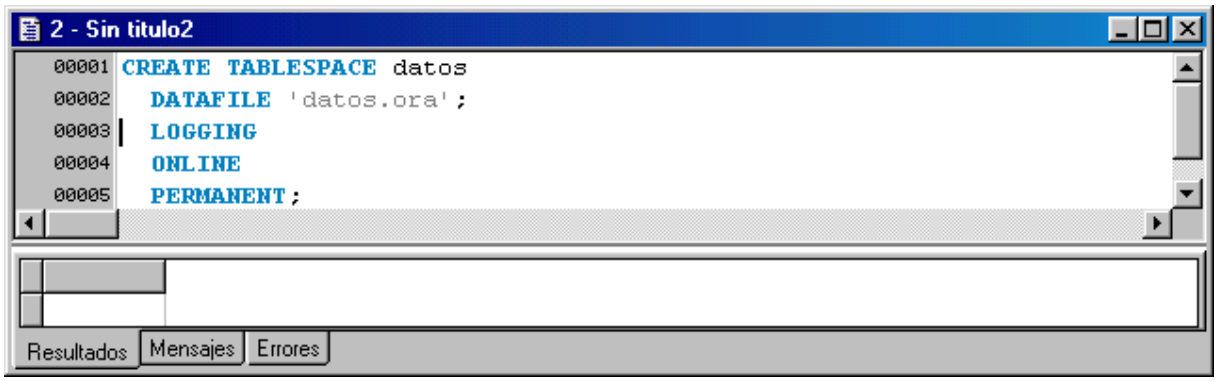

Figura 7.34: Código del *tablespace* generado por la opción de construcción de *tablespaces* con los datos Figura 7.33

Para terminar, también encontraremos tres casillas de verificación *LOGGIN* (permite guardar los atributos de los objetos en el diccionario de datos), *ONLINE* (permite que el *tablespace* esté disponible y *PERMANENT* (distingue entre permanente o temporal).

En la Figura 7.34 se puede ver el resultado de la generación del código del *tablespace*. En este caso, se ha generado el código de un *tablespace simple*.

El control de errores de esta opción es similar al del resto de la aplicación. La única diferencia es que no puede haber un campo numérico vacío, y por tanto si el usuario deja algún campo vacío, se toma el valor por defecto y se le informa al usuario.

7.8.3.9 Construcción de perfiles de usuario

Medusa también incluye una opción para la construcción de perfiles de usuarios, mediante la sentencia `CREATE PROFILE`. Se puede tener acceso a esta opción haciendo click en *Oracle* | *Construir perfil*, o bien, mediante ALT + F. En la Figura 7.35 se puede ver el aspecto de este formulario. Tal y como se observa en dicha figura, el formulario está compuesto básicamente de tres zonas.

La primera zona es el nombre del perfil que deseamos construir. La segunda zona (se encuentra a la izquierda), representa los límites de los recursos. A continuación explicamos dichos límites (cláusulas) todos referidos a la sentencia `CREATE PROFILE`:

- Sesiones / usuario. Representa el límite de sesiones por usuario. Se corresponde con la cláusula `SESSIONS_PER_USER`.
- CPU / sesión. Limita el tiempo de CPU por sesión (en centésimas de segundo). Se corresponde con la cláusula `CPU_PER_SESSION`.
- CPU / llamada. Limita el tiempo de CPU (en centésimas de segundo) para las llamadas (`EXECUTE` o `FETCH`). Se corresponde con la cláusula `CPU_PER_CALL`.
- Tiempo de conexión. Limita el total de tiempo de una sesión (expresado en minutos). Se corresponde con la cláusula `CONNECT_TIME`.
- Tiempo de inactividad. Limita el periodo continuo de inactividad durante una sesión (expresado en minutos). Las consultas largas (*long-running queries*) y otras operaciones no están sujetas a este límite. Se corresponde con la cláusula `IDLE_TIME`.
- Lecturas lógicas / sesión. Especifica el número de bloques de datos leídos en una sesión, incluyendo bloques leídos de memoria y disco. Se corresponde con la cláusula `LOGICAL_READS_PER_SESSION`.
- Lecturas lógicas / llamada. Especifica el número de bloques de datos leídos en una llamada para procesar una orden SQL (`EXECUTE` o `FETCH`). Se corresponde con la cláusula `LOGICAL_READS_PER_CALL`.
- Límites combinados. Especifica el total de costo de los recursos de una sesión (expresado en unidades de servicio). Oracle calcula el total de unidades de servicio como una suma ponderada de `CPU_PER_SESSION`, `CONNECT_TIME`, `PRIVATE_SGA` y `LOGICAL_READS_PER_SESSION`. Se corresponde con la cláusula `COMPOSITE_LIMIT`. Para más información se puede consultar **[**ORA02**]**.
- SGA privado. Especifica la cantidad de espacio privado que una sesión puede reservar en el *shared pool* del *System Global Area* (SGA). Se puede utilizar K o M para especificar este límite en kilobytes o megabytes (por defecto está expresado en bytes). Este límite sólo es aplicable en el caso de utilizar una arquitectura *multi-thread*. El espacio privado de una sesión en el SGA incluye el área privada SQL y PL/SQL, pero no las áreas compartidas. Se corresponde con la cláusula `PRIVATE_SGA`.

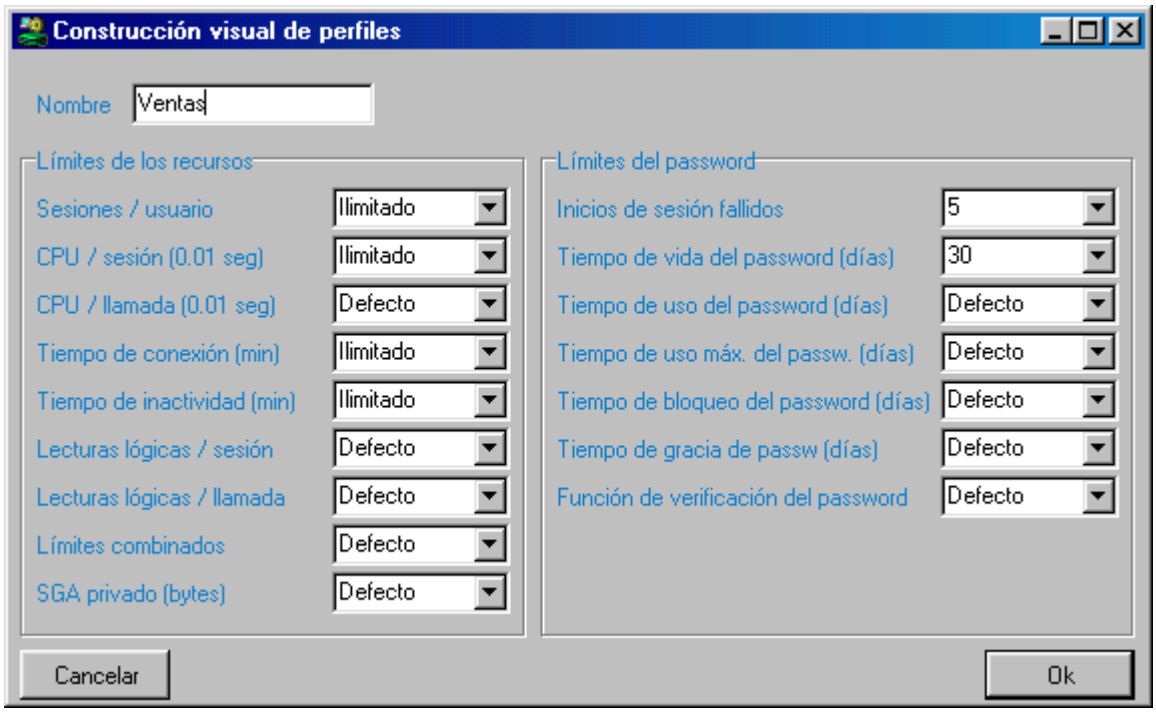

Figura 7.35: Aspecto del formulario de generación de perfiles de usuario

La tercera zona (se encuentra a la derecha), representa los límites de las contraseñas. Dichos límites son (todas las cláusulas referidas son de la sentencia `CREATE PROFILE`):

- Inicios de sesión fallidos. Especifica el número de intentos fallidos de login en la cuenta de usuario antes de que ésta sea bloqueada. Se corresponde con la cláusula `FAILED_LOGIN_ATTEMPTS`.
- Tiempo de vida del password. Limita el número de días en que se puede utilizar la misma contraseña para autentificarse ante el sistema. La contraseña expira si no se cambia en ese período, provocando que se rechacen futuras conexiones. Se corresponde con la cláusula `PASSWORD_LIFE_TIME`.
- Tiempo de uso del password. Especifica el número de días antes de que una contraseña no se pueda usar. Si se le asigna un valor, entonces, debe asignar un valor ilimitado al *Tiempo de uso máx. de passw*. Se corresponde con la cláusula `PASSWORD_REUSE_TIME`.
- Tiempo de uso máx. del passw. Especifica el número de cambios de contraseña antes de que la contraseña pueda ser reutilizada. Si se le asigna un valor, entonces, debe asignar un valor ilimitado al *Tiempo de uso del password*. Se corresponde con la cláusula `PASSWORD_REUSE_MAX`.

- Tiempo de bloqueo del password. Especifica el número de días que una cuenta estará bloqueada después del *Inicios de sesión fallidos*. Se corresponde con la cláusula `PASSWORD_LOCK_TIME`.
- Tiempo de gracia de passw. Especifica el número de días después de que el tiempo de gracia ha comenzado, durante el cual se muestra un mensaje de advertencia y se permite la conexión. Si la contraseña no es cambiada durante el período de gracia, ésta expirará. Se corresponde con la cláusula `PASSWORD_GRACE_TIME`.
- Función de verificación del password. Permite indicar como argumento la rutina de verificación del *script* (PL/SQL) a emplear para verificar la contraseña. Oracle proporciona un *script* por defecto, pero se puede construir uno a medida. También se puede utilizar un valor especial (`NULL`), para indicar que no se realizará la verificación de la contraseña. Para más información se puede consultar **[**ORA02**]**. Se corresponde con la cláusula `PASSWORD_VERIFY_FUNCTION`.

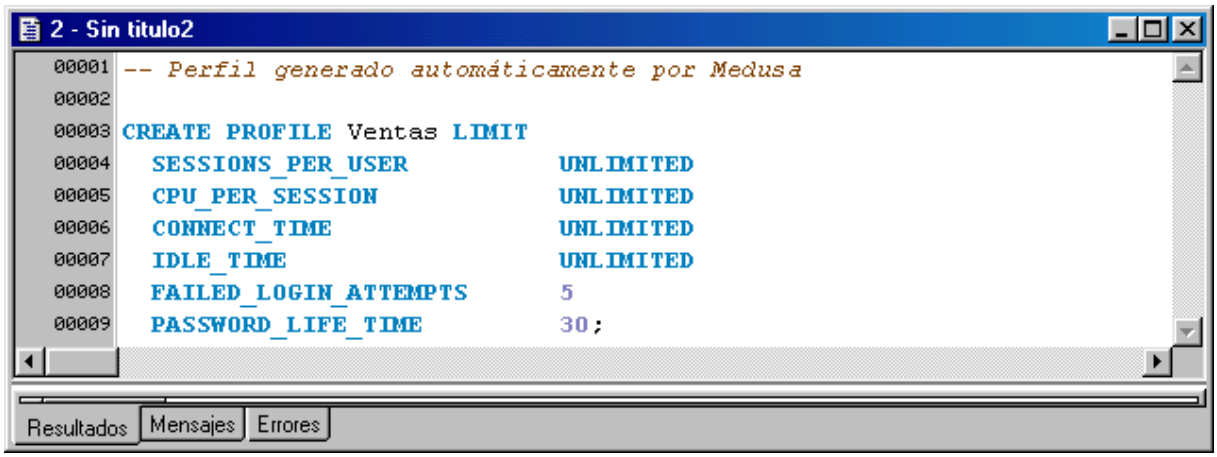

Figura 7.36: Código generado del perfil Ventas con los datos de la Figura 7.35

7.8.3.10 Construcción de usuarios

Otra de las opciones que incluye *Medusa* es la construcción visual de usuarios. El aspecto del formulario de creación de usuarios se puede ver en la Figura 7.37. Se puede tener acceso a esta opción en el menú *Oracle* y después, haciendo click en *Construir usuario*. Se ofrece un acceso directo a esta opción desde la aplicación, el cual es ALT + U.

El formulario de creación de usuarios, está compuesto principalmente por el nombre, la contraseña (también se permite la identificación externa) y los *tablespaces* por defecto y temporal (son listas desplegables que se cargan automáticamente al acceder al formulario con los *tablespaces* disponibles). Opcionalmente se pueden asignar cuotas a los tablespaces, activando la casilla de verificación *Habilitar cuota*, y posteriormente ingresando el valor en el campo de edición correspondiente (se puede emplear K para Kilobytes y M para Megabytes, tal y como muestra la Figura 7.37). Podremos asignar un perfil al usuario que estamos creando, simplemente escribiendo el nombre del perfil deseado, en la casilla titulada *Perfil*.

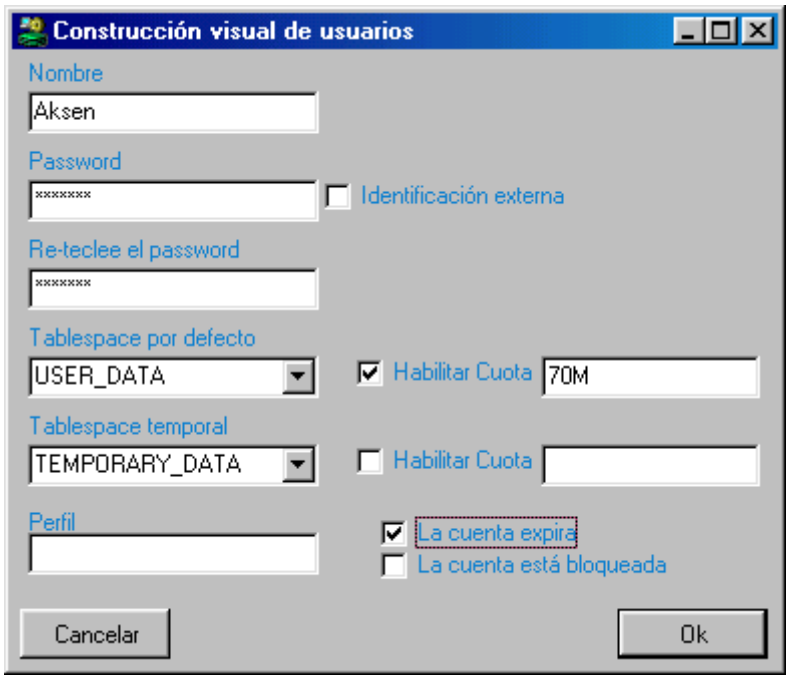

Figura 7.37: Aspecto del formulario de creación de usuarios

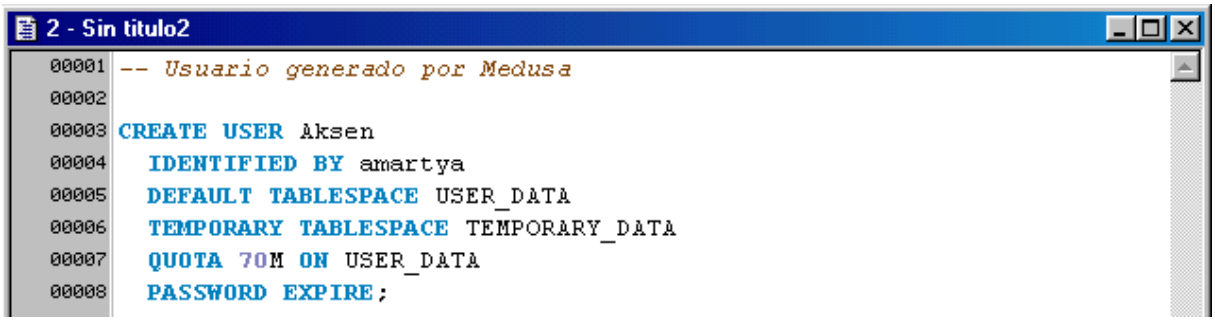

Figura 7.38: Código generado de la opción de creación de usuarios, según los datos de la Figura 7.37

Para finalizar, también podremos configurar opciones básicas de la cuenta de usuario, como son, si la cuenta expira o no, o si la cuenta está bloqueada o no, marcando o desmarcando las casillas de verificación correspondientes. En la Figura 7.38 se puede observar el código resultante de esta opción.

7.8.3.11 Construcción de índices

Otra opción que también se incluye en *Medusa*, es la posibilidad de crear índices de forma visual. Tendremos acceso a esta opción en el menú *Oracle* | *Construir índice*, o bien, mediante el acceso de teclado ALT + I. Tal y como se observa en la Figura 7.39, se dispone de tres tipos básicos de índices, que son: Índices simples (o estándar), De clave inversa (son índices que invierten el orden de los bytes antes de almacenar la(s) clave(s) del índice), Bitmap (índice que utiliza una función de mapeo para identificar la relación de las filas con sus respectivos RowID's).

Para crear un índice primero ingresaremos el nombre del índice (en el campo etiquetado por *Nombre*). Posteriormente seleccionaremos la tabla de la lista desplegable que muestra todas las tablas del usuario disponibles. Acto seguido seleccionaremos el tipo de índice que deseamos crear. Al seleccionar la tabla, se cargarán automáticamente todas las columnas en la lista de la izquierda denominada *Columnas de la tabla*, ordenadas alfabéticamente. Una vez que se han cargado las columnas, podremos seleccionar una (o varias manteniendo pulsada la tecla `CTRL`, o incluso un rango manteniendo pulsada la tecla `SHIFT`) y moverla a la lista que formarán las columnas del índice, pulsando el botón *Agregar*. Es importante resaltar que el orden en la creación de índices tiene gran relevancia y por ello, se han incluido un par de botones que suben o bajan la selección una posición (denominados respectivamente *Subir* y *Bajar*). Por último, sólo queda decir que también se pueden eliminar columnas de la lista de *Columnas del índice*, simplemente seleccionando las columnas deseadas y pulsando el botón *Quitar*. Entonces, volverán a la lista de *Columnas de la tabla*.

En la Figura 7.40 se puede observar el código generado automáticamente con la opción de generación de índices. Tal y como se observa en dicha figura, el índice está definido sobre la tabla `PAISES` y sólo afecta al campo `NOMBRE`.

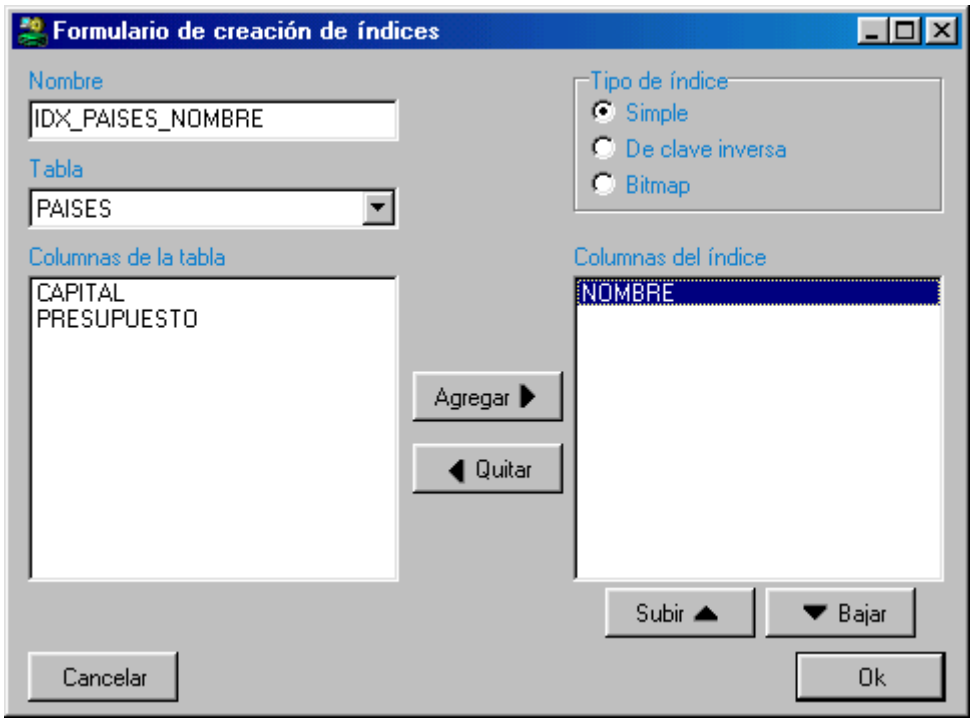

Figura 7.39: Aspecto del formulario de creación de índices

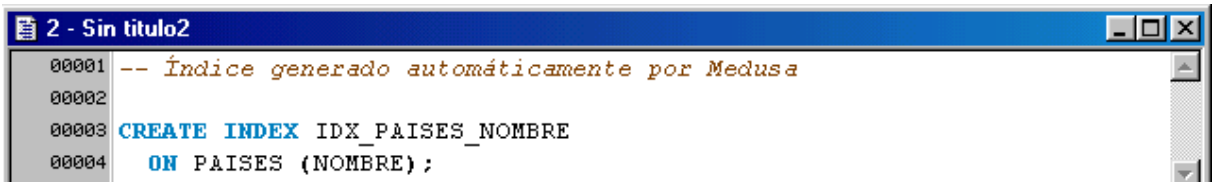

Figura 7.40: Código generado del índice `IDX_PAISES_NOMBRE` con los datos de la Figura 7.39

Al igual que el resto de los formularios, éste también se puede cerrar, abortando la operación, mediante la tecla *escape* (ESC).

7.8.4 Reconstrucción de la sentencia `CREATE TABLE`

Una opción también muy interesante es la posibilidad de obtener la sentencia `CREATE TABLE`, de una tabla concreta para obtener así sus características. Podremos acceder a esta opción mediante el menú *Oracle* | *Reconsruir sentencia CREATE TABLE*, o bien, mediante ALT + N. La idea es poder ver los atributos, sus restricciones, sus referencias, comentarios, etc. También puede

ser útil si queremos crear tablas iguales para otro usuario, pero no tenemos las sentencias `CREATE TABLE`. En la Figura 7.41, se muestra el aspecto del formulario.

Figura 7.41: Aspecto del formulario de reconstrucción de tablas

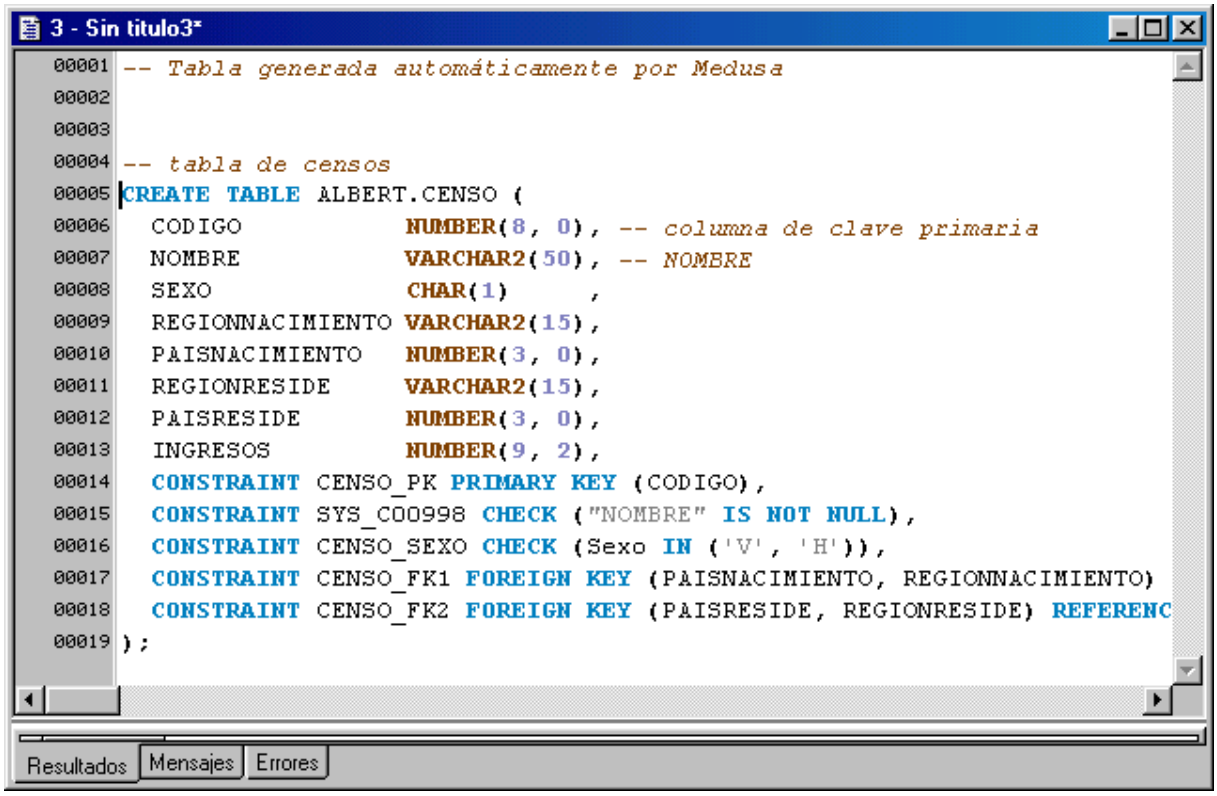

Figura 7.42: Resultado de la reconstrucción de la sentencia `CREATE TABLE`

El proceso de reconstrucción de la sentencia DDL de la tabla se lleva a cabo efectuando múltiples consultas a las vistas que forman el diccionario de datos de Oracle, entre las cuales están `ALL_TABLES`, `ALL_CONSTRAINTS` entre otras.

Tal y como se observa en la Figura 7.42 la reconstrucción incluye las restricciones en forma de restricciones de tabla, junto con los nombres de las restricciones, que pueden ser generados

automáticamente por Oracle cuando la tabla fue realmente creada. Igualmente se incluyen los comentarios de la tabla y de sus atributos.

Se puede salir de este formulario pulsando el botón cerrar (aspa que está en el borde superior derecho) o bien pulsando la tecla *escape* (ESC).

El control de errores es muy sencillo, el único error que puede ocurrir en este caso es que no se ingrese el nombre de la tabla, o bien, que la tabla no exista (lógicamente ambos bajo control).

7.8.5 Gestión visual de permisos

En esta opción, conseguiremos conceder y revocar permisos a un usuario de forma visual y fácil. Tendremos acceso a esta opción mediante el menú *Oracle* | *Gestión de permisos*, o bien, su acceso de teclado alternativo que es ALT + G. En la Figura 7.43 se observa como el formulario, posee ocho categorías de permisos, entre las que encontramos los roles, privilegios, tablas, vistas, vistas materializadas, procedimientos, funciones y paquetes. Al seleccionar una categoría, se cargarán automáticamente los permisos que pueden ser concedidos y revocados sobre esa categoría concreta. En función de la categoría se habilitará la selección múltiple de permisos, sólo estando disponible para las categorías de roles y privilegios. Igualmente, en función de la categoría, se habilitará la lista desplegable de objetos, donde se podrá seleccionar el objeto concreto sobre el que se desee realizar la operación.

Si se observa la Figura 7.43 con detenimiento, se puede ver como la categoría seleccionada es la de roles y por tanto se ha habilitado la selección múltiple y se hace uso de ésta. Seleccionando múltiples roles.

Si no se disponen privilegios para otorgar un determinado permiso, *Medusa* mostrará un mensaje de error para avisar al usuario del problema. Por el contrario, si la concesión se lleva acabo sin ningún tipo de contratiempo, también se mostrará un mensaje para avisar al usuario de que se ha podido llevar a cabo sin problemas.

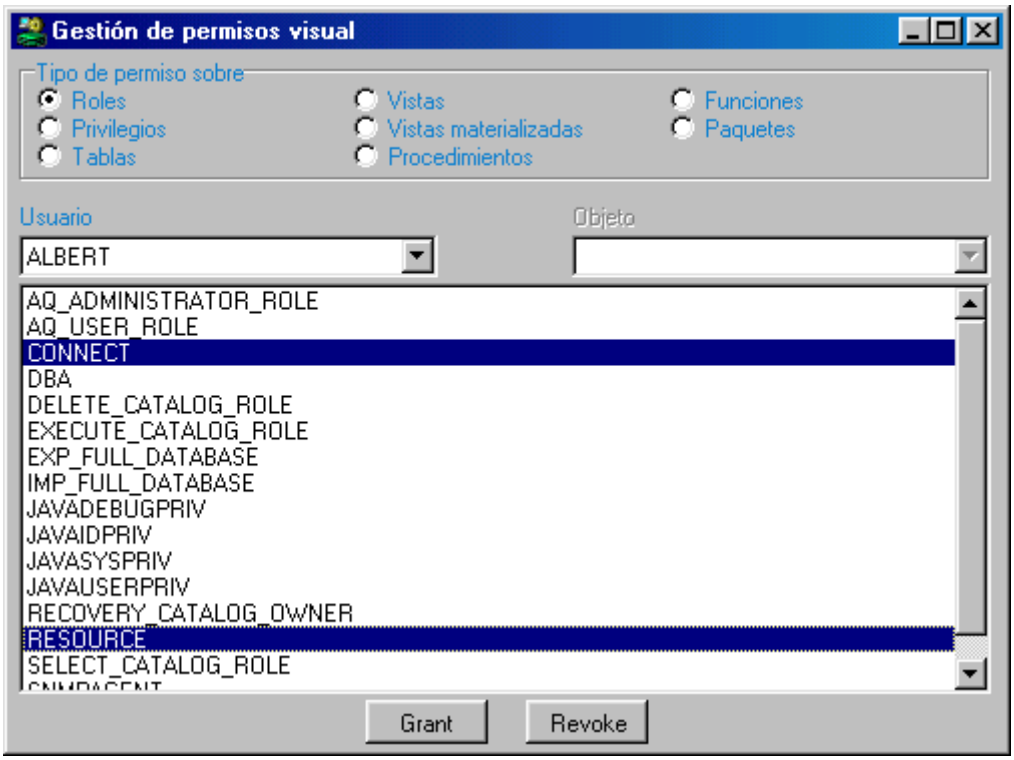

Figura 7.43: Aspecto visual del formulario de gestión de permisos

7.8.6 Formato de fechas

Medusa incorpora un formulario que facilita la manipulación de formatos de fechas, con el objetivo, de no tener que recordar los formatos, los cuales son fáciles de confundir y olvidar, ya que existen multitud de formatos. Tendremos acceso a esta opción, desde el menú *Oracle* | *Formatos de fechas*, o bien, mediante la opción ALT + D.

El formulario de manipulación de formatos de fechas, se muestra en la Figura 7.44, donde se puede observar, que se ha creado un formato que representa el número de la semana del año, seguido del año. El formulario, dispone de una tabla en la parte superior, y un campo de edición llamado expresión. En la tabla se muestran los formatos disponibles. Por último, existe un botón que copia el formato construido al portapapeles para que esté disponible posteriormente y pueda ser pegado en cualquier ventana de edición.

El funcionamiento es muy sencillo, el usuario va tecleando la expresión deseada y cuando quiera, puede hacer doble click sobre cualquier elemento de la rejilla y éste se añadirá justo donde se encuentre el cursor en la parte de la expresión en ese momento.

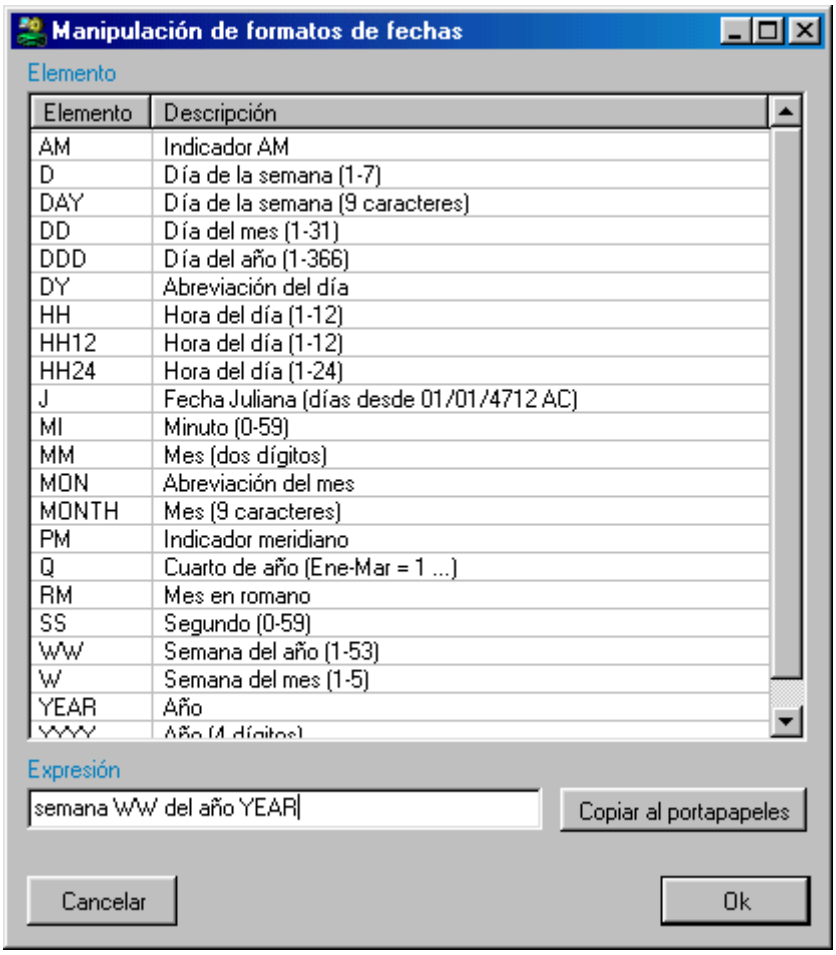

Figura 7.44: Aspecto del formulario de manipulación de fechas

Este formulario al igual que el resto, también se puede cerrar mediante la tecla *escape*, o bien, pulsando el botón *cancelar*.

7.8.7 Gestión visual de comentarios de tablas y columnas en el diccionario de datos

Hemos incluido en *Medusa*, una opción para la tan olvidada manipulación de comentarios en el diccionario de datos. Podremos acceder a esta opción mediante el menú *Oracle* | *Comentarios*, o bien, mediante su acceso desde teclado ALT + M. Se trata de un formulario donde se muestran los comentarios de las tablas y las columnas. En la Figura 7.45, se muestra el aspecto del formulario de manipulación de comentarios. Se observa como se ha seleccionado la tabla CENSO, y se ha editado el comentario, añadiendo la actual selección de texto. Por eso está el botón *Aplicar*

activado, porque se ha modificado el comentario y no se ha guardado. Ambos botones *Aplicar* tienen un significado muy similar: guardar los cambios hechos en la base de datos.

En la zona de las columnas, se ha seleccionado la columna código. Lógicamente las columnas disponibles están directamente relacionadas con la tabla seleccionada. De forma similar a como ocurre con el comentario de la tabla, al no haberse modificado en este caso el comentario de la columna, no se encuentra disponible el botón *Aplicar*.

Los comentarios de tablas, se pueden obtener de la vista `USER_TAB_COMMENTS` o `ALL_TAB_COMMENTS`, mientras que los comentarios de columnas, se pueden obtener de la vista `USER_COL_COMMENTS` o `ALL_COL_COMMENTS`.

El comando de Oracle empleado para modificar el comentario de una tabla es `COMMENT ON TABLE <Tabla> IS <Comentario>`. El comando empleado para modificar el comentario de una columna es muy similar y sigue la sintaxis `COMMENT ON COLUMN <Tabla>.<Columna> IS <Comentario>`.

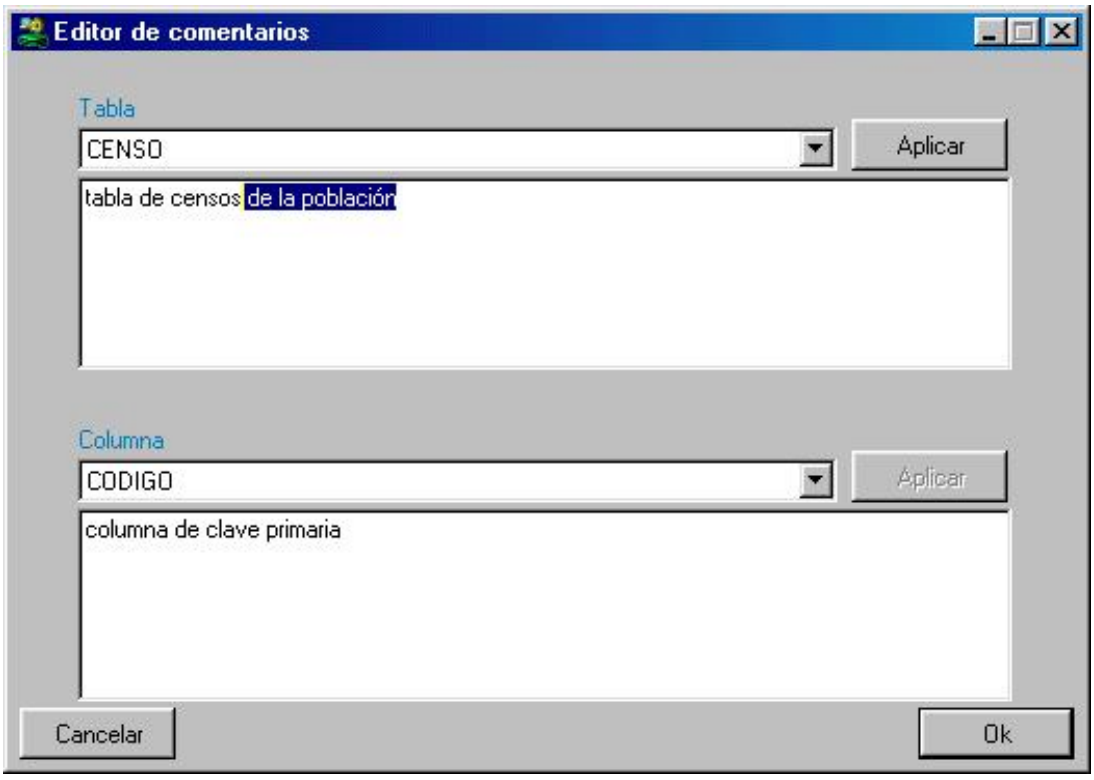

Figura 7.45: Aspecto del formulario de manipulación de comentarios

La idea de este formulario, es simplificar la tarea de consultar estos comentarios o de comentar tablas y columnas empleando los comandos de Oracle, sin que el usuario necesite recordarlos.

7.9 Menú Ayuda

En esta sección veremos las opciones de este menú. Tal y como se detalló en la sección 7.4.9, este menú está compuesto por la ayuda de la aplicación (junto con un acceso directo al índice de contenidos de la ayuda), la ayuda de Oracle y el formulario *Acerca de*.

7.9.1 Ayuda de *Medusa*

Desde el entorno de *Medusa* disponemos de un acceso directo a la ayuda del programa para poder consultarla de forma fácil con sólo pulsar la tecla `F1`, como se observa en Figura 7.46.

Figura 7.46: Aspecto de la ayuda en línea de *Medusa*

7.9.2 Documentación y ayuda de Oracle integrada

Medusa integra, además de la ayuda propia del programa, también la ayuda y documentación de Oracle, En caso de no estar configurada, primero tendremos que configurarla en el cuadro de diálogo de configuraciones (*Herramientas* | *Configuración*) en la pestaña Oracle, hacemos click sobre el botón Documentación y seleccionamos el fichero índice donde resida la documentación de Oracle, bien sea desde el disco duro en caso de haberla instalado o desde el mismo CD-ROM, indicándole el camino. Se dispone también de un acceso directo a esta ayuda, simplemente pulsando la tecla `F2`.

7.9.3 Formulario *Acerca de*

Se trata del formulario típico que solemos encontrar en casi todas las aplicaciones, que nos informa, en nuestro caso, el tamaño del ejecutable, el número de líneas aproximado del código fuente, la versión de la aplicación (actualmente la versión 3), las direcciones de correo electrónico de contacto y las páginas web. El aspecto de este formulario, se puede observar en la Figura 7.47.

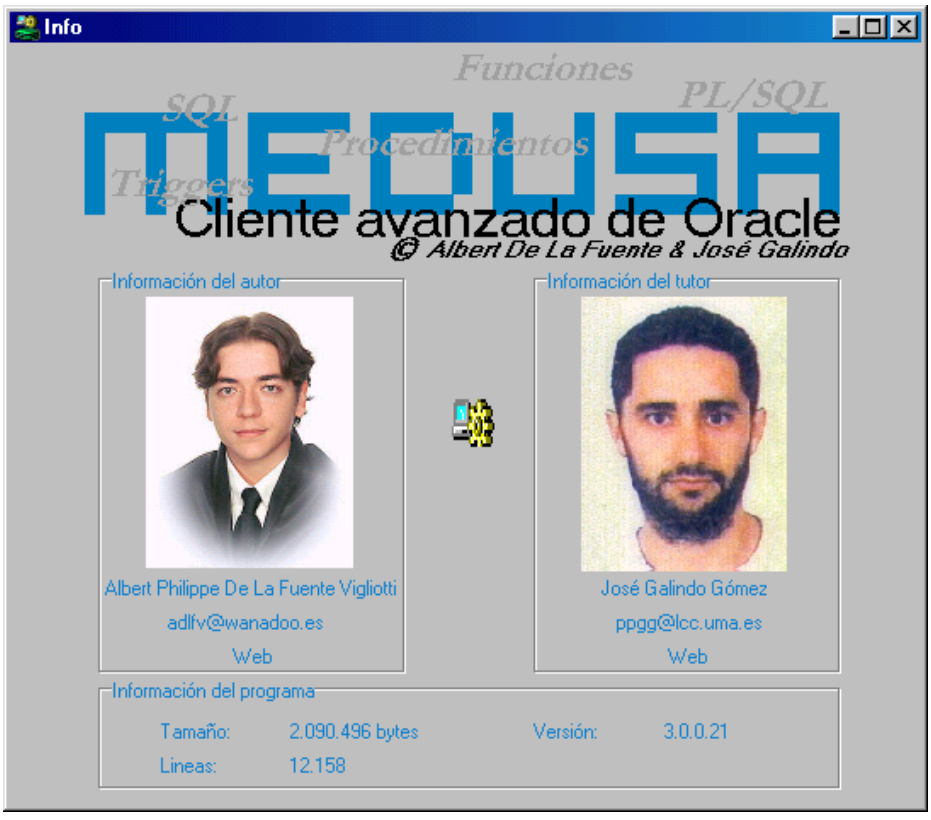

Figura 7.47: Aspecto del formulario *Acerca de*

7.10 Opciones accesibles desde los menús contextuales y mediante combinaciones de teclas

En esta sección veremos las opciones disponibles en *Medusa* que no poseen un acceso desde el menú principal ni desde ninguna de las barras de herramientas. Entre estas opciones encontramos las marcas de línea, la selección del realzado sintáctico, el cambio de tamaño de fuente del editor y de la rejilla.

7.10.1 Utilizando las marcas de línea (*bookmarks*) del editor

Una opción que puede ser muy útil sobre todo en la edición de ficheros grandes, son las *bookmarks* o marcas de líneas. Sobre el editor, podemos pulsar en el botón derecho del ratón y seleccionar en el menú contextual, la opción *Conmutar marcas de línea*, que a su vez contiene un submenú con las 10 posibles marcas de línea que podremos utilizar en todo el documento. La marca se añadirá sobre la línea donde esté el cursor del teclado en ese momento. En la Figura 7.48 se muestra cómo se ha añadido la marca número 0 en la línea 14, empleando el menú contextual. Otro método también es situarse sobre la línea que se desea marcar y pulsar la combinación de teclas CTRL + SHIFT + 0, para añadir la marca 0, e igualmente cambiando el número para añadir otro índice de marca. Desde el menú contextual, también se pueden limpiar todas las marcas, simplemente haciendo click sobre *Limpiar* del menú *Conmutar marcas de línea*. Sólo se puede tener una marca de línea en una línea.

Para ir de forma rápida a una bookmark, igualmente se puede hacer de dos formas, desde el menú contextual en *Ir a marca de línea* y seleccionando la marca deseada o pulsando la combinación de teclas CTRL + 0, para saltar a la marca 0.

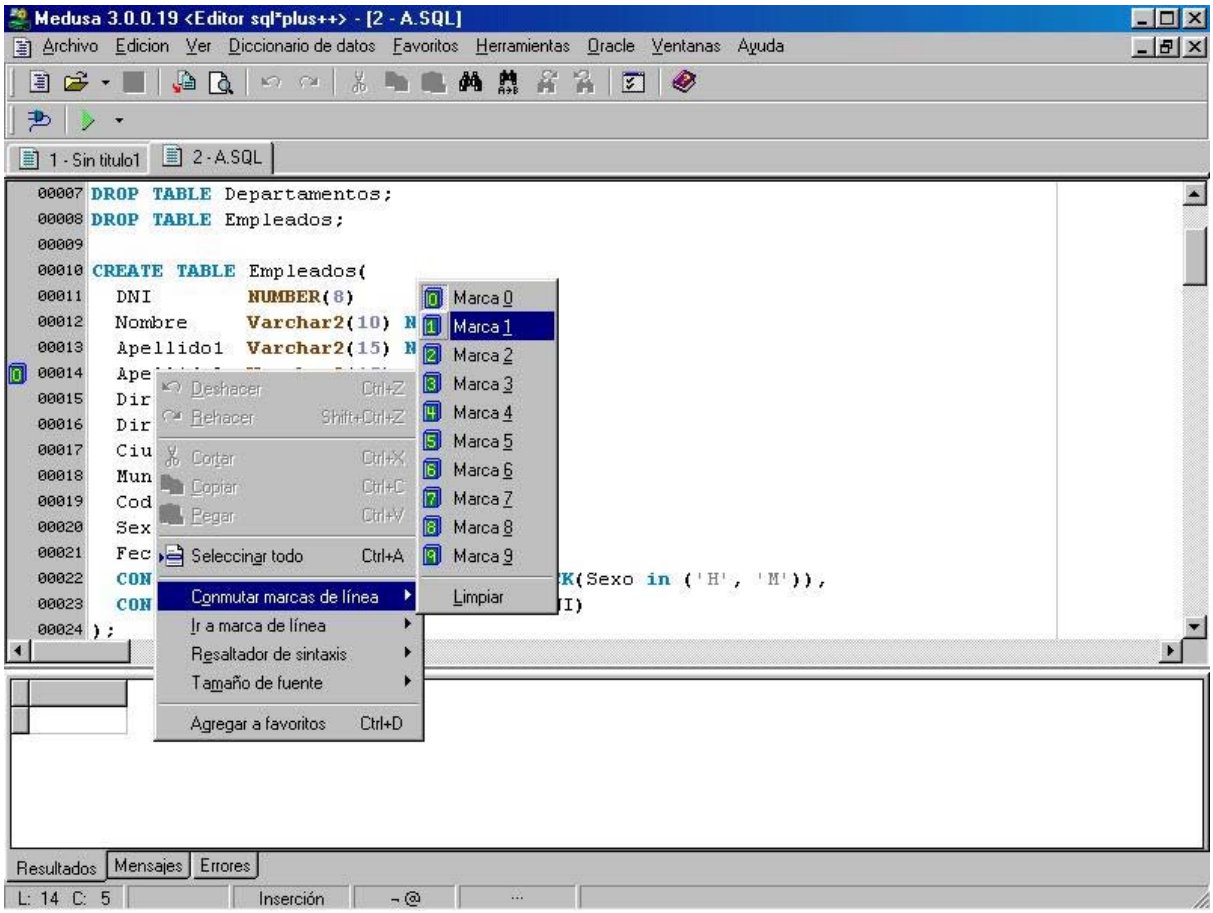

Figura 7.48: Conmutando una marca de línea

Cabe aclarar que una marca sólo puede estar en una línea, pero una línea puede tener varias marcas, es decir, por ejemplo, en la línea 14 se podría añadir la marca 1 y aunque sólo se vería la última marca, las dos hacen referencia a esa línea. Eso significa que si pulsamos la combinación de teclado CTRL + 0, tanto como la combinación de teclado CTRL + 1, ambas nos llevarán a la línea 14. Otro detalle importante es que, por ejemplo, si se tiene la marca 0 en la línea 14 y después se selecciona otra línea con el cursor del teclado y se pulsa sobre el menú correspondiente de conmutación de la marca, ésta se situará sobre la nueva línea que se tenga seleccionada y se eliminará de la línea donde anteriormente estaba.

7.10.2 Seleccionando el realzado de sintaxis del editor

De forma similar a las marcas de línea (bookmarks), se puede seleccionar el lenguaje de realzado de sintaxis. Actualmente *Medusa* dispone de dos lenguajes de realzado de sintaxis SQL

(incluido PL/SQL) y Java. Esta opción puede resultar interesante para poder conmutar al realzado de sintaxis de Java al generar automáticamente un bloque Java, lo cual se explicará más adelante.

Para acceder a los realzados de sintaxis, se debe hacer click sobre el botón derecho, sobre la ventana de edición y seleccionar el menú *Resaltador de sintaxis*, y después elegir el resaltado deseado. Por defecto el realzado de sintaxis es SQL, salvo que se genere un bloque Java, en cuyo caso es Java.

Se puede observar el aspecto del menú de selección del realce sintáctico, junto con el realce sintáctico de Java en la Figura 7.49.

Sólo se puede tener como mucho un tipo de realce sintáctico activo, aunque se puede desactivar el realzado sintáctico para una ventana de edición, simplemente desmarcando ambas casillas. En las opciones de configuración se puede desactivar permanentemente el realce sintáctico por defecto (SQL y PL/SQL).

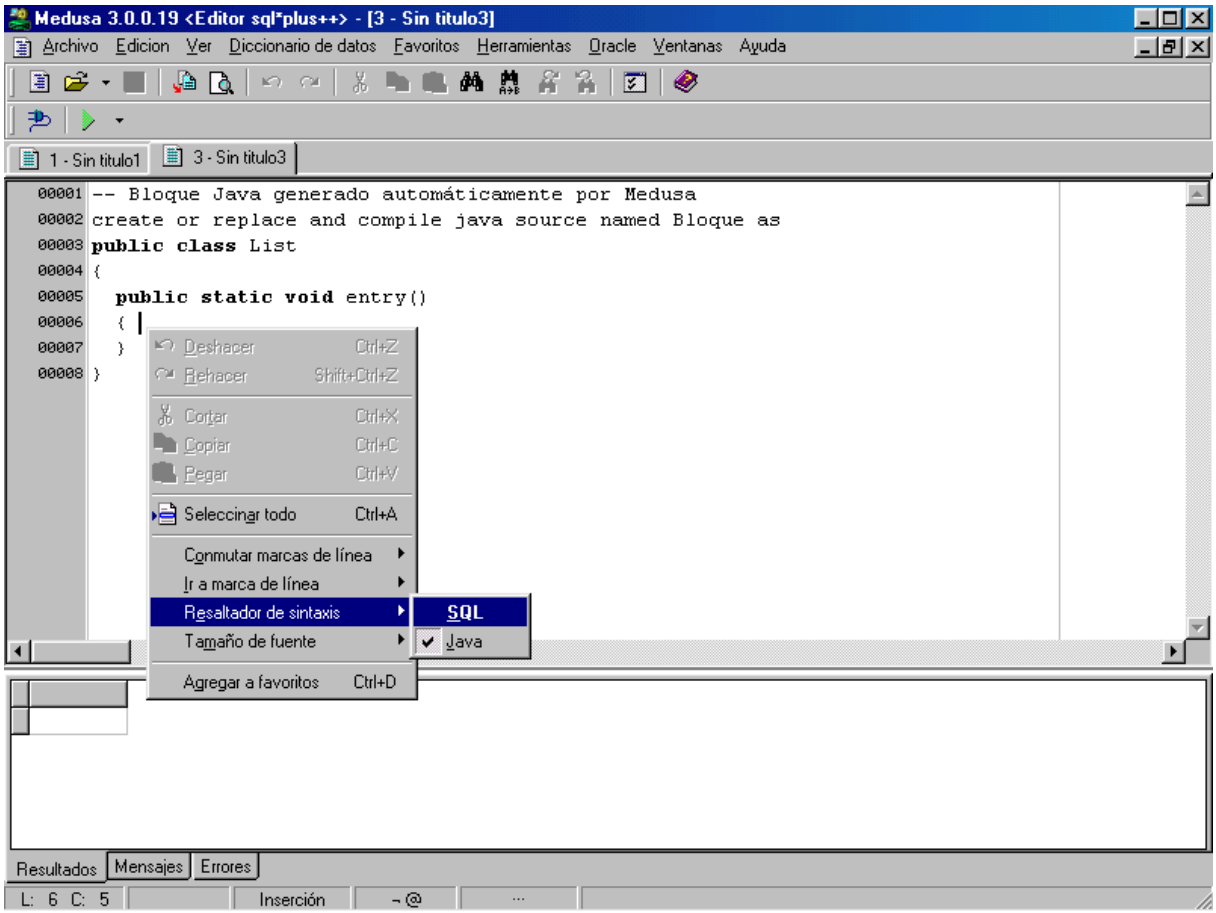

Figura 7.49: Selección del realce sintáctico del documento

7.10.3 Cambiando el tamaño de fuente del editor

Al igual que en el caso anterior, se puede seleccionar el tamaño de la fuente del editor e igualmente de la rejilla de los resultados, simplemente haciendo click sobre el elemento deseado (editor o rejilla) sobre el botón derecho y seleccionando la fuente deseada del menú *Tamaño de fuente*. La fuente aplicada actualmente estará marcada y la fuente por defecto estará en negrita. En la Figura 7.50 se puede ver la forma del menú donde se puede cambiar el tamaño de fuente (el aspecto del menú contextual asociado a la rejilla es muy parecido al menú contextual asociado al editor).

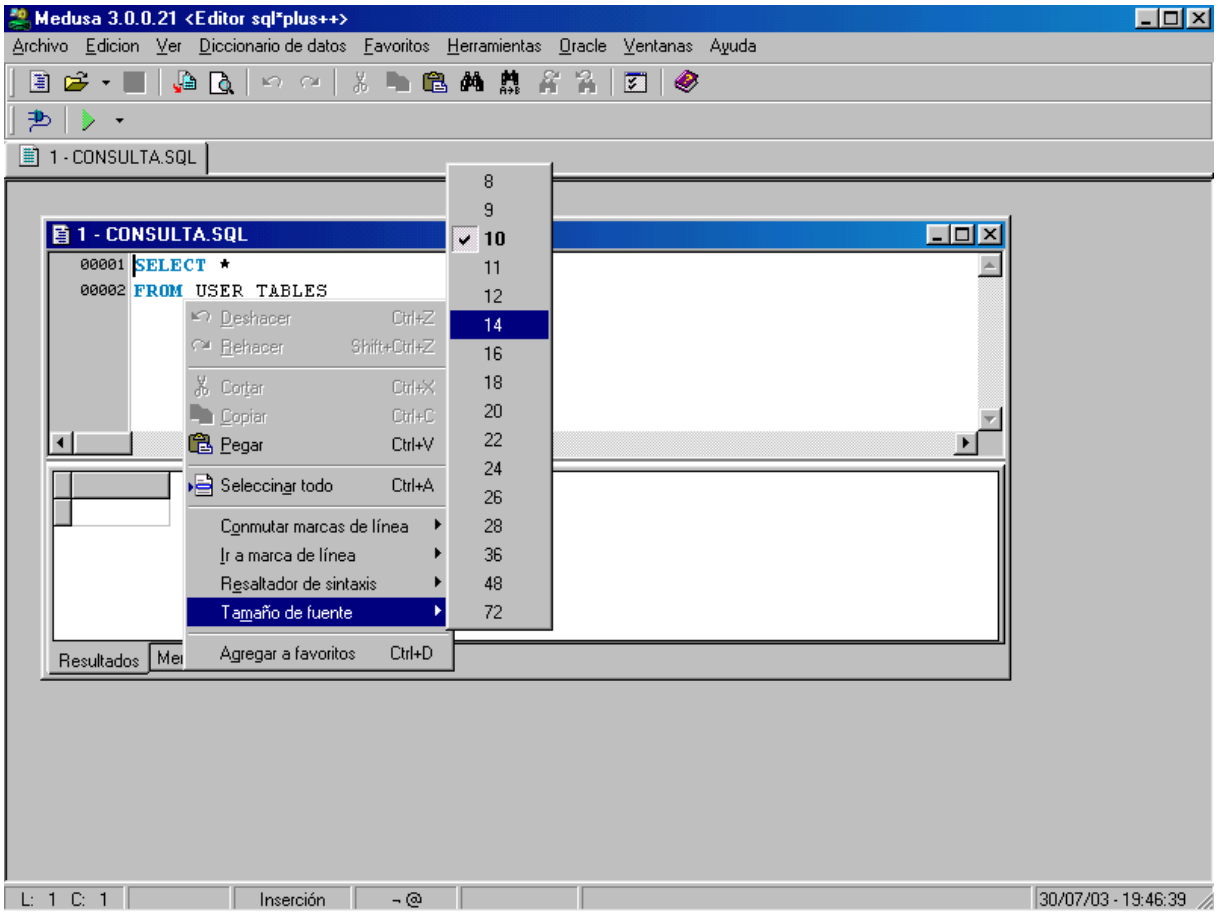

Figura 7.50: Cambiando el tamaño de fuente del editor

7.11 Opciones de configuración

Prácticamente todas las opciones de configuración de *Medusa*, las podemos visualizar y editar, desde el formulario de configuración, el cual podemos abrir, mediante el acceso directo de la barra de herramientas o bien, mediante el menú en *Herramientas* | *Configuración*.

Se ha cuidado mucho la configuración de *Medusa*, para permitir que sea bastante flexible, con el objetivo de que el usuario pueda personalizar casi a su antojo el entorno y funcionamiento del programa. La configuración se divide en siete secciones bien diferenciadas.

7.11.1 Configuración del editor

El aspecto de esta pestaña de configuración es el mostrado en la Figura 7.51.

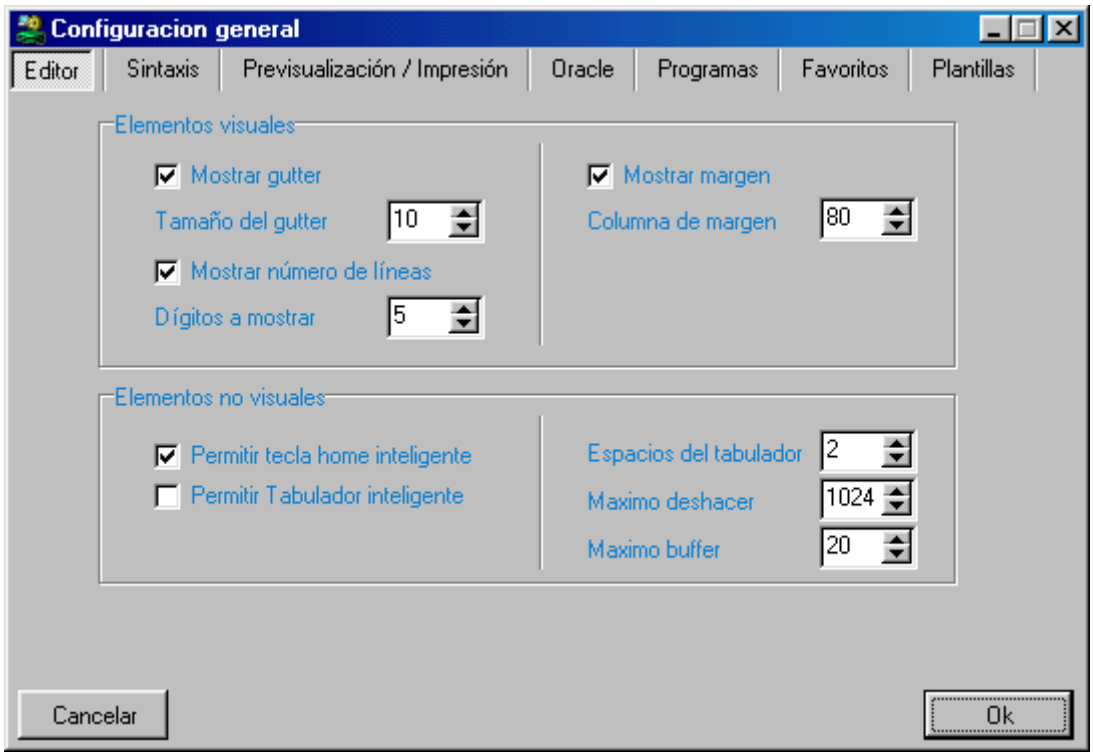

Figura 7.51: Pestaña *Editor* de las opciones de configuración

Tal y como se observa en la Figura 7.51, se incluyen numerosas opciones de configuración del editor, entre las que podemos encontrar:

- Mostrar gutter. Muestra el gutter, con un tamaño fijo indicado por el tamaño del gutter, en pixels.
- Mostrar numeración de líneas. Muestra en el gutter (margen izquierdo del editor), el número de líneas, también se puede indicar el número de dígitos a mostrar, aunque este es un aspecto meramente estético. Para que se muestre la numeración, el gutter debe estar activado.
- Mostrar margen. Una opción recomendable por estética de programación, es no sobrepasar el margen, pues bien, esta opción nos permite mostrar el margen del editor, para no sobrepasarnos y además es completamente configurable.
- Permitir tecla *home* inteligente. Una opción que no suelen incluir los programas de Microsoft Windows, aunque es muy útil para los usuarios, es la tecla *home* inteligente. Si está activada la casilla de verificación, al pulsar la tecla *home*, el cursor, no se va al comienzo de la línea, sino que se va al comienzo del texto. Si se vuelve a pulsar la tecla *home*, entonces si se irá al comienzo de la línea. Ocurre igual en el programa *Kwrite* del sistema operativo *Linux*.
- Permitir tabulador inteligente. Permite configurar el tabulador, para que cuando se tabule una línea, se busque el tabulador más relacionado en la línea superior o inferior. Por defecto esta opción está deshabilitada, puesto que puede resultar no muy práctico para algunos usuarios.
- Espacios del tabulador. Indica el número de espacios que se dejará en el editor al pulsar la tecla tabulador. Por defecto está a dos espacios por tabulador.
- Máximo deshacer. Indica el número máximo de operaciones de deshacer y rehacer que se podrán realizar en la edición. Por defecto 1024.
- Máximo *buffer*. Indica el tamaño máximo del *buffer* circular de textos. Ver sección 7.6.4. Por defecto 20.

7.11.2 Configuración del realzado de sintaxis

El aspecto de esta pestaña de configuración es el mostrado en la Figura 7.52.

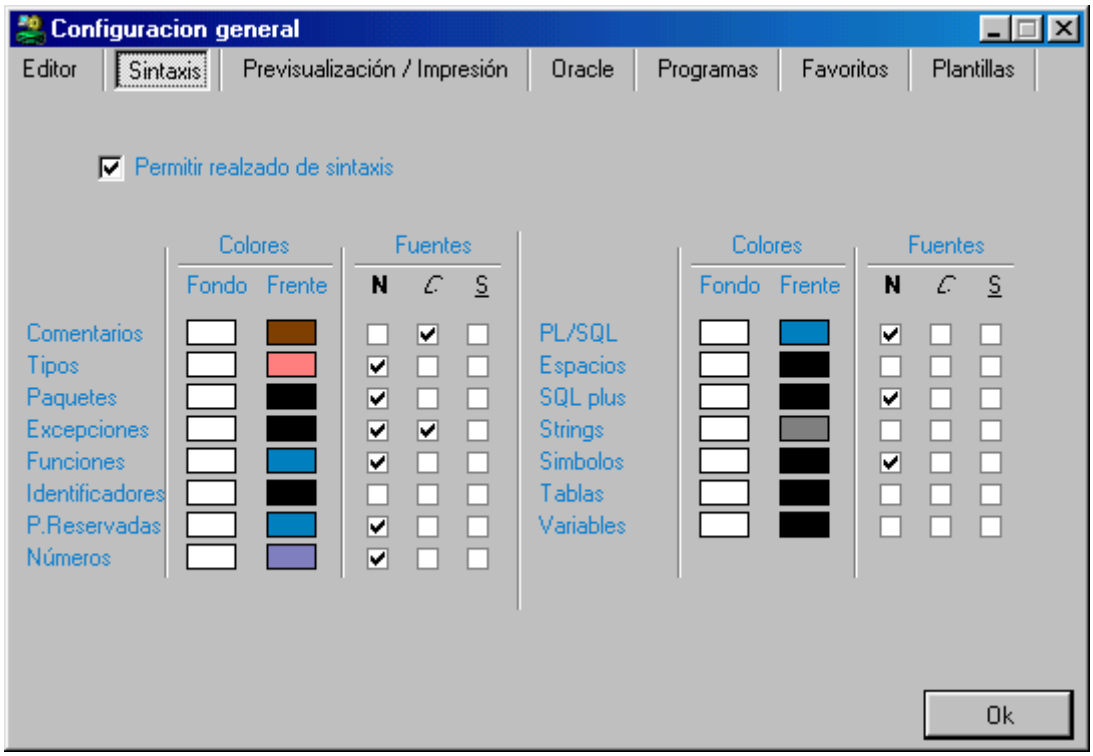

Figura 7.52: Pestaña *Sintaxis* de las opciones de configuración

Tal y como se observa en la Figura 7.52, el formulario básicamente contiene las agrupaciones de palabras reservadas, junto con dos colores (fondo y frente) y los atributos de la fuente (**N**egrita *C*ursiva y S̲ubrayada). Entre las agrupaciones principales, podemos encontrar los comentarios, tipos, funciones, palabras reservadas, números, PL/SQL, strings (texto entre comillas simples) y símbolos. También se puede apreciar, que se dispone de la opción de anular el realzado de sintaxis, aunque esto no es recomendable, puesto que el realzado ayuda a evitar errores al programar y además facilita la visualización. Esta pestaña de realzado sólo afecta a la sintaxis de SQL y PL/SQL, pero no a la de Java, que sólo ser resaltan en negritas las palabras reservadas.

7.11.3 Configuración de la previsualización y de la impresión

El aspecto de esta pestaña de configuración es el mostrado en la Figura 7.53, donde se puede observar, que las principales opciones incluidas son la impresión en blanco y negro, mostrar pie de página, permitir el realzado de sintaxis, enumerar las líneas, el nombre del usuario del programa y, por último, los márgenes del documento. En caso de estar activa la opción de mostrar el pie de página, se mostrará una línea y debajo, la numeración de la página junto con el número de páginas. En la sección 7.5.2 se vio como previsualizar la impresión.

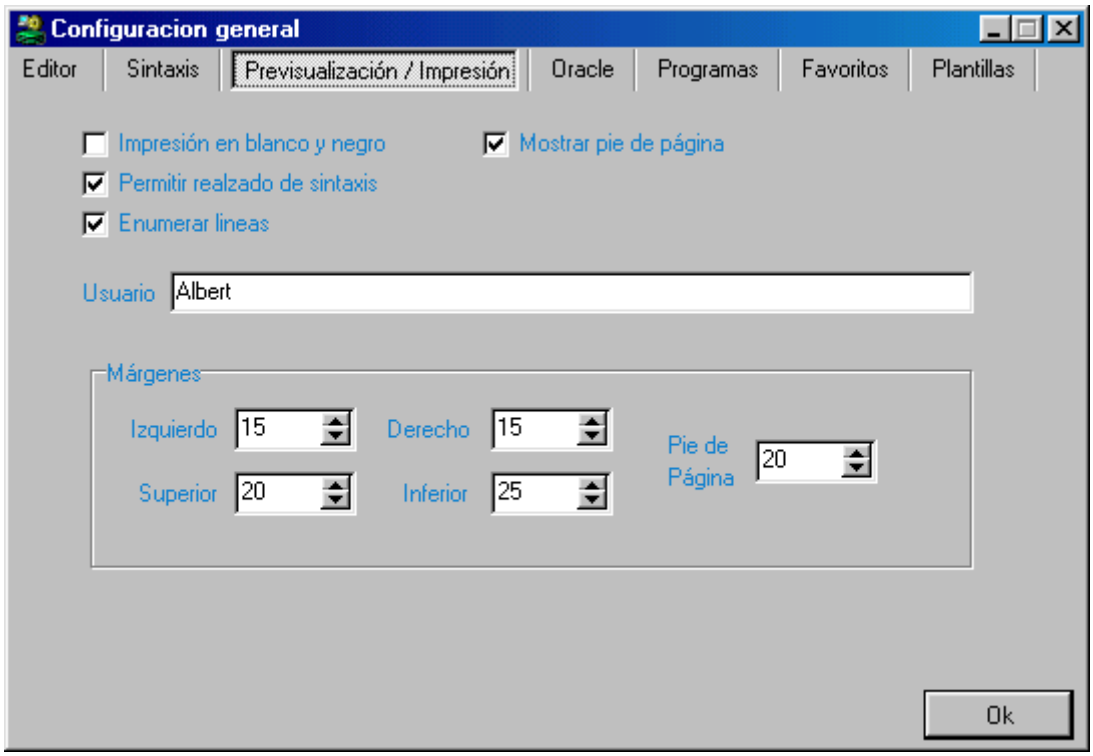

Figura 7.53: Pestaña *Previsualización / Impresión* de las opciones de configuración

7.11.4 Configuración de Oracle

El aspecto de esta pestaña de configuración es el mostrado en la Figura 7.54, donde se puede observar, que las opciones básicamente son dos:

- Documentación. Se utiliza para seleccionar el camino hacia la documentación y la ayuda de Oracle. El camino se mostrará justo debajo, en nuestro caso es `H:\Doc\index.htm`, puesto que no se ha instalado la documentación en el disco duro, sino que se utilizará desde el mismo CD-ROM de Oracle. Para tener acceso a la documentación, basta con pulsar la tecla `F2` desde el editor o elegir la opción *Ayuda de Oracle* desde el menú *Ayuda*.
- Protocolo de conexión. Existen actualmente tres protocolos de conexión, los cuales veremos a continuación.
  - Conectar como ventana activa. La conexión se efectuará, con el usuario y la contraseña de la ventana activa en ese momento. La primera vez (por no haber ventana activa), se solicitará el usuario y la contraseña. (ver Figura 7.4).

- Conectar como nuevo usuario. La conexión se efectuará siempre solicitando el nombre de usuario y la contraseña, cada vez que se abra una nueva ventana.
- Conectar como. La conexión se efectuará con los datos proporcionados en este mismo formulario. Opcionalmente se puede marcar la casilla *Guardar password de auto-login en ini*, que lo que hace es guardar la contraseña en el fichero de configuración. Si no está marcada esta casilla, entonces, la primera vez se solicitará la contraseña y el resto de veces ya no. Hay que tener precaución al guardar la contraseña en el fichero ini, puesto que cualquiera podría tener acceso a la base de datos.

Para cualquier protocolo de conexión que seleccione el usuario, tendremos disponible la opción de conectar automáticamente (*Habilitar conexión automática*), que lo que hace es que al crear una nueva ventana, se intente conectar automáticamente según el protocolo seleccionado. De esta forma si no se marca esta casilla, no se conectará hasta que el usuario pulse el botón conectar, pero sin embargo, sí está marcada la casilla se conectará con sólo crear una nueva ventana de edición.

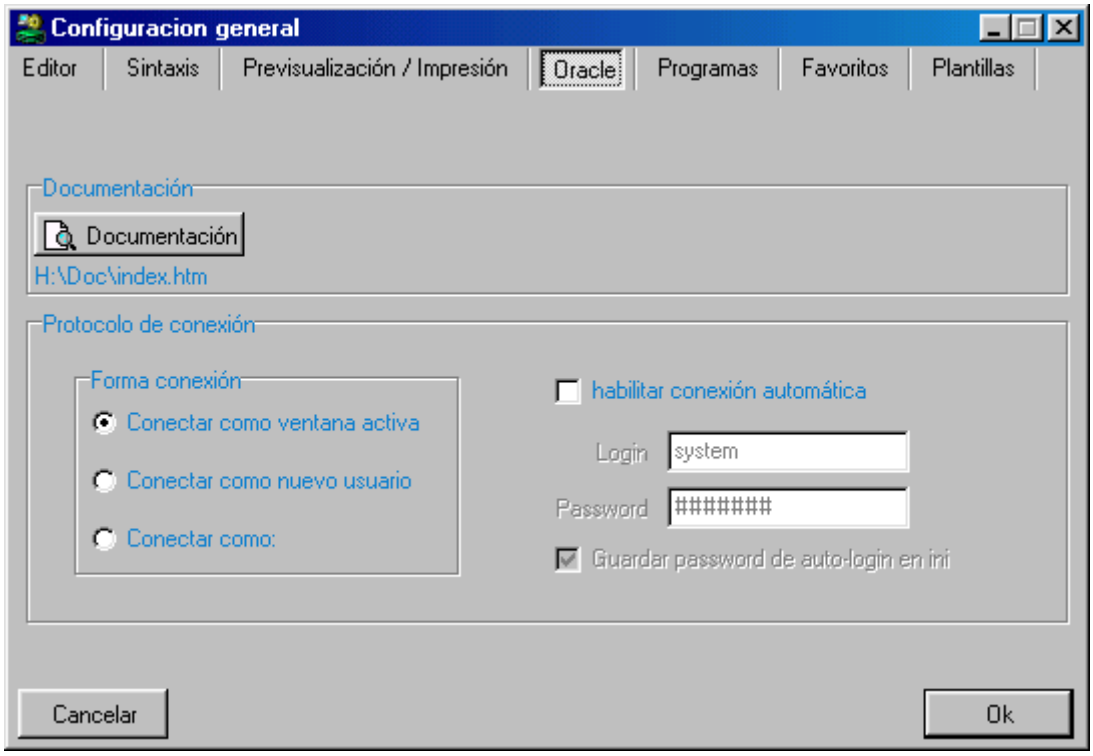

Figura 7.54: Pestaña *Oracle* de las opciones de configuración

7.11.5 Configuración de los programas favoritos (menú Herramientas)

El aspecto de esta pestaña de configuración es el mostrado en la Figura 7.56, donde se puede apreciar que simplemente hay una lista con cinco opciones de configuración de los programas favoritos, donde pulsando el botón correspondiente, nos permitirá añadir el programa. Al seleccionar un programa, después hay que darle un nombre lo suficientemente descriptivo editando la entrada correspondiente al menú, que será el que aparezca en el menú *Herramientas*. De esta forma el usuario podrá personalizar el menú de programas favoritos, para que simplemente haciendo un click, se ejecute el programa indicado con los argumentos indicados Opcionalmente se pueden incluir argumentos en los campos correspondientes a cada programa.

Se dispone de una variable de entorno de *Medusa* representada por **`%fich`**, que representa el nombre del fichero que está en disco guardado y que actualmente está cargado en la ventana de edición activa. Esta variable, se utiliza para pasar el nombre del fichero actual como argumento al programa cuando se invoque. Si el fichero no se ha guardado aun en disco, no tiene nombre y el usuario desea ejecutar un programa con argumento **%fich**, *Medusa* mostrará un mensaje de error para informar de la situación al usuario, brindándole la posibilidad de guardarlo en ese momento. Por el contrario si el usuario una vez que ha guardado, modifica el fichero y después solicita la ejecución de un programa con argumento **%fich**, *Medusa* no tendrá en cuenta las últimas modificaciones.

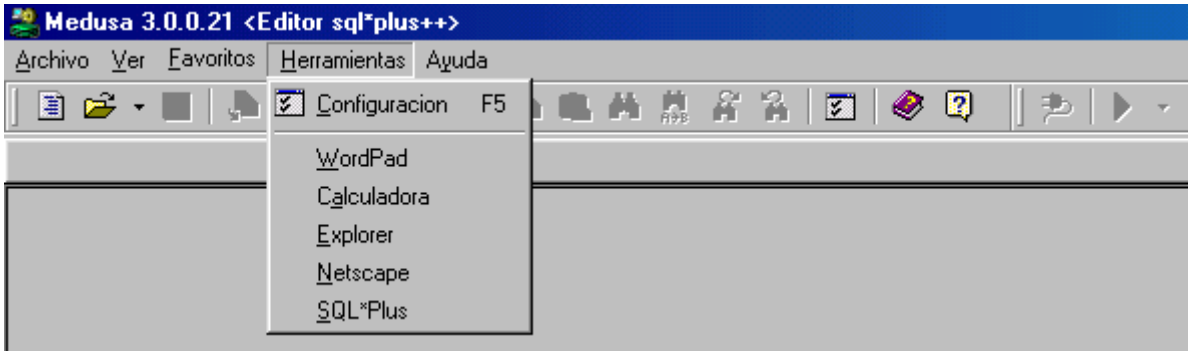

Figura 7.55: Aspecto del resultado en el menú principal

En la Figura 7.55 se puede observar el aspecto del menú que se forma al configurar los programas tal y como se observa en la Figura 7.56.

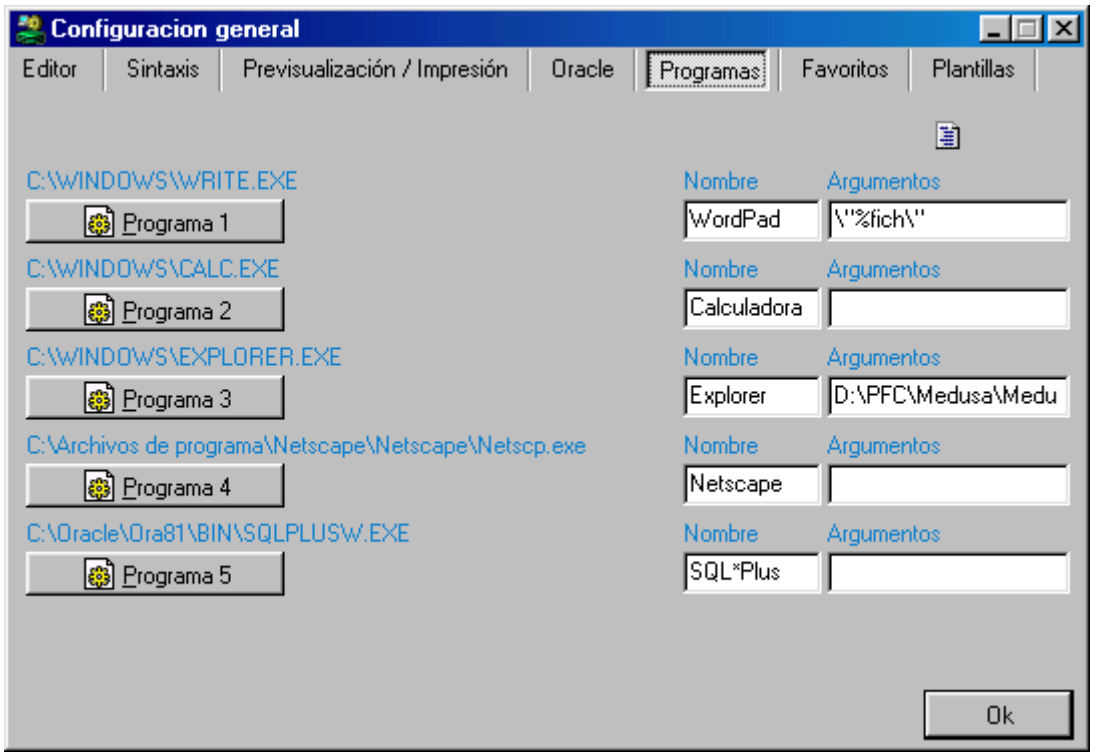

Figura 7.56: Pestaña *Programas* de las opciones de configuración

7.11.6 Configuración del menú de sentencias favoritas (menú Favoritos)

Similar al menú de consultas al diccionario de datos, disponemos de un menú de *scripts* o consultas, pero que éste es configurable, de sentencias favoritas, donde podremos organizar las consultas o los *scripts*, en carpetas y subcarpetas. Los favoritos pueden ser utilizados para tareas repetitivas o como ficheros base para sentencias comunes (*una forma de heredar de un fichero ya creado y depurado, similar al mecanismo de herencia en la programación*).

La forma de agregar una consulta al menú favoritos, es simplemente haciendo click sobre el botón derecho en la ventana de edición y seleccionar Agregar a favoritos, después se abrirá una ventana, donde podremos crear la estructura que deseemos para organizar este elemento que agregaremos. Una vez creada la estructura (jerarquía de directorios), crearemos el fichero y asociaremos la sentencia.

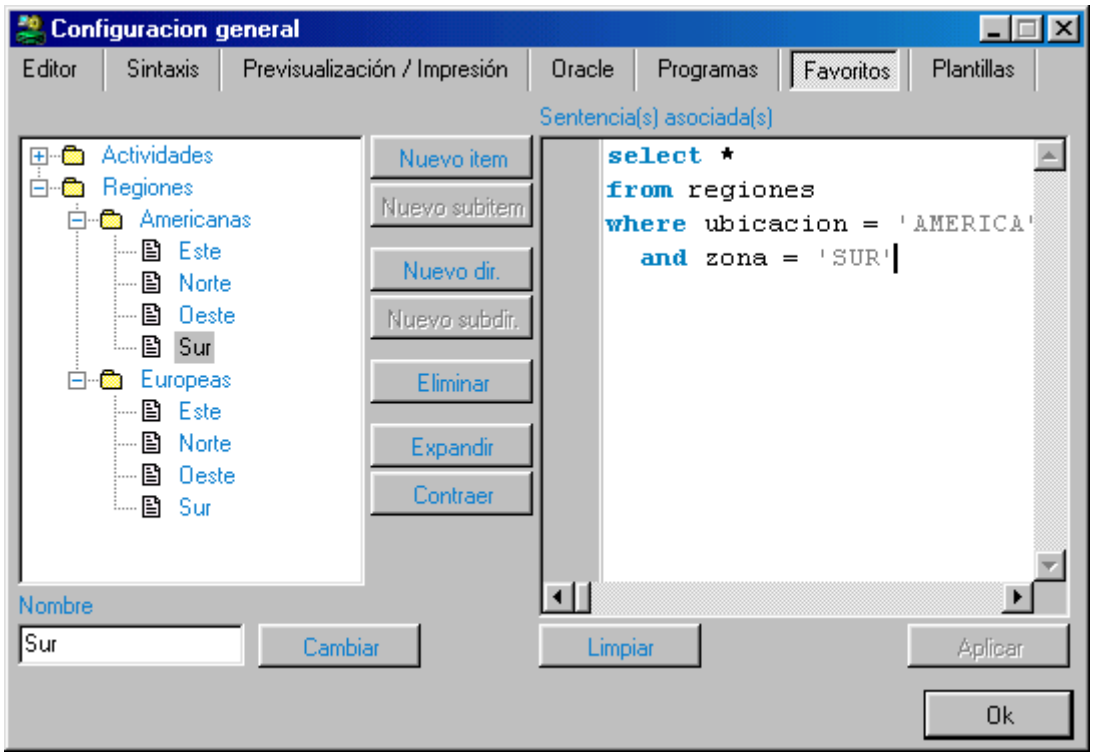

Figura 7.57: Editor de favoritos

El aspecto del gestor de favoritos, es el que se muestra en la Figura 7.57, donde se pueden crear items de sentencias y subitems (incluidos dentro de directorios o categorías), al igual que directorios y subdirectorios.

Los items representan elementos finales, es decir: consultas, sentencias de inserción, bloques PL/SQL, en general, cualquier código de usuario. El item que tratamos se muestra en la parte derecha de la ventana. Por otro lado las categorías (directorios o carpetas), son contenedores de items, que se utilizan para clasificar y organizarlos por categorías y se muestran a la izquierda de la ventana. Internamente se almacenarán como directorios dentro de Windows, conteniendo en su interior los items correspondientes.

En el centro de la ventana hay una lista de botones:

El primer botón (*Nuevo item*), crea un nuevo elemento en el mismo nivel donde se encuentre el elemento seleccionado en el árbol de la izquierda. Así si queremos crear un nuevo item en el nivel más interno del árbol, bastará con seleccionar la categoría *"Regiones"* o la categoría *"Actividades"* indistintamente y pulsar sobre este botón. Después tendremos que agregar el nombre que recibirá ese item, en la parte inferior donde se indica.

El segundo botón (*Nuevo subitem*), funciona de forma idéntica al anterior salvo en que éste crea un nuevo item dentro de la categoría que se haya seleccionado en el árbol de la izquierda. Así si queremos crear una nueva consulta en las regiones americanas que sean todas las regiones, bastará con seleccionar la carpeta Americanas que está dentro de Regiones y pulsar el botón nuevo subitem y posteriormente escribir la consulta asociada en el editor que encontramos en la parte derecha.

El tercer botón (*Nuevo dir.*), funciona de forma muy similar al primer botón, salvo que en lugar de crear un item, crea una nueva categoría para organizar en su interior los items. Después de crearla, se debe editar el nombre de ésta utilizando el cuadro de edición etiquetado con *"Nombre"*.

El cuarto botón (*Nuevo subdir.*), funciona de forma similar al tercer botón, salvo que crea una nueva categoría dentro de la que se tenga seleccionada en el árbol de la izquierda.

El quinto botón (*Eliminar*), simplemente elimina el elemento seleccionado, sea una categoría o un item. Hay que comentar que si el elemento es una categoría, se pide confirmación al usuario puesto que se eliminarán también todos los posibles items y subcategorías contenidos en su interior. También se puede pulsar la tecla de suprimir (SUPR) y tendrá el mismo efecto.

El sexto botón (*Expandir*) y el séptimo (*Contraer*), expanden y contraen todo el árbol respectivamente. Igualmente se dispone de accesos de teclado, simplemente pulsando las teclas más (+) y menos (-) del teclado numérico respectivamente.

El botón *Aplicar*, de la parte inferior derecha guarda los cambios realizados en el editor dentro del item que se encuentre el usuario editando. Si se ha editado la sentencia y no se ha pulsado este botón, los cambios se perderán.

El botón *Limpiar*, limpia el editor, borrando así todo lo que éste contenga sin pedir ninguna confirmación.

El botón *Cambiar*, que está más a la izquierda se emplea para cambiar el nombre de un item o categoría que ya existe.

Hay que matizar que no es posible mover items de forma cómoda simplemente arrastrando y soltando, por el contrario, se puede hacer pero bajo Windows abriendo un explorador en el directorio donde se deseen mover los elementos. Otro comentario es que se puede en cualquier momento editar el contenido de un item, simplemente seleccionándolo y modificándolo en el editor que se encuentra en la derecha.

7.11.7 Configuración de las plantillas de autocompletado por código

El aspecto de esta pestaña de configuración es el mostrado en la Figura 7.58, donde podemos observar que se compone básicamente de dos partes. La parte superior, es una lista, donde encontraremos los elementos (claves y comentarios) de autocompletado. Para obtener una información más detallada sobre el autocompletado por código, se puede consultar la sección 7.6.5, donde se explica con todo detalle su funcionamiento.

La parte inferior, incluye el texto asociado al elemento de autocompletado.

Por último sólo queda comentar que en la parte superior derecha, se incluyen botones para la gestión de la lista de autocompletado por código. El botón *Añadir*, añade una nueva cabecera, esto es, un nuevo elemento a las plantillas de autocompletado por código. Para ello, se utiliza el formulario mostrado en la Figura 7.59, en el cual debemos indicar simplemente el nombre (clave) y opcionalmente un comentario. Una vez que hayamos añadido el elemento, sólo nos quedará rellenar el código asociado a ese elemento. El botón de *Editar*, se utiliza para editar las cabeceras, es decir, el nombre (o clave) y la descripción. El código asociado, se puede editar de forma directa simplemente escribiendo el código que se desee en la parte inferior. Para finalizar, también disponemos de un botón *Eliminar*, que lo que hace es borrar una cabecera, junto con su código asociado.

En la parte de código, hay un carácter especial, que es la barra vertical (*"|"*), que representa el cursor del editor, es decir, el texto al sustituirse, dejará el cursor justo donde está ese carácter. Si hay varias barras, como por ejemplo, el caso de utilizar el operador de concatenación de SQL (*"| |"*), habría que incluir tres barras para la primera aparición, puesto que la primera será donde se deje el cursor de teclado. Otro caso similar es cuando se quiere utilizar ese carácter, en cuyo caso, se tendrá que incluir el carácter dos veces, porque la primera aparición será para el cursor.

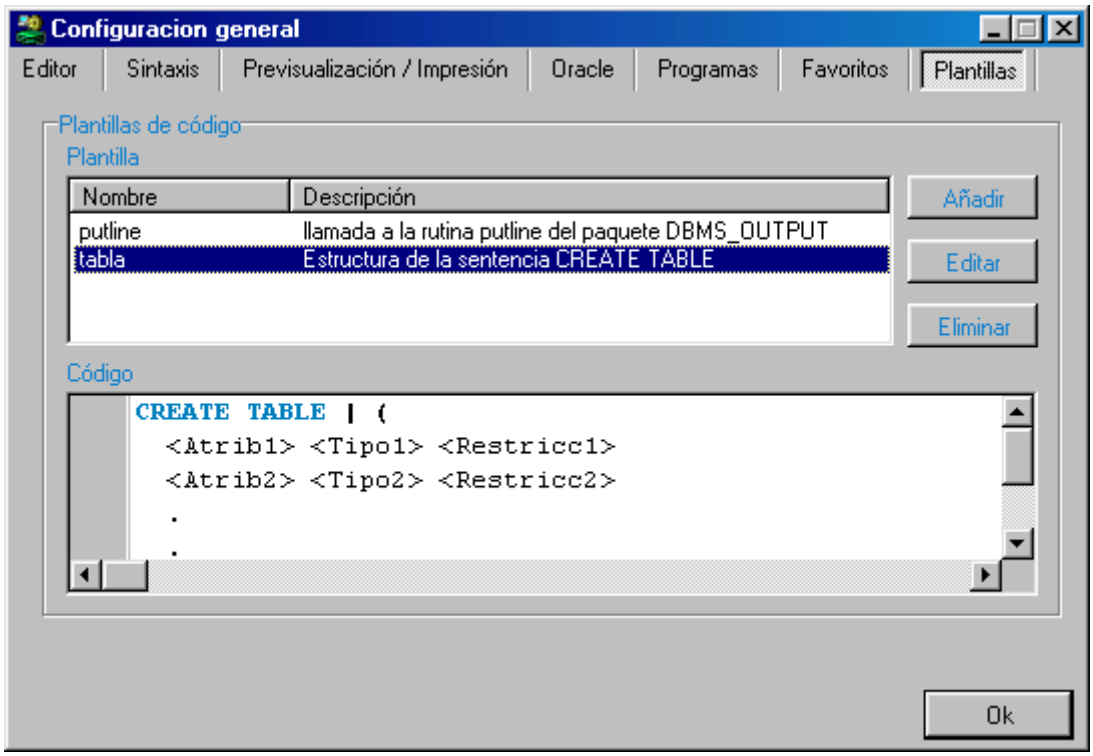

Figura 7.58: Pestaña *Plantillas* de las opciones de configuración

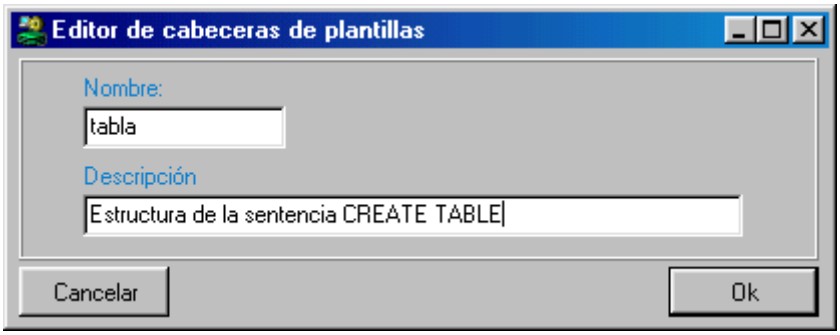

Figura 7.59: Aspecto del formulario de gestión de elementos de autocompletado

El formulario de la Figura 7.58 no dispone de un botón para cancelar la operación, puesto que los cambios que el usuario va haciendo se van guardando automáticamente. La única excepción es el último cambio, que para que sea guardado, hay que pulsar el botón *Ok* o bien cerrar el formulario si se desea descartar el último cambio.

7.12 Anécdotas

A continuación expondremos las respuestas a algunas preguntas que podrían ser formuladas por cualquier usuario *curioso*. Esta sección ya no trata sobre aspectos funcionales de la aplicación, sino simplemente curiosidades.

7.12.1 ¿Por qué comienza con la versión 3?

No, *Medusa*, al igual que la mayoría de los programas, no comienzan por una versión tres, sino que ha comenzado siendo un prototipo, que fue evolucionando poco a poco, adquiriendo nuevas funcionalidades.

La primera versión de *Medusa* que fue publicada y que vio la luz pública, fue la versión uno, la cual estaba implementada con los componentes ODAC (*Oracle Direct Access Components*).

Medusa nació en principio bajo formato SDI (*Single Document Interface*), donde únicamente se podía tener una única sesión por cada instancia del programa. El programa fue ideado como un trabajo para participar en el primer concurso de programación que propuso el profesor José Galindo de la asignatura Administración de bases de datos. Afortunadamente me convertí en uno de los ganadores, concretamente en la categoría de programación de un cliente de Oracle y esto fue lo que me ofreció la posibilidad de que *Medusa* se convirtiese en mi proyecto de fin de carrera. Simplemente así comenzó todo, fue una gran oportunidad que no dudé en aprovechar.

7.12.2 ¿Por qué se llama *Medusa*?

El nombre quizás no fue el más apropiado, aunque a mi me gustó mucho en la época en la que iba a presentarme al concurso.

El nombre de *Medusa* no posee ninguna relación con la mitología griega, como muchas personas pueden pensar. Este nombre, simplemente, surgió de una canción del grupo llamado *Antrax* que me gustaba en aquel entonces.

7.12.3 ¿Cuál es la composición que se escucha en la ventana *"acerca de..."*?

La composición que se reproduce en la ventana de *"acerca de..."*, es una versión reducida de *The Ecstasy Of Gold*, de *Enio Moricone*.

Conclusiones y líneas futuras

Desde que nacemos, la vida es un largo camino, por el cual siempre vamos aprendiendo y adquiriendo nuevas experiencias. Medusa, sin duda alguna, ha sido una experiencia inolvidable, donde se han aprendido muchas cosas.

Esperamos que Medusa pueda ser el punto de partida para otros desarrollos similares, de forma que esto no sea una despedida definitiva, sino simplemente un "hasta pronto"...

Conclusiones

En el presente proyecto fin de carrera hemos realizado una aplicación para el acceso a los datos y metadatos de una base de datos Oracle, con una interfaz amigable a nivel de usuario para facilitar todo el trabajo.

Esta aplicación se ha implementado con el lenguaje de programación Delphi. En los primeros capítulos, se ha explicado porqué se ha escogido este lenguaje de programación.

Tras el estudio de nuestros requerimientos consideramos que el Sistema Gestor de Bases de Datos (SGBD) a escoger debía cumplir los siguientes aspectos:

- Ser capaz de gestionar de forma eficiente grandes cantidades de información.
- Realizar gran cantidad de operaciones.

Por estos motivos, hemos elegido como gestor de base de datos Oracle. Igualmente, en los primeros capítulos, se ha explicado porqué se ha escogido este Sistema Gestor de Bases de Datos.

A lo largo del desarrollo de *Medusa*, hemos aprendido numerosos aspectos, como son por ejemplo cómo implementar motores de autocompletado, desde código y listas visuales, lo cual facilita mucho al usuario la interacción con el programa y la rapidez a la hora de programar, al igual que reduce los errores.

También hemos aprendido la importancia del diccionario de datos de Oracle y lo desconocido que es, por lo cual, hemos facilitado el acceso a los datos de éste, integrando un gran conjunto de consultas predefinidas en los menús de *Medusa* para familiarizar al usuario con éste y además, ayudar a saber el significado de cada una de las columnas de las consultas, comentando para que se utiliza la consulta y qué es lo que representan las columnas de una determinada consulta.

Otro aspecto importante a recalcar, es que se han construido interfaces visuales de ayuda al usuario para la construcción de objetos de forma grafica, haciendo más cómoda la labor. Se han incluido interfaces para facilitar la creación de procedimientos, funciones, disparadores (*triggers*),

paquetes, bloques de Java, Secuencias, Sinónimos, etc. Al igual que también se ha facilitado la construcción de formatos de fechas de forma gráfica e interactiva.

Con el objetivo de facilitar también al usuario, y dotar de una visión global de la base de datos, (conjunto de tablas que la conforman, sus atributos, restricciones, tipos, relaciones, etc.) se ha implementado una opción para reconstruir la sentencia `CREATE TABLE` a partir de una tabla de entrada. También se ha incluido una opción para que a partir de un conjunto de tablas se genere un listado con los atributos de éstas y sus relaciones a otras tablas, teniendo la posibilidad de incluir las tablas referenciadas por algunas de las tablas del conjunto (para implementar esta opción, hay que recorrer el posible grafo que se forma, teniendo en cuenta que el grafo puede contener ciclos, y evitar que la aplicación se quede en un bucle infinito realizando varias veces el recorrido por las mismas relaciones).

Para finalizar, sólo queda decir que la gran finalidad de *Medusa*, ha sido facilitar la interacción con el Sistema Gestor de Bases de Datos (SGBD), dotando también de algunas funciones extras hechas a medida con el mismo objetivo.

Comparativa entre Medusa y otros programas clientes de Oracle comerciales

Con el objetivo de tener una visión global de lo que ha sido el desarrollo, se ha incluido una tabla reducida, a modo comparativo, para observar las características de los principales programas que se han tomado como ejemplo para realizar *Medusa*.

	Ora-Tools	PL/SQL Developer	SQL Plus worksheet	SQL Navigator	Medusa
Fácil edición y visualización de datos en una sola ventana	Sí	Sí	Sí	Sí	Sí
Realzado de sintaxis	Sí	Sí	No	Sí	Sí
Exportación a múltiples formatos	No	Sólo tabla	No	No	Sí
Buffer de texto circular (*buffer* que contiene las últimas sentencias)	No	No	Fijo (50)	No	Config.
Fácil acceso al diccionario de datos desde menú	No	No	No	No	Sí
Lista configurable de programas	No	No	No	No	Sí
Visión global de la base de datos, incluyendo las tablas, atributos, relaciones, etc.	No	No	No	No	Sí
Construcción visual de objetos de la base de datos	No	Parcial	No	Parcial	Parcial
Edición simultánea de diversas órdenes (interfaz *MDI*)	No	Si	No	Si	Si
Autocompletado por código	No	No	No	No	Sí
Lista de autocompletado visual	Sí	No	No	No	Sí
Depurador integrado para programación de bloques PL/SQL	Simple	Completo	No	Sí	No
Posibilidades de configuración	Baja	Alta	Baja	Media	Media
Gestión de proyectos	No	Sí	No	No	No
Hecho en Delphi	Sí	Sí	No	Sí	Sí
Menú de favoritos	No	No	No	No	Sí
Posibilidad de grabar macros	No	Sí	No	No	No
Posibilidad de configurar los accesos de teclado	No	Sí	No	Sí	No
Edición directa de los resultados obtenidos	No	Sí	No	Sí	No
Exportación de objetos	No	Sí	No	No	No

	Ora-Tools	PL/SQL Developer	SQL Plus worksheet	SQL Navigator	Medusa
Asistente de consultas integrado de forma visual	No	Sí	No	Sí	No
Asistente de sentencias de inserción, modificación y eliminación	No	No	No	Sí	No
Documentación de Oracle integrada	No	Sí	No	No	Sí
Generación de informes de forma visual	No	Sí	No	No	No
Posibilidad de crear marcas de línea (*bookmarks*)	Sí	Sí	No	Sí	Sí
Posibilidades de configuración de la rejilla de resultados	Baja	Muy alta	Ninguna	Media	Baja
Entorno MDI	No	Sí	No	Sí	Sí
Planificación gráfica	No	No	No	Sí	No
Facilidad, y comodidad de utilización del programa (escala 1 al 10, 1 representa poco intuitivo, 10 representa muy intuitivo)	9	7	1	6	8
Versión comercial (share significa shareware)	Share 30 dias (49$)	Pago (150$)	Pago	Pago (795$)	Gratis
(*) Multiplataforma	No	No	No	No	No
(*) Incluye entorno de generación de tablas bajo modelo Entidad / Relación	No	No	No	No	No
(*) Posibilidad de expansión mediante plugins de usuario	No	No	No	No	No
(*) Autocompletado paramétrico	No	No	No	No	No

Tabla comparativa de los programas tomados como ejemplo

(*): Se explican a continuación, dentro del epígrafe "Tendencias futuras".

Tendencias futuras

Desde siempre, ha sido muy importante el almacenamiento de datos, su recuperación, mantenimiento y relaciones. Oracle, siempre ha sido pionero en este campo desde hace ya muchos años. No por nada es una de las compañías con más potencial económico del mundo. Dada la importancia de los datos y de tener un sistema completo e integrado, que interactúe con éstos, *Medusa* puede crecer en numerosos aspectos, entre los cuales se proponen las siguientes ideas.

- Portabilidad de *Medusa* a otros sistemas operativos. Sería muy útil portar *Medusa* a otros sistemas operativos como *Linux* por ejemplo, para cubrir un mayor abanico de necesidades, aprovechando que existe una versión de Oracle para *Linux*.
- Integrar en *Medusa* nuevas tecnologías como *XML* (eXtensible Markup Language, Lenguaje eXtensible de Marcas). Una opción interesante en el desarrollo, sería integrar XML dentro de *Medusa*, de forma que se puedan generar documentos XML que posteriormente mediante una aplicación XSL se puedan convertir a otros formatos, integrar en la web, etc.
- Exportar a otros formatos. Una posible mejora en el programa, sería añadir más formatos de exportación como por ejemplo *Latex, Word, Excel,* así como formatos gráficos (bmp, jpeg...), u otros formatos que puedan ser útiles para integrar consultas y resultados en documentos, para facilitar el trabajo.
- Implementar un entorno completo de modelado Entidad / Relación. Esta opción aunque aspire a muy altas metas, podría ser muy interesante, integrar un sistema de modelado Entidad / Relación, para poder construir tablas de forma visual y realizar un mantenimiento de la base de datos de una forma amena, utilizando las últimas tecnologías de la programación visual.
- Implementar un interfaz modular. Igualmente, esta opción podría llegar a ser compleja de implementar, pero sería muy útil, tener el programa principal y un conjunto de módulos (*plugins*) que se puedan integrar en *Medusa* de forma natural para formar una aplicación mayor en cuanto a prestaciones.
- Agregar motor de autocompletado paramétrico. Podría ser muy útil dotar a *Medusa*, de la opción de autocompletado paramétrico (al comenzar a escribir el nombre de la ruti-

na, cuando se abre el paréntesis, se muestran en forma de *hints* los parámetros de la rutina), lo cual puede ayudar bastante a la hora de programar.

- Incluir posibilidades de grabación de macros. Sería una buena idea incluir opciones para grabar, reproducir y gestionar macros, que puedan facilitar el trabajo del usuario.
- Dotar a *Medusa* de control de transacciones y edición de datos. Una funcionalidad muy importante es el control de transacciones, que junto con la edición de datos de forma visual, pueden ayudar al usuario para no tener que escribir sentencias `UPDATE` y lo pueda hacer de forma cómoda y directa.
- Integración de un depurador PL/SQL y Java. Esta opción podría ser muy útil para usuarios programadores en PL/SQL.
- Integración de ayudantes para construcción de sentencias del DML de SQL, como `SELECT`, `UPDATE`, `INSERT` y `DELETE`. Una de las grandes ventajas que posee PL/SQL Builder y SQL Navigator son los asistentes para construcción de sentencias, los cuales pueden ser muy útiles para los usuarios menos experimentados o para operaciones muy complejas.
- Configuración de accesos de teclado. Podría ser útil para los usuarios que utilicen *Medusa* de forma habitual, ofrecerles la posibilidad de configurar los accesos rápidos de teclado a las distintas opciones del programa, como también a funcionalidades de edición.
- Generación de informes. Una idea interesante sería incluir la posibilidad de incluir la generación de informes (sobre consultas empotradas referentes a administración y otros propósitos) de forma gráfica y tener las opciones de imprimir, exportar a otros formatos, etc.
- Posibilidades de exportación e importación de objetos. Esta opción podría ser muy interesante, el poder importar objetos (procedimientos, funciones, tablas...) de otros usuarios o exportar objetos para poder compartirlos con otros usuarios.
- Incluir planificador de proyectos gráfico. Podría ser interesante incluir un planificador gráfico para los usuarios avanzados de Oracle.
- Agregar opciones de configuración de sintaxis de Java. Una opción útil para programadores de Bases de Datos sería incluir la posibilidad de configuración de colores del realzado de sintaxis de Java, e incluir palabras reservadas de Oracle.

- Agregar opciones de configuración de fuentes. Con el objetivo de que *Medusa* se lo más configurable posible, puede ser una buena idea incluir opciones para personalizar el tipo de fuente, el tamaño de letra... tanto del editor, como de la rejilla de resultados.
- Realizar una precarga de datos y objetos de la Base de Datos. Podría ser interesante realizar una precarga de los objetos de la Base de Datos, para que puedan estar disponibles para el usuario simplemente haciendo Drag & Drop.
- Manipulación de los resultados de una consulta. Una buena idea, sería incluir opciones para la manipulación de forma visual de los resultados de una consulta, como por ejemplo, ordenarlos por columnas, efectuar consultas sobre ese resultado, resaltar ciertas filas, modificar datos...
- Añadir opciones de administración. Una idea atractiva sería incluir opciones de administración de forma gráfica, para facilitar el trabajo de administradores dentro de *Medusa*.
- Agregar opciones para permitir al usuario la selección de el pre-procesado de las listas de autocompletado. Realizar un procesado de todos los objetos de la base de datos accesibles por el usuario (procedimientos, funciones, disparadores...) y sus argumentos si procede para brindarle al usuario la posibilidad de tener las listas completamente actualizadas.
- Complementar la opción de "Estructura de la BD". El objetivo es poder seleccionar la fuente de datos, es decir, poder seleccionar si las vistas son con prefijo `USER_` o con prefijo `ALL_`, de forma que se tenga más versatilidad. Esta funcionalidad no se ha implementado porque se observó que se complicaba el procesado en el caso de que varios usuarios tuviesen tablas con el mismo nombre.
- Complementar la opción de "Reconstruir sentencia `CREATE TABLE`". De forma similar al caso anterior se puede ofrecer la posibilidad de seleccionar la fuente de datos para la reconstrucción.
- Incluir opciones de backup. Se trata de dotar a *Medusa* de facilidades para realizar backup's y recovery's. Se planteó la posibilidad de incluir un interfaz que facilite la construcción de sentencias para utilizar las herramientas `IMP` y `EXP`, pero no se pudo llevar a cabo con las versiones de la librería por las limitaciones éstas.
- Completar la opción de "Reconstruir sentencia `CREATE TABLE`". De forma que se pueda tener una lista con múltiples tablas de las que se desea obtener la sentencia DDL, similar a la opción de "Estructura de la BD".

- Incluir un interfaz gráfico para la sentencia `CREATE TABLE` de SQL. Esta opción puede ser muy interesante, para facilitar la construcción de tablas de forma fácil.
- Incluir opciones para que el usuario pueda configurar las combinaciones de teclas del editor (*keystrokes*).
- Incluir opciones para la configuración de las listas de autocompletado.
- Incluir opciones para la configuración del de autocompletado por código.
- Incluir opciones para la configuración del realzado de la línea actual (color de fondo, color de selección...).

Referencias

Las referencias, son la ayuda y la base de todo, sin ellas muchos desarrollos serían impensables. Para abordar un proyecto de investigación o desarrollo, es necesario buscar e indagar, sobre trabajos de otras personas, para intentar mejorar o innovar y obtener resultados satisfactorios.

Referencias bibliográficas

En esta sección, se incluyen las referencias bibliográficas consultadas a lo largo del proyecto, para obtener ideas, o a modo de consulta para obtener información en general.

Oracle

[ABB02**]** Oracle 9i: Guía de aprendizaje.
Michael Abbey, Mike Carey y Ian Abamson.
Oracle Press, 2002.

[ELM02**]** Fundamentos de sistemas de bases de datos (3ª edición)
Armes A. Elmasri y Shamkant B. Navathe.
Addison Wesley, 2002.

[LON02**]** Oracle 9i: Manual del administrador.
Kevin Loney y Marlene Theriault.
Oracle Press, 2002.

[ORA02**]** Manual de referencia de usuario de Oracle
Oracle Corporation.
Oracle Corporation, 2002.

[OWE96**]** Building intelligent Databases with Oracle PL/SQL, triggers and stored procedures.
Kevin T. Owens.
Prentice Hall, 1996.

[SQL99**]** Manual de referencia de usuario de SQL*Plus
Oracle Corporation.
Oracle Corporation, 1999.

[URM98**]** Oracle 8: Programación PL/SQL.
Oracle Press.
Osborne, 1998

Delphi

[CHA98**]** Guía práctica para usuarios Delphi 4.
Francisco Charte Ojeda.
Anaya Multimedia, 1998.

[CHA99**]** Guía práctica para usuarios Delphi 5.
Francisco Charte Ojeda.
Anaya Multimedia, 1999.

[TEI00**]** Guía de desarrollo Delphi 5.
Steve Teixeira y Xavier Pacheco.
Prentice Hall, 2000.

InstallShield Express – Edición limitada para Delphi

[INS01**]** Manual de usuario de InstallShield Express – Edición limitada para Delphi.
InstallShield, 2001.

Shalom Help Maker

[SHA03**]** Manual de usuario de Shalom Help Maker.
Shalom, 2003.

Recursos de Internet

La red de Internet, es una de las mayores fuentes de información actualmente. Todo lo que no se consigue en los libros, generalmente se consigue en la red. Aquí se incluyen algunos enlaces que se han consultado. Hay recursos a los cuales he logrado llegar, a través de buscadores y que ya no recuerdo los enlaces.

Oracle

[W3ARA] Manuales de ayuda de la utilización de las librerías Direct Oracle Access, junto con el programa PL/SQL Developer.
http://www.allroundautomations.com/

[W3DBA] Algunos recursos.
http://www.bikle.com/TThier/DBAtips/sql/

[W3ORA] Sitio web de Oracle
www.oracle.com

[W3QUE] Sitio web de Quest Software
http://www.quest.com/

[W3CRL] Sitio web de CrLab (OraTools)
www.crlab.com/

Delphi

[W3MAR] La cara oculta de Delphi 4 de Ian Marteens, Intuitive Sight online.
http://www.latiumsoftware.com/descarga/lcod4.php

[W3CAN] Mastering Delphi 6 de Marco Cantú, online.
http://www.sybex.com/

[W3TIN] Algunos recursos
http://tinn.sourceforge.net

[W3SYN] Manuales de ayuda de la utilización de las librerías SynEdit, junto con Algunos recursos
http://synedit.sourceforge.net

[W3WOU] Algunos recursos
http://home.planet.nl/~woute596

Shalom Help Maker

[W3SHA] Se puede descargar Shalom Help Maker
http://www.danish-shareware.dk/soft/shelpm/index.html