TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Diseño y desarrollo del servidor GESPROT (GEStión de laboratorios de PRÓTesis dentales) y su base de datos.

AUTOR: Mª Pilar Pérez Quílez
DIRECTOR: Cristina López Bravo

01 / 2005

Autor	Mª Pilar Pérez Quílez
E-mail del Autor	pilarpq@mixmail.com
Director(es)	Cristina López Bravo
E-mail del Director	Cristina.lopez@upct.es
Codirector(es)	
Título del PFC	Diseño y desarrollo del servidor GESPROT (GEStión de laboratorios de PRÓTesis dentales) y su base de datos.
Descriptores	

Resúmen

Desarrollo de una aplicación que permita conocer y gestionar a través de Internet los datos necesarios para la gestión y administración de un laboratorio de Prótesis Dentales (pedidos, facturas, inventario, etc)

Se dispone de un sistema informático con las siguientes características:

- Sistema operativo Windows XP.
- Gestor de Base de Datos Access de Microsoft.

La autora de este documento se ha encargado del diseño de la base de datos, fuente de la información de la aplicación, así como del diseño, desarrollo y puesta en marcha del servidor GESPROT.

Titulación	Ingenieria Técnica de Telecomunicación
Intensificación	Telemática
Departamento	Tecnología de la información y las comunicaciones
Fecha de Presentación	01- 2005

Capítulo 1.- Introducción

1.1 Objetivos del Proyecto y Arquitectura Básica

El director de un Laboratorio de Prótesis Dentales nos propone el desarrollo de una aplicación que le permita conocer y gestionar a través de Internet los datos necesarios para la gestión y administración de su laboratorio (pedidos, facturas, inventario, etc.).

Los requisitos de esta aplicación, a la que hemos llamado GESPROT, son:

- Generación de facturas, albaranes y presupuestos.
- Guardar historial de pacientes.
- Información de cobros (perdidos terminados, entregados, cobrados o pendientes de pago).
- Control de inventario (salida y entradas de materiales en el almacén), distribución de materiales entre los distintos protésicos (quién retira el producto del almacén y en qué pedido lo usa).
- Negociación con proveedores (lista de fabricantes que ofrecen las materias primas para hacer los productos).

Para ello dispone de un sistema informático, ubicado en su propia empresa, con las siguientes características: sistema operativo Windows XP y Gestor de Base de Datos Access de Microsoft. Las funciones de este sistema serán, por una parte, proporcionar los servicios de gestión solicitados, y por otra, albergar la base de datos que permitirá desarrollar las diferentes funciones de gestión. Tanto la interfaz de acceso a la aplicación, como la forma en la que los datos se guardan en la base de datos, quedan a cargo de los diseñadores del proyecto.

La arquitectura de esta aplicación de gestión, que describiremos con mayor detalle en próximos apartados, responde a un modelo en tres capas o modelo vista-controlador. Este modelo, permite un acceso simultáneo y remoto al servidor del Laboratorio Dental, a través de una aplicación Web.

A grandes rasgos, el sistema puede verse del siguiente modo:

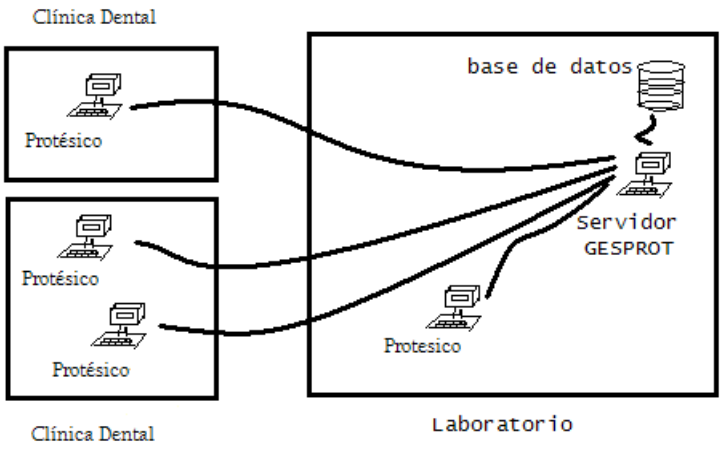


Figura 1: Topología física de la aplicación

Los clientes/usuarios de esta aplicación son los protésicos empleados en el Laboratorio. Usan la información almacenada en la base de datos para gestionar pedidos, consultar el precio de los materiales que consumen, obtener información a cerca de sus pacientes o generar facturas. Sus perfiles, también, se guardan en la base de datos, con fines de identificación, de tal forma que sólo los empleados del Laboratorio puedan acceder a la información proporcionada por la misma.

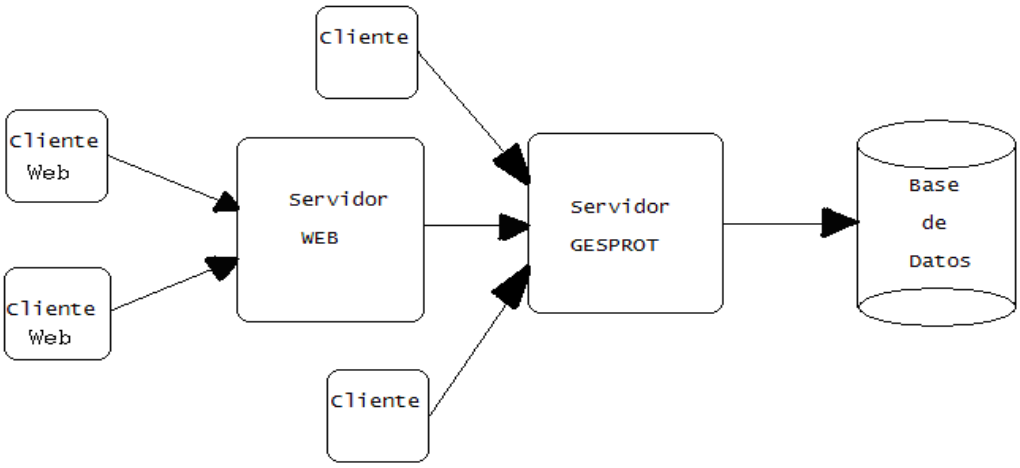


Figura 2: Topología de la aplicación

Una vez definidos los requisitos de la aplicación y su arquitectura básica, indicaremos como se he realizado la asignación de tareas entre los dos integrantes del equipo de trabajo de este proyecto. La autora de este documento se ha encargado del diseño de la base de datos, fuente de información de la aplicación GESPROT, así como del diseño, desarrollo y puesta en marcha del servidor GESPROT, mientras que a su

compañera le corresponde el diseño, desarrollo y puesta en marcha de los clientes (Web y del controlador), así como la configuración del servidor Web.

Al plantearnos, cuáles podían ser los medios más eficaces, fáciles de encontrar en el mercado y económicos para el desarrollo del sistema solicitado, decidimos usar Access como gestor de la base de datos -en este caso en particular, motivado por ser la herramienta ya disponible por el Laboratorio Dental-, SQL como lenguaje para trabajar con la base de datos, y Java+RMI para la implementación del servidor GESPROT.

1.2 Descripción de capítulos

Llegados a este punto, ya conocemos a grandes rasgos cuáles son los requisitos de nuestro trabajo y de qué medios disponemos para llevarlos a cabo, por lo que pasamos a describir el contenido de los capítulos que conforman este proyecto final de carrera.

Capítulo 2.- Herramientas

La aplicación GESPROT, se sirve de una serie de herramientas *software* para su correcto funcionamiento, que se describirán brevemente en este segundo capítulo. En concreto, es necesario disponer de un motor de base de datos (SGDB: *System Generator Data Base*) para almacenar la información: Access; un lenguaje para comunicarse con el mismo: SQL; y otro que facilite le manejo de los datos, así como la implementación de la arquitectura (distribuida) elegida: Java + RMI.

Capítulo 3.- Diseño de la base de datos

En este capítulo se enumeran las tablas que constituyen la base de datos, sus claves, atributos y relaciones.

Capítulo 4.- Implementación del servidor GESPROT

El servidor GESPROT debe facilitar la gestión de información a los empleados del Laboratorio de Prótesis Dental. Para ello proporcionará métodos para añadir, eliminar y modificar los datos almacenados en la base de datos. Asimismo, debe proporcionar métodos específicos que permitan realizar consultas complejas, a partir de las cuales el cliente GESPROT puede calcular estadísticas (por ejemplo, obtener un gráfico con el número de coronas realizadas en los últimos 6 meses), generar facturas, o distribuir los pedidos entre los distintos empleados.

La implementación de dichos métodos se describe en este capítulo, así como la forma de instalar y ejecutar el servidor GESPROT.

Capítulo 5.- Cliente Demo

Breve descripción del diseño y manejo de una aplicación sencilla que ejecuta las operaciones que ofrece el servidor.

Capítulo 6.- Conclusiones

En este capítulo se resumen las conclusiones obtenidas a lo largo de este proyecto, y se proponen un conjunto de líneas de trabajo futuras, todas ellas con el objetivo de ampliar las funciones que actualmente ofrece el servidor GESPROT.

Capítulo 2.- Herramientas

El primer paso para desarrollar el proyecto es elegir las herramientas de las que se va a hacer uso. A continuación, se muestra un breve resumen de las mismas y las razones por las que han sido elegidas.

2.1 Arquitectura Básica: Modelo n-capas. Patrón Modelo-Vista-Controlador

Antes de describir los medios utilizados en el desarrollo del proyecto, describiremos cuál es la arquitectura en la que se basa todo el sistema.

El modelo que usamos es el denominado modelo en *n*-capas. Las capas son cada uno de los elementos que intervienen en la comunicación. En este caso en concreto, usamos un patrón que consta de tres capas o categorías: Modelo, Vista y Controlador (MVC).

2.1.1 Modelo

Es la parte encargada de atender las peticiones, la única que tiene acceso a la base de datos y, por tanto, la capa que desarrollaremos en este proyecto. La llamaremos servidor GESPROT o, simplificando, servidor.

El servidor no tiene información sobre las otras dos categorías: Vista y Controlador, y proporciona a éstas métodos para insertar, modificar, eliminar o recuperar información de la base de datos.

2.1.2 Vista

Es la presentación del Modelo, es decir, "lo que el usuario verá". Únicamente puede usar los métodos que permiten recuperar información a través del Modelo, en nuestro caso, el Servidor.

2.1.3 Controlador

Es el encargado de realizar actualizaciones en el servidor, además de notificar a la Vista los cambios que se produzcan en el Modelo.

El MVC queda definido gráficamente en la siguiente figura (la parte contenida en el recuadro es la desarrollada en éste proyecto):

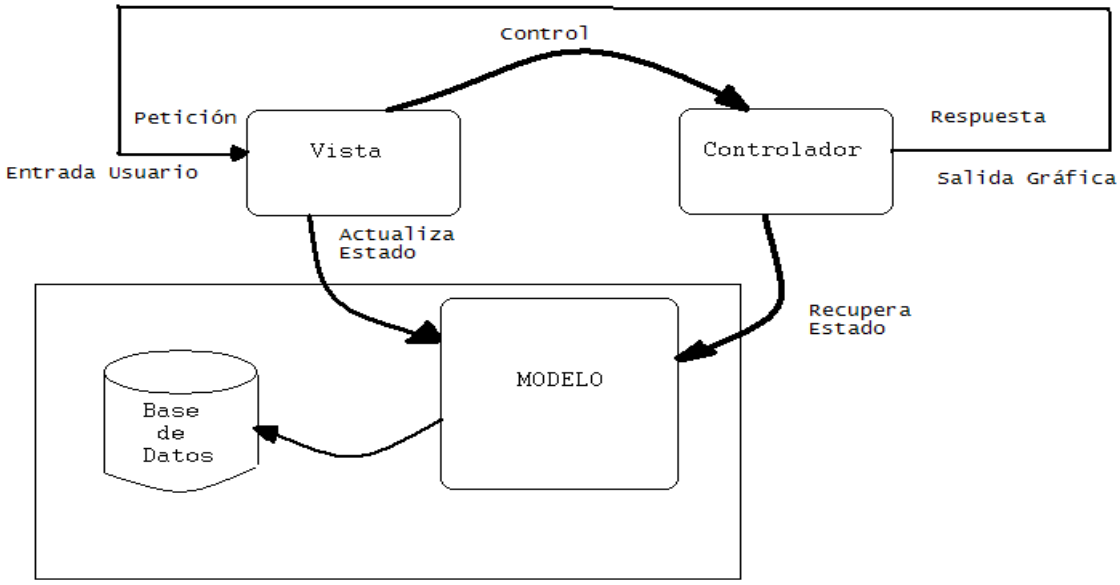


Figura 3: Modelo-Vista-Controlador

2.2. Servidor. Acceso Remoto

Dadas las características del servidor se necesita usar como lenguaje de programación uno que sea orientado a objetos como C++ o Java.

2.2.1 Java

Elegimos Java ya que, además de proporcionar mecanismos de herencia, polimorfismo y otras características útiles para el desarrollo de nuestra aplicación, facilita la integración de las aplicaciones escritas en éste lenguaje dentro de arquitecturas distribuidas, gracias a RMI.

Explicamos brevemente algunas de las características básicas de Java que lo hacen idóneo para la implementación del servidor.

- Herencia: Podemos hacer que varias clases afines compartan atributos y métodos y mejoren las prestaciones de la clase superior o global añadiendo otros (atributos y métodos) nuevos. La clase global y sus "herederas" se relacionan mediante las palabras "es_un".
- Composición: Para hacer que distintos objetos puedan comunicarse y compartir atributos o métodos, se usa este mecanismo. La relación entre las clases que lo usan es la de "tiene_un".
- Polimorfismo: Varios objetos que usen un método de igual firma, es decir, mismo nombre, parámetros y variable de retorno, ejecutan distinto

código al ser llamados. Es útil en clases que usen uno de los dos mecanismos anteriores.

Por último, añadir que Java, permite escribir dos tipos de programas. Los *applets* diseñados para trasportarse por Internet y ejecutarse en la *World Wide Web* y aplicaciones diseñadas para ejecutarse en la computadora del usuario.

2.2.2. Java + RMI

Para hacer más sencilla la creación de código implementamos el servidor como un programa sencillo, que trabajará para clientes situados en la misma máquina. Una vez que funcione extenderemos el código para hacer que el programa servidor proporciones servicio a aplicaciones remotas a través de Internet. Para ello usamos RMI.

RMI (*Remote Method Invocation*) es una tecnología de sistemas distribuidos sencilla, ya que no se preocupa de gestionar la colaboración de objetos de distintas plataformas programados en diferentes lenguajes. Es Java quién se encarga de solucionar los problemas de heterogeneidad. Así, su API es más sencillo y natural al no contemplar qué tipos de datos (y sus tamaños) existen o no en los lenguajes en los que se implementan el servidor y el cliente.

Ésta tecnología permite el acceso simultáneo de varios clientes, y controla la concurrencia de los mismos creando un hilo diferente para atender las peticiones de cada uno.

El cliente y el servidor pueden estar localizados en máquinas distintas, por tanto, el servidor debe proporcionar su dirección para que al cliente le sea fácil encontrarlo. Esto puede hacerse proporcionando una sencilla dirección URL (modo que usaremos en éste servidor) del tipo:

```
rmi://nombre_host:puerto/nombre_objeto
```

Y también usando el registro RMI (`rmiregistry`) que comunica objetos cliente con servidores.

Por último, notar que casi siempre el paso de parámetros entre el servidor y el cliente se hará por valor, puesto que se transmiten objetos primitivos, lo que hace que no sea necesaria su serialización. Aún así, se usa la clase `Serializable` facilitando ampliaciones en el diseño del servidor y la base de datos.

2.3. Base de Datos

2.3.1. SQL

Java no permite un acceso directo a cualquier base de datos, pero sí que proporciona las clases necesarias para ejecutar y obtener resultados a partir de sentencias SQL. Estas clases son: `PreparedStatement` y `ResultSet`.

SQL (*Structured Query Language*) es el lenguaje estándar de comunicación con cualquier base de datos más extendido. La independencia de SQL respecto del SGBD con el que se interaccione hace que el diseño del servidor sea extensible a otros sistemas que no usen la misma base de datos.

El lenguaje SQL proporciona una interfaz de lenguaje *declarativo* de alto nivel, de manera que el usuario sólo tiene que especificar cuál es el resultado esperado, dejando que el SGBD se encargue de la optimización actual y de las decisiones sobre cómo se ejecutará la consulta.

2.3.2. Access

Se trata de un potente sistema SGBD (*Sistema de gestión de base de datos*) proporcionado por Microsoft Office.

Usaremos únicamente el Microsoft Jet Engine de Access que nos proporciona el Acceso a los datos heterogéneos a través de ODBC (*Open Database Connectivity*), validación de datos, control de concurrencia usando bloqueos y la optimización de consultas.

ODBC es el mecanismo de comunicación con bases de datos que incluye Windows en sus Sistemas Operativos, a partir de la versión 98.

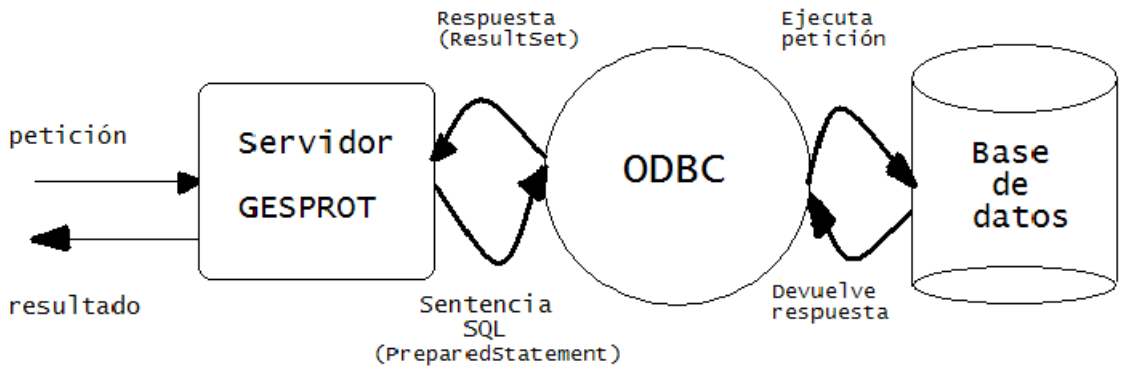


Figura 4: Herramientas que intervienen en la comunicación: Java-SQL-ODBC-Access

Capítulo 3.- Diseño de la Base de Datos

El primer paso en el diseño de la base de datos es saber exactamente qué datos se deben de almacenar. Esta decisión, se toma en colaboración con el director del Laboratorio de Prótesis Dental.

Los datos a almacenar se pueden agrupar en cinco grandes grupos:

- Clientes. El grupo clientes engloba a los clientes directos del Laboratorio, es decir, a las clínicas dentales o a los dentistas. Son los que encargan los pedidos y los que abonan las facturas, aunque no sean los destinatarios finales del producto.

  Los destinatarios finales son los Pacientes. De los pacientes, se guarda un pequeño historial, para facilitar posibles repeticiones de trabajos, o disponer de información como el color dental, para la fabricación de nuevas piezas.

- Empleados. El grupo empleados se corresponde con los empleados del Laboratorio, es decir, con los protésicos. Dentro del grupo empleados, se define una estructura jerárquica (director, administrador, empleado sin atribuciones especiales), cuyas funciones son:

  - Director: Se encarga de gestionar el ejercicio de la empresa. Controla las operaciones de compra y venta, contratación de personal, asignación de pedidos entre los distintos protésicos empleados, ...
  - Administrador: Controla la contabilidad de la empresa (balance diario de caja, nóminas, seguros sociales,...). Ayuda al director del laboratorio a tomar las decisiones de compra-venta,...
  - Empleado sin atribuciones especiales: Son la "mano de obra", los encargados de fabricar los productos de la empresa (coronas, fundas molares,...)

  A cada grupo jerárquico le corresponde una clave de acceso diferente.

- Inventario. El grupo inventario engloba por una parte a los proveedores y por otra a las materias primas disponibles en el laboratorio, tales como: resinas, cerámica o silicona.

- Contabilidad: A este grupo pertenecen tanto los pedidos (trabajos en curso), como las ventas (trabajos finalizados). A partir de los elementos de este grupo se podrán generar albaranes y facturas.

En el siguiente diagrama se muestran los diferentes grupos y subgrupos de datos:

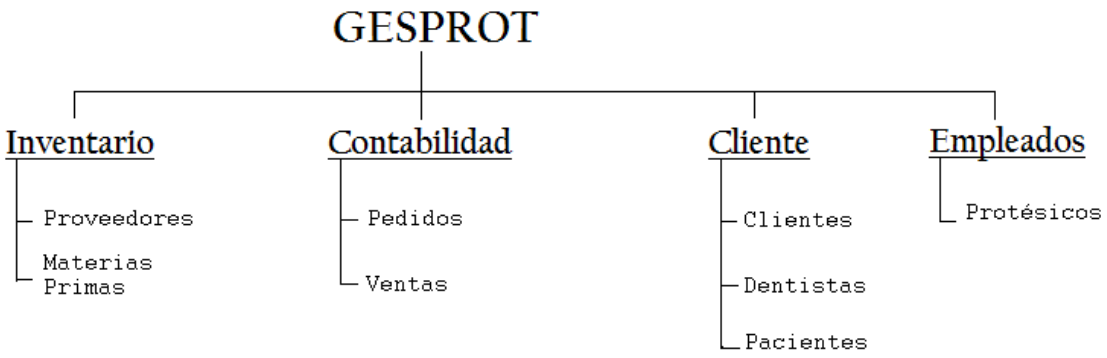


Figura 5: Estructura de la empresa

3.1 Tablas de la base de datos

Atendiendo al esquema de la base de datos se crean las distintas tablas que son necesarias para albergar todos los componentes de la empresa y se definen los atributos de cada una de ellas, atributos (columnas de las tablas) que satisfacen las necesidades de almacenamiento de datos del laboratorio. Los nombres de dichas tablas son, por orden alfabético: Clínicas, Dentistas, Pacientes, Pedidos, Productos, Protésicos, Proveedores y Ventas.

Todas ellas compartirán tres atributos indispensables para crear cualquier fila dentro de la tabla:

- FechaInsercion: Atributo de tipo *fecha/hora* que define la fecha en la que la tupla fue creada en la tabla. Se usa para, por ejemplo, saber cuando se contrató determinado empleado en la tabla protésicos, o cuál fue la fecha en la que se encargó un pedido en la tabla del mismo nombre.

- Nombre: Campo de tipo *texto* que incluye el nombre del objeto que se introduce en la tabla. Es decir, el nombre del Protésico, Producto, de la Clínica,...

- ID: Columna de tipo *número* que sirve para identificar inequívocamente cada elemento nuevo que se introduce en la tabla. Es Access quien

asigna un valor a este campo, por eso se le define también como *autonumérico*.

El atributo *Notas*, se usa para albergar un *array* de *bytes* a partir del cual el servidor podrá recomponer un archivo en el que guardará anotaciones sobre el elemento que se alberga en una determinada fila. Todas las tablas tienen una columna de éste tipo (*Objeto OLE*) salvo una cuyo archivo anotaciones es especial: Pacientes. En esta tabla el archivo es más completo que el Notas, el fichero guarda un historial del paciente al que se le fabrica el producto con información del tipo: color del diente, forma de la dentadura, o tamaño de los incisivos.

Los demás atributos de la tabla son propios de cada elemento de la estructura de la empresa que define. Los tipos de datos de cada columna son como los descritos hasta ahora, es decir, *fecha/hora*, *Texto*, *número*, *autonumérico u objeto OLE*. A continuación, se explican brevemente los formatos y la información que pueden contener cada uno de estos tipos:

- Fecha/hora: El tipo *fecha/hora* tiene el siguiente formato: día/mes/año (por ejemplo, 01/01/1970). Puede almacenar valores de fecha y hora para los años del 100 al 9999.
- Texto: Texto o combinaciones de texto y números. Éste tipo equivale a `String` en Java. El tamaño máximo de este campo es 255 caracteres.
- Número: Son los datos *numéricos* utilizados en cálculos matemáticos.
- Autonumérico: Número secuencial (incrementado de uno a uno) único, o número aleatorio que Microsoft Access asigna cada vez que se agrega un nuevo registro a una tabla. Los campos *Autonumérico* no se pueden actualizar.
- Objeto OLE: Objeto (como por ejemplo una hoja de cálculo de Microsoft Excel, un documento de Microsoft Word, gráficos, sonidos u otros datos binarios) vinculado o incrustado en una tabla de Microsoft Access.

La inserción de las columnas y propiedades de la base de datos se hace en el modo diseño. A continuación, se muestra un ejemplo de una tabla en dicho modo y de la hoja de datos resultante.

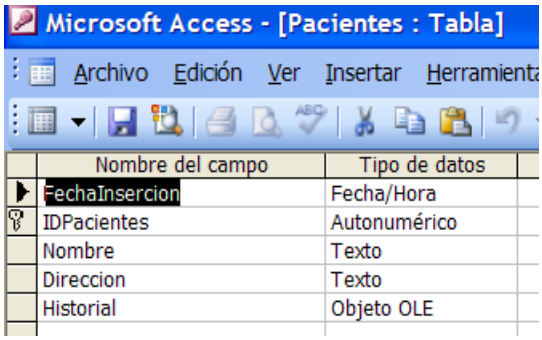


Figura 6: Vista diseño de la tabla Pedidos

Figura 7: Vista hoja datos de la tabla Pedidos

3.1.1 Tabla Clínicas

La tabla Clínicas almacena la información concerniente a las clínicas dentales para las que trabajan los protésicos de la empresa. Con los datos de ésta tabla pueden generarse facturas (ya que contiene la dirección y otros datos importantes) y pueden almacenarse, por ejemplo, anotaciones con detalles que los usuarios de la aplicación deban recordar. Se explican detalladamente los campos de los que está compuesta Clínicas a partir de la siguiente figura:

Figura 8: Tabla Clínicas

- FechaInserción: Atributo de tipo *fecha/hora* que indica el momento en el que la Clínica encargó su primer contrato al laboratorio.

- IDClínicas: Campo que identifica la clínica de manera única mediante un número.

La información que contienen las otras columnas es la que corresponde a su *Nombre de Campo*. Aunque resulte obvia, al ser esta la primera tabla explicaremos la información que contiene cada tabla.

- NIF/CIF: Campo que contiene el número de identificación fiscal de la empresa o persona de la fila.
- Nombre: Campo que identifica a la clínica por su nombre. Los nombres (si se trata de un sujeto) se ponen separados por comas, primero los apellidos y después el nombre de pila. Si se trata de una empresa se escribe primero el nombre y después separado por comas el tipo de empresa (S.L.,S.A., Cooperativa,...).
- Dirección: Lugar en el que está situada siguiendo el siguiente orden:

  Calle, Nº puerta, piso, escalera, letra.

- Ciudad: Ciudad a la que pertenece. Los nombres de ciudades compuestos se separan por espacios en blanco.
- CP: 5 dígitos con el distrito postal de la Clínica.
- Provincia: Comarca en la que está situada la clínica poniendo primero el nombre y después el tipo de autonomía (Región, Comunidad, Principado,...).
- País: Nombre, en castellano, del pais en el que se trabaja.
- Tfno: Número de teléfono de la clínica, sin el prefijo del país pero sí el de la región.
- Fax: Número de fax sin el prefijo del país pero sí el de la región.
- Email: Correo electrónico de la clínica.
- Notas: Características especiales de la empresa a recordar por los protésicos.

3.1.2 Tabla Dentistas

Es muy parecida a la tabla Clínicas, ya que almacena también información correspondiente a clientes de la empresa. Ésta vez los datos son los de los Dentistas. La Vista Diseño de la tabla es la que sigue:

Dentistas : Tabla

Nombre del campo	Tipo de datos
FechaInsercion	Fecha/Hora
IDDentistas	Autonumérico
NIF/CIF	Número
Nombre	Texto
Direccion	Texto
Ciudad	Texto
CP	Número
Provincia	Texto
Pais	Texto
Tfno	Número
Fax	Número
Email	Texto
Notas	Objeto OLE

Figura 9: Tabla Dentistas

El contenido de cada campo está explicado en el apartado *3.1.1 Tabla Clínicas.*

3.1.3 Tabla Pacientes

En ésta tabla se guardan los datos necesarios del usuario final de los productos de la empresa. Es interesante guardar información del Paciente a modo de historial para facilitar la realización de sucesivos trabajos a la misma persona, independientemente del Dentista o Clínica que encargue el pedido. Así, la tabla Pacientes tiene la siguiente forma:

Pacientes : Tabla

Nombre del campo	Tipo de datos
FechaInsercion	Fecha/Hora
IDPacientes	Autonumérico
NIF/CIF	Número
Nombre	Texto
Direccion	Texto
Historial	Objeto OLE

Figura 10: Tabla Pacientes

El contenido del resto de los campos está debidamente explicado en el apartado *3.1.1 Tabla Clínicas.*

3.1.4 Tabla Pedidos

La tabla pedidos contiene información acerca de un trabajo concreto, asignado a un protésico concreto, encargado por una clínica y/o dentista concretos y cuyo destinatario es un Paciente determinado.

El trabajo fue encargado en una fecha y ha de entregarse un día concreto. También han de almacenarse Notas con características especiales del encargo o incidencias de cualquier tipo derivadas del mismo.

Para la realización del pedido es necesario el uso de un determinado número de productos cuyos nombres y acumulación de precios ha de almacenarse (para facilitar las labores de inventario dentro de la empresa).

Por último, es necesario conocer el precio total del trabajo (precio de los productos usados en el mismo, más el de la mano de obra necesaria para su realización) y si éste ha sido ya cobrado o no, por lo que se añade una columna booleana (*IncluidoEnVentas*) que lo indica.

Además el NIF/CIF se refiere a su número de identificación ya sea fiscal o referente a una identificación propia de la actividad de la empresa, por ejemplo:

nº factura expedida en el ejercicio en curso/mes en el que se expide, (23/03: Factura número veintitrés expedida en el mes de marzo del año fiscal 2005).

El resto de información queda almacenada en la base de datos del siguiente modo:

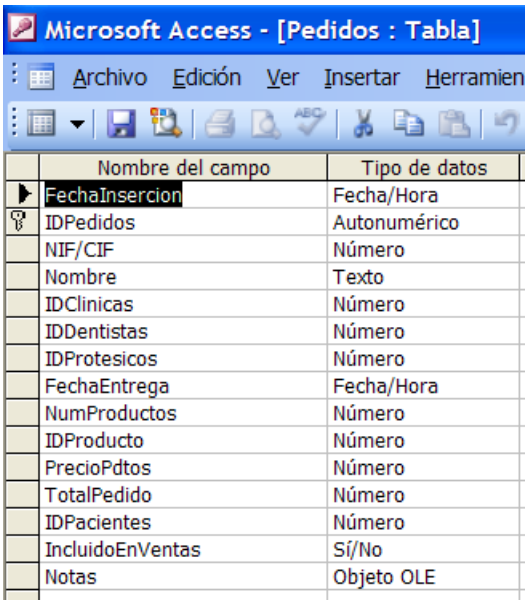


Figura 10: Tabla Pedidos

El contenido del resto de los campos está debidamente explicado en el apartado *3.1.1 Tabla Clínicas.*

3.1.4 Tabla Productos

Guarda información correspondiente a los productos almacenados en el almacén: Nombre, precio, tipo (coronas, fundas, masilla, cola,...), existencias en el almacén, existencias encargadas a los proveedores y Notas con alguna característica a resaltar sobre el producto en cuestión.

Figura 11: Tabla Productos

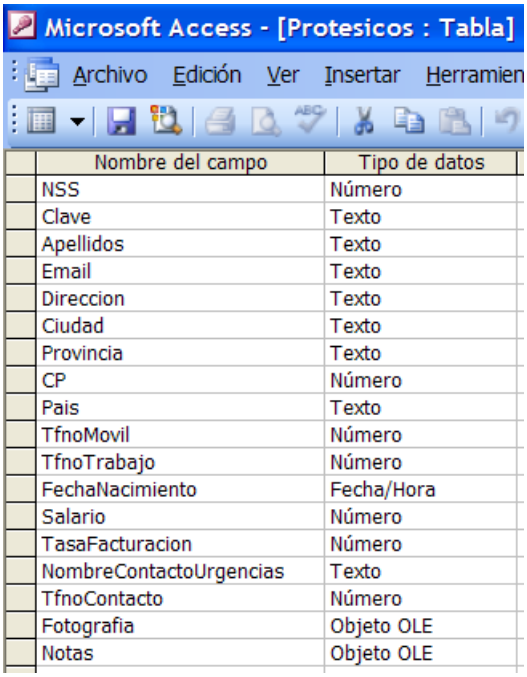
El contenido de los campos nombre, IDproducto y notas está explicado en el apartado *3.1.1 Tabla Clínicas.* El formato del resto es el que sigue:

- PrecioUnidad: Valor de una unidad del producto, en euros y con el I.V.A.
- TipoProducto: Categoría entre todos los tipos de producto (resinas, ganchos,...) a la que pertenece.
- NIF/CIF: Código de barras o identificación fiscal del producto.
- NumExistAlmacen: Unidades del producto almacenadas en el Laboratorio.
- UDPedidas: Nº de productos a la espera de ser recibidas del proveedor.

3.1.5 Tabla Protésicos

Se trata de la tabla más extensa de todas las de la base de datos. Contiene información acerca de los empleados del laboratorio, por tanto, los usuarios de esta aplicación. Debido a la extensión de la tabla se muestran en la siguiente figura

únicamente los campos especiales, omitiéndose los de *FechaInserción, Nombre e IDProtésico* cuyas características ya conoce el lector.



Nombre del campo	Tipo de datos
NSS	Número
Clave	Texto
Apellidos	Texto
Email	Texto
Direccion	Texto
Ciudad	Texto
Provincia	Texto
CP	Número
Pais	Texto
TfnoMovil	Número
TfnoTrabajo	Número
FechaNacimiento	Fecha/Hora
Salario	Número
TasaFacturacion	Número
NombreContactoUrgencias	Texto
TfnoContacto	Número
Fotografia	Objeto OLE
Notas	Objeto OLE

Figura 12: Tabla Protésicos

- NSS: Campo en el que se almacena el número de la seguridad social del empleado.
- Clave: Contraseña que cada empleado usa, dependiendo de ésta el usuario tiene restringidos o no los permisos de uso a la base de datos, del siguiente modo:
    - Director y Administrador: Permiso de escritura y lectura en todas las tablas, es decir, operaciones de inserción, modificación, consulta y eliminación.

    Se muestran algunos ejemplos de operaciones que pueden realizarse: Modificar precios de los productos (modificar tabla Productos), insertar nuevos clientes (Añadir fila a tabla Dentistas o Consultas), despedir empleados (Eliminar fila en Protésicos), aumentar salario a los mismos (Modificar fila/s en Protésicos), cobrar trabajos (Modificar fila en Pedidos y Añadir fila en Ventas),...
- TasaDeFacturación: Este atributo almacena la media de Pedidos entregados hasta el momento por el protésico, define el rendimiento del

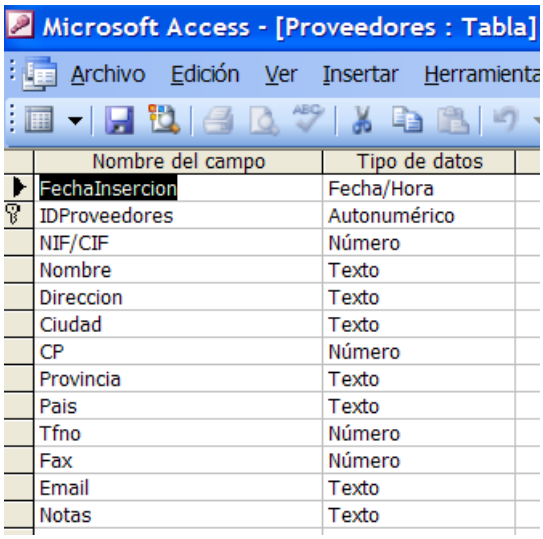
empleado de cara a posibles incentivos, ascensos dentro de la estructura de la empresa, etc.

- Salario: Se trata del salario base, a éste valor se pueden añadir en el momento del cobro del empleado las horas extras y/u otros incentivos que la empresa estime oportuno.
- Fotografía: Campo que, al igual que el campo *Notas* ya descrito, almacena un grupo de *bytes* que al ser recompuestos en el cliente mostrarán una imagen (*vease Método mostrarImagen en el Capítulo 4*).

La información almacenada en el resto de los campos es la correspondiente a su respectivo *Nombre de Campo*. El contenido de éstos se encuentra debidamente explicado en el apartado *3.1.1 Tabla Clínicas*.

3.1.6 Tabla Proveedores

La tabla Proveedores contiene los datos necesarios para identificar a los suministradores de los distintos productos que usa la empresa. Sus campos son los siguientes:



Nombre del campo	Tipo de datos
FechaInsercion	Fecha/Hora
IDProveedores	Autonumérico
NIF/CIF	Número
Nombre	Texto
Direccion	Texto
Ciudad	Texto
CP	Número
Provincia	Texto
Pais	Texto
Tfno	Número
Fax	Número
Email	Texto
Notas	Texto

Tabla 13: Tabla Proveedores

El contenido de cada campo está explicado en el apartado *3.1.1 Tabla Clínicas*.

3.1.7 Tabla Ventas

Almacena los datos que definen un trabajo ya terminado y cobrado al cliente, ya sea éste una Clínica o un Dentista. Así como el importe de dicho pedido ya entregado.

Figura 14: Tabla Ventas

El contenido de cada campo está explicado en el apartado *3.1.1 Tabla Clínicas*, salvo el de precio (que contiene el valor en euros del importe de la venta con el I.V.A.) y el de NIF/CIF que cumplen la misma función que en el apartado *3.1.4 Tabla Pedidos*.

3.2 Relaciones entre tablas

Una base de datos consta de tablas y las relaciones entre dichas tablas. Cada tabla tiene atributos que las definen y filas o tuplas cuyo contenido son los datos que interesa guardar. Se pueden añadir características especiales a los atributos de manera que estos campos definan relaciones entre las filas de las tablas o unicidad a las mismas. Las características especiales que se usan en ésta base de datos son las siguientes:

- Clave Primaria: Hace que el valor de ese campo sea irrepetible, haciendo que la fila de la que forma parte sea única.
- Indexado: Obliga a que el valor de un atributo de una tabla que hace referencia a un campo de otra exista realmente en ésta última. Normalmente la columna a la que se hace referencia es el atributo clave de la tabla.

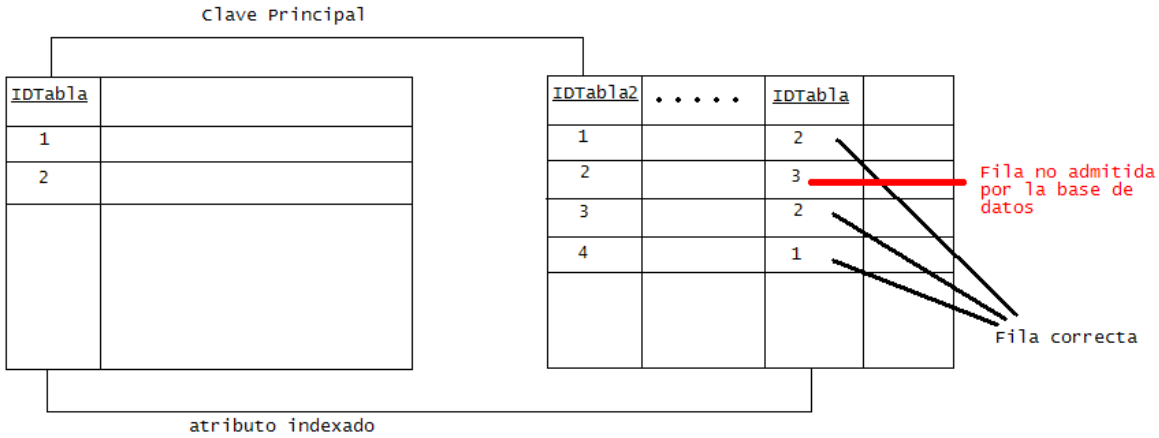


Figura 15: Esquema de claves e indexado

Para Access las características Clave Primaria e Indexado se diferencian únicamente en los duplicados que puedan hacerse de esos campos. Así podemos diferenciar entre la *relación 1 a N*, que en Access se denota con *Indexado Sin Duplicados*, y la relación *N a N*, que se denomina *Indexado Con Duplicados.*

El campo usado como clave, y por tanto, el utilizado para indexar tablas, es el que servirá como identificador de cada una de las filas que conforman las distintas tablas de la base de datos.

En la siguiente figura se muestran los campos en los que se definen las relaciones de indexado de las columnas de la base de datos:

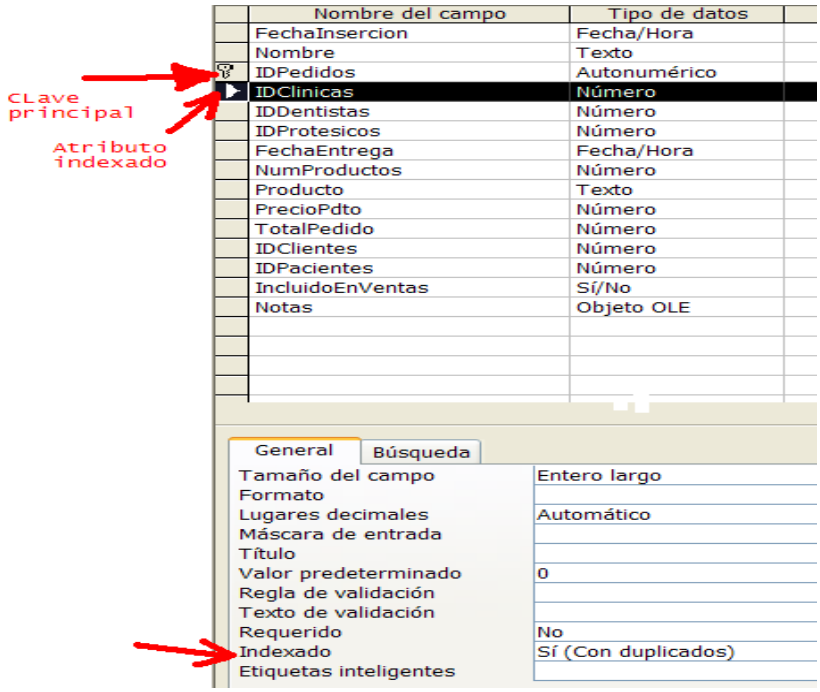


Figura 16: Claves e indexado en Access

3.2.1 Relaciones entre tablas

El campo *IDTabla* hace que cada fila sea única dentro de la tabla. Es ésta característica la que lo hace idóneo de ser la *Clave Primaria* de cada una de dichas tablas. Así, se usará este mismo campo para relacionar unívocamente algunos atributos de *Pedidos* y *Ventas* con atributos del resto de tablas de la base de datos del siguiente modo:

La tabla *Pedidos* guarda una referencia al Cliente, ya sea *Clínica* o *Dentista*, que encargó el trabajo, al *Protésico* que lo realiza y al *Paciente* que lo recibirá. Para asegurarnos que los datos a los que se hace referencia existen realmente en la tabla usamos *Indexado Con Duplicados*.

Los campos elegidos para relacionar *Pedidos* con el resto de tablas son los campos *Clave Primaria* de cada una de ellas, es decir, *IDClínicas, IDDentistas, IDProtésicos e IDPacientes.*

De la misma manera y haciendo referencia a las mismas tablas se relacionan los valores en la tabla *Ventas* de los campos *IDClínicas, IDDentistas, IDProtésicos e IDPacientes* con los valores de los atributos de mismo nombre en las tablas *Clínicas, Dentistas, Protésicos y Pacientes.*

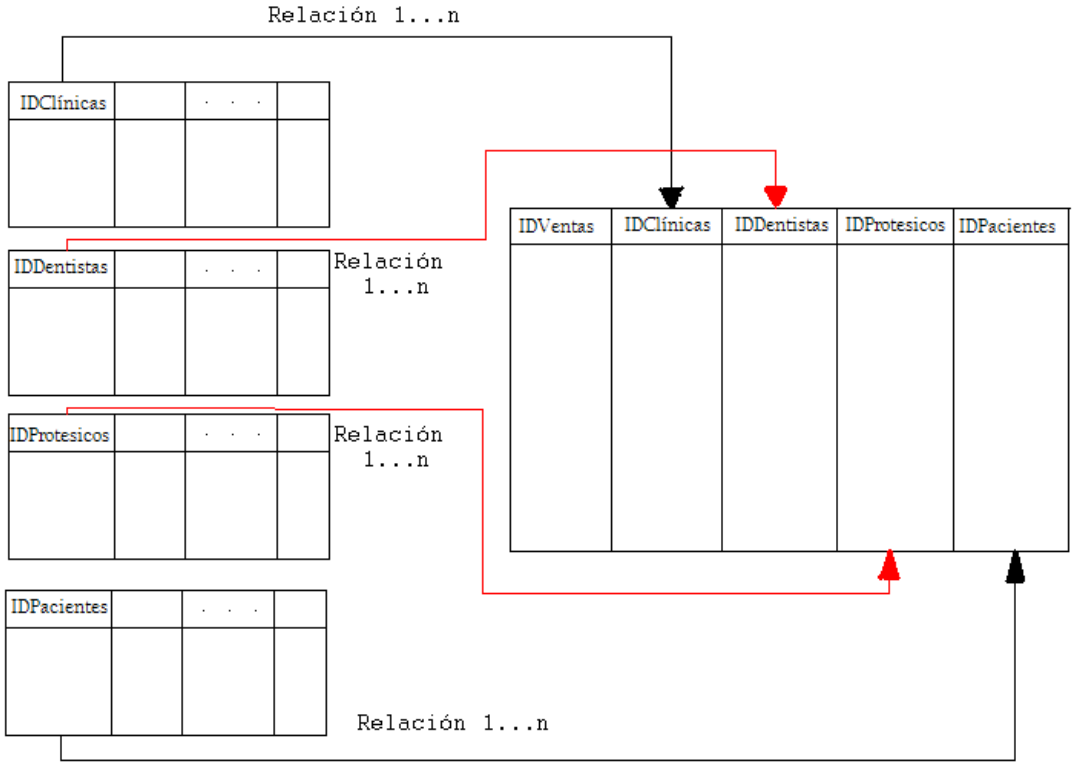


Figura 17: Indexado de la tabla Ventas con otras tablas

Capítulo 4.- Implementación del servidor

El servidor debe comunicarse con dos elementos diferentes (Servidor Web y Base de Datos) utilizando idiomas distintos (Java+RMI y SQL). Se distinguirá entonces entre *métodos y objetos que se usan para la comunicación con la base de datos y métodos que usa el Servidor Web o el cliente para realizar peticiones.*

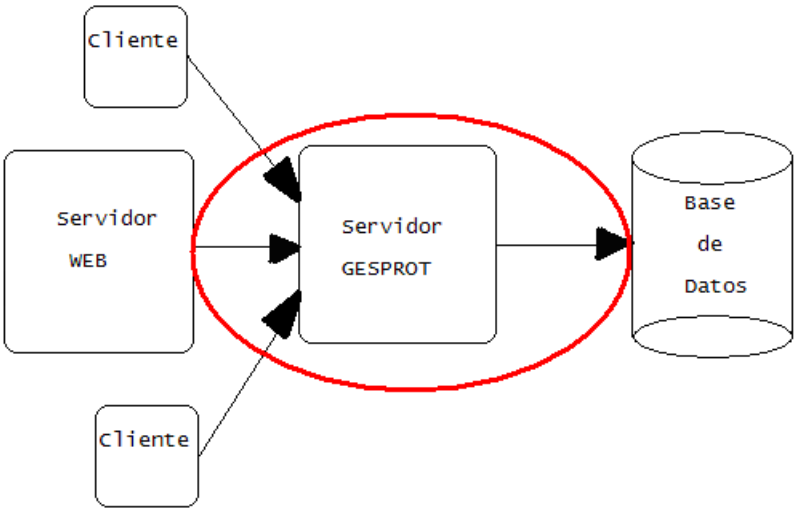


Figura 18: Relaciones del servidor

Los paquetes y clases que componen la implementación del servidor se relacionan de la siguiente manera:

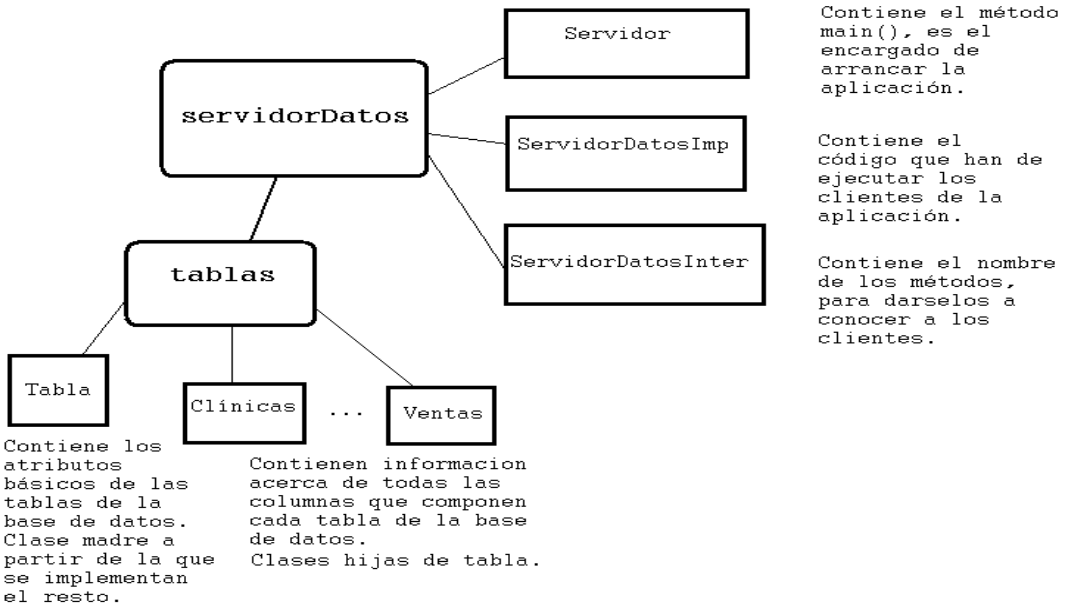


Figura 19: Paquete ServidorDatos

Antes de explicar detalladamente cuales son las clases y métodos del *servidor GESPROT*, y su funcionamiento, se informa al lector que el código del programa se encuentra disponible en el CD del proyecto adjunto.

4.1.- Clase Servidor y conexión con la base de datos

Es la clase *servidor* la que arranca esta parte de la aplicación, es decir, el *servidor GESPROT*.

Carga el archivo *servidor.conf* a partir del cual consigue el puerto, la máquina y la identidad RMI con los cuales iniciará y compondrá una *url* del servidor para darlo a conocer a los posibles clientes del servidor.

Los siguientes métodos pertenecen a la clase *ServidorImpl*, *n*o se incluyen en la interfaz del Servidor (*ServidorInter)* ya que no es necesario que los usuarios de la aplicación lo conozcan.

- ExtablecerConexiónBD: Método privado de la clase *ServidorImpl*, se le llama al crear un objeto de la misma. Carga el archivo *bases.conf* a partir del cual se extraen la *url* de nuestra base de datos y el *driver* del ODBC que la manejará. Conocidos éstos se establece la conexión con la base de datos.
- Finaliza: Al cerrar la aplicación se debe terminar también con la conexión a la base de datos usando éste método.
- GetRuta: Proporciona la dirección del archivo *bases.conf*.

4.2.- Clases y métodos del servidor

Los métodos de *ServidorDatosInter* (implementados en *ServidorDatosImpl*) son visibles para el servidor Web o para cualquier otro cliente que lo usara.

El servidor proporciona métodos a los clientes para realizar consultas o actualizaciones en la base de datos. Para proporcionar estos servicios necesita conocer detalladamente las tablas de las que esta formada, así se crea una clase que satisfaga esa necesidad: *Tabla.*

4.2.1.-Métodos y objetos que definen la base de datos. Clase Tabla

Para comunicarse con la base de datos, el servidor debe tener una representación (objeto) de cada una de sus filas, para así recoger la información antes de insertarla o al extraerla de la base de datos. Por tanto se define un subpaquete Java al que se llama `tablas`.

Es en las clases de éste paquete donde usaremos el mecanismo de *herencia* que proporciona Java, dado que todas las tablas comparten tres atributos (*Nombre*, *FechaInserción e IDTabla)* se define una clase genérica o superclase llamada `Tabla`.

El contenido del paquete `Tablas` es el único que guarda una relación directa con la base de datos, esto hace que la implementación del servidor GESPROT sea extensible a otras empresas o bases de datos con contenidos diferentes, basta con crear clases que hereden de *Tabla* y cuyos atributos sean los de las tablas de la nueva base de datos.

Existen varios tipos de tablas en esta base de datos: *Clínicas*, *Dentistas*, *Pacientes*, *Pedidos*, *Productos*, *Protésicos*, *Proveedores y Ventas*. Definiremos una *clase* por cada tabla. Todas ellas serán hijas o subclases de `Tabla`. Los atributos de cada subclase serán equivalentes a cada columna de la tabla de la base de datos que le corresponde.

La clase Tabla tiene las características que siguen:

- Atributos: Los atributos de `Tabla` son `FechaInsercion`, `Nombre` e `IDTabla`. Los valores de los dos primeros se introducen al crear el objeto, es decir, son parámetros del constructor. El valor del tercero será introducido por el SGBD en la fila que corresponda dentro de la tabla de la base de datos.
- Métodos: Los métodos de `Tabla` sirven para obtener o insertar información en la base de datos, son sobrescritos (haciendo uso del *polimorfismo* de Java) por todas sus subclases adaptándolos cada una de ellas a sus características particulares. Así, únicamente describiremos los métodos de la clase madre. Los métodos de las clases hijas funcionaran de manera parecida.
  - `getAtrib`: Método que devuelve el nombre del atributo que está colocado en determinada posición en la base de datos.

- `getAtribNum`: Método que devuelve el número de atributo según el orden en el que están situados en la tabla la base de datos.
- `getAtributosDeseados`: Devuelve una tabla con los atributos rellenos sólo con las columnas requeridas por el cliente.
- `getName`: Método que devuelve el nombre de la clase.
- `getTupla`: Devuelve el número de atributos que definen a cada tupla de la tabla.
- `getParametros`: Extrae todos los atributos de una fila.
- `getParametrosBásicos`: Extrae los atributos *`Nombre`* y *`FechaInserción`* de la tabla. En esta tabla los métodos `getParametros` y `getParametrosBasicos` son idénticos.
- `setParametros`: Introduce los atributos en la tabla.

4.2.2.- Métodos para la comunicación entre el Servidor Web y la base de datos.

Aprovechando que Java distingue los métodos por sus firmas asignamos el mismo nombre a varios métodos que realizen la misma función pero que usen parámetros distintos para llevarla a cabo, facilitando así el manejo de la aplicación *servidor GESPROT* a los programadores de clientes que la usen.

Es en estos métodos donde se realiza la comunicación con la base de datos, por tanto, es donde se usa el lenguaje SQL. Todos los métodos tienen un esquema de operación parecido, con los siguientes pasos:

- Comprueba que el cliente tiene permiso para ejecutar la operación que pide. Los protésicos administradores o director tienen permiso de lectura y escritura (*PEsc*) mientras que los protésicos usuarios solo tienen permiso de lectura (*PLect*).
- Compone una cadena con las columnas (de la tabla con la que se comunicará) que intervendrán en la operación, y otra cadena, si fuese

necesaria, con interrogantes en los lugares que más tarde se insertará un valor que todavía no se conoce.

- Se crea la sentencia SQL que corresponda a la operación requerida por el usuario.
- Se crea un objeto `PreparedStatement` con la sentencia SQL y, si es necesario, se cambian los interrogantes por sus correspondientes valores. (Estos datos se introducen al llamar al método, pero no son conocidos en el momento de escribir el código).
- Se ejecuta la sentencia.
- Se recoge el resultado de la operación usando el objeto `ResultSet` devuelto por la sentencia anterior.
- Se extrae la información requerida del `ResultSet`. Si la petición abarca varias filas se debe crear un objeto de la tabla que corresponda, por cada fila que satisfaga la petición.
- Se cierran los objetos `ResultSet` y `PreparedStatement`.

Una vez conocida la estructura de los métodos de consulta/modificaciónes de *ServidorImpl* los dividiremos en las siguientes categorias según la petición que satisfagan:

- AñadeTupla: Inserta una nueva fila en la base de datos. Para crear una nueva fila únicamente es necesario proporcionar el *Nombre* y la *Fecha de Inserción.*

```
  public String añadeTupla(Tabla tabla, String Clave) throws
RemoteException;
```

- EliminaTupla: Borran una o varias filas de la base de datos. Hay que ser cautos al usar éste método ya que no se podrá usar nunca más el identificador correspondiente a la fila que se elimina. Se puede eliminar a partir de uno de los tres atributos básicos para una fila.

```
 public String eliminaTupla(Tabla tabla,int ID, String Clave) throws
RemoteException;

  public String eliminaTupla(Tabla tabla,Date FechaInsercion, String
Clave) throws RemoteException;
```

```
public String eliminaTupla(Tabla tabla,String Nombre, String Clave)
throws RemoteException;
```

- GetTupla/s: Métodos que devuelven una o varias filas de la tabla según uno de los tres atributos básicos (*Nombre, FechaInsercion o IDTabla)*.

```
public Vector getTupla(Tabla tabla, int ID, String Clave) throws
RemoteException;

public Vector getTuplas(Tabla tabla, Date FechaInsercion, String
Clave) throws RemoteException;

public Vector getTuplas(Tabla tabla, String Nombre,String Clave)
throws RemoteException;
```

- GetTuplaBasica: Método que a partir del identificador de la fila muestra el *Nombre* y la *FechaInserción* de la misma.

```
public Vector getTuplaBasica(Tabla tabla, int ID, String Clave)
throws RemoteException;
```

- GetAtributos: Métodos para recuperar información de la base de datos atendiendo a patrones impuestos por el cliente. Las sentencias SQL variarán según el tipo de consulta que se haga :
  - Si es un patrón de *caracteres, números o fecha*. Podemos hacer una consulta especificando la parte que conocemos y pidiendo a la base de datos que devuelva todos los que cumplan ese requisito pero que no sepamos cómo acaba. Los caracteres "comodín" son:
    - &, que representa una cadena en la posición en la que se sitúa.
    - _, que representa un carácter en la posición ocupada.

```
public Vector getAtributos(String[] atributosDeseados, Tabla
tabla,String atributoEvaluado, String condicion,boolean equals, String
Clave)throws RemoteException;

public Vector getAtributos(String[] atributosDeseados, Tabla
tabla,String atributoEvaluado, int condicion,boolean equals, String
Clave)throws RemoteException;

public Vector getAtributos(String[] atributosDeseados, Tabla
tabla,String atributoEvaluado, Date condicion,boolean equals, String
Clave)throws RemoteException;
```

- Devuelve los atributos que tienen valores nulos o no nulos (valor nulo no es lo mismo que valor cero).

```
public Vector getAtributos(String[] atributosDeseados, Tabla tabla,String atributoEvaluado, boolean equals, String Clave)throws RemoteException;
```

- Devuelven una serie de columnas de las filas de una tabla comprendidas en un intervalo concreto de los valores de un atributo especificado por el usuario: El componente que se indica puede ser una *fecha* o un *entero*.

```
public Vector getAtributos( String[] atributosDeseados, Tabla tabla, String atributoFechaEvaluado, Date inicioIntervalo, Date finIntervalo, String Clave)throws RemoteException;
```

```
public Vector getAtributos( String[] atributosDeseados, Tabla tabla, String atributoNumericoEvaluado, int inicioIntervalo, int finIntervalo, String Clave)throws RemoteException;
```

- Métodos que devuelven una serie de atributos de las filas de una tabla a partir de otros que tienen un valor o valores indicados por el cliente. Los tipos de valores sobre los que el cliente pregunta pueden ser de tipo *String*, *fecha* o *entero*. Además puede que el usuario pregunte por uno o varios valores, por ejemplo, puede que el cliente busque las filas de la tabla *Productos* cuyos precios sean 6 y 12 euros.

```
public Vector getAtributos(String[] atributosDeseados, Tabla tabla, String atributoEvaluado, String condicion, String Clave)throws RemoteException;
```

```
public Vector getAtributos(String[] atributosDeseados, Tabla tabla, String atributoEvaluado, int condicion, String Clave)throws RemoteException;
```

```
public Vector getAtributos(String[] atributosDeseados, Tabla tabla, String atributoEvaluado, Date condicion, String Clave)throws RemoteException;
```

```
public Vector getAtributos(String[] atributosDeseados, Tabla tabla, String atributoEvaluado, String[] condicion, String Clave)throws RemoteException;
```

```
public Vector getAtributos(String[] atributosDeseados, Tabla tabla, String atributoEvaluado, Date[] condicion, String Clave)throws RemoteException;
```

```
    public Vector getAtributos(String[] atributosDeseados, Tabla
tabla, String atributoEvaluado, int[] condicion, String Clave)throws
RemoteException;
```

- GetInfo: Métodos para hacer consultas acerca de los valores incluidos en la base de datos.
    - Realiza operaciones algebraicas con los valores de las columnas de las tablas: Salvo con las columnas que sean de tipo *OLE*, en la base de datos o *bytes[]* en las tablas del servidor) las funciones *max* y *min* se pueden usar tanto para números como para fechas.

```
  public int getInfo(Tabla tabla, String atributoEvaluado, int
operacionDeseada, String Clave)throws RemoteException;
```

    - Devuelve el valor de un atributo de una tabla (indexada) a partir del nombre de una fila de otra tabla.

```
  public Vector getInfo(Tabla tabla, String[] atributosDeseados, Tabla
tablaIndexada, String atributoNombre, String nombre, String Clave)
throws RemoteException;
```

- ModificarTupla: Métodos para cambiar el valor de los atributos en una fila de la tabla de datos. Los atributos pueden ser de tipo entero, String, fecha o un archivo, según la columna de la base de datos que se deseen modificar. Estos cambios se realizan según uno de los atributos básicos (*Nombre*, *FechaInsercion* o identificador) de las filas.

```
    public      String      modificarTupla(Tabla      tabla,      String
atributoModificar, String valorNuevo, boolean b, int ID, String
Clave)throws RemoteException;

    public      String      modificarTupla(Tabla      tabla,      String
atributoModificar, String valorNuevo, boolean b, String Nombre, String
Clave) throws RemoteException;

    public      String      modificarTupla(Tabla      tabla,      String
atributoModificar, String valorNuevo, boolean b, Date FechaInsercion,
String Clave) throws RemoteException;

    public      String      modificarTupla(Tabla      tabla,      String
atributoModificar, int valorNuevo, int ID, String Clave) throws
RemoteException;

    public      String      modificarTupla(Tabla      tabla,      String
atributoModificar, int valorNuevo, String Nombre, String Clave) throws
RemoteException;
```

```
    public String modificarTupla(Tabla tabla, String
atributoModificar, int valorNuevo, Date FechaInsercion, String Clave)
throws RemoteException;

    public String modificarTupla(Tabla tabla, String
atributoModificar, Date valorNuevo, int ID, String Clave) throws
RemoteException;

    public String modificarTupla(Tabla tabla, String
atributoModificar, Date valorNuevo, String Nombre, String Clave)throws
RemoteException;

    public String modificarTupla(Tabla tabla, String
atributoModificar, Date valorNuevo, Date FechaInsercion, String
Clave)throws RemoteException;
```

4.2.3.- Consultas SQL. Conversión de tipos de datos Access-Java y viceversa

Las consultas se deben realizar usando lenguaje SQL. Es antes de usar éste método cuando deben adaptarse los tipos de datos que no coincidan entre Java y Access.

Así, hacemos que el archivo que contendrá *Notas* o una *Fotografía* sea un `array` de *bytes*, tipo de datos que se puede almacenar como objeto OLE en la base de datos.

Menos fácil es la conversión del tipo *date* (2001-01-01) de Java al tipo *fecha/hora* de Access (01/01/2001), ya que es necesaria la implementación de un método privado (*toAccess)* en *ServidorImpl* que transforme los datos *date* a *fecha/hora*.

A continuación, se muestran los formatos de cada sentencia según la operación que se desee realizar, y un ejemplo de una ejecución específica. Notar que para cualquier consulta al especificar información al gestor de bases de datos Access ha de tenerse en cuenta el formato de los datos que éste entiende, es decir, los campos de tipo *fecha/hora* han de introducirse entre dos caracteres almohadilla (#) y los que sean de tipo *carácter* entre comillas, no importa que sean dobles o simples. Los campos de tipo numérico no tienen una notación especial.

- AñadeTupla:

```
    "INSERT INTO "+tabla.getName()+" "+ cadenaAtributos+" VALUES
"+cadenaInterrogantes;
```

Se desea introducir en la tabla una nueva fila en la tabla Protesicos cuyo nombre sea Esther Colero García y la fecha de inserción el veinte de enero de dos mil cinco

(2005-01-20), la consulta SQL que realizará el *servidor GESPROT* tendrá la siguiente forma:

```
INSERT INTO Protesicos (FechaInsercion, Nombre ) VALUES
(20/01/2005, Colero García, Esther)
```

- EliminaTupla:

```
"DELETE FROM "+tabla.getName()+" WHERE ID"+tabla.getName()+" = ? "
```

Eliminaremos la fila anterior a partir del IDProtesicos de Esther:

```
DELETE FROM Protesicos WHERE IDProtesicos = 9;
```

A partir del Nombre:

```
DELETE FROM Protesicos WHERE Nombre = 'Chocolatero García,
Francisco';
```

A partir de la Fecha de Inserción:

```
DELETE FROM Protesicos WHERE FechaInsercion = #20/01/2005#;
```

- GetTupla/s:

  - A partir del *IDTabla.*

```
"SELECT "+cadenaAtributos+" FROM "+tabla.getName()+" WHERE
(("+tabla.getName()+"."+"ID"+tabla.getName()+")="+ID+");";
```

Supongamos que queremos saber los datos de la fila de la tabla Protesicos cuyo IDProtesicos es el 6:

```
SELECT Protesicos.FechaInsercion, Protesicos.IDProtesicos,
Protesicos.Nombre, Protesicos.NSS, Protesicos.Clave,
Protesicos.Apellidos, Protesicos.Email, Protesicos.Direccion,
Protesicos.Ciudad, Protesicos.Provincia, Protesicos.CP,
Protesicos.Pais, Protesicos.TfnoMovil, Protesicos.TfnoTrabajo,
Protesicos.FechaNacimiento, Protesicos.Salario,
Protesicos.TasaFacturacion, Protesicos.NombreContactoUrgencias,
Protesicos.TfnoContacto, Protesicos.Fotografia, Protesicos.Notas FROM
Protesicos WHERE ((Protesicos.IDProtesicos)=6);
```

  - A partir de la *FechaInserción.*

```
"SELECT "+cadenaAtributos+" FROM "+tabla.getName()+" WHERE
((("+tabla.getName()+"."+"FechaInsercion)=#"+fecha+"#));";
```

Deseamos ahora saber los datos de la/s fila/s cuya FechaInserción es 17/01/2005, la consulta que ejecutaremos será:

```
SELECT Protesicos.FechaInsercion, Protesicos.IDProtesicos,
Protesicos.Nombre, Protesicos.NSS, Protesicos.Clave,
Protesicos.Apellidos, Protesicos.Email, Protesicos.Direccion,
Protesicos.Ciudad, Protesicos.Provincia, Protesicos.CP,
Protesicos.Pais, Protesicos.TfnoMovil, Protesicos.TfnoTrabajo,
Protesicos.FechaNacimiento, Protesicos.Salario,
Protesicos.TasaFacturacion, Protesicos.NombreContactoUrgencias,
```

```
Protesicos.TfnoContacto, Protesicos.Fotografia, Protesicos.Notas FROM
Protesicos WHERE ((Protesicos.FechaInsercion)=#17/01/2005#);
```

- A partir del *nombre*.

```
"SELECT "+ cadenaAtributos+ " FROM "+tabla.getName()+" WHERE
"+tabla.getName()+".Nombre IN ('"+Nombre+"');";
```

Se desea conocer todos los campos de la/s fila/s de la tabla Protesicos cuyo nombre sea Esther Colero Sánchez, para ello se realiza la siguiente consulta a la base de datos:

```
SELECT Protesicos.FechaInsercion, Protesicos.IDProtesicos,
Protesicos.Nombre, Protesicos.NSS, Protesicos.Clave,
Protesicos.Apellidos, Protesicos.Email, Protesicos.Direccion,
Protesicos.Ciudad, Protesicos.Provincia, Protesicos.CP,
Protesicos.Pais, Protesicos.TfnoMovil, Protesicos.TfnoTrabajo,
Protesicos.FechaNacimiento, Protesicos.Salario,
Protesicos.TasaFacturacion, Protesicos.NombreContactoUrgencias,
Protesicos.TfnoContacto, Protesicos.Fotografia, Protesicos.Notas FROM
Protesicos WHERE Protesicos.Nombre IN ('Colero García, Esther');
```

- GetAtrib:

  - Si es un patrón de *caracteres*.

```
"SELECT "+ cadenaAtributos+ " FROM "+tabla.getName()+" WHERE
((("+tabla.getName()+"."+ atributoEvaluado + ") LIKE/NOT LIKE " +
condicion + "));";
```

Sabemos que el nombre de nuestra Paciente comienza por María y no sabemos mas podemos hacer una consulta especificando la parte que conocemos y pidiendo a la base de datos que devuelva los *IDs* y el *IDPacientes* de todos los que cumplan ese requisito pero que no sepamos cómo acaba. Los caracteres "comodín" son &, que representa una cadena en la posición en la que se sitúa y _, que representa un carácter en la posición ocupada.

```
SELECT Pacientes.IDPacientes, Pacientes.FechaInsercion FROM
Pacientes WHERE (((Pacientes.NOMBRE) LIKE María&));
```

  - Si es un patrón *numérico*.

```
"SELECT "+ cadenaAtributos+ " FROM "+tabla.getName()+" WHERE
((("+tabla.getName()+"."+ atributoEvaluado + ") LIKE/NOT LIKE '" +
condicion + "'));";
```

Si quisiéramos conocer el *nombre* y el *IDProductos* de aquellos productos agotados haríamos la siguiente consulta a Access:

```
SELECT Productos.IDProductos, Productos.Nombre FROM Productos
WHERE (((Productos.NumExistAlmacen) LIKE 0));
```

- Si es un patrón de tipo *fecha/hora*.

```
"SELECT "+ cadenaAtributos+ " FROM "+tabla.getName()+" WHERE
((("+tabla.getName()+"."+ atributoEvaluado + ") LIKE #" + condicion +
"#));";
```

Sabemos que un pedido fue encargado en diciembre de 2004 pero no sabemos que día y queremos conocer su *nombre* y su *IDPedidos*, la consulta que debemos hacer es la siguiente.

```
SELECT Pedidos.IDPedidos, Pedidos.Nombre FROM Pedidos WHERE
(((Pedidos.FechaInsercion) LIKE _ _/11/2004));
```

- Devuelve los atributos que tienen valores nulos o no nulos (valor nulo no es lo mismo que valor cero).

```
"SELECT "+ cadenaAtributos+ "FROM "+tabla.getName()+" WHERE
((("+tabla.getName()+"."+ atributoEvaluado + ") Is Null));";
```

Si se desea conocer cuales de todas las clínicas cuentan con dirección de correo electrónico y cuales son éstos se realiza la siguiente consulta:

```
SELECT Clinicas.IDClinicas, Clinicas.Email FROM Clinicas WHERE
(((Clinicas.Email) Is not null));
```

- Devuelven una serie de columnas de las filas de una tabla comprendidas en un intervalo concreto de los valores de un atributo especificado por el usuario.

  - Si es un intervalo de *fechas/hora*.

```
"SELECT "+ cadenaAtributos+ " FROM "+tabla.getName()+" WHERE ((("+
tabla.getName()+"."+atributoFechaEvaluado+ ") Between #"+
inicioIntervalo+ "# And #"+ finIntervalo+"#));";
```

Nos gustaría conocer el *IDVentas* y el *Nombre* de las Ventas realizadas entre las fechas: 17/01/2005 y 09/09/2009. Para ello ejecutamos la siguiente consulta:

```
SELECT Ventas.IDVentas, Ventas.Nombre, Ventas.FechaInsercion
FROM Ventas WHERE (((Ventas.FechaInsercion) Between #2005-01-17# And
#2009-09-09#));
```

  - Si es un intervalo *numérico*.

```
"SELECT "+ cadenaAtributos+ " FROM "+tabla.getName()+" WHERE ((("+
tabla.getName()+"."+atributoNumericoEvaluado+ ") Between "+
inicioIntervalo+ " And "+ finIntervalo+"));";
```

Si quisiéramos conocer el *nombre* y el *IDProductos* de los productos cuyos precios estuvieran en el intervalo de números definido entre dos valores: 16 y 25, usaríamos la siguiente sentencia:

```
SELECT Productos.IDProductos, Productos.Nombre FROM Productos
WHERE (((Productos.PrecioUnidad) Between 16 And 25));
```

- Métodos que devuelven una serie de atributos de las filas de una tabla a partir de otros que tienen un valor o valores indicados por el cliente.
    - El tipo de la columna a evaluar es *String* y solo se desea buscar un valor específico.

```
"SELECT "+ cadenaAtributos+ " FROM "+tabla.getName()+" WHERE
(("+tabla.getName()+"."+atributoEvaluado+") = "+condicion+");";
```

Para conocer las *fechas* en la que fue encargado y deberá ser entregado un pedido del que se conoce el nombre (Mandíbula Superior) se realiza la siguiente consulta SQL a Access:

```
SELECT Pedidos.FechaInsercion, Pedidos.FechaEntrega FROM Pedidos
WHERE ((Pedidos.Nombre) = 'Mandibula Inferior');
```

    - Si los valores de la columna son *numéricos* y solo se pregunta por un valor determinado.

```
"SELECT "+ cadenaAtributos+ " FROM "+tabla.getName()+" WHERE
(("+tabla.getName()+"."+atributoEvaluado+") = "+condicion+");";
```

Un Dentista nos ha llamado por teléfono a nuestro móvil para encargarnos un trabajo, no nos acordamos de su *nombre* pero sabemos que es un cliente habitual de la empresa. Para conocer quién es nuestro cliente anónimo consultamos a la base de datos de la siguiente manera:

```
SELECT Protesicos.IDProtesicos, Protesicos.Nombre,
Protesicos.FechaInsercion FROM Protesicos WHERE
((Protesicos.TfnoMovil)=655553434);
```

    - Si los valores de la columna son de *fecha/hora* y solo se pregunta por un valor determinado.

```
"SELECT "+ cadenaAtributos+ " FROM "+tabla.getName()+" WHERE (("+tabla.getName()+"."+atributoEvaluado+" )= #"+condicion+"#);";
```

Conocemos la fecha de entrega de un pedido (09/09/2009) y nos gustaría saber tanto el día n que fue encargado como su *nombre* e *IDPedidos*. La consulta es:

```
SELECT Pedidos.IDPedidos, Pedidos.Nombre, Pedidos.FechaInsercion FROM Pedidos WHERE ((Pedidos.FechaEntrega)=#09/09/2009#);
```

- Si los valores de la columna son *Strings* y se pregunta por varios valores.

```
"SELECT "+ cadenaAtributos+ " FROM "+tabla.getName()+" WHERE (("+tabla.getName()+"."+atributoEvaluado+")='"+condicion[i]+"')AND((…) AND
+tabla.getName()+"."+atributoEvaluado+")='"+condicion[NumCondiciones-1] + "');";
```

Queremos conocer los IDProtesicos y la fecha en la que fueron insertados en la tabla aquellos empleados cuyo *nombre* sea Antonio o Rita. La consulta debe ser:

```
SELECT Protesicos.FechaInsercion, Protesicos.IDProtesicos FROM Protesicos WHERE ((Protesicos.Nombre) = 'Antonio') AND (Protesicos.Nombre) = 'Rita'));
```

- Si los valores de la columna son *enteros* y se pregunta por varios valores.

```
"SELECT "+ cadenaAtributos+ " FROM "+tabla.getName()+" WHERE (("+tabla.getName()+"."+atributoEvaluado+")="+condicion[i]+")AND((…)AN D +tabla.getName()+"."+atributoEvaluado+")="+condicion[NumCondiciones-1] + ");";
```

Deseamos conocer los *nombres* y los *IDProveedores* de aquellos Proveedores cuyos teléfonos sean:

```
SELECT Proveedores.Nombre, Protesicos.IDProveedores FROM Proveedores WHERE ((Proveedores.Tfno) = 968798765) AND (Proveedores) = 967098765));
```

- Si los valores de la columna son *fechas/hora* y se pregunta por varios valores.

```
"SELECT "+ cadenaAtributos+ " FROM "+tabla.getName()+" WHERE (("+tabla.getName()+"."+atributoEvaluado+")=#"+fecha[i]+"#) AND ((…) AND tabla.getName()+"."+atributoEvaluado+")=#"+fecha[NumCondiciones-1]+"#);";
```

Queremos saber ahora los *nombres* de los Pedidos que deben entregarse los días 25 y 26 de enero de 2005, ésta vez consultamos:

```
SELECT Pedidos.Nombres FROM Pedidos WHERE ((Pedidos.FechaInsercion) =#25/01/2005#) AND (Pedidos.FechaInsercion) = =#26/01/2005#));
```

- GetInfo: Métodos para hacer consultas acerca de los valores incluidos en la base de datos.
    - Realiza operaciones algebraicas con los valores de las columnas de las tablas. Las operaciones algebraicas pueden ser valor máximo (*Max*), valor mínimo *(Min)*, suma (*Sum)*, y media *(Avg)*.

```
"SELECT Max("+tabla.getName()+"."+atributoEvaluado+")AS "+ varResultado+" FROM "+tabla.getName();
```

Queremos saber cuanto vale el producto más caro. Usamos la sentencia:

```
SELECT Max(Productos.PrecioUnidad)AS MáxDeID FROM Productos
```

- Devuelve el valor de un atributo de una tabla (indexada) a partir del nombre de una fila de otra tabla.

```
"SELECT "+cadenaAtributos+" FROM "+tabla.getName()+" INNER JOIN "+ tablaIndexada.getName() + " ON "+ tabla.getName()+".ID"+tablaIndexada.getName()+" = "+tablaIndexada.getName()+".ID"+tablaIndexada.getName()+" WHERE ((("+tabla.getName()+"."+atributoNombre+") = '"+nombre+"'));";
```

Para saber el precio total de todos los productos usados en el Pedido Mandíbula Superior se ejecuta la siguiente sentencia:

```
SELECT Productos.Nombre, Productos.PrecioUnidad FROM Pedidos INNER JOIN Productos ON Pedidos.IDProductos = Productos.IDProductos WHERE (((Pedidos.Nombre) = 'Mandibula Superior'));
```

- ModificarTupla: Métodos para cambiar el valor de los atributos en una fila de la tabla de datos. Los atributos pueden ser de tipo entero, *String*, *fecha* o un *archivo*, según la columna o columnas de la base de datos que se deseen modificar. Estos cambios se realizan según uno de los atributos básicos (*Nombre, FechaInsercion o identificador*) de las filas.

```
UPDATE "+tabla.getName()+" SET "+atributoModificar+"=? WHERE ID"+tabla.getName()+" = "+ ID+";";
```

Deseamos rellenar alguno de los campos de Esther a través de su *IDProtésicos*, para ello usamos las siguientes sentencias según el tipo de atributo que deseemos modificar.

```
UPDATE Protesicos SET ((Protesicos.Direccion) = 'Calle Bolos, 3,1ªB') WHERE ((Protesicos.IDProtesicos) = 9);

UPDATE Protesicos SET ((Protesicos.Tfno) = 968782035) WHERE ((Protesicos.IDProtesicos) = 9);

UPDATE Protesicos SET ((Protesicos.Fotografia) = 'C:/ProyectoPilar/GESPROT/imagen.jpg') WHERE ((Protesicos.IDProtesicos) = 9);

UPDATE Protesicos SET ((Protesicos.FechaNacimiento) = #16/12/1981#) WHERE ((Protesicos.IDProtesicos) = 7);
```

Estas modificaciones podrian haberse hecho a partir del *Nombre* de Esther o de su *FechaInsercion* en la base de datos, mostramos un ejemplo de la modificación de la dirección respecto a cada uno de esos atributos básicos:

```
UPDATE Protesicos SET ((Protesicos.Direccion) = 'Calle Bolos, 3,1ªB') WHERE ((Protesicos.Nombre) = 'Colero García, Esther');

UPDATE Protesicos SET ((Protesicos.Direccion) = 'Calle Bolos, 3,1ªB') WHERE ((Protesicos.FechaInsercion) = #20/01/2005#);
```

Notar que mediante la ejecución de las dos sentencias anteriores quedarán modificadas todas las filas cuyos *nombres* sean Esther Colero García en el primer caso o aquellas tuplas cuya *FechaInsercion* sea 2005-01-20.

En el siguiente capítulo se muestran algunos ejemplos de consultas al *servidor GESPROT* hechas por el cliente.

4.2.4.- Otros métodos: mostrarImagen(bytes [ ] imagen)

ServidorImpl proporciona también un método para recomponer los *arrays* de bytes guardados en las tablas, si éste ha sido construido a partir de un archivo de imagen. Sirve para mostrar las fotografías almacenadas en la base de datos.

4.3.- Acceso remoto al servidor

Para hacer que *ServidorImpl* proporcione acceso remoto a sus clientes usamos RMI. Para ello basta con hacer que la clase herede de `UnicastRemoteObject` y que cada uno de sus métodos de consulta/modificación de la base de datos sean capaces de lanzar la excepción `RemoteObject`.

La clase `UnicastRemoteObject` hace que, compilando de manera especial el servidor se generen los *Stub* y *Skeleton* necesarios en las arquitecturas distribuidas que usan RMI. El comando de compilación especial es el siguiente:

```
rmic servidorDatos.ServidorDatosImp
```

donde *servidorDatos* es el paquete al que pertenece la implementación del servidor (*ServidorDatosImp*).

4.4.-Asociación de la base de datos al ODBC

El servidor debe conocer la *url* de la base de datos y el *driver ODBC* que la manejará.

Para asociar la base de datos al *ODBC* se deben seguir los siguientes pasos (suponiendo que se trabaja con *Windows Office*):

- Lo primero es seguir la siguiente ruta para en contrar el Administrador de Datos ODBC:

Inicio/Panel de Control/Herramientas Administrativas/Origenes de Datos ODBC.

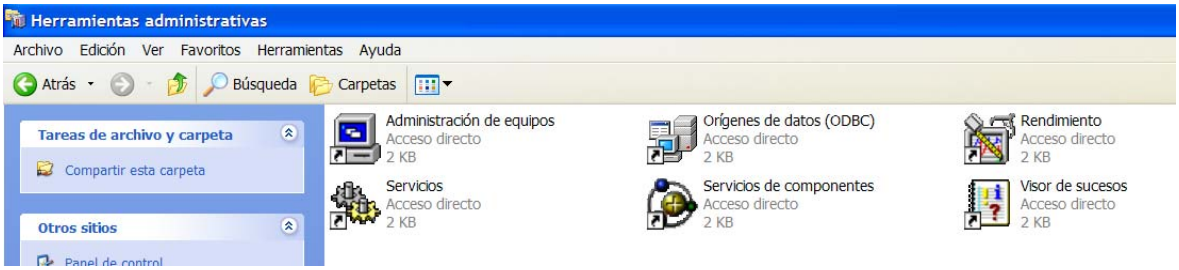


Figura 19: Acceso a Orígenes de datos (ODBC)

- Una vez dentro de éste se pulsa el botón agregar

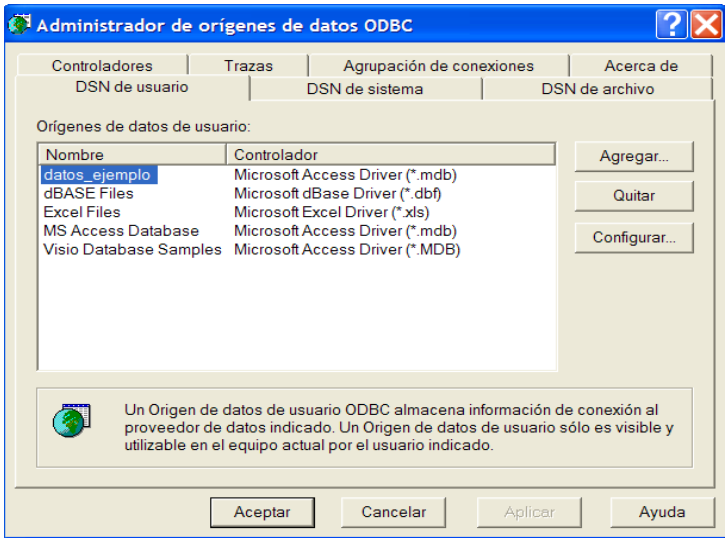


Figura 20: Administrador de orígenes de datos ODBC

- Añadir un *driver* para bases de datos con extensión *.mdb*.

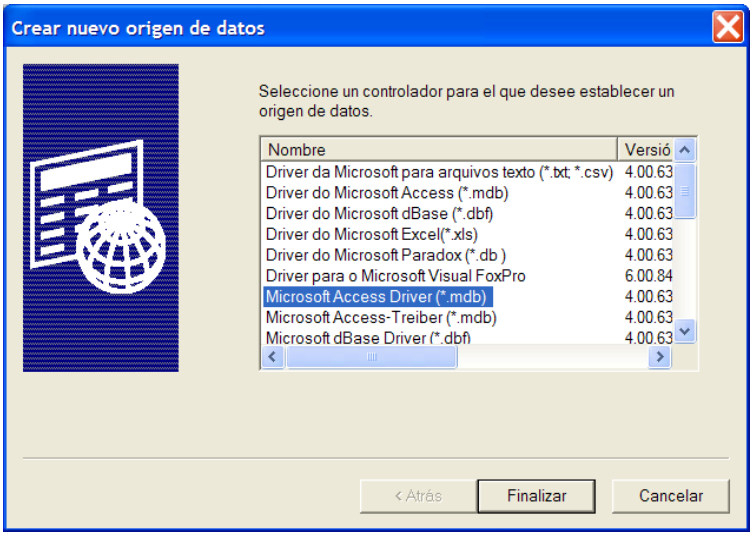


Figura 21: Crear origen de datos

- Se asigna un nombre al *driver* que usaremos

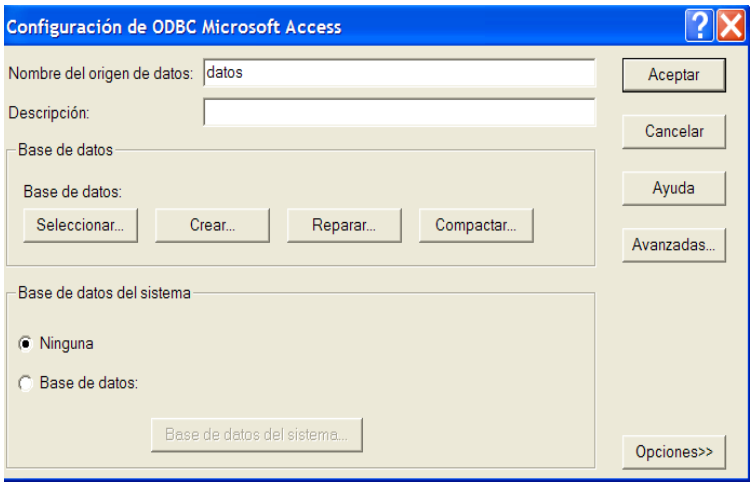


Figura 22: Asignar nombre al *driver*

- Pulsando en la opción seleccionar se indica la dirección de la base de datos

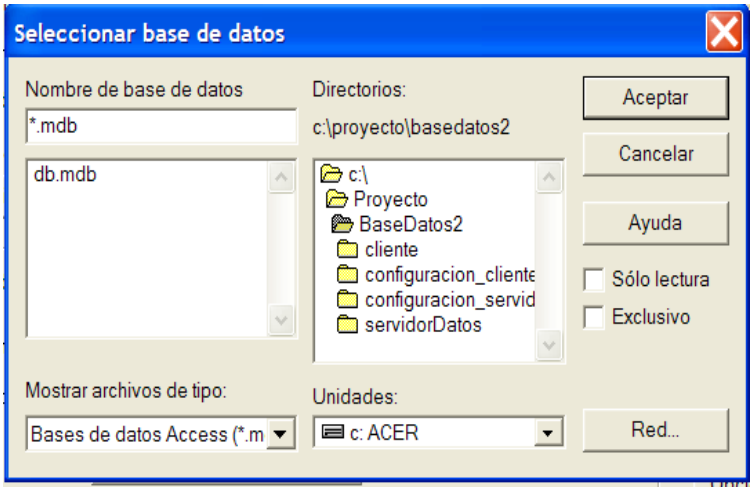


Figura 23: Seleccionar Base de Datos

- Se aceptan todos los cambios y ya tenemos el *driver ODBC* que manejará nuestra base de datos.

4.5.- Compilación y puesta en marcha del servidor

Es necesario que las clases .java que componen la aplicación estén completamente compiladas antes de arrancar el servidor. Para compilar se proporciona un archivo *compila_servidor.bat* con los siguientes datos:

```
set DIRECTORIO_JAVA=$JSDK
set CLASSPATH=%CLASSPATH%;$PATH_BASE_DATOS
%DIRECTORIO_JAVA%\bin\javac $PATH_BASE_DATOS\*.java
%DIRECTORIO_JAVA%\bin\rmic servidorDatos.ServidorDatosImp
```

En la primera línea se especifica el compilador Java que se está usando en ese momento, así como su dirección (*C:\j2sdk1.4.2_04*).

En el segundo se indica la ruta de acceso a la aplicación completa (*C:\ProyectoPilar\GESPROT*).

Las dos últimas son las que realmente compilan la aplicación: una compila las clases **.java* y la otra se usa para proporcionar el acceso remoto.

Una vez compilada la aplicación ha de usarse un segundo archivo *arranca_servidor.bat* con la siguientes líneas:

```
set DIRECTORIO_JAVA=$JSDK
set CLASSPATH=%CLASSPATH%;$PATH_BASE_DATOS
%DIRECTORIO_JAVA%\bin\java servidorDatos.Servidor
```

Donde las dos primeras líneas son idénticas a las del archivo *compila_servidor.bat*, y su función la misma. Es la tercera línea la que ejecuta la clase *Servidor* (clase que contiene el método *main()*.

Capítulo 5.- Cliente Demo

No es el objeto de éste proyecto la implementación del *Cliente de la aplicación* por tanto el *Cliente Demo* no tendrá un diseño vistoso ni explicaremos detalladamente el contenido de sus archivos, únicamente nos detendremos en la parte que tenga que ver con la interacción con el *servidor*. Notar que las posibilidades que *servidor GESPROT* ofrece son mucho más amplias que las desarrolladas, a modo de ejemplo, en el *Cliente Demo.*

El cliente Demo consta de tres archivos: *Aplicación*, *Diseño*, *y PanelFicha.* Los dos primeros se encargan de conectar con el *servidor GESPROT* y establecer el *Frame* que se mostrará al cliente y sus dimensiones. Es el último quien se encarga de insertar botones y cajas de texto a través de los que el usuario introduce información al sistema y decide qué operación desea ejecutar. Es, por tanto, *PanelFicha* el encargado de manejar eventos en el programa *Cliente Demo.*

A la aplicación le corresponde el siguiente entorno gráfico:

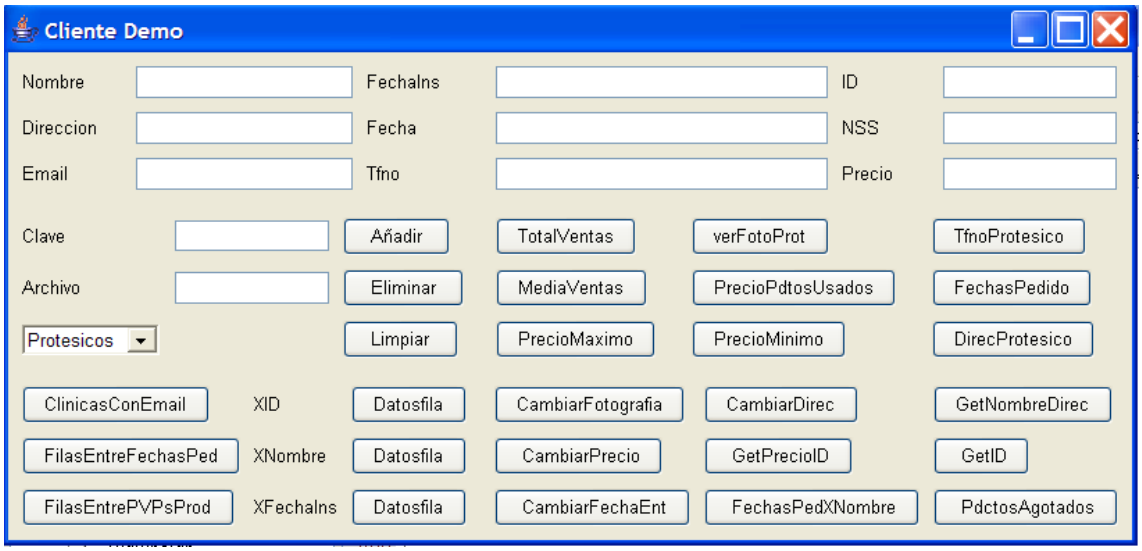


Figura 24: Entorno gráfico de Cliente Demo.

A partir de éste iremos mostrando según el tipo de consulta al servidor que proporcionen sus botones las posibilidades que ofrece la aplicación *servidor GESPROT.*

5.1.-Métodos de consulta en Cliente Demo.

Los métodos que interaccionan con el *servidor GESPROT* tienen la siguiente forma:

- Extraen los datos que se necesita que aporte el cliente a través del entorno gráfico.
- Transforman los *Strings* recibidos del frame en *enteros* (método: `int Integer.ParseInt(String ent))` o en `Date (Date date = new Date(00)),` en los casos en los que sea necesario.
- Se crea un objeto del tipo de *`Tabla`* en el que se desee hacer la consulta.
- Se introducen en un *array* todos los nombres de las columnas cuyo valor desea ser consultado o modificado.
- Se realiza la consulta a la base de datos usando el método que sea necesario.
- Si el vector que nos devuelve el *servidor* como resultado no está vacio se extraen, mediante un *bucle for* los elementos del mismo tipo que la tabla con la que hicimos la consulta. A partir de ésta extraemos los valores de las columnas que pedimos.

Todo el código puede lanzar en cualquier momento excepciones ya que se encuentra insertado dentro de un bucle *try/catch.*

Notar que todos los métodos de servidor *GESPROT* son aplicables a cualquiera de las tablas de la base de datos y que pueden insertarse, consultarse y/o modificarse atributos de cualquiera de los tipos que contiene la base de datos (*enteros, Strings, Dates, y archivos).*

5.2.-Añadir objetos en la base de datos

Se usa para insertar nuevas tuplas en la base de datos el botón *AñadirFila* el cual llamará al método: `procesarAñadir()` que hará la siguiente consulta a la base de datos:

```
Protesicos tabla = new Protesicos();


resultado = servidorDatos.añadeTupla(tabla,Clave);
```

Vemos el resultado visto por el usuario del cliente y la modificación en la base de datos:

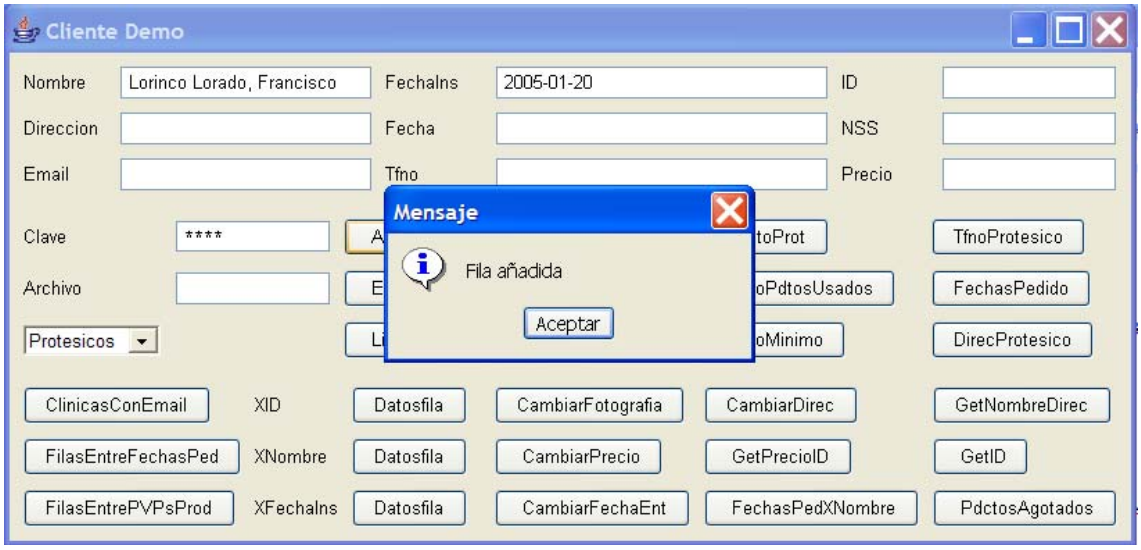


Figura 25: Inserción de una fila en la base de datos

Protesicos : Tabla

FechaInsercion	IDProtesicos	Nombre
01/01/1970	2	Cael Diente, Joaquin
01/01/1970	4	Niente Sordera, Ramon T.
26/11/2003	5	Colero Sanchez, Esther
17/01/2005	6	Chocolatero García, Francisco
20/01/2005	7	Lorinco Lorado, Francisco

Figura 26: Fila añadida en base de datos

5.3.- Extracción de datos de la base de datos

Para conocer el contenido de cada columna de las tablas de la base de datos se debe especificar tanto los nombres de campo de cada una de ellas como la tabla a la que pertenece. Existen varias formas de consulta que comentamos a continuación.

5.3.1 Extracción de todos los campos de una fila

Esta consulta puede hacerse a través del ID de la fila, de su Nombre o de su Fecha de Inserción en la tabla. Vemos el resultado de la consulta a la tabla *Protésicos* hecha a partir del nombre de la fila que deseamos conocer: Francisco Lorinco Lorado. Para ello usamos el botón *DatosFila* que a través del método *procesarBuscarFilasXNombre()* realiza la siguiente sentencia de consulta al *servidor GESPROT:*

```
Protesicos tabla = new Protesicos();
resultado = servidorDatos.getTuplas(tabla,fechaInsercion,Clave);
```

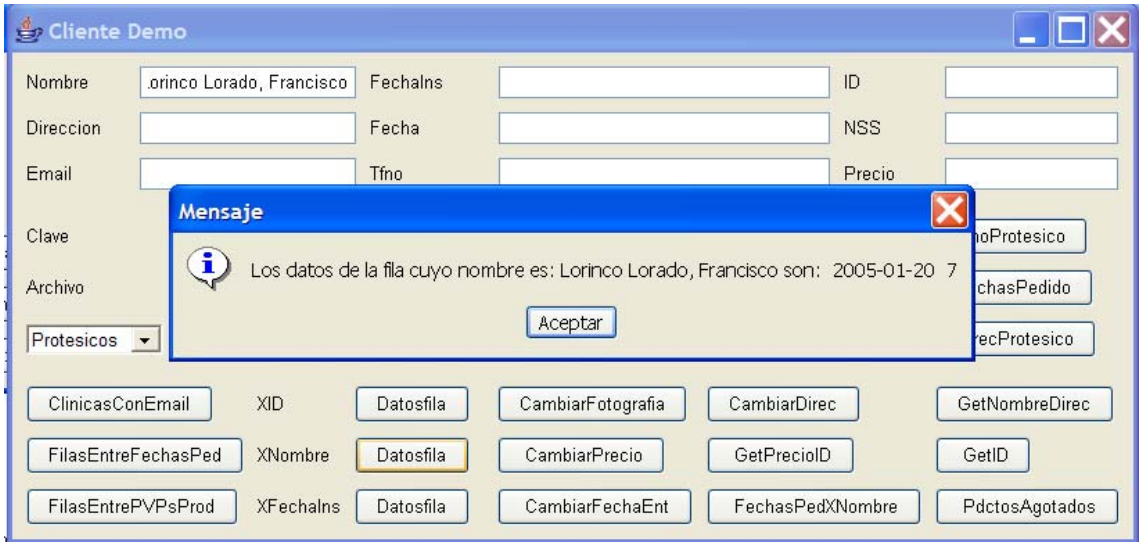


Figura 27: Consulta extracción de datos de una fila completa

5.3.2 Consultas a campos específicos: Extracción de dato o datos numéricos, de tipo Fecha/hora o Caracteres a partir del Nombre, el IDTabla o la FechaInserción

Para ello la consulta a la base de datos debe hacerse también a través del método `resultado = getAtrib(…),` usado en la forma que corresponda, como ejemplo se muestra la sentencia usada por el método `procesarBuscarPrecioIDXNombre(),` que se llama a través del botón *GetPrecioID* y que realiza la consulta en la tabla *Productos.*

```
Productos tabla = new Productos();
atributosDeseados[0] = "PrecioUnidad";
atributosDeseados[1] = "IDProductos";
res=servidorDatos.getAtributos(atributosDeseados,tabla,"Nombre",nombre,Clave);
```

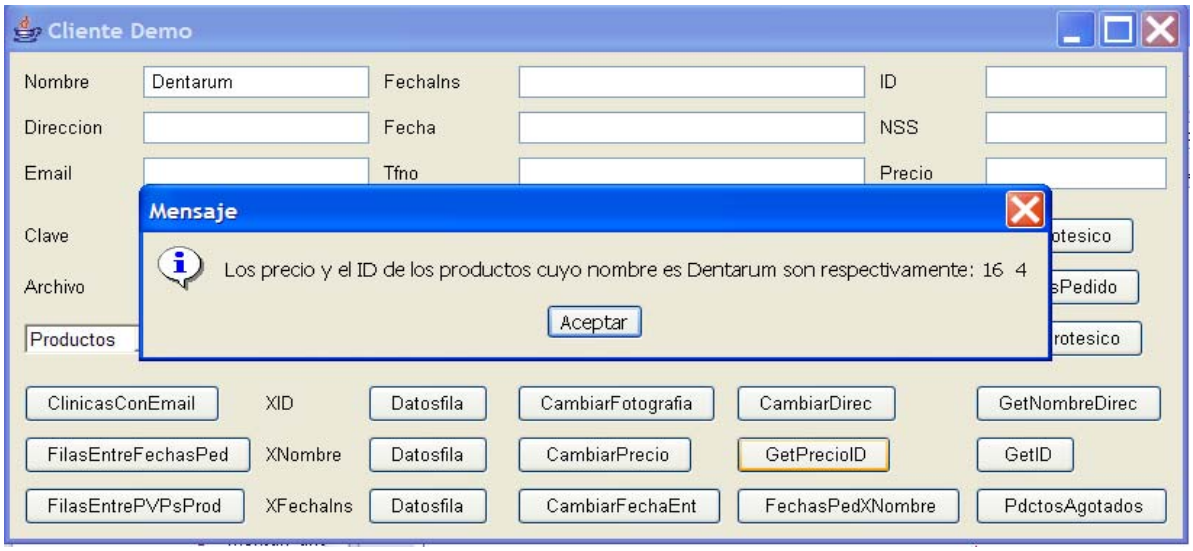


Figura 28: Consultas a campos específicos.

5.3.3.- Consultas a campos específicos de filas situadas entre intervalos específicos de números enteros o fechas.

Se usa de nuevo para ésta consulta un método getAtrib(...), que por la forma especial de su resultado mostramos en éste apartado. Esta vez usamos el botón *FilasEntreFechasPed* que llama al método `procesarBuscarFilasEntreFechas()` que realiza la consulta mediante el siguiente código:

```
Ventas tabla = new Ventas();

resultado = servidorDatos.getAtributos(atributosDeseados,tabla,
"FechaInsercion",fechaInsercion,fechaInsercion1, Clave);
```

El resultado es el que sigue:

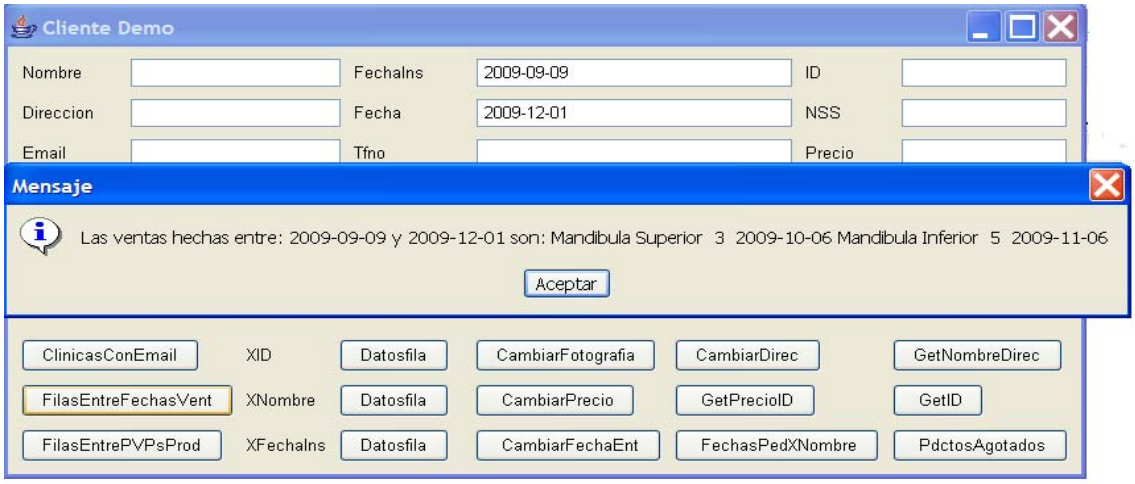


Figura 29: Consulta columnas específicas de atributos situados en un intervalo

5.3.4.- Extracción de columnas específicas de las filas en las cuales el valor de un campo está o no relleno (campo nulo o no nulo)

Nos encontramos con otro tipo especial de consulta getAtrib() en la que se estudia el contenido de un campo para ver si está o no relleno y se devuelve en consecuencia el valor de otros campos que se piden en la sentencia de consulta. Para mostrar su funcionamiento usamos el botón *ClinicasConEmail* que usa el método `procesarClinicasConEmail()` y que realiza la siguiente consulta al servidor con su correspondiente resultado:

```
Clincas tabla = new Clinicas();

resultado = servidorDatos.getAtributos(atributosDeseados, tabla,
"Email",true, Clave);
```

Figura 30: Consultas a campos nulos (o no nulos)

5.3.5.- Extracción de datos de filas en las que una de sus columnas cumplen un patrón específico

Se trata del último de los resultados proporcionados por el método getAtrib(...). Nosotros usaremos un patrón numérico para saber cuales de las filas en la tabla *Productos* tienen el valor cero en su campo "NumExistAlmacen". En el *Cliente Demo* el botón *PdctosAgotados* es el encargado de realizar de llamar al método `procesarProductosAgotadosEnAlmacen()`, que obtiene el resultado que a continuación se muestra tras realizar la siguiente consulta al *servidor GESPROT*.

```
Productos tabla = new Productos();

resultado = servidorDatos.getAtributos(atributosDeseados, tabla,
"NumExistAlmacen", 0,true, Clave);
```

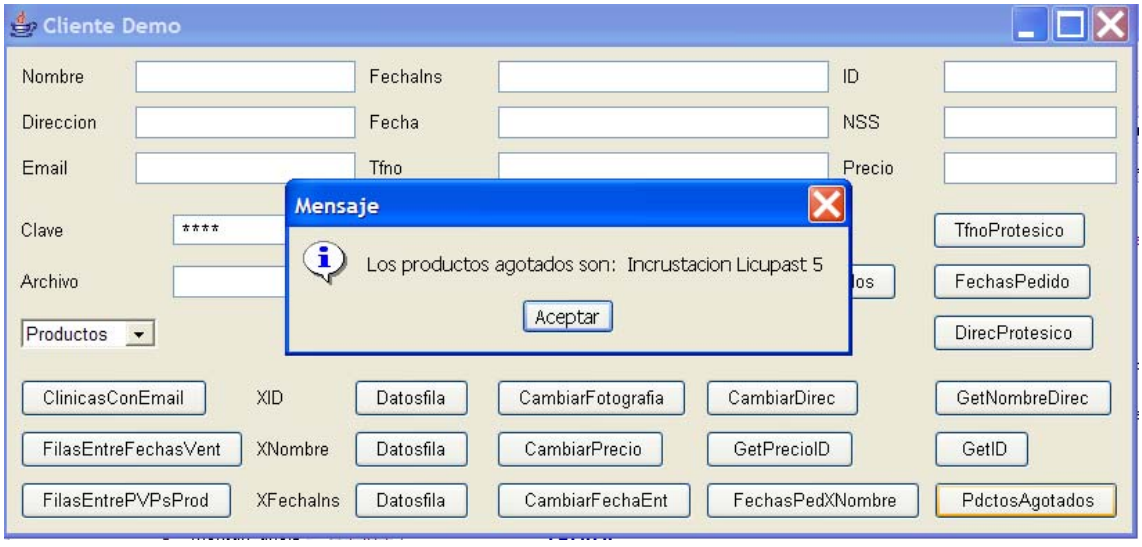


Figura 31: Extracción de columnas en cuya fila otra columna cumple un patrón

5.3.6.- Consultas de agrupación de datos a partir de una de las columnas de una tabla de la Base de Datos

Existe un tipo de método entre los que proporciona el *servidor GESPROT* con el cual podemos calcular tanto la suma, el valor mínimo, máximo o la media de los valores de todos los campos de una columna en una tabla de la base de datos. Éste método es getInfo(...), del cual mostramos un ejemplo en el *Cliente Demo* en el cual a partir del botón *TotalVentas* el método `procesarTotalVenta()`, ejecuta la consulta al servidor:

```
Ventas tabla = new Ventas();
total = servidorDatos.getInfo(tabla,atributoDeseado,3,Clave);
```

Cuyo resultado es:

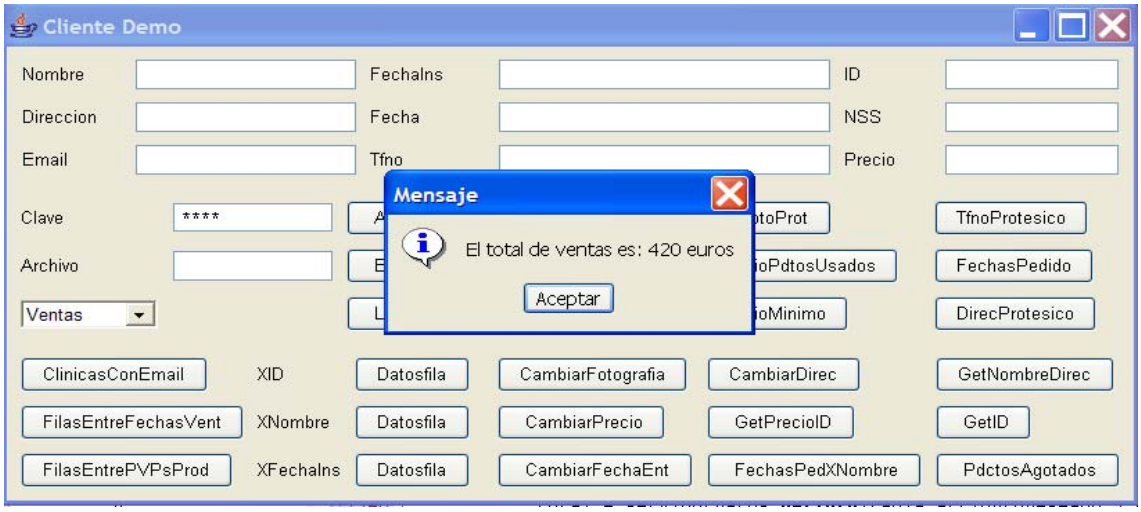


Figura 32: Consultas de agrupación de datos

5.3.7.- Consultas de datos mediante tablas enlazadas

La otra utilidad del método getInfo(...) de la base de datos es la de consultar columnas de una tabla a través del valor de un campo *Nombre* de otra tabla con la que mantiene relación de indesado a través de un campo con nombre y valor comunes. Para mostrar cómo funciona ésta compleja consulta se proporciona el método `procesarPrecioUnidadesPdtoUsadasEnPedidos()`, al que se llama usando el botón *PrecioPdtosUsados* y que realiza la consulta a la tabla del siguiente modo:

```
Pedidos tabla = new Pedidos();
Vector resultado = servidorDatos.getInfo(tabla, atributosDeseados,
tablaIndexada, "Nombre",nombre, Clave );
```

Donde *tabla* es aquella cuyo valor de campo *Nombre* debe contener un determinado valor y *tablaIndexada* es la tabla cuyos atributos se desea conocer.

El resultado de la consulta es el que sigue:

Figura 33: Consultas a campos de tablas indexadas

5.4.- Modificación de datos en la base de datos

Las modificaciones dentro de la base de datos pueden hacerse a partir de uno de los tres atributos básicos de cada tabla (*FechaInsercion*, *Nombre o IDTabla)*. Notar que los cambios pueden realizarse en cualquiera de los campos de la base de datos, ya sea su contenido de tipo *numérico*, *carácter*, *fecha/hora u Objeto OLE*. En éste último basta con proporcionar la ruta del nuevo archivo a insertar.

Como ejemplo de una modificación en la base de datos proporcionamos varios botones de los cuales usaremos *CambiarDireccion* mediante el cual llegaremos al método `procesarCambiarDireccion()` que se encargará de hacer la siguiente llamada a uno de los métodos *modificarTupla(...)* que proporciona *servidorGESPROT*.

```
Protesicos tabla = new Protesicos();
s = servidorDatos.modificarTupla (tabla,"Direccion", direccion, true, id, Clave);
```

Insertaremos la dirección *Avda. de la Esperanza*, *número 9*, *piso tercero*, *puerta A*, al Protésico *Francisco Lorinco Lorado* usando su IDProtésicos, como se ve a continuación:

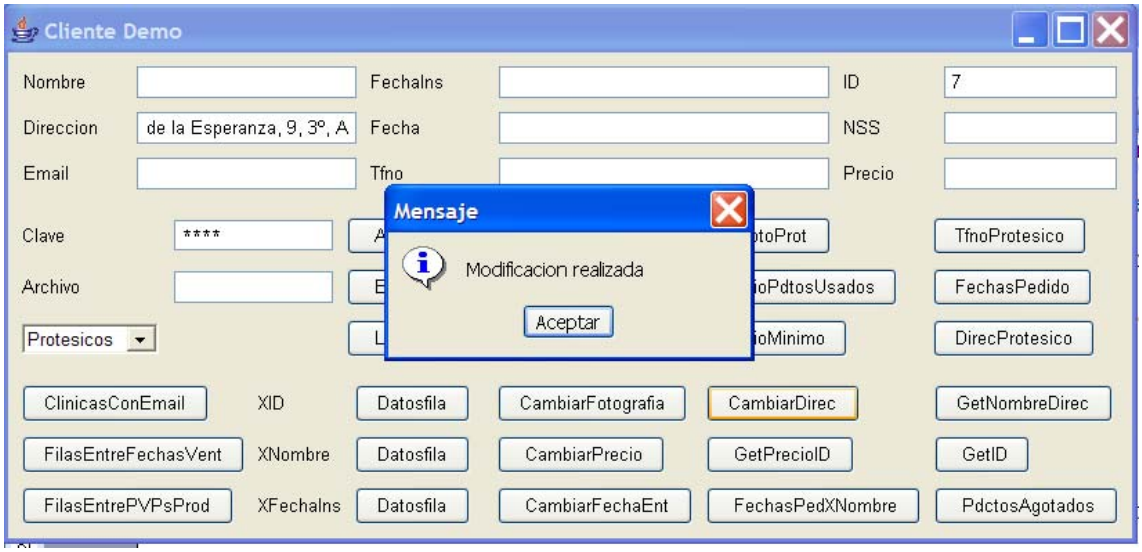


Figura 34: Modificación de atributos en la base de datos

5.5.- Eliminación de filas de la base de datos

Por último mostrar un ejemplo de la eliminación de filas en la base de datos. Las tuplas pueden desaparecer en función de uno de los tres atributos básicos de una fila en la base de datos (*Nombre*, *FechaInsercion o IDTabla*). Para proporcionar un ejemplo eliminaremos de nuestra base de datos al protésico Francisco con el que hemos estado trabajando hasta ahora a partir de su *IDProtésicos*, usaremos, por tanto, el botón *Eliminar* que llama al método `EliminarFilaXID()`, cuya interacción con el servidor tiene la siguiente forma:

```
Protesicos tabla = new Protesicos();
resultado = servidorDatos.eliminaTupla(tabla,id,Clave);
```

El resultado tanto en el entorno gráfico como en la base de datos es el siguiente:

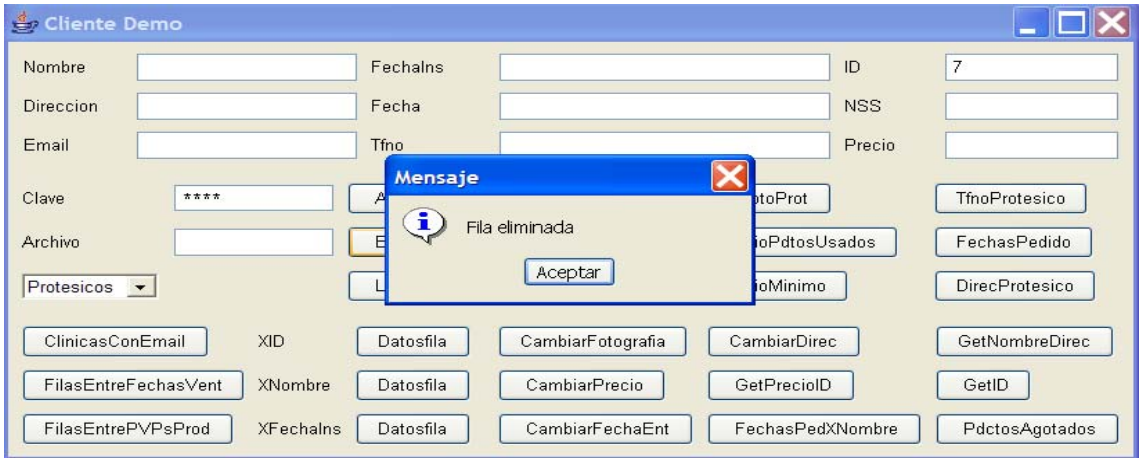


Figura 35: Eliminación de una fila en la base de datos

Protesicos : Tabla

FechaInsercion	IDProtesicos	Nombre
01/01/1970	2	Cael Diente, Joaquin
01/01/1970	4	Niente Sordera, Ramon T.
26/11/2003	5	Colero Sanchez, Esther
17/01/2005	6	Chocolatero García, Francisco
#Eliminado	#Eliminado	#Eliminado

Figura 36: Fila eliminada en la base de datos

Capítulo 6.- Conclusiones

El diseño de una base de datos para una laboratorio comprende la realización de un modelo E-R en el que se identifiquen claramente, tanto cuales van a ser los datos que deben almacenarse y consultar para la correcta gestión del laboratorio, como las relaciones entre esos datos para evitar así el almacenamiento de información redundante.

Es importante separar los datos concernientes a la empresa en cuatro grandes grupos: *Inventario*, *Contabilidad*, *Clientes y Empleados*. Y a partir de éstos deducir cuales serán las tablas, cuantos atributos almacenarán, cuales serán las tablas relacionadas y mediante qué atributos serán indexadas.

Debido a la naturaleza de la base de datos en la que los datos se dividen en tablas (Clases) y éstas a su vez se componen de filas (Objetos de las clases) el mejor lenguaje de programación que puede usarse para implementar el servidor de la base de datos es uno orientado a objetos, en concreto *Java* dadas sus características funcionales de *herencia, composición/agregación y polimorfismo.*

Es conveniente, en consecuencia, el uso de *RMI* para dotar al programa de acceso remoto, ya que es un tipo de *arquitectura distribuida* fácilmente implementable en *Java.*

El uso de lenguaje SQL para el acceso a la base de datos hace del programa *servidor GESPROT* fácilmente adaptable a otro tipo de base de datos que no sea necesariamente la proporcionada por Microsoft.

Notar además que *servidor GESPROT* es fácilmente adaptable a cualquier base de datos, basta con modificar el paquete *tablas* adaptándolo a las que contenga la nueva base de datos.

Servidor GESPROT proporciona además acceso a cualquier campo de la base de datos que sostenga, ya sea éste de tipo *numérico*, *carácter*, *fecha/hora u Objeto OLE*, siempre y cuando el cliente tenga permiso para realizar ese acceso.

Las consultas/modificaciones que proporciona la base de datos se resumen en los siguientes tipos: *Añadir filas, consultas a filas completas o atributos específicos, consultas de agrupación de datos, consultas a tablas indexadas, consultas a filas en intervalos numéricos o de fecha/hora, modificación de campos específicos y eliminación de filas.*

Por último reseñar las líneas futuras que pueden seguirse a partir de éste diseño:

- Cliente Web o cliente que realice facturas, inventario, y demás operaciones de gestión usando los datos de la base de datos.
- Introducción de seguridad con SSL.
- Posibilidad de crear nuevas tablas.

Bibliografía

Para el desarrollo de éste proyecto se ha utilizado la siguiente bibliografía:

- William Grosso, "*Java RMI*," Ed. O'Reilly. 1-56592-452-5. 01/2002.
- Agustin Froufe, JavaServer Pages. "*Manual de usuario y tutorial*," Ed. RA-MA. 84-7897-490-36. 11/2001.
- Deitel & Deitel," *Cómo programar en Java*," Ed. Prentice-Hall. 970-17-0044-9. 1998
- R. Elmasri, S.B. Navathe, "*Fundamentos de Sistemas de Bases de Datos*," Ed. Addison-Wesley. 84-7829-051-6. 2002
- http://java.sun.com/j2se/1.4.2/docs/api/