Apuntes

de

Java

Tema 12:
JDBC

Uploaded by

Ingteleco

http://ingteleco.webcindario.com
ingtelecoweb@hotmail.com

TEMA 12: JDBC

12.1.- INTRODUCCIÓN

Actualmente, el desarrollo de cualquier aplicación de cierta importancia, se programe en el lenguaje en el que se programe, requiere de manera imprescindible el acceso a bases de datos. Los programas en Java no son una excepción y para ello JavaSoft junto con la colaboración de los fabricantes de bases de datos más importantes (Sybase, Informix, Oracle e IBM entre otros) han desarrollado lo que se conoce como JDBC.

La mayoría de las bases de datos existentes en el mercado tienen muy pocas cosas en común a excepción del lenguaje de consultas SQL. Cada base de datos tiene su propio API particular que debe ser aprendido por el programador de aplicaciones para poder acceder a ellas. Esto implica que, además de la dificultad con la que se encuentra el programador a la hora de acceder a una base de datos desconocida, el código de las aplicaciones que acceden a bases de datos será dependiente de la base de datos usada y si ésta cambia, el código de la aplicación deberá ser modificado.

Para solucionar este problema surge el driver ODBC de Microsoft que proporciona un API estándar para el acceso a cualquier base de datos. Este API permite el acceso a cualquier tipo de base de datos de una manera estándar, siendo el propio driver quien se encarga, de forma transparente al programador, de tratar las particularidades de cada tipo de base de datos. Así, un programador podrá desarrollar una aplicación que acceda a una base de datos de un determinado tipo de la misma forma que lo haría si la base de datos fuese de otro tipo diferente. Con el driver ODBC las aplicaciones dejan de ser dependientes de la base de datos a la que acceden, se podrá cambiar la base de datos sin necesidad de tener que modificar una sola línea de código.

Sin embargo, el driver ODBC presenta un problema: está limitado a una única plataforma (la plataforma Windows de su fabricante).

Este problema se soluciona con el API JDBC de Java, cuyo funcionamiento se va a tratar en este tema. El API JDBC, gracias a la tecnología Java, proporciona a las aplicaciones Java un mecanismo estándar e independiente de la plataforma para el acceso a la mayoría de las bases de datos existentes. El API JDBC define un conjunto de clases e interfaces que proporcionan toda la funcionalidad que el acceso a bases de datos requiere, tal como la ejecución de consultas SQL o el tratamiento de los resultados. Cada fabricante de base de datos se encargará de proporcionar un driver JDBC específico para su base de datos. Este driver estará constituido por un conjunto de clases que implementarán, acorde a las particularidades de la base de datos para la que está destinado, las interfaces definidas en el API JDBC.

En resumen...

- JDBC es un conjunto de clases e interfaces Java que permiten la ejecución de sentencias SQL contra una base de datos.
- JDBC permite manipular cualquier base de datos SQL. No es necesario hacer un programa para manipular Oracle, otro para Sybase, etc... Un mismo programa puede manipular cualquier base de datos.
- Uniendo JDBC con la tecnología Java tenemos programas que se pueden ejecutar en cualquier plataforma y que pueden manipular cualquier base de datos.

La figura 12.1 muestra distintas configuraciones que se pueden adoptar con el uso de JDBC. Como se puede observar el acceso a base de datos a través de JDBC requiere siembre del uso de un driver JDBC específico para la base de datos que vayamos a usar.

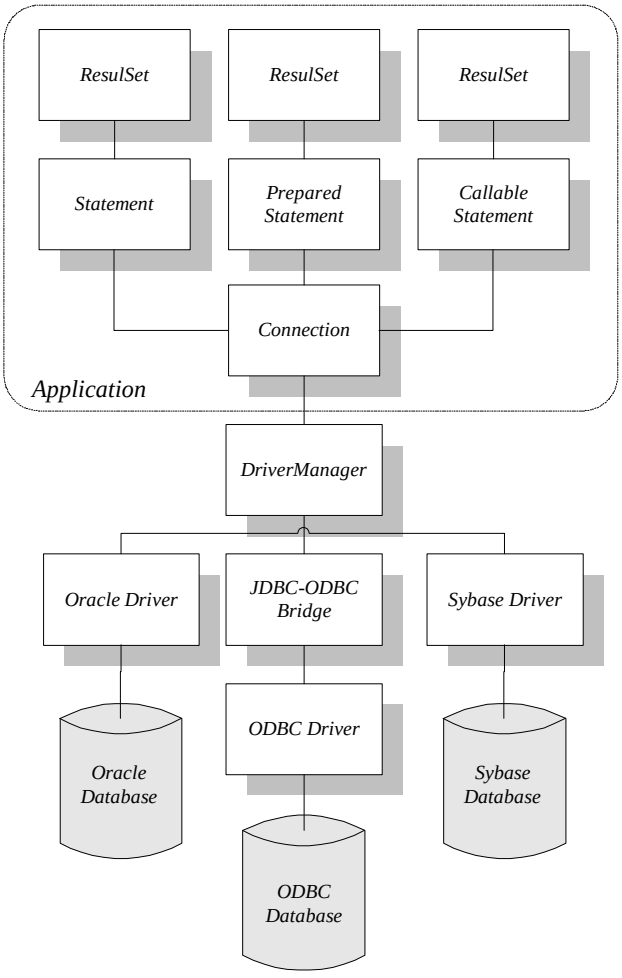

Figura 12.1: Posibles configuraciones JDBC

El objetivo de esta asignatura no es conocer los detalles de cada una de las configuraciones, sino aprender a programar una aplicación que acceda a una base de datos a través de alguna de ellas. Nosotros, en la asignatura, optaremos por la que hace uso del driver JDBC-ODBC Bridge (la más común y sencilla de utilizar), mostrada en la rama central de la figura. Este driver, como su propio nombre indica, actúa de puente entre el API JDBC de Java y el driver ODBC de Microsoft. La función de este driver será transformar todas las llamadas que la

aplicación realice sobre el API JDBC a llamadas sobre el driver ODBC, de tal forma que finalmente quien realice el acceso a la base de datos sea el propio driver ODBC. Este mecanismo nos permitirá el acceso mediante JDBC a cualquier base de datos que sea soportada por el driver ODBC.

12.2.- CREACIÓN DE UNA BASE DE DATOS ACCESS

Antes de acceder a una base de datos desde un programa, es obvio que lo primero que habrá que hacer es crear la base de datos con la que vamos a trabajar. Existen múltiples SGBD que nos permiten crear y administrar bases de datos, nosotros vamos a emplear, posiblemente, el más sencillito de todos que es Microsoft Access.

A continuación se irán mostrando, mediante un ejemplo, los pasos a seguir para crear una base de datos en Access. En nuestro ejemplo crearemos una base de datos de alumnos compuesta por una única tabla donde se almacenarán todos los datos de los alumnos.

- Abrir Microsoft Access

  Para ello, seleccionar en los programas del menú de inicio de Windows, dentro de las herramientas de Microsoft Office, el icono etiquetado como Microsoft Access.

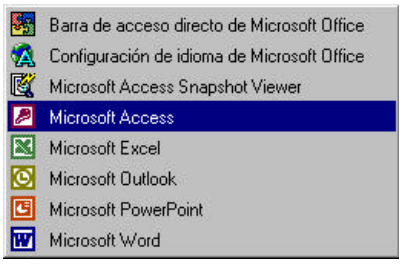

Figura 12.2: Icono de acceso a Microsoft Access

- Crear una base de datos en blanco

  Se presentará un dialogo donde seleccionaremos la opción "Base de datos de Access en blanco " y pulsaremos el botón Aceptar. (Ver la figura 12.3)

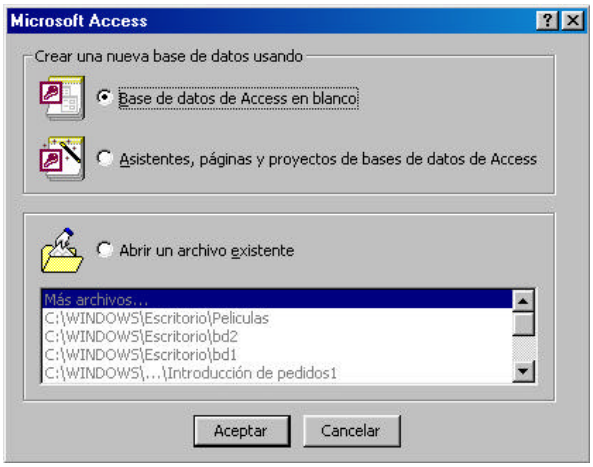

Figura 12.3: Base de datos de Access en blanco

A continuación se nos mostrará otro diálogo donde deberemos indicar dónde queremos guardar la base de datos que vamos a crear y con qué nombre. La guardaremos, por ejemplo, en el directorio c:\Alumno\LabInfII y con el nombre AlumnosBD. (Ver figura 12.4)

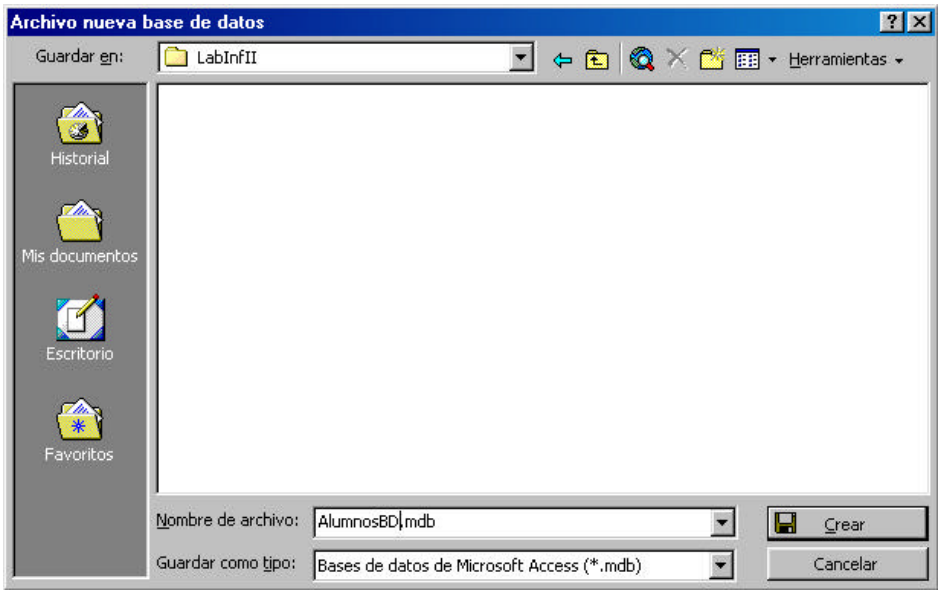

Figura 12.4: Guardar la base de datos con un nombre

- Crear una tabla

Se nos mostrará un diálogo donde podremos crear tablas, consultas, formularios, etc. A nosotros solo nos será útil para la creación de tablas. Hacemos doble clic sobre el elemento etiquetado como "Crear una tabla en vista de diseño ". (Ver figura 12.5)

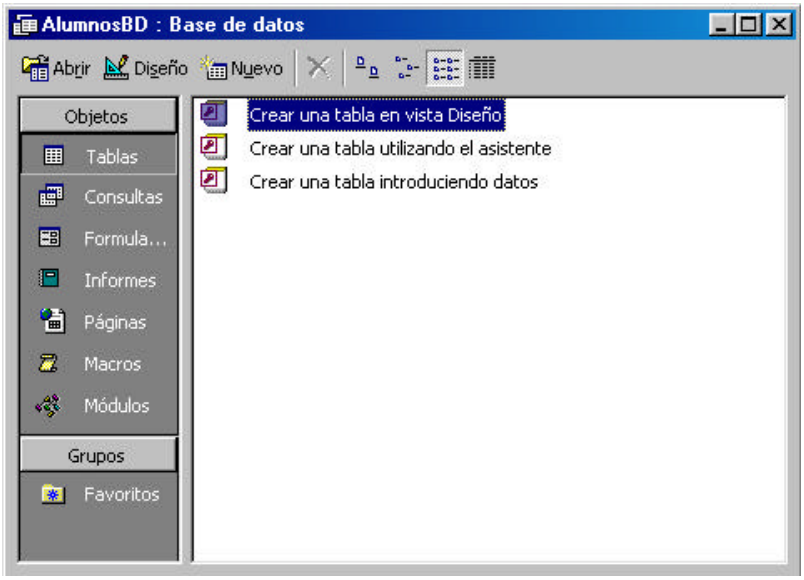

Figura 12.5: Creación de una tabla en vista de diseño

A continuación se nos mostrará un formulario donde podremos deberemos introducir los campos (columnas) de nuestra tabla y el tipo de datos de cada campo. Este formulario se muestra en la figura 12.6.

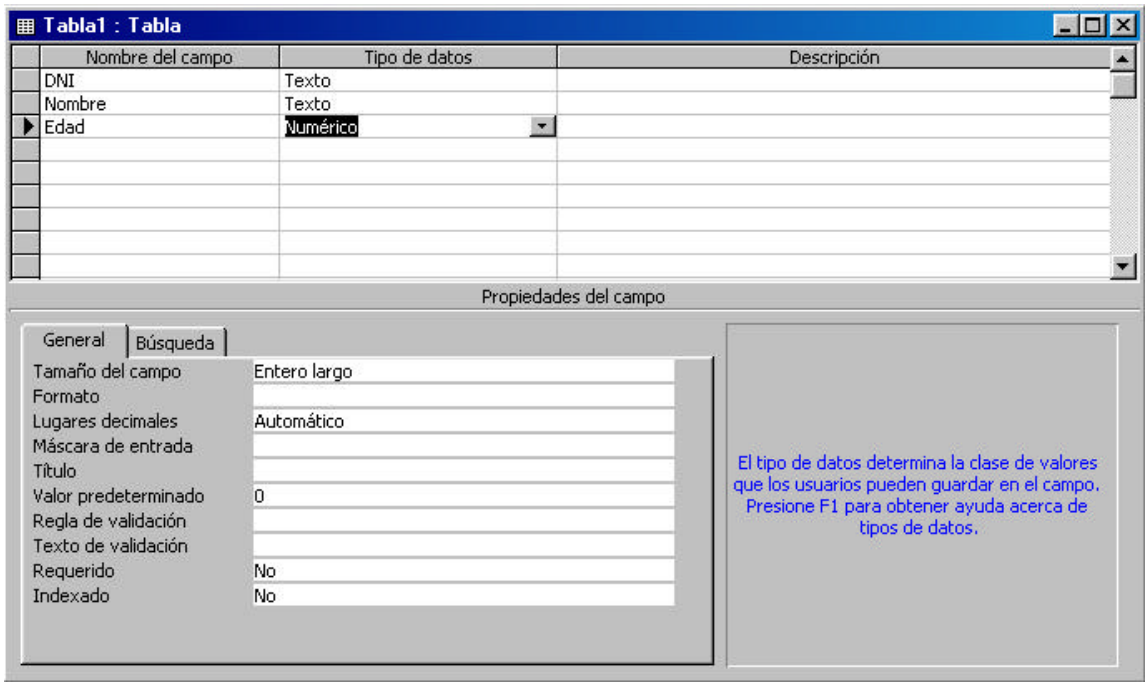

Figura 12.6: Edición de los campos de una tabla

Como ya hemos dicho nuestra base de datos va contener una sola tabla de alumnos con la siguiente estructura:

CAMPO	TIPO DE DATOS
DNI	Texto
Nombre	Texto
Edad	Numérico

Una vez que hemos introducido los campos de nuestra tabla, pulsamos sobre el icono "Guardar" y nos aparece un diálogo donde introduciremos el nombre que le queremos dar a la tabla que hemos creado. A la tabla la llamaremos "Alumnos" y pulsaremos el botón Aceptar. (Ver figura 12.7)

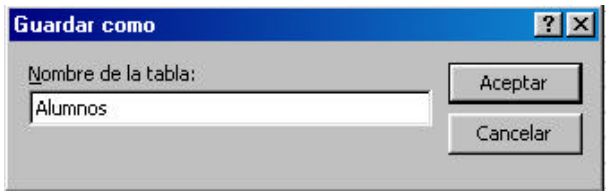

Figura 12.7: Asignación de un nombre a la tabla

Se nos presentará un mensaje dónde se nos advertirá que no hemos definido ninguna clave principal para esa tabla y Access no preguntará si queremos que él nos cree una (ver figura 12.8). Para el uso que nosotros vamos a hacer de la tabla no es necesario que tengamos definida ninguna clave primaria para la tabla y por tanto responderemos "No". No obstante y si quisiésemos, podríamos haber definido alguno de los campos de nuestra tabla como clave primaria (por ejemplo el DNI).

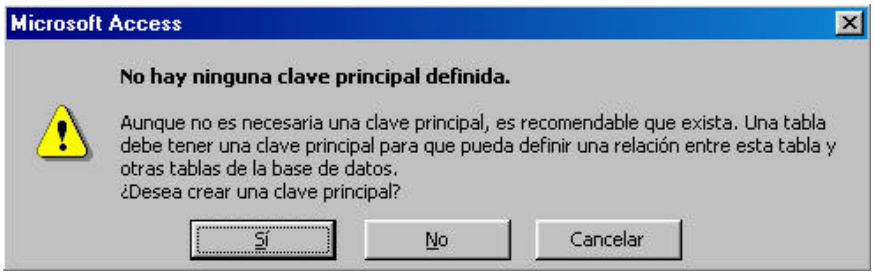

Figura 12.8: Mensaje de clase primaria no definida

En estos momento ya tenemos creada la tabla "Alumnos" en nuestra base de datos. Cerramos el formulario donde hemos introducido los campos de la tabla y vemos como se ha creado un icono llamado "Alumnos" que representa a nuestra tabla. (Ver figura 12.9)

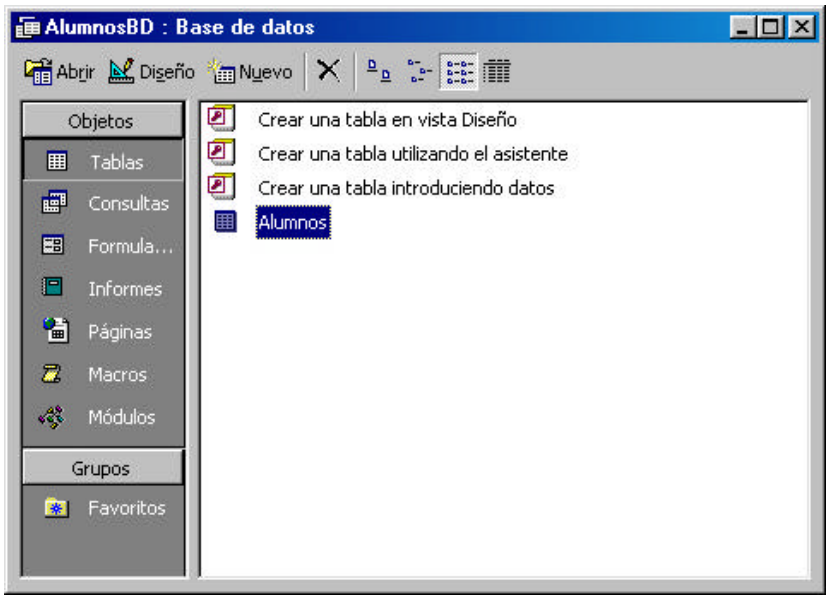

Figura 12.9: Tabla creada con éxito

- Cerrar Microsoft Access

  El trabajo ha terminado. La base de datos "AlumnosBD" ha sido creada con una tabla llamada "Alumnos". Cerramos Access y ya podemos hacer programas en Java que accedan a nuestra base de datos.

12.3.- REGISTRO DE LA BASE DE DATOS EN ODBC

Cómo ya hemos indicado en la introducción, nosotros vamos a usar la configuración de JDBC que se apoya sobre el driver ODBC de Microsoft. Para poder acceder mediante esta configuración a nuestra base de datos de alumnos, es requisito imprescindible registrar ésta en el driver ODBC con un determinado nombre. Más adelante podremos comprobar cómo el nombre con el que registremos nuestra base de datos en ODBC es el nombre con el que nos vamos a referir a ella en nuestros programas.

A continuación se muestran los pasos a seguir para el registro de una base de datos en ODBC.

- Acceder a la configuración del driver ODBC

  Para ello accederemos al Panel de Control de Windows y seleccionaremos el icono ODBC. (Ver figura 12.10)

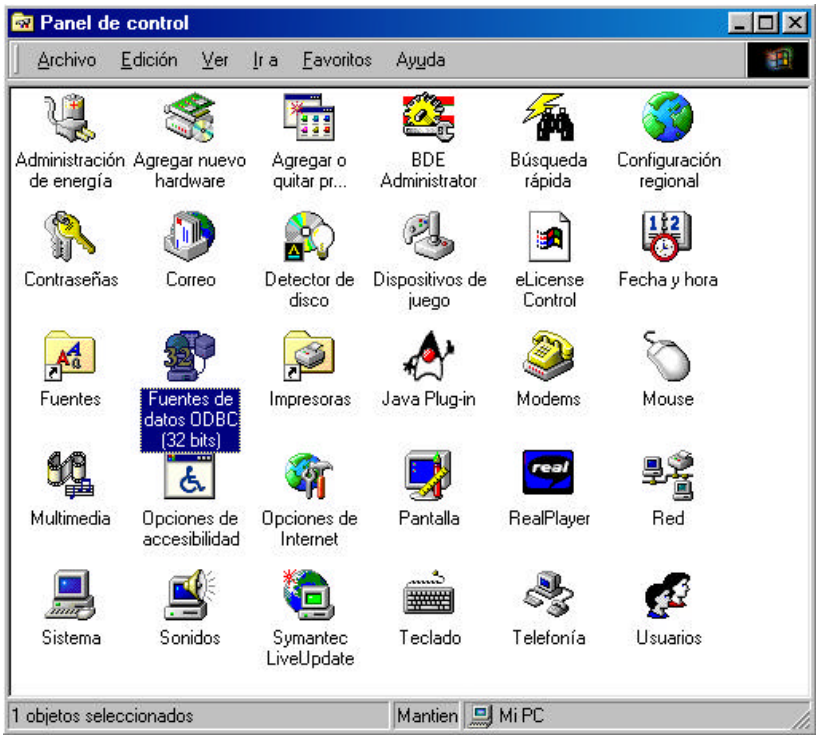

Figura 12.10: Acceso a la configuración del driver ODBC

- Registro de la base de datos en ODBC

  Se nos presentará un dialogo con múltiples pestañas. Seleccionamos la pestaña "DSN de Sistema " y pulsamos el botón Agregar. (Ver figura 12.11)

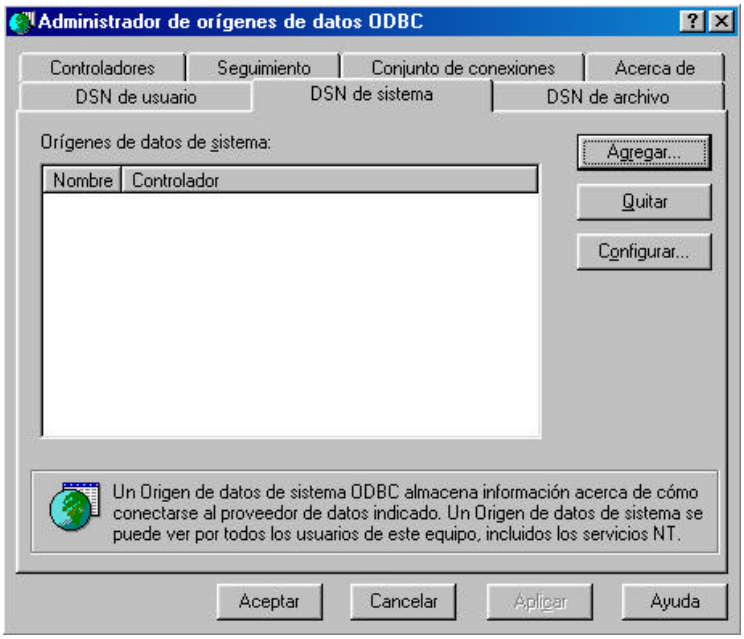

Figura 12.11: Acceso a la pestaña DSN de sistema

A continuación se muestra una pantalla con todos los drivers disponibles. Seleccionamos el Driver para Microsoft Access y pulsamos el botón Finalizar. (Ver figura 12.12)

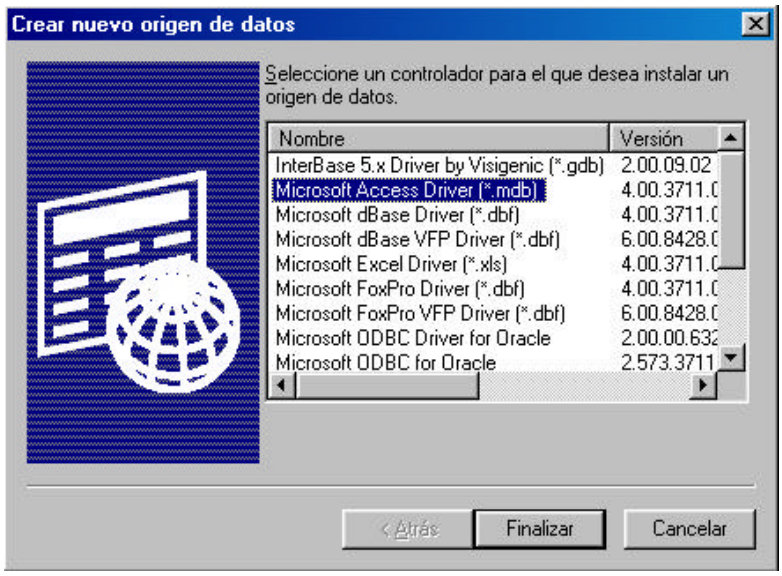

Figura 12.12: Selección del driver Microsoft Access Driver.

A continuación se nos presentará una pantalla donde registrar y configurar nuestra base de datos (ver figura 12.13). En el campo "Nombre del origen de datos" escribiremos el nombre con el que queremos registrar nuestra base de datos. Este será el nombre con el que después identificaremos la base de datos en nuestros programas, en este caso la registraremos, por ejemplo, con el nombre "AlumnosBD".

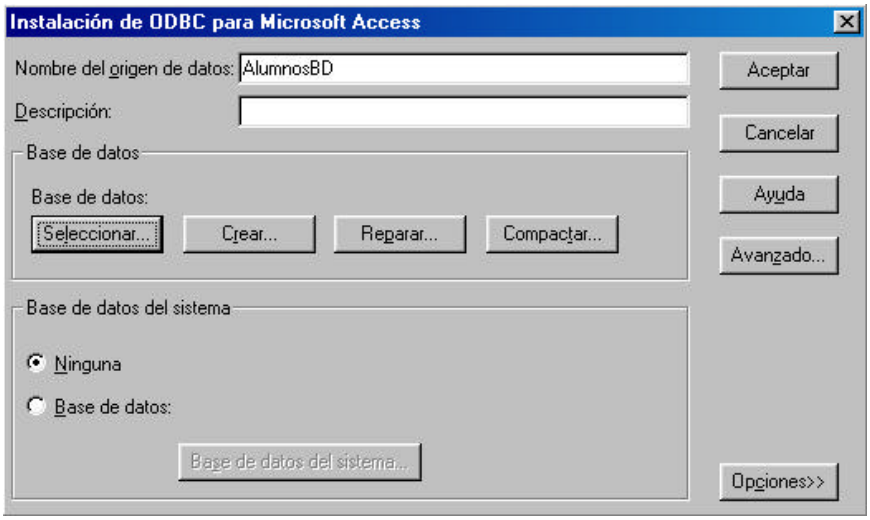

Figura 12.13: Asignación de un nombre a la base de datos en ODBC

Después, pulsaremos el botón "Seleccionar..." y seleccionaremos el fichero donde hemos guardado nuestra base de datos (si recordamos el apartado 12.2, este fichero era C:\Alumno\LabInfII\AlumnosBD.mdb). Se muestra en la figura 12.14.

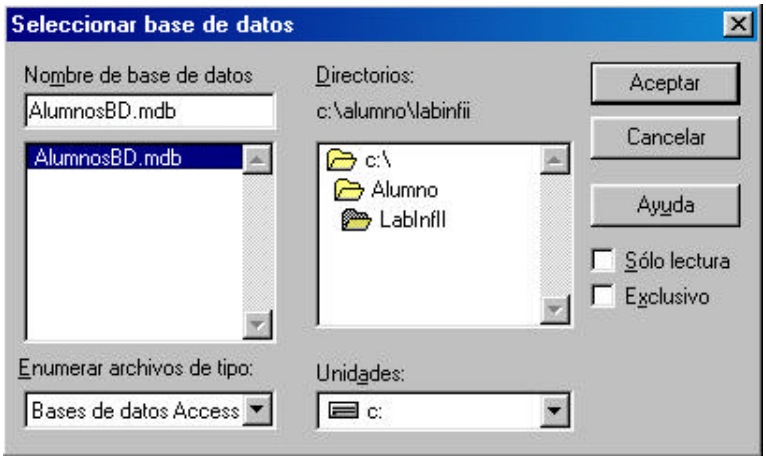

Figura 12.14: Selección del fichero que contiene la base de datos

Una vez seleccionada la base de datos, se nos visualiza la pantalla anterior (figura 12.15) donde ya, en la sección "Base de Datos ", aparece reflejada la ubicación de nuestra base de datos. Pulsaremos el botón Aceptar.

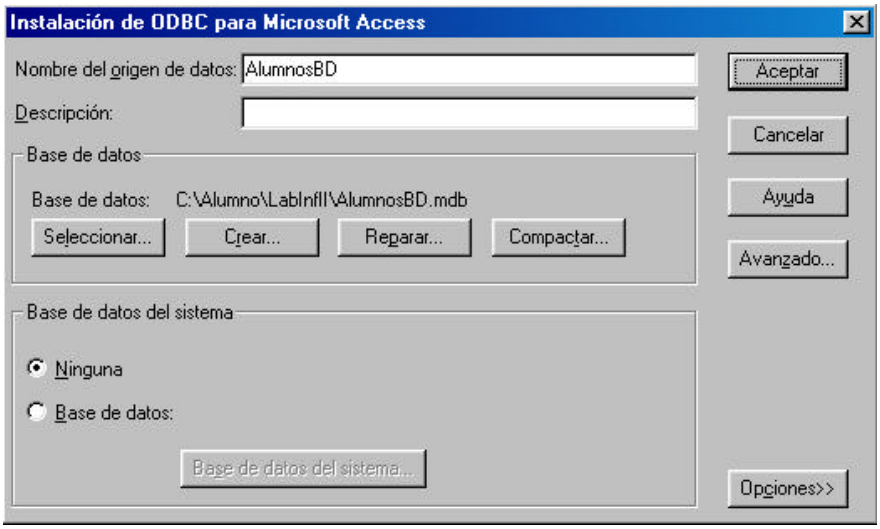

Figura 12.15: Registro de la base de datos en ODBC

Nuestra base de datos ya ha sido registrada en ODBC. Se nos mostrará una pantalla (figura 12.16) donde aparecerá un icono con el nombre que le hemos dado a nuestra base de datos (AlumnosBD), confirmándonos que la base de datos ha sido registrada correctamente. Para terminar pulsaremos el botón Aceptar.

Es importante destacar que el nombre que le damos a la base de datos al registrarla en ODBC no tiene porqué coincidir con el nombre del fichero que contiene la base de datos. Podemos registrarla con el nombre que queramos, éste es el nombre que usaremos en nuestros programas para referirnos a esa base de datos.

Así, ODBC dado el nombre de una fuente de datos (AlumnosBD) sabe exactamente dónde está ubicada esa base de datos (en C:\Alumno\LabInfII\AlumnosBD.mdb) y que tipo de base de datos es (una base de datos Access), conociendo de esta forma qué driver tiene que usar para poder manejarla.

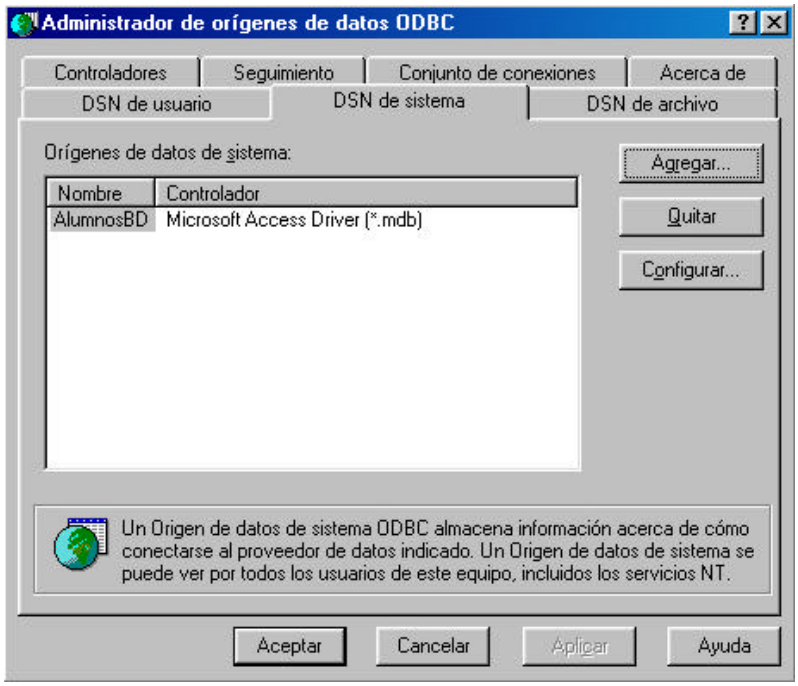

Figura 12.16: Base de datos AlumnosBD registrada en ODBC

12.4.- ACCESO A BASES DE DATOS MEDIANTE JDBC

Ha llegado el momento de explicar cómo se realiza el acceso a una base de datos mediante JDBC. Para ello tomaremos como ejemplo un sencillo programa que se encarga de realizar una consulta SQL sobre la base de datos de alumnos que hemos creado anteriormente y de mostrar los resultados de ésta por pantalla. Recordemos que la base de datos de alumnos tenía tres campos: el DNI (texto), el nombre (texto) y la edad (numérico) del alumno. Este es el código del programa:

```
import java.sql.*; // paquete donde están las clases necesarias para JDBC

public class JDBCSample{

     public static void main(java.lang.String[] args){

          try{
               // Cargamos el driver JDBC que vayamos a usar
               Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          }catch(ClassNotFoundException e){
               System.out.println("Unable to load Driver Class");
          }

          try{
               // Todos los accesos a bases de datos deben ir entre try/catch
               // Establecemos una conexión con nuestra base de datos
               String url = "jdbc:odbc:AlumnosBD";
               Connection con = DriverManager.getConnection(url,"","");

               // Creamos y ejecutamos una sentencia SQL
               Statement stmt = con.createStatement();
```

```
                    ResultSet rs = stmt.executeQuery("SELECT * FROM ALUMNOS");

                    // Tratamos los resultado obtenidos en la consulta SQL
                    while(rs.next()){
                         String nombre = rs.getString("NOMBRE");
                         int edad = rs.getInt("EDAD");
                         System.out.println("Nombre: " + nombre + "  Edad: " +
                         edad);
                    }
                    rs.close();
                    stmt.close();
                    con.close();
               }catch(SQLException se){
               // Informamos al usuario de los errores SQL que se han producido
                    System.out.println("SQL Exception: " + se.getMessage());
                    se.printStackTrace(System.out);
               }
          }
     }
```

A continuación y basándonos en el código anterior, iremos comentando los pasos básicos a dar a la hora de acceder a una base de datos mediante JDBC.

12.5.- IMPORTACIÓN DE LAS CLASES NECESARIAS

Todo el conjunto de clases e interfaces que constituyen JDBC se encuentran dentro del paquete java.sql. Por tanto lo primero que tendremos que hacer es importar las clases e interfaces de este paquete:

```
import java.sql.*;
```

12.6.- CARGA DEL DRIVER JDBC

El primer paso que deberemos dar a la hora de acceder a una base de datos mediante JDBC será cargar el driver JDBC que vayamos a usar. La carga del driver se realiza únicamente con una sentencia:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

El método anterior tomará como parámetro un String que identifique a la clase del driver JDBC. En nuestro caso usaremos el driver JDBC-ODBC Bridge que se corresponde con la clase "sun.jdbc.odbc.JdbcOdbcDriver ".

Los drivers son los que permiten que una misma aplicación trabaje con distintos tipos de bases de datos (Oracle, Sybase, Access,...). Su misión es la de transformar las sentencias SQL emitidas por el programa al lenguaje nativo de la base de datos.

La documentación del driver será la que nos dé el nombre de la clase a utilizar. Si por ejemplo el nombre de la clase del driver fuese jdbc.DriverXYZ, cargaríamos el driver con la siguiente línea de código:

```
Class.forName("jdbc.DriverXYZ");
```

12.7.- ESTABLECER UNA CONEXIÓN CON LA BD

Una vez que tenemos cargado nuestro driver JDBC, deberemos establecer una conexión con la base de datos a la que queramos acceder. Para ello usaremos la siguiente sentencia genérica:

```
Connection con = DriverManager.getConnection(URL,"MyLogin","MyPassword");
```

El método getConexión() recibe como parámetros la URL de la base de datos con la que nos queremos conectar y el username y la password necesarios para la conectarnos a esa base de datos.

La URL de la base de datos depende del driver JDBC que estemos usando. La forma general es:

```
jdbc:<subprotocolo>:<subnombre>
```

La documentación del driver será la que nos dé las pautas para formar la URL correctamente. En caso de que el driver usado sea JDBC-ODBC Bridge, la URL comenzará por la cadena "jdbc:odbc: " y terminará con el nombre de la fuente de datos con la que la base de datos ha sido registrada en ODBC.

En nuestro ejemplo, la base de datos fue registrada en ODBC con el nombre "AlumnosBD " y no se requiere ni username ni password para poder conectarse a ella, con lo cual la conexión se realiza de la siguiente manera:

```
String url = "jdbc:odbc:AlumnosBD";
Connection con = DriverManager.getConnection(url,"","");
```

El objeto de tipo Connection obtenido representa una sesión abierta con la base de datos. Es decir, provee de un contexto en el que emitir sentencias SQL y obtener resultados. Una misma aplicación puede tener varias conexiones a una misma base de datos y/o varias conexiones a distintas bases de datos.

12.8.- CREAR UN OBJETO JDBC STATEMENT

Los objetos Statement se usan para ejecutar sentencias SQL sobre la base de datos. Para obtener un objeto Statement utilizaremos el método createStatement() del objeto Connection que obtuvimos anteriormente. Se hará de la siguiente forma:

```
Statement stmt = con.createStatement();
```

Una vez que hemos creado un objeto Statement podremos usarlo para ejecutar sentencias SQL sobre la base de datos y obtener sus resultados.

Las sentencias SQL podrán ser consultas que devuelvan unos resultados o sentencias que modifiquen de alguna forma la base de datos. Para ello, el objeto Statement nos proporciona dos métodos distintos:

- El método executeUpdate()
- El método executeQuery()

12.9.- MÉTODO EXECUTEUPDATE

Este método se usa para ejecutar todas las sentencias SQL que impliquen la creación, modificación o borrado de una tabla (INSERT, DELETE, UPDATE y comandos DDL –CREATE, DROP, ALTER,...). Por tanto, con este método realizaremos sentencias de creación de tablas (CREATE), modificación de la estructura de tablas (ALTER), destrucción de tablas (DROP), inserción de filas (INSERT), modificación de filas (UPDATE) y destrucción de filas (DELETE).

A continuación se presentan una serie de ejemplos que muestran como ejecutar este tipo de sentencias:

- Ejecución de una sentencia SQL que inserta una nueva fila en la tabla de alumnos.

```
String sentencia = "INSERT INTO ALUMNOS VALUES('666','JORGE RUIZ',26)";
stmt.executeUpdate(sentencia);
```

- Ejecución de una sentencia SQL que elimina una fila de la tabla de alumnos.

```
String sentencia = "DELETE FROM ALUMNOS WHERE EDAD = 20";
stmt.executeUpdate(sentencia);
```

- Ejecución de una sentencia SQL que modifica el contenido de una fila de la tabla alumnos.

```
String sentencia = "UPDATE ALUMNOS SET EDAD = 19 WHERE DNI = '666'";
stmt.executeUpdate(sentencia);
```

Como se puede observar lo único que hay que hacer para ejecutar una sentencia SQL con el método executeStatement es pasarle como parámetro (en forma de String) la sentencia SQL que queramos ejecutar. El objeto Statement se encargará de enviar la sentencia SQL a la base de datos para que se ejecute y de devolvernos los resultados. Ya hablaremos de los resultados más adelante.

12.10.- METODO EXECUTEQUERY

El método executeQuery se usa para ejecutar sobre la base de datos todas las sentencias SQL de tipo "SELECT ". En nuestro ejemplo aparece la siguiente sentencia:

```
stmt.executeQuery("SELECT * FROM ALUMNOS");
```

Lógicamente, las sentencias SQL que ejecutemos pueden ser todo lo complejas que queramos, no hay restricciones de ningún tipo, se puede ejecutar cualquier sentencia que sea permitida por el lenguaje SQL. Para ello, lo único que tendremos que hacer es pasarle como parámetro al método executeQuery la sentencia SQL que queremos ejecutar. Aquí tenemos un ejemplo un poco más complejo:

```
String query = "select dni, nombre from Alumnos where edad = 22";
stmt.executeQuery(query);
```

De poco valdría el poder ejecutar sentencias SQL si no tendríamos forma de procesar los resultados que éstas nos devuelven (sobre todo en el caso de las consultas tipo "SELECT "). Por ello, tanto el método executeUpdate como el método executeQuery nos devolverán como valor de retorno los resultados generados por las consultas SQL que ejecutan.

12.11.- MANIPULAR LOS RESULTADOS GENERADOS

Cuando ejecutamos una consulta SQL mediante el método executeQuery (será, por tanto, una consulta de tipo "SELECT "), los resultados forman una especie de pseudotabla que contiene todas las filas devueltas por la consulta SQL. Por ejemplo, esta sería la representación visual de la tabla que devolvería como resultado la consulta SQL "SELECT NOMBRE, EDAD FROM ALUMNOS " (suponiendo que previamente hubiésemos insertado datos en esa tabla, algo que podríamos haber hecho directamente desde Microsoft Access):

NOMBRE	EDAD
Jane Markham	23
Louis Smith	21
Woodrow Lang	32
John Smith	19

Este tipo de representación no es la más apropiada para los programas Java. En su lugar, JDBC usa el interfaz java.sql.ResultSet para encapsular los resultados de la consulta SQL. Podríamos pensar en un ResulSet como un objeto que representa la tabla generada como resultado de una consulta SQL y que tendrá una serie de métodos que nos permitirán navegar entre las filas de la tabla y recuperar los valores de sus columnas.

El siguiente programa ejecuta la consulta SQL anterior y trata sus resultados:

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT NOMBRE, EDAD FROM ALUMNOS");
while(rs.next()){
     System.out.print("Nombre: " + rs.getString("NOMBRE"));
     System.out.print(", ");
     System.out.println("Edad: " + rs.getInt("EDAD"));
}
rs.close();
stmt.close();
```

En el código anterior el método executeQuery nos devuelve un ResultSet que contiene los resultados de la consulta SQL. Este ResultSet será una estructura de datos que simula la tabla mostrada anteriormente.

Mediante un bucle vamos recorriendo una a una todas las filas de la tabla mediante el método next() del ResultSet. Nada más obtener el ResultSet que nos devuelve el método executeQuery, estaremos situados por encima de la primera fila. Esto significa que para acceder a la primera fila (y por tanto a los datos del ResultSet) deberemos comenzar haciendo una primera llamada al método next(). Las sucesivas llamadas al método next() nos irán desplazando una a una por las filas del ResultSet. En caso de que no existan más filas en el ResultSet, el método next() nos devolverá false. Hacer notar como en la versión JDBC 1.0, que nosotros usaremos en la asignatura, los ResultSet solamente se pueden recorrer hacia delante y nunca hacia atrás.

Una vez que estamos situados en una determinada fila del ResultSet, podremos consultar los valores de cada columna mediante métodos del tipo getXXX(). Usaremos un método getXXX() diferente en función del tipo de datos que contenga la columna que vayamos a consultar. Así, si la columna contiene valores enteros usaremos el método getInt(), si contiene

valores de texto usaremos el método getString(). La siguiente tabla resume los métodos más usados para consultar los valores de cada columna y el tipo de datos SQL para el que se suelen usar:

MÉTODO	TIPO DE DATOS SQL
getInt()	INTEGER
getLong()	BIGINT
getFloat()	REAL
getDouble()	FLOAT, DOUBLE
getString()	CHAR, VARCHAR, LONGVARCHAR
getBoolean()	BIT
getDate()	DATE
getTime()	TIME

De todos estos métodos es especialmente útil el método getString(). Este método nos permite leer no sólo valores de texto sino cualquier otro tipo de datos, devolviéndonos su valor en forma de String. De esta forma, podremos leer valores numéricos con el método getString() y éste nos devolverá ese número convertido en una cadena de caracteres.

Los métodos de tipo getXXX() reciben como parámetro el identificativo de la columna cuyos datos se quieren leer. De cada método getXXX() tenemos dos versiones:

1. Uno que recibe como parámetro un String con el nombre de la columna que se quiere leer. Por ejemplo: getString("nombre "). No tiene importancia si el nombre de la columna está en mayúsculas o en minúsculas.

2. Otro que recibe como parámetro un int que identifica el orden de la columna que se quiere leer. Por ejemplo: getString(1). En este caso estaríamos leyendo la columna "Nombre " (Ojo, el orden es con respecto a la tabla que nos devuelve la consulta SQL y no con respecto a la tabla de Alumnos original). El orden de las columnas empieza a contar a partir del 1 y no desde el 0.

El método executeUpdate también nos devolverá el resultado de su ejecución. Las sentencias ejecutadas con este método son sentencias de manipulación de tablas y como tal, no devuelven filas de una tabla como ocurría con el método executeQuery. El método executeUpdate, en cambio, nos devuelve un valor numérico (int) que representa el número de filas de la tabla que han sido afectadas por la consulta SQL. Así, si la consulta SQL es de borrado de filas, nos devolverá el número de filas borradas, si es de modificación, el número de filas modificadas. Si las consultas son de creación, modificación o borrado de tablas (consultas DDL) el método nos devolverá 0.

12.12.- CERRAR LOS RECURSOS ABIERTOS

Siempre deberemos cerrar los recursos que usemos para acceder a la base de datos. En este caso, cerraremos las conexiones (Connection), los Statement y los ResultSet mediante el método close().

En el código del ejemplo inicial:

```
rs.close();
```

```
stmt.close();
con.close();
```

12.13.- TRATAMIENTO DE LAS EXCEPCIONES

La mayor parte de las operaciones que nos proporciona el API JDBC lanzarán la excepción java.sql.SQLException en caso de que se produzca algún error en el acceso a la base de datos (por ejemplo: errores en la conexión, sentencias SQL incorrectas, falta de privilegios,...). Por este motivo es necesario dar un tratamiento adecuado a estas excepciones y encerrar todo el código JDBC entre bloques try/catch.

En el código de nuestro ejemplo tratamos las excepciones mostrando un mensaje de error al usuario:

```
try{
          ...
}catch(SQLException se){
          // Informamos al usuario de los errores SQL que se han producido
          System.out.println("SQL Exception: " + se.getMessage());
          se.printStackTrace(System.out);
}
```

12.14.- EJECUCIÓN DE SENTENCIAS COMPLEJAS: JOINS

La base de datos en la que nos hemos basado en nuestro ejemplo estaba formada por una única tabla. Lo normal es que las bases de datos tengan más de una tabla y muy a menudo surgirán en nuestros programas situaciones en la que tengamos que ejecutar consultas que involucren a más de una tabla al mismo tiempo. Es lo que habitualmente se conoce en bases de datos como JOINS.

En estos casos la manera de proceder es exactamente igual que la indicada anteriormente. A continuación veremos un ejemplo.

Supongamos que tenemos una base de datos formada por tres tablas: tabla ALUMNOS, tabla ASIGNATURAS y tabla NOTAS. Esta es la estructura de las tablas:

TABLA ALUMNOS

CAMPO	TIPO
DNI	Texto
Nombre	Texto
Edad	Numérico

TABLA NOTAS

CAMPO	TIPO
DNI	Texto
Código	Numérico
Nota	Numérico

TABLA ASIGNATURAS

CAMPO	TIPO
Código	Numérico
Nombre	Texto
Curso	Numérico

Ejercicio Propuesto: Crear la base de datos usando Access.

Supongamos ahora que la base de datos ha sido registrada en ODBC con el nombre "NotasBD ".

Ejercicio Propuesto: Registrar la base de datos en ODBC.

Si quisiésemos construir un programa que nos visualice las notas que un determinado alumno (por ejemplo, Juan) ha sacado en las asignaturas de 3º, tendríamos que hace una consulta SQL que involucre a las tres tablas. La consulta SQL sería la siguiente:

```
SELECT * FROM ALUMNOS, NOTAS, ASIGNATURAS WHERE ALUMNOS.DNI = NOTAS.DNI AND
        NOTAS.CODIGO = ASIGNATURAS.CODIGO AND ALUMNOS.NOMBRE = 'JUAN'
        AND CURSO = 3
```

Y el programa que ejecuta esa consulta y visualiza los resultado por pantalla sería el siguiente:

```
import java.sql.*;

public class JDBCJoin{

     public static void main(java.lang.String[] args){

          try{
               // Cargamos el driver JDBC que vayamos a usar
               Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          }catch(ClassNotFoundException e){
               System.out.println("Unable to load Driver Class");
          }

          try{
               // Establecemos una conexión con nuestra base de datos
               String url = "jdbc:odbc:NotasBD";
               Connection con = DriverManager.getConnection(url,"","");

               // Creamos y ejecutamos una sentencia SQL
               Statement stmt = con.createStatement();
               String consulta = "SELECT * FROM ALUMNOS, NOTAS, ASIGNATURAS
               WHERE ALUMNOS.DNI = NOTAS.DNI AND NOTAS.CODIGO =
               ASIGNATURAS.CODIGO AND ALUMNOS.NOMBRE = 'JUAN' AND CURSO = 3";
               ResultSet rs = stmt.executeQuery(consulta);

               // Tratamos los resultados obtenidos en la consulta SQL
               while(rs.next()){
                    int codigo = rs.getInt("CODIGO");
                    String nombre = rs.getString(8);
                    int nota = rs.getInt("NOTA");
                    System.out.println(codigo + "\t" + nombre + "\t" + nota);
               }
               rs.close();
               stmt.close();
               con.close();
          }catch(SQLException se){
          // Informamos al usuario de los errores SQL que se han producido
               System.out.println("SQL Exception: " + se.getMessage());
               se.printStackTrace(System.out);
          }
     }
}
```

Comentarios al programa: Fijarse como hay algunas columnas en tablas diferentes que se llaman de la misma manera. Por ejemplo, la tabla Alumnos y la tabla Asignaturas tienen una columna que se llama "Nombre" (en un caso se refiere al nombre del alumno y en otro al

nombre de la asignatura). A la hora de tratar los datos del ResultSet ¿cómo podemos acceder a una columna u otra si ambas se llaman igual? ¿Qué pasa si hacemos rs.getString("Nombre ")?

En este caso el acceso a los datos del ResultSet a través del nombre de la columna produce ambigüedad en algunos casos al haber varias columnas con el mismo nombre. Lo apropiado, en este caso, es usar números para acceder a cada columna.

¿Cómo podemos saber el número que le corresponde a cada columna de la tabla resultado? Simplemente hay que ver qué columnas se han recogido en la tabla resultado (en este caso, al usar el *, se han recogido todas las columnas de las tres tablas) y en qué orden hay que ir colocando cada columna en la tabla resultado (siempre en el mismo orden en que se ha hecho referencia a las tablas en la cláusula FROM; en este caso primero las de la tabla Alumnos, luego las de la tabla Notas y luego las de la tabla Asignaturas). Según esto el número que le corresponde a cada columna es el siguiente:

1. DNI (Tabla Alumnos)
2. Nombre (Tabla Alumnos)
3. Edad
4. DNI (Tabla Notas)
5. Código (Tabla Notas)
6. Nota
7. Código (Tabla Asignaturas)
8. Nombre (Tabla Asignaturas)
9. Curso

Si se observa el código, se puede ver como para acceder a la columna que contiene el nombre de la asignatura se ha usado el número 8.

```
String nombre = rs.getString(8);
```

12.15.- EJERCICIOS

12.15.1.- Gestión de Matriculaciones en Carreras de la UD

Este ejercicio consiste en la realización de un programa que permita llevar a cabo un pequeño control de las matriculaciones de alumnos en las carreras de la Universidad de Deusto.

Este programa deberá mantener información relativa a los alumnos y a las carreras que se ofertan en la UD. La base de datos sobre la que se apoya nuestro programa podría tener la siguiente estructura:

TABLA ALUMNOS

CAMPO	TIPO
DNI	Texto
Nombre	Texto
Poblacion	Texto
Telefono	Texto
FechaNato	Texto
CodCarrera	Numérico

TABLA CARRERAS

CAMPO	TIPO
CodCarrera	Numérico
Titulacion	Texto
Años	Numérico
Campus	Texto

De la base de datos anterior se deduce que un alumno únicamente puede estar matriculado de una sola carrera al mismo tiempo. Partiremos de este supuesto para hacer nuestra aplicación.

La interacción del usuario con nuestro programa se llevará a cabo a través de la interfaz gráfica de usuario de que muestra en la figura 12.17.

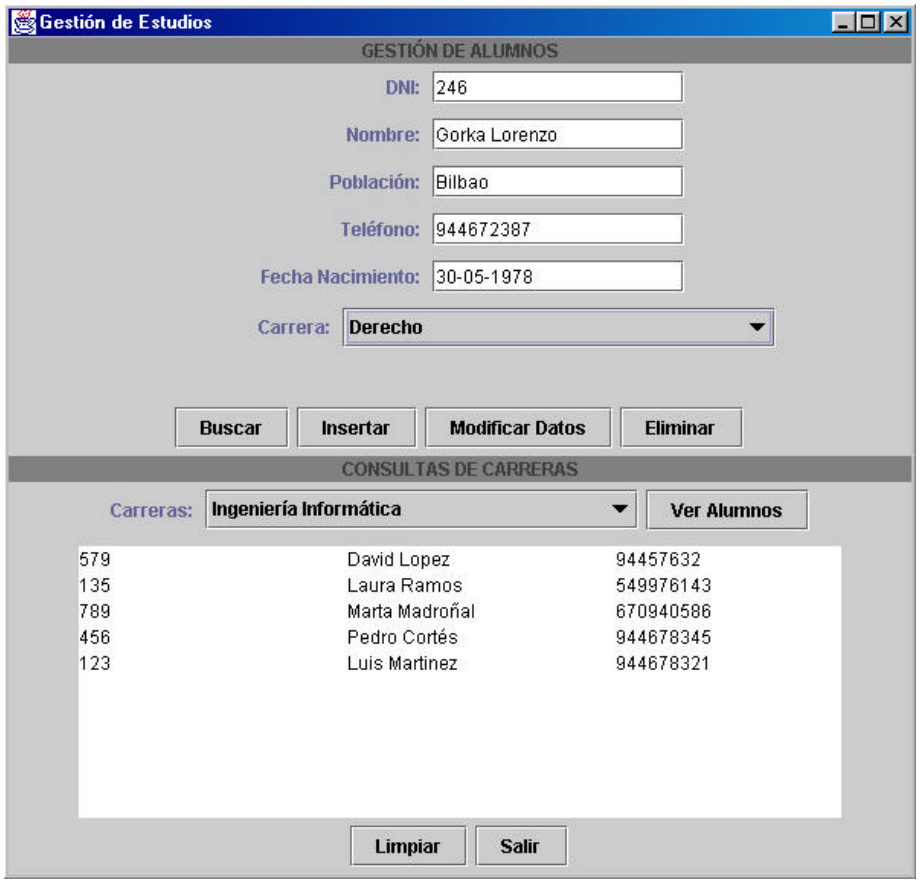

Figura 12.17: Interfaz de usuario de la aplicación

Como se puede apreciar en la figura 12.17, la interfaz está dividida en dos secciones:

- Gestión de Alumnos: donde se realizan la altas, bajas, modificaciones y búsquedas de los alumnos.
- Consultas de Carreras: donde se sacan listados de los alumnos que cursan una determinada carrera.

A continuación se detallan cada una de las funciones que debe soportar el programa a realizar:

- Matriculaciones de Alumnos

  El objetivo será insertar un nuevo alumno, matriculándolo en una determinada carrera. Para ello se introducirán los datos del alumno, se seleccionará la carrera en la que se matricula (en la lista desplegable), se pulsará el botón "Insertar" y el alumno será insertado en la base de datos.

  Nota: Se debe comprobar que el DNI del alumno introducido no coincida con el DNI de otro alumno que ya esté en la base de datos. Si así es, se visualizará un mensaje de error.

- Búsquedas de Alumnos

El objetivo será visualizar los datos de un alumno dado su DNI. Para ello se introducirá el DNI del alumno en la caja de texto correspondiente, se pulsará el botón "Buscar" y los datos del alumno serán visualizados en las cajas de texto.

Nota: En caso de que no exista ningún alumno con ese DNI, se visualizará un mensaje de error.

- Borrado de Alumnos

  El objetivo será eliminar un determinado alumno dado su DNI. Para ello de introducirá el DNI del alumno en la caja de texto correspondiente, se pulsará el botón "Eliminar" y el alumno será borrado de la base de datos.

  Nota: En caso de que no exista ningún alumno con ese DNI, se visualizará un mensaje de error.

- Modificación de los Datos de los Alumnos

  El objetivo será modificar los datos de un alumno ya existente en la base de datos dado su DNI. Para ello, el procedimiento general a seguir será el siguiente:

  - Se introducirá el DNI del alumno en la caja de texto correspondiente y se pulsará el botón "Buscar".
  - Los datos del alumno se visualizarán en las cajas de texto.
  - Se reeditarán los datos que se quieran modificar (no se contempla la modificación del DNI del alumno) y se pulsará el botón "Modificar Datos".
  - Los datos del alumno con ese DNI deberán ser modificados en la base de datos.

  Nota: Se deberá comprobar que el DNI del alumno a modificar exista en la base de datos, por si el usuario de forma malintencionada intentase modificar el DNI. Si así es, se visualizará un mensaje de error.

- Listado de los Alumnos Matriculados en una Carrera

  El objetivo es visualizar un listado con los datos de todos los alumnos que están matriculados de una determinada carrera. Para ello se seleccionará en la segunda lista desplegable la carrera que se quiere consultar, se pulsará el botón "Ver Alumnos" y los datos de todos los alumnos de esa carrera se visualizarán en el área de texto de la parte inferior (visualizar el DNI, nombre y teléfono).

  Nota: Si en la carrera seleccionada no hubiese matriculado ningún alumno, se visualizaría en el propio área de texto un mensaje que indique tal circunstancia.

La figura 12.18 muestra el contenido de las listas desplegables y la visualización de los mensajes de error.

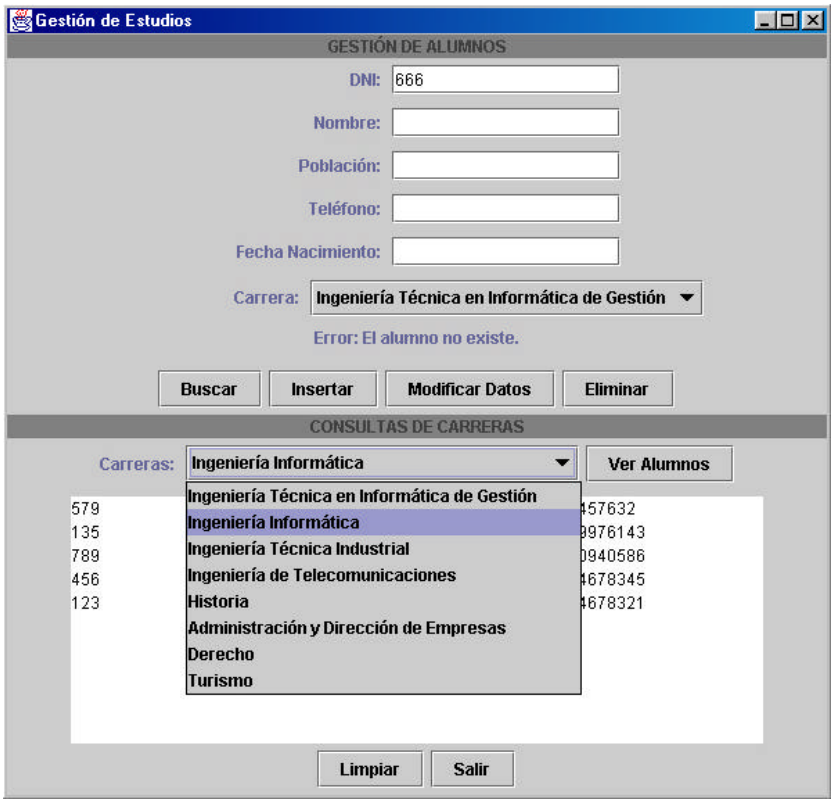

Figura 12.18: Contenido de las listas desplegables

Partiremos del supuesto de que las carreras de la UD son constantes y se encuentran ya insertadas en la tabla CARRERAS de la base datos (puedes realizar la inserción de los datos directamente desde Access). Ambas listas desplegables contendrán los nombres de todas las carreras de la UD, tal y como se muestra en la figura 12.18. Además en esta figura se muestra la visualización del mensaje de error "Error: El alumno no existe ".

12.15.2.- Ampliación: Gestión de Profesores y Asignaturas

Este ejercicio consiste en ampliar la base de datos del ejercicio anterior para que además soporte la información relativa a las asignaturas de cada carrera y a los profesores que las imparten.

Para ello, a la base de datos anterior se le deberían añadir estas tres tablas:

TABLA PROFESORES

CAMPO	TIPO
DNI	Texto
Nombre	Texto
Antigüedad	Numérico
Despacho	Texto
Extensión	Numérico
Email	Texto

TABLA PROF_ASIG

CAMPO	TIPO
DNI	Texto
CodAsig	Numérico

TABLA ASIGNATURAS

CAMPO	TIPO
CodAsig	Numérico
Nombre	Texto
Curso	Numérico
Creditos	Numérico
Tipo	Texto
CodCarrera	Numérico

Supuestos:

- Un profesor puede impartir clase de varias asignaturas.

- Una misma asignatura puede ser impartida por varios profesores.
- Cada asignatura sólo se imparte en única carrera.

Introducir algunos datos en la base de datos y realizar las siguientes consultas:

- Visualizar todas las asignaturas de la carrera de "Ingeniería Informática" junto con los profesores que imparten cada una de las asignaturas y el curso en el que se imparten.
- Visualizar las carreras en las que el profesor "Pepito Grillo" imparte alguna asignatura.
- Visualizar el número de asignaturas que se imparten en cada una de las carreras.
- Visualizar cuántos créditos constituyen las asignaturas del curso 3º de "Ingeniería Técnica en Informática de Gestión".
- Visualizar los profesores que imparten las asignaturas de 2º de "Derecho".
- Visualizar todas las carreras con una duración inferior a tres años, junto con el plan de estudios (asignaturas que la forman y su tipo) de cada una de ellas (clasificadas por cursos).
- Visualizar qué asignaturas imparte el profesor "Pepito Grillo", junto con la carrera y el curso en que imparte cada una de ellas.
- Visualizar todos aquellos profesores que comparten asignatura con el profesor "Pepito Grillo".
- Visualizar todas las asignaturas del tipo "Libre Elección" que hay en la carrera de "Derecho".

Realizar, además, las siguientes operaciones:

- Insertar, dentro de la carrera de "Ingeniería Informática", una nueva asignatura con las siguientes características:

  CodAsig: 778 Curso: 4º Tipo: "Troncal"

  Nombre: "ADSI" Créditos: 9

- Incrementa en una unidad la antigüedad de todos los profesores.
- Eliminar todas las asignaturas de la carrera de "Derecho" que pertenezcan al 5º curso y que sean de "Libre Elección".
- La asignatura "Etica" deja de impartirse. Eliminar todos aquellos profesores que en estos momentos solamente impartan esa asignatura (ojo, los profesores que además de "Etica" estén impartiendo otra asignatura no deben ser eliminados). Por último, elimina la asignatura "Etica" de la base de datos.

Nota: la visualización de los resultados de las consultas puedes realizarla en la propia pantalla de línea de comandos de MS-DOS, no hace falta que crees una nueva interfaz de usuario para este ejercicio.