ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

MEDUSA 2: UN CLIENTE AVANZADO DE ORACLE CON OPCIONES DE ADMINISTRACIÓN

Realizado por

FRANCISCO JAVIER PULIDO ARREBOLA

Dirigido por

JOSÉ GALINDO GÓMEZ

Departamento

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA

Málaga, Septiembre de 2005

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 2:

UN CLIENTE AVANZADO DE ORACLE

CON OPCIONES DE ADMINISTRACIÓN

Realizado por D. Francisco Javier Pulido Arrebola

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__

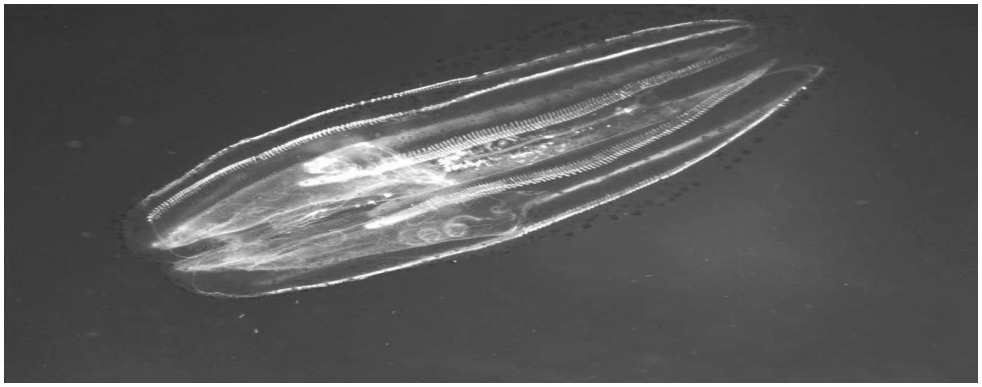
El Presidente El Secretario El Vocal

Fdo. _____ Fdo. _____ Fdo. _____

Este proyecto no sería lo mismo sin la ayuda de esas personas que nunca han dejado de estar ahí. Gracias a José Luis, Jesús, Pedrito, David, Javi, Ale… por el día a día, a Inma, Rocío y Ana porque siempre puedo contar con ellas y a Albert, que además de hacer la primera versión me ayudó en el camino hacia ésta segunda. Gracias a José Galindo, por sus sabios consejos y su confianza en mí. Y sobre todo a mi familia, que llevan toda mi vida apoyándome y sé que lo seguirán haciendo.

Quizá nunca sepa agradecéroslo, pero lo intentaré.

Índice



1

Introducción

En el mundo del cine se utiliza con normalidad la expresión "Segundas partes nunca fueron buenas". Y en Medusa 2, ¿estaremos ante una segunda parte o ante una mejora de la primera? ¿Habremos conseguido que esta segunda parte sí sea realmente buena?

Simplemente, sigan leyendo y juzguen ustedes mismos.

.

1.1 Medusa, primera versión

La primera parte de Medusa nació hará ya dos años, por Septiembre de 2003, como continuación de un concurso de programación organizado por el profesor José Galindo. Una primera versión, mucho más simple que el PFC "*Medusa, un cliente avanzado de Oracle*", que lo podremos descargar en [W3MED], se hizo con el primer premio en la categoría de implementación de un cliente de Oracle. Gracias a esto, el programa fue creciendo tanto en funcionalidades como en contenido hasta la entrega final del proyecto fin de carrera.

Desde mis primeras conversaciones, encontré este proyecto como una experiencia enriquecedora y estimulante, que daría paso a un periodo de pruebas, investigación y diseño para acabar con esta segunda versión.

El reto a superar se presentaba lejano desde un principio, Medusa ya era un programa con cierta complejidad, que usaba componentes adicionales a los de la VCL de Delphi, y que como cualquier programa tenía sus puntos débiles y sus complicaciones de expansión. El primer paso fue la instalación y prueba, comprender su funcionamiento y buscar ideas para posibles mejoras.

Cuando quedó comprendido el comportamiento y características del software con el que tendríamos que enfrentarnos llegó una fase de comprensión del código. Con esto nos referimos no sólo a los detalles de implementación, sino también a las características y funciones de los componentes utilizados, pues sobre ellos deberíamos de construir cualquier progreso.

Una vez decididas las características a añadir y modificar en la nueva versión, realizamos una búsqueda de nuevos componentes, que nos ayudarían en el proceso de desarrollo y le darían una mayor calidad al resultado. Además de las distintas funciones que puedan aportar se evaluó también la ayuda que poseían, puesto que unos componentes complejos y sin ayuda pueden retrasar y dificultar el proyecto más que agilizarlo.

Empezamos el desarrollo y nos fuimos encontrando con las dificultades típicas que debemos afrontar cuando estamos ante componentes de programación completamente nuevos para nosotros. A medida que el desarrollo fue avanzando también se procuró ir depurando todos aquellos fallos de los que adolecía el programa base e ir cambiando todo lo que fuera necesario para que el programa pudiera expandir sus características.

Algunos de los avances deseados chocaban con una implementación que no estaba preparada para tales particularidades, con lo que debíamos rehacer el diseño para que manteniendo unas funciones idénticas, el desarrollo diera cabida a estas nuevas evoluciones. Las distintas mejoras introducidas así como algunos de los errores subsanados los describiremos mas adelante, pero antes de nada, vamos a hacer una pequeña descripción del que ha sido la base de nuestro trabajo, además de los objetivos principales que deseábamos obtener:

1.1.1 Características de la primera versión

Medusa pretendía desarrollar una aplicación que intentara mejorar algunos de los aspectos del programa de comunicación SQL*Plus de Oracle.

En síntesis sus principales objetivos fueron:

- Facilitar la tarea de programación, consulta y modificación de la BD.
- Simplificar operaciones típicas en la gestión de la BD.
- Ofrecer interfaces gráficos de creación de objetos.

Medusa se convierte en un completo cliente de Oracle. Posee un 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 incluyó dentro de la misma ventana donde se encuentra el editor, una rejilla donde se obtendrán los resultados. Medusa podía ejecutar cualquier tipo de sentencia de Oracle (consultas, sentencias DDL, inserciones de datos, actualizaciones, eliminación de datos, bloques PL/SQL, concesión de permisos, etc). Igualmente, se disponía de una pestaña que nos informaba de los posibles errores que podían darse en las sentencias, marcándolos en caso de producirse, en el margen izquierdo del editor. Otro aspecto muy

importante era la creación de formularios que ofrecían facilidades para la creación de objetos de la base de datos, como procedimientos, vistas, perfiles o usuarios, entre otros. También se incluyeron facilidades para crear, acceder y modificar comentarios de tablas y columnas en el diccionario de datos, así como facilidades para la manipulación de formatos de fechas y otras tareas habituales.

Para obtener más información sobre la primera versión de Medusa consultar [MED03].

1.1.2 Detalles de implementación

Puesto que en la aplicación final no podemos ver los detalles de implementación, vamos a hacer un pequeño análisis de qué nos encontramos la primera vez que vimos el código de esta primera versión:

Medusa constaba de unos 30 formularios con alrededor de unas 12.000 líneas de código. La compilación estaba bien definida en procedimientos y funciones, aunque ciertas veces se utilizaban algunos procedimientos para demasiadas diversas funciones, lo que acarreaba en ocasiones una difícil legibilidad y seguimiento de la ejecución, así como procedimientos demasiado extensos.
Otro *handicap* a tener en cuenta era la desorganización del código, lo que hacía que la comprensión para otro programador ajeno al desarrollo se dificultara.

El código de Medusa poseía al menos 20 procedimientos declarados y no utilizados, procedimientos con todo su código entre comentarios o procedimientos cuya funcionalidad no era necesaria. Se procedió a limpiar el código y organizarlo, condiciones casi necesarias para su correcta comprensión, con lo que el código bajó, para hacernos una idea, alrededor de las 1.000 líneas.

Además de esto, pudimos observar como la estabilidad de la aplicación se conseguía en ciertos casos, gracias a una reducción de la libertad de la que disponía el usuario (por ejemplo, los formularios de construcción visual se mostraban de forma modal, con lo que se

evitaba que el usuario pudiera realizar cualquier otra función paralela a ésta), cuestión que sin duda deberíamos de modificar.

1.2 Medusa 2: Objetivos principales

El primer objetivo será la realización de un análisis en profundidad sobre los requerimientos del sistema. Primero, se intentarán arreglar aquellos fallos que surjan en las pruebas realizadas, en las que se testeará el programa como si del usuario final se tratase. En la *sección 5.3* describiremos con más detalle algunos de estos fallos. También se dotará al programa de todas aquellas mejoras que vayan creyéndose necesarias, ya sean detalles pocos significativos como importantes evoluciones. Y una vez con conocimiento del ámbito en el que nos movemos, así como de las carencias específicas encontradas, construiremos un software que cubra con mayor eficacia las necesidades de cualquier usuario.

Como base, vamos a proporcionar a nuestro cliente de Oracle un nuevo interfaz gráfico, más configurable y atractivo, mayor flexibilidad y potencia en la visualización y en el tratamiento de los resultados devueltos, y un explorador de objetos con el que acceder cómodamente a nuestros elementos de trabajo (tablas, vistas, procedimientos, funciones…).

Continuando con el trabajo de la primera versión se aumentarán algunas de las características que ya poseía la primera versión. Se incluirán más consultas predefinidas al diccionario de datos, algunos constructores visuales de objetos más y un manejo más intuitivo y sencillo de las plantillas de texto.

Se dará un completo soporte para el control de transacciones, con posibilidades de definir puntos de guarda, realizar rollback´s sobre dichos puntos o sobre la transacción completa, etc.

Puesto que uno de los objetivos de esta segunda versión concierne a los resultados devueltos por el SGBD, sería necesario un sistema de almacenamiento de éstos resultados devueltos. Para ello vamos a añadir más formatos de exportación, como Excel

o XML, además de un completísimo generador de informes, con el que también podremos guardar la información en un soporte no informático.

Por último de éstas características principales, vamos a añadir a nuestra aplicación el multi-lenguaje, con el que usuarios no hispanohablantes también podrán utilizar Medusa como su cliente de Oracle. *Medusa 2* estará disponible en [W3MED].

1.3 Trabajos relacionados

En esta sección nos dedicamos a ver las características principales de otros programas comerciales de administración y gestión de bases de datos Oracle. Los llamados entornos de desarrollo integrados (IDE, *Integrated Desktop Environment*) poseen muchas características atrayentes, y sobre todo la capacidad de realizar prácticamente cualquier proceso desde dentro, sin la necesidad de interactuar directamente con el SGBD. 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 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.

El precio de SQL Navigator oscila alrededor de los 800$ por una sola licencia de un año. Su web está disponible en **[**W3QUE**]**.

1.3.3 OraTools de CoreLab

OraTools es una herramienta poderosa y versátil suplementaria con las herramientas de depuración y administración que posee Oracle. Algunas de sus principales utilidades son OraDesigner, OraExplorer y PL/SQL Debugger.

El precio de OraTools para un solo desarrollador es de 49$, mientras que el precio para una empresa es de 585$. Su web está disponible en **[**W3CRL**]**.

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.

El precio de PL/SQL Developer dependerá del número de usuarios. Para un solo usuario el precio es de 180$, y mientras más usuarios lo utilicen mayor será el precio de la licencia, hasta los 6.000$ de una licencia sin límite de usuarios. Se puede descargar de **[**W3ARA**]**.

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 veremos las características y funciones generales de los sistemas gestores de bases de datos, en particular de Oracle, así como una comparación con los que fueron sus antecesores, los sistemas de archivos. Describiremos los distintos elementos que forman una base de datos Oracle y detallaremos como realizar paso a paso la instalación.

Capítulo 3. Delphi

Hablaremos de las características generales del lenguaje de programación Delphi, así como de la instalación de su compilador.

Capítulo 4. Componentes adicionales

En este capítulo conoceremos los componentes adicionales a la VCL de Delphi que hemos utilizado: el interfaz de acceso a Oracle, que proporciona la librería DOA, el reconocimiento sintáctico que nos brinda SynEdit, la rapidez en la construcción de los objetos de la BD, gracias a la ayuda de FlyTreeView, así como los múltiples componentes utilizados de Developer Express.

Capítulo 5. Desarrollo de *Medusa 2*

Explicaremos *Medusa* en profundidad. Para ello vamos a empezar detallando algunos de los componentes de programación que hemos utilizado, describiremos la estructura de Medusa, tanto la interna (formularios) como la externa (ficheros necesarios). Enumeraremos algunos de los fallos que han sido subsanados en nuestra versión y por último hablaremos de las ampliaciones, mejoras y nuevas funciones que presenta Medusa 2 con respecto a su antecesor.

Capítulo 6. Detalles finales: Ayuda, instalación y traducción

En este capítulo explicamos cómo hacer y cómo compilar los ficheros de ayuda para el usuario, que se incluyen en *Medusa*, la técnica seguida para realizar la instalación y los detalles que conciernen a la traducción de una aplicación.

Capítulo 7. Medusa 2: Manual de usuario

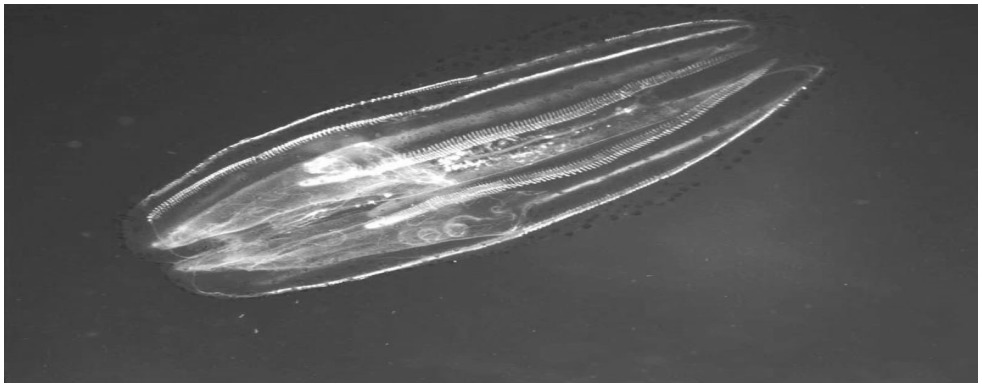
Explicamos detalladamente el manual de usuario de la aplicación, incluyendo cada uno de los menús y opciones que nos encontraremos dentro de *Medusa 2*, junto con algunos comentarios y trucos.

Conclusiones y líneas futuras

Concluimos esta memoria describiendo brevemente las conclusiones que han derivado de la realización de este PFC, un seguimiento de la obtención de los objetivos prefijados y la enumeración de un conjunto de ideas que pudieran servir de partida para siguientes versiones.

Referencias bibliográficas

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



2

Oracle

En 1977 se fundó Software Development Laboratories, que posteriormente tomaría el nombre de Oracle. Sus fundadores, Larry Ellison, Bob Miner y Ed Oates construyeron el primer sistema de gestión de bases de datos como producto comercial.

Desde entonces Oracle ha mantenido una posición líder en el mercado de las bases de datos relacionales y sus servicios han crecido mucho más allá de la gestión de bases de datos.

2.1 ¿Qué es Oracle?

En este capítulo, expondremos una descripción de Oracle, actualmente uno de los SGBDR (Sistemas Gestores de Bases de Datos Relacionales) más extendidos y potentes del mercado.

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 (*Structured Query Language*) para definir y manipular los datos (DDL y DML). Además, tiene un lenguaje 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**]**.

2.2 Sistemas gestores de bases de datos

Un **Sistema Gestor de Bases de Datos** (SGBD) consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a dichos datos. La colección de datos se denomina normalmente **base de datos** y contendrá toda la información relevante para una empresa. El objetivo principal de un SGBD es proporcionar una forma de almacenar y recuperar la información de una base de datos de manera que sea tanto *práctica* como *eficiente*.

Los SGBD se diseñan para gestionar grandes cantidades de información. La gestión de los datos implica tanto la definición de estructuras para almacenar la información como la provisión de mecanismos para la manipulación de la información. Además, los SGBD deben proporcionar la fiabilidad de la información almacenada, a pesar de las caídas del sistema o

los intentos de acceso sin autorización. Si los datos van a ser compartidos entre diversos usuarios, el sistema debe evitar posibles resultados anómalos.

2.2.1 SGBD frente a sistemas de archivos

Vamos a exponer un ejemplo que nos va a mostrar algunas de las características negativas de los sistemas de archivos que se corrigieron con la aparición de los SGBD, así como las funciones principales que debe de ofrecer todo SGBD.

Considérese parte de una empresa de cajas de ahorros que mantiene información acerca de todos los clientes y cuentas de ahorros. Una manera de mantener la información en un computador es almacenarla en archivos del sistema operativo. Para permitir a los usuarios manipular la información el sistema tiene un número de programas de aplicación que manipula los archivos, incluyendo:

- Un programa para efectuar cargos o abonos en una cuenta.
- Un programa para añadir una cuenta nueva.
- Un programa para calcular el saldo de una cuenta.
- Un programa para generar las operaciones mensuales.

Estos programas de aplicación serán escritos por programadores de sistemas en respuesta a las necesidades de la organización bancaria. Si las necesidades se incrementan, se añaden nuevos programas de aplicación al sistema.

Los registros permanentes son almacenados en varios archivos y se escriben diferentes programas de aplicación para extraer registros y para añadir registros a los archivos adecuados. Antes de la llegada de los SGBD, las organizaciones normalmente han almacenado la información usando tales sistemas. Mantener información de la organización en un sistema de procesamiento de archivos tiene una serie de inconvenientes importantes:

- **Redundancia e inconsistencia de datos.** Debido a que los archivos y programas de aplicación son creados por diferentes programadores en un largo período de tiempo,

los diversos archivos tienen probablemente diferentes formatos y los programas pueden estar escritos en diferentes lenguajes. Más aún, la misma información puede estar duplicada en diferentes lugares (archivos). Por ejemplo, la dirección y número de teléfono de un cliente particular puede aparecer en un archivo que contenga registros de cuentas de ahorros y en un archivo que contenga registros de una cuenta corriente. Esta redundancia conduce a un almacenamiento y coste de acceso más altos. Además, puede conducir a **inconsistencia de datos;** es decir, las diversas copias de los mismos datos pueden no coincidir. Por ejemplo, un cambio en la dirección del cliente puede estar reflejado en los registros de las cuentas de ahorro pero no estarlo en el resto del sistema.

- **Aislamiento de datos.** Debido a que los datos están dispersos en varios archivos, y los archivos pueden estar en diferentes formatos, es difícil escribir nuevos programas de aplicación para recuperar los datos apropiados.

- **Problemas de integridad.** Los valores de los datos almacenados en la base de datos deben satisfacer ciertos tipos de **restricciones de consistencia.** Por ejemplo, el saldo de una cuenta bancaria no puede nunca ser más bajo de una cantidad predeterminada (por ejemplo 25 €). Los desarrolladores hacen cumplir esas restricciones en el sistema añadiendo el código apropiado en los diversos programas de aplicación. Sin embargo, cuando se añaden nuevas restricciones, es difícil cambiar los programas para hacer que se cumplan. El problema es complicado cuando las restricciones implican diferentes elementos de datos de diferentes archivos.

- **Problemas de atomicidad.** Un sistema de un computador, como cualquier otro dispositivo mecánico o eléctrico, está sujeto a fallo. En muchas aplicaciones es crucial asegurar que, una vez que un fallo ha ocurrido y se ha detectado, los datos se restauran al estado de consistencia que existía antes del fallo. Consideremos un programa para transferir 50 € desde la cuenta A a la B. Si ocurre un fallo del sistema durante la ejecución del programa, es posible que los 50 € fueron eliminados de la cuenta A pero no abonados a la cuenta B, resultando un estado de la base de datos inconsistente. Claramente, es esencial para la consistencia de la base de datos que ambos, el abono y el cargo tengan lugar, o que ninguno tenga lugar. Es decir, la transferencia de fon-

dos debe ser *atómica*: ésta debe ocurrir en ellos por completo o no ocurrir en absoluto. Es difícil asegurar esta propiedad en un sistema de procesamiento de archivos convencional.

- **Anomalías en el acceso concurrente.** Conforme se ha ido mejorando el conjunto de ejecución de los sistemas y ha sido posible una respuesta en tiempo más rápida, muchos sistemas han ido permitiendo a múltiples usuarios actualizar los datos simultáneamente. En tales sistemas un entorno de interacción de actualizaciones concurrentes puede dar lugar a datos inconsistentes. Considérese una cuenta bancaria A, que contiene 500 €. Si dos clientes retiran fondos (por ejemplo 50 € y 100 € respectivamente) de la cuenta A en aproximadamente el mismo tiempo, el resultado de las ejecuciones concurrentes puede dejar la cuenta en un estado incorrecto (o inconsistente). Supongamos que los programas se ejecutan para cada retirada y escriben el resultado después. Si los dos programas funcionan concurrentemente, pueden leer ambos el valor 500 €, y escribir después 450 € y 400 €, respectivamente. Dependiendo de cuál escriba el último valor, la cuenta puede contener bien 450 € o bien 400 €, en lugar del valor correcto, 350 €. Para protegerse contra esta posibilidad, el sistema debe mantener alguna forma de supervisión. Sin embargo, ya que se puede acceder a los datos desde muchos programas de aplicación diferentes que no han sido previamente coordinados, la supervisión es difícil de proporcionar.

- **Problemas de seguridad.** No todos los usuarios de un sistema de bases de datos deberían poder acceder a todos los datos. Por ejemplo, en un sistema bancario, el personal de nóminas necesita ver sólo esa parte de la base de datos que tiene información acerca de varios empleados del banco. No necesitan acceder a la información acerca de las cuentas de clientes. Como los programas de aplicación se añaden al sistema de una forma *ad hoc*, es difícil garantizar tales restricciones de seguridad.

2.2.2 Funciones de los SGBD

Codd, el creador del modelo relacional, estableció una lista con los ocho servicios que debe ofrecer todo SGBD. Para más información sobre ellos consultar **[**W3COD**].**

- Un SGBD debe proporcionar a los usuarios la capacidad de almacenar datos en la base de datos, acceder a ellos y actualizarlos. Esta es la función fundamental de un SGBD y por supuesto, el SGBD debe ocultar al usuario la estructura física interna (la organización de los ficheros y las estructuras de almacenamiento).

- Un SGBD debe proporcionar un catálogo en el que se almacenen las descripciones de los datos y que sea accesible por los usuarios. Este catálogo es lo que se denomina diccionario de datos y contiene información que describe los datos de la base de datos (metadatos). Normalmente, un diccionario de datos almacena:
  - Nombre, tipo y tamaño de los datos.
  - Nombre de las relaciones entre los datos.
  - Restricciones de integridad sobre los datos.
  - Nombre de los usuarios autorizados a acceder a la base de datos.
  - Esquemas externo, conceptual e interno, y correspondencia entre los esquemas.
  - Estadísticas de utilización, tales como la frecuencia de las transacciones y el número de accesos realizados a los objetos de la base de datos.

Algunos de los beneficios que reporta el diccionario de datos son los siguientes:

- La información sobre los datos se puede almacenar de un modo centralizado. Esto ayuda a mantener el control sobre los datos, como un recurso que son.
- El significado de los datos se puede definir, lo que ayudará a los usuarios a entender el propósito de los mismos.
- La comunicación se simplifica ya que se almacena el significado exacto. El diccionario de datos también puede identificar al usuario o usuarios que poseen los datos o que los acceden.
- Las redundancias y las inconsistencias se pueden identificar más fácilmente ya que los datos están centralizados.
- Se puede tener un historial de los cambios realizados sobre la base de datos.
- El impacto que puede producir un cambio se puede determinar antes de que sea implementado, ya que el diccionario de datos mantiene información sobre cada tipo de dato, todas sus relaciones y todos sus usuarios.

- Se puede hacer respetar la seguridad.
- Se puede garantizar la integridad.
- Se puede proporcionar información para auditorias.

℘ Un SGBD debe proporcionar un mecanismo que garantice que todas las actualizaciones correspondientes a una determinada transacción se realicen, o que no se realice ninguna. Una *transacción* es un conjunto de acciones que cambian el contenido de la base de datos de un estado coherente a otro estado coherente. Una transacción en el sistema informático de la empresa inmobiliaria sería dar de alta a un empleado o eliminar un inmueble. Una transacción un poco más complicada sería eliminar un empleado y reasignar sus inmuebles a otro empleado. En este último caso hay que realizar varios cambios sobre la base de datos. Si la transacción falla durante su realización, por ejemplo porque falla el *hardware*, la base de datos quedará en un estado inconsistente. Algunos de los cambios se habrán hecho y otros no, por lo tanto, los cambios realizados deberán ser deshechos para devolver la base de datos a un estado consistente. En la sección 2.5 se explican las transacciones con más detalle.

℘ Un SGBD debe proporcionar un mecanismo que asegure que la base de datos se actualice correctamente cuando varios usuarios la están actualizando concurrentemente. Uno de los principales objetivos de los SGBD es el permitir que varios usuarios tengan acceso concurrente a los datos que comparten. El acceso concurrente es relativamente fácil de gestionar si todos los usuarios se dedican a leer datos, ya que no pueden interferir unos con otros. Sin embargo, cuando dos o más usuarios están accediendo a la base de datos y al menos uno de ellos está actualizando datos, pueden interferir de modo que se produzcan inconsistencias en la base de datos. El SGBD se debe encargar de que estas interferencias no se produzcan en el acceso simultáneo.

℘ Un SGBD debe proporcionar un mecanismo capaz de recuperar la base de datos en caso de que ocurra algún suceso que la dañe. Como se ha comentado antes, cuando el sistema falla en medio de una transacción, la base de datos se debe devolver a un estado consistente. Este fallo puede ser en algún dispositivo *hardware*

o un error del *software*, que hagan que el SGBD aborte, o puede ser a causa de que el usuario detecte un error durante la transacción y la aborte antes de que finalice. En todos estos casos, el SGBD debe proporcionar un mecanismo capaz de recuperar la base de datos llevándola a un estado consistente.

- Un SGBD debe proporcionar un mecanismo que garantice que sólo los usuarios autorizados pueden acceder a la base de datos. La protección debe ser contra accesos no autorizados, tanto intencionados como accidentales.

- Un SGBD debe ser capaz de integrarse con algún software de comunicación. Muchos usuarios acceden a la base de datos desde terminales. En ocasiones estos terminales se encuentran conectados directamente a la máquina sobre la que funciona el SGBD. En otras ocasiones los terminales están en lugares remotos, por lo que la comunicación con la máquina que alberga al SGBD se debe hacer a través de una red. En cualquiera de los dos casos, el SGBD recibe peticiones en forma de mensajes y responde de modo similar. Todas estas transmisiones de mensajes las maneja el gestor de comunicaciones de datos. Aunque este gestor no forma parte del SGBD, es necesario que el SGBD se pueda integrar con él para que el sistema sea comercialmente viable.

- Un SGBD debe proporcionar los medios necesarios para garantizar que tanto los datos de la base de datos, como los cambios que se realizan sobre estos datos, sigan ciertas reglas. La integridad de la base de datos requiere la validez y consistencia de los datos almacenados. Se puede considerar como otro modo de proteger la base de datos, pero además de tener que ver con la seguridad, tiene otras implicaciones. La integridad se ocupa de la calidad de los datos. Normalmente se expresa mediante restricciones, que son una serie de reglas que la base de datos no puede violar. Por ejemplo, se puede establecer la restricción de que cada empleado no puede tener asignados más de diez inmuebles. En este caso sería deseable que el SGBD controlara que no se sobrepase este límite cada vez que se asigne un inmueble a un empleado.

Además, de estos ocho servicios, es razonable esperar que los SGBD proporcionen un par de servicios más:

- Un SGBD debe permitir que se mantenga la independencia entre los programas y la estructura de la base de datos. La independencia de datos se alcanza mediante las vistas o subesquemas. La independencia de datos física es más fácil de alcanzar, de hecho hay varios tipos de cambios que se pueden realizar sobre la estructura física de la base de datos sin afectar a las vistas. Sin embargo, lograr una completa independencia de datos lógica es más difícil. Añadir una nueva entidad, un atributo o una relación puede ser sencillo, pero no es tan sencillo eliminarlos.

- Un SGBD debe proporcionar una serie de herramientas que permitan administrar la base de datos de modo efectivo. Algunas herramientas trabajan a nivel externo, por lo que habrán sido producidas por el administrador de la base de datos. Las herramientas que trabajan a nivel interno deben ser proporcionadas por el distribuidor del SGBD. Algunas de ellas son:

  - Herramientas para importar y exportar datos.
  - Herramientas para monitorizar el uso y el funcionamiento de la base de datos.
  - Programas de análisis estadístico para examinar las prestaciones o las estadísticas de utilización.
  - Herramientas para reorganización de índices.
  - Herramientas para aprovechar el espacio dejado en el almacenamiento físico por los registros borrados y que consoliden el espacio liberado para reutilizarlo cuando sea necesario.

2.2.3 Visión de los datos en un SGBD

Un sistema de bases de datos es una colección de archivos interrelacionados y un conjunto de programas que permitan a los usuarios acceder y modificar estos archivos. Uno de los propósitos principales de un SGBD es proporcionar a los usuarios una visión *abstrac-*

ta de los datos. Es decir, el sistema esconde ciertos detalles de cómo se almacenan y mantienen los datos.

2.2.3.1 Abstracción de datos

Para que el sistema sea útil debe recuperar los datos eficientemente. Como muchos usuarios de sistemas de bases de datos no están familiarizados con computadores, los desarrolladores esconden la complejidad a los usuarios a través de varios niveles de abstracción para simplificar la interacción de los usuarios con el sistema:

• **Nivel físico:** El nivel más bajo de abstracción describe *cómo* se almacenan realmente los datos. En el nivel físico se describen en detalle las estructuras de datos complejas de bajo nivel.

• **Nivel lógico:** El siguiente nivel más alto de abstracción describe *qué* datos se almacenan en la base de datos y qué relaciones existen entre esos datos. La base de datos completa se describe así en términos de un número pequeño de estructuras relativamente simples. Aunque la implementación de estructuras simples en el nivel lógico puede involucrar estructuras complejas del nivel físico, los usuarios del nivel lógico no necesitan preocuparse de esta complejidad. Los administradores de bases de datos, que deben decidir la información que se mantiene en la base de datos, usan el nivel lógico de abstracción.

• **Nivel de vistas:** El nivel más alto de abstracción describe sólo parte de la base de datos completa.

A pesar del uso de estructuras más simples en el nivel lógico, queda algo de complejidad, debido a la variedad de información almacenada en una gran base de datos. Muchos usuarios del sistema de base de datos no necesitan toda esta información. En su lugar, tales usuarios necesitan acceder sólo a una parte de la base de datos. Para que su interacción con el sistema se simplifique, se define la abstracción del nivel de vistas. El sistema puede proporcionar muchas vistas para la misma base de datos.

2.2.3.2 Ejemplares y esquemas

Las bases de datos van cambiando a lo largo del tiempo conforme la información se inserta y borra. La colección de información almacenada en la base de datos en un momento particular se denomina un *ejemplar* de la base de datos. El diseño completo de la base de datos se llama el *esquema* de la base de datos. Los esquemas son raramente modificados, si es que lo son alguna vez.

El concepto de esquemas y ejemplares de bases de datos se puede entender por analogía con un programa escrito en un lenguaje de programación. Un *esquema* de base de datos corresponde a las declaraciones de variables (junto con definiciones de tipos asociadas) en un programa. Cada variable tiene un valor particular en un instante de tiempo. Los valores de las variables en un programa en un instante de tiempo corresponde a un *ejemplar* de un esquema de bases de datos.

Los SGBD tienen varios esquemas divididos de acuerdo a los niveles de abstracción que se han discutido. El **esquema físico** describe el diseño físico en el nivel físico, mientras que el **esquema lógico** describe el diseño de la base de datos en el nivel lógico. Una base de datos puede tener también varios esquemas en el nivel de vistas, a menudo denominados **subesquemas**, que describen diferentes vistas de la base de datos.

De éstos, el esquema lógico es con mucho el más importante, en términos de su efecto en los programas de aplicación, ya que los programadores construyen las aplicaciones usando el esquema lógico. El esquema físico está oculto bajo el esquema lógico, y puede ser fácilmente cambiado usualmente sin afectar a los programas de aplicación. Los programas de aplicación se dice que muestran independencia física de datos si no dependen del esquema físico y, por tanto, no deben ser modificados si cambia el esquema físico.

2.2.4 Lenguajes de los SGBD

Podríamos definir un lenguaje como un conjunto de instrucciones que de acuerdo a una sintaxis ayudan a realizar las distintas funciones que ha de cumplir un SGBD. Si atendemos al tipo de función distinguimos: lenguajes de definición y lenguajes de manipulación.

2.2.4.1 Lenguaje de Definición de Datos (DDL)

Una vez finalizado el diseño de una base de datos y escogido un SGBD para su implementación, el primer paso consiste en especificar el esquema conceptual y el esquema interno de la base de datos, y la correspondencia entre ambos. En muchos SGBD no se mantiene una separación estricta de niveles, por lo que el administrador de la base de datos y los diseñadores utilizan el mismo lenguaje para definir ambos esquemas, es el *data definition language* (DDL). El SGBD posee un compilador de DDL cuya función consiste en procesar las sentencias del lenguaje para identificar las descripciones de los distintos elementos de los esquemas y almacenar la descripción del esquema en el catálogo o diccionario de datos. Se dice que el diccionario contiene *metadatos*: describe los objetos de la base de datos.

Cuando en un SGBD hay una clara separación entre los niveles conceptual e interno, el DDL sólo sirve para especificar el esquema conceptual. Para especificar el esquema interno se utiliza un *lenguaje de definición de almacenamiento* (LDA). Las correspondencias entre ambos esquemas se pueden especificar en cualquiera de los dos lenguajes. Para tener una verdadera arquitectura de tres niveles sería necesario disponer de un tercer lenguaje, el *lenguaje de definición de vistas* (LDV), que se utilizaría para especificar las vistas de los usuarios y su correspondencia con el esquema conceptual.

Un ejemplo de comandos de DDL en Oracle serían:
`Create Table` *(Crea una tabla),* `Create View` *(Crea una vista),* `Drop Table` *(Elimina una tabla),* `Drop Squema` *(Elimina un esquema) o* `Alter` *Table (Modifica una tabla).*

2.2.4.2 Lenguaje de Manipulación de Datos (DML)

Una vez creados los esquemas de la base de datos, los usuarios necesitan un lenguaje que les permita manipular los datos de la base de datos: realizar consultas, inserciones, eliminaciones y modificaciones. Este lenguaje es el que se denomina *data manipulation language* (DML). Hay dos tipos de DML: los procedurales y los no procedurales.

Con un DML *procedural* el usuario (normalmente será un programador) especifica qué datos se necesitan y cómo hay que obtenerlos. Esto quiere decir que el usuario debe especificar todas las operaciones de acceso a datos llamando a los procedimientos necesarios para obtener la información requerida. Las sentencias de un DML procedural deben estar embebidas en un lenguaje de alto nivel, ya que se necesitan sus estructuras (bucles, condicionales, etc.) para obtener y procesar cada registro individual. A este lenguaje se le denomina *lenguaje anfitrión*. Las bases de datos jerárquicas y de red utilizan DML procedurales.

Un DML *no procedural* se puede utilizar de manera independiente para especificar operaciones complejas sobre la base de datos de forma concisa. En muchos SGBD se pueden introducir interactivamente instrucciones del DML desde un terminal o bien embeberlas en un lenguaje de programación de alto nivel. Los DML no procedurales permiten especificar los datos a obtener en una consulta o los datos que se deben actualizar, mediante una sola y sencilla sentencia. El usuario o programador especifica qué datos quiere obtener sin decir cómo se debe acceder a ellos.

A los DML no procedurales también se les denomina *declarativos*. Las bases de datos relacionales utilizan DML no procedurales, como SQL. Los lenguajes no procedurales son más fáciles de aprender y de usar que los procedurales, y el usuario debe realizar menos trabajo, siendo el SGBD quien hace la mayor parte.

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 sólo 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.

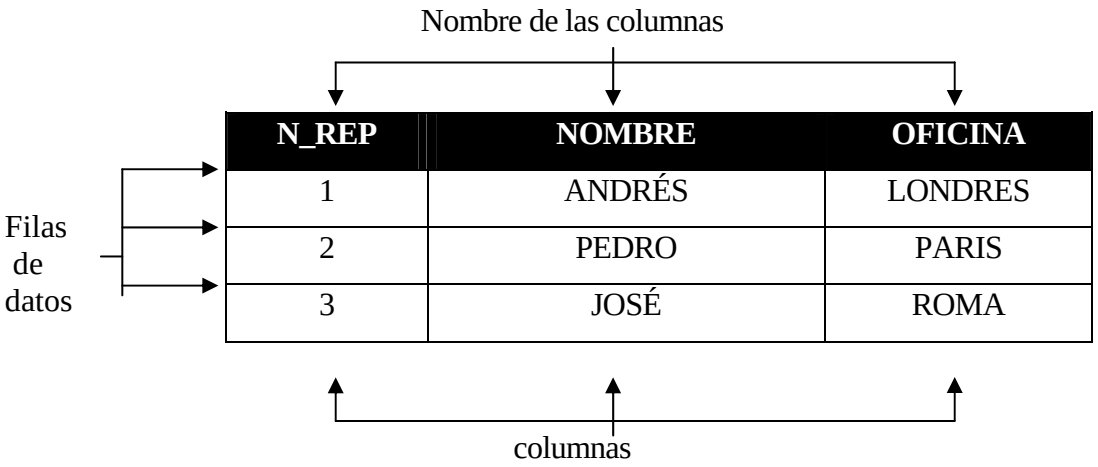


Figura 2.1: Ejemplo de la estructura de una tabla

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.

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,
- Espacios de tablas,
- Usuarios,
- Bibliotecas,
- Índices y clusters,
- Vistas,
- Secuencias,
- Trabajos,

- Procedimientos, funciones, paquetes y disparadores,
- Sinónimos,
- Privilegios y roles,
- Instantáneas o vistas materializadas.

2.3.1 Tablas (Tables)

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**]**.

2.3.2 Espacios de tablas (Tablespaces)

Un espacio de tablas es una división lógica de una base de datos. Toda base de datos consta, al menos, de un espacio de tablas (llamado espacio de tablas SYSTEM). Se pueden utilizar otros espacios de tablas para agrupar a los usuarios o aplicaciones, con el fin de facilitar el mantenimiento y mejorar el rendimiento. Algunos espacios de tablas de este tipo serían USERS, para uso general, y RBS, para los segmentos de anulación. Un espacio de tablas sólo puede pertenecer a una base de datos.

Cada espacio de tablas consta de uno o más archivos, llamados datafiles o archivos de datos, que se almacenan en disco. Un archivo de datos sólo puede pertenecer a un único

espacio de tablas. El tamaño de los archivos de datos se puede modificar después de su creación. Para crear nuevos espacios de tablas habrá que crear nuevos archivos de datos. También hay que reseñar que una vez que un archivo de datos se ha añadido a un espacio de tablas, ese archivo de datos no se puede sacar del espacio de tablas ni se puede asociar con otro espacio de tablas.

2.3.3 Restricciones (Constraints)

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.

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* (PRIMARY KEY) 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 cláusula `DEFAULT`. No es propiamente una restricción y se utiliza para generar un valor por defecto 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 que define.

Para especificar la naturaleza de la relación entre tablas, se utiliza una restricción de *clave externa* (FOREIGN KEY). 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.

2.3.4 Usuarios (Users)

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.

2.3.5 Bibliotecas (Libraries)

Una biblioteca es un objeto asociado a un esquema de usuario que representa a una biblioteca compartida del Sistema Operativo. Su principal utilidad va a ser en las declaraciones de bloques PL/SQL, al crear procedimientos o funciones, pues con la palabra clave "call_spec" podremos invocar funciones que estén escritas en lenguajes de 3ª Generación, como Java o C.

2.3.6 Índices (Indexes)

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).

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.

Los índices constan de un valor clave (columnas por las que se ordena) y de un identificador de fila (RowID). Se puede indexar una sola columna o un conjunto de columnas. Oracle almacena usualmente 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 RowID del valor que coincida con la consulta proporciona a Oracle la ubicación física de la fila asociada, reduciendo así el tiempo necesario para localizar los datos.

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.

A partir de Oracle7.3 pueden crearse *índices de mapas de bits*. Como hemos dicho, 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 son aceleradas 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.

También 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. 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**]**.

2.3.7 Agrupaciones (Clusters)

Las tablas relacionadas con una llave externa y 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.

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 lectura como si las tablas se almacenaran en ubicaciones diferentes. Sin embargo, el rendimiento de las operaciones de inserción, actualización y eliminación en las tablas agrupadas puede ser notablemente inferior que las mismas operaciones realizadas en tablas no agrupadas.

2.3.8 Vistas (Views)

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 forma que el usuario puede no necesitar saber si es una tabla o una vista.

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, como por ejemplo, si se usan funciones de grupo con GROUP BY.

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, para ofrecer a cada usuario el acceso exclusivo a los datos que necesita y de la forma que necesita.

2.3.9 Secuencias (Sequences)

Las secuencias proporcionan una lista consecutiva de números unívocos no repetidos que sirve para simplificar las tareas de programación.

La primera vez que una consulta llama a una secuencia, se devuelve un valor inicial predeterminado. En cada consulta siguiente, se obtendrá un valor incrementado según el incremento especificado. 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 Trabajos (Jobs)

Podríamos definir un trabajo como un bloque PL/SQL que se va a ejecutar con una periodicidad establecida. Tienen utilidad para casos de tareas repetitivas, que se van a ejecutar periódicamente.

No existen privilegios asociados con el uso de los trabajos, o lo que es lo mismo, cualquier usuario podrá utilizar el paquete *DBMS.JOB* para añadir nuevos trabajos. Cuando creamos un trabajo nos convertiremos en su propietario, y sólo nosotros tendremos la posibilidad de cambiar, eliminar o forzar la ejecución de dicho trabajo. En cuanto a la distinción entre los distintos trabajos, se realiza por un número, que es asignado por la secuencia *SYS.JOBSEQ* al añadir el trabajo y que no podrá ser alterado de ninguna manera.

2.3.11 Procedimientos (Procedures)

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. 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 devuelven 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 directo 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.

2.3.12 Funciones (Functions)

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.

2.3.13 Paquetes (Packages)

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*).

2.3.14 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 según cuando deban ejecutarse:

- 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 de sustitución o `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.

2.3.15 Sinónimos (Synonyms)

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.

2.3.16 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 con un nombre 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 y los roles se retiran con el comando `REVOKE`.

Para conceder privilegios mediante roles, se crea un rol de base de datos, se conceden privilegios al rol y, finalmente, se asigna el rol a los usuarios que se requiera.

2.3.17 Instantáneas o vistas materializadas

Una vista materializada es un tipo de vista que no se actualiza cuando se cambian las tablas base, mientras que las vistas sí lo hacen. En las vistas materializadas podremos definir el tipo de refresco, así como el intervalo de refresco, para que los datos se actualicen cuando lo necesitemos.

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 de la vista materializada, podría utilizar un *registro de vistas materializadas* (LOG) con el fin de mejorar el rendimiento de las operaciones de actualización de la vista materializada. 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 que contienen los datos cifrados y almacenados por el sistema. Hay vistas accesibles por el usuario en el diccionario que resumen y visualizan convenientemente la información a los usuarios. Los usuarios muy pocas veces tienen acceso a dichas tablas. Los prefijos especiales USER, ALL y DBA se usan respectivamente para hacer referencia a la vista con objetos del usuario (objetos del esquema que posee el usuario), vistas con objetos para los que el usuario tiene autorización de acceso 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: *login* y *password*
- 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.
- Información de las dependencias entre los objetos.
- Información del aspecto de los datos de la base de datos

Algunas de las vistas más importantes y útiles del diccionario de datos están en la tabla 2.1:

NOMBRE	DESCRIPCIÓN
USER_CATALOG	Todas las tablas, vistas, secuencias y sinónimos propiedad del usuario.
USER_TAB_COLUMS	Todas las columnas de las tablas propiedad del usuario.
USER_DEPENDENCIES	Muestra las dependencias existentes entre los objetos propiedad del usuario.
ALL_CONSTRAINTS	Describe todas las restricciones de las tablas accesibles por el usuario
ALL_OBJECTS	Todos los objetos a los que el usuario puede acceder.
ALL_CATALOG	Todas las tablas, vistas, secuencias y sinónimos a los que el usuario tiene acceso.
DBA_DATA_FILES	Describe los archivos de la base de datos.
DBA_SOURCE	Describe el código fuente de todos los objetos almacenados en la base de datos.

Tabla 2.1: Algunas de las vistas del diccionario de datos.

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 rendimiento del sistema y puede conceder permiso de acceso a algunos usuarios sobre las vistas de estas tablas. Podemos ver algunos ejemplos en la tabla 2.2.

NOMBRE	DESCRIPCIÓN
V$DATABASE	Contiene información sobre la BD (nombre …).
V$DATAFILE	Contiene información sobre los *datafiles.*
V$FIXED_TABLE	Contiene información sobre todas las tablas de ejecución dinámica y vistas de la BD.
V$SESSION	Contiene información sobre las sesiones actuales (usuario, programa…).
V$PROCESS	Contiene información sobre los procesos activos.
V$CONTROLFILE	Contiene una lista con los ficheros de control.

Figura 2.2: Tablas dinámicas de rendimiento

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 Transacciones

Se llama **transacción** a una colección de operaciones que forman una única unidad lógica de trabajo. Un sistema de base de datos debe asegurar que la ejecución de las transacciones se realice adecuadamente a pesar de la existencia de fallos: o se ejecuta la transacción completa o no se ejecuta en absoluto. Además debe gestionar la ejecución concurrente de las transacciones evitando introducir inconsistencias.

Para asegurar la integridad de los datos se necesita que el sistema de base de datos mantenga las siguientes propiedades de las transacciones:

- **Atomicidad.** O todas las operaciones de la transacción se realizan adecuadamente en la base de datos o ninguna de ellas.
- **Consistencia.** La ejecución aislada de la transacción (es decir, sin otra transacción que se ejecute concurrentemente) conserva la consistencia de la base de datos.
- **Aislamiento.** Aunque se ejecuten varias transacciones concurrentemente, el sistema garantiza que para cada par de transacciones T_i y T_j, se cumple que para los efectos de T_i, o bien T_j ha terminado su ejecución antes de que comience T_i, o bien que T_j ha comenzado su ejecución después de que T_i termine. De este modo, cada transacción ignora al resto de las transacciones que se ejecuten concurrentemente en el sistema.
- **Durabilidad.** Tras la finalización con éxito de una transacción, los cambios realizados en la base de datos permanecen, incluso si hay fallos en el sistema.

2.5.1 Definición de transacciones en SQL

En la norma SQL se especifica el comienzo de una transacción explícitamente. Las transacciones se terminan con una de las instrucciones SQL siguientes:

- **Commit**: Confirma la transacción actual.

- **Rollback:** Aborta la transacción actual, por lo que el sistema vuelve al estado en el que estaba al empezar la transacción.

También se pueden definir puntos de guarda (*Savepoints*) que nos permitirán realizar *rollbacks* a una parte específica de la transacción. De esta forma podremos deshacer parte de una transacción, sin tener que deshacerla completamente.

2.5.2 Procesamiento de transacciones multiusuario

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.

Oracle ofrece 3 niveles distintos de aislamiento, que son:

- READ COMMITED: Cada consulta que ejecuta una transacción ve solamente los datos que han sido confirmados con anterioridad al inicio de ejecución de la consulta (no de la transacción): Se llama también consistencia a nivel de sentencia.
- SERIALIZABLE: Las transacciones ven solamente los datos que han sido confirmados con anterioridad al inicio de la transacción, exceptuando los cambios que ellas mismas realicen: También llamado consistencia a nivel de transacción.

- READ ONLY: Igual que *Serializable*, pero además no permite que la transacción realice ninguna modificación en los datos.

2.6 Instalación paso a paso de Oracle

El formato de distribución de Oracle (en este caso concreto la versión 8.1.7) 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.

2.6.1 Requerimientos de Oracle

La ocupación en disco oscilará entre los 350 megabytes, para una instalación mínima, y los alrededor de 650 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, aunque lo recomendado son al menos 128 megabytes.

El S.O. necesario es Windows XP/NT/2000. Si usamos otra versión de Windows o cualquier otro sistema operativo como Linux o UNIX deberíamos adquirir la versión para estas plataformas, puesto que no hay compatibilidad entre las distintas plataformas. Para obtener más información sobre Oracle, puede visitar la web de Oracle[1].

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

2.6.2 Comenzando la instalación

Para iniciar la instalación 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 archivo *SETUP.exe* que se encuentra en el directorio raíz del CD.

Figura 1.2: Ventana del menú de inicio de instalación de la versión 8.1.7

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

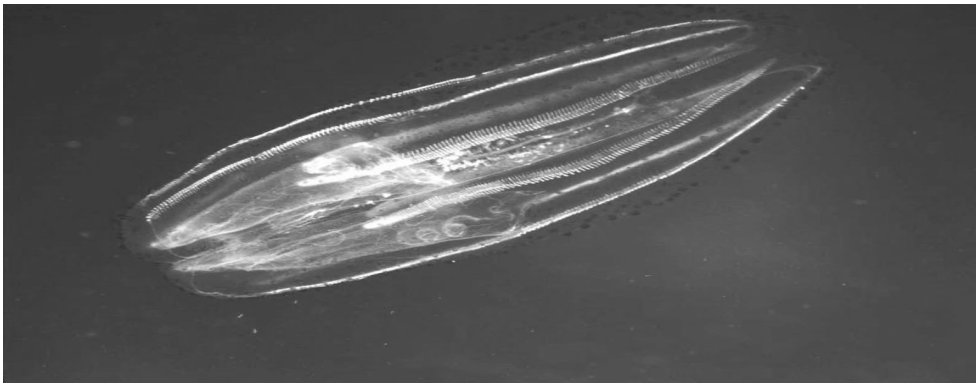
Ahora pulsamos el botón correspondiente a la instalación. Tras unos instantes, aparecerá una pantalla de bienvenida. Tras pulsar el botón *Siguiente*, podemos cambiar el directorio donde se instalará Oracle. Pulsamos de nuevo el botón *Siguiente*, lo que nos lleva al formulario donde se selecciona el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

Figura 2.2: 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. En la Figura 2.3 podemos ver una vista en árbol sobre los distintos componentes que podemos seleccionar para incluir en nuestra instalación.

Pulsando alguna vez más en el botón *Siguiente* llegamos a una ventana en la que vemos 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.



3

Delphi

El nombre Delphi hace referencia al oráculo de Delfos. Borland eligió ese nombre para resaltar su principal mejora con respecto a su antecesor (Turbo Pascal), que sería su conectividad con bases de datos Oracle (oráculo, en español).

3.1 ¿Qué es Delphi?

Delphi es un lenguaje de programación y un entorno de desarrollo rápido de software (RAD) diseñado para la programación de propósito general con énfasis en la programación visual. Es producido comercialmente por la empresa estadounidense Borland Software Corporation[1]. En sus diferentes variantes, permite producir ejecutables binarios para Windows y Linux; y también para la plataforma .NET de Microsoft.

El principal uso de Delphi es para el desarrollo de aplicaciones visuales, de bases de datos cliente-servidor y multicapas. Debido a que es una herramienta de propósito múltiple, se usa también para proyectos de casi cualquier tipo, incluyendo aplicaciones de consola, CGI (Common Gateway Interface**)** y servicios del sistema operativo.

Ahora haremos un breve recorrido en el tiempo, mientras repasamos cada una de las versiones de Delphi, y a continuación veremos las distintas ediciones que comercializa.

3.1.1 Siete versiones y sumando

En la primavera de 1995 apareció **Delphi 1**. Algunas de sus propiedades originales que atrajeron a primera vista fueron su enfoque orientado a objetos y basado en formularios, su compilador extremadamente rápido, su gran soporte para bases de datos, su estrecha integración con la programación para Windows y su tecnología de componentes. Pero el elemento más importante era el lenguaje Pascal orientado a objetos, que es la base de todo lo demás.

¡**Delphi 2** era incluso mejor! Entre sus propiedades añadidas más importantes estaban las siguientes: El Multi Record Object y la cuadrícula para bases de datos mejorada, el soporte para automatización OLE y el tipo de datos variantes, el soporte e integración totales de Windows 95, el tipo de datos de cadena larga y la herencia de formulario visual.

[1] La página Web de Borland Software Corporation está disponible en http://www.borland.com

Delphi 3 añadió la tecnología Code Insight, el soporte de depuración DLL, las plantillas de componentes, el TeeChart, el Decisión Cube, la tecnología WebBroker, los paquetes de componentes, los ActiveForms y una sorprendente integración con COM, gracias a los interfaces.

Delphi 4 nos trajo el editor AppBrowser, nuevas propiedades de Windows 98, mejor soporte OLE y COM, componentes de bases de datos ampliados y muchas más clases principales de la VCL (Visual Component Library) añadidas, como el soporte para acoplamiento, restricción y anclaje de los controles.

Delphi 5 añadió a este cuadro muchas mejoras en el IDE (Integrated Development Environment), soporte ampliado para bases de datos (con conjuntos de datos específicos de ADO e Interbase), una versión mejorada de *MIDAS (Multitier Distributed Applicaction Services)* con soporte para Internet, la herramienta de control de versiones TeamSource, capacidades de traducción, el concepto de marcos y nuevos componentes.

Delphi 6 añadió a todas estas propiedades el soporte para el desarrollo multiplataforma con la nueva biblioteca de componentes para multiplataforma (CLX), una biblioteca en tiempo de ejecución ampliada, el motor para bases de datos dbExpress, un soporte excepcional de servicios Web y XML (eXtensible Markup Language), un poderoso marco de trabajo de desarrollo Web, más mejoras en el IDE y multitud de componentes y clases.

Delphi 7 aparece en el mercado en el año 2002. Al aparecer proporcionó más robustez a estas nuevas tecnologías con mejoras y arreglos, como el soporte de SOAP y DataSnap, y ofreció soporte para tecnologías más novedosas, como los temas de Windows XP o UDDI (Universal Description, Discovery, and Integration*)*, pero lo más importante es que permitía disponer rápidamente de un interesante conjunto de herramientas de terceras partes: el motor de generación de informes RAVE, la tecnología de desarrollo de aplicaciones Web IntraWeb y el entorno de diseño ModelMaker.

3.1.2 Ediciones de Delphi 7

Las ediciones actuales de Delphi 7 son las siguientes:

- **La edición "Personal":** Dirigida a quienes empiezan a utilizar Delphi y a programadores esporádicos. No soporta programación de bases de datos ni ninguna de las características avanzadas de Delphi.

- **La edición "Profesional Studio":** Dirigida a desarrolladores profesionales. Posee todas las características básicas, más soporte para programación de bases de datos, soporte básico para servidores Web (WebBroker) y algunas herramientas externas como ModelMaker e IntraWeb.

- **La edición "Enterprise Studio":** Está dirigida a desarrolladores que crean aplicaciones para empresas. Incluye todas las tecnologías XML y de servicios Web avanzados, soporte de CORBA, internacionalización, arquitectura en tres niveles y muchas otras herramientas.

- **La edición "Architect Studio":** Añade a la edición Enterprise el soporte de *Bold*, un entorno para la creación de aplicaciones dirigidas en tiempo de ejecución por un modelo UML (Unified Modelling Language) y capaces de proyectar sus objetos tanto sobre una base de datos como sobre una interfaz de usuarios, gracias a una gran cantidad de componentes avanzados.

3.2 El lenguaje de programación Delphi

El entorno de desarrollo para Delphi se basa en una extensión orientada a objetos del lenguaje de programación Pascal conocida como Object Pascal o Pascal orientado a objetos.

La mayoría de los lenguajes de programación modernos soportan OOP (Object-oriented programming). Los lenguajes OOP se basan en tres conceptos fundamentales: la encapsulación (normalmente implementada mediante clases), la herencia y el polimorfismo. Aunque se puede escribir código Delphi sin comprender las características principales del lenguaje, no es posible dominar este entorno hasta que se comprende totalmente el lenguaje de programación.

En este capítulo vamos a tratar los siguientes temas:

- Clases y objetos.
- Encapsulación.
- Constructores y destructores.
- Objetos y memoria.
- Herencia y polimorfismo.
- Trabajo con excepciones.

3.2.1 Clases y objetos

Una clase es un tipo de datos definido por el usuario, que posee un estado (su representación o sus datos internos) y algunas operaciones (su comportamiento o métodos). Un objeto es una instancia de una clase o una variable del tipo de datos definido por la clase. Los objetos son entidades reales. Cuando el programa se ejecuta, los objetos ocupan parte de la memoria para su representación interna. La relación entre objeto y clase es la misma que entre variable y tipo.

Como en la mayor parte del resto de los lenguajes orientados a objetos (como Java y C++), en Delphi una variable de tipo clase no proporciona el almacenamiento para el objeto, sino sólo un puntero o referencia al objeto en la memoria. Antes de utilizar el objeto, se debe reservar memoria para él mediante la creación de una nueva instancia o asignando una instancia ya existente a la variable. Normalmente esto se hará con una llamada al constructor predefinido disponible para cada clase denominado *Create*, que luego podrá ser redefinido si se quiere.

El comportamiento de una clase se define mediante sus métodos, que se definen mediante la palabra clave *function* o *procedure*, según si tienen un valor de retorno o no. Es también importante destacar algunas de las características de los métodos en Delphi:

- Delphi soporta la sobrecarga de métodos. Esto significa que se pueden tener dos métodos con el mismo nombre, siempre que se marquen los métodos con la palabra clave *overload* y que las listas de los parámetros de los dos métodos sean diferentes.
- Los métodos pueden tener uno o más parámetros con valores predefinidos. Si estos parámetros se omitiesen en la llamada al método, se asignaría el valor predefinido.
- Dentro de un método se puede usar la palabra clave *Self* para acceder al objeto actual.

3.2.2 Encapsulación

El concepto de encapsulado de objetos describe normalmente un objeto como una *caja negra*, en la que no se conoce el interior, simplemente se sabe como interactuar con ella o cómo usarla, sin tener en cuenta su estructura. Una clase puede tener cualquier cantidad de datos y cualquier número de métodos. Sin embargo, para conseguir una buena técnica orientada a objetos, los datos deberían estar ocultos o encapsulados dentro de la clase que los usa.

Delphi implementa este encapsulado basado en clases pero todavía soporta el encapsulado clásico basado en módulos, que usa la estructura de unidades. Todo identificador que se declare en la sección de interfaz de una unidad resulta visible a otras unidades del programa, siempre que se utilice una sentencia *uses* que se refiere a la unidad que define el identificador. Por otro lado, los identificadores declarados en la sección de implementación de la unidad serán locales a esa unidad.

3.2.2.1 Privado, protegido y público

En el caso de un encapsulado basado en clases, el lenguaje Pascal orientado a objetos tiene, como C++, tres especificadores de acceso: *private, protected y public*. Un cuarto, *publisher*, controla la RTTI (Run Time Type Information) y la información en tiempo de diseño, proporcionando la misma disponibilidad de cara a la programación que si fuera *public*. A continuación se describen los tres especificadores de acceso clásicos:

- **La directiva *private*:** Denota campos y métodos de clase no accesibles fuera de la unidad (el archivo de código fuente) que declara la clase.

- **La directiva *protected*:** Se utiliza para indicar métodos y campos con visibilidad limitada. Sólo la clase actual y sus clases heredadas pueden acceder a los elementos protegidos. Para ser más precisos, sólo la clase, las subclases y cualquier código presente en la misma unidad que la clase pueden acceder a los miembros protegidos.
- **La directiva *public*:** Denota campos y métodos a los que se puede acceder libremente desde cualquier otra parte de un programa así como en la unidad en la que se definen.

3.2.2.2 Propiedades

Las propiedades son un mecanismo de orientación a objetos muy sensato o una aplicación práctica muy bien pensada de la idea de encapsulado. Básicamente, se tiene un nombre que oculta por completo los datos de implementación. Esto permite modificar la clase ampliamente sin que afecte al código que la utiliza. Una buena definición de propiedades es la de campos virtuales. Desde la perspectiva del usuario de la clase que las define, las propiedades poseen una apariencia exactamente igual a la de los campos, ya que, por lo general se puede leer o escribir su valor. Por ejemplo, se puede leer el valor de la propiedad *Caption* de un botón, llamado *Button1*, y asignarla a la propiedad *Text* de un cuadro de edición, llamado *Edit1*, simplemente así:

```
Edit1.Text := Button1.Caption;
```

3.2.3 Constructores y destructores

Para asignar la memoria a objetos, podemos llamar al método *Create*. Este es un constructor, un método especial que podemos aplicar a una clase para asignar memoria a una instancia de dicha clase. El constructor devuelve la instancia, que puede asignarse a una variable para almacenar el objeto y usarlo más tarde. El constructor por defecto *TObject.Create* inicializa todos los datos del nuevo caso a cero. Para que los datos de dicho caso comiencen con un valor diferente a cero, hay que escribir un constructor personalizado.

Del mismo modo que una clase puede tener un constructor personalizado, también puede tener un destructor personalizado, un método declarado con la palabra clave *Destructor* y llamado *Destroy*. Al igual que una llamada al constructor asigna memoria para el objeto, un destructor libe-

ra la memoria. Los destructores son necesarios sólo para objetos que adquieren recursos externos en sus constructores o durante su vida útil.

3.2.4 Objetos y memoria

La administración de memoria en Delphi está sujeta a tres normas, al menos si se permite que el sistema trabaje en armonía sin violaciones de acceso y sin consumir memoria innecesaria:

- Todo objeto ha de ser creado antes de que pueda usarse.
- Todo objeto ha de ser destruido tras haberlo utilizado.
- Todo objeto ha de ser destruido sólo una vez.

Con motivo de respetar estas reglas hemos elaborado una lista de directrices:

- Llamar siempre a *Free* para destruir objetos, en lugar de llamar al destructor *Destroy*.
- Utilizar *FreeAndNil* o cambiar las referencias del objeto a *nil* después de haber llamado a *Free*, a no ser que la referencia quede inmediatamente después fuera de alcance.

3.2.5 Herencia y polimorfismo

Herencia es el mecanismo que permite a una clase de objetos incorporar atributos y métodos de otra clase, añadiéndolos a los que ya posee. La clase que hereda las características de otra y la clase de partida reciben los calificativos de "subclase" y "superclase", respectivamente. De ahí que, en numerosas ocasiones, la relación de herencia aparezca también referenciada como "superclase/subclase". Por otro lado, también suele ser muy habitual hablar en términos de "clase padre" y "clase hija", dado lo intuitivo de ambos términos.

La contribución más interesante del mecanismo de herencia al desarrollo de software, es la flexibilidad que proporciona para capturar y aprovechar al máximo las características comunes de diferentes clases de objetos. Dicha flexibilidad se manifiesta, fundamentalmente, en dos sentidos:

- Por un lado, permite recoger los aspectos comunes de dos o más clases de objetos con el máximo nivel de detalle (a nivel de atributo y de método).
- Por otro, ofrece la posibilidad de establecer tantos niveles de abstracción (o de especialización, según se mire) como sean necesarios para reflejar fielmente nuestro modelo de la realidad. Surge, de este modo, el concepto de jerarquía de clases.

Podemos definir polimorfismo como el mecanismo que permite definir e invocar funciones idénticas en denominación e interfaz, pero con implementaron diferente. Esta definición introduce un aspecto muy importante del polimorfismo: la asociación, o vínculo, entre cada llamada a una de estas funciones polimorfismo y la implementación concreta finalmente invocada. Cuando éste vinculo puede establecerse en tiempo de compilación, se suele hablar de vinculación estática (*static binding*). Por contra, cuando la implementación a emplear puede determinarse en tiempo de ejecución, el término empleado es el de vinculación dinámica (*dynamic binding*).

El concepto de polimorfismo está estrechamente ligado al concepto de herencia, dado que las funciones polimórficas sólo pueden definirse entre clases que guardan entre sí una relación de parentesco (clases con un antecesor común).

3.2.6 Trabajo con excepciones

Otra característica clave de Delphi es el soporte de excepciones. Las excepciones hacen que los programas sean más robustos ya que proporcionan un modo estándar de notificar y gestionar errores y situaciones inesperadas. Las excepciones hacen que los programas sean más fáciles de escribir, leer y depurar porque permiten separar el código de gestión de errores del código normal. Al obligar a mantener una división lógica entre el código y la gestión de errores y al conmutar al manejador de errores automáticamente, se consigue que la lógica real resulte más limpia y clara. En tiempo de ejecución, las bibliotecas de Delphi crean excepciones cuando algo va mal.

Desde el punto del código en que se crea, la excepción se pasa a su código de llamada, y así sucesivamente. Por último, si ninguna parte del código controla la excepción, la VCL se encarga de ella, mostrando un mensaje estándar de error y tratando de continuar el programa proporcionando

el siguiente mensaje del sistema o petición al usuario. Todo este mecanismo se basa en cuatro palabras clave:

- **Try:** Delimita el comienzo de un bloque protegido de código.
- **Except**: Delimita el final de un bloque protegido de código e introduce las sentencias de control de excepciones.
- **Finally:** Se usa para especificar bloques de código que han de ejecutarse siempre, incluso cuando se dan excepciones. Este bloque se usa generalmente para realizar operaciones de limpieza que siempre se deberían ejecutar, como cerrar archivos o tablas de bases de datos, liberar objetos y liberar memoria y otros recursos adquiridos en el mismo bloque de programa.
- **Raise:** Es la sentencia usada para generar la excepción. La mayoría de las excepciones que encontramos en programación en Delphi las genera el sistema, pero también se pueden crear excepciones propias en el código, cuando se descubren datos no válidos o incoherentes en tiempo de ejecución. La palabra clave *raise* también puede usarse dentro de un controlador para volver a crear una excepción, es decir, para propagarla al siguiente controlador.

3.3 Instalación paso a paso

El formato de distribución de Delphi (en este caso concreto la versión 7) es el CD, en el cual se incluye *Delphi 7*, *InterBase Server*, *InstallShield*, *TeamSource*, *ModelMaker* y un amplio conjunto de aplicaciones externas para diversas funciones. 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 no nos interesen en el CD, sin instalarlas en nuestro sistema.

3.3.1 Requerimientos de Delphi

La ocupación en disco oscilará entre los 150 megabytes, para una instalación típica de la versión *Standard*, y los alrededor de 400 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 64 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, 128 megabytes de memoria es una configuración adecuada, aunque lo recomendado son al menos 256 megabytes.

El software necesario será Windows 95/98 o NT/2000/XP. También existe una versión para Linux, pero en este caso su nombre no es Delphi, sino Kylix.

3.3.2 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.

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 7. 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 podemos ver una lista con las distintas posibilidades de instalación.

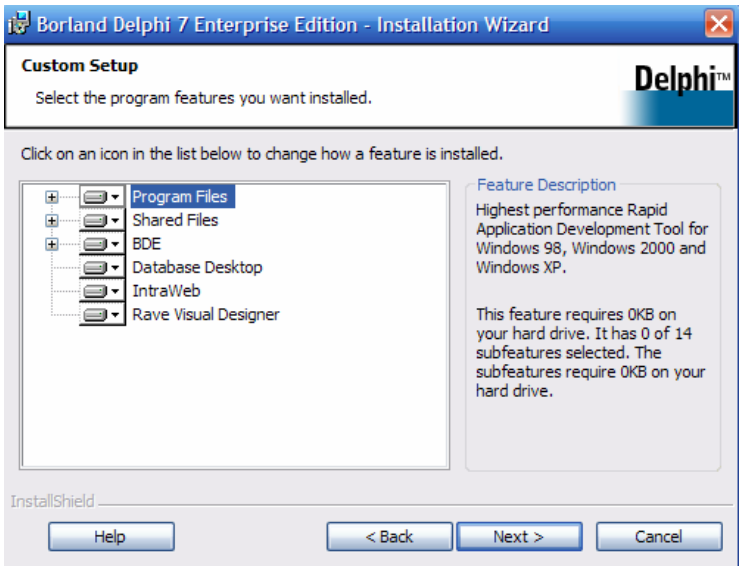
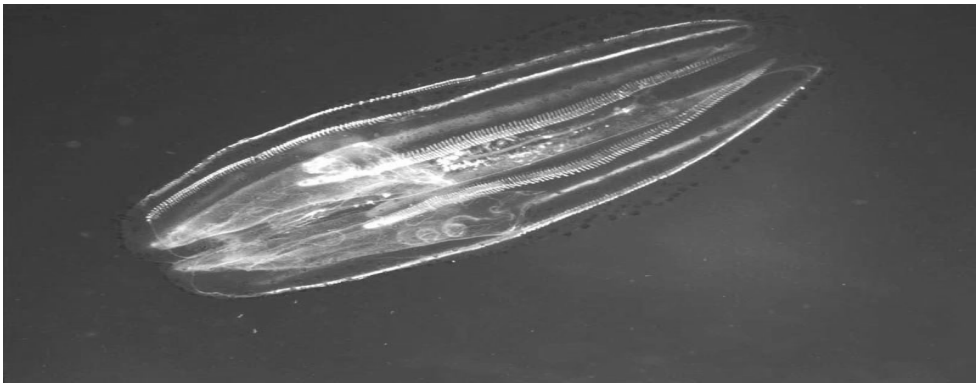


Figura 3.2 Selección de los elementos a instalar en Delphi 7

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.



4

Componentes

Delphi incluye una biblioteca de clases bien diseñada denominada VCL (Visual Component Library, Biblioteca de Componentes Visuales), pero además de poder utilizar en un programa estos componentes estándar (botones, rejillas, conjuntos de datos, etc.), es posible crear nuevos componentes o mejorar los ya existentes.

En Internet podremos encontrar un gran número de componentes, tanto gratuitos como comerciales, disponibles para los proyectos a los que no les basten con los que vienen en Delphi.

4.1 Un poco de teoría

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 **[**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).

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

DOA son una familia de componentes 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 SGBD Oracle. La versión de DOA utilizada ha sido la 4.0.5 y 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`, `SetTransaction`...).
- 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

La familia de componentes de acceso directo a Oracle 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`.

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. Dependiendo de la versión se nos mostrará un diálogo semejante al de la Figura 4.1.

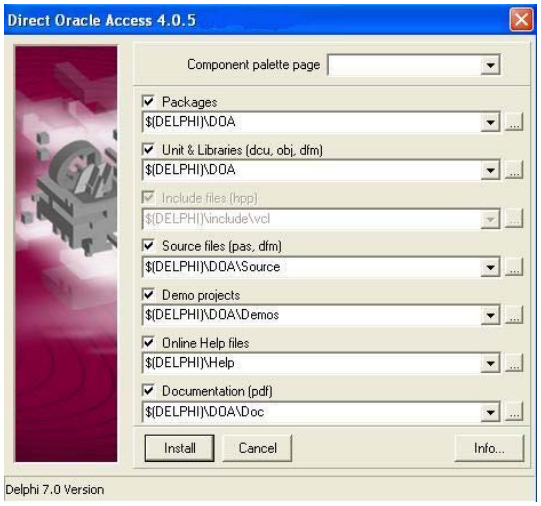


Figura 4.1: Instalación de los componentes de Acceso Directo a Oracle (DOA)

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.

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

SynEdit son una serie de componentes y objetos, que permiten realzar la sintaxis de un determinado lenguaje cambiando el color de las constantes, palabras reservadas…
SynEdit incorpora multitud de unidades. Cada unidad se encarga de un lenguaje en concreto. SynEdit también nos ofrece la posibilidad de crear nuestro propia unidad de resaltado de sintaxis y poder emplearla asociada con en el control `TSynEdit`.

En Medusa, lo vamos a utilizar para realzar el texto de la ventana que utilizamos cuando abrimos una nueva sesión. De esta forma tendremos resaltado el lenguaje SQL y será mucho más intuitiva y cómoda la visión de las sentencias. SynEdit, se puede descargar gratuitamente 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 y copiamos la carpeta source, con los archivos fuente necesarios, en la ubicación que deseemos (por ejemplo en la carpeta de Delphi creamos una carpeta con el nombre SynEdit).
2. Ahora tenemos que informarle a Delphi de la nueva ruta de los componentes SynEdit. Para ello debemos añadir el directorio donde hemos instalado los componentes, de la siguiente forma.
    a. En el menú *Tools*, hacemos click sobre *Environment Options.*
    b. Ahora vamos a la pestaña *Library*.
    c. En *Library path*, añadimos el directorio donde estén los fuentes de los componentes SynEdit poniendo un punto y coma al final de linea que ya hay por defecto, sin ningún espacio intermedio pues Delphi no lo reconoce, y añadimos *$(DELPHI)\SynEdit*, tal y como se ilustra en la Figura 4.2.
3. Abrimos el fichero SynEdit_D6.dpk que se encuentra ubicado dentro del directorio *Packages* donde hayamos descomprimido anteriormente el fichero *zip*.
4. Ahora compilamos el paquete. Como muestra la Figura 4.3, debemos pulsar *Compile*.
5. Una vez que hayamos compilado, instalamos el paquete. Para ello, debemos pulsar I*nstall*.

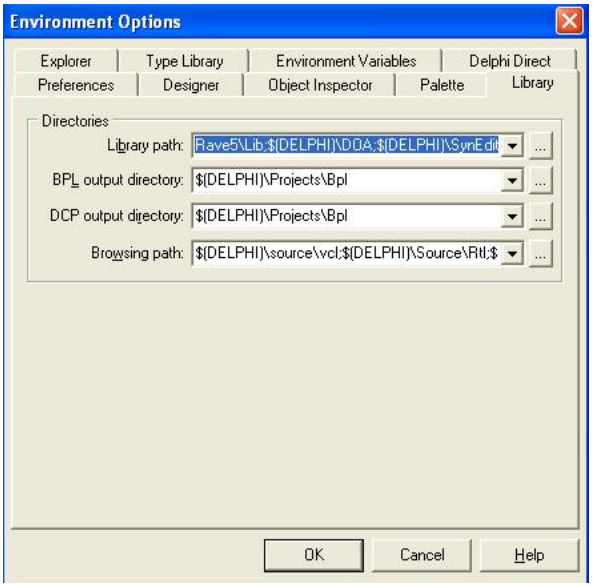


Figura 4.2: Añadiendo la ruta de búsqueda de los componentes.

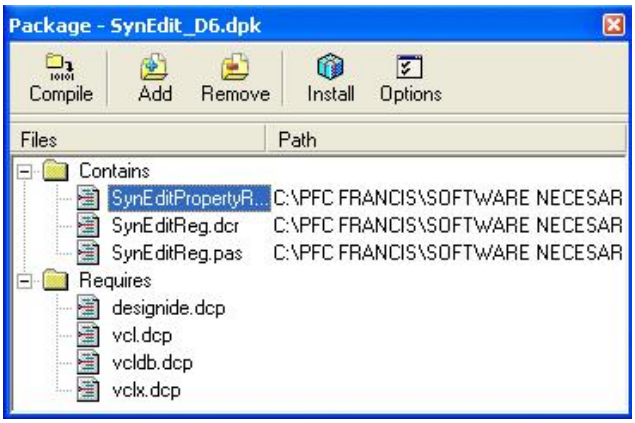


Figura 4.3: Instalación del paquete `SynEdit_D6.dpk`

4.4 Accutime

Accutime es un sencillo componente similar al estándar de Delphi TTimer, con la diferencia de ser más completo y potente, con funciones para obtener el tiempo que transcurre entre dos eventos, como si fuera un cronómetro, por así decirlo.

Éste componente ha sido utilizado en la elaboración de Medusa, con el propósito de medir el tiempo que transcurre al ejecutar una consulta, y aunque también se podría haber utilizado el TTimer, la lógica utilizada con Accutime resulta mucho más intuitiva y simple.
Accutime se puede obtener en [W3ACU].

4.4.1 Instalación paso a paso

1. Descomprimimos el fichero accutime.zip en cualquier carpeta que deseemos.
2. Puesto que existen dos versiones, una para Delphi 3 y otra para Delphi 4, ejecutamos el paquete diseñado para la posterior, llamado *acutime.dpk*.
3. Al abrirse automáticamente Delphi nos mostrará una ventana como la de la Figura 4.4, en la que nos pregunta si queremos reconvertir el paquete al nuevo formato, pulsamos en sí (*Yes*).
4. Al igual que ya hicimos con los componentes SynEdit compilamos el paquete y después lo instalamos.
5. Una vez instalado el componente podremos acceder a él desde la pestaña System de la VCL.

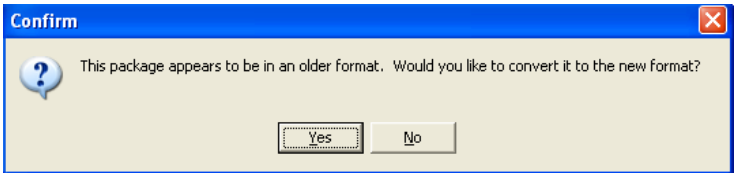


Figura 4.4: Conversión del paquete Accutime

4.5 FlyTreeViewPro Suite

La suite de componentes FlyTreeView representa la más rápida tecnología disponible en el mercado para la VCL de Delphi. Están basados en la tecnología *RAPID* sin utilizar la arquitectura MS TreeView y consiguen velocidades realmente sorprendentes con respecto a ésta.

Han sido incluidos para el desarrollo del árbol de visualización de objetos de la BD, que ya describiremos más ampliamente en el Capítulo 5.4.2.1. Puesto que puede haber ciertos objetos, como sinónimos o vistas, que existan en grandes cantidades, necesitábamos la mayor velocidad en la generación del árbol. Para hacernos una idea podemos ver los resultados de los tests en la Tabla 4.1. La *suite* FlyTreeViewPro la podemos obtener en [W3FLY].

Operation	Standard MS TreeView	FlyTreeView RE/RapidTree
Without sorting (SortType=stNone)		
Adding 20,000 nodes	~5 sec	0.2 sec
Clear 20,000 nodes	~1min 38 sec	0.12 s
Alpha sorting 20,000 nodes	~6 sec	0.27 sec
Without sorting (SortType=stNone)		
Adding 20,000 nodes	~37 sec	0.45 sec
Clear 20,000 nodes	~1min 38 sec	0.12 s
Alpha sorting 20,000 nodes	~7 sec	0.2 sec

Tabla 4.1 Tests de los componentes TreeView

4.5.1 Características

Algunas de las características principales de FlyTreeView son:

- Rompe la barrera de 65535 nodos visibles (limitación de MS TreeView).
- Soporte completo para la multiselección, Drag´n´Drop y el AutoScroll.
- Unido a la funcionalidad de los estándar Grid y TreeView.
- Configurable en tiempo de diseño y ejecución con un editor potente de propiedades.

- Método *AssignTo* para asignar el control completo a otro control.
- Capacidad para inspeccionar y traducir la entrada y la salida de datos con los eventos *OnGetNodeData* y *OnValidateNodeData*.
- Posibilidad de guardar y cargar la imagen del árbol como si fuera una plantilla.
- La vista en árbol está organizada en columnas, en las cuales cada celda puede ser personalizada individualmente.

4.5.2 Los componentes

Los componentes visuales que componen la familia FlyTreeView son:

- `TRapidTree.` Fue el primer componente creado con la tecnología RAPID. Es completamente compatible con las funcionalidades de TCustomGrid y TTreeView a pesar de utilizar objetos TFlyNode en vez de los estándares TTreeNode.
- `TISPlugEdit.` Editor multisección con diferentes caminos de entrada de datos. Permite la edición de más de 45 tipos de datos simples (enteros, moneda…), más datos complejos (Dirección IP, Fecha, Colores…) y sus combinaciones.
- `TFlyTreeViewPro.` Componente con todas las características indicadas anteriormente.
- `TPropertiesTreePro.` Inspector de objetos en tiempo de ejecución que soporte diferentes estilos de dibujo y agrupamiento de propiedades.

4.5.3 Instalación paso a paso

1. Extraemos el contenido del archivo ejecutable en cualquier carpeta que deseemos (por ejemplo la carpeta de Delphi/FlyTreeView).
2. Una vez que termina la extracción nos aparece un diálogo preguntándonos si queremos compilar e instalar automáticamente los paquetes, aceptamos y elegimos la versión de Delphi que utilicemos.
3. La instalación añade automáticamente la ruta de búsqueda en la opción *browsing path* de la pestaña *Library* de las opciones del entorno de Delphi, con lo que la instalación se completará así de simple.

4.6 La familia de componentes de Developer Express

Developer Express posee una amplia gama de componentes que podría casi compararse con la propia VCL de Delphi. Muchos y muy variados son los componentes que nos ofrecen, desde los estándares ListView, Grid o TreeView hasta componentes más específicos para realizar diagramas de flujo o calendarios. Los productos de Developer Express los podemos obtener en [W3DEV].

4.6.1 Los componentes

Dentro de la gama de componentes de Developer Express hemos utilizado en el desarrollo de Medusa las siguientes *suites*:

- `Express Bars.` Es una completísima colección de componentes para la realización de menús y barras de herramientas.
- `Express Forum Library`. Colección de controles basados en los de la VCL de Delphi para las tareas más comunes.
- `Express Printing System`. Crea informes sobre casi cualquier tipo de componente, ya sea de la VCL de Delphi o de otros componentes de Developer Express.
- `Express Quantum Grid`. Quizás el más potente VCL Grid para Delphi. Posee multitud de nuevas características, innumerables capacidades extras de configuración, además de un aspecto elegante y moderno.
- `Express Vertical Grid`. Control de Grid invertido, que muestra la rejilla de resultados en formato vertical, además de poseer la mayoría de características suplementarias de su hermano mayor el *Quantum Grid*.

Para una mayor información visite [W3IDE].

4.6.2 Instalación paso a paso

Vamos a detallar la instalación de la suite de componentes *ExpressBars* puesto que el resto de suites siguen un proceso de instalación idéntico:

1. Ejecutamos el archivo setup.exe.
2. Aparecerá una pantalla de bienvenida, simplemente pulsamos en *Next*.
3. Aceptamos los términos de la licencia y pulsamos de nuevo en *Next*.
4. Podemos ver ahora las distintas opciones de instalación en la ventana de la Figura 4.5. Las marcamos todas y continuamos.
5. Se muestra un diálogo preguntándonos si queremos incluir en el *path* la ruta de los archivos fuente, la marcaremos si tenemos pensado modificar los archivos fuente.
6. Por último nos aparecerán los directorios destino de la instalación tal y como se muestra en la Figura 4.6.

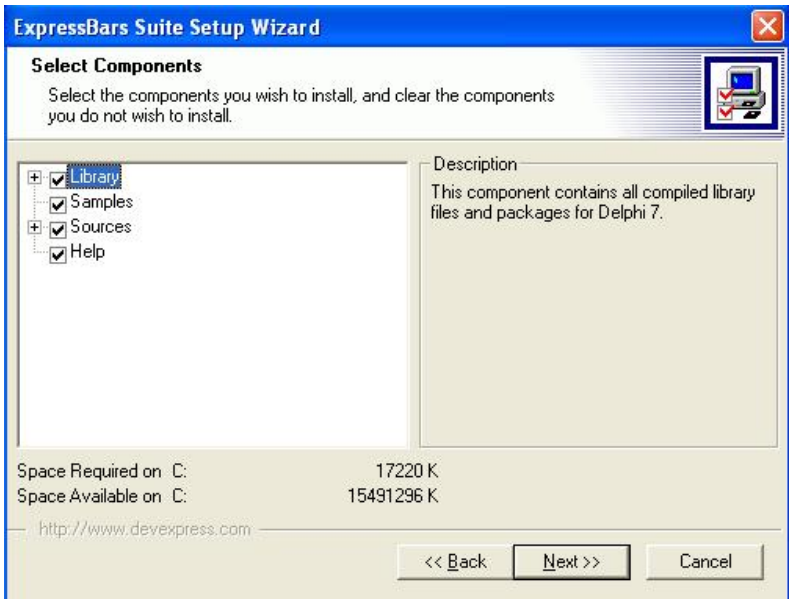


Fig 4.5: Opciones de instalación de la *suite* de componentes ExpressBars

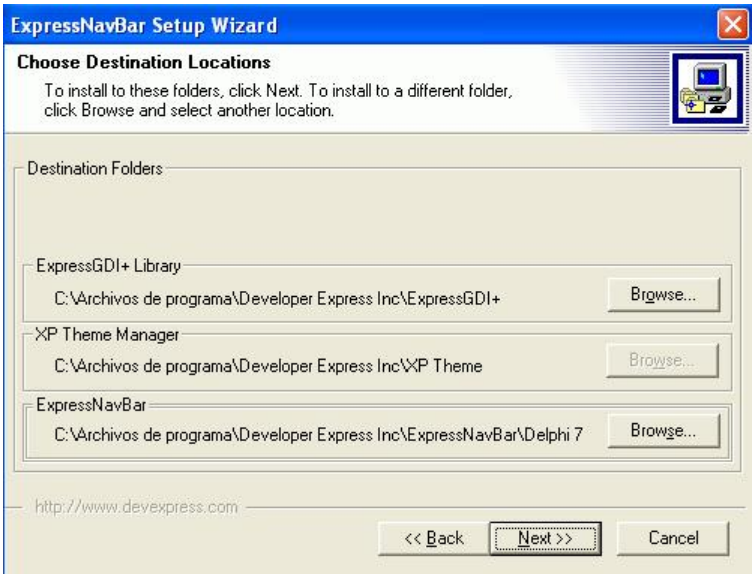
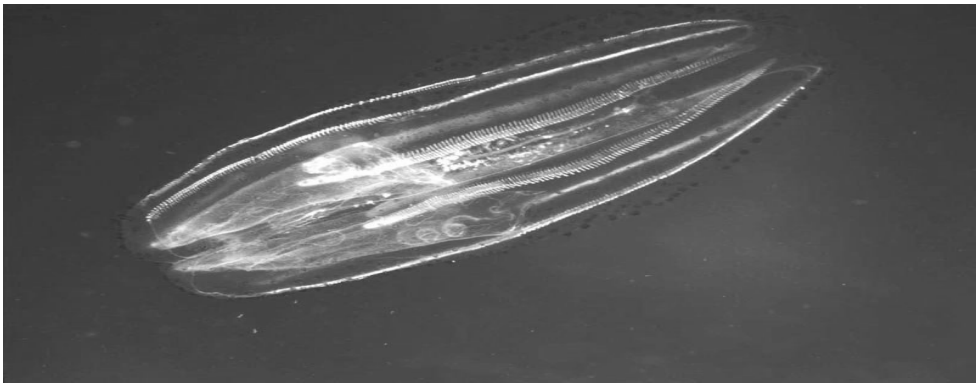


Figura 4.6: Carpetas destino de la instalación del componente ExpressNavBar

4.6.2.1 Instalación del enlace para impresión de QuantumGrid

Vamos a necesitar también la instalación del paquete *dxPScxGridLnkD7.bpl*, que nos servirá para la impresión de nuestra rejilla de resultados. Para ello nos vamos al menú de *Delphi Project*, pinchamos alternativamente sobre *Options* y después *Packages*, y añadiremos el paquete *ExpressPrinting System Cross Platform Library* y luego el paquete de diseño *Printing System ReportLink ExpressQuantumGrid 5.*



5

Desarrollo

El desarrollo de Medusa se ha basado en la mejora. Mejorar en características, en facilidad de uso, incrementar las posibilidades, mejorar la potencia, la estabilidad, simplemente, mejorar. En este capítulo, describiremos los detalles que han hecho que Medusa evolucione, además de documentar mejor su desarrollo para facilitar futuras modificaciones.

5.1 Preámbulo

En este capítulo vamos a tratar de dar una visión aproximada de la estructura y funcionamiento de nuestra aplicación. Empezaremos dando algunos conceptos sobre los componentes de programación que hemos utilizado y algunas nociones de Oracle que nos ayudarán a comprender cómo hemos implementado las distintas partes de *Medusa 2*. Tras esto enumeraremos los formularios que forman *Medusa 2* y describiremos los archivos externos que son necesarios.

Este proceso de desarrollo de una aplicación empezó con un análisis de requerimientos, en el que también fuimos encontrando pequeños errores (*bugs*) que fuimos arreglando o que desaparecían al cambiar el diseño. Dedicaremos un apartado a describir algunos de los fallos que encontramos en la primera versión.

En el Apartado 4 de esta sección mostraremos algunos diagramas UML que nos han ayudado tanto a comprender la primera versión de Medusa, como a elaborar nuestra aplicación.

Por último y puesto que estamos ante un proyecto "de mejora" no vamos a describir la totalidad de las funciones que posee Medusa, hablaremos de aquello que resulta nuevo, de lo que hemos modificado para darle mayor funcionalidad al resultado final.

5.2 Estructura de Medusa

Medusa contiene multitud de formularios y archivos asociados. Básicamente, podemos decir que *Medusa* se estructura en nueve partes, que son:

1. Ventana principal.
2. Ventana hija.
3. Ventanas de configuración.

4. Ventanas relacionadas con la edición.
5. Ventanas relacionadas con los objetos de una BD Oracle.
6. Ventanas relacionadas con características de la BD.
7. Ventanas para el control de transacciones.
8. Ventanas para la elaboración de informes.
9. Contenedor global de objetos.

5.2.1 Componentes de programación empleados

En esta sección, se describen brevemente los componentes más importantes empleados en los formularios de *Medusa*. Se han empleado multitud de componentes, mayoritariamente de las paletas *Standard*, *Additional*, *Win32* y *Data Access* de Delphi, los componentes de DOA, componentes de SynEdit, componentes de FlyTreeViewPro y por último de las suites de Developer Express.

A continuación haremos una breve descripción sobre ellos, que después nos ayudará a comprender cómo funcionan los distintos formularios de Medusa 2.

5.2.1.1 Componentes de la VCL de Delphi

Son los componentes de programación que vienen incluidos con el IDE de Delphi. Abarcan una gran cantidad de desarrollos posibles y nos van a servir para realizar las tareas más comunes de la aplicación. Algunos de los más importantes que hemos utilizado los describiremos brevemente a continuación.

5.2.1.1.1 TPopupMenu

El `TPopupMenu` es un componente perteneciente a la paleta *Standard* que utilizaremos para definir los menús que emergen al pulsar, sobre otro determinado componente, con el botón derecho. Una vez que lo añadimos a un formulario podemos definir el conjunto de acciones que

tendrá el menú, así como los procedimientos asociados a cada *item* del menú. Con él añadiremos una mayor cantidad de opciones de configuración del entorno, sin tener que pasar por la ventana de configuración propiamente dicha.

5.2.1.1.2 TImageList

Contenedor de imágenes. Almacenará los iconos que luego utilizaremos para hacer que el interfaz de usuario sea más intuitivo y atractivo. Nos da la posibilidad de compartirlo con otros formularios, de manera que podamos utilizar el contenedor de imágenes desde cualquier formulario, siempre y cuando éste tenga el formulario que contiene el `TImageList` en su lista de usados (*uses*).

5.2.1.1.3 TPageControl

Control de páginas utilizado para almacenar las páginas que pueden estar a la vez en el mismo control. En el formulario de configuración, por ejemplo, lo utilizamos para contener las páginas que forman todas las opciones de configuración estática de la aplicación.

5.2.1.1.4 TActionList

Mantiene una lista de acciones que pueden ser usadas por componentes y controles, tales como opciones de menú y botones. Nos permitirá centralizar algunas acciones que son utilizadas en varias ocasiones, de manera que simplemente tendremos que enlazarlas o llamar a la acción correspondiente.

5.2.1.1.5 TDataSource

Este componente de la paleta *Data Access* suministra un interfaz entre un conjunto de datos (*dataset*) y controles *data-aware* en un formulario. En nuestro proyecto lo vamos a utilizar como intermediario entre un conjunto de datos Oracle (`TOracleDataSet`) y una rejilla de resultados. Lo que mostrará la rejilla o tabla de resultados serán los datos que provengan del efecto de abrir el conjunto de datos Oracle.

5.2.1.1.6 TTreeView

Representa una ventana que muestra jerárquicamente una lista de datos, tales como archivos y directorios de un disco, las entradas de un índice o los títulos de un documento. Cada nodo del árbol consistirá en una etiqueta y opcionalmente una imagen. Éste nodo podrá tener una sublista de nodos dependientes de él. El usuario de la aplicación podrá contraer o desplegar la lista de subnodos.

5.2.1.1.7 TExcelApplication

Componente de conexión a la aplicación Excel de Microsoft. Se utiliza en conjunción con los componentes `TExcelWorkbook` y `TExcelWorksheet`, que representan un libro y una hoja en Excel, respectivamente. Utilizaremos dichos componentes para exportar los resultados a este formato mundialmente extendido.

5.2.1.1.8 TXMLDocument

Componente que podremos encontrar en la pestaña *Internet* de la VCL de Delphi. Gracias a este componente podremos disponer de los resultados en uno de los formatos más actuales y con gran repercusión en el mundo de las bases de datos en Internet. De esta forma podremos intercambiar los resultados con otros usuarios en un formato altamente estándar, que se puede integrar con facilidad en cualquier aplicación.

5.2.1.2 Componentes de Direct Oracle Access

Son los componentes que utilizamos para acceder al SGBD y con ellos conseguiremos un diálogo bidireccional entre nuestra aplicación y Oracle. Aunque el paquete contiene muchos más componentes que utilizamos, sólo vamos a hablar sobre los componentes visuales usados.

5.2.1.2.1 TOracleLogon

Componente capaz de crear un formulario de conexión a una base de datos Oracle. Podremos personalizar las cadenas de conexión, guardar o no un historial de conexiones, o si debemos de guardar el *password* junto con el usuario. En conexión con una sesión de Oracle, crearemos una sesión con el usuario/*password* que hayamos introducido en el formulario de conexión.

5.2.1.2.2 TOracleSession

Componente que representa una sesión de Oracle. Se pueden definir características propias de las sesiones, como el nivel de aislamiento, el tipo de conexión (normal, *SYSDBA*, *SYSOPER*) o si queremos realizar un *rollback* al salirnos sin confirmar la transacción.

5.2.1.2.3 TOracleDataset

Conjunto de datos que está conectado con una sesión de Oracle. El contenido del atributo *SQL* se lanza al abrir el *Dataset*. El conjunto de datos que resulta de lanzar la sentencia podrá ser utilizado después por otros componentes de visualización de datos, como el `TGrid` (rejilla de resultados).

5.2.1.2.4 TOracleScript

Similar al `TOracleDataSet`, la diferencia entre ambos estriba en que éste se utiliza para lanzar *scripts* a Oracle, mientras que el anterior se utiliza para lanzar consultas. El atributo *Lines* es el encargado de almacenar las líneas del *script*.

5.2.1.3 Componentes de SynEdit

Vamos a describir tan sólo el componente `TSynEdit`, puesto que es el único que hemos utilizado para esta segunda versión de la aplicación. Los componentes `TSynSQLSyn, TSynExporterHTML, TSynAutocompletionProposal, TSynAutoComplete`, también están incluidos en formularios de Medusa, pero no vamos a hacer énfasis en ellos pues no nos ha hecho falta añadir ninguno de ellos.

5.2.1.3.1 TSynEdit

Es un componente que a primera vista puede parecer similar al `TMemo`, pero que posee muchas otras características, como resaltador de sintaxis, marcadores de texto o capacidad de búsqueda de cadenas. Utilizaremos `TSynEdit` con la sintaxis de Oracle definida en cualquier formulario de la aplicación que haga falta un editor de texto para lanzar consultas, sentencias o *scripts*.

5.2.1.4 Componentes de FlyTreeViewPro: TRapidTree

Componente muy similar en apariencia y características a `TTreeView`, permite, por ejemplo, definir distintas imágenes según si el nodo está o no seleccionado, definir en tiempo de diseño el estado del nodo entre un conjunto de estados posibles o guardar la configuración del árbol en disco. Para mayor información sobre este componente y el resto de la *suite*, consultar la Sección 4.5.

5.2.1.5 Componentes de Developer Express

Developer Express cuenta con multitud de *suites* de componentes, así que vamos a definir cada componente independientemente de a cual pertenezca. En apartados siguientes explicaremos las funciones que desempeñan en *Medusa 2*.

5.2.1.5.1 TdxBarManager

Componente completísimo para la gestión de la interfaz de usuario. Viene a sustituir a `TMainMenu` de la VCL de Delphi. Con él podremos construir barras de herramientas mucho más configurables y llamativas, además de darnos la posibilidad de configurar en tiempo de ejecución el aspecto y comandos que configurarán el menú. Medusa 1 no lo utilizaba, en su lugar usaba el componente TCoolbar de la VCL de Delphi, y los resultados fueron unas posibilidades de configuración nulas, no se podían mover las barras de herramientas, ni desanclarlas y por supuesto, ningún tipo de personalización de los menús.

Una vez que añadamos uno de estos componentes a un formulario principal podremos definir los comandos que especificarán el menú. Para ello añadiremos `TdxBarSubitems` para definir submenús, `TdxBarLargeButtons` en el caso de que necesitemos botones grandes y `TdxBarButtons` si los necesitamos pequeños, y así hasta un sin fin de opciones que incluyen barras de progreso, *Combobox* de fechas, de colores, de edición, vistas en árbol o contenedores de elementos.

5.2.1.5.2 TdxBarPopupMenu

Componente para configurar los menús emergentes. Es muy similar al `TPopupMenu`, pero se basa en enlaces a elementos del gestor de barras, de manera que resultará necesario para los menús que surgen al pulsar sobre algún componente Developer Express.

5.2.1.5.3 TcxSplitter

Basado en `TSplitter`, añade algunas propiedades interesantes como *HotZone*, que permite disponer de algunos aspectos muy atractivos visualmente.

5.2.1.5.4 TcxGroupBox

Componente que vamos a utilizar como contenedor de otros componentes. Podremos incluir dentro de un `TcxGroupBox` prácticamente cualquier otro componente visual, incluido otro `TcxGroupBox`. De esta manera podremos agrupar componentes que hacen funciones similares o que ocupan la misma porción de pantalla y que nos conviene mover como uno sólo.

5.2.1.5.5 TcxGrid

Uno de los elementos de mayor potencia que jamás hemos utilizado es el `TcxGrid`. Va más allá de ser una simple rejilla de resultados, con infinidad de opciones de configuración, compuesto por `TcxGridLevels` y `TcxGridDBTableViews`, tan sólo podremos utilizar un conjunto de sus posibilidades, puesto que Medusa lo utiliza en *runtime* [1] y no podemos configurar los niveles y la forma de visualización de los datos en tiempo de diseño. Para un mayor conocimiento

[1] Tiempo en que el usuario está ejecutando la aplicación

de las posibilidades ver la Sección 7.4.2, donde hablaremos del manejo de los resultados devueltos en un `TcxGrid`.

5.2.1.5.6 TdxComponentPrinter

En la creación de informes de *Medusa 2* utilizamos `TdxComponentPrinter` como el enlace entre los resultados de un `TcxGrid` y la visualización de un informe. Con una gran cantidad de opciones de configuración del informe, necesitaremos indicarle a qué debe de enlazar con la posibilidad de impresión. También posee formularios de configuración de página, de márgenes o de impresión propios que habrá que establecer adecuadamente.

5.2.1.5.7 TdxPSFileBasedExplorer

Explorador basado en archivos que nos ayudará en la construcción de nuestro explorador de informes. Le indicaremos la ruta en la que guardamos nuestros informes y en el caso de algún cambio guardaremos en el registro la posible nueva configuración.

5.2.1.5.8 TdxPSEngineController

Controlador del motor del sistema de impresión. Carga y guarda en el registro las propiedades que cambie el usuario en la previsualización e impresión de los informes.

5.2.1.5.9 TdxDBVerticalGrid

Cuadrícula de resultados que tiene la particularidad de mostrar los resultados de forma vertical. El funcionamiento y configuración es prácticamente similar al `TcxGrid`, aunque no posee las características de definición de niveles, comportamiento, estilos o vistas, entre otros.

5.2.2 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.

5.2.2.1 Módulos y formularios de trabajo

La primera versión de Medusa ya poseía una buena cantidad de formularios, pero puesto que hemos añadido muchos y modificado prácticamente todos los que ya existían, vamos a describir brevemente los más importantes y para qué son utilizados. La extensión *.dfm* corresponde a un formulario de Delphi (Delphi form) y la extensión *.pas* a un archivo pascal (Pascal file), o lo que es lo mismo, un archivo de código fuente relacionado con un formulario.

- 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). Típico formulario *acerca de* que muestra los datos del autor, tutor y el programa.
- FrmConfiguracion (UConfiguracion.dfm, UConfiguracion.pas). Formulario que permite personalizar el programa, accediendo a multitud de opciones.
- FrmConsultaDatos (UConsultadatos.dfm, UConsultadatos.pas). Formulario para consultar los datos de una tabla, sin la posibilidad de editarlos.
- FrmDescribe (UDescribe.dfm, UDescribe.pas). Formulario que describe las columnas de una tabla.
- 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*.
- FrmExplorador (UExplorador.dfm, UExplorador.pas). Formulario construido con tan sólo 3 componentes y que muestra el explorador de informes.
- FrmInforme (UInforme.dfm, UInforme.pas). Similar al editor puesto que permite la ejecución de sentencias, pero que permite la visualización y almacenamiento de informes con los resultados devueltos.

- FrmModificarDatos (UModificardatos.dfm, UModificardatos.pas). Formulario que muestra las filas de una tabla y que permite la inserción, eliminación o modificación de los datos.
- FrmPersonFiltro (UPersonfiltro.dfm, UPersonfiltro.pas). Permite personalizar el filtro del árbol de objetos, de manera que sólo se muestren los objetos que cumplan la condición establecida.
- FrmPrincipal (UPrincipal.dfm, UPrincipal.pas). Formulario que permite el acceso a todas las opciones iniciales del programa. El punto de partida del usuario.
- FrmPropiedades (UPropiedades.dfm, UPropiedades.pas). Formulario capaz de mostrar en un *grid* o rejilla vertical las propiedades de cualquier objeto de una base de datos Oracle.
- FrmRenametable (URenametable.dfm, URenametable.pas). Formulario muy simple que pide la introducción del nuevo nombre al que renombrar la tabla.
- FrmSavePoint (USavePoint.dfm, USavePoint.pas). Formulario que recoge el nombre del punto de guarda de la transacción.
- FrmSplash (USplash.dfm, USplash.pas). Formulario que muestra la ventana de presentación del programa.
- FrmVerCodigo (UVerCodigo.dfm, UVerCodigo.pas). Formulario que muestra el código asociado a procedimientos y funciones.
- FrmVisualIndexes (UVisualIndexes.dfm, UVisualIndexes.pas). Formulario que facilita la creación de índices de forma visual.
- FrmVisualJob (UVisualJob.dfm, UVisualJob.pas). Formulario de creación visual de trabajos.
- FrmVisualLibrary (UVisualLibrary.dfm, UVisualLibrary.pas). Permite definir una librería de funciones para su utilización en bloques PL/SQL.
- 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 el esquema básico del código.

- 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.
- FrmVisualTabla (UVisualTabla.dfm, UvisualTabla.pas). Formulario que permite crear tablas de forma visual. Se pueden definir las columnas de la tabla, las restricciones y las comprobaciones (*checks*). También tiene soporte para definir tabla temporales (*temporary tables*) y tablas agrupadas (*clustered tables*).
- FrmVisualTablespaces (UVisualTablespaces.dfm, UvisualTablespaces.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.

5.2.2.2 Ficheros de datos

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

5.2.2.2.1 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 5.1 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 2*, al hacer clic en cualquier elemento del menú *Diccionario de datos*. Este menú se explica en el Manual de usuario, en la Sección 7.7.3.

```
-- 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 5.1: Ejemplo del formato de la consulta a la vista `ALL_TAB_COMMENTS`

5.2.2.2.2 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 5.2.

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

Listado 5.2: 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 el Listado 5.3, donde *nombre* es el texto que se sustituirá por las líneas inferiores.

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

Listado 5.3: Sintaxis de un elemento de la lista de autocompletado

5.2.2.2.3 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. Un ejemplo lo podemos observar en el Listado 5.4.

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

Listado 5.4: 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 5.1.

Figura 5.1: Lista de autocompletado visual filtrada por la letra "a"

5.2.2.2.4 Plantillas: Plantillas de código

Además de las funciones de autocompletado vamos a disponer en Medusa de un cómodo interfaz de plantillas. Organizadas por carpetas, completamente configurables, tendremos la posibilidad de añadir, eliminar o modificar a nuestro antojo las que vienen por defecto. El aspecto del árbol de plantillas es similar al que aparece en la Figura 5.2.

Aparte de ser completamente configurables, el uso de las plantillas de texto resulta realmente sencillo, bastará con un doble clic sobre el icono de una plantilla para añadirla a la sesión activa. De esta forma acciones como declarar variables, constantes o utilizar funciones SQL resultarán muchos más rápidas e intuitivas. El contenido de las carpetas de plantillas que vienen definidas por defecto será explicado en el Capítulo 7.5, donde describiremos también todas las características del uso de plantillas.

Todas las plantillas incluidas más las que defina el usuario se encuentran en *<raiz>\Plantillas*, y en el caso de que defina nuevas carpetas de plantillas también se encontrarán en este directorio con el nombre de la carpeta que se defina. El manejo de plantillas lo explicaremos en la Sección 7.5 del Manual de usuario.

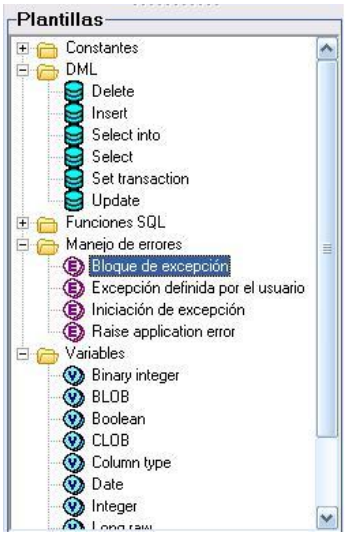


Figura 5.2 Árbol de plantillas de código

5.2.2.2.5 Favoritos

Todos los ficheros de texto guardados como favoritos se almacenarán en el directorio *<raiz>\Favoritos*, y en el caso de crear nuevas carpetas para organizar los favoritos, al igual que con las plantillas, los guardaremos en esta misma carpeta con el nombre que hayamos definido. El contenido de estos ficheros de texto serán sentencias o conjuntos de sentencias que hayamos guardado desde Medusa después de utilizarlas en alguna sesión.

Para acceder a los favoritos lo haremos desde el menú *Favoritos* de Medusa que como se muestra en la Figura 5.3 cargará todas las carpetas y ficheros que tengamos definidos.

Figura 5.3: Carga de los favoritos en Medusa

5.2.2.2.6 Filtros

El directorio <raiz>/<Filtros> viene definido al instalar *Medusa 2*, pero está vacío. Esto es así porque lo utilizaremos para guardar filtros que utilizaremos posteriormente para filtrar los resultados de una consulta. Para ello, nos bastará con pulsar sobre el botón de filtrado de la barra de navegación de la rejilla de resultados, establecer el filtro que queremos y guardarlo en disco, tal y como se explica en la Sección 7.4.2.2 del Manual de usuario.

Para una mayor organización hemos definido la carpeta <Filtros> como base, de manera que será la primera que nos salga al intentar guardar o cargar un nuevo filtro, y aunque no sea necesario el almacenamiento en esta carpeta, si resultaría muy recomendable utilizarla con este propósito. De esta forma, no tendremos problemas para recordar después donde guardamos el filtro o filtros que utilizamos.

5.2.2.2.7 Informes

El directorio <raiz>/<Informes> será nuestro punto de partida para almacenar todos aquellos informes que vayamos creando. Al instalar *Medusa 2* veremos que viene definida la carpeta <DBA> en la que encontraremos algunos informes de muestra que nos pueden servir como punto de partida para experimentar con el uso de informes.

En el momento que ya estemos familiarizados con la definición de informes, nosotros mismos podremos crear nuestros informes personalizados y almacenarlos en disco, de manera que con el explorador de informes podamos tener acceso a todos ellos.

El uso del explorador de informes, así como de la forma de generar nuevos informes será ampliamente detallado en el *Manual de usuario de Medusa 2*, Sección 7.7.5.

5.2.2.2.8 Ayuda

Por último, la carpeta <raiz>/<Help> tiene almacenado los archivos necesarios para la ejecución de la ayuda de Medusa. Aunque nosotros mismos podamos abrir el archivo *Medusa.hlp* de ayuda, sería de mayor ayuda hacerlo desde dentro de la aplicación. Simplemente pulsando F1 sobre cualquier formulario podremos tener acceso a la ayuda sobre dicho formulario, con lo que nos ahorraremos el tiempo de búsqueda.

5.3 Bugs & fixes

En este apartado vamos a hablar sobre algunos pequeños arreglos que hemos tenido que realizar a lo largo del desarrollo. La primera versión de Medusa resultó ser muy estable, pero como en cualquier aplicación, siempre se van encontrando algunos detalles que no funcionan adecuadamente.

Prácticamente casi ninguno de los fallos que hemos arreglado influía en la estabilidad del programa, por lo que conseguir eliminarlos ha resultado sencillo en la mayoría de los casos. Algunos de estos fallos no hemos necesitado arreglarlos, pues al cambiar la implementación, simplemente teníamos que intentar que no volvieran a darse, y que por supuesto no aparecieran otros nuevos.

En el siguiente listado enumeramos algunos de los fallos de los que hemos hablado y que bastará con ejecutar la primera versión y seguir los pasos que indicamos para comprobar que realmente existen.

- No se guarda la configuración de las barras de iconos, al iniciar el programa siempre las pone activadas, aunque se hubieran desactivado en la ejecución anterior, además de que a menudo no estaban sincronizados (botón activo y barra no visible).
- La lista de páginas se vuelve a poner a 1(vuelve a contar desde el principio) cuando se pulsa una de ellas 2 veces, es decir, no solo cambiar de *tab*, sino que se selecciona.

- Se activan los botones de cortar y copiar sin que hubiera texto seleccionado si hay 2 sesiones abiertas y se cambia de una a otra.
- Al desconectar sigue saliendo en la barra de estado el último usuario conectado.
- Al exportar a texto (*txt)* sale un mensaje "Ahora" por cada columna del resultado que vayamos a exportar.
- La creación visual de secuencias no funciona correctamente cuando no pongo un valor máximo o mínimo.
- Al definir los argumentos de un procedimiento o función, en el creador visual correspondiente, si intentamos borrar un nombre que ya hubiéramos escrito, siempre se quedará la primera letra escrita en el campo *nombre*, aunque hayamos borrado todo el contenido.
- El constructor visual de procedimientos y funciones tiene activados cuadros de edición cuando no deberían de estarlo. Al eliminar todos los argumentos los cuadros siguen activos, así como el botón eliminar, sin que haya nada que eliminar.
- La creación visual de vistas da un problema al crear una vista materializada con refresco en *nunca*, pues no deja aceptar diciendo que introduzca el periodo, que lógicamente no se puede introducir.
- En el formulario de creación visual de índices existe un problema con la activación de los botones de agregar y quitar columnas. A veces se quedan activados sin que haya ninguna columna sobre la que realizar la acción.
- Al aparecer el formulario `Acerca de` el programa no responde y no se puede cerrar el formulario ni el programa.
- No se crean los favoritos, ni los archivos ni las carpetas, tan solo se muestran en el árbol, pero no se crean en disco, ni tampoco se eliminan adecuadamente.

5.4 Diagramas UML

En toda elaboración de un sistema es conveniente formalizar un análisis de los requisitos funcionales que debe de cumplir la aplicación final. Para la documentación del sistema vamos a utilizar diagramas UML (Unified Modeling Language), más concretamente diagramas de casos de uso.

Hemos elegido UML porque nos ayudará a capturar la idea del sistema y será una forma estándar de comunicación con quien se quiera involucrar en el proceso de desarrollo. También nos dará una visión general de las funciones que hemos creído necesarias para nuestro producto.

En cuanto a los distintos diagramas, UML comprende diagramas de clases, de secuencia, de casos de uso, de estados, de colaboración… pero en nuestro sistema sólo vamos a basarnos en los diagramas de casos de uso. Decidimos tomar esta decisión pues el objetivo del proyecto no es realizar un análisis y diseño de un sistema, sino mejorar un sistema ya implementado, y con este objetivo, y sin que la primera versión contara con documentación UML, el proceso de análisis y diseño se haría tan largo que impediría la implementación, que es en realidad el fin último de nuestro proyecto.

A continuación vamos a describir brevemente los diagramas de casos de uso, sus objetivos y sus elementos fundamentales.

5.4.1 Diagramas de casos de uso

Los objetivos de los diagramas de casos de uso son los siguientes:

- Capturar los requisitos funcionales del sistema y expresarlos desde el punto de vista del usuario.
- Guiar todo el proceso de desarrollo del sistema de información.

Los casos de uso proporcionan, por tanto, un modo claro y preciso de comunicación entre cliente y desarrollador. Desde el punto de vista del cliente proporcionan una visión de "caja negra" del sistema, esto es, cómo aparece el sistema desde el exterior sin necesidad de entrar en los detalles de su construcción. Para los desarrolladores, suponen el punto de partida y el eje sobre el que se apoya todo el desarrollo del sistema en sus procesos de análisis y diseño.

5.4.1.1 Descripción de la técnica

Un caso de uso es una secuencia de acciones realizadas por el sistema, que producen un resultado observable y valioso para un usuario en particular, es decir, representa el comportamiento del sistema con el fin de dar respuestas a los usuarios.

Aquellos casos de uso que resulten demasiado complejos se pueden descomponer en un segundo nivel, en el que los nuevos casos de uso que intervengan resulten más sencillos y manejables. Para especificar este comportamiento existen una serie de recomendaciones o técnicas que se aplican dependiendo del momento del desarrollo que se esté y de la complejidad del caso de uso. Puede ser desde una simple descripción textual que recoja un requisito funcional a una especificación del caso de uso, incluso un conjunto de diagramas.

5.4.1.2 Elementos fundamentales

Los diagramas de casos de uso presentan dos tipos de elementos fundamentales:

1. **Actores**. Un actor es algo o alguien que se encuentra fuera del sistema y que interactúa con él. En general, los actores serán los usuarios del sistema y los sistemas externos al que se esté desarrollando. Si se habla de usuarios, un actor es el papel que puede llevar a cabo en cuanto a su forma de interactuar con el sistema, es decir, un único actor puede representar a muchos usuarios diferentes y de la misma forma, un usuario puede actuar como actores diferentes.

2. **Casos de uso**. Un caso de uso representa el comportamiento que ofrece el sistema de información desde el punto de vista del usuario. Típicamente será un conjunto de transacciones ejecutadas entre el sistema y los actores. Para facilitar la comprensión de los casos de uso del sistema de información en el análisis, es posible agruparlos en paquetes según funcionalidades semejantes o relacionadas.

Además de estos elementos, un diagrama de casos de uso presenta *relaciones*. Las relaciones pueden tener lugar entre actores y casos de uso o entre casos de uso. La relación entre un actor y un caso de uso es una relación de *comunicación*, que indica que un actor interviene en el caso de uso.

5.4.2 Diagramas de Medusa 2

Los diagramas de casos de uso de nuestro proyecto están divididos en paquetes, pues la gran cantidad de ellos nos ha obligado a agruparlos por características de semejanza. El primer nivel se presenta en la Figura 5.4. Algunos paquetes poseen un segundo nivel, en las Figuras 5.7, 5.10 y 5.17 se describen los paquetes que forman este segundo nivel.

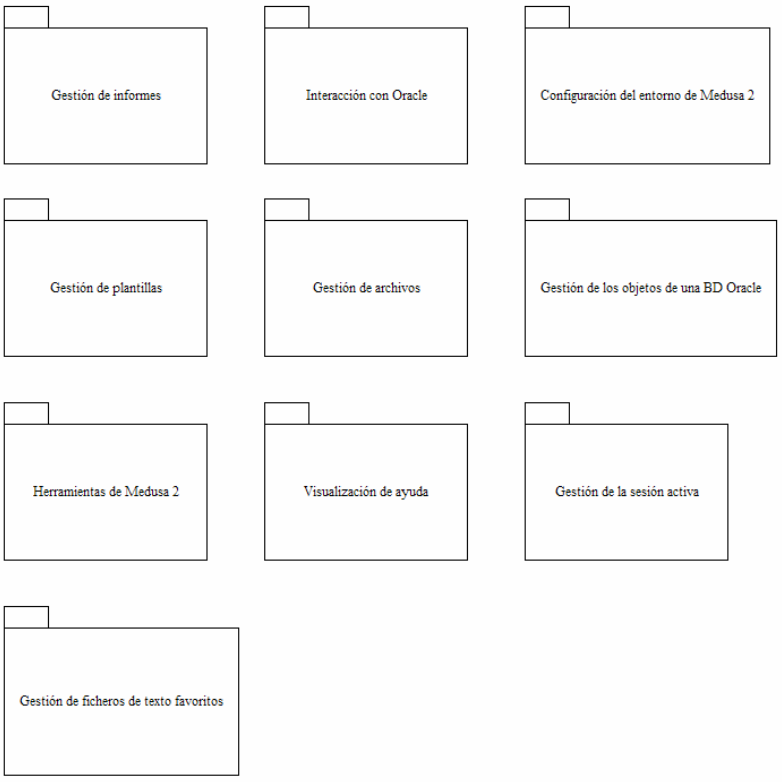


Figura 5.4: Primer nivel de paquetes

A partir de aquí vamos a ir mostrando en sucesivas figuras el contenido de cada paquete, en los que se simboliza un actor con el símbolo y un caso de uso con , como se especifica en UML.

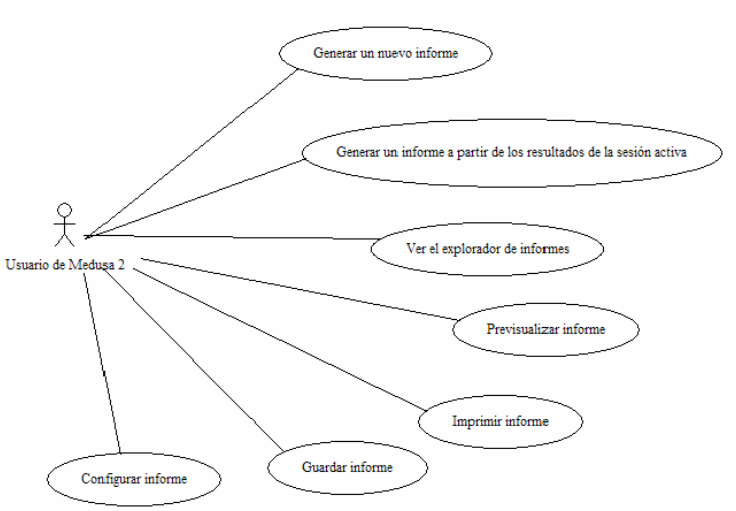


Figura 5.5: Diagrama de casos de uso del paquete gestión de informes

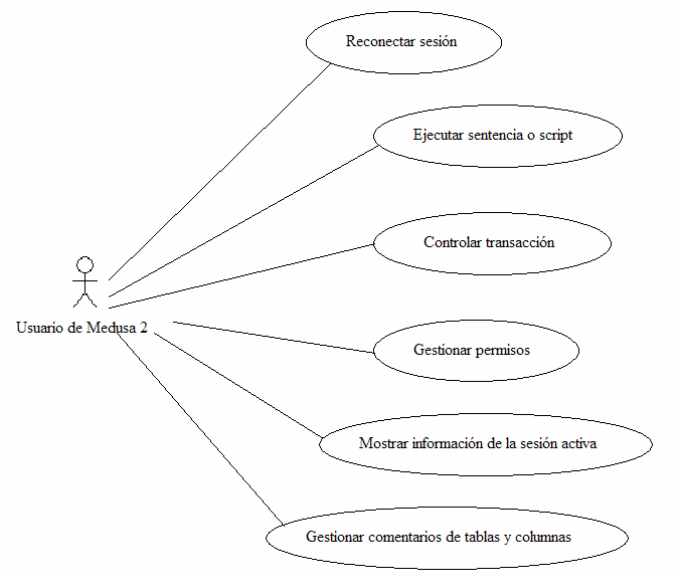


Figura 5.6: Diagrama de casos de uso del paquete interacción con Oracle

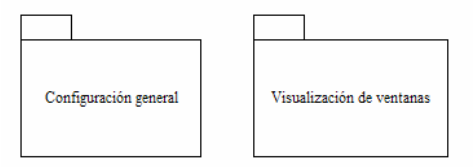


Figura 5.7: Contenido del paquete Configuración del entorno de Medusa 2

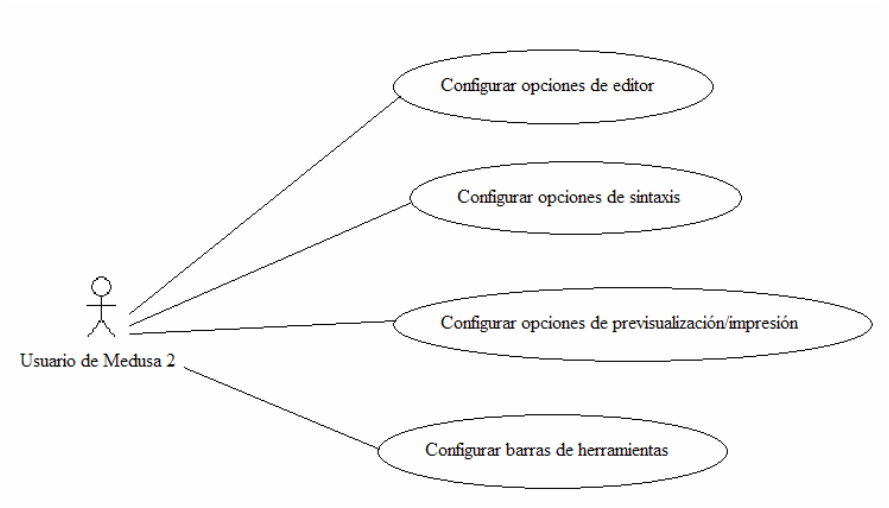


Figura 5.8: Diagrama de casos de uso del paquete Configuración general

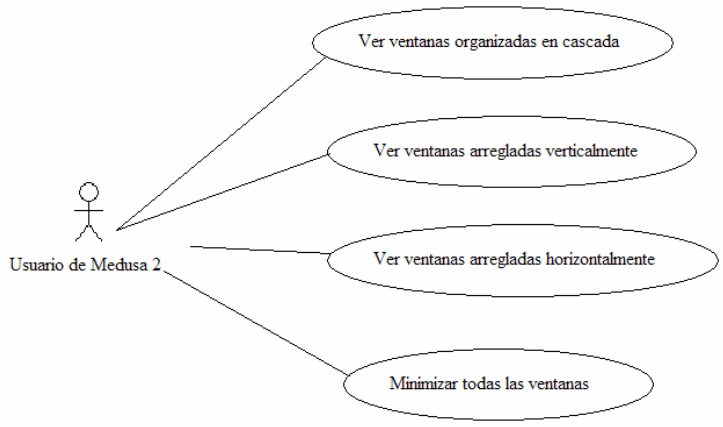


Figura 5.9: Diagrama de casos de uso del paquete visualización de ventanas

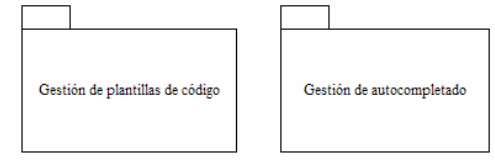


Figura 5.10: Contenido del paquete Gestión de plantillas

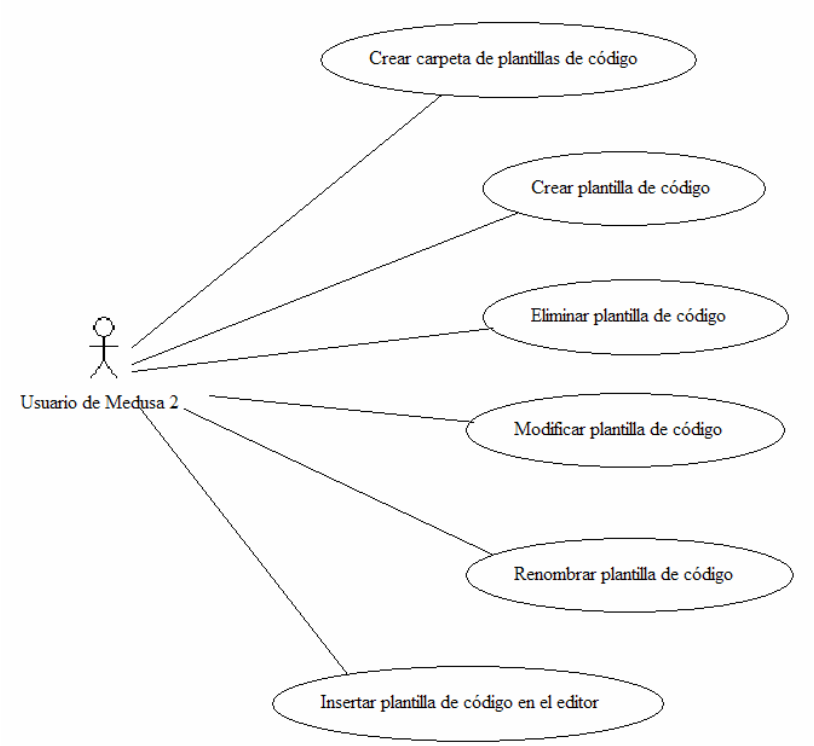


Figura 5.11: Diagrama de casos de uso del paquete Gestión de plantillas de código

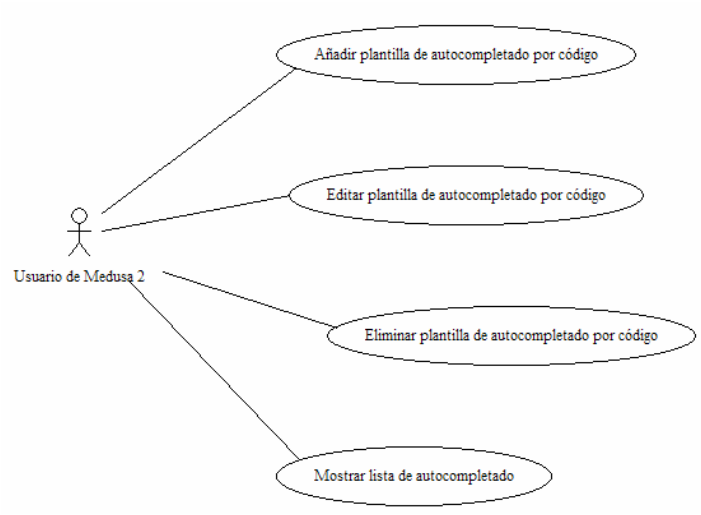


Figura 5.12: Diagrama de casos de uso del paquete Gestión de autocompletado

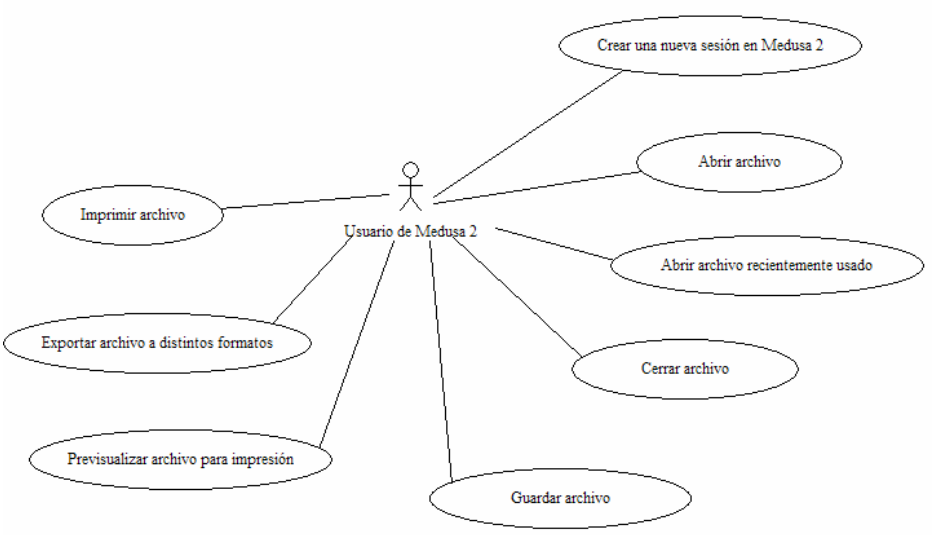


Figura 5.13: Diagrama de casos de uso del paquete Gestión de archivos

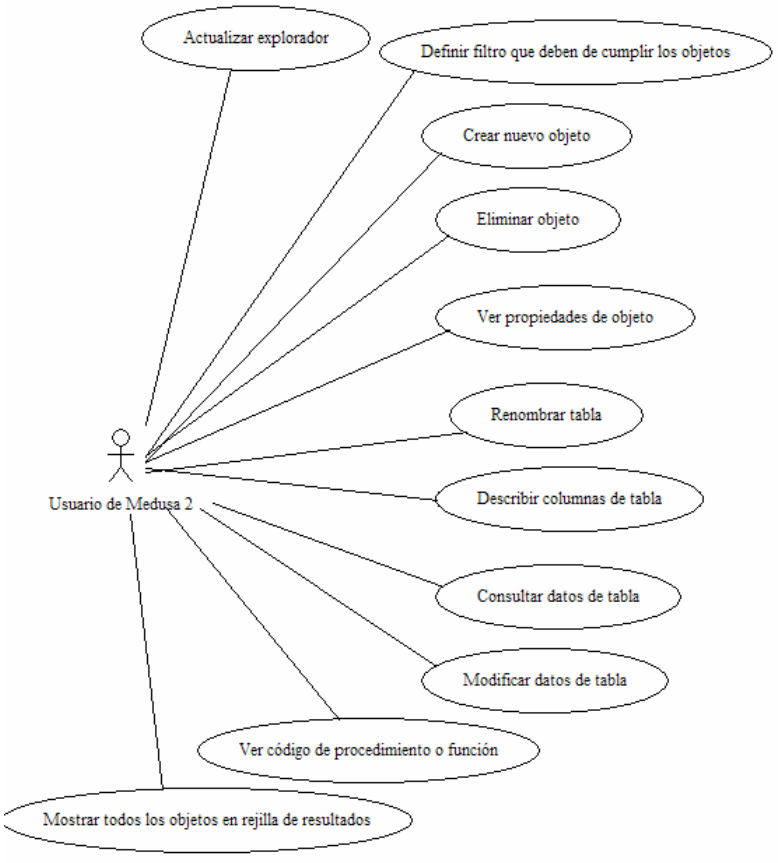


Figura 5.14: Diagrama de casos de uso del paquete Gestión de los objetos de una BD Oracle

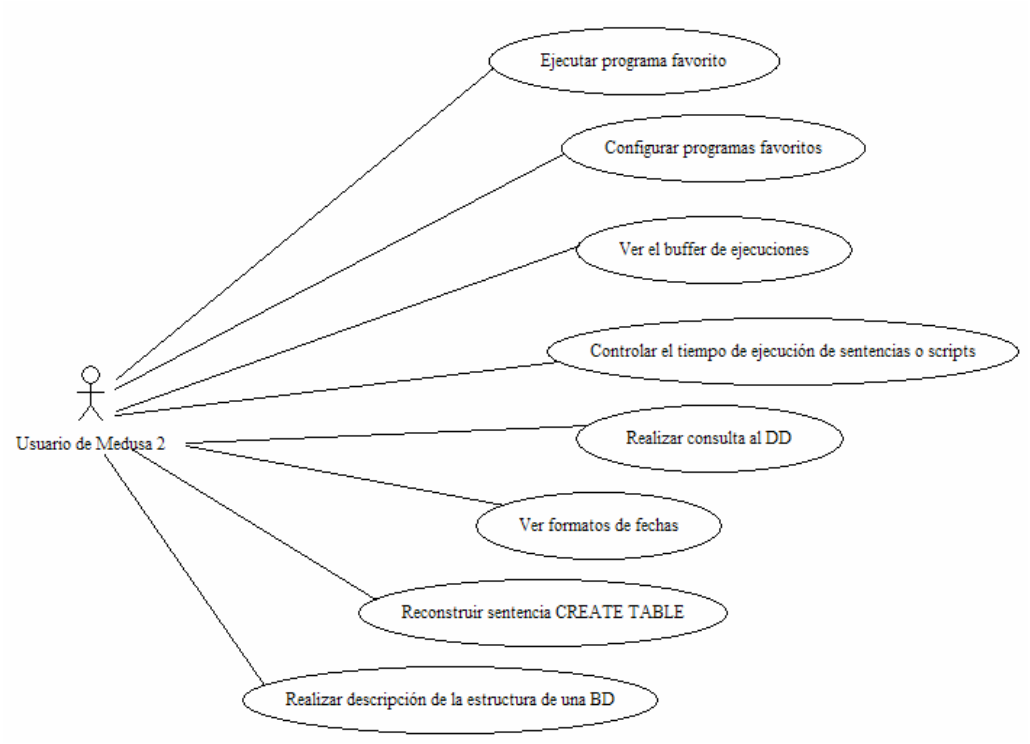


Figura 5.15: Diagrama de casos de uso del paquete Herramientas de Medusa 2

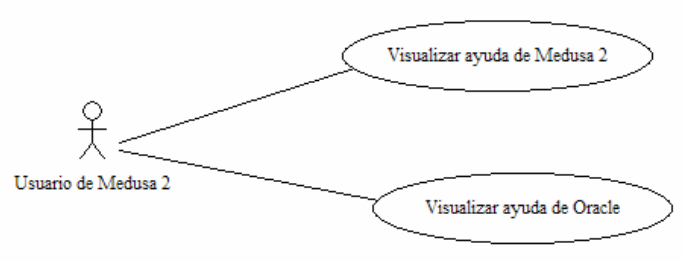


Figura 5.16: Diagrama de casos de uso del paquete Visualización de ayuda

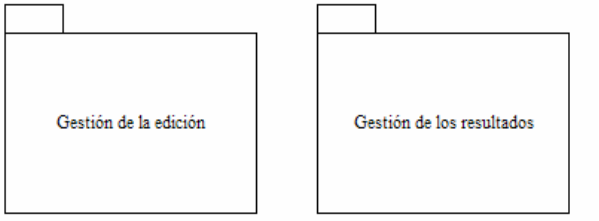


Figura 5.17: Contenido del paquete Gestión de la sesión activa

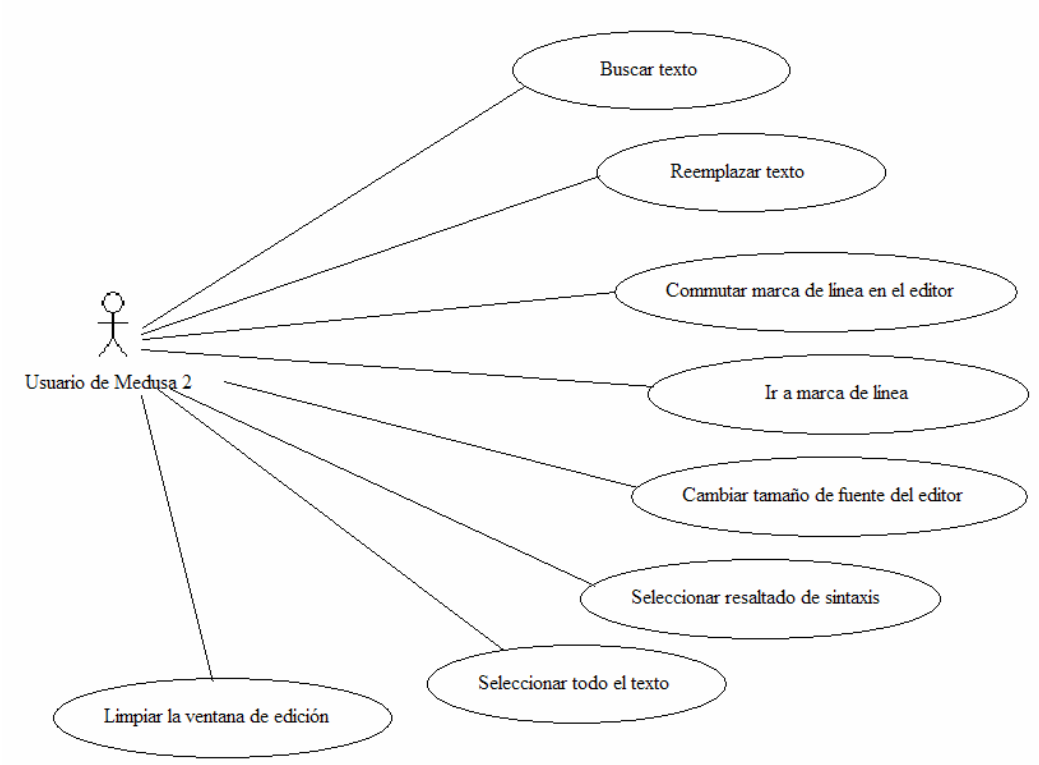


Figura 5.18: Diagrama de casos de uso del paquete Gestión de edición

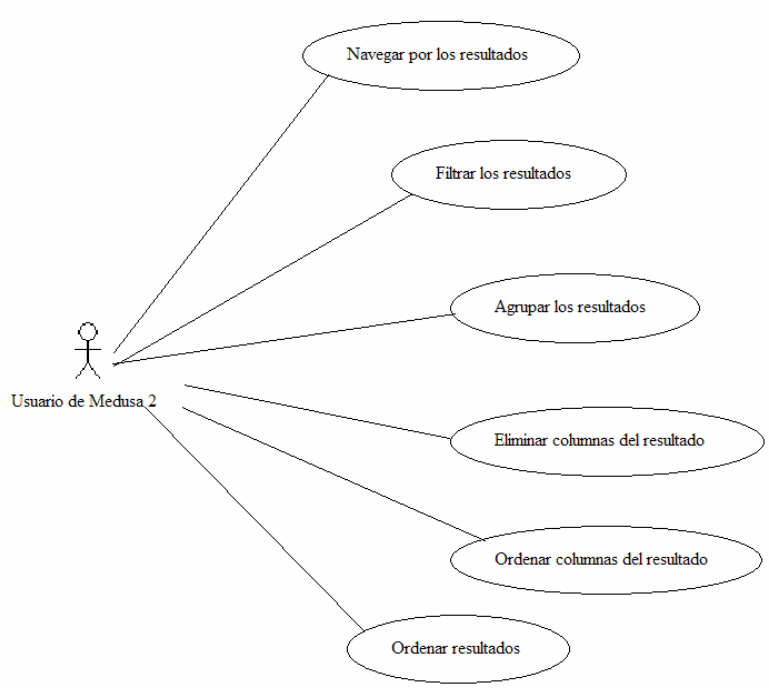


Figura 5.19: Diagrama de casos de uso del paquete Gestión de los resultados

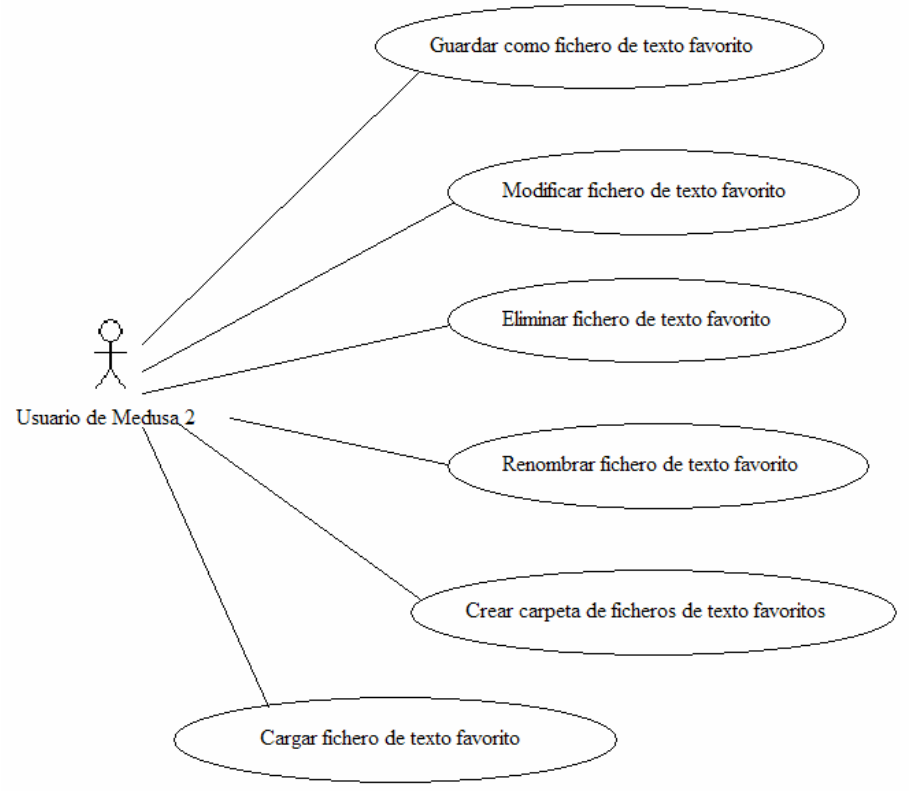


Figura 5.20: Diagrama de casos de uso del paquete Gestión de ficheros de texto favoritos

5.5 Ampliaciones, mejoras y nuevas funciones

En el desarrollo de *Medusa 2* se han ido modificando algunas características y se han añadido otras nuevas. En algunas ocasiones hemos tenido que volver a implementar algunas partes para acomodarlas a los nuevos componentes, pero siempre hemos respetado todas las características de que disfrutaba la primera versión.

La descripción de la aplicación completa la vamos a hacer en el *Manual de usuario* (Capítulo 7), donde haremos énfasis en cada parte de Medusa, independientemente de si es una característica de ésta o de la primera versión, con objeto de tener un Manual completo de *Medusa 2*. En los siguientes apartados vamos a dar detalles tanto de implementación como de uso, enumerando

los cambios más importantes que hemos establecido, y que pueden hacer que *Medusa 2* sea nuestro cliente de Oracle favorito.

5.5.1 Ampliaciones y mejoras

Como criterio para la diferenciación entre las ampliaciones y mejoras, y las nuevas funciones, vamos a valorar si la nueva característica se define en una parte de la aplicación que ya existía, o si por el contrario se ha definido completamente el segmento que la contiene.

De todas formas, se puede decir que vamos a hablar de lo nuevo en Medusa, por lo que la diferenciación es simplemente una forma de organizar las nuevas funciones.

5.5.1.1 Interfaz de usuario

El usuario pasa la mayoría de su tiempo interactuando con el interfaz de la aplicación, por lo que un interfaz intuitivo, cómodo y configurable, a la vez que sencillo, permitirá al usuario perder el mínimo tiempo posible. Medusa 2 parte de esa base y se ha intentado cuidar al máximo cada detalle, de manera que la aplicación resulte tan potente como sencilla de manejar.

Para conseguir esto, el primer reto que nos marcamos fue cambiar el antiguo interfaz de usuario, que podemos ver en la Figura 5.21, y que resultaba muy poco cómodo de utilizar. Y esto no lo decimos por alguna razón, sino más bien por muchas: los botones eran muy pequeños y necesitábamos apuntar bien con el puntero para poder pulsar, el menú de barras se basaba en un par de barras construidas con un componente `TCoolbar` de la VCL de Delphi, que, por ejemplo, sólo podíamos mover en horizontal y que si lo movíamos cerca de donde estuviera otra barra, ésta desaparecía, con el consiguiente despiste por parte del usuario.

Figura 5.21: Antigua barra de herramientas

Para arreglar esto, hemos utilizado el componente `TdxBarManager`, y aunque fue un poco costoso al principio construir de nuevo todos los menús del formulario principal, el resultado final valió la pena. Se añadían características como poder personalizar los botones de cada barra, moverlas a cualquier parte de la pantalla, en la que se acoplaban perfectamente, o poder dejarlas también "*al vuelo*", o lo que es lo mismo, sin anclar en ningún sitio a modo de ventana.

Otro escollo con el que tuvimos que enfrentarnos era la construcción de menús en tiempo de ejecución, como los favoritos o la lista de los últimos archivos abiertos, que se crean una vez se arranca el programa, y que al utilizar distintos componentes la implementación era completamente diferente.

La barra de herramientas y el menú de la aplicación de Medusa 2 se muestran en la Figura 5.22.

Figura 5.22: Menú y barras de herramientas de Medusa 2 por defecto

Con sólo cambiar las barras de herramientas o el menú no podemos decir que hemos mejorado el interfaz de usuario, serán otros muchos detalles, grandes y pequeños, los que nos permitan asegurarlo.

Una mejora importante está en la forma de interactuar del usuario con respecto a los formularios. En *Medusa 1* todos los formularios o ventanas que se mostraban, ya fueran de creación de objetos, como de cualquier otro tipo, se mostraban de forma modal. La forma modal de mostrar un formulario se refiere a que mientras esté activo ese formulario no se podrá utilizar ninguna parte del programa que no sea de ese formulario. Por decirlo de otra forma, hasta que no cerremos el formulario modal, no podremos hacer nada más. En el caso de que el formulario sea, por ejemplo, de información sobre la base de datos, nos resultaría mucho más cómodo poder dejarlo abierto para consultarlo mientras usamos otras características.

Medusa 2 crea los formularios a medida que se van necesitando y, según el tipo de formulario, podremos crear cuantas instancias del formulario nos apetezca, además de poder tenerlas abiertas todas a la vez. Así, podremos modificar las filas de una tabla, mientras tenemos a medio construir otra tabla y estamos listando los datos de la tabla a la que vamos a hacer referencia. Como se ve en la Figura 5.23 no tendremos límite si nos gusta aprovechar al máximo la multitarea.

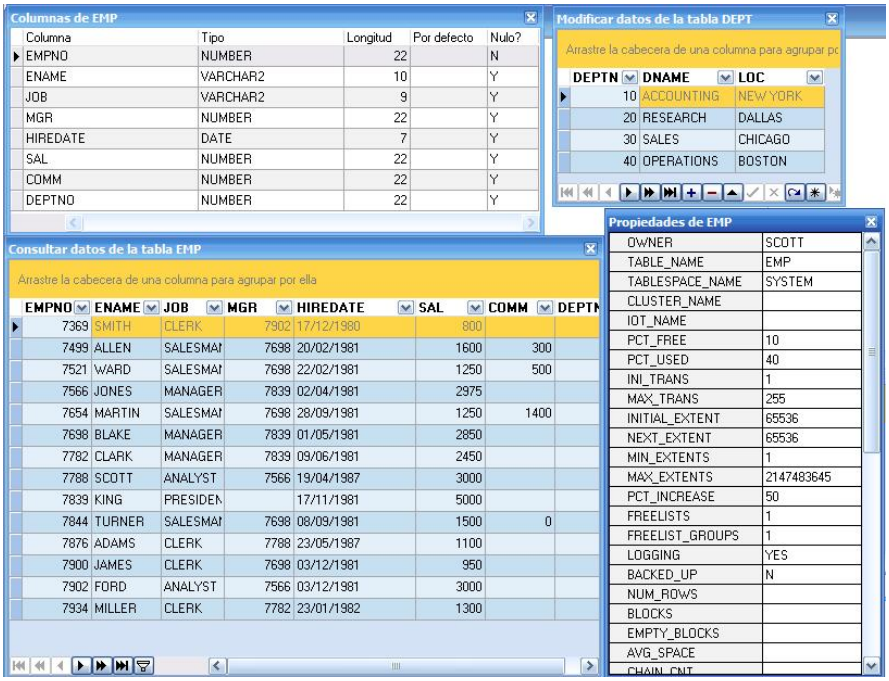


Figura 5.23: Podemos visualizar a la vez la descripción de la tabla EMP, sus propiedades y sus filas, todo ello mientras modificamos los datos de la tabla DEPT.

5.5.1.2 Constructores visuales

Los constructores visuales nos van a permitir crear nuevos objetos en una base de datos Oracle. Esta característica ya estaba en la primera versión, pero hemos incluido algunas mejoras muy interesantes. Como partida, se observó que los constructores visuales no eran tales, pues su funcionamiento consistía en crear una nueva ventana con el código necesario, pero que después debíamos de lanzar nosotros mismos, y en algún caso, incluso arreglar los posibles fallos.

En Medusa 2 los constructores visuales lo son realmente, pues si queremos crear por ejemplo una secuencia, una vez que rellenemos el formulario y aceptemos, la secuencia habrá sido creada. El problema en este caso podría ser que estuviéramos limitados a las posibilidades que nos

brindara el formulario, pero para eso se ha creado un botón denominado *Ver SQL* que nos mostrará cómo estamos construyendo internamente la sentencia que será lanzada. Aquí podremos modificar en lenguaje SQL para incluir mayores características, por si con las que nos brinda el formulario no nos bastara. Tras esto se puede guardar y/o lanzar la sentencia.

En las Figura 5.24 podemos ver la correspondencia entre el modo formulario y el modo SQL. Estos dos modos, además de ser usados por usuarios expertos también servirán a los novatos, que tendrán especificada la sentencia necesaria para crear los distintos objetos, por lo que además de realizar la tarea que necesitaban, aprenderán cómo se hace.

Por último, se han añadido algunos constructores visuales nuevos, el de biblioteca y el de trabajo, cuyo funcionamiento, además del resto, será detallado en el *Manual de usuario* en el Capítulo 7.7.6.3.

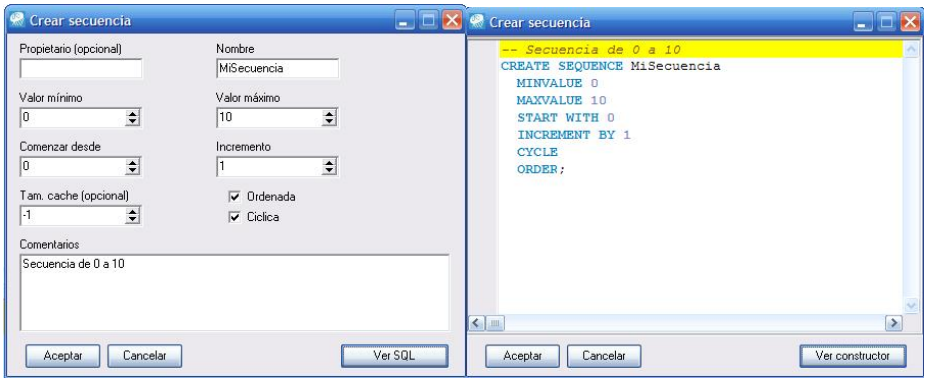


Figura 5.24: El modo formulario a la izquierda y el SQL a la derecha

5.5.1.3 Completando opciones

En algunas de las partes de nuestra aplicación, simplemente hemos dejado o adaptado las funciones anteriores, y en el caso de que estuvieran bien nos ha bastado con añadir mayor cantidad de opciones. Tal es el caso de las consultas al diccionario de datos, cuyo funcionamiento era correcto, así que lo adaptamos al nuevo interfaz y le añadimos algunas consultas más.

En Medusa 2 vamos a tratar con todos los tipos de objetos de una base de datos Oracle, por lo que convendría que estuviesen presentes todas las consultas al diccionario de datos que estén

relacionadas con estos objetos. Se han añadido nuevas consultas sobre, por ejemplo, bibliotecas, trabajos o clusters, en todos los submenús (*USER, ALL, DBA*).

Otro de los menús que ha sido completado es el de exportación a otros formatos. En *Medusa 1*, estaban disponibles las exportaciones a formato texto (*txt*) y formato de página web (*html*). En *Medusa 2*, se han incluido dos nuevos formatos, Excel y XML. La exportación a Excel nos creará un archivo con la extensión *xls*, en el que estarán los resultados ordenados por columnas tal y como estaban en la rejilla de resultados. En cuanto a la exportación a XML, el archivo resultante seguirá las reglas marcadas por la versión 1.0 de XML.

Otros detalles también han conseguido dar mayor calidad a *Medusa*. Por ejemplo, hemos añadido un botón *Limpiar*, que limpiará el área de edición de la sesión activa. También hemos añadido una lista de sesiones en el menú ventanas o un menú emergente en el control de sesiones, que vienen a subsanar deficiencias concretas. Otro dato importante a tener en cuenta es la ayuda de la aplicación, que se ha enlazado correctamente, de manera que pulsando *F1* sobre cualquier formulario podamos ver la ayuda específica. Todo esto quedará mejor descrito en el *Manual de usuario*.

5.5.1.4 Creación de formularios

En el apartado del interfaz de usuario ya hablamos sobre los formularios. *Medusa 1* utilizaba la creación modal de formularios y creaba todos los formularios de la aplicación al arrancar la aplicación. Otro de los objetivos que nos marcamos al estudiar la primera versión fue la creación de todos los formularios, tanto los antiguos como los nuevos, en el momento que fueran necesarios. De esta forma conseguiríamos que Medusa consumiera menos memoria, a pesar de tener casi el doble de formularios y un ejecutable que pasaba a ocupar de 2Mb a 6.5Mb.

La creación de formularios en *runtime* se puede realizar de varias formas, según si vamos a necesitar crear una sola instancia o un número indefinido. En los Listados 5.5 y 5.6 podemos ver la técnica de creación de un único formulario y de muchos, respectivamente.

```
Creación y visualización del formulario para crear un traba-
jo (sólo podrá haber uno a la vez):
      If not Assigned (FrmVisualJob) then
        Application.CreateForm(TFrmVisualJob, FrmVisualJob);
      FrmVisualJob.Show;
Código necesario cuando cerramos el formulario:
      Action:= caFree;
      FrmVisualJob:=nil;
```

Listado 5.5: Creación de una sola instancia del formulario FrmDescribe

```
Creación y visualización del formulario para describir una
tabla (podremos crear tantos como queramos):
      Application.CreateForm(TFrmDescribe, FrmDescribe);
      FrmDescribe.Show;
Código necesario cuando cerramos el formulario:
      Action:= caFree;
```

Listado 5.6: Creación de múltiples instancias del formulario FrmDescribe

5.5.2 Nuevas funciones

En este apartado vamos a hablar de nuevas funciones que han sido incorporadas en Medusa 2. Ninguna de ellas estaba presente en la primera versión, por lo que intentaremos explicar aproximadamente cual ha sido el proceso que hemos seguido en su desarrollo. De todas formas, para una mayor comprensión de cómo utilizarlas será necesario consultar el *Manual de usuario (*Cap. 7*).*

5.5.2.1 Resultados

Cuando lancemos a Oracle cualquier sentencia nos devolverá unos resultados, ya sea mensaje, ya sea un conjunto de datos. Al ver la primera versión de Medusa, uno de los primeros requisitos que se estableció fue que el manejo sobre los resultados fuera más potente.

En *Medusa 1*, el manejo sobre los resultados era prácticamente nulo, tan sólo se podía redimensionar el tamaño de las columnas y cambiarlas de orden. En *Medusa 2*, gracias al uso del `TcxGrid` y todos sus descendientes, hemos conseguido una gran versatilidad en el manejo de los resultados. Ahora podemos agrupar por una o varias columnas, filtrar utilizando cualquier condición o grupo de condiciones que se nos ocurran, eliminar columnas o utilizar marcadores para movernos por los registros.

La administración de los resultados conforma una gran cantidad de opciones que ya explicaremos en el Capítulo 7.4.2 con detenimiento.

5.5.2.2 Árbol de objetos

Cuando ejecutemos por primera vez *Medusa 2*, si estamos familiarizados con la primera versión, una de las cosas que más nos llamarán la atención es la parte izquierda de la pantalla. En ella, y separada por un divisor de la zona de sesiones, podremos ver dos vistas en árbol. La de la parte superior es el árbol de objetos y la inferior es el árbol de plantillas de código (explicado en la Sección 7.5).

El árbol de objetos va a ser una forma cómoda y sencilla de consultar e interactuar con el gestor de base de datos Oracle. Vamos a poder no sólo ver de una forma visual los distintos objetos organizados, sino que también podremos realizar las acciones más típicas sobre ellos de forma visual. Podremos crear, modificar, eliminar o ver las propiedades de prácticamente todos los tipos de objetos de una base de datos. Y en el caso de los objetos más utilizados, como las tablas, tendremos muchas más posibilidades.

El árbol de objetos se actualizará al cambiar el usuario de sesión activa, pero también podremos decidir actualizarlo si hemos realizado algún cambio en otra sesión y deseamos verlo en la otra.

Los objetos que serán mostrados dependerán del filtro que haya sido seleccionado. Podrá configurarse un filtro personalizado, pero por defecto tenemos 3 posibles:

- Todos los objetos: Se refiere a todos los objetos a los que tiene acceso el usuario, sean o no de su propiedad (vistas con el prefijo ALL_).
- Mis objetos: Son todos los objetos propiedad del usuario (vistas con el prefijo (USER_).
- Objetos ABD: Todos los objetos de la base de datos. Se necesitará tener privilegios de administrador y estar conectado como tal para poder acceder a esta opción (vistas con el prefijo DBA_).

5.5.2.3 Árbol de plantillas de código

Por debajo del árbol de objetos, en la esquina inferior izquierda, podremos ver otra vista en árbol similar a la del explorador de Windows. Este componente visual contiene un conjunto de carpetas con plantillas de texto, cuyo uso resulta realmente muy sencillo y rápido.

Todas las carpetas que aparezcan estarán almacenadas en la carpeta *<raiz>/Plantillas*, y dentro de ellas estarán las distintas plantillas. Estas plantillas se usan para acelerar el trabajo, de forma que la inserción de variables, constantes o tipos sea más rápida. También nos servirán para manejar errores, realizar operaciones de manipulación de datos y escribir funciones SQL.

El contenido de estas carpetas es configurable a través de un menú emergente, que saldrá al pulsar sobre el árbol con el botón derecho. Nos permitirá crear nuevas carpetas, crear nuevas plantillas, renombrar, modificar o eliminar entre algunas de sus posibilidades.

5.5.2.4 Informes

El menú principal de *Medusa 2* cuenta ahora con nuevos ítems, uno de ellos es el de informes. Dentro de informes vamos a tener dos opciones, generar informe y ver el explorador de informes.

La creación de informes surge por la necesidad de poseer copias escritas de nuestro trabajo. Resulta bastante útil la exportación a distintos formatos, pero sea cual sea el formato, vamos a ne-

cesitar de un ordenador para mostrar en qué hemos estado trabajando. Con los informes este problema quedará resuelto, si por ejemplo hemos creado una base de datos para la gestión de una tienda, extraeremos el contenido y las propiedades de las tablas que formarán la base de datos. Así, el cliente podrá ver aproximadamente cómo quedará su base de datos.

El uso de informes no sólo se va a dedicar a sacar listados para mostrar a otras personas. Sacar información de la base de datos y tenerla impresa será una de sus tareas más importantes. Con esto, nos referimos a que no sólo se utilizarán informes para listar datos de una tabla, podemos crear informes con cualquier resultado de una sentencia sobre el gestor de bases de datos.

Cuando pinchemos sobre *generar un nuevo informe* veremos que nos saldrá un formulario con una rejilla de resultados (que será el contenido de nuestro informe) y 3 botones. Si pulsamos sobre el botón situado más a la izquierda nos aparecerá un cuadro de edición. Aquí escribiremos una consulta y pulsando el botón central obtendremos los resultados. Si queremos podremos filtrarlos, agruparlos o realizar cualquier otra función que se nos permite, y cuando lo tengamos todo listo pulsaremos el botón situado más a la derecha. Aparecerá una previsualización del informe que podremos configurar como nos apetezca para después guardar o no el informe.

El explorador de informes nos va a mostrar todos los informes que hayamos creado. Por defecto, Medusa contiene una carpeta llamada DBA, en la que hay algunos informes hechos, y que pueden servir como arquetipo para la elaboración de los propios de cada usuario.

La generación de informes engloba muchos menús de aplicación distintos. Si queremos aprender a crear informes con toda la potencia que nos brinda Medusa 2 nos convendrá leer el *Manual de usuario* (Sección 7.7.5), pero sobre todo el uso de la aplicación y la creación de muchos informes, será lo que de verdad nos haga crear informes adecuadamente.

5.5.2.5 Control de transacciones

El control de transacciones en Oracle se basa en 3 comandos: *commit, rollback y savepoint*. *Medusa 2* nos va a evitar el uso de estos comandos, y nos va proveer de un interfaz visual para gestionar el control de transacciones.

Cuando iniciemos una nueva sesión estaremos creando una transacción. Medusa posee tres botones para controlar las transacciones. Con *commit* confirmaremos los cambios que hayamos hecho en la transacción y con *rollback* los desharemos. Además de estas opciones básicas podremos definir puntos de guarda (puntos de la transacción a los que volver en el caso de que queramos deshacer cambios).

Medusa 2 tiene una interfaz multisesión, con lo que podremos tener abiertas todas las sesiones que queramos a la vez, y cada una de ellas podrá tener definidos sus propios puntos de guarda. De todo esto se encargará nuestra aplicación, que cuando cambiemos de sesión activa cargará en el menú de programa los puntos de guarda que hayamos definido anteriormente.

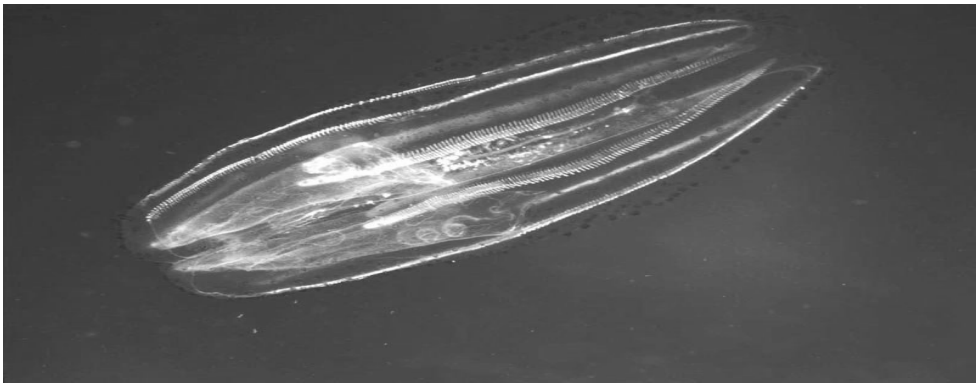
5.5.2.6 Tiempo de sentencias y scripts

En el menú herramientas ha sido incluido una característica para medir el tiempo que tardan las sentencias que lanzamos a Oracle. Para utilizarla bastará con que esté pulsado el botón que lo simboliza, la figura de un cronómetro, y lanzar una sentencia o un *script*.

Esta opción que puede parecer un poco extraña al principio, resultará muy útil cuando, por ejemplo, definamos índices o aumentemos los datos de una tabla, o la cambiemos de *tablespace*, por ejemplo, ya que podremos ver la diferencia en el tiempo de acceso.

Un caso que podría considerarse curioso al principio consistiría en lanzar dos veces la misma sentencia. Observaremos que la primera vez la aplicación nos mostrará un mensaje con el tiempo que ha sido empleado y que será superior a cero. Pero al lanzar la sentencia la segunda vez, si el número de resultados que devuelve es pequeño, el tiempo será de 0.0 segundos. Esto es así, porque los datos que habían sido devueltos permanecían en caché y no hizo falta acceder de nuevo a la base de datos.

En el caso de un *script* es útil para saber el tiempo que necesitamos para un conjunto de sentencias que pueden ser una o varias transacciones.



6

Detalles finales

Una vez realizada una aplicación no se deben olvidar unos detalles que sin ser necesarios dotan al programa de una mayor calidad.

La ayuda de Medusa 2 resolverá nuestras dudas y nos hará más fácil el proceso de adaptación. Con la instalación facilitaremos la puesta en marcha y aseguraremos la integridad de los ficheros, y con la traducción, vamos a conseguir atraer a aquellos usuarios que no sean hispanohablantes.

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. 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 gratuitamente 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 cómo, realmente, tiene una apariencia sencilla como la de un editor.

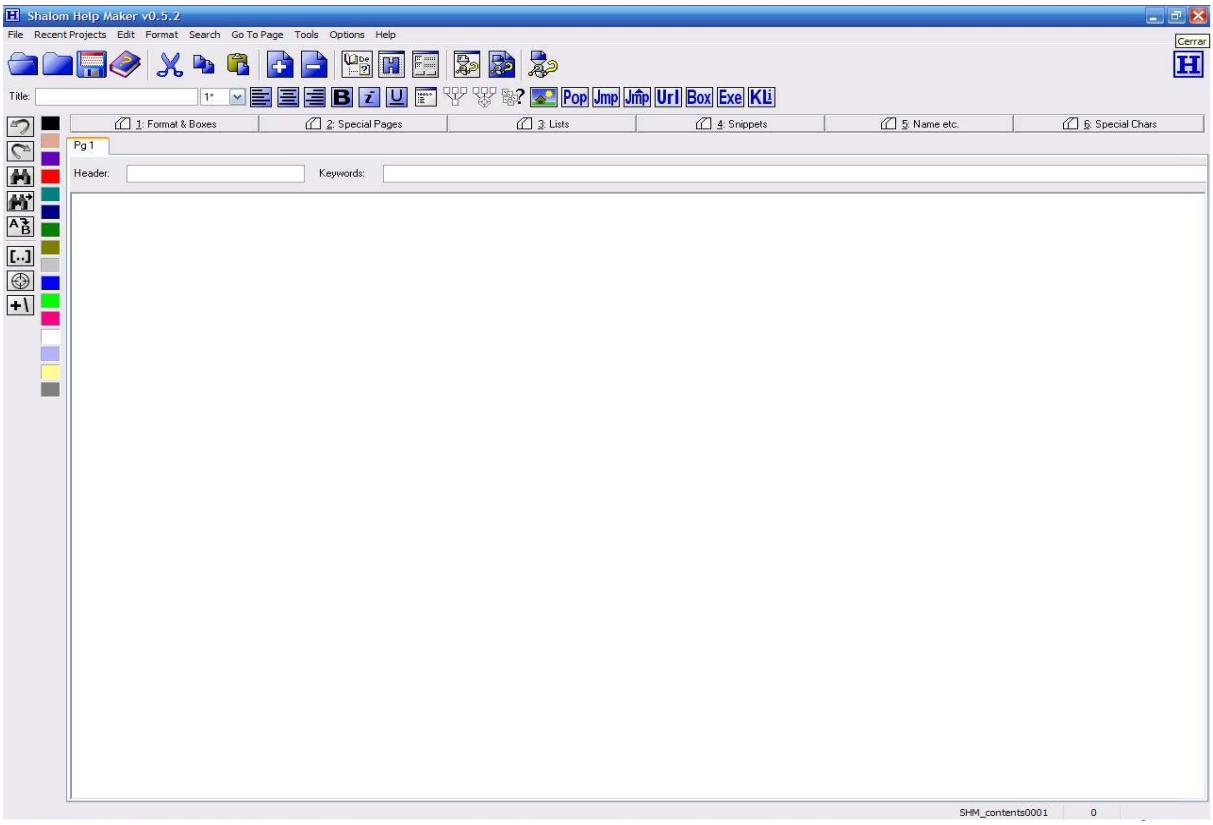


Figura 6.1: Pantalla inicial de Shalom Help Maker

Para obtener más información y una guía de la utilización de Shalom Help Maker se puede consultar el manual **[**SHA03**]**.

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 clic 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 Generando y probando la ayuda

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

6.1.2.1 Formas de organizar el proyecto de ayuda

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

6.1.2.1.1 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 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.

6.1.2.1.2 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 sí cambian. Por eso es mejor utilizar nombres de página constantes si tenemos la necesidad de hacer referencia a páginas específicas. Shalom Help Maker tiene una opción para generar estas constantes automáticamente.

6.1.2.2 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.

6.1.2.2.1 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 Listado 6.1 se muestra 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

6.1.2.2.2 Abriendo el fichero de ayuda cuando el usuario hace clic en un menú

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

```
procedure TFrmPrincipal.MIAyudaAyudaClick(Sender: TObject);
begin
  Application.HelpCommand(HELP_INDEX, 0);
end;
```

Listado 6.2: Forma de invocar la ayuda

6.1.2.2.3 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 de la ayuda

6.1.2.2.4 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 X Express Edition

El toque final del desarrollo de una aplicación es generar su instalación. De forma concisa podemos decir que una instalación es el paquete usado para instalar nuestros archivos y programas en la máquina del usuario.

La función principal de la instalación es transferir los archivos de la aplicación desde el disco de instalación hasta el ordenador del usuario, y gracias al S.O. Windows y a la ayuda de InstallShield (en la Figura 6.2 podemos ver la pantalla inicial del programa) este proceso va a resultar realmente sencillo. El resultado obtenido va a ser una instalación coherente de los archivos que nos permitirá la perfecta ejecución de *Medusa 2*.

Para obtener más información de la utilización de InstallShield X Express Edition, se puede consultar **[**INS04**]**.

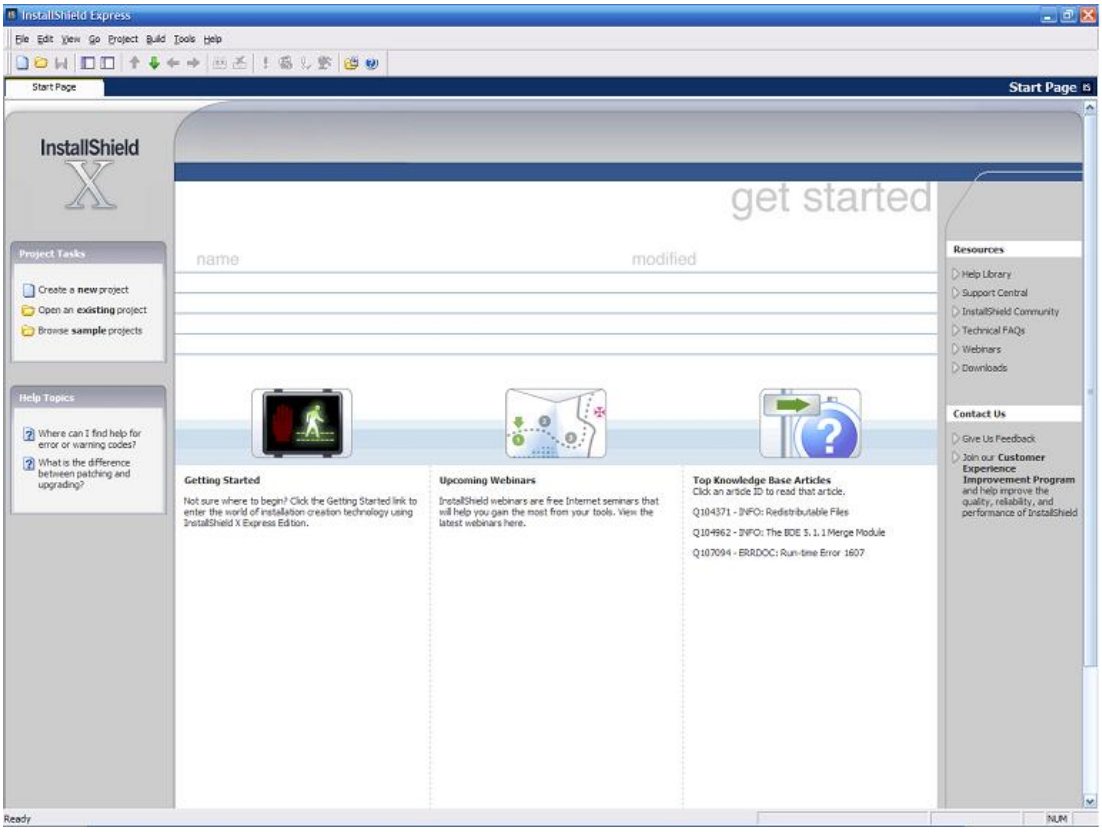


Figura 6.2: Pantalla inicial de InstallShield Express – Edición limitada para Borland

6.2.1 Los proyectos de InstallShield X Express Edition

InstallShield X Express Edition 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 *ise*. 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.

6.2.2 Características de InstallShield X Express Edition

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 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 configu-

ración 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 qué 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.

6.3 Traducción

La traducción de una aplicación puede realizarse de múltiples formas. También se puede realizar de diferentes formas el cambio de idioma. Por ejemplo, se puede seleccionar el idioma al realizar la instalación, se puede seleccionar dentro de la aplicación o se pueden distribuir distintas instalaciones para cada idioma. En *Medusa 2* empezamos optando por incluir el cambio de idioma dentro del programa, utilizando la herramienta de Delphi External Translation Manager (explicado en la Sección 6.3.1),, con lo que teníamos en el mismo ejecutable todos los idiomas incluidos. Delphi nos creaba un fichero de librería del idioma, llamado Medusa2.ENG, para inglés, y Medusa2.ESP, para español. Pero a medida que íbamos avanzando en la traducción, surgían inconvenientes como:

- No poder traducir los mensajes que se lanzaban vía código, como los *Showmessage* de Delphi.
- El tamaño del ejecutable crecía al tener que albergar más opciones que cuando sólo se poseía un idioma.
- El sistema de traducción utilizado no reconocía el formulario de *login* del componente DOA que nos permitía acceder al SGBD.
- El proceso de traducción se hacía demasiado lento, pues era difícil buscar en el interfaz de E.T.M. las cadenas a traducir.
- Quizás para el usuario final no tenía sentido un cambio de idioma dentro del programa, pues no es normal cambiar el idioma con el que se ha instalado una aplicación.
- Al generar la instalación de *Medusa 2* sólo se puede hacer en un idioma, por lo que habría que escoger en qué idioma hacerla.
- Si decidíamos no incluir el cambio de idioma dentro de *Medusa 2*, el sistema instalaría la aplicación teniendo en cuenta la configuración local que tenga el usuario, por lo que si tiene un S.O. en español, no podría instalar la versión inglesa.

Por estos motivos decidimos no seguir con el proceso de traducción que llevábamos y realizar instalaciones independientes para cada idioma. De todas formas, vamos a explicar un poco sobre la herramienta de traducción que intentamos utilizar, por si se eligiera utilizar en siguientes versiones.

6.3.1 External Translation Manager

La suite de herramientas de traducción de Borland nos va a permitir simultanear el desarrollo de aplicaciones con distintos idiomas. Las herramientas de traducción están integradas con el IDE, pero también podremos usarlas de forma externa. Para ello, ejecutaremos el archivo etm70.exe que está situado en la carpeta Bin de Delphi. También deberíamos echar un vistazo a las opciones de traducción en Delphi, que encontraremos en el menú `Tools | Translation tools options` del IDE de Delphi 7.

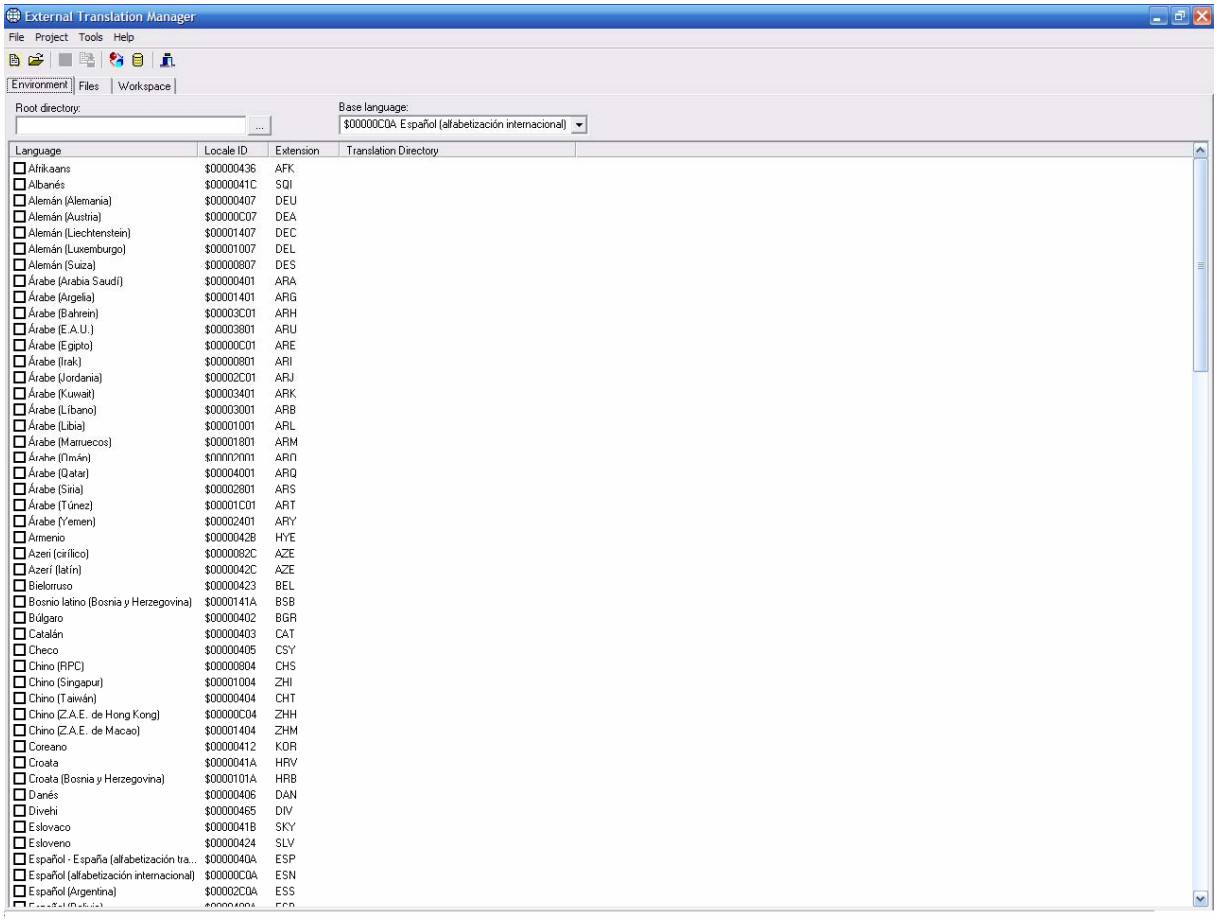


Figura 6.3: Aspecto de la pantalla inicial de ETM

6.3.2 Los proyectos de External Translation Manager

Para crear un nuevo proyecto con las herramientas de traducción de Borland, empezaremos indicando la ruta de nuestro proyecto y elegiremos los idiomas que vamos a incluir (Figura 6.3). Después en la pestaña de *Workspace*, encontraremos todos los formularios que integran nuestro

proyecto y al pinchar sobre cualquiera de ellos nos aparecerán todas las cadenas que podremos traducir.

En la Figura 6.4 podemos ver un ejemplo de cómo traduciríamos la ventana Principal de nuestro proyecto. En ella observamos que hay muchos campos que no son cadenas de texto, como los colores de fuentes, estilos, alineaciones… que están incluidos para el caso de que al cambiar el texto tengamos que reorganizar su posición o forma. De esta forma realizaríamos la traducción de todo el proyecto, aunque tendremos que tener cuidado también con los mensajes que aparecen vía código, para informar de errores o como diálogos predefinidos.

Una vez que hayamos terminado la traducción añadiremos los lenguajes en Delphi y estableceremos el que queramos que sea el lenguaje por defecto. Para comprobar los cambios que vayamos realizando Delphi nos permite cambiar el idioma activo, en `Project | Language | Set active`.

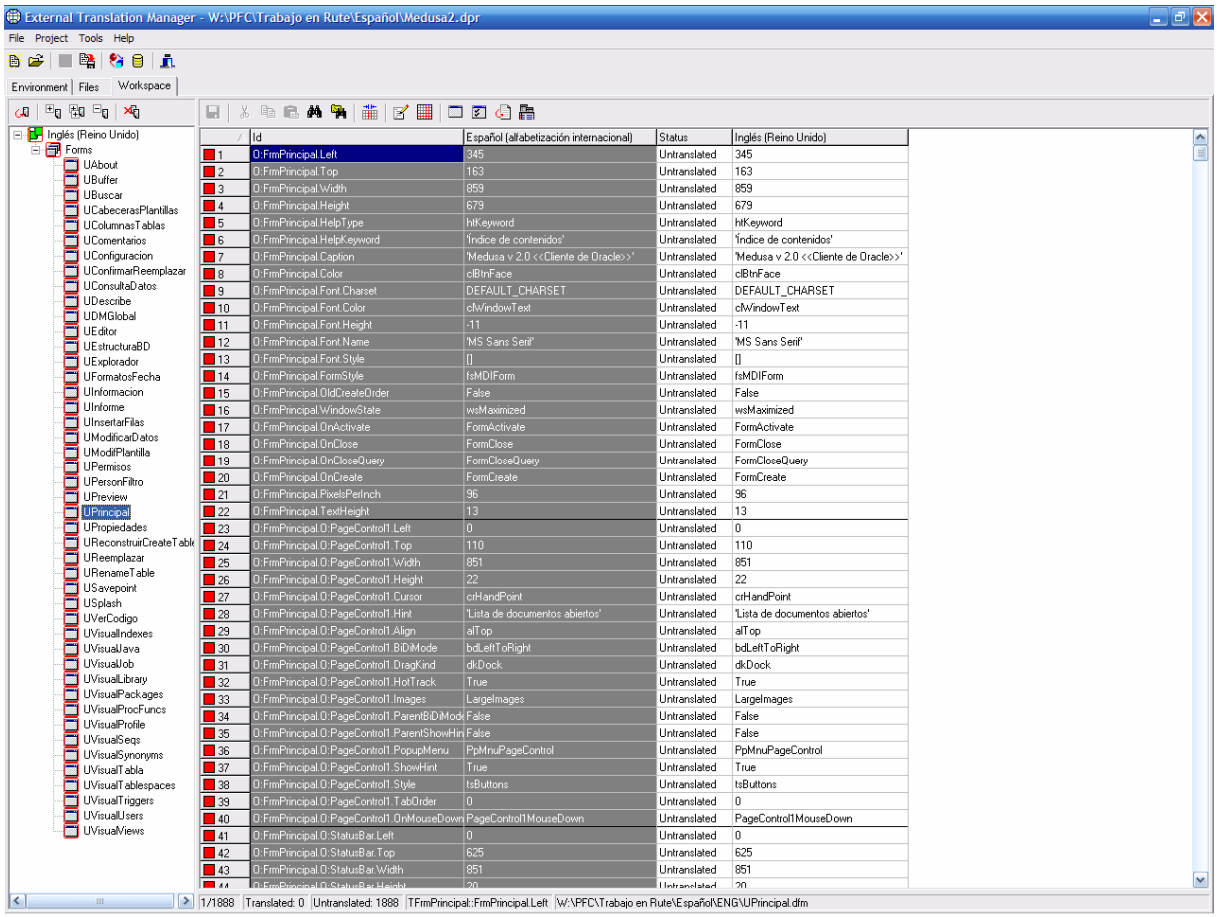
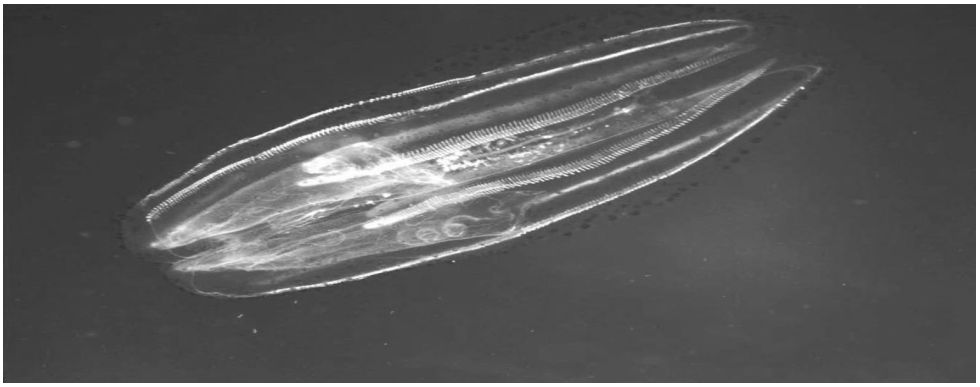


Figura 6.4: Aspecto de la traducción del formulario principal



7

Manual de usuario

Uno de los aspectos más importantes cuando queremos iniciarnos en el uso de un nuevo programa es el manual de usuario. Una buena ayuda nos va a proporcionar una documentación de las características de la aplicación, así como algunos trucos y consejos que nos permitan sacarle el máximo partido.

7.1 Introducción

Medusa 2 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). Estos 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 pueda 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).

Medusa 2 se puede decir que es mucho más que un completo editor. Vamos a tener un completo explorador de objetos con el que realizar las funciones comunes como crear, ver o eliminar objetos de una BD Oracle, de forma visual y sin necesidad de escribir ni una línea código. Además de esto podremos realizar multitud de funciones típicas, como conceder o revocar permisos, controlar la transacción actual, ver vistas del diccionario de datos, medir el tiempo de ejecución de una sentencia o *script*… tan cómoda y sencillamente como nos lo permite el interfaz visual de la aplicación.

Medusa 2 posee muchas más características que no están involucradas con el acceso a Oracle, como la exportación a distintos formatos, la generación de informes con los datos de una consulta o las múltiples posibilidades de configuración del entorno. Y todo ello, en una aplicación *multilenguaje* que puede ser usada mucho más allá de nuestras fronteras.

7.1.1 Instalación

Primero, debemos instalar la aplicación. Para instalar la aplicación, será necesario hacer doble clic sobre el ejecutable "Setup.exe", dentro de la carpeta de instalación del idioma que deseemos. 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. El formulario de estado de la instalación lo podemos ver en la Figura 7.1.

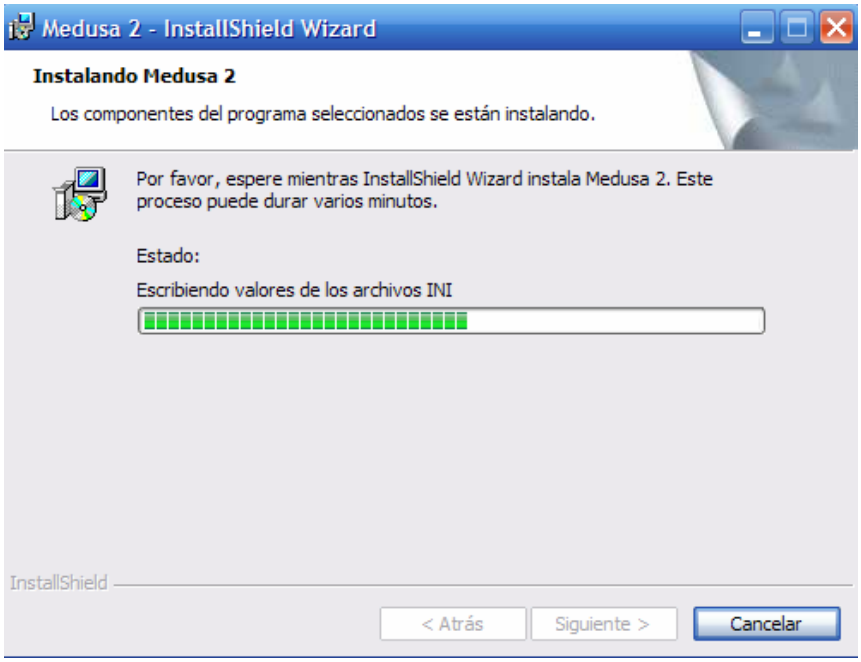


Figura 7.1: Aspecto de la ventana de instalación de Medusa 2

El proceso de desinstalación se puede hacer 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 clic en el acceso directo que encontraremos en el menú inicio o en el acceso del escritorio. 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. Tiene un interés meramente estético. Una vez que esté cargada la aplicación, tendremos acceso a la ventana principal de *Medusa 2*.

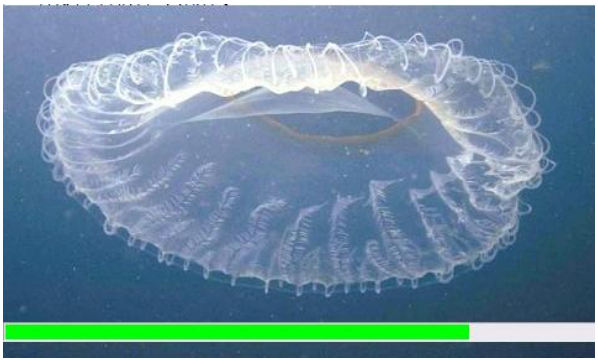


Figura 7.2: Aspecto del formulario de *carga*

7.1.2 El entorno de Medusa 2, una primera aproximación

La ventana principal del programa, presenta un aspecto similar al que se observa en la Figura 7.3. La ventana principal, con pocas opciones habilitadas a priori, nos servirá simplemente para configurar el entorno y de interfaz para crear nuevas sesiones (conexiones al SGBD con un determinado *login/pass*), ya que es una aplicación con una interfaz MDI (*Multiple Document Interface*, Interfaz de Múltiples Documentos) que soporta también multiusuario (conexión simultánea a Oracle con distintos usuarios).

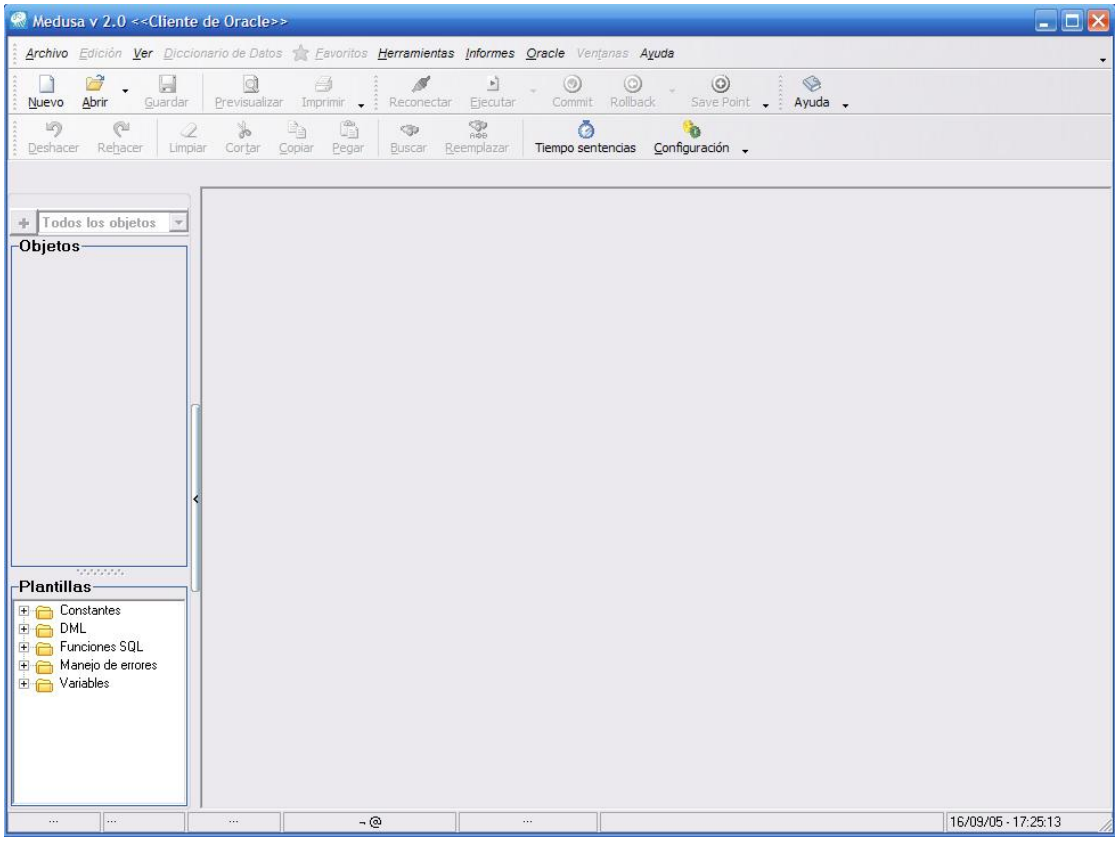


Figura 7.3: Aspecto de la ventana principal de *Medusa 2*

La ventana principal, está compuesta por seis zonas bien diferenciadas:

1. **Menú principal.** Se descompone a su vez en diez opciones. Todos los menús, se explicarán con más detalle, un poco más adelante, en la Sección 7.2.
   - Menú *Archivo*
   - Menú *Edición*
   - Menú *Ver*
   - Menú *Diccionario de datos*
   - Menú *Favoritos*
   - Menú *Herramientas*
   - Menú *Informes*
   - Menú *Oracle*
   - Menú *Ventanas*
   - Menú *Ayuda*
2. **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.
3. **Pestañas de las ventanas de edición.** Son pestañas vinculadas a cada una de las sesiones abiertas, 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 sesión activa en ese momento. Cuando se dispone de una sesión activa 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, además de usando el menú Ventanas.
4. **Área de las ventanas clientes.** Es el espacio de la aplicación reservado para las ventanas clientes, donde podremos tener varias a la vez y estar trabajando sobre la sesión activa.
5. **Área de objetos y plantillas.** Se sitúa a la izquierda y está separada por medio de un divisor móvil del área de ventanas clientes. En su parte superior está la zona correspondiente al explorador de objetos de la BD y en la inferior las plantillas de código. Ambas se explican en las Secciones 7.6 y 7.5, respectivamente.

6. **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 clic 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 sobrescribir el texto borrando lo que hubiera escrito) y 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@Forma de conexión (si es normal se obvia).
   - Panel 5. Muestra el número de filas obtenidas al realizar la consulta a la base de datos.
   - 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.

7.1.3 Empezando a utilizar Medusa 2

Para poder comenzar a trabajar realmente con toda las opciones que ofrece *Medusa 2*, debemos crear una nueva sesión, sea creando un nuevo fichero (*Archivo* | *Nuevo*) o abriendo un fichero existente (*Archivo* | *Abrir*) o abriendo un documento reciente (*Archivo* | *Abrir* | y seleccionando un elemento del menú de la parte inferior). Una vez que creemos una nueva sesión, 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 (Figura 7.4), el cual aparece-

rá automáticamente al crear la primera sesión o si pulsamos el botón de reconexión 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 introducir 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). 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.

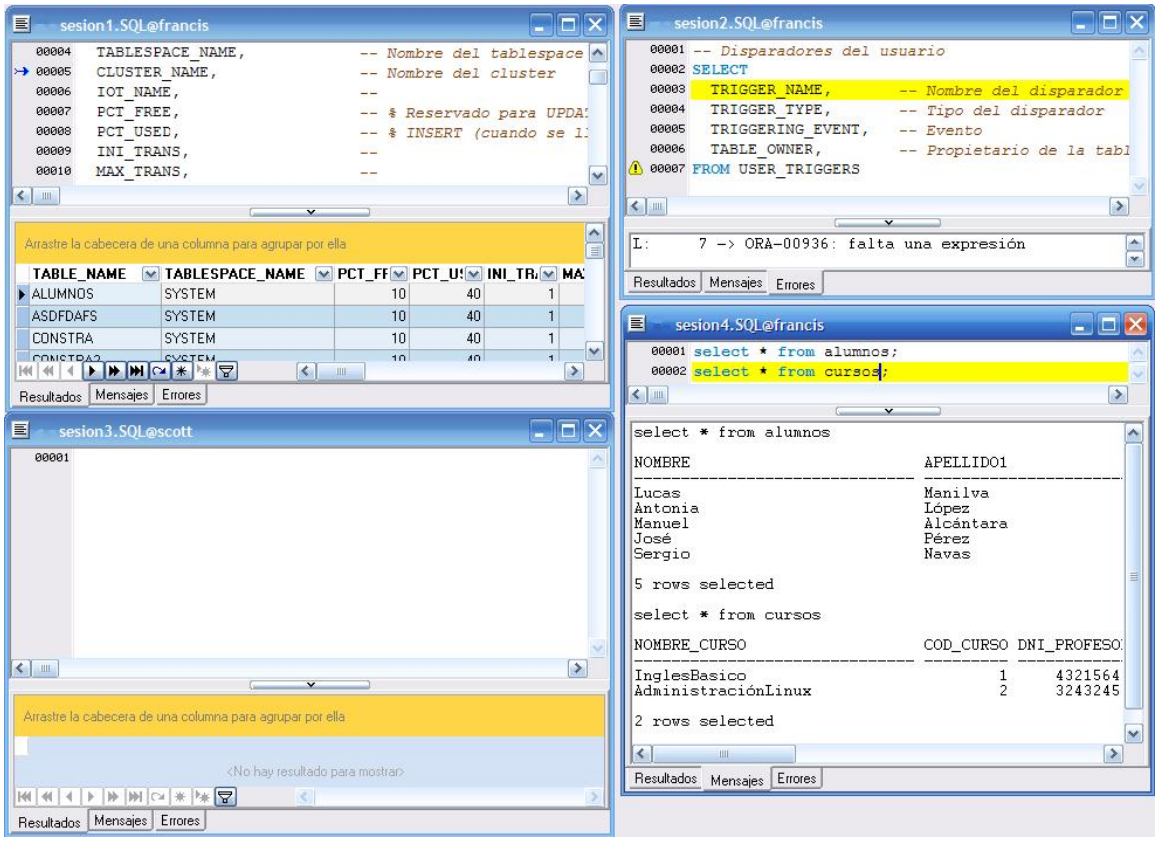


Figura 7.5: Aspecto de cuatro sesiones abiertas en Medusa 2

Una de las características fundamentales de nuestra aplicación va a ser el interfaz MDI. En la Figura 7.5 se muestra el aspecto de *Medusa 2* trabajando con cuatro sesiones. Los títulos 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 o no modificado que no 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 sesión número uno de la Figura 7.5 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 en el Apartado 7.1.2). 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 total de filas en la barra de estado, tal y como se ha explicado anteriormente).

En la sesión número dos se puede ver que se ha producido un error que 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 sesión número tres se puede ver que nos hemos reconectado a Oracle con otro nombre de usuario (Scott). Finalmente, la sesión 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.

7.2 Introducción a los menús de la aplicación

Tal y como se ha comentado anteriormente, al crear una nueva sesión tendremos accesibles numerosas opciones nuevas que no pueden usarse sin una sesión abierta. Entre estas opciones podemos destacar.

- **Menú Archivo:** 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.

  Este menú está compuesto por las opciones más habituales de trabajo con ficheros, como crear, abrir, cerrar, guardar o exportar. También están las opciones de configurar la impresora, imprimir y salir de la aplicación. Todas estas opciones las veremos con más detenimiento en la Sección 7.7.1.

- **Menú Edición:** Este menú, que antes de abrir una sesión está deshabilitado, 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 7.7.2.4). 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 7.8.1 que hace referencia a la configuración del editor.

- **Menú Ver*:*** Nos permite interactuar con la configuración del aspecto visual de la aplicación. Podemos personalizar las barras de herramientas, mostrar u ocultar la barra de estado y también mostrar u ocultar el *gutter* del editor. La personalización de las barras de herramientas se explica posteriormente, en la Sección 7.3.

- **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 las columnas, su significado y el significado de la vista en cuestión. Esto se explica con más detalle en el Apartado 7.7.3.

- **Menú Favoritos:** Nos proporciona un fácil acceso a las ficheros *sql* favoritos del usuario: un conjunto de sus sentencias más habituales o a las que desee tener un acceso más rápido. De una forma sencilla, podremos cargar una sentencia o un *script* para poder editarlo como si fuese un fichero base, o ejecutarlo, además de poder configurarlos de una forma sencilla. Esto se verá más adelante, en la Sección 7.8.5.

- **Menú Herramientas:** Nos permite de forma sencilla, el acceso a la configuración de la aplicación, así como la medición del tiempo que se emplea en ejecutar una sentencia o *script*. La configuración de *Medusa 2*, se trata con detenimiento en la Sección 7.7.4.

- **Menú Informes:** A través de este menú vamos a poder controlar el uso de informes (conjunto de datos presentados de una forma elegante para posteriormente mostrarlos a otras personas), tanto su creación como su visualización y posterior impresión. Aunque tendremos posibilidades de crear informes desde casi cualquier rejilla de resultados de la aplicación, aquí tendremos la posibilidad de crear un nuevo informe a pantalla completa o ver nuestro explorador de informes. Para mayor información consultar la Sección 7.7.5.

- **Menú Oracle:** Sin duda el menú más importante de *Medusa 2*. En este menú, encontraremos las siguientes opciones: (las cuales, se detallan una a una a lo largo de este capítulo).

    - Reconexión a la base de datos. Permite reconectar la sesión activa para conectarnos con distintos privilegios o también como un usuario distinto. Aparecerá el diálogo de conexión (Figura 7.4) explicado en la Sección 7.1.2.

- Ejecución de sentencias / *scripts*. Permite enviar el documento al Sistema Gestor de Bases de Datos (SGBD) de Oracle para recibir las respuestas.
- Obtención de información de la sesión. Se muestran varios datos importantes, como la versión de Oracle instalada, el identificador de sesión, el identificador de usuario, sus espacios de tablas (por defecto y temporal) y alguna información adicional, como los roles que posee el usuario, las limitaciones y sus privilegios (ver Sección 7.7.6.1).
- Obtención de la estructura de la base de datos. 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 7.7.6.2.
- Creación de objetos de la base de datos. Existen formularios para facilitar la creación de objetos. Todas estas funciones se tratan en la Sección 7.7.6.3.
- Reconstrucción de la sentencia `CREATE TABLE`. A partir del nombre de una tabla, se obtienen todos sus atributos, tipos, restricciones, comentarios, etc. (ver Sección 7.7.6.4).
- Control de transacciones. Podremos controlar a través de este menú el desarrollo de la transacción actual, definiendo *savepoints*, deshaciendo partes de la transacción o confirmándola. Ver la Sección 7.7.6.5 para mayor información.
- 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 7.7.6.6.
- Generación de formatos de fechas de forma gráfica. Nos permite de forma gráfica, la creación de formatos personalizados de fechas (ver Sección 7.7.6.7).
- 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 en la Sección 7.7.6.8.

℘ **Menú Ventanas:** En este menú encontraremos todas las opciones usuales de gestión de ventanas de una aplicación MDI, como pueden ser arreglo en cascada, arreglo horizontal, arreglo vertical y minimización de todas las ventanas. Por último, también tendremos la posibilidad de movernos a través de las ventanas, pues tenemos una lista de las sesiones

abiertas en la aplicación, y con tan sólo hacer clic sobre un nombre cambiaremos la sesión activa.

- **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 2* 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.*

7.3 Barras de Herramientas

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

Figura 7.6: Aspecto de las barras de herramientas con todos los botones habilitados

En la Figura 7.6 podemos ver los botones de la barra de herramientas que aparecerán por defecto cuando instalemos *Medusa 2*. Todos ellos, así como las distintas barras a las que están asociados son completamente configurables desde el entorno.

En *Medusa 2* vamos a disponer de 4 barras de herramientas además del menú principal, cuyo comportamiento y características será igual que el resto de barras. En la Figura 7.7 podemos ver las barras de herramientas de la aplicación sin estar ancladas a ningún lugar.

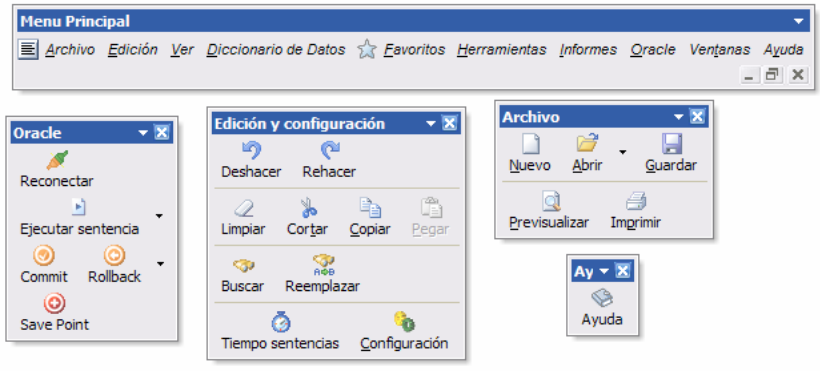


Figura 7.7: Barras de herramientas de Medusa 2 sin anclar

Las posibilidades de configuración de las que vamos a disponer nos van a dar la capacidad de establecer el entorno a nuestro gusto personal, facilitándonos el acceso a aquellas partes de la aplicación que usemos con mayor frecuencia. A continuación vamos a exponer las distintas formas que vamos a tener para configurar las barras de herramientas:

- Al pinchar sobre cualquier barra de herramientas dejando el botón izquierdo pulsado veremos que podemos arrastrar la barra a donde queramos. Si la arrastramos lejos de los bordes de la pantalla la barra permanecerá sin anclar, como si tuviéramos una ventana abierta. En cuanto la acerquemos a algún borde de la pantalla la barra se acomodará automáticamente al lugar, ya sea horizontal o verticalmente. Para cerrar la barra nos bastará con pulsar la *cruz* aparece en la esquina superior derecha.
- Si pinchamos con el botón derecho en cualquier barra de herramientas nos aparecerá un menú desplegable similar al de la Figura 7.8. En él vamos a poder configurar qué barras queremos que estén visibles, igual que si lo hacemos desde el menú *Ver*. Si pulsamos sobre *Configurar* accederemos al formulario de configuración de las barras de tareas, que explicaremos más adelante.
- Cada barra posee un pequeño botón situado en el extremo derecho con un pequeño triángulo. Su función va a ser la de visualizar de una forma rápida qué botones queremos ver de la barra. Al igual que en el resto de opciones si el botón tiene a la izquierda la señal ☑ estará visible en el interfaz de usuario, y en el caso contrario no lo estará. La última opción que aparece, llamada *Configurar*, nos mostrará el diálogo de configuración de las barras de tareas. El menú desplegable del que hablamos lo podemos ver en la Figura 7.9.

℘ El formulario de configuración de las barras de herramientas nos va a permitir gestionar de una forma muy potente el interfaz de nuestra aplicación. Si queremos vamos a poder crear nuevas barras de herramientas, eliminarlas, renombrarlas, configurar los botones que las forman, añadir nuevos comandos a los menús o personalizar la forma en que se muestran los menús.

Este formulario tiene 3 pestañas cuyo contenido podemos ver en la Figura 7.10 y que comprende:

- **Barras de herramientas**: Aquí también podremos mostrar u ocultar las barras de herramientas, pero además tenemos la opción de crear barras propias para después añadirle los comandos que queramos y así no tendremos que ceñirnos a las que trae *Medusa 2* por defecto.
- **Comandos**: Organizados por categorías, aquí podremos encontrar todas las opciones de menú que tiene nuestra aplicación. Con sólo arrastrar hasta una barra de herramientas habremos colocado el comando en la barra de menú correspondiente. También en el caso de que queramos desvincular algún botón de una barra de herramientas lo arrastraremos fuera de su barra, con lo que el comando quedará desvinculado de la barra en cuestión.
- **Opciones**: Esta pestaña comprende la configuración visual de las barras de herramientas. Tendremos disponibles 4 animaciones de menú distintas, la opción de mostrar u ocultar las sugerencias al dejar quieto el puntero sobre una opción de menú o botón. Otra opción es ver los iconos grandes, pues aunque la mayoría de comandos de la aplicación están configurados como botones grandes, hay algunos, como las distintas vistas al diccionario de datos, que pueden ser grandes o pequeños. Por último, vamos a tener también la opción de decidir si queremos que los menús se muestren tras un corto retraso o que se muestren primero los utilizados recientemente.

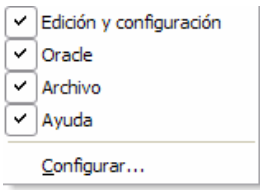


Figura 7.8: Menú desplegable de las barras de herramientas

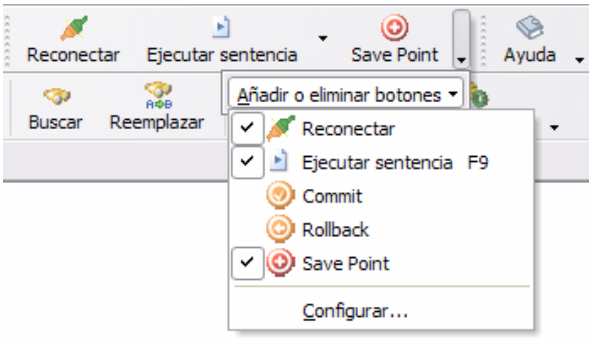


Figura 7.9: Menú desplegable para añadir o eliminar botones

Figura 7.10: Las 3 pestañas del formulario de configuración de barras de herramientas

7.4 Opciones de edición y tratamiento de resultados

La sesión activa que tengamos va a tener siempre dos zonas bien diferenciadas, y separadas por un divisor, la zona de edición y la zona de resultados. En la zona de edición es donde vamos a escribir las sentencias o *scripts* que lanzaremos al SGBD y en la zona de resultados tendremos las respuestas que nos devolverá Oracle, ya sea en forma de errores como en forma de filas de una tabla resultado.

7.4.1 Características avanzadas de la ventana de edición

Aquí vamos a describir algunas características de Medusa 2 como editor de texto. La posibilidad de definir marcas de línea, seleccionar el realzado de texto (palabras reservadas que cambian de color) o cambiar la fuente del editor, van a hacer que nuestra aplicación permita un manejo potente y ágil del texto con el que elaboraremos las sentencias.

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

Una opción que puede ser 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 7.11 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 clic sobre *Limpiar* del menú *Conmutar marcas de 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.

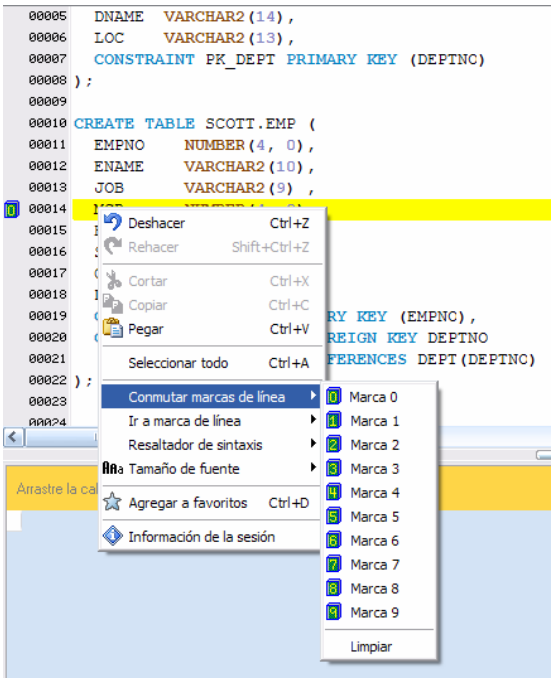


Figura 7.11: Conmutando una marca de línea

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.

7.4.1.2 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 2* 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 clic con el botón derecho sobre la ventana de edición y seleccionar el menú *Resaltador de sintaxis*, y después elegir el resaltado de-

seado. 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 7.12

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).

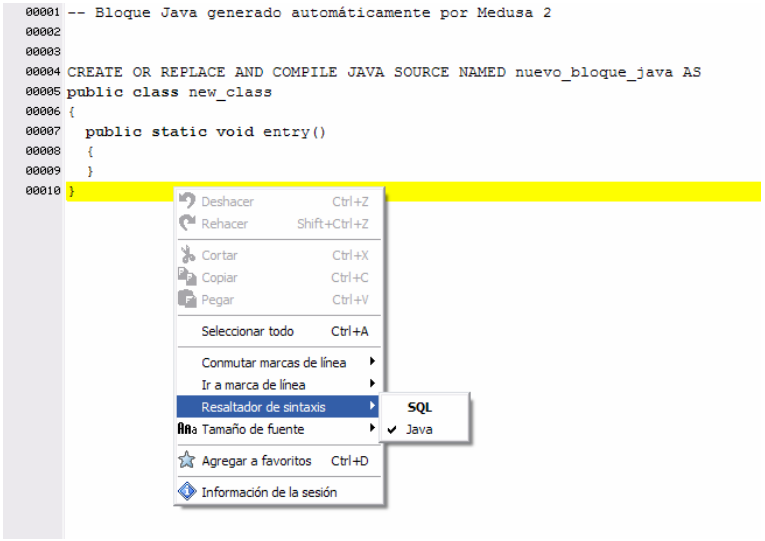


Figura 7.12: Selección del realzado sintáctico del documento

7.4.1.3 Cambiando el tamaño de fuente del editor

Se puede seleccionar el tamaño de la fuente del editor e igualmente de la rejilla de los resultados, simplemente haciendo clic 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 7.13 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.

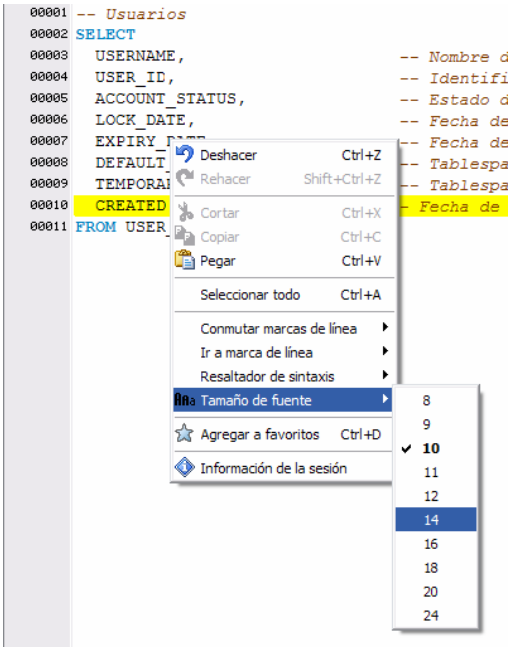


Figura 7.13: Cambiando el tamaño de fuente del editor

7.4.2 Manejo de resultados

Medusa 2 se caracteriza por una gran flexibilidad en el manejo de resultados. Las principales funciones que vamos a poder realizar sobre los resultados de una consulta van a ser la navegación, filtrado y agrupamiento. La navegación consiste en poder recorrer los registros que componen el resultado, el filtrado en seleccionar propiedades que deben de cumplir los datos y el agrupamiento en hacer grupos con los resultados según un parámetro. Además de éstas, también tendremos otras características no relacionadas con el manejo de datos, como el cambio de tamaño de la fuente del editor o la exportación de los resultados a ficheros con distintos formatos.

7.4.2.1 Navegación

En la Figura 7.14 podemos ver el resultado de realizar una consulta. Marcado por una elipse están las barras de navegación. La barra 1 es la básica que tendremos en todas las rejillas de resultados, y la 2 es la barra de navegación que se utiliza en los formularios de modificación de los datos de una tabla. A continuación vamos a describir la función de todos los botones de la barra 2 (la barra 1 es igual pero con menor cantidad de botones) de izquierda a derecha:

 Ir a la primera fila de los resultados.

 Ir a la página anterior de resultados.

 Ir a la fila anterior.

 Ir a la fila siguiente.

 Ir a la página siguiente de resultados.

 Ir a la última fila de los resultados.

 Inserta una fila en la tabla.

 Elimina la fila seleccionada de la tabla.

 Edita el campo seleccionado.

 Envía los datos al SGBD para actualizar los cambios en la BD.

 Cancela la modificación en curso, no actualiza los cambios en la BD.

 Refresca los resultados por si han sido cambiados.

 Guarda una marca en la fila actual con la que podremos recordarla para ir posteriormente, sólo se puede definir una marca.

 Ir a la marca guardada anteriormente

 Filtrar resultados (abre el *Constructor avanzado de filtros*, explicado en el siguiente apartado).

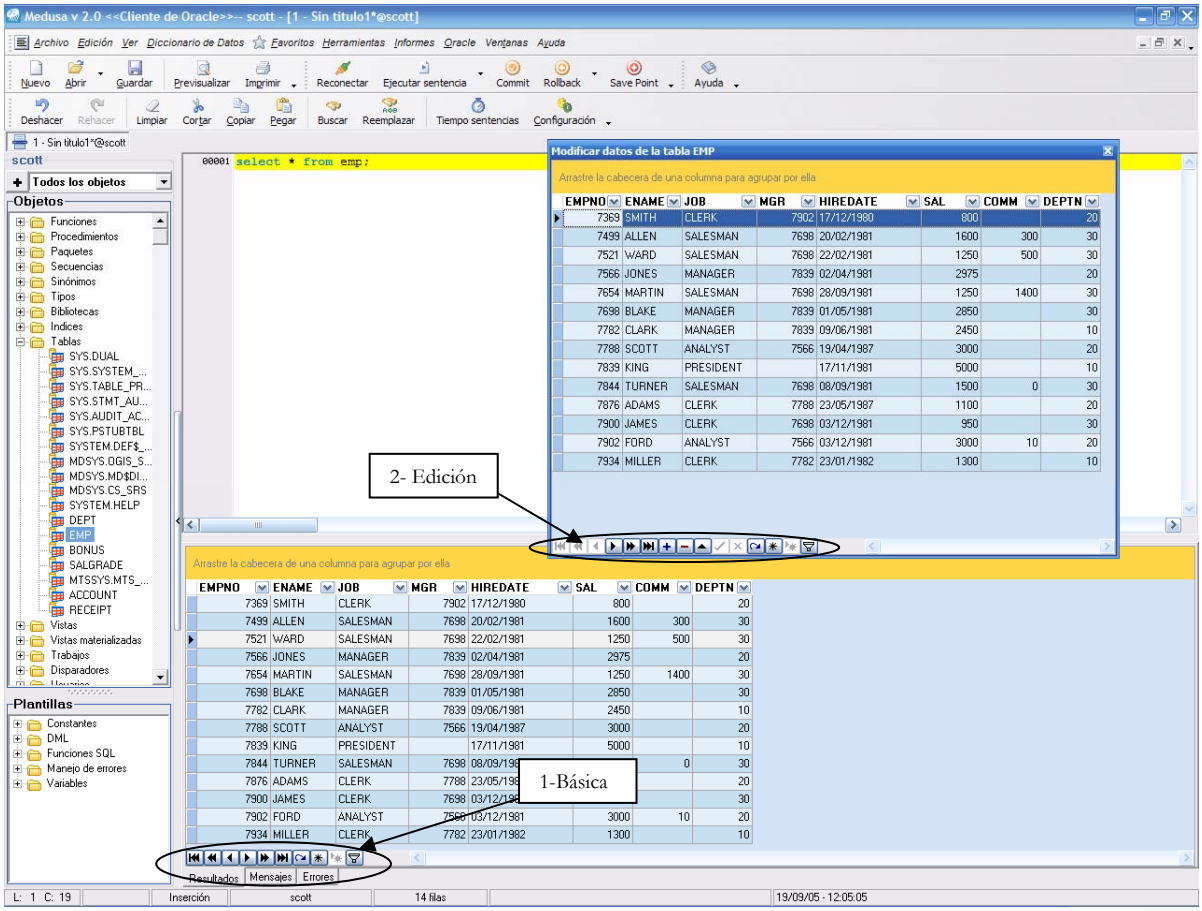


Figura 7.14: Distintas barras de navegación en Medusa 2

7.4.2.2 Filtrado

En Medusa 2 vamos a tener muy variadas formas de filtrar datos. También podemos eliminar columnas del resultado, lo haremos simplemente arrastrándolas hacia fuera hasta que se coloque una gran cruz encima del nombre de la columna. Las columnas eliminadas del resultado no podrán ser recuperadas posteriormente.

Para filtrar los datos por el valor de una columna pulsaremos sobre el pequeño botón que está situado a la derecha del nombre de la columna (Figura 7.14) y nos aparecerá una lista con todos los valores diferentes de la columna encontrados en los resultados, la etiqueta *All* y *Custom*. Si pulsamos sobre algún valor la tabla quedará automáticamente filtrada utilizando como propiedad que el valor de esa columna sea igual al que hemos pinchado. Si pulsamos sobre *All* deshare-

mos el filtro que acabábamos de establecer y si pulsamos *Custom* nos aparecerá el diálogo de la Figura 7.15.

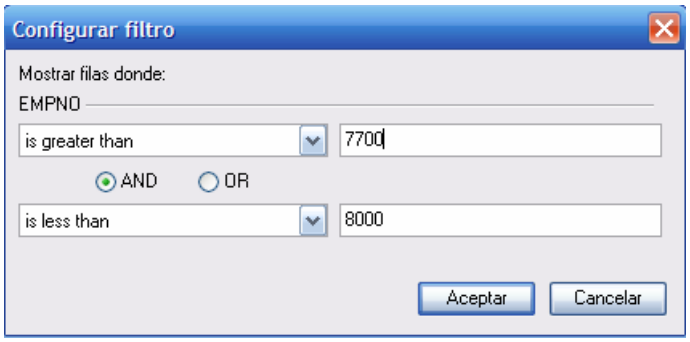


Figura 7.15: Configurar un filtro básico sobre la columna EMPNO

En este formulario podemos definir las condiciones que queremos que cumpla el filtro sobre la columna, en la Figura 7.15 se ha definido un filtro sobre la columna EMP para que sólo se muestren los resultados de aquellos empleados cuyos números de empleado estén entre el 7700 y el 8000. Al aceptar el nuevo filtro la rejilla de resultados mostrará el filtro que tenemos seleccionado y filtrará los resultados. En la Figura 7.16 se muestra el resultado de realizar el filtrado.

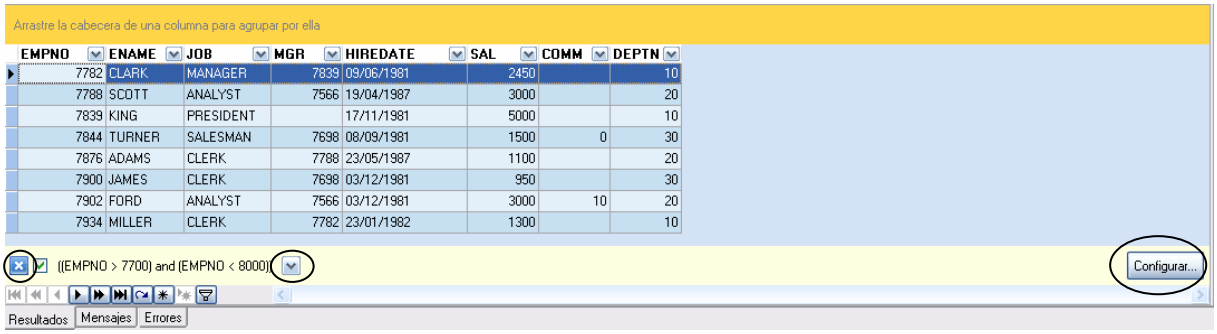


Figura 7.16: Resultado de aplicar el filtro de la Figura 7.15

La barra de filtro que vemos en la Figura 7.16 nos va a indicar los filtros que hay activos sobre los resultados. Si pulsamos el botón X situado más a la izquierda desharemos el filtro, si pulsamos el botón con una flecha hacia abajo veremos un historial de los filtrados utilizados recientemente y si pulsamos sobre `Configurar` aparecerá el formulario de configuración avanzada de filtros de la Figura 7.17.

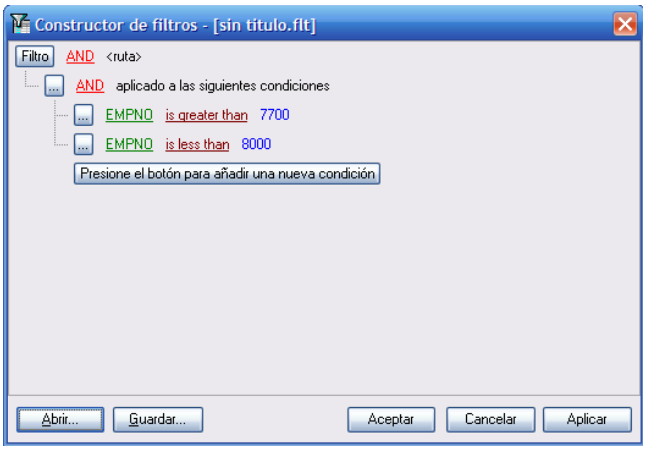


Figura 7.17: Formulario de construcción avanzada de filtros

Para definir filtros avanzados debemos de aprender a manejar el constructor de filtros. Al abrirse, se mostrarán los filtros activos, y tendremos la posibilidad de modificarlos. Para ello, vamos a pulsar el botón `Filtro` y nos aparecerán las siguientes opciones (las mismas que si pulsamos sobre cualquier botón con la etiqueta con puntos suspensivos):

- **Añadir grupo**: Agrega un grupo de condiciones, ajeno a los que haya definidos y unido a ellos por un AND o un OR.
- **Añadir condición**: Añade una condición dentro de un grupo ya definido.
- **Limpiar todos** (o eliminar fila si pulsamos sobre una condición): Elimina todos los filtros establecidos (o elimina la condición sobre la que habíamos pulsado).

Como vemos en la Figura 7.18 hemos definido más condiciones, de manera que ahora vamos a ver en los resultados sólo a aquellos empleados cuyo número de empleado esté entre 7700 y 8000 y trabajen como oficinistas o hayan recibido alguna comisión. En la Figura 7.19 vemos el resultado de aplicar el filtro, fijémonos que la barra de filtro nos muestra la trascripción del filtro que hemos construido.

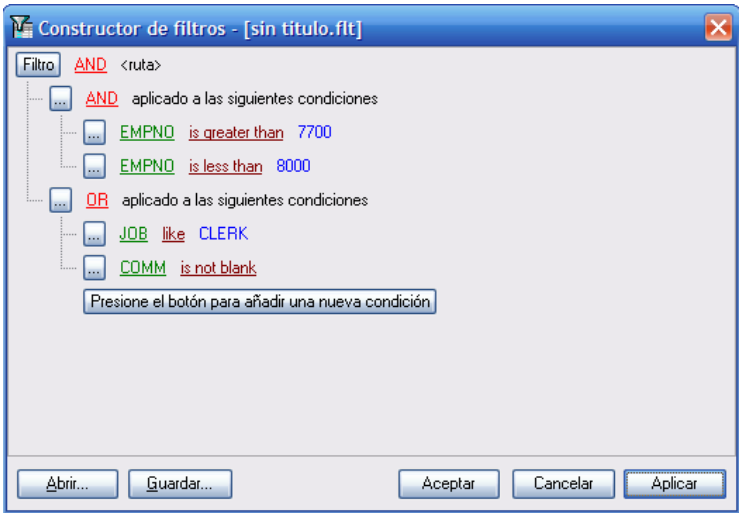


Figura 7.18: Construcción de un filtro personalizado

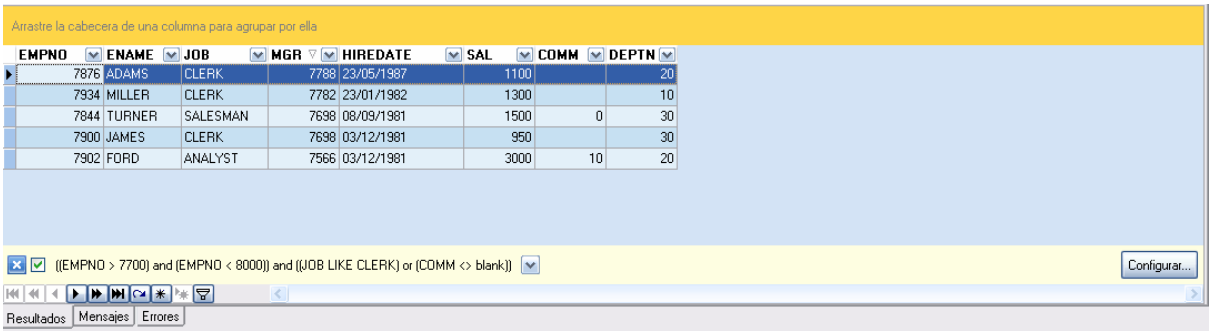


Figura 7.19: Resultado de aplicar el filtro de la Figura 7.18

En la Figura 7.19 también podemos observar algo de lo que no habíamos hablado anteriormente, la ordenación de resultados por el valor de una columna. En el caso de la figura, hemos ordenado de mayor a menor por el valor de la columna MGR. Para ordenar por el valor de cualquier otra columna bastará con pulsar sobre ella y si lo volvemos a hacer cambiará la orientación de la flecha de ordenación, es decir, pasará de ordenar de mayor a menor a ordenar de menor a mayor. Dependiendo del tipo de columna se ordenará alfabéticamente, por orden de fecha o por el valor numérico.

Otra circunstancia que deberíamos de tener en cuenta es la frecuencia con la que utilizamos ciertos filtros, pues si solemos trabajar con las mismas tablas y las filtramos habitualmente igual, sería muy cómodo para nosotros poder almacenar los filtros que usamos. *Medusa 2* responde a este requerimiento con la posibilidad de guardar y cargar posteriormente nuestros filtros. Para ello, solamente deberemos de pulsar sobre `Abrir` o `Guardar` del formulario de configuración de filtros

(Figura 7.17), que los buscará o almacenará en la carpeta *Filtros* situada en el directorio raíz de la aplicación.

7.4.2.3 Agrupación

Perfectamente compatible con el filtrado, la ordenación y la navegación podemos usar sobre los resultados la agrupación por el valor de una columna. Para realizar una agrupación nos bastará con arrastrar el nombre de la columna por la que queremos agrupar a la zona superior amarilla de resultados. Si queremos aplicar al resultado otra agrupación simplemente volveremos a realizar la operación. En la Figura 7.20 podemos ver las filas de la tabla EMP agrupadas primero por el número de departamento y después por el trabajo del empleado.

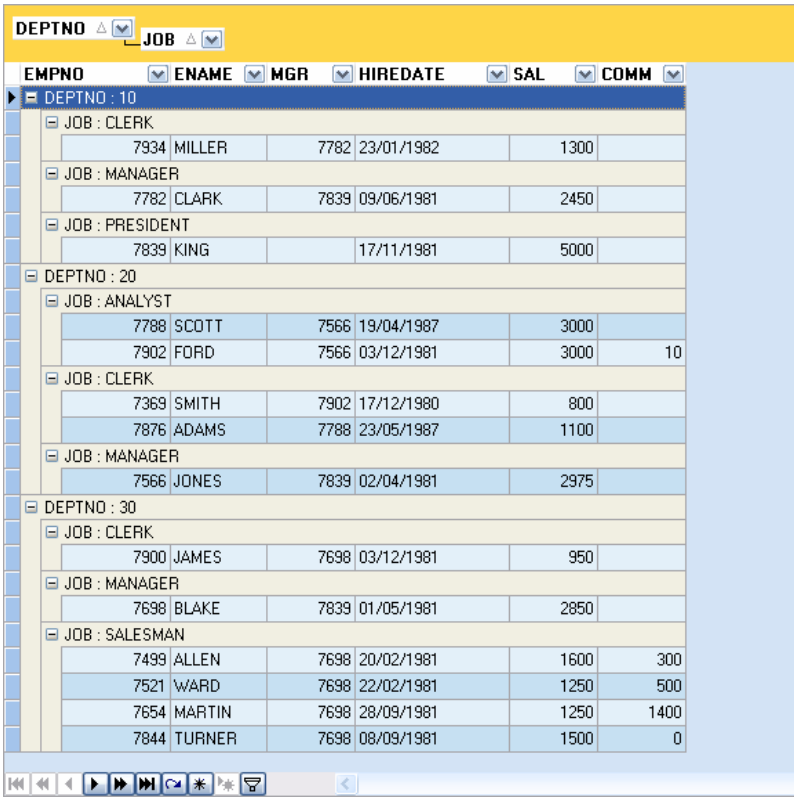


Figura 7.20: Agrupando resultados por los atributos DEPTNO y JOB

Si queremos deshacer la agrupación arrastraremos el nombre de la columna que está agrupada fuera de la zona amarilla de agrupación. De esta forma los resultados volverán a desagruparse.

7.5 Plantillas de código

Las plantillas de código son una forma ágil y sencilla de escribir código. Utilizándolas, conseguiremos agilizar el trabajo diario y minimizar errores de escritura. Podremos organizarlas a nuestro antojo e ir creando las que vayamos creyendo necesarias.

En la parte izquierda de la pantalla, podemos ver, separado del área de ventanas cliente por un divisor, el explorador de objetos (arriba) y las plantillas de código (abajo). Estas dos vistas en árbol pueden ocultarse con tan sólo pulsar sobre el divisor, y arrastrando el divisor podremos redimensionar su tamaño.

El manejo de las plantillas de código resulta realmente sencillo, y se va a realizar simplemente pulsando con el botón derecho (o Mayúsculas-F10) sobre cualquier parte del área de plantillas. El menú de la Figura 7.21 es el que nos ayudará a configurar las plantillas que tendremos disponibles en *Medusa 2*. En cuanto a su uso, es tan fácil como pinchar dos veces sobre una plantilla para añadir su contenido al editor de la sesión activa.

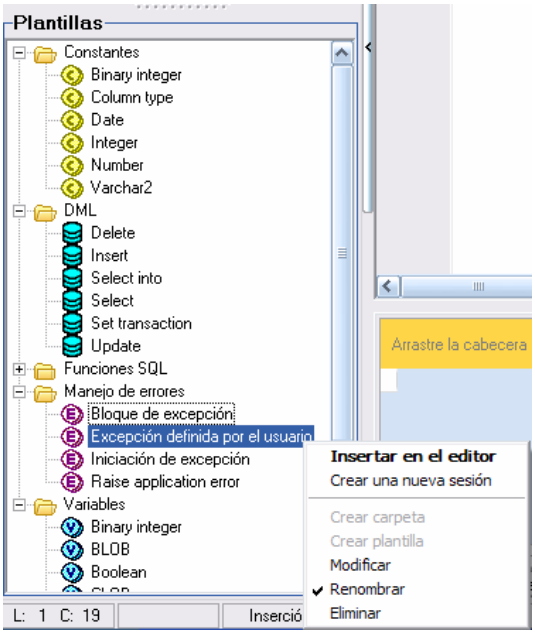


Figura 7.21: Menú desplegable para gestionar las plantillas de código

Como podemos ver en la Figura 7.21 el menú para gestionar las plantillas de código tiene las siguientes opciones:

- **Insertar en el editor**: Realiza la misma acción que si pulsamos dos veces sobre cualquier plantilla, es decir, insertar su contenido en la posición del cursor del editor de la sesión activa.
- **Crear una nueva sesión**: Crea una nueva sesión en *Medusa 2*. Esta opción se ha incluido porque podemos acceder a las plantillas sin estar conectados a la BD, pero si intentamos insertar una plantilla, la aplicación nos mostrará un mensaje de error diciendo que no estamos conectados.
- **Crear carpeta**: Crea una nueva carpeta para contener plantillas de código definidas por el usuario. El nombre que se le dará por defecto es *Nueva carpeta* y si creamos después otras *Nueva carpeta(1), Nueva carpeta (2)…*
- **Crear plantilla**: Crea una nueva plantilla de código vacía. Dentro de la carpeta que hayamos pinchado se creará una nueva plantilla con el nombre por defecto *plantilla*. Al igual que en las carpetas las siguientes plantillas se nombrarán *plantilla(1), plantilla(2)…*
- **Modificar**: Modifica el contenido de la plantilla. Al pinchar sobre esta opción nos aparecerá un formulario como el de la Figura 7.22 con el contenido actual de la plantilla y que podremos modificar editándolo y pulsando sobre `Aceptar.`
- **Renombrar**: En el caso de que esté activado podremos, pinchando sobre cualquier plantilla o carpeta que ya estuviera seleccionada, renombrar la plantilla o carpeta. Si no está activada la opción, no podremos renombrar con dos toques no seguidos del ratón.
- **Eliminar**: Elimina la plantilla o carpeta seleccionada. La aplicación nos mostrará antes un mensaje de confirmación y en caso afirmativo eliminará la plantilla o carpeta del sistema. También podremos pulsar la tecla *Supr* del teclado para realizar esta acción.

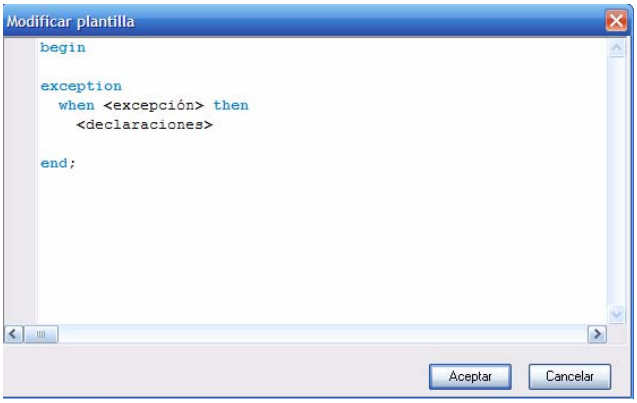


Figura 7.22: Formulario para modificar el contenido de una plantilla de código

Cuando instalemos *Medusa 2* dispondremos de muchas plantillas para utilizar organizadas por carpetas. Como vimos en la Figura 7.21 son:

- **Constantes**: Contiene plantillas para crear constantes en bloques PL/SQL, como *Integer, Number, Varchar2, Date*…
- **DML**: Contiene plantillas para sentencias de DML, como *Select, Select into, Update, Delete*…
- **Funciones SQL**: Contiene una gran cantidad de funciones SQL como *Add_months, Upper, Max, Sum, To_char*, Sqrt…
- **Manejo de errores**: Contiene plantillas para el manejo de errores en Oracle, tales como *definir un bloque de error, inicializar una excepción, lanzar una excepción*…
- **Variables**: Contiene plantillas para definir variables en bloques PL/SQL, como *boolean, long, integer, BLOB, number*…

7.6 Explorador de objetos

A la izquierda de la pantalla, por encima del árbol de plantillas de código, se encuentra el explorador de objetos. El explorador de objetos de *Medusa 2* nos va a permitir acceder cómodamente a los objetos de una BD Oracle. Sus principales funciones son mostrar los distintos objetos organizados por carpetas y poder realizar las funciones más usuales sobre ellos de forma visual. El aspecto del explorador con todos sus nodos contraídos lo podemos ver en la Figura 7.23. Para mayor información sobre cada tipo de objeto, así como de sus propiedades, características y usos, consultar el manual de Oracle [ORA02].

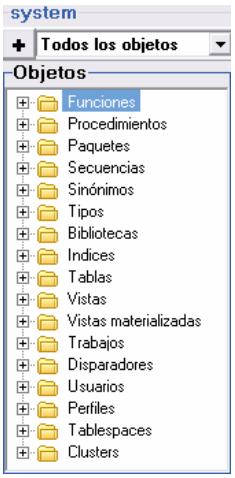


Figura 7.23: Explorador de objetos con todos sus nodos contraídos.

El explorador de objetos está formado por 17 carpetas. Cada carpeta representa un contenedor de un tipo de objeto y tendrá un icono propio que podemos ver en la Tabla 7.1. Estas 17 carpetas, simbolizan a todos los tipos de objetos que hay en Oracle 8 y son no configurables, es decir, no podremos añadir o eliminar ninguna. Cada vez que pulsemos dos veces sobre cualquier carpeta o pulsemos sobre el símbolo + que hay a su izquierda el nodo se expandirá mostrando todos los objetos de ese tipo que cumplan el filtro seleccionado en la parte superior del Explorador de Objetos (Figura 7.23). El apartado 7.6.2 explica este filtro de objetos.

Icono	Tipo de Objeto
	Procedimiento
	Función
	Paquete
	Secuencia
	Sinónimo
	Tipo
	Biblioteca
	Índice
	Tabla
	Vista
	Vista materializada
	Trabajo
	Disparador
	Usuario
	Perfil
	*Tablespace*
	*Cluster*

Tabla 7.1: Representación de los 17 iconos del explorador de objetos

7.6.1 Menús del explorador de objetos

Cuando pulsemos con el botón derecho del ratón (o Mayúsculas-F10) sobre cualquier *item* del explorador nos aparecerá un menú desplegable, que será distinto según si hemos pulsado una carpeta o un objeto, y en este caso, también será distinto dependiendo del objeto que hayamos pulsado. Cabe destacar que todos los formularios que aparezcan al pulsar las distintas opciones no son modales, es decir, que podemos tener todos los que queramos abiertos a la vez. Si deseamos ver el

código de una función, las propiedades y columnas de una tabla o consultar los datos de otra tabla, todo a la vez, no habrá ningún problema. Podemos tener abiertos simultáneamente todos los formularios que deseemos.

Las opciones realizables sobre una carpeta del explorador de objetos (ver Figura 7.24) son:

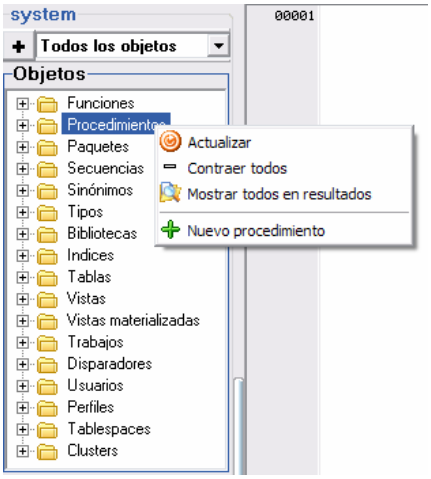


Figura 7.24: Menú que se despliega cuando pulsemos con el botón derecho sobre una carpeta

- **Actualizar**: Actualiza el explorador de objetos, dejando expandidas las carpetas que así lo estuvieran antes de refrescar el árbol.
- **Contraer todos**: Contrae todas carpetas que estuvieran expandidas.
- **Mostrar todos en resultados**: Muestra todos los objetos de ese tipo en la rejilla de resultados de la sesión activa. De esta manera podremos filtrarlos o agruparlos con mayor flexibilidad y potencia.
- **Nuevo**: Dependiendo de la carpeta que hayamos pulsado podremos crear un nuevo objeto de ese tipo. Al pulsar sobre la opción nos aparecerá un formulario de creación visual. Los formularios de creación de objetos los describiremos más adelante, en la Sección 7.7.6.3 (menú Oracle).

Al pulsar con el botón derecho sobre un objeto nos aparecerá otro menú desplegable diferente según el tipo de objeto, como dijimos anteriormente, pero que básicamente es como el de la Figura 7.25. Las nuevas opciones que podemos realizar sobre el objeto son:

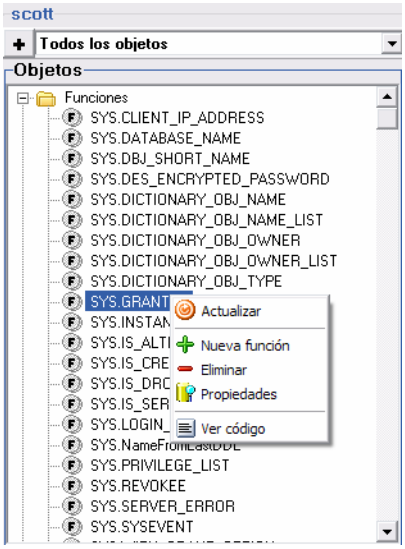


Figura 7.25: Menú al pulsar sobre un objeto

- **Eliminar**: Elimina el objeto seleccionado. Al pulsar sobre la opción nos aparecerá un diálogo de confirmación y podremos ver en la pestaña de mensajes de la sesión activa algunos consejos. Estos consejos nos advierten de los riesgos de eliminar el objeto además de darnos unas instrucciones que deberíamos de seguir después de la eliminación. Un ejemplo lo podemos ver en la Figura 7.26 que es la consecuencia de intentar eliminar una tabla.

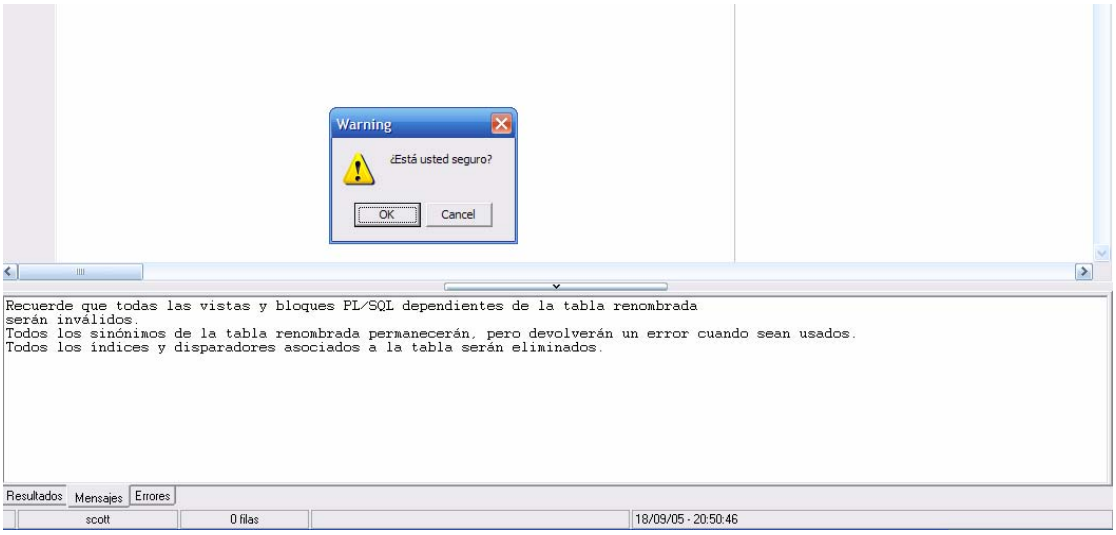
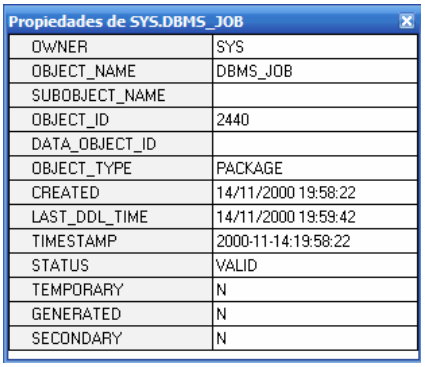


Figura 7.26: Diálogo de confirmación que elimina una tabla

- **Propiedades**: Nos muestra algunas propiedades del objeto seleccionado. En la Figura 7.27 podemos ver el formulario resultante al hacer clic sobre las propiedades del paquete de sistema `DBMS_JOB`. Las propiedades mostradas son diferentes para cada tipo de objeto, para más información sobre las propiedades de los objetos de una base de datos Oracle consultar [W3DOC] y el Manual de Oracle [ORA02].



OWNER	SYS
OBJECT_NAME	DBMS_JOB
SUBOBJECT_NAME	
OBJECT_ID	2440
DATA_OBJECT_ID	
OBJECT_TYPE	PACKAGE
CREATED	14/11/2000 19:58:22
LAST_DDL_TIME	14/11/2000 19:59:42
TIMESTAMP	2000-11-14:19:58:22
STATUS	VALID
TEMPORARY	N
GENERATED	N
SECONDARY	N

Figura 7.27: Propiedades del paquete SYS.DBMS_JOB

- **Ver código**: Esta opción sólo será visible en el caso de que hayamos pulsado sobre un procedimiento o función. El resultado será una ventana con el código asociado al procedimiento o función. Esta ventana no es editable, sirve para ver el código pero no para cambiarlo. Un ejemplo lo podemos ver en la Figura 7.28, que muestra el código asociado a la función de sistema `SYS.PRIVILEGE_LIST`.

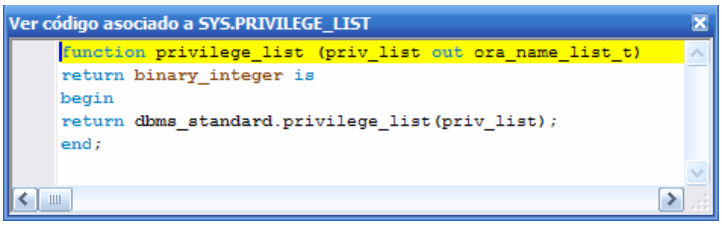


Figura 7.28: Formulario para ver el código asociado a procedimientos y funciones

Por último, vamos a describir el menú emergente perteneciente al objeto *Tabla*, que es el más completo de todos puesto que este tipo de objeto resulta ser además de uno de los más importantes, quizá también, el que será más utilizado por el usuario de *Medusa 2*. Las opciones posibles a ejecutar sobre cualquier objeto de tipo *tabla* las podemos ver en la Figura 7.29 y son:

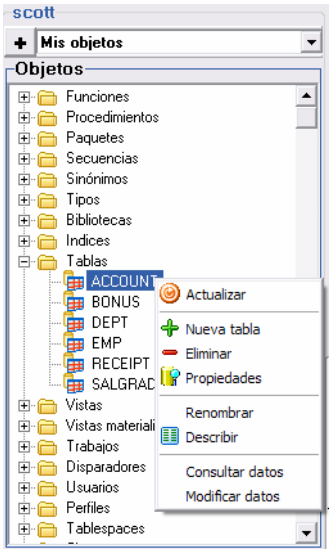
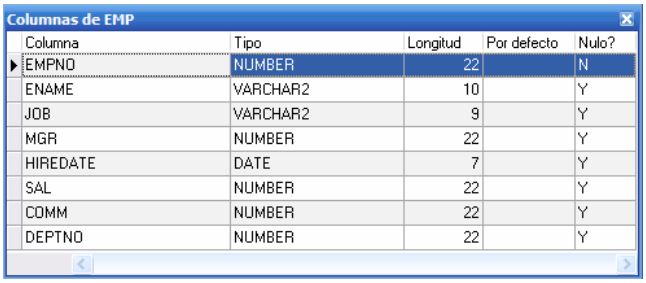


Figura 7.29: Menú emergente del objeto `Tabla`

- **Renombrar**: Cambia el nombre de la tabla. Al pulsar sobre la opción se nos mostrará un formulario para que introduzcamos el nuevo nombre, y al hacerlo y aceptar la modificación la tabla habrá sido renombrada y podremos ver en la pestaña de `mensajes` algunas líneas que nos informan sobre los cambios que debemos de tener en cuenta al modificar el nombre de una tabla.
- **Describir:** Muestra las columnas de la tabla seleccionada en un nuevo formulario. De cada columna obtendremos información sobre su nombre, tipo, longitud, su valor por defecto y si puede o no ser nulo (ver Figura 7.30).

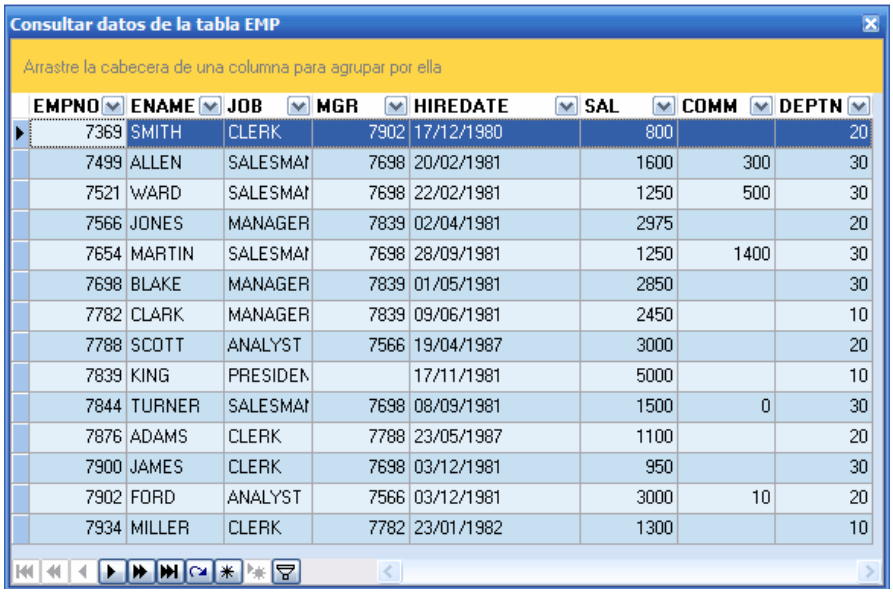


Columna	Tipo	Longitud	Por defecto	Nulo?
EMPNO	NUMBER	22		N
ENAME	VARCHAR2	10		Y
JOB	VARCHAR2	9		Y
MGR	NUMBER	22		Y
HIREDATE	DATE	7		Y
SAL	NUMBER	22		Y
COMM	NUMBER	22		Y
DEPTNO	NUMBER	22		Y

Figura 7.30: Columnas de la tabla EMP del usuario Scott

- **Consultar datos:** Abre un formulario que consiste tan sólo en una rejilla de resultados. Aquí podremos ver todas las filas de la tabla seleccionada, teniendo la posibilidad de movernos entre ellas, pero sin poder realizar ninguna acción sobre los datos. En la Figura 7.31 podemos ver el resultado de consultar los datos de la tabla EMP, utilizada ante-

riormente. Esto facilita ver toda la tabla de forma cómoda. Con la barra de botones inferior podemos filtrar, ordenar o agrupar los datos, como se vio en la Sección 7.4.2.



EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTN
7369	SMITH	CLERK	7902	17/12/1980	800		20
7499	ALLEN	SALESMAI	7698	20/02/1981	1600	300	30
7521	WARD	SALESMAI	7698	22/02/1981	1250	500	30
7566	JONES	MANAGER	7839	02/04/1981	2975		20
7654	MARTIN	SALESMAI	7698	28/09/1981	1250	1400	30
7698	BLAKE	MANAGER	7839	01/05/1981	2850		30
7782	CLARK	MANAGER	7839	09/06/1981	2450		10
7788	SCOTT	ANALYST	7566	19/04/1987	3000		20
7839	KING	PRESIDEN		17/11/1981	5000		10
7844	TURNER	SALESMAI	7698	08/09/1981	1500	0	30
7876	ADAMS	CLERK	7788	23/05/1987	1100		20
7900	JAMES	CLERK	7698	03/12/1981	950		30
7902	FORD	ANALYST	7566	03/12/1981	3000	10	20
7934	MILLER	CLERK	7782	23/01/1982	1300		10

Figura 7.31: Filas de la tabla EMP

℘ **Modificar datos**: A primera vista parece similar a la opción anterior, pues también muestra todos los datos de una tabla, pero en este caso la barra de navegación de resultados va a tener más botones. El propósito de estos botones va a ser la inserción o eliminación de filas por un lado, o la edición de los datos existentes en la tabla por otro. Las funciones de todos los botones de la barra de navegación ya las vimos en la Sección 7.4.2.1, de manera que lo que vamos a explicar son más peculiaridades que funciones:

- Cuando intentemos editar un dato de tipo numérico no podremos escribir ninguna letra, tan sólo números.
- Cuando modifiquemos alguna celda o dato la actualización será inmediatamente enviada al SGBD, de manera que tenemos que tener cuidado con los cambios que hacemos.
- Si intentamos modificar algún registro con un valor no válido nos saldrá el mensaje de error correspondiente, como podemos ver en la Figura 7.33, en la que intentamos cambiar la clave primaria de un empleado a otro valor ya existente.
- Si queremos editar un campo de tipo fecha nos aparecerá un calendario para seleccionar la fecha deseada, como se ve en la Figura 7.32.

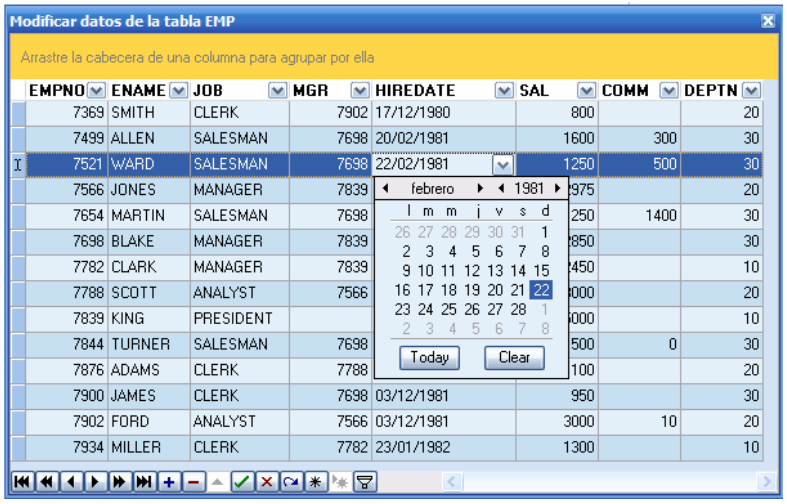


Figura 7.32: Formulario para manejar los datos de la tabla EMP

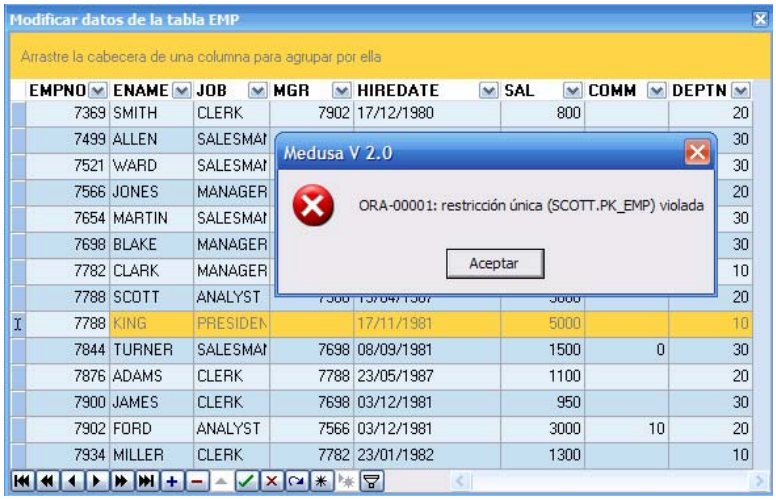


Figura 7.33: Error al intentar cambiar el número del empleado King

7.6.2 Filtro de objetos

El explorador de objetos nos muestra los objetos de un determinado tipo en su carpeta correspondiente, pero ¿qué objetos muestra? La respuesta es todos aquellos que cumplan el filtro superior que esté seleccionado.

Encima de la vista en árbol tenemos una caja de selección y un botón con el símbolo + a su izquierda. En la caja podemos seleccionar el filtro que deseamos que cumplan los objetos y si pulsamos el botón situado a su izquierda, abriremos el formulario de configuración de filtro personali-

zado. *Medusa 2* tiene capacidad para 4 filtros, tres predefinidos y uno personalizado. Los filtros predefinidos son:

- **Todos los objetos**: Muestra todos los objetos que son accesibles por el usuario.
- **Mis objetos**: Muestra todos los objetos propiedad del usuario
- **Objetos ABD**: Muestra todos los objetos de la base de datos (sólo accesible si se está conectado como administrador).

El diálogo para personalizar el filtro lo tenemos en la Figura 7.34, y tiene las siguientes características:

- **Nombre del filtro**: Es el nombre que aparecerá en la caja de selección para que recordemos la utilidad del filtro.
- **Propiedad**: Según la propiedad que tengamos seleccionada nos aparecerá el valor que debe de cumplir a la derecha, las propiedades disponibles son:
  - **Propietario:** Lo utilizaremos si queremos ver los objetos propiedad de un determinado usuario
  - **Fecha de creación**: Filtraremos los objetos teniendo en cuenta su fecha de creación.
  - **Fecha de último uso**: Filtraremos los objetos teniendo en cuenta la fecha en la que fueron utilizados por última vez.
  - **Estado**: Puede ser válido o inválido, si un objeto tiene un estado inválido es porque se creó con fallos de compilación.

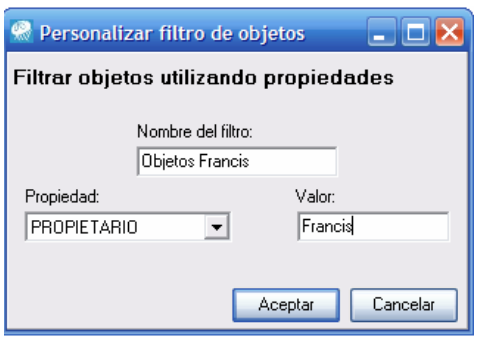


Figura 7.34: Personalizando el filtro a los objetos del usuario Francis

7.7 Menús de Medusa 2

En la Sección 7.2 vimos una breve introducción a cada menú. Aquí explicaremos con más detenimiento las opciones que forman los distintos menús, haciendo mayor hincapié en las características más complejas.

7.7.1 Menú Archivo

En esta sección veremos las opciones más importantes de este menú. Tal y como se detalló en la Sección 7.2.1, este menú está compuesto por las opciones más habituales de trabajo con ficheros. Entre estas opciones detallamos a continuación las siguientes.

- Nuevo. Crea una nueva sesión.
- Abrir. Permite abrir un documento del disco de extensión *sql* (se incluye una lista con los documentos abiertos recientemente).
- Cerrar. Cierra la sesión activa.
- Guardar. Guarda en disco la ventana de edición de la sesión como un documento *sql*.
- Guardar como. Permite guardar con otro nombre.
- Exportar. Exporta los resultados de la ventana activa y/o la ventana de edición a otros formatos (como XML o Excel). Se pueden exportar tanto sentencias como *scripts*. En el caso de las sentencias, si se ha ejecutado una consulta, también se exportarán los resultados que devuelve Oracle. Los distintos formatos de exportación, así como sus características se explican con más detalle en la Sección 7.7.1.1.
- Previsualizar. Permite previsualizar la ventana de edición antes de imprimir (ver Sección 7.7.1.2).
- Configurar la impresora. Nos permite configurar los parámetros de la impresora antes de imprimir.
- Imprimir. Imprime la ventana de edición de la sesión activa.

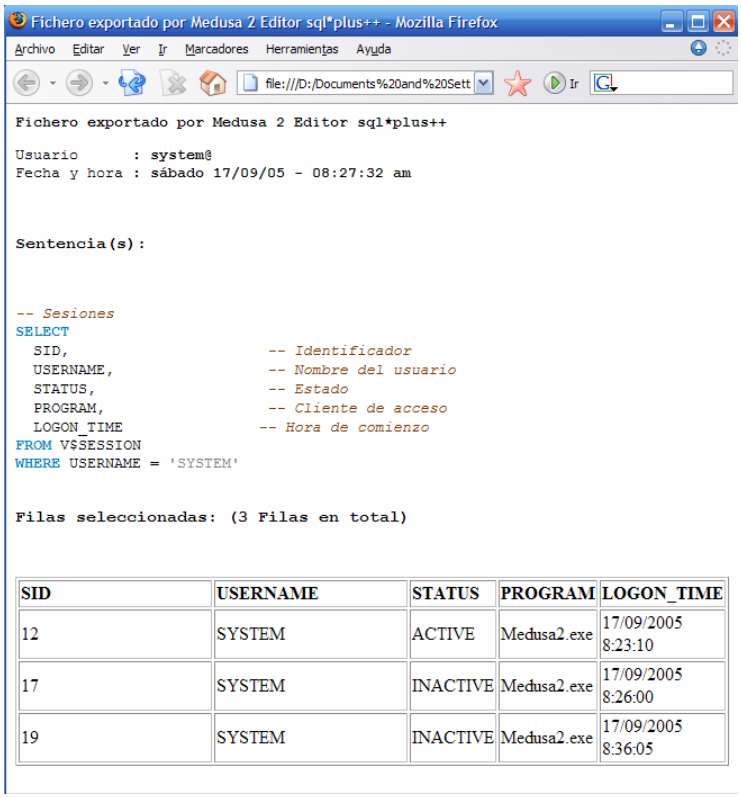
7.7.1.1 Exportar

Para exportar a cualquiera de los formatos disponibles en *Medusa 2* accederemos al menú *Archivo* | *Exportar* y seleccionaremos el formato deseado. En principio veremos que están separados los formatos en dos partes diferenciadas, por un lado los formatos TXT y HTML y por otro los formatos Excel y XML. Esto es así, porque los dos primeros van a exportar el texto de la ventana de edición de la sesión activa además de los resultados, si los hay, y los dos siguientes exportarán tan sólo los resultados. Nótese que tanto en Excel como en XML sólo tiene sentido exportar un conjunto de resultados.

La activación de los formatos de exportación anteriormente nombrados se hará cuando existan resultados en la sesión activa, y se desactivarán si no existen resultados.

A continuación describiremos algunos detalles sobre la exportación al formato HTML, completamente similar al TXT, y en las Figuras 7.35, 7.36, 7.37 y 7.38 veremos imágenes sobre los distintos formatos de exportación.

7.7.1.1.1 Exportando un fichero a HTML



SID	USERNAME	STATUS	PROGRAM	LOGON_TIME
12	SYSTEM	ACTIVE	Medusa2.exe	17/09/2005 8:23:10
17	SYSTEM	INACTIVE	Medusa2.exe	17/09/2005 8:26:00
19	SYSTEM	INACTIVE	Medusa2.exe	17/09/2005 8:36:05

Figura 7.35: Documento HTML exportado abierto con Firefox

En el documento exportado, se muestra la versión de *Medusa 2* con que fue exportado el fichero y la fecha de exportación del documento. También podemos ver 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 que 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 sentencia. En este caso se exportarán los resultados dependiendo de si se ha lanzado una consulta o no. Si la consulta se modifica después de obtener los primeros resultados y se exporta sin ejecutar de nuevo la consulta, *Medusa* no será capaz de reconocerlo y el resultado será incoherente.

7.7.1.1.2 Exportando un fichero a TXT

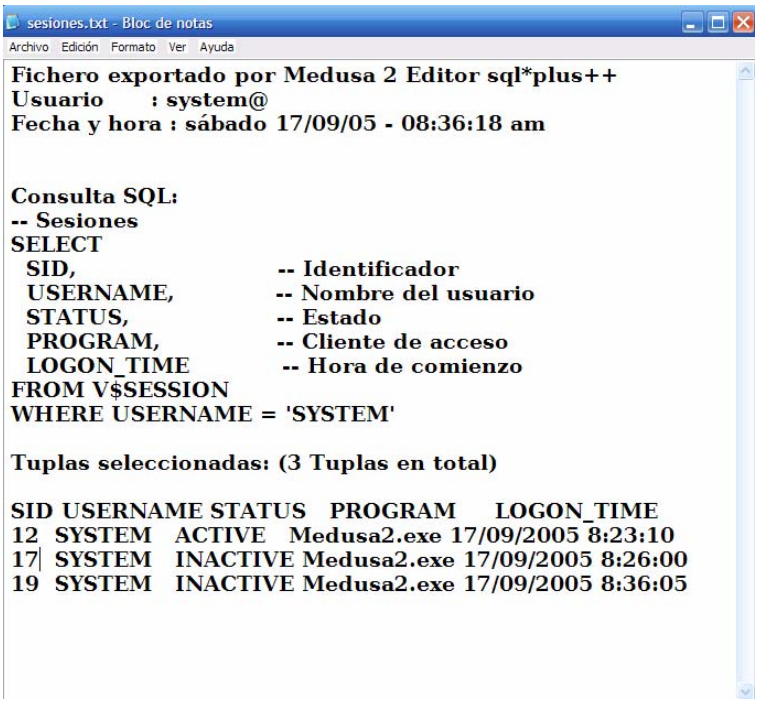


Figura 7.36: Documento TXT exportado abierto con el Bloc de notas

7.7.1.1.3 Exportando un fichero a XML

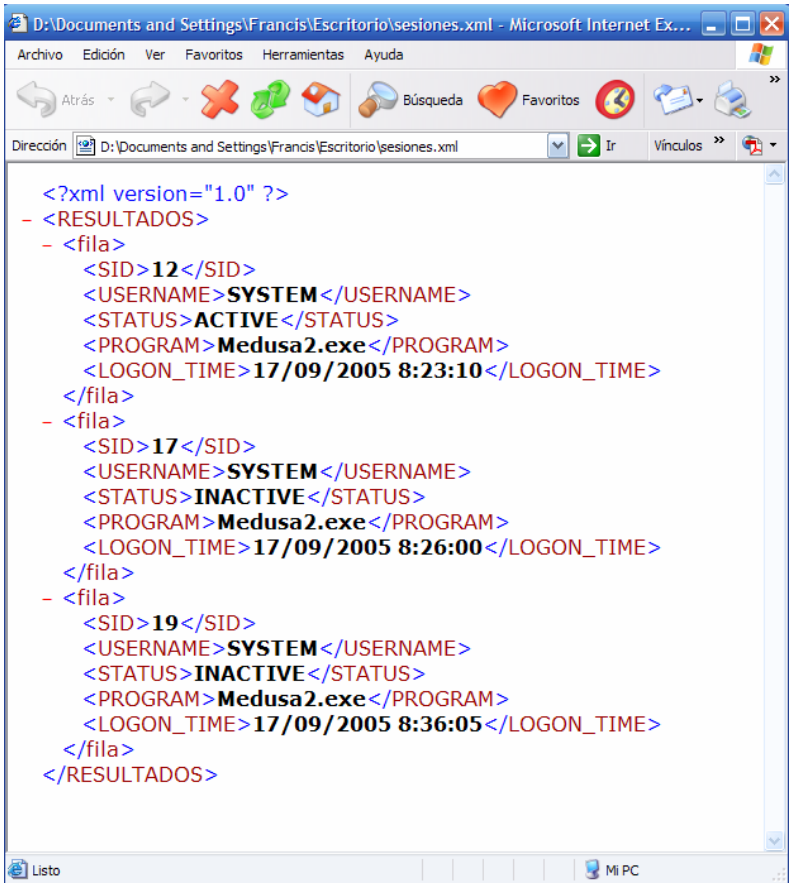


Figura 7.37: Documento XML exportado abierto con Internet Explorer

7.7.1.1.4 Exportando a Excel

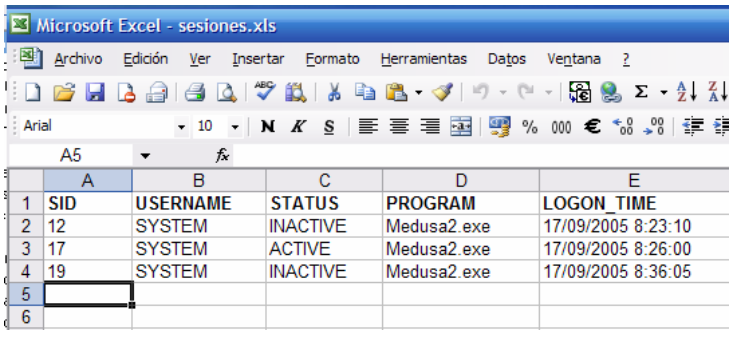


Figura 7.38: Documento Excel exportado abierto con Excel 2003

En la Figura 7.38 podemos ver el fichero Excel exportado por nuestra aplicación abierto con Microsoft Excel. Los resultados se mostrarán en una hoja alineados por filas y columnas tal y como los teníamos en *Medusa 2*. La 1ª fila corresponde al nombre de las columnas.

7.7.1.2 Previsualización de impresión

De forma similar, si queremos previsualizar el documento, basta con hacer clic 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 7.7.4.1).
- 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 7.39 no se alcanza a ver el pie de página, aunque en ese momento si está activa la opción.

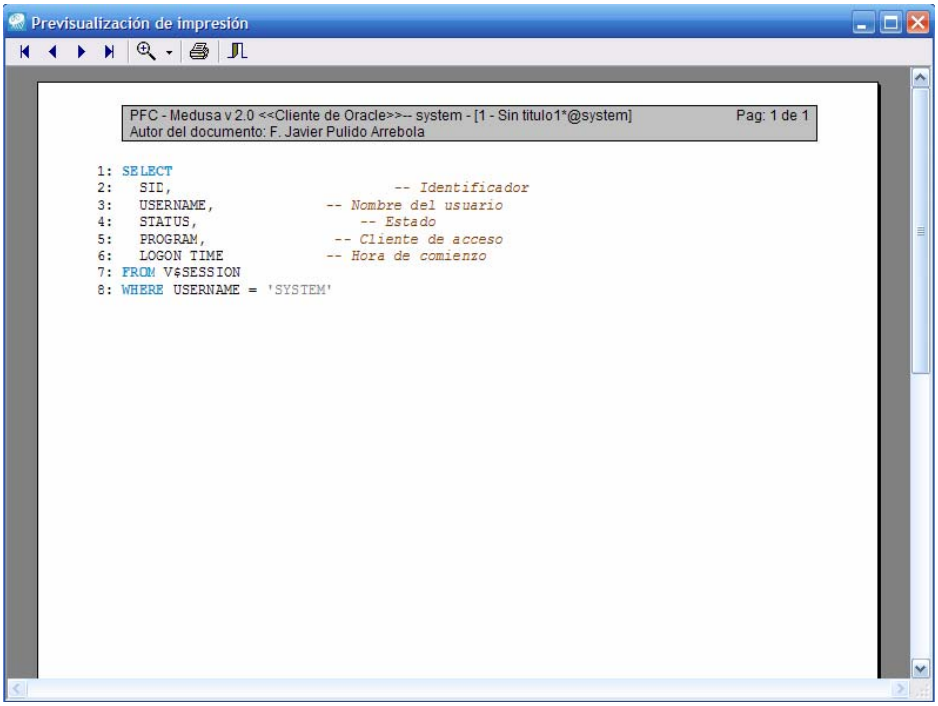


Figura 7.39: Previsualizando un documento

7.7.2 Menú Edición

En esta sección veremos las opciones más importantes de este menú. Tal y como se detalló en la Sección 7.2.2, este menú está compuesto por las opciones más habituales de edición del texto. Entre estas opciones detallamos a continuación las siguientes:

7.7.2.1 Búsqueda de texto

Este formulario es muy simple, y se emplea para buscar un texto determinado en una ventana de edición. El aspecto de este formulario, se puede ver en la Figura 7.40. 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.

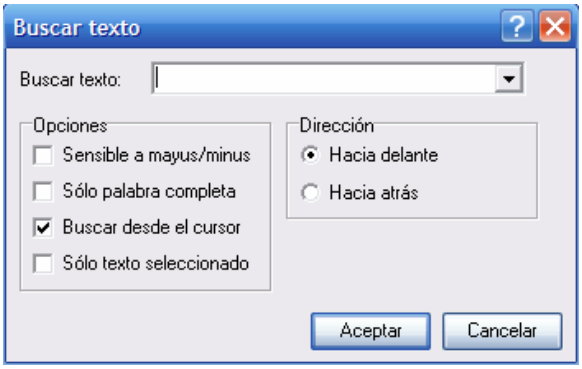


Figura 7.40: Aspecto del formulario de búsqueda de texto

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.

7.7.2.2 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 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 7.41.

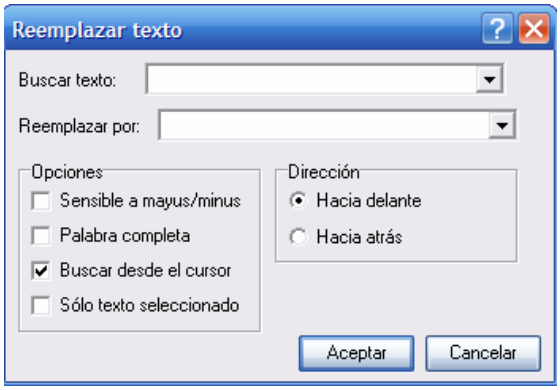


Figura 7.41: Aspecto del formulario de *reemplazado de texto*

Al encontrar un texto a reemplazar se pedirá una confirmación. El aspecto de este formulario, se puede observar en la Figura 7.42. Se puede observar, que es un formulario muy simple, con únicamente cuatro botones.

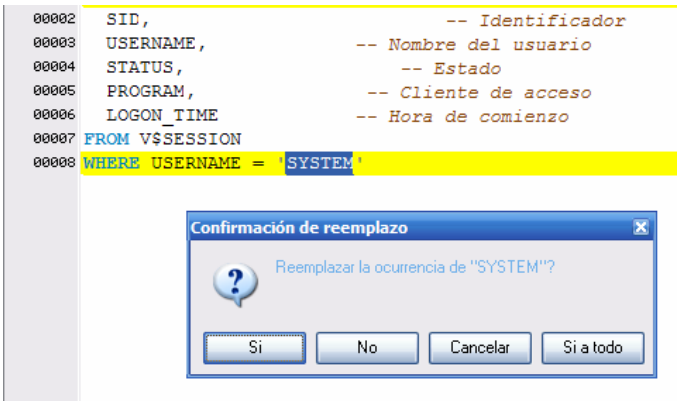


Figura 7.42: Aspecto del formulario de *confirmación de reemplazo*

El primer botón *Sí*, 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 *Sí 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.

7.7.2.3 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 2*, 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 7.43 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.

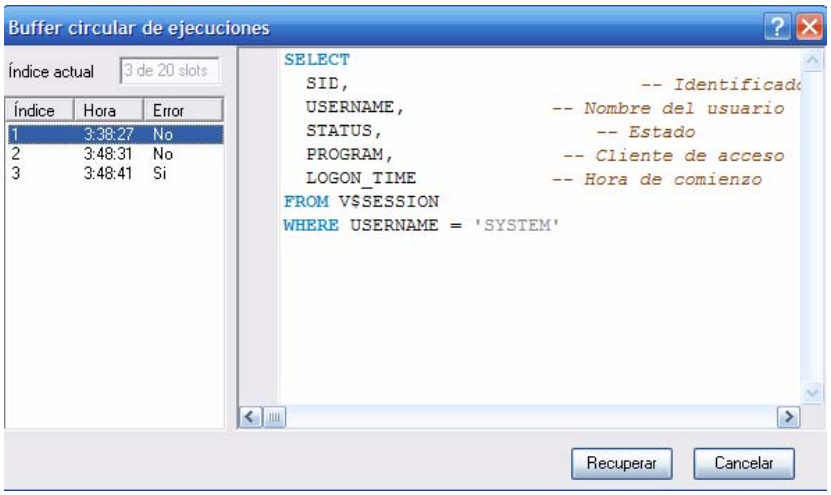


Figura 7.43: Aspecto del *buffer* de textos circular configurable

Se puede recuperar el texto del *buffer* seleccionando un *slot* o elemento de la lista de la izquierda y pulsando el botón *Recuperar*, 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. En el cuadro superior se muestra el índice más reciente, que en este caso es el número tres, 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.

7.7.2.4 Autocompletado por código

Medusa 2 dispone del 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 7.8.6, donde también se detalla cómo personalizar la lista.

7.7.2.5 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.

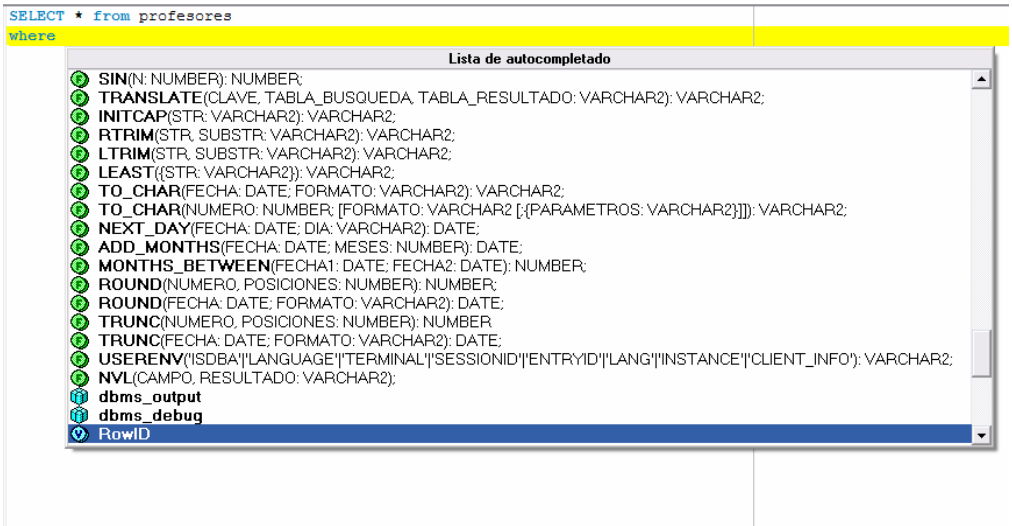


Figura 7.44: Lista de autocompletado

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 7.44 se observa el aspecto de la lista de autocompletado, donde el elemento seleccionado es la variable *RowID*.

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*.

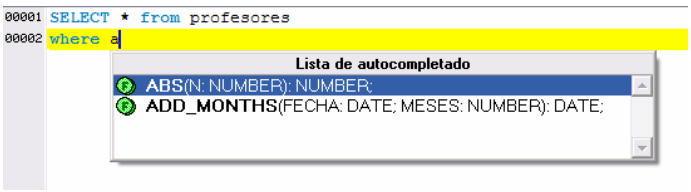


Figura 7.45: Aspecto de la lista de autocompletado filtrada por la letra *"a"*

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 Figura 7.45, 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"*.

7.7.3 Menú Consultas al diccionario de datos

Medusa 2 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, clusters, vistas, tipos, bibliotecas, secuencias, 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 2*, son consultas con comentarios, donde se comenta brevemente la utilidad de esa vista y el significado de cada columna, de forma similar ha como se muestra en la Figura 7.43, salvo que en ese caso se eliminaron algunos comentarios, algunas columnas y se ha añadido una cláusula WHERE.

7.7.4 Menú Herramientas

En este menú de la aplicación tenemos la lista de programas favoritos, que ya se comentará en el Apartado 7.7.4.1.4, el acceso al menú de configuración, comentado en la Sección 7.7.4.1, y el botón correspondiente al tiempo de sentencias. Si el botón correspondiente al tiempo de sentencias y representado por un cronómetro está pulsado (aparece como hundido en la barra de herramientas) cada vez que ejecutemos una sentencia o *script* nos aparecerá un mensaje con el tiempo empleado. En cambio, si el botón no aparece pulsado nunca nos aparecerá un mensaje con el tiempo empleado.

Medir el tiempo que tarda en ejecutarse una sentencia o *script* nos puede ayudar a comprobar si los cambios que hemos realizado en los distintos componentes que están involucrados, han conseguido reducir el tiempo de acceso, o por el contrario, no lo han conseguido. También nos puede permitir cuantificar de alguna forma la repercusión del tamaño de resultados que devuelve el SGBD y el tiempo empleado en la ejecución.

En la Figura 7.46 vemos el diálogo que nos muestra el tiempo empleado en ejecutar el *script* escrito en la ventana de edición.

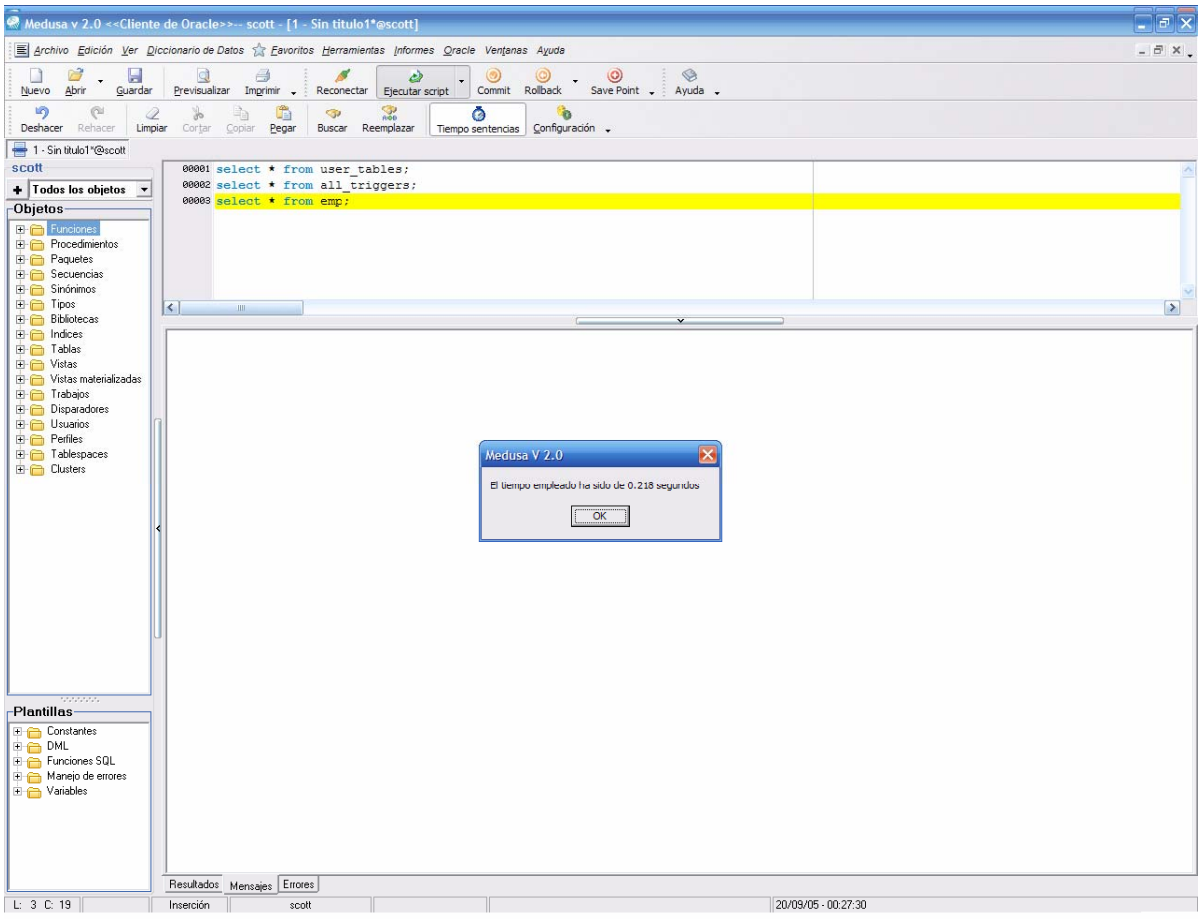


Figura 7.46: Diálogo que nos informa del tiempo empleado en ejecutar la sentencia o *script*

7.7.4.1 Opciones de configuración

Prácticamente todas las opciones de configuración de *Medusa 2*, 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 2*, 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.

7.7.4.1.1 Configuración del editor

Tal y como se observa en la Figura 7.47, se incluyen numerosas opciones de configuración del editor, entre las que podemos encontrar:

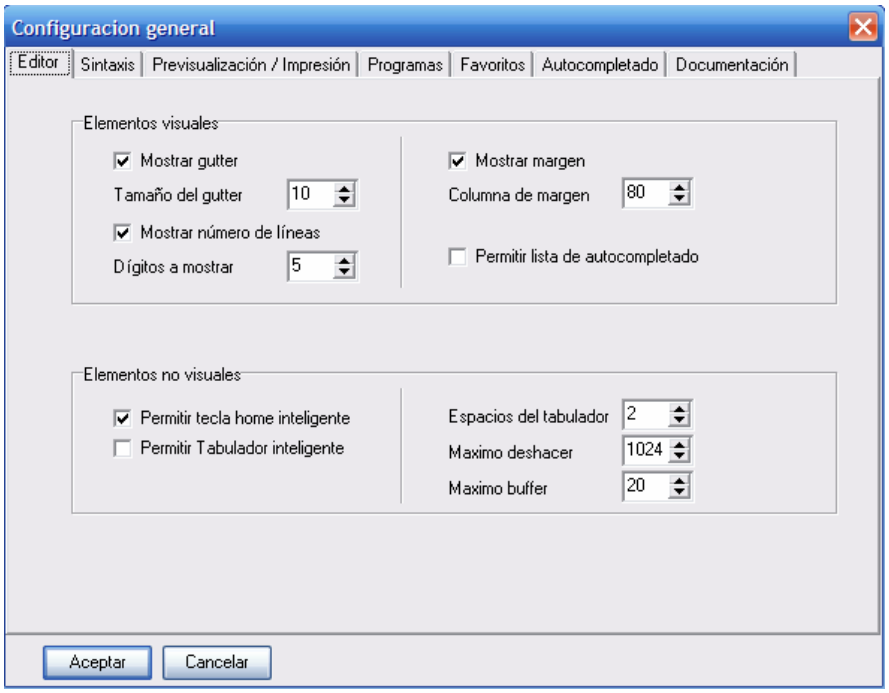


Figura 7.47: Pestaña Editor de las opciones de configuración

- 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 lista de autocompletado. En el caso de que esté activada cuando pongamos un punto en el editor nos saldrá automáticamente la lista de autocompletado.
- 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 7.7.2.3. Por defecto 20.

7.7.4.1.2 Configuración del realzado de sintaxis

Tal y como se observa en la Figura 7.48, 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, con la que sólo se resaltan en negritas las palabras reservadas.

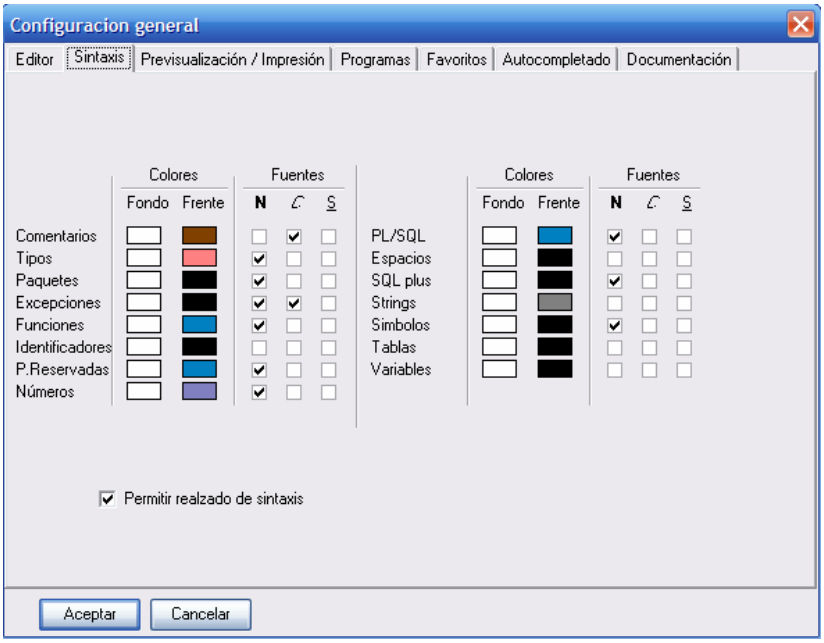


Figura 7.48: Pestaña Sintaxis de las opciones de configuración

7.7.4.1.3 Configuración de la previsualización y de la impresión

El aspecto de esta pestaña de configuración es el mostrado en la Figura 7.49, 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 7.7.1.2 se vio como previsualizar la impresión.

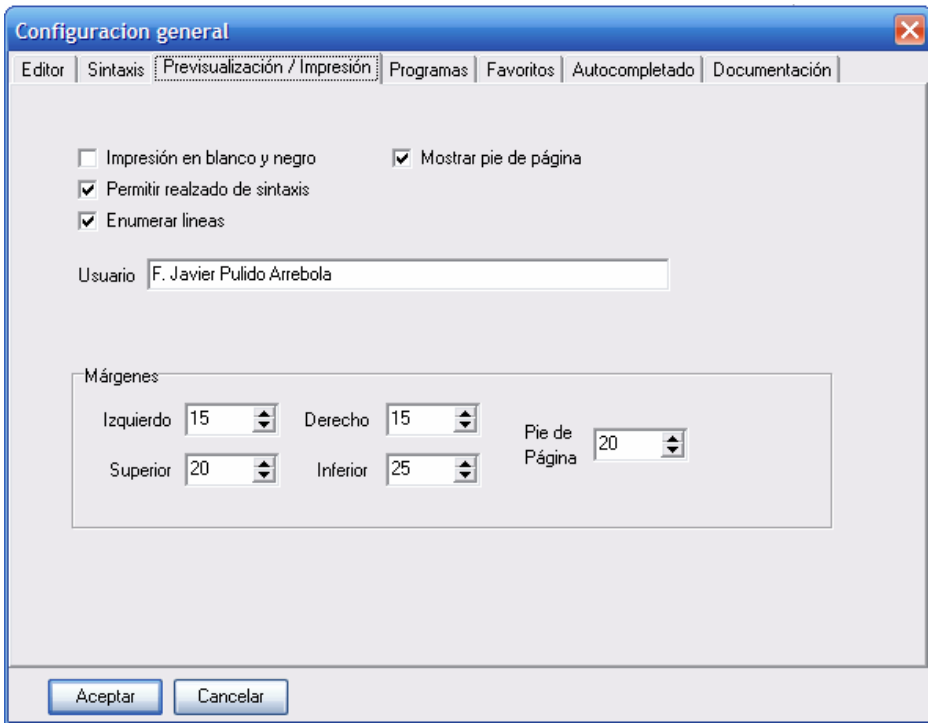


Figura 7.49: Pestaña Previsualización / Impresión de las opciones de configuración

7.7.4.1.4 Configuración de los programas favoritos (menú Herramientas)

El aspecto de esta pestaña de configuración es el mostrado en la Figura 7.51, 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 2* 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 2* 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 2* no tendrá en cuenta las últimas modificaciones.

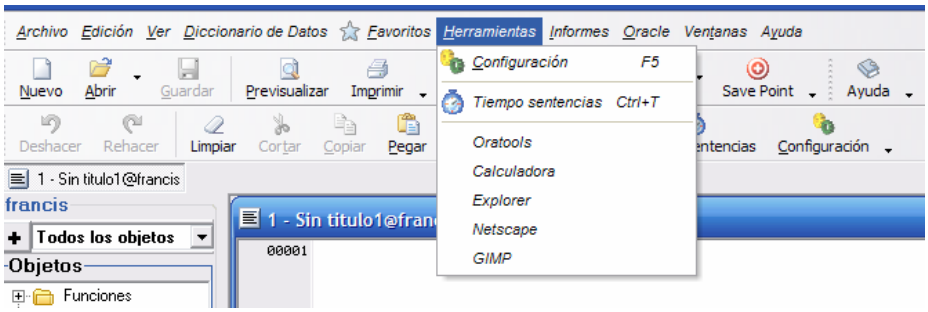


Figura 7.50: Aspecto del resultado en el menú principal

En la Figura 7.50 se puede observar el aspecto del menú que se forma al configurar los programas tal y como se observa en la Figura 7.51.

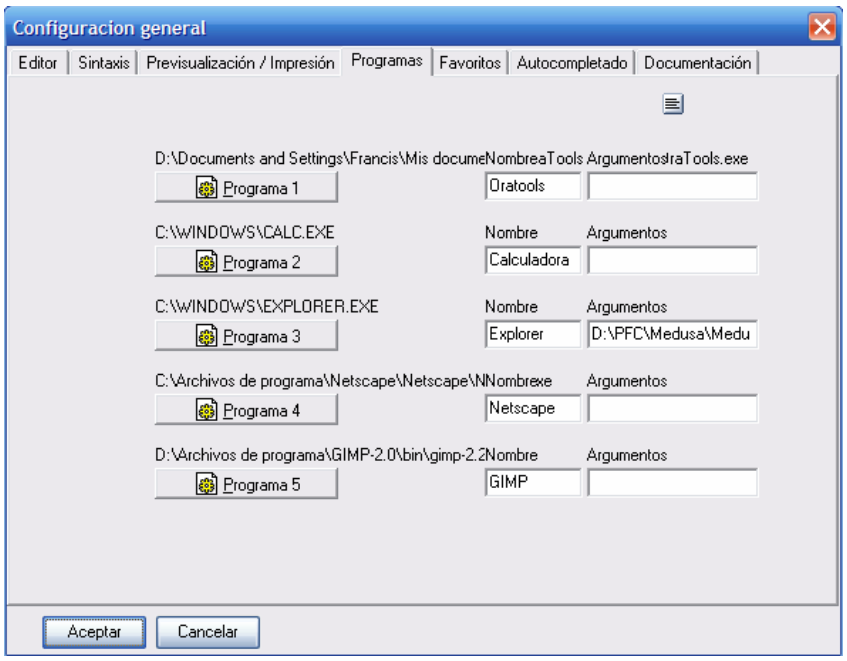


Figura 7.51: Pestaña *Programas* de las opciones de configuración

7.7.4.1.5 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 clic 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.

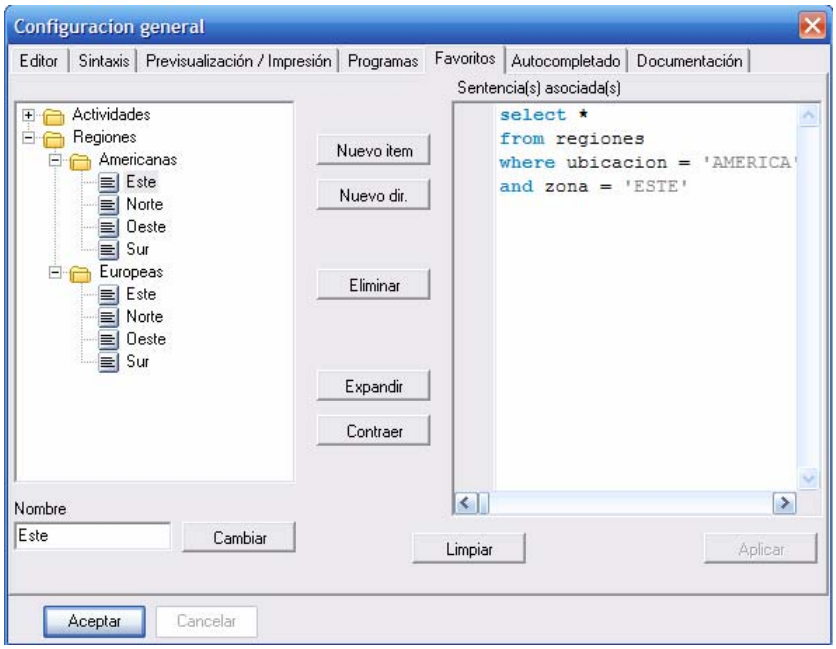


Figura 7.52: Editor de favoritos

El aspecto del gestor de favoritos es el de la Figura 7.52. Se pueden crear *ítems* de sentencias, al igual que directorios y subdirectorios.

Los ítems representan elementos finales, es decir: consultas, sentencias de inserción, bloques PL/SQL, en general, cualquier código de usuario. El *ítem* que tratamos se muestra en la parte derecha de la ventana. Por otro lado las categorías (directorios o carpetas), son contenedores de

ítems, 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 ítems correspondientes.

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

El primer botón (*Nuevo ítem*), 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 *ítem* 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 *ítem*, en la parte inferior donde se indica.

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

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

El cuarto botón (*Expandir*) y el quinto (*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 ítem 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 ítem o categoría que ya existe.

Hay que matizar que no es posible mover ítems 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 ítem, simplemente seleccionándolo y modificándolo en el editor que se encuentra en la derecha.

7.7.4.1.6 Configuración de las plantillas de autocompletado por código

El aspecto de esta pestaña de configuración es el mostrado en la Figura 7.53, 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 7.7.2.5, 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 7.53, 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.

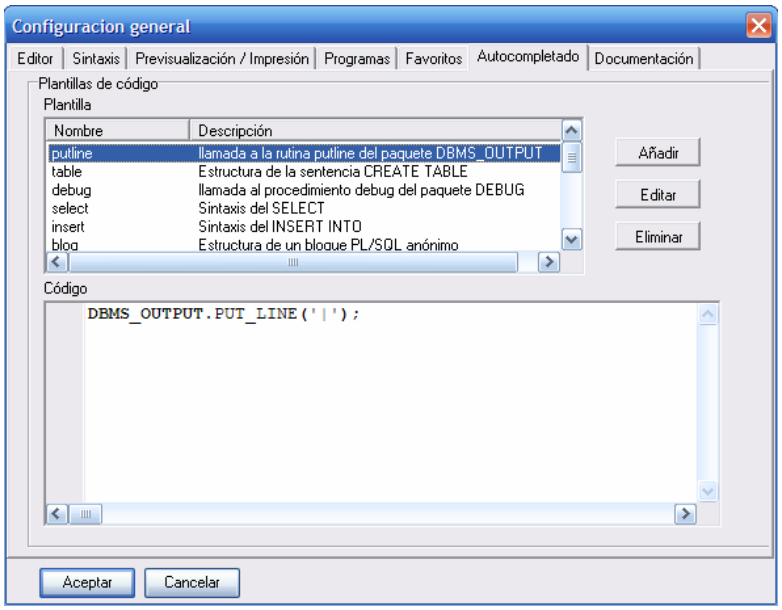


Figura 7.53: Pestaña *Autocompletado* de las opciones de configuración

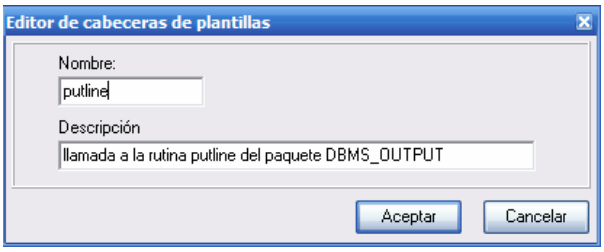


Figura 7.54: Aspecto del formulario de gestión de elementos de autocompletado

7.7.4.1.7 Documentación

El aspecto de esta pestaña de configuración es el mostrado en la Figura 7.55.

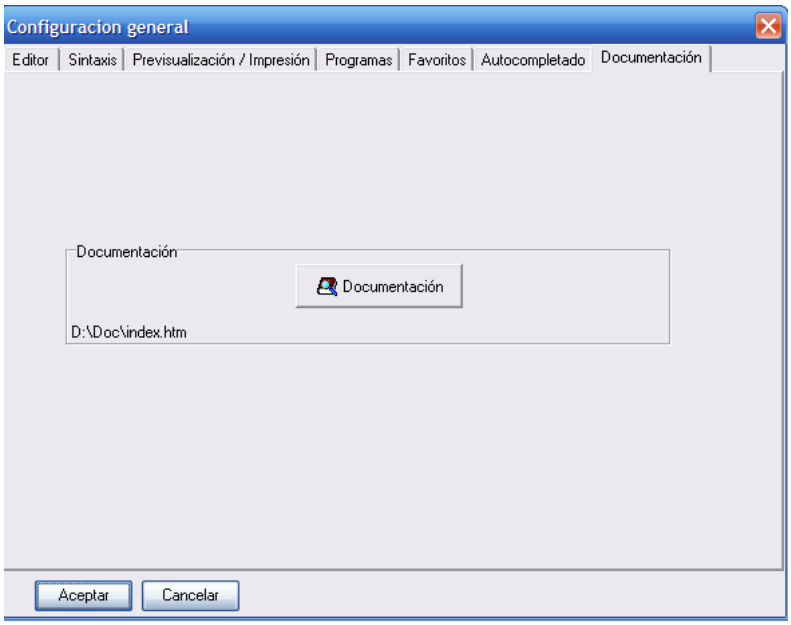


Figura 7.55: Pestaña *Documentación* de las opciones de configuración

Se utiliza para seleccionar el camino hacia la documentación y la ayuda de Oracle. El camino se mostrará justo debajo y en nuestro caso es `D:\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* (Sección 7.7.7).

7.7.5 Menú Informes

Este menú de Medusa 2 posee un par de opciones, pero resultan ser realmente completas. Al pinchar sobre `Generar informe` vemos que nos aparecen dos opciones, `Nuevo` y `A partir de resultados`. La primera de las opciones nos crea un formulario de creación de informes, que utilizaremos para ver a pantalla completa los resultados de una consulta, con los que después elaboraremos el informe. La segunda opción genera directamente la previsualización del informe a partir de los resultados que tengamos en el área de resultados de la sesión activa. Por último, tenemos el explorador de informes del que hablaremos un poco más adelante.

7.7.5.1 Generar un nuevo informe

En la Figura 7.56 podemos ver el aspecto del formulario de creación de nuevos informes. Aunque se puede crear un informe desde casi cualquier rejilla de resultados del programa, es importante poder ver a pantalla completa los resultados. Una importante cuestión que hay que tener en cuenta cuando generemos informes es que la previsualización del informe va a ser exactamente igual a como estemos viendo los resultados en pantalla, por poner un ejemplo, si redimensionamos una columna y la hacemos el doble de ancha, en la previsualización del informe veremos que el cambio también se ha visto reflejado. También es importante destacar que una vez que estemos previsualizando el informe, podremos configurar a nuestro antojo pero no podremos cambiar la disposición de los resultados, para ello deberemos de volver a generar el informe con los cambios realizados.

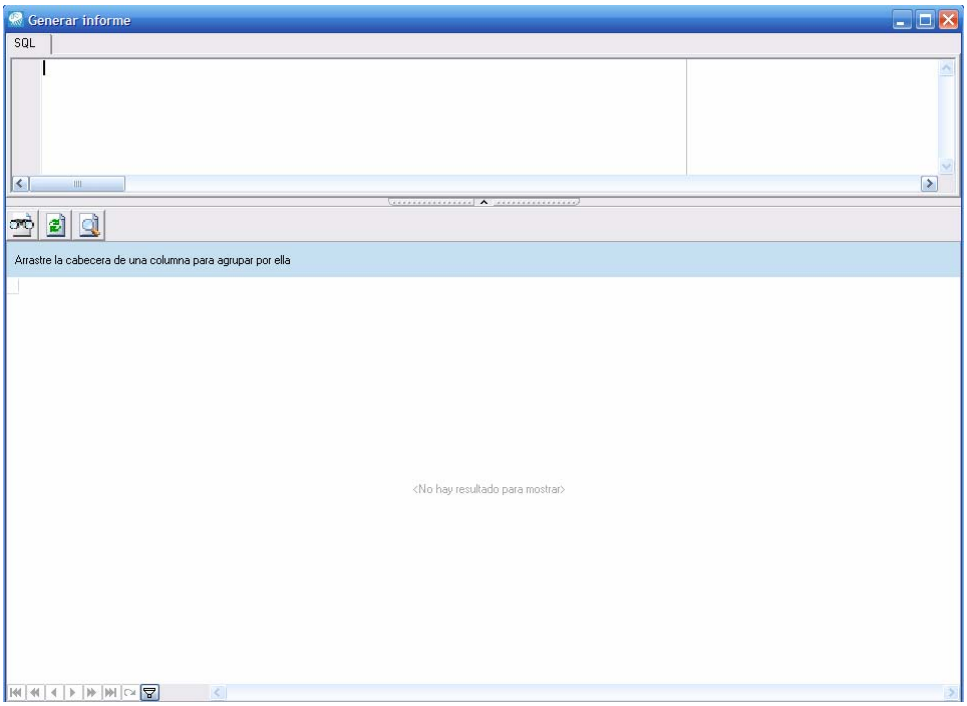


Figura 7.56: Formulario para generar un nuevo informe

El funcionamiento del formulario de generación de nuevos informes es bastante sencillo. Tenemos tan sólo 3 botones con los que interactuar, el de la izquierda ocultará/mostrará la ventana de edición para que una vez que tengamos creado el informe podamos verlo a pantalla completa. El botón central lanza la consulta que haya en la ventana de edición al SGBD, con la consiguiente

respuesta por su parte mandando los resultados a la rejilla inferior. El tercer botón nos envía directamente a la previsualización de informes, que vamos a describir a continuación.

7.7.5.2 Previsualización de informes

Una vez que tengamos preparados los datos que queremos incluir en el informe, hayamos definido los filtros, los agrupamientos y las ordenaciones que necesitemos, procederemos a previsualizar nuestro informe. En la Figura 7.57 podemos ver la previsualización de un informe creado con los datos de la tabla EMP, que ya hemos utilizado anteriormente, agrupados por el número de departamento (*deptno*), filtrados para ver tan sólo los empleados con unas ciertas profesiones (*Manager* o *Salesman*) y ordenados primero por el número de departamento, ascendentemente, y dentro del agrupamiento, por el salario de forma descendente.

Los menús que poseemos en este completo formulario los describiremos a continuación, al hablar sobre el explorador de informes, puesto que son formularios iguales excepto con la particularidad de que el explorador tiene un submenú llamado `explorador` para explorar el disco en busca de informes. Además desde ambos sitios podremos modificar la configuración del informe (no los datos, como hemos explicado anteriormente).

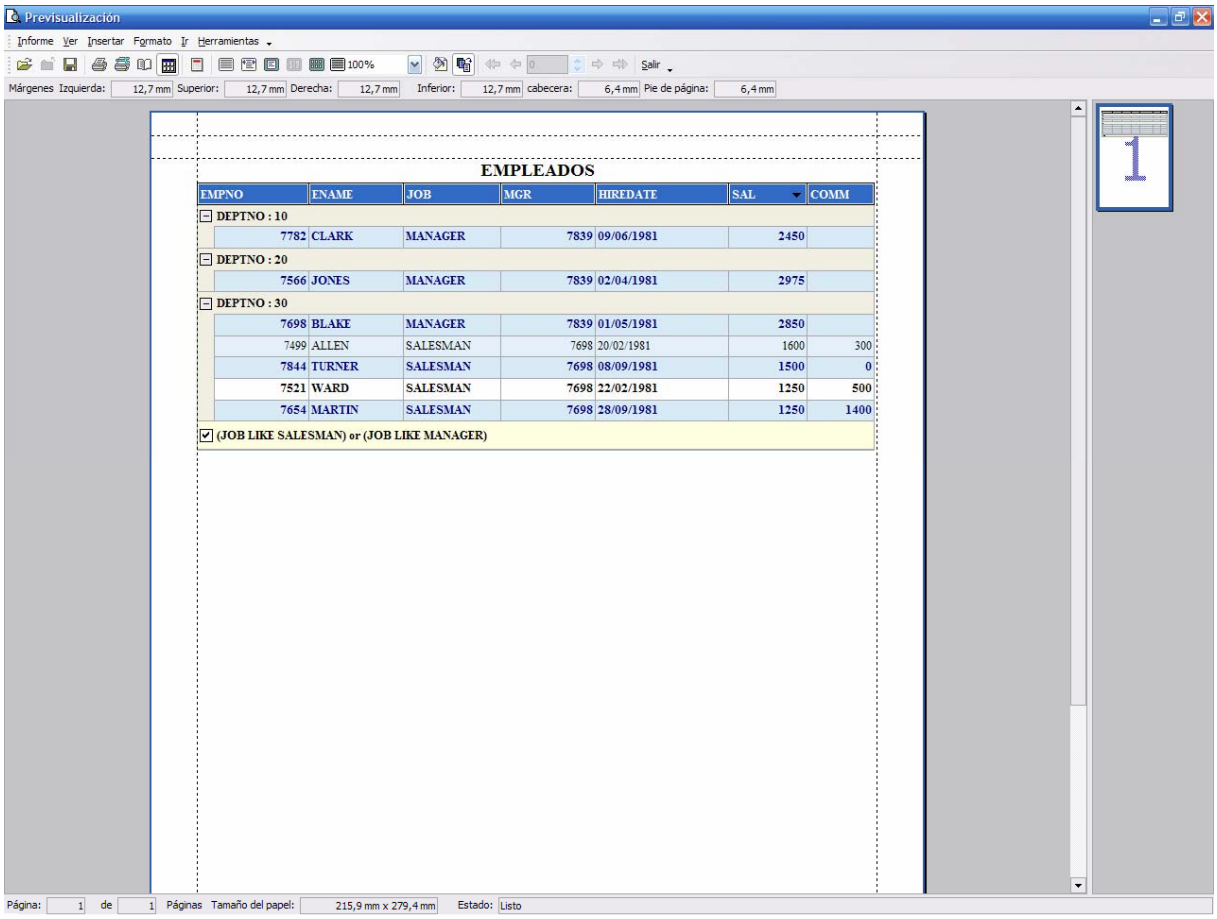


Figura 7.57: Previsualización de informe

7.7.5.3 Explorador de informes

El formulario de exploración de informes nos va a servir para organizar y examinar los informes que hemos ido creando con *Medusa 2*. Posee también un menú y una barra de herramientas del mismo aspecto y funcionalidad que el que poseemos en la ventana principal de la aplicación. La función de cada menú y sus opciones las explicamos a continuación:

- **`Informe:`**

  - **Reconstruir**: Actualiza el informe con las características que hemos seleccionado.
  - **Cargar**: Carga el informe que tengamos seleccionado en el explorador.
  - **Cerrar**: Cierra el informe activo.
  - **Guardar**: Guarda el informe activo.

- **Imprimir**: Abre el diálogo de impresión sobre el informe activo.
- **Configurar página**: Abre el diálogo de configuración de página que podemos ver completo en las Figuras 7.58 y 7.59.
- **Salir**: Abandonar el explorador de informes.

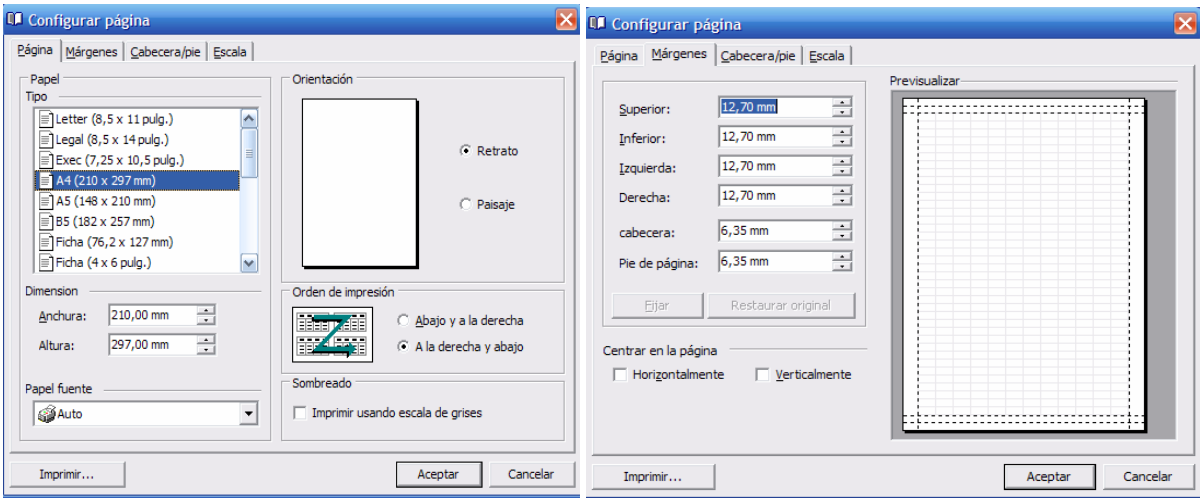


Figura 7.58: Pestañas `Página` y `Márgenes` del formulario de Configuración de página

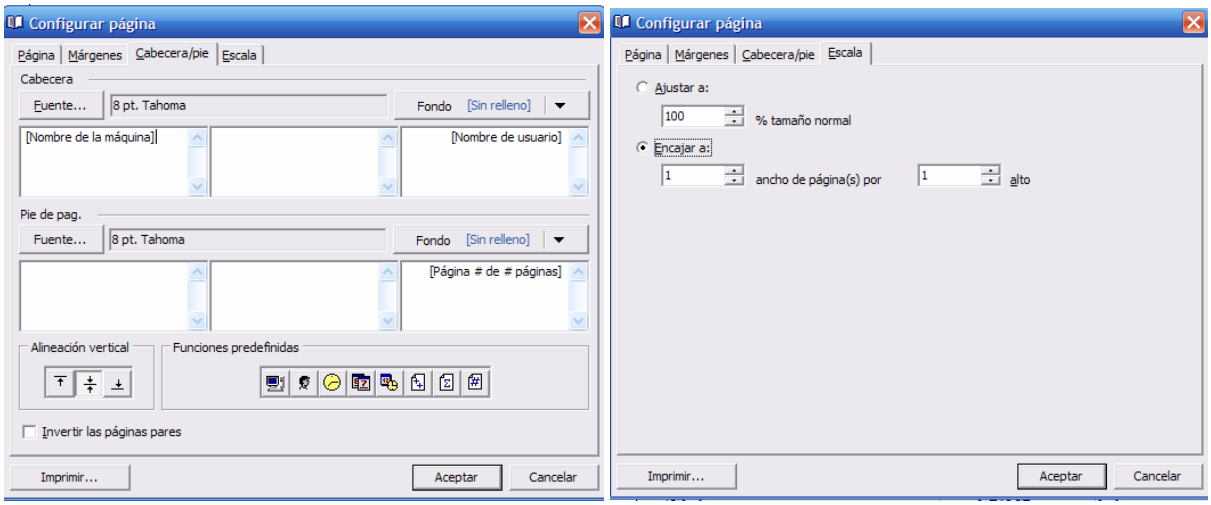


Figura 7.59: Pestañas `Cabecera/Pie` y `Escala` del formulario de Configuración de página

En la Figura 7.58 (pestaña `Página`) vemos como vamos a tener opciones de definir el tipo de papel, la orientación, las dimensiones del papel, el orden de impresión o si queremos imprimir en blanco y negro. En la pestaña `Márgenes` definiremos el tamaño de los márgenes que va a tener la página que imprimiremos.

En la Figura 7.59 (pestaña Cabecera/Pie) vemos un lugar donde podemos configurar la cabecera y pie del documento. En la figura vemos como hemos utilizado una variable propia que indica el nombre de la máquina en la parte izquierda del encabezado, el nombre de usuario (de Windows) en la parte derecha y el número de página sobre el total en la parte derecha del pie de página. También podremos escribir un texto personalizado en vez de utilizar las funciones predefinidas del programa. Para saber qué hace cada función, bastará con poner el puntero encima y el sistema nos informará. La opción de *invertir las páginas pares* la utilizaremos en caso de que queramos encuadernar los informes.

En la pestaña Escala podremos configurar que el informe se cree a una escala distinta del 100%. Será útil para crear varios informes en una página o incrementar el tamaño de informes con pocos datos.

- **Explorador:**

  - **Subir un nivel**: Sube un nivel en el explorador con respecto al actual.
  - **Cambiar ruta**: Cambia a la ruta especificada.
  - **Refrescar**: Refresca el explorador por si se ha hecho algún cambio desde fuera de la aplicación.
  - **Crear carpeta**: Crea una nueva carpeta que utilizaremos para organizar los informes.
  - **Eliminar**: Elimina la carpeta o informe seleccionado.
  - **Renombrar**: Renombra la carpeta o informe seleccionado.
  - **Propiedades**: Muestra una descripción, que puede editar el usuario, y una previsualización a pantalla completa del informe.

- **Ver:**

  - **Márgenes**: Muestra las líneas imaginarias que definen los márgenes de página.
  - **Barra de márgenes**: Si está activada podremos ver la barra de márgenes.
  - **Barra de estado**: Si está activada podremos ver la barra de estado.
  - **Explorador**: Si está pulsado veremos a la izquierda un árbol con las carpetas e informes de que disponemos.

- **Vistas en miniatura**: Si está pulsado veremos a la derecha unas vistas en miniatura que representan a las distintas páginas que forman el informe.
- **Cabecera y pie**: Si está pulsada veremos la barra de cabecera y pie de página, por defecto está desactivada.
- ***Zoom***: Nos permite cambiar el tamaño de visualización del informe. Existen muchas opciones como definir el % del tamaño de página que queremos ver, ver toda la página, varias páginas, ajustar al ancho de página…
- **Cabecera de página**: Si está activado podremos ver el texto que hayamos colocado en la cabecera, sino lo está no lo veremos.
- **Pie de página**: Exactamente igual que el anterior pero aplicado al pie de página.

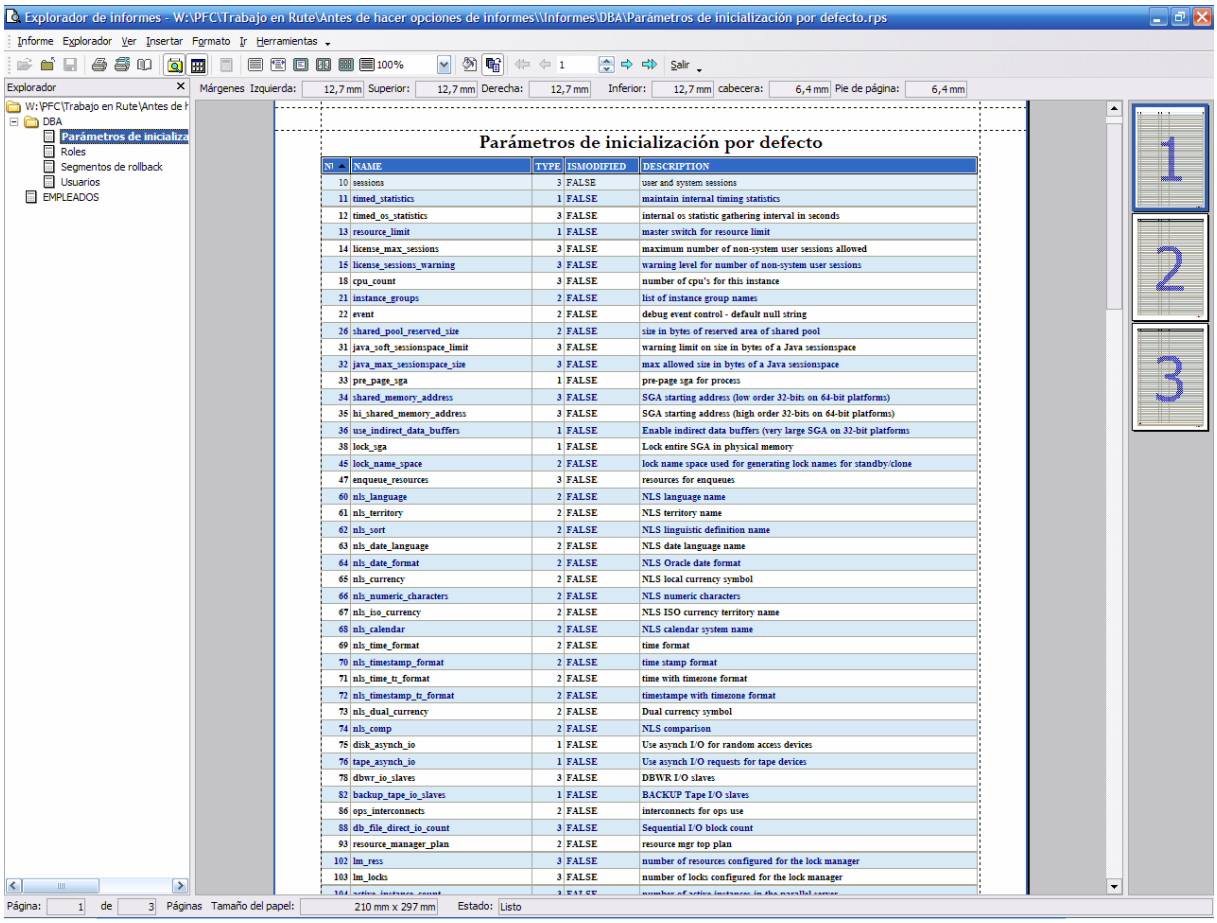


Figura 7.60: Explorador de informes

℘ `Insertar:`

Este menú tiene todas las funciones predefinidas que podemos insertar en la cabecera y pie de página, así que si no estamos modificando ninguno de estos dos, todas las opciones del

menú permanecerán desactivadas. Las funciones predefinidas ya se han explicado en el formulario de configuración de página (ver Figura 7.59).

℘ **`Formato:`**

- **Título:** Permite definir el título del informe así como el formato del título (tipo de letra, alineación…).
- **Numeración de página:** Define el formato que vamos a utilizar para numerar las páginas (I,II,III…1,2,3…A,B,C…) además del número de la página de inicio.
- **Fecha y hora:** Define el formato de fecha y hora que utilizaremos para las funciones predefinidas.
- **Ajustar a la página:** Ajusta el contenido de los datos a la página. Resulta realmente útil ya que no es normal que los datos se ajusten adecuadamente a la página, y con esta opción el informe quedará con una mejor presentación.
- **Fondo:** Permite definir un fondo que se aplicará a los informes. Puede ser una textura, un gradiente o un color sólido.

℘ **`Ir:`** Tiene las todas las opciones necesarias para navegar por las páginas que conforman el informe. Podemos movernos directamente a la primera, última o al número de página que establezcamos.

℘ **`Herramientas:`** Accediendo a este menú podemos configurar distintas opciones del formulario, como si queremos hacer *zoom* con el botón central del ratón o si queremos poner las unidades de medida en pulgadas, milímetros o centímetros.

7.7.6 Menú Oracle

En esta sección veremos las opciones más importantes de este menú. Tal y como se detalló en la Sección 7.2, 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.

7.7.6.1 Información de la sesión

Podremos tener acceso a esta opción haciendo clic en el menú `Oracle` y después haciendo clic en `Información de la sesión`. Se nos mostrará un formulario con diversa información, no sólo de la sesión, sino de la versión de Oracle instalada, información del usuario (roles y limitaciones) y privilegios de la sesión. En la Figura 7.61 se muestra toda la información que recopila el formulario.

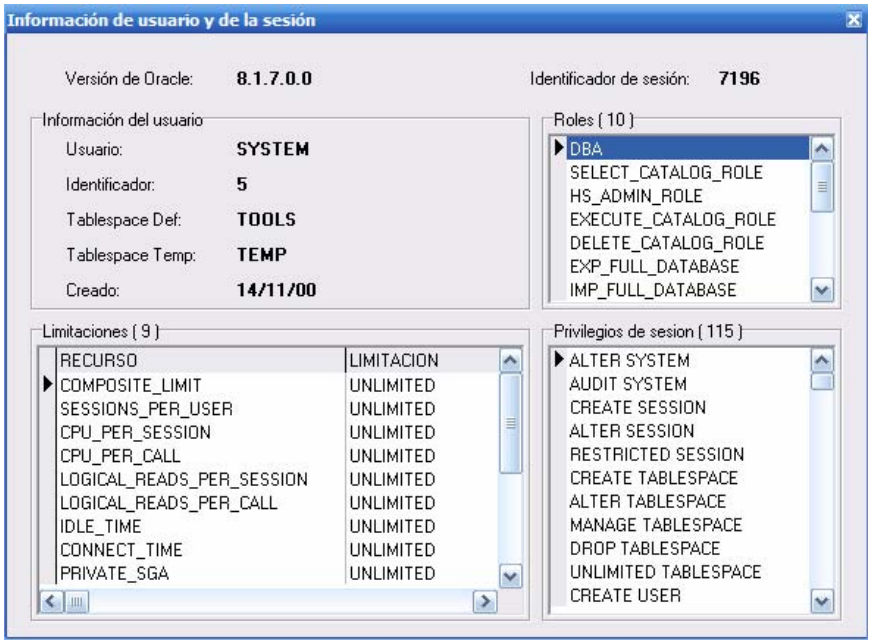


Figura 7.61: Información disponible de la sesión

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

7.7.6.2 Estructura de la Base de Datos

Otra opción que está disponible en *Medusa 2*, es la posibilidad de obtener un listado de los atributos y relaciones de un conjunto de tablas. En la Figura 7.62, 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. 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 7.63 se puede observar el resultado del listado 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.

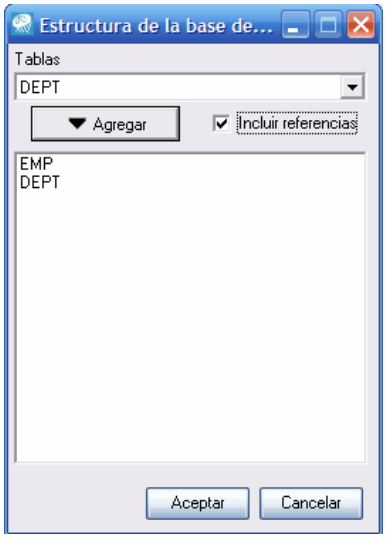


Figura 7.62: Aspecto del formulario de Estructura de la base de datos

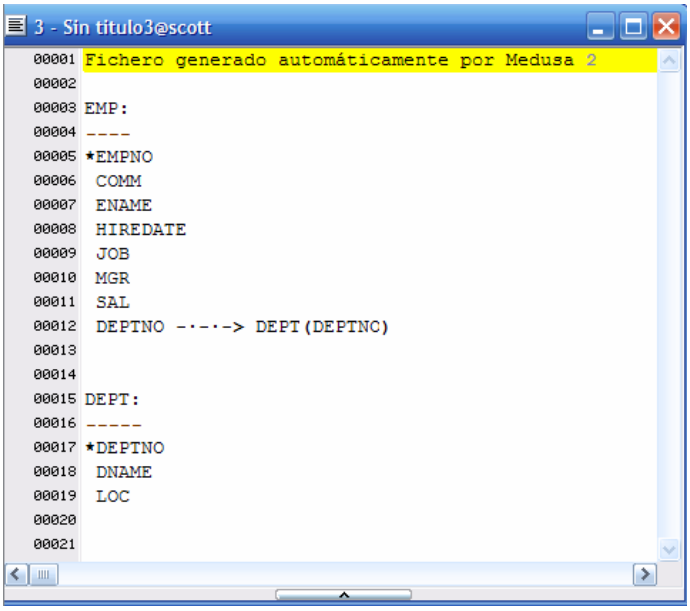


Figura 7.63: Resultado devuelto por Medusa 2

7.7.6.3 Creación 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 2*. El aspecto de los formularios es similar, todos disponen de un botón `Cancelar` y un botón `Aceptar`. También está disponible en casi todos los formularios, excepto en la creación de bibliotecas y trabajos, un botón `Ver SQL` que nos va a permitir ver, grabar y modificar la sentencia que internamente estamos lanzando al SGBD para crear el objeto.

Cuando estemos construyendo el objeto vamos a tener la opción de ver en tiempo real cómo se está realizando la sentencia. Además de eso podremos guardar la sentencia en disco, lanzarla a Oracle, modificarla, por si queremos cambiar algo con lo que no estemos de acuerdo o extender alguna funcionalidad, seleccionar su texto, cortarlo, copiarlo, pegarlo… Para cambiar de nuevo al modo de constructor visual volveremos a pulsar el mismo botón, que ahora habrá cambiado su etiqueta a `Ver Constructor`, y en el caso de que hayamos realizado alguna modificación el sistema nos preguntará si queremos cambiar al constructor perdiendo los cambios, pues no es posible aplicar los cambios al constructor.

7.7.6.3.1 Creación de tablas

Esta opción incluida en *Medusa 2*, facilitará la creación de tablas de forma visual. Sobre todo puede ayudar a los usuarios menos experimentados, aunque también puede ser un buen punto de partida para la creación de tablas por parte de usuarios avanzados. Cuando accedamos al constructor de tablas, ya sea desde el explorador de objetos o desde el menú `Oracle`, veremos que está formado por 4 pestañas (Figura 7.64):

- **`General:`**

  Aquí definiremos cuestiones relacionadas con el almacenamiento de la tabla, su nombre y propietario, si va a pertenecer a algún *cluster*, la duración o los comentarios. En la Figura 7.64 podemos ver la pestaña indicada anteriormente. De arriba abajo y de izquierda a derecha vamos a describir cada componente del formulario. Primero definiremos el propietario y nombre de la tabla, después si queremos especificar algún espacio de tablas donde guardarla seleccionaremos de entre las posibles, si queremos que Oracle la almacene en la predeterminada simplemente no escogeremos ninguna.

  El *PCTFREE* nos indica el mínimo porcentaje de página que se reservará como espacio libre para futuras actualizaciones de filas que ya existen en la página. El *PCTUSED* es el porcentaje al que debe de descender el espacio libre para que se vuelvan a introducir filas. En el formulario los vemos configurados a 10 y 40 respectivamente, pues es el valor por defecto que utiliza Oracle. También es importante reseñar que la suma de ambos no puede superar 100, aunque de eso no debemos preocuparnos ya que el sistema no nos dejará superar esa suma.

  Por otro lado, podemos incluir la tabla dentro de un cluster existente, para ello escogeremos el cluster de entre los posibles en la lista despegable, y si queremos especificar alguna columna de la tabla lo haremos en el siguiente cuadro de edición. Una importante reseña en cuanto a este cuadro de edición se refiere es que todos los que veamos en el constructor con una etiqueta "…" nos abrirán un formulario como el de la Figura 7.66 donde podemos escoger las columnas, definidas en la pestaña `Columnas`, que queremos incluir.

  Además de estas características que hemos descrito anteriormente, vamos a poder crear tablas temporales (cuando activemos la opción se desactivarán lógicamente las opciones de almacenamiento y *cluster*) con sólo activar la casilla correspondiente. Tam-

bién podemos escoger si queremos que las filas que introduzcamos sigan después de hacer un *commit* o no.

Por último podemos escribir un comentario, que nos sirva tanto a nosotros como a otros administradores que utilicen la tabla, para describir cualquier cosa que nos parezca interesante sobre la tabla.

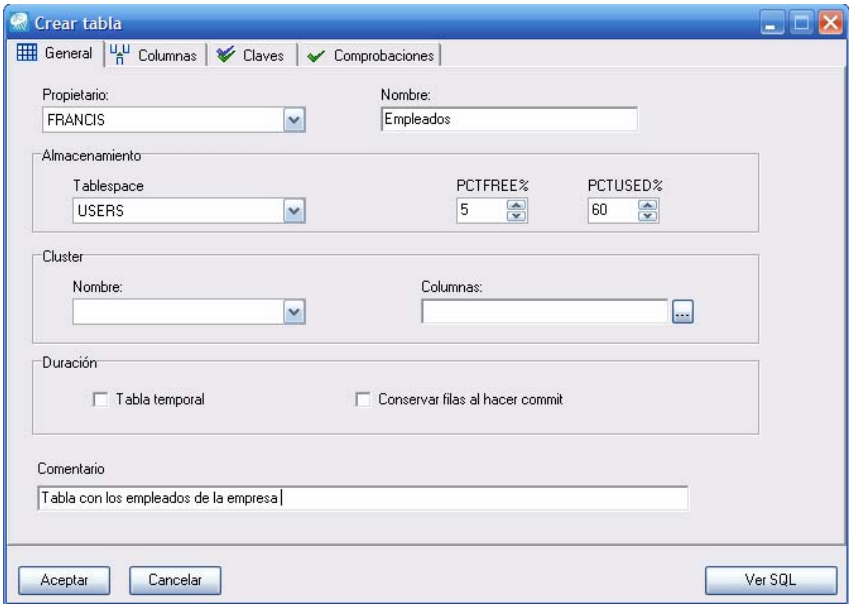


Figura 7.64: Pestaña General del constructor de tablas

℘ `Columnas:`

En esta pestaña, que podemos ver en la Figura 7.65, definiremos las columnas que queremos que tenga nuestra tabla. Al igual que el resto de pestañas el manejo es muy intuitivo y para crear una nueva columna simplemente pincharemos sobre `Añadir columna` y después definiremos sus características. Si queremos añadir otra columna volveremos a pulsar sobre el botón y si queremos eliminar alguna la seleccionaremos en la lista superior y pincharemos sobre `Eliminar columna.` Las características de una columna que tenemos que rellenar son: el nombre, el tipo (los disponibles son Number, Varchar2, Date, CLOB, BLOB y LONG), la longitud (si hemos elegido un tipo numérico o Varchar2), y en el caso de que queramos que la columna no admita valores nulos activaremos la casilla *Not Null*. También podemos definir una restricción sobre la columna para que su valor no pueda estar repetido, para ello activaremos *Unique*. Por último, si queremos que la columna tenga un valor por defecto lo escribiremos en el recuadro correspondiente (*Por defecto*).

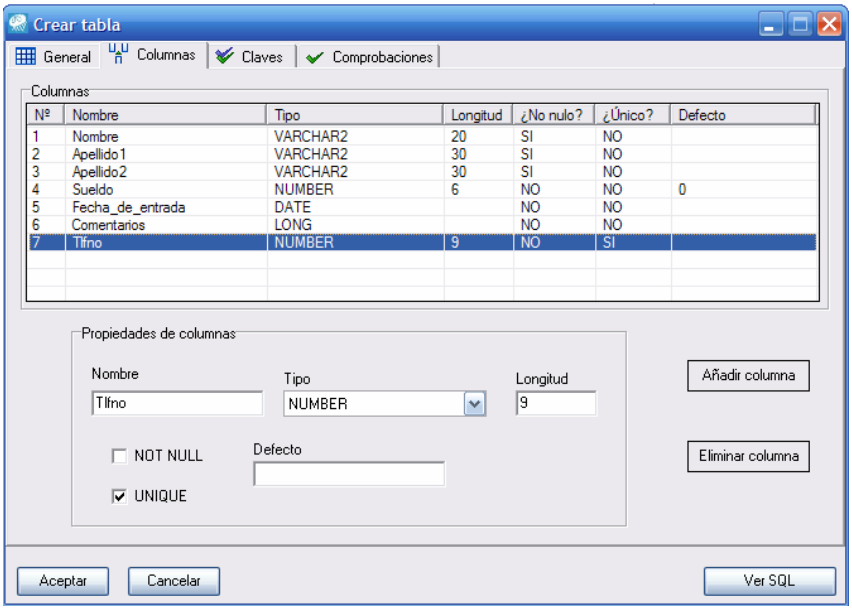


Figura 7.65: Pestaña Columnas del constructor de tablas

Claves:

Tenemos 3 tipos de claves: primarias, foráneas y únicas o candidatas. Aquí vamos a poder crear sobre la tabla cualquiera de ellas. La apariencia del formulario que las controla la podemos ver en la Figura 7.67. En él vamos a definir el nombre, tipo (primaria, foránea o única), las columnas que involucra (una o varias), y en el caso de que queramos definir una clave que hace referencia a otra tabla (foránea) se nos activarán las listas desplegables para seleccionar la tabla a la que haremos referencia y dentro de esta, la columna. El sistema sólo nos va a mostrar las columnas de la tabla que sean clave primaria o única, pues son a las únicas a las que podemos referenciar, y en el caso de que no especifiquemos ninguna columna por defecto se elegirá la clave primaria de la tabla para hacerle referencia. La última casilla que encontramos se refiere a si queremos o no que la clave esté habilitada, en el caso de que no la marquemos la clave se creará, pero desactivada.

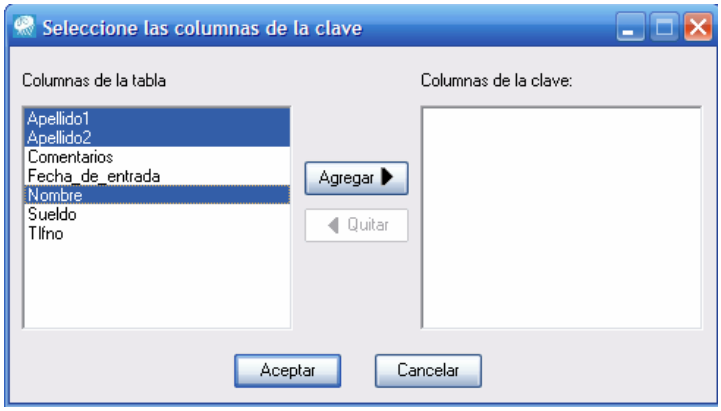


Figura 7.66: Seleccionando las columnas que formarán la clave

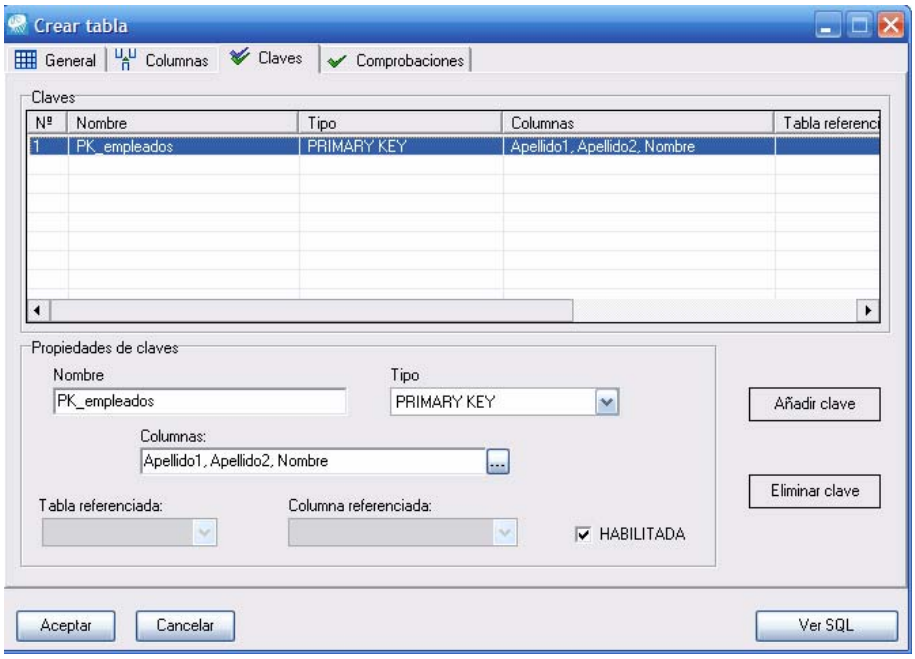


Figura 7.67: Pestaña Claves del constructor de tablas

- **Comprobaciones:**

Una forma fácil y simple de comprobar la consistencia de los datos en algunas ocasiones es definir comprobaciones. Estas comprobaciones se encargan de verificar que se cumple una condición o conjunto de condiciones antes de insertar o actualizar la fila de la tabla. Para definir una comprobación le daremos un nombre y especificaremos la condición que queremos que se cumpla. Al igual que con las claves podemos definirla como activada o desactivada a priori. En la Figura 7.68 podemos ver un ejemplo de creación de comprobaciones.

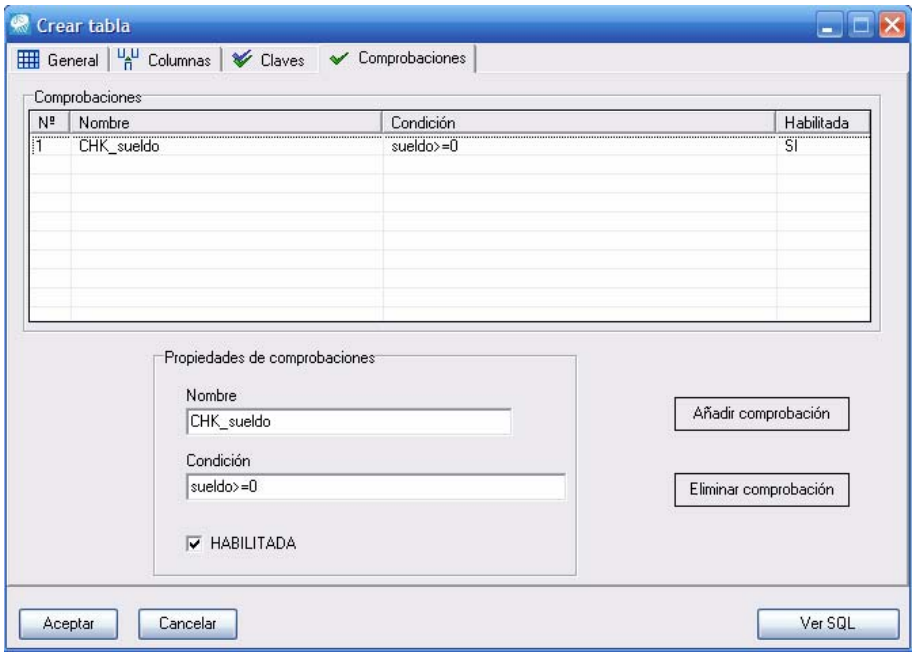


Figura 7.68: Pestaña Comprobaciones del constructor de tablas

El formulario tiene también un completo control de errores. Si hemos dejado algún dato necesario sin rellenar el sistema nos informará del error y de donde se encuentra. Si, por ejemplo, dejamos sin rellenar la longitud de una columna que es de tipo cadena, la aplicación nos informará del error y nos dirá que la columna tiene datos necesarios sin rellenar. También en el caso de que ocurra un error al lanzar la sentencia, ya sea desde el constructor o desde el modo SQL el manejo de errores será similar al que se realiza en la ventana de edición de la sesión activa, esto es, se marcará el lugar exacto de los fallos en la ventana de edición de SQL del constructor y en la pestaña de errores de la sesión activa se indicarán los tipos de fallos que han aparecido.

En la Figura 7.69 podemos ver el resultado de pulsar sobre el botón Ver SQL con los datos que hemos ido introduciendo en el constructor de tablas.

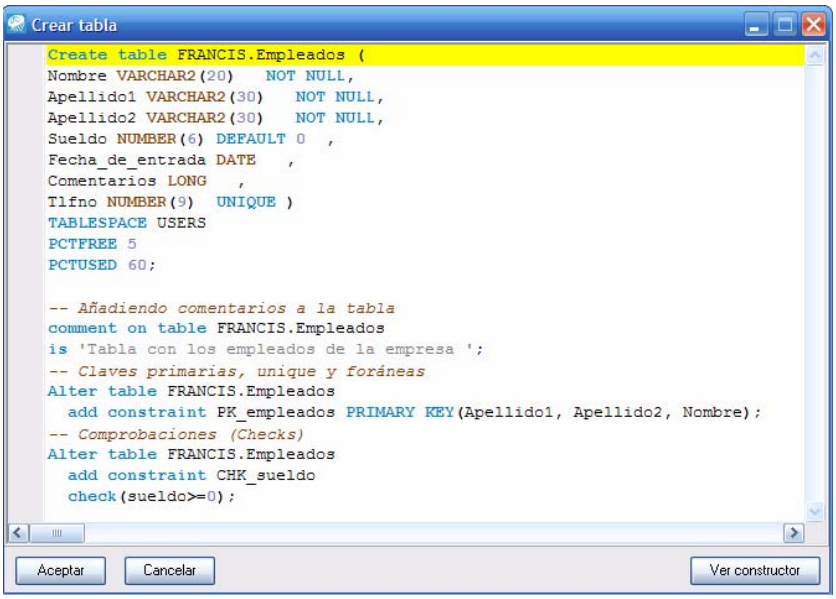


Figura 7.69: Código SQL asociado a los datos del constructor

7.7.6.3.2 Creación de procedimientos y funciones

Podremos acceder a esta opción haciendo clic en el menú *Oracle* y posteriormente en *Construir procedimiento / función*. En la Figura 7.70 se puede ver el aspecto de este formulario. En la Figura 7.71 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 agregaremos los datos referentes a ese parámetro como son su nombre, si es de entrada o de salida y su tipo y pulsaremos el botón *Añadir*. 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*.

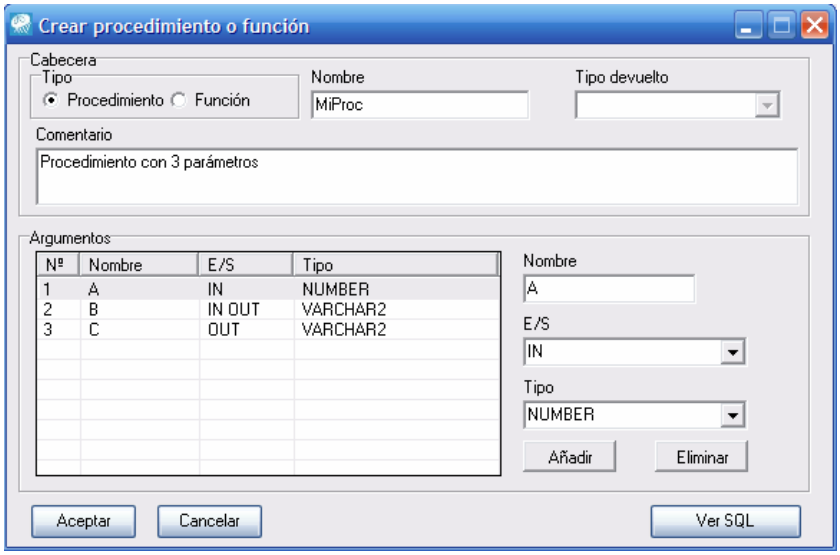


Figura 7.70: Creación visual de procedimientos y funciones

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 *Aceptar*, 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 *Aceptar*, momento en el cual *Medusa 2* 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.

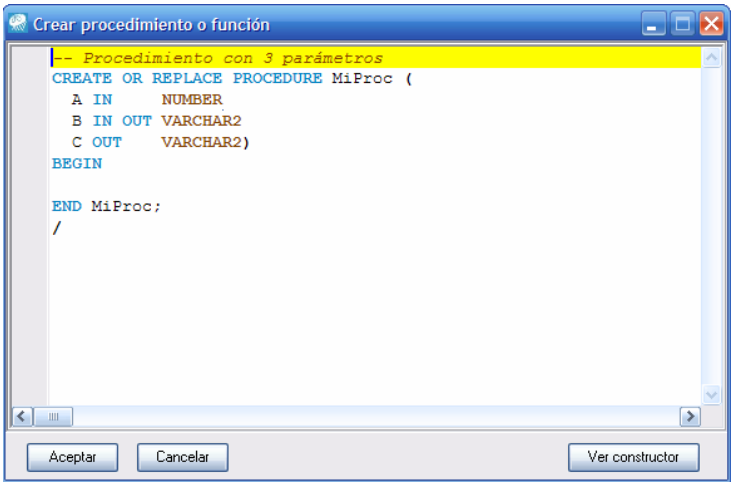


Figura 7.71: Aspecto del formulario tras pulsar sobre Ver SQL

7.7.6.3.3 Creación de disparadores

Medusa 2 también dispone de un formulario para crear de forma visual disparadores (*triggers*). Podremos acceder a esta opción, haciendo clic en el menú *Oracle* y posteriormente en *Construir disparador*.

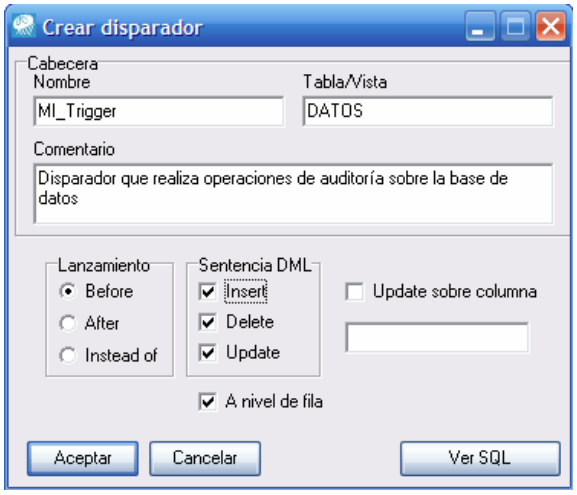


Figura 7.72: Aspecto del formulario de creación de disparadores

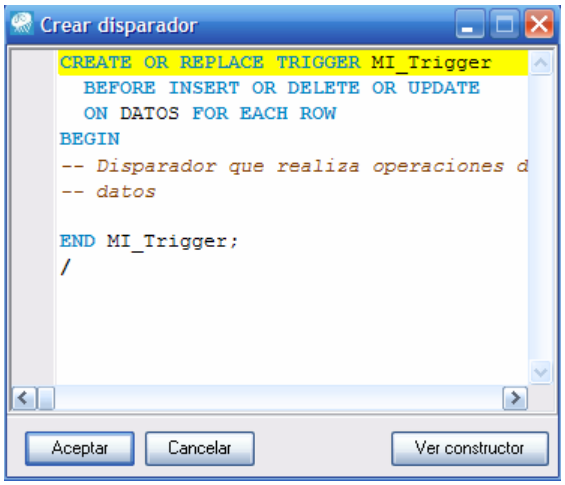


Figura 7.73: Aspecto del formulario tras pulsar sobre Ver SQL

En la Figura 7.72 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 7.73, se muestra el código generado por la aplicación, al que se accede pulsando *Ver SQL*.

Igualmente que en el caso anterior, si el usuario olvidase complementar correctamente los datos, el programa le advertirá que no se puede seguir hasta que no rellene los datos faltantes, situación que no ocurre, por supuesto, si lanzamos la sentencia desde el modo SQL.

7.7.6.3.4 Creación de paquetes

Medusa 2 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*. La definición del paquete, junto con el cuerpo del paquete, se genera en una nueva ventana de edición. Ambas partes estarán separadas por la barra de compilación *"/"* de bloques de código de Oracle. En la Figura 7.74, se puede observar el aspecto visual de este formulario.

El código generado, incluye el usuario del programa, la fecha de creación y el comentario (ver en la Figura 7.75). 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.

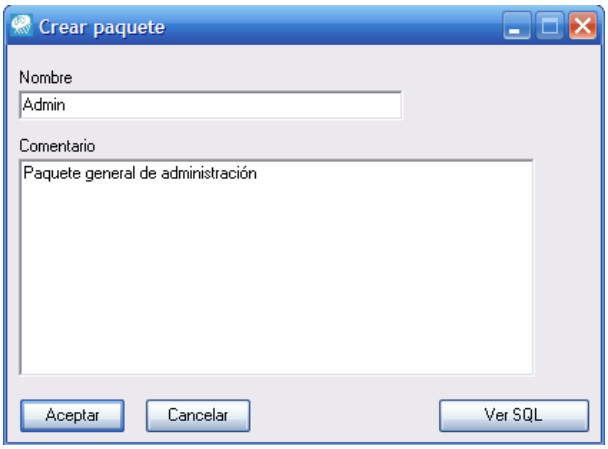


Figura 7.74: Aspecto del formulario de construcción visual de paquetes

```
CREATE OR REPLACE PACKAGE Admin IS

  -- Autor      : F. Javier Pulido Arrebola
  -- Creado     : 17/09/2005 5:15:00
  -- Paquete general de administración

  -- Declaraciones de tipo públicas
  TYPE <NombreTipo> IS <TipoDatos>;
  -- Declaraciones de constantes públicas
  <NombreConstante> CONSTANT <TipoDatos> := <Valor>;
  -- Declaraciones de variables públicas
  <NombreVariable> <TipoDatos>;
  -- Declaraciones de funciones y procedimientos públicas
  FUNCTION <NombreFuncion>(<Parametro> <TipoDatos>) RETURN <TipoDatos>;

END Admin;
/
-- Cuerpo del paquete
CREATE OR REPLACE PACKAGE BODY Admin IS

  -- Autor      : F. Javier Pulido Arrebola
  -- Creado     : 17/09/2005 5:15:00
  -- Paquete general de administración

  -- Declaraciones de tipo privadas
  TYPE <NombreTipo> IS <TipoDatos>;
  -- Declaraciones de constantes privadas
  <NombreConstante> CONSTANT <TipoDatos> := <Valor>;
  -- Declaraciones de variables privadas
  <NombreVariable> <TipoDatos>;

  -- Implementación de funciones y procedimientos
  FUNCTION <NombreFuncion>(<Parametro> <TipoDatos>) RETURN <TipoDatos> IS
    <VariableLocal> <TipoDatos>;
  BEGIN
    <Sentencias>;
    RETURN(<Result>);
  END;
```

Figura 7.75: Código asociado al contenido del constructor visual

7.7.6.3.5 Creación de bloques Java

De forma similar a los paquetes, *Medusa 2* 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 7.76, 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 7.77, *Medusa 2* conmuta automáticamente la opción del realzado sintáctico a modo Java.

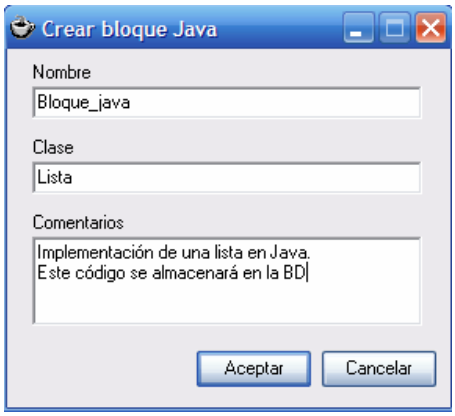


Figura 7.76: Aspecto del formulario de generación de bloques Java

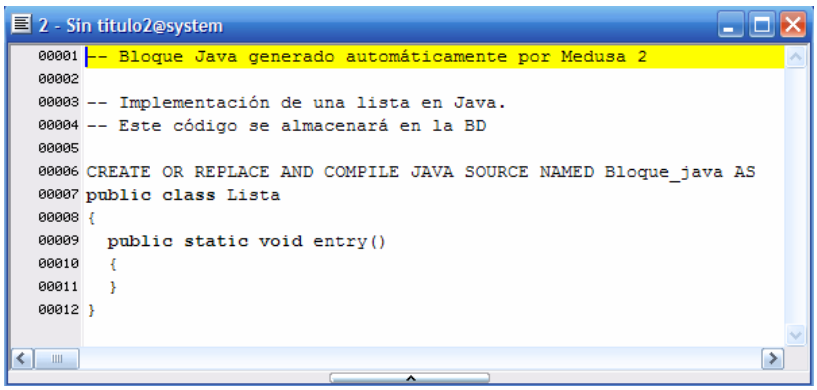


Figura 7.77: Bloque java generado por el constructor

7.7.6.3.6 Creación de secuencias

De forma similar a los casos anteriores, *Medusa 2* 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 clic en el menú *Oracle* y posteriormente haciendo clic en el menú *Construir Secuencia*.

En la Figura 7.78, 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.

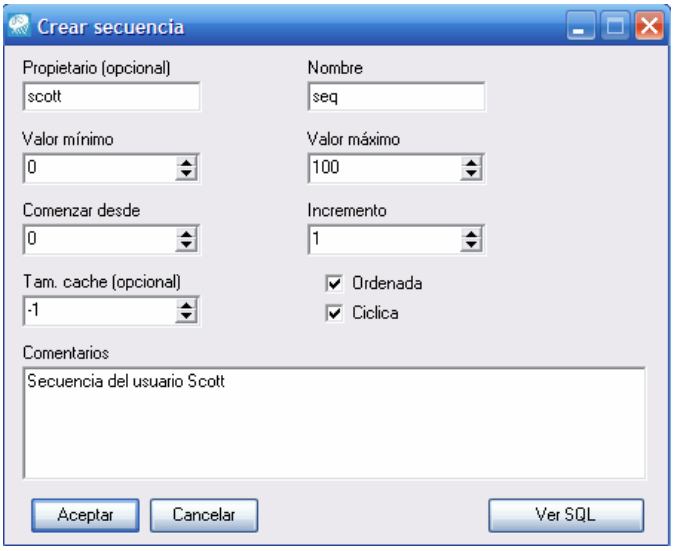


Figura 7.78: Aspecto del formulario de generación de secuencias

En la Figura 7.79, se puede ver el código generado a partir del formulario mostrado en la Figura 7.78, 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.

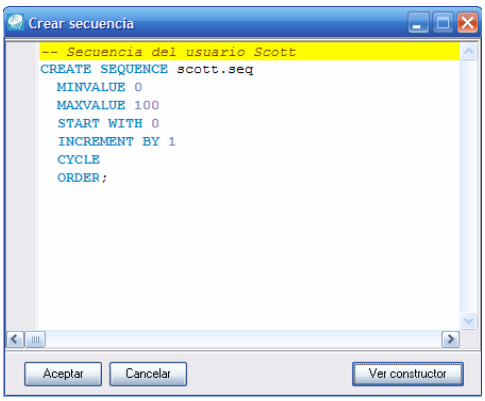


Figura 7.79: Aspecto del formulario tras pulsar sobre Ver SQL

7.7.6.3.7 Construcción de bibliotecas

El formulario de construcción de bibliotecas es realmente simple, puesto que para crear una biblioteca tan sólo hay que dar un nombre que la represente y definir el archivo *.dll* de librería que contendrá las funciones que después utilizaremos. En la Figura 7.80 podemos ver el formulario.

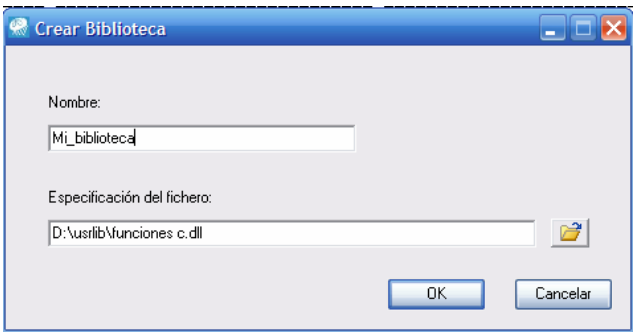


Figura 7.80: Formulario de creación de bibliotecas

7.7.6.3.8 Construcción de sinónimos

Medusa 2 incluye también una opción para la construcción de sinónimos de forma fácil. Se puede tener acceso a esta opción haciendo clic en el menú *Oracle* y posteriormente en *Construir sinónimos*. Tal y como se observa en la Figura 7.81, 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 7.82 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.

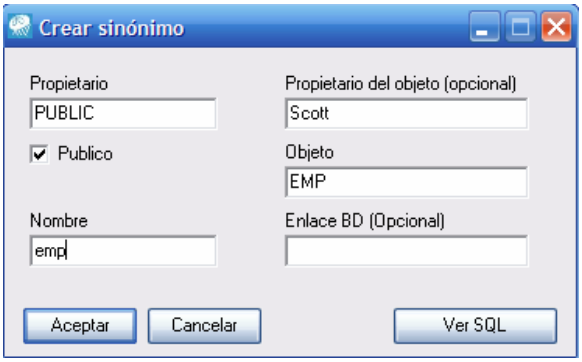


Figura 7.81: Aspecto visual del formulario de generación de sinónimos

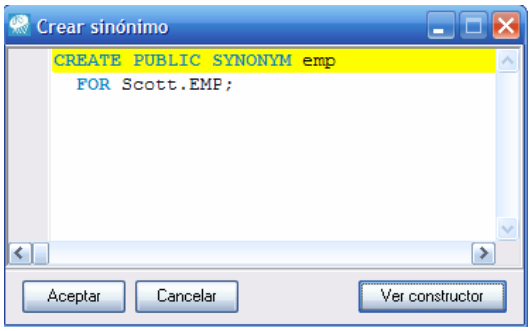


Figura 7.82: SQL asociado al constructor de un sinónimo

7.7.6.3.9 Construcción de trabajos

Para programar un trabajo en Oracle disponemos del constructor visual de trabajos. En la generación de un trabajo debemos de especificar el intervalo en que queremos que se ejecute además de qué queremos que haga el trabajo. En la Figura 7.83 vemos como se crearía un trabajo que ejecutaría mensualmente el procedimiento *Calcula_sueldo()*. Una vez que aceptemos el sistema nos informará que el trabajo ha sido programado y nos mostrará el número que Oracle le ha asignado.

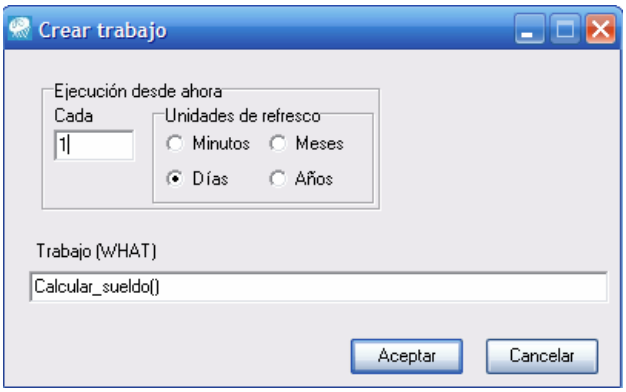


Figura 7.83: Creación de trabajos

7.7.6.3.10 Creación de vistas

De forma similar a los casos anteriores, *Medusa 2* también incorpora un interfaz para facilitar la creación de vistas y vistas materializadas de una forma visual, lo cual puede ser bastante útil. Se puede tener acceso a esta opción haciendo clic en el menú *Oracle* y posteriormente en *Construir vista*.

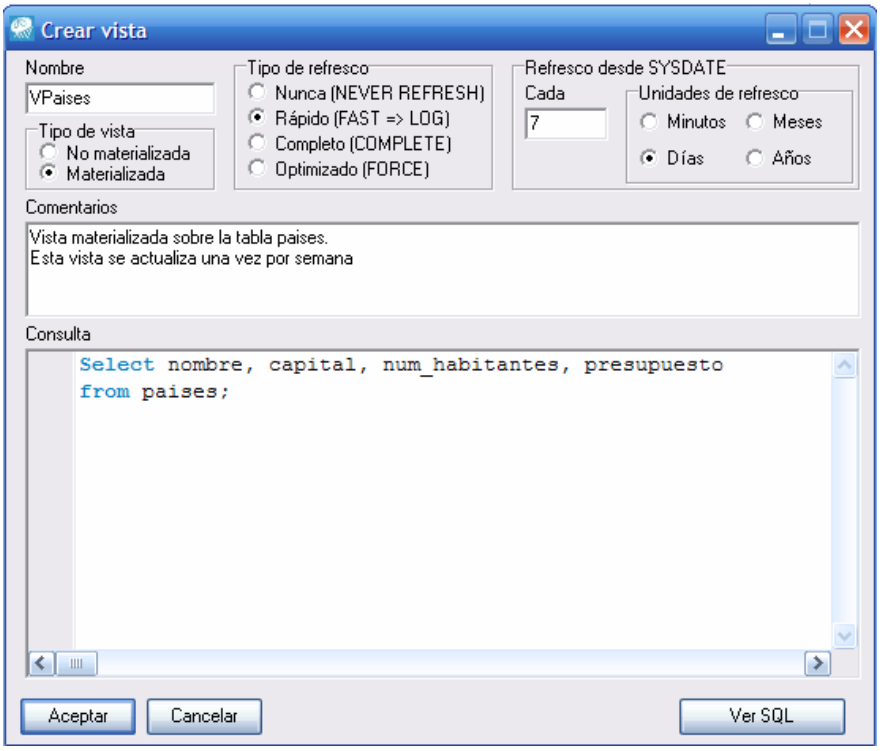


Figura 7.84: Aspecto visual del formulario de creación de vistas

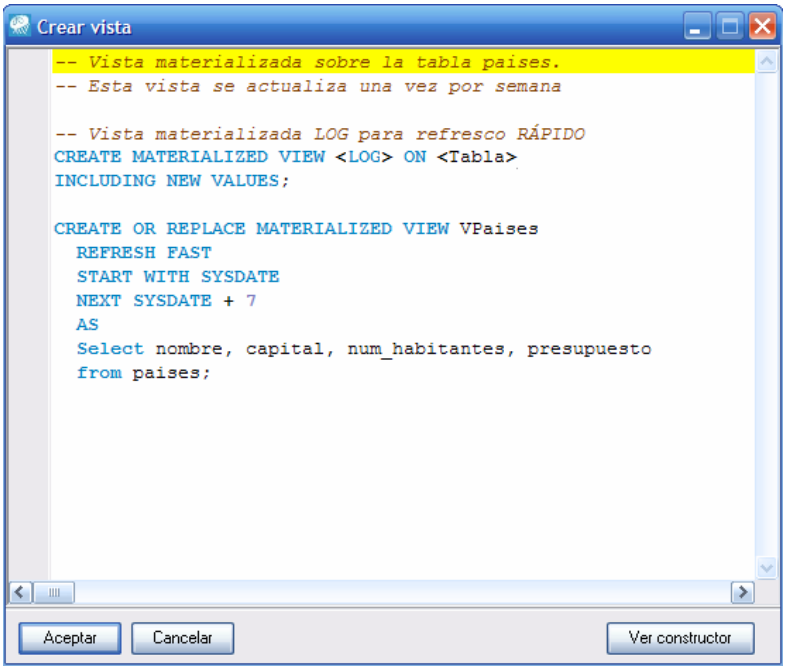


Figura 7.85: Aspecto de la sentencia al pulsar sobre *Ver SQL*

En la Figura 7.84, 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 introducimos 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 7.85, 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 necesaria 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.

7.7.6.3.11 Creación de espacios de tablas

Medusa 2 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 7.86.

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.

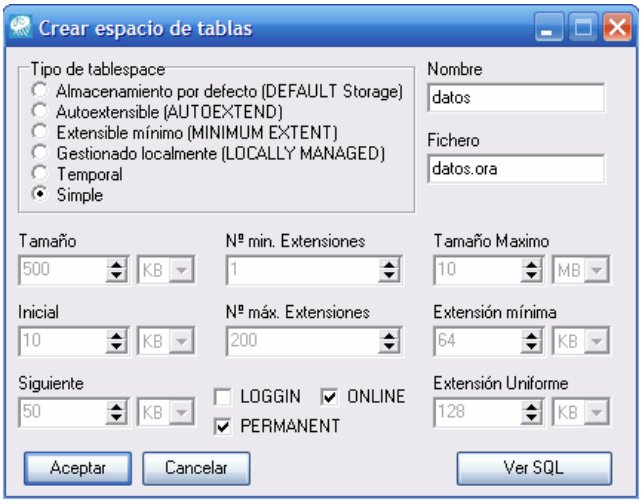


Figura 7.86: Aspecto del formulario de construcción visual de *tablespaces*

La tercera zona, es la que encontramos a partir de la mitad del formulario hasta abajo, 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`.

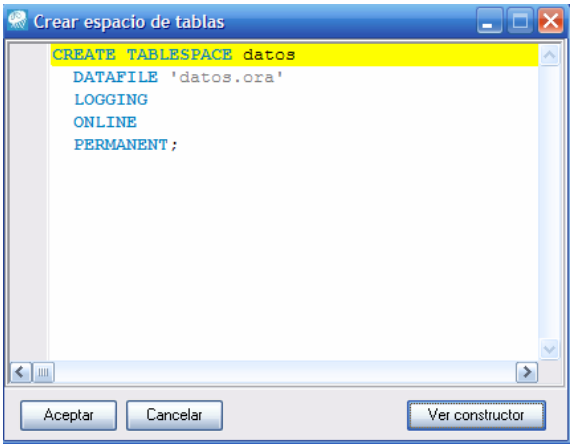


Figura 7.87: Aspecto del formulario tras pulsar sobre Ver SQL

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 7.87 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.

7.7.6.3.12 Creación de perfiles de usuario

Medusa 2 también incluye una opción para la creación de perfiles de usuarios, mediante la sentencia `CREATE PROFILE`. Se puede tener acceso a esta opción haciendo clic en *Oracle | Crear| Perfil*. En la Figura 7.88 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 7.88: 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 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. Se corresponde con la cláusula `PASSWORD_VERIFY_FUNCTION`.

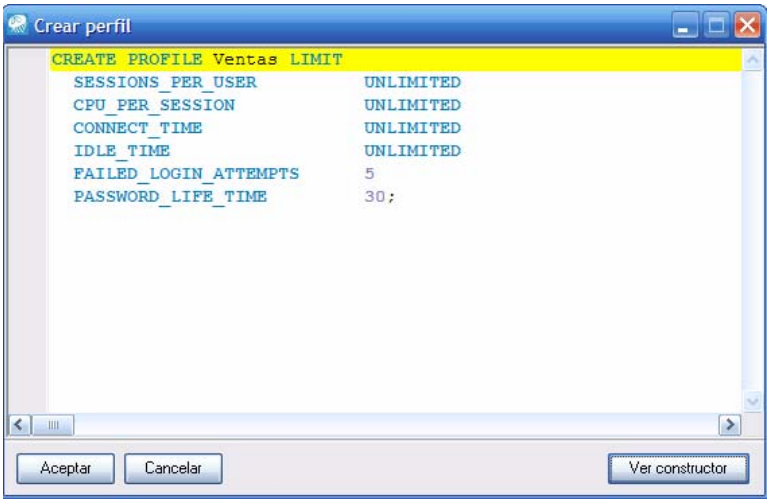


Figura 7.89: Aspecto del formulario tras pulsar sobre Ver SQL

7.7.6.3.13 Creación de usuarios

Otra de las opciones que incluye *Medusa 2* es la construcción visual de usuarios. El aspecto del formulario de creación de usuarios se puede ver en la Figura 7.90. Se puede tener acceso a esta opción en el menú *Oracle* y después, haciendo clic en *Crear usuario*, o mediante el Explorador de objetos.

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 introduciendo el valor en el campo de edición correspondiente (se puede emplear K para Kilobytes y M para Megabytes, tal y como muestra la Figura 7.90. Podremos asignar un perfil al usuario que estamos creando, simplemente escribiendo el nombre del perfil deseado, en la casilla titulada *Perfil*.

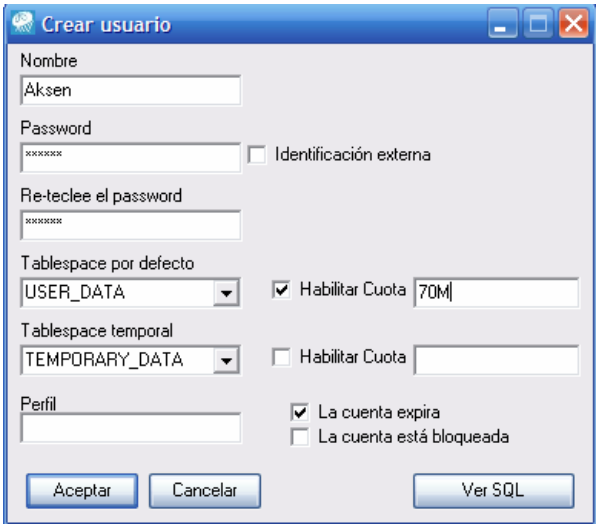


Figura 7.90 Aspecto del formulario de creación de usuarios

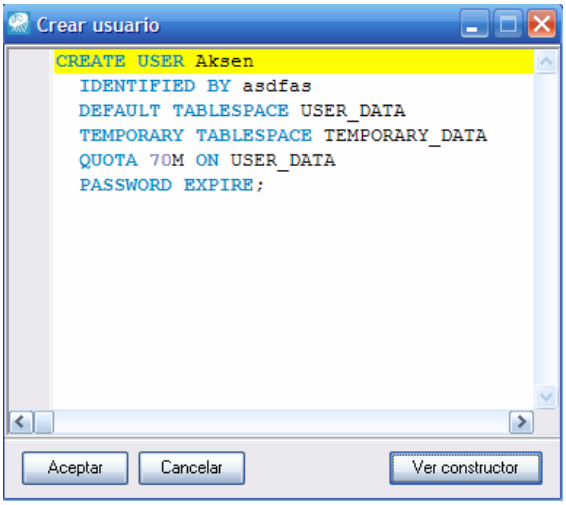


Figura 7.91: Aspecto del formulario tras pulsar sobre Ver SQL

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 7.91 se puede observar el código resultante de esta opción.

7.7.6.3.14 Creación de índices

Otra opción que también se incluye en *Medusa 2*, es la posibilidad de crear índices de forma visual. Tendremos acceso a esta opción en el menú *Oracle* | *Construir índice*. Tal y como se observa en la Figura 7.92, 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 introducimos 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 7.93 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 `ALUMNOS` y sólo afecta al campo `NOMBRE`.

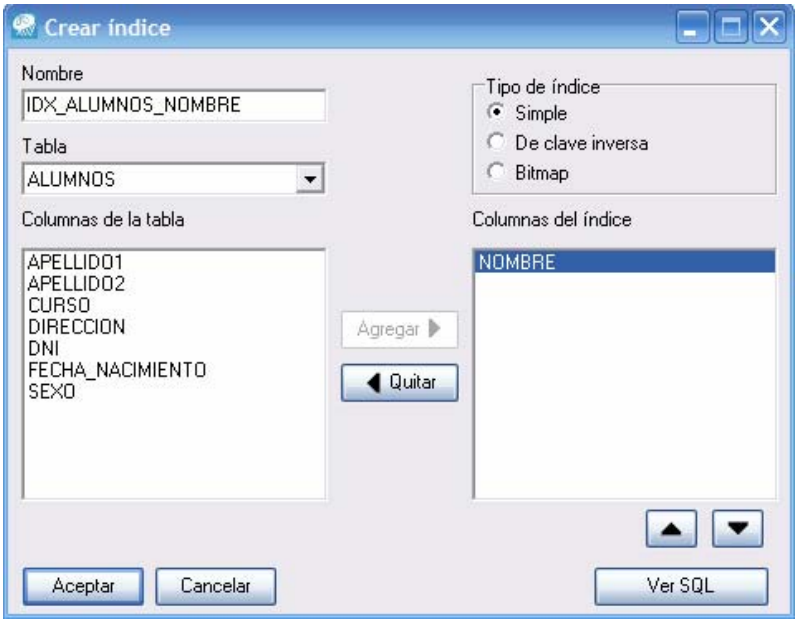


Figura 7.92: Aspecto del formulario de creación de índices

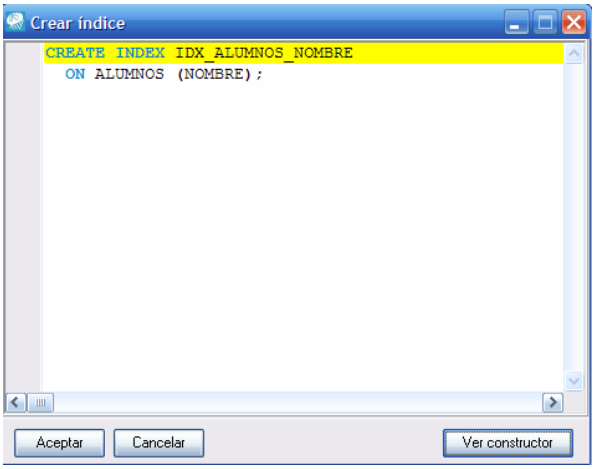


Figura 7.93: Aspecto del formulario tras pulsar sobre Ver SQL

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

7.7.6.4 Reconstrucción de la sentencia CREATE TABLE

Una opción del Menú *Oracle* 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*. 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 7.94, se muestra el aspecto del formulario.

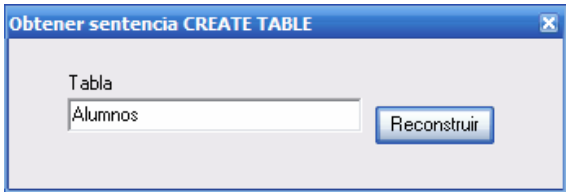


Figura 7.94: Aspecto del formulario de reconstrucción de tablas

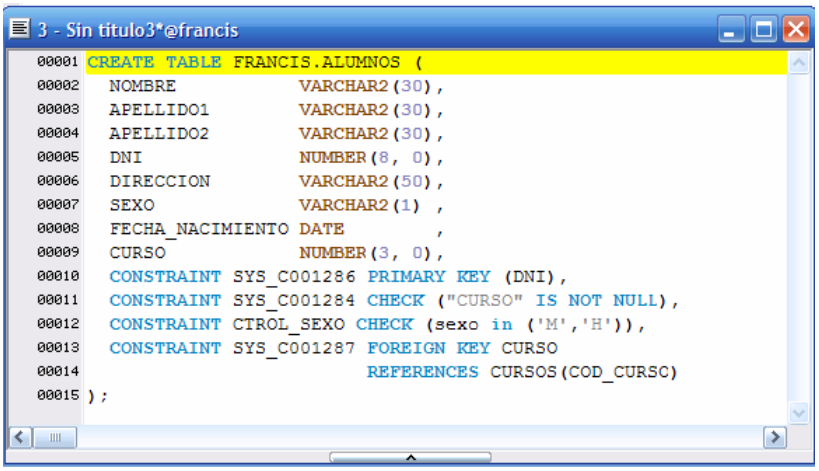


Figura 7.95: Resultado de la reconstrucción de la sentencia CREATE TABLE

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 y ALL_CONSTRAINTS entre otras.

Tal y como se observa en la Figura 7.95 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).

7.7.6.5 Control de transacciones

Una importante característica básica en cualquier SGBD y también en *Medusa 2* es el control de transacciones. Básicamente, el control de transacciones en Oracle se realiza con 3 comandos: *commit, rollback y savepoint.* Si pulsamos sobre el Menu `Oracle | Control de transacciones` podemos ver los 3 comandos mencionados. El uso de los 3 es bastante intuitivo, pincharemos `Commit` cuando queramos confirmar la transacción actual, `Rollback` si queremos deshacer la transacción y `Savepoint` si queremos establecer un punto de guarda. Al hacer clic sobre `Savepoint` nos aparecerá un diálogo para que introduzcamos el nombre del punto de guarda. Una vez definido tendremos la posibilidad de deshacer la transacción hasta ese punto de guarda, o hasta cualquier otro que hayamos definido, pulsando sobre `Rollback` y extendiendo el menú como aparece en la Figura 7.88. Para más información sobre las transacciones mirar la Sección 2.5 o [ORA02].

Cuando hayamos deshecho la transacción veremos que habrán desaparecido del menú el *savepoint* seleccionado y todos los *savepoints* definidos con posterioridad a el. También podremos acceder a las mismas opciones desde la barra de herramientas, pues están visibles por defecto los botones de *commit, rollback y savepoint*. Por último, es importante reseñar que cada sesión va a tener sus propios puntos de guarda, de manera que si definimos unos puntos de guarda estos sólo serán accesibles desde la sesión que los creo, característica evidente por otro lado.

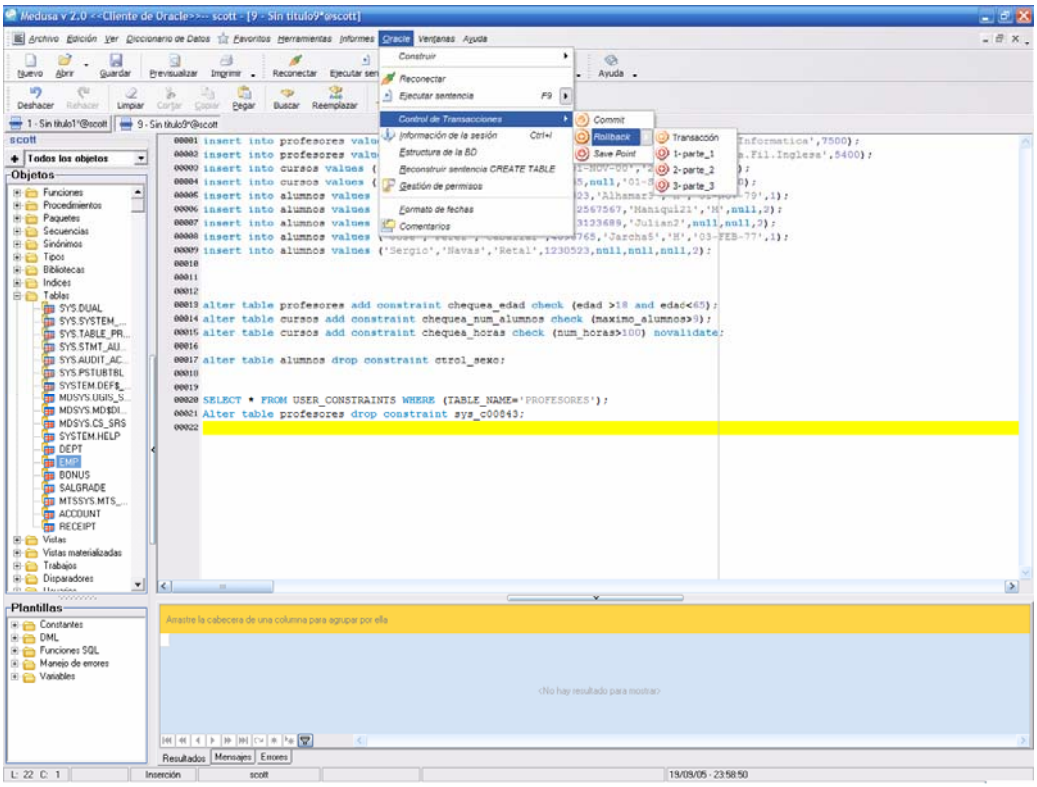


Figura 7.96: Utilizando los puntos de guarda sobre la transacción actual

7.7.6.6 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`. En la Figura 7.97 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 7.97 con detenimiento, se puede ver cómo 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 2* 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.

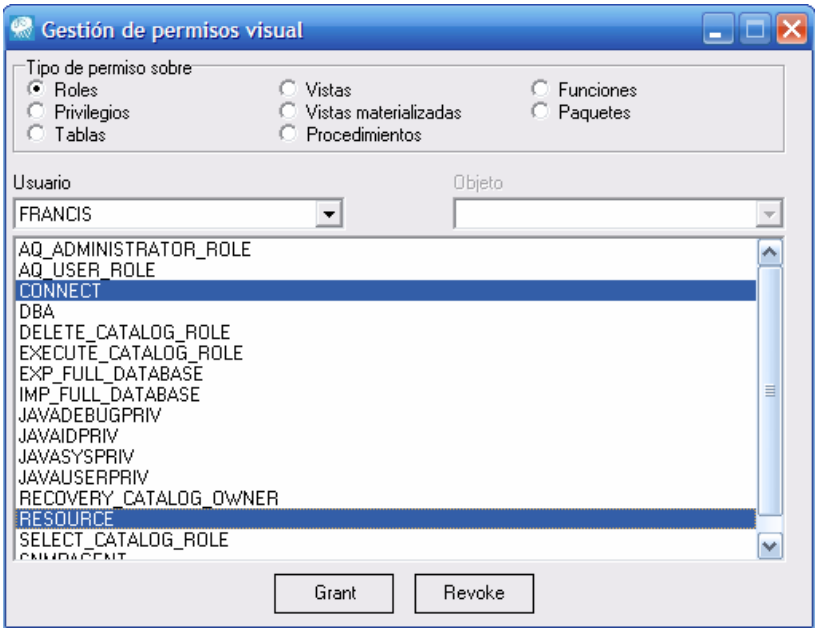


Figura 7.97: Aspecto visual del formulario de gestión de permisos

7.7.6.7 Formato de fechas

Medusa 2 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*.

El formulario de manipulación de formatos de fechas se muestra en la Figura 7.98, 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 clic sobre cualquier elemento de la tabla y éste se añadirá justo donde se encuentre el cursor en la parte de la expresión en ese momento.

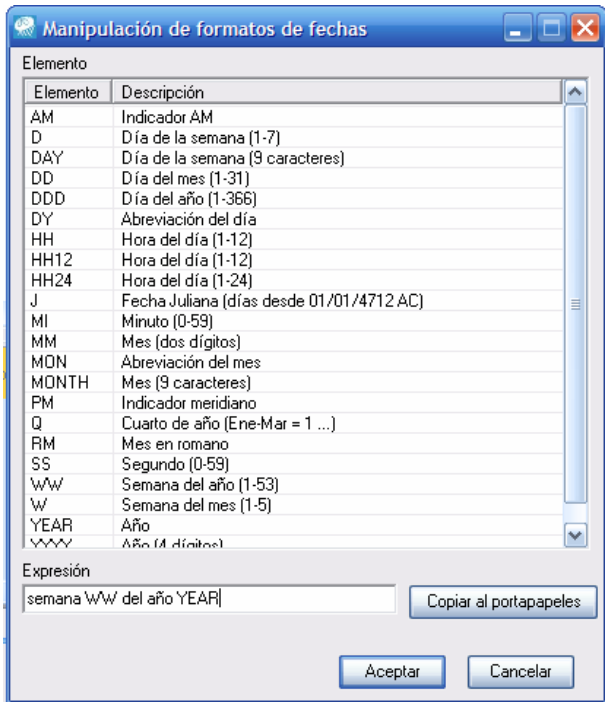


Figura 7.98: Aspecto del formulario de manipulación de fechas

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

Hemos incluido en *Medusa 2*, una opción para la tan olvidada manipulación de comentarios en el diccionario de datos. Podremos acceder a esta opción mediante el menú *Oracle* | *Comentarios*. Se trata de un formulario donde se muestran los comentarios de las tablas y las columnas. En la Figura 7.99 se muestra el aspecto del formulario de manipulación de comentarios. Se observa como se ha seleccionado la tabla PROFESORES, 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>`.

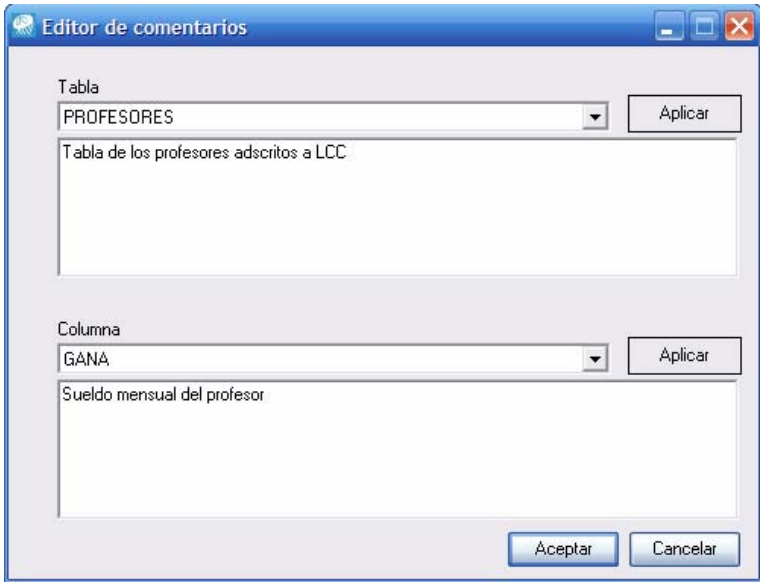


Figura 7.99: Aspecto del formulario de manipulación de comentarios

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.

7.7.7 Menú Ayuda

En esta sección veremos las opciones de este menú. Tal y como se detalló en la Sección 7.2.10, este menú está compuesto por la ayuda de la aplicación, la ayuda de Oracle y el formulario *Acerca de*.

7.7.7.1 Ayuda de *Medusa 2*

Desde el entorno de *Medusa 2* 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 7.100.

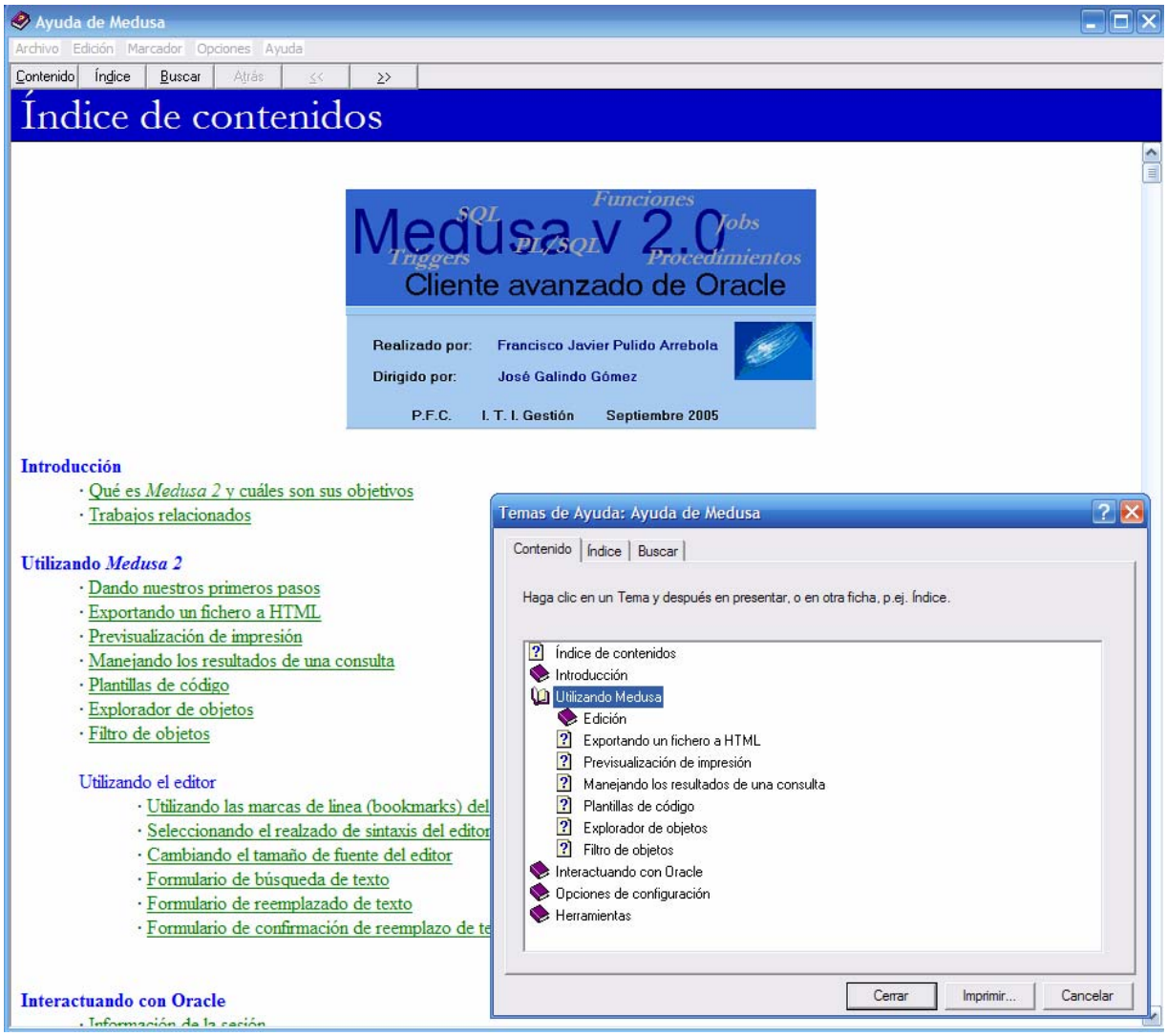


Figura 7.100: Aspecto de la ayuda en línea de *Medusa 2*

7.7.7.2 Documentación y ayuda de Oracle integrada

Medusa 2 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 clic 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.

7.7.7.3 Formulario *Acerca de*

Se trata del formulario típico que solemos encontrar en casi todas las aplicaciones, que nos informa, en nuestro caso, de la versión de la aplicación y las direcciones de correo electrónico. El aspecto de este formulario, se puede observar en la Figura 7.101.

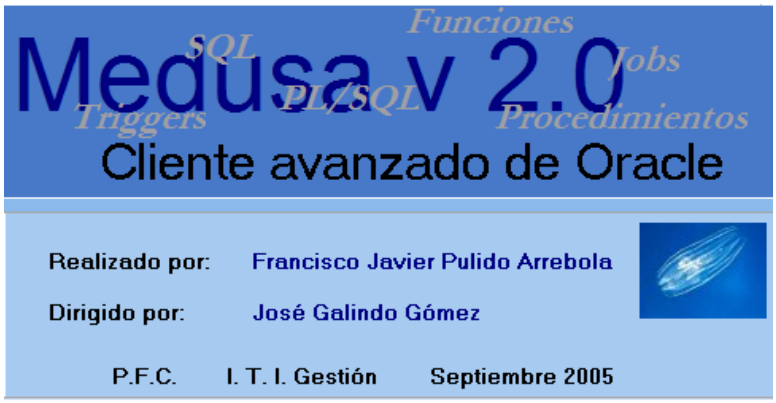


Figura 7.101: Aspecto del formulario *Acerca de*

Para cerrar el formulario nos bastará con pulsar cualquier tecla o pinchar en cualquier parte del formulario.

Conclusiones y líneas futuras

Al final del camino, hacemos un alto, miramos atrás y vemos lo lejos que queda el comienzo. Todo el tramo recorrido nos ha curtido en mil batallas, nos ha llenado de experiencia y nos ayudará en la siguiente aventura.

Esta es la segunda versión de Medusa y espero que no sea la última…

Conclusiones

La realización de este PFC ha llevado consigo un periodo que ha ido pasado por etapas y en el que se han ido mezclando las distintas partes de que consta la elaboración de un PFC. La experiencia ha resultado muy positiva, no sólo por los conocimientos adquiridos, sino más bien por la forma de adquirirlos. Y es que, si por algo se caracteriza un PFC es por la construcción de un trabajo propio, hecho con la originalidad del alumno y la búsqueda por su parte de la ayuda necesaria.

Para hacer de *Medusa 2* la aplicación que es ahora hemos tenido que ir avanzando en muchos sentidos, de cada cual hemos obtenido un conocimiento teórico, a la vez que la aplicación de esos conocimientos al desarrollo de Medusa nos enriquecía en el uso práctico. Algunas de estas fases han sido las siguientes:

Análisis de una aplicación: El punto de partida fue un análisis de la primera versión de *Medusa*. Se hizo especial hincapié en comprender el funcionamiento del programa, ver las características que poseía y las limitaciones que podrían ser arregladas. En cuanto al código, la comprensión resultó una tarea difícil al principio, pues al intrincado código que lo elaboraba, había que sumar la utilización de componentes de programación que eran completamente desconocidos a priori. Y no se hizo demasiado largo gracias a que ya conocíamos con anterioridad el lenguaje Delphi y teníamos algunos conocimientos sobre Oracle. Después de esto se consiguió dejar un código mucho más claro, organizado y documentado.

Estudio de las ampliaciones, mejoras y modificaciones: Puesto que no estábamos ante una aplicación comercial y no debíamos de responder a unos requerimientos específicos se trató en todo momento de que el programa fuera fácil de usar, y didáctico en algunos puntos. La facilidad en el uso no se refiere a un número limitado de características, sino a que cualquier usuario sin conocimientos previos de la aplicación pudiera utilizar *Medusa 2* para cubrir sus necesidades. Los requerimientos que se han cubierto han sido los que yo, como usuario de un cliente de Oracle, he creído que resultarían más útiles. También se han incluido en el desarrollo muchas ideas interesantes que José Galindo, director del PFC, me comentó, así como algunas otras que amigos míos, que

utilizan clientes de Oracle con normalidad (SQL-Plus, SQL-Plus WorkSheet), me comentaron que beneficiarían al resultado final.

Utilización y búsqueda de información: Una de las fases más importantes en este PFC ha sido la utilización de los archivos de ayuda. *Medusa 2* utiliza bastantes componentes de programación y ha hecho falta un intenso conocimiento de cada componente para poder integrarlo y utilizarlo en el desarrollo. Para conseguir el resultado expuesto he necesitado aprender a manejar y buscar en la extensa documentación de Oracle, Delphi, SynEdit, FlyTreeViewPro, DOA y Developer Express. En algunos casos ha sido necesario incluso la modificación del código fuente de los componentes para el correcto acoplamiento con el resto del programa. Internet también ha sido una herramienta muy utilizada al principio en la búsqueda de componentes de programación adecuados, y después para cuestiones específicas y dudas concretas.

Redacción de la memoria: A medida que el desarrollo iba creciendo también lo hacía la memoria que describía el proceso que se seguía. Resulta tan importante la realización de una aplicación como la elaboración de una memoria, en la que se detalle el proceso que se ha ido siguiendo y que podrá servir como partida para que otras personas puedan ampliar el trabajo en un futuro. El manual de usuario es la parte más importante de esta memoria. Describe el funcionamiento de cada parte del programa y valdrá como base para los usuarios que decidan hacer de *Medusa 2* su cliente de Oracle, aunque la ayuda de la aplicación incluye la misma información y podrá consultarse en el archivo *Medusa2.hlp*.

Acerca de Medusa 2

Los objetivos marcados en el inicio eran hacer *Medusa* más completo, más versátil, pero ante todo más útil. Los beneficios aportados contribuirían a que las tareas cotidianas, como consultar o modificar los datos de una tabla, se realizaran no sólo más rápido, sino también mucho más fácilmente.

Con la inclusión del árbol de objetos hemos conseguido una visión global y particular de los objetos de la base de datos. Podemos realizar las tareas típicas como crear, eliminar, modificar o ver las propiedades de los objetos, de una forma gráfica.

Los constructores visuales de objetos también nos permitirán ver el código SQL asociado, así como alterarlo, que servirá a unos usuarios para dar mayor complejidad y a otros para aprender más acerca de Oracle y SQL.

Las opciones añadidas a la rejilla de resultados, tales como filtrar, nos permitirán una gran potencia en la visualización de aquellos datos que nos interesen, y con las consultas actualizables podremos borrar, insertar o modificar de una forma ágil, rápida y sencilla.

La generación de informes y la exportación de resultados nos van a permitir compartir nuestro trabajo. Aunque es muy beneficioso tener los resultados en formatos actuales como XML o formatos tan extendidos como TXT, HTML o Excel, puede ser que necesitemos mostrar los resultados en un soporte que no sea digital. Los informes nos permitirán sacar una infinidad de listados diferentes, sobre objetos de la base de datos, sobre características propias de la BD, sobre los datos que están contenidos…

Después de este pequeño análisis de las características principales que hemos implementado, se han cumplido los objetivos prefijados, pues *Medusa* ha pasado de ser un editor completo a un verdadero cliente de Oracle, con funciones realmente útiles para el uso y la administración de una base de datos.

Tendencias futuras

Desde siempre, ha sido muy importante el almacenamiento de datos, su recuperación, mantenimiento y relaciones. Oracle, ha sido pionero en este campo desde hace ya muchos años. Dada la importancia de los datos y de tener un sistema completo e integrado, que interactúe con éstos, *Medusa 2* puede crecer en numerosos aspectos, entre los cuales se proponen las siguientes ideas.

1. Portabilidad de *Medusa 2* a otros sistemas operativos. Sería muy útil portar *Medusa 2* a otros sistemas operativos como *Linux* por ejemplo, para cubrir un mayor abanico de necesidades, aprovechando que existe una versión de Oracle para *Linux*.

2. Implementar un entorno completo de modelado Entidad / Relación Extendido (EER). 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. Para este objetivo puede resultar interesante el PFC de José David Quero [EER04] aunque el lenguaje utilizado no sea Delphi.
3. Implementar un interfaz modular. 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.
4. 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 rutina, 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.
5. Incluir posibilidades de grabación de macros. Incluir opciones para grabar, reproducir y gestionar macros, que puedan agilizar el trabajo del usuario.
6. Integración de un depurador PL/SQL y Java. Esta opción podría ser muy útil para usuarios programadores.
7. Posibilidades de exportación e importación de objetos. Poder importar objetos (procedimientos, funciones, tablas...) de otros usuarios o exportar objetos para poder compartirlos con otros usuarios.
8. Incluir planificador de proyectos gráfico. Sería interesante incluir un planificador gráfico para los usuarios avanzados de Oracle. Así como un completo menú para realizar todas las tareas pertinentes al uso de proyectos.
9. Incluir opciones de copias de seguridad. 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 de éstas.
10. Modificar el uso de plantillas (ver Apartado 7.5) para que los parámetros necesarios fueran introducidos a través de un formulario gráfico.
11. Añadir opciones de configuración de la sesión, como el nivel de aislamiento, realizar un *rollback* al desconectar o la posibilidad de realizar un *commit* automático después de cada sentencia de DML, ello haría a la aplicación aún más configurable.

12. Realizar una opción dentro del objeto tabla para insertar un número determinado de filas aleatorias, sin tener en cuenta el contenido pero que cumplieran todas las restricciones de integridad. El código de Medusa 2 contiene esta función pero no ha sido incluida en la aplicación final al no funcionar adecuadamente para cualquier situación de la tabla (claves foráneas, restricciones, comprobaciones…).
13. Dar mayores posibilidades de configuración al filtro personalizado del árbol de objetos y poder definir varios filtros.
14. Incluir un constructor visual de consultas. Sería de gran ayuda no sólo para usuarios inexpertos, sino también para realizar complejas consultas por usuarios avanzados. Para esta función puede resultar interesante la consulta del PFC de Rafael F. Oliva [FDB03].
15. Mayor capacidad de configuración del árbol de objetos, con la posibilidad de definir nuevas carpetas y especificar su contenido.
16. Incluir una opción para buscar objetos dentro del árbol, y poder devolver el resultado en una carpeta definida por el usuario.
17. Utilizar el *External translation manager* para traducir *Medusa 2* a una mayor cantidad de idiomas, tales como alemán, francés, italiano o portugués.
18. Realizar un segundo nivel en el árbol de objetos, de manera que tuviéramos algunas carpetas comunes para todos los objetos en las que pudiéramos ver a quién referencia el objeto y por quién es referenciado. Dependiendo del tipo de objeto también tendríamos posibilidad de ver e interactuar con sus componentes. Un ejemplo serían las columnas, restricciones, claves, índices, disparadores… en el objeto tabla, o procedimientos, funciones, tipos, variables, constantes… en paquetes.
19. En el segundo nivel del explorador de objetos, descrito anteriormente, poder configurar la agrupación de los objetos por su propietario.
20. Mayores opciones de configuración visual del entorno, como los colores de los distintas rejillas de resultados, el tipo y tamaño de letra del editor …
21. Realizar un botón (*dropdown*) de acceso al buffer circular de ejecuciones, de manera que podamos acceder fácilmente a los comandos ejecutados anteriormente.
22. Visualización de estadísticas (accesos a memoria, ordenación de filas, CPU utilizada) al realizar una consulta.
23. Cuando modificamos los datos de una tabla con el formulario visual (ver Figura 7.33), sería una buena idea colorear de distinto color aquellos datos que han sido modificados. Así, no tendríamos porque recordar qué datos hemos modificado ya que los tendríamos

resaltados. Además, sería bueno dar la posibilidad de deshacer o cancelar los cambios antes de confirmarlos.

24. El árbol de objetos posee una característica que nos permite ver las propiedades de cualquier objeto. Además de esto, sería muy beneficioso incluir un formulario visual donde pudiésemos modificar las propiedades del objeto. Desde luego, el formulario debería tener en cuenta que cada objeto tienes sus propiedades características.
25. Sería de gran utilidad que las opciones que podemos realizar sobra las tablas en el Explorador de Objetos (descripción de las columnas, consulta y modificación de datos) las pudiésemos también disponibles sobre las vistas.
26. Dentro del menú herramientas (Sección 7.7.4) podríamos incluir una opción para compilar los objetos no válidos. Al pulsar sobre la opción se nos mostraría un formulario listando todos los objetos que necesitamos recompilar, y con un simple clic los podríamos compilar todos.
27. Hacer de *Medusa* un cliente de FSQL (Fuzzy SQL) una extensión de SQL que permite expresiones difusas, imprecisas o flexibles, ver **[**FDB06**]** y [W3FQL] para mayor información. Un cliente visual lo tenemos en [FDB03]. En la misma línea, también se puede incluir la herramienta de modelado difuso, Fuzzy EER.
28. Realizar una opción importar, para pasar un fichero a una tabla compatible existente, o una nueva tabla que se cree automáticamente a partir de los tipos de los datos del fichero.
29. Incluir más formatos de exportación, como .tex de Latex, y dar la opción de exportar o no la sentencia.
30. En el formulario de creación visual de tablas, conseguir que se puedan crear claves foráneas que involucren a más de una columna, pues actualmente sólo se pueden crear claves que referencian a una única columna.
31. Dar a *Medusa* la posibilidad de realizar auditorias de la base de datos. De esta forma podríamos controlar los accesos y usos de los objetos de la base de datos por parte de los usuarios.

Referencias

En la realización de un proyecto de investigación o desarrollo será necesaria la consulta de trabajos de otras personas. Estos trabajos nos pueden ayudar a intentar mejorar o innovar nuestro proyecto, y en algunos casos, serán un punto de partida válido desde el que llegar a más altas metas.

Referencias bibliográficas

En esta sección, se incluyen referencias bibliográficas consultadas a lo largo de la realización del proyecto, para obtener ideas, o a modo de consulta para obtener información en general. Las dividiremos en grupos por su temática.

Oracle y bases de datos

[ABB00**]** Oracle 8i: Guía de aprendizaje.
Michael Abbey, Michael J. Corey.
Oracle Press, 2000.

[SIL02**]** Fundamentos de bases de datos (4ª edición)
Abraham Silberschatz, Henry F. Korth, S. Sudarshan.
Editorial McGraw-Hill, 2002.

[LON00**]** Oracle 8i: Manual del administrador.
Kevin Loney.
Oracle Press, 2000.

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

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

[FDB06**]** Fuzzy Databases: Modeling, Design and Implementation
Galindo J., Urrutia A., Piattini M.
To publish by Idea Group Publishing Hershey,
USA, 2006.

Delphi

[CAN03**]** La Biblia de Delphi 7.
Marco Cantú.
Anaya Multimedia, 2003.

[CHA03**]** Programación con Delphi 7 y Kylix 3.
Francisco Charte Ojeda.
Anaya Multimedia, 2003.

[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.

[MAR97**]** La cara oculta de Delphi 4.
Ian Marteens.
Intuitive Sight online, 1997.

[DDG02**]** Delphi 7 Developer´s Guide.
Borland Software Corporation, 2002.

Proyectos fin de carrera

[MED03**]** Medusa, un cliente avanzado de Oracle.
Albert Philippe De La Fuente Vigliotti.
I.T.I. Gestión, Universidad de Málaga, 2003.

[SIB01**]** Desarrollo orientado a objetos de un S.I. para la gestión de una tienda de bricolaje.
José Antonio Guerrero Colón
I.T.I. Gestión, Universidad de Málaga, 2001.

[EER04**]** Aplicación Web para el Diseño de Bases de Datos usando Esquemas EER y Traducción Automática a SQL.
José David Quero Sánchez
I.T.I. Sistemas, Universidad de Málaga, 2004.

[FDB03**]** Visual FSQL: Gestión Visual de Bases de Datos Difusas en ORACLE a través de Internet usando FSQL.
Rafael Francisco Oliva Moreno
I.S.I, Universidad de Málaga, 2003.

InstallShield Express – Edición limitada para Delphi

[INS04**]** Manual de usuario de InstallShield X Express
InstallShield, 2004.

Shalom Help Maker

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

External Translation Manager

[ETM02] Manual de usuario de External Translation Manager version 7.0.
Borland Software Corporation, 2002.

Recursos de Internet

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.

Oracle y bases de datos

[W3DOC] Documentación en línea Oracle 8i (versión 8.1.7)
http://www.tahiti.oracle.com/pls/tahiti/tahiti.homepage

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

[W3ADG] Guía del desarrollador de aplicaciones de Oracle 8i
http://download-west.oracle.com/docs/cd/A87860_01/doc/appdev.817/a76939/toc.htm

[W3COD] Leyes de Codd sobre los Sistemas Gestores de Bases de Datos Relacionales
http://isp.webopedia.com/TERM/C/Codds_Rules.html

[W3SAG**]** Guía del administrador de servidores Oracle (versión 8)
http://www-rohan.sdsu.edu/doc/oracle/server803/A54641_01/toc.htm

[W3FQL] FSQL (Fuzzy SQL) A Fuzzy Query Language:
http://www.lcc.uma.es/~ppgg/FSQL/

Programas relacionados

[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/

[W3CLU**]** Club Delphi, el punto de encuentro de los programadores en Delphi.
http://www.clubdelphi.com/

[W3SOU**]** SourceForge.net, el mayor sitio Web de desarrollo y descarga de código abierto y aplicaciones de software libre.
http://sourceforge.net/

[W3ABC**]** Página Web interesante con multitud de herramientas, ejemplos y recursos.
http://www.delphiabc.com/

[W3HAN**]** Página Web para descargar documentación y muchos más recursos de Delphi.
http://www.handyarchive.com/

Componentes de Delphi

[W3DEV**]** Sitio Web de los componentes Developer Express.
http://www.devexpress.com/

[W3IDE**]** Información sobre los distintos productos de Developer Express.
http://www.devexpress.com/Products/Locator/ByIDE.xml?ideid=delphi7

[W3ACU**]** Sitio Web en el que podremos obtener Accutime.
www.programmers.net/mirrors/DSP/newl/midxd40f.htm

[W3FLY**]** Sitio Web de los componentes FlyTreeViewPro.
http://www.9rays.net/

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

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

Medusa

[W3MED**]** La primera versión de *Medusa* y esta se pueden descargar desde:
http://www.lcc.uma.es/~ppgg/PFC

Shalom Help Maker

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