

JDBC: acceso a bases de datos

8.1. Introducción

8.1.1. ¿Qué es ODBC?

Open Database Connectivity (ODBC) es una interface de aplicaciones (API) para acceder a datos en sistemas gestores de bases de datos tanto relacionales como no relacionales, utilizando para ello SQL (Lenguaje de Consulta Estructurado).

Todas las aplicaciones que soporten ODBC reconocerán una instrucción común de Lenguaje de Consulta Estructurado (SQL).

Las aplicaciones ODBC o aplicaciones cliente envían peticiones a un servidor de bases de datos. El gestor del driver ODBC determina qué fuente de datos usar y qué driver ODBC puede comunicar con esa fuente de datos en particular. La petición se envía luego a través del driver al servidor - normalmente una aplicación de base de datos. Esta base de datos puede ser local, o en el mismo ordenador, o remota. Los datos solicitados se devuelven a través del gestor del driver ODBC, entonces a la aplicación del cliente. El lenguaje ODBC es una combinación de llamadas de función ODBC API y lenguaje SQL.

Antes de continuar, es útil conocer los siguientes términos:

- **Sistema de Gestión de Bases de Datos (DBMS):** Aplicación que permite a los usuarios almacenar, procesar, y recuperar información en una base de datos
- **Fuente de Datos (DSN):** Los datos a los cuales quiere acceder (como a DBMS) e información para localizar datos (como la ruta o dirección IP)
- **Lenguaje de Consulta Estructurado (SQL):** Un lenguaje de programación estándar que controla e interactúa con una DBMS
- **Aplicación Cliente:** La aplicación que solicita datos (mediante SQL) de una fuente de datos usando ODBC
- **Query:** Recuperación, manipulación, o modificación de datos desde una fuente de datos enviando instrucciones SQL
- **Tabla:** Recolección de datos cuya estructura lógica es en forma de campos (atributos) y registros es decir, columnas y filas.
- **Driver o controlador ODBC:** Un fichero DLL, fichero conectado dinámicamente (Windows) que envía una consulta SQL para acceder a datos almacenados en una base de datos y entregar datos a la aplicación cliente

8.1.2. ¿Qué es y que hace JDBC?

JDBC es una API de Java para ejecutar sentencias SQL. (Como punto de interés, JDBC es nombre de una marca registrada y no es un acrónimo, a pesar de todo, JDBC es a menudo interpretado como “Java DataBase Connectivity”). Consta de un conjunto de clases e interfaces escrito en lenguaje de programación Java.

Usando JDBC es fácil enviar sentencias SQL a virtualmente cualquier base de datos relacional. En otras palabras, con la API JDBC no es necesario escribir un programa para acceder a una base de datos tipo Access, otro programa para acceder a una base de datos tipo Oracle y así para cada tipo de base de datos. Uno puede escribir un solo programa usando la API JDBC y el programa será capaz de enviar sentencias SQL a la base de datos apropiada. Y, con una aplicación escrita en Java, uno no tiene por qué preocuparse por escribir diferentes programas para diferentes plataformas. La combinación de JDBC permite al programador escribir una vez y ejecutar en cualquier sitio.

Java, siendo robusto, seguro, fácil de usar, fácil de entender, y automáticamente descargable en una red, es un excelente lenguaje base para aplicaciones con bases de datos. Lo que es necesario es una forma para que las aplicaciones Java puedan entenderse con bases de datos de diferentes tipos. JDBC es el mecanismo para hacer esto.

JDBC extiende lo que puede hacerse con Java. Por ejemplo, con Java y la API de JDBC, es posible publicar una página web que usa información obtenida de una base de datos remota. O una compañía puede usar JDBC para conectar todos sus empleados (incluso si estos están usando un conglomerado de máquinas Windows, Macintosh y UNÍS) a una o más bases de datos internas vía una intranet.

De una forma simple, JDBC posibilita hacer tres cosas:

- Establecer una conexión con una base de datos
- Enviar sentencias SQL
- Procesar los resultados

8.1.3. JDBC versus ODBC y otras APIs

La API ODBC de Microsoft es probablemente la interface de programación para acceder a bases de datos relacionales más extensamente usada. Ofrece la posibilidad de conectar a casi la totalidad de bases de datos. Entonces, ¿por qué no usar simplemente ODBC desde Java?

La respuesta es que se puede usar ODBC desde Java, pero esto se hace mejor con la ayuda de JDBC en la forma de un Puente JDBC-ODBC, el cual trataremos en breve. La pregunta es ahora ¿Por qué se necesita JDBC? Hay varias respuestas para esta pregunta:

- ODBC no es apropiado para su uso directo desde Java porque usa una interface en C. Las llamadas desde Java a código C nativo tienen un número de inconvenientes en la seguridad, implementación, robustez y portabilidad de las aplicaciones.
- Una traducción literal de la ODBC API en C a una API en Java no sería deseable. Por ejemplo, Java no tiene punteros y ODBC hace un copioso uso de ellos. Se puede pensar en JDBC como ODBC traducido a un interface orientado a objetos que es natural para programadores en Java.
- ODBC es duro de aprender. Mezcla características elementales con otras más avanzadas y tiene complejas opciones incluso para las consultas más simples. JDBC, por otro lado, fue diseñado para mantener simples las cosas simples mientras permite posibilidades más avanzadas cuando se requieren.
- Una API en Java como JDBC es necesaria para conseguir una solución “puramente Java”. Cuando se usa ODBC, el gestor de controladores (driver manager) y los controladores (drivers) ODBC deben ser manualmente instalados en cada máquina cliente. Sin embargo, cuando el driver JDBC está escrito totalmente en Java, el código JDBC es automáticamente instalable, portable y seguro en todas las plataformas Java desde computadoras en red hasta mainframes.

En definitiva, la JDBC API es una interface natural de Java para las abstracciones y conceptos básicos de SQL. Está fundamentada en ODBC por lo que los programadores familiarizados con ODBC encontrarán fácil de aprender JDBC. JDBC mantiene las características de diseño básicas de ODBC. La gran diferencia es que JDBC está basada y refuerza el estilo y virtudes de Java y, por supuesto, es fácil de usar.

8.1.4. Controladores JDBC

8.1.4.1. Tipos de controladores (Drivers) de JDBC

Java puede acceder a la base de datos mediante un driver o controlador JDBC apropiado que pueda utilizar las clases que permiten el acceso a la base de datos. Existen diversos tipos de drivers, que utilizan distintos protocolos. En particular, en esta asignatura se utilizará el llamado Puente JDBC-ODBC, que se explicará más adelante.

8.1.4.2. Cómo obtener los Drivers de JDBC

Para conseguir la información actualizada sobre drivers, acceder a la página web sobre drivers para JDBC de Sun, donde se mantiene un registro actualizado de vendedores de drivers:

<http://industry.java.sun.com/products/jdbc/drivers>

8.1.5. JDBC-ODBC Bridge

El Puente JDBC-ODBC es un controlador JDBC que implementa operaciones JDBC traduciéndolas en operaciones ODBC. Para ODBC aparece como una aplicación normal. El Puente implementa JDBC para cualquier base de datos para la cual haya disponible un driver ODBC.

El Puente está implementado en Java y usa métodos nativos de Java para llamar a ODBC. Se instala automáticamente con el Java Development Kit como el paquete `sun.jdbc.odbc`.

Si es posible, usar un driver JDBC 100% Java en lugar de un Puente y un driver ODBC. Esto elimina completamente la configuración en el cliente requerida por ODBC.

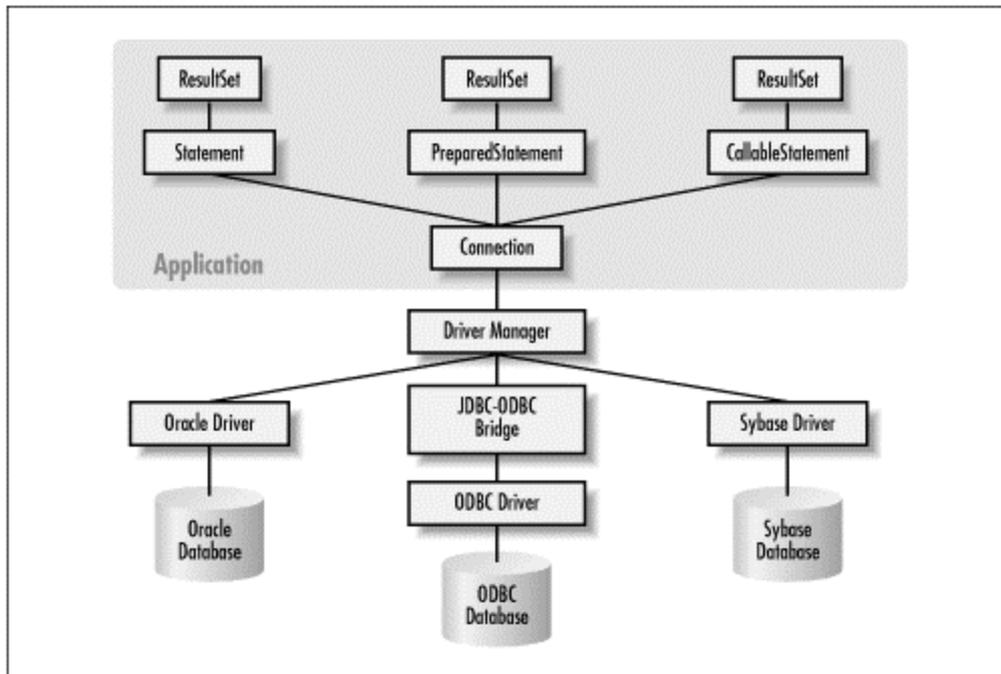


Figura 8.1. JDBC de forma esquemática

8.2. JDBC 3.0 API

La JDBC 3.0 API comprende dos paquetes:

- El paquete `java.sql`
- El paquete `javax.sql`, que añade capacidades de la parte servidor.

Los dos paquetes se consiguen automáticamente cuando se baja la **Java 2 Platform, Standard Edition, Version 1.4 (J2SE)** de la web de Sun.

8.2.1. Paquete `java.sql`

Provee la API para acceso y procesamiento de datos guardados en una fuente de datos (usualmente una base de datos relacional) usando el lenguaje de programación Java. Esta API incluye un ámbito donde diferentes drivers pueden ser instalados dinámicamente para acceder a diferentes fuentes de datos. Aunque la JDBC API está preparada principalmente para pasar sentencias SQL a una base de datos, permite leer y escribir datos desde cualquier fuente de datos con un formato tabular.

El paquete `java.sql` contiene el interface de programación para lo siguiente (la información que se muestra a continuación está obtenida directamente de la documentación de la API):

- **Making a connection with a database via the DriverManager facility**
 - `DriverManager` class -- makes a connection with a driver
 - `Driver` interface -- provides the API for registering and connecting drivers based on JDBC technology ("JDBC drivers"); generally used only by the `DriverManager` class
 - `DriverPropertyInfo` class -- provides properties for a JDBC driver; not used by the general user

- **Sending SQL statements to a database**
 - `Statement` -- used to send basic SQL statements
 - `PreparedStatement` -- used to send prepared statements or basic SQL statements (derived from `Statement`)
 - `CallableStatement` -- used to call database stored procedures (derived from `PreparedStatement`)
 - `Connection` interface -- provides methods for creating statements and managing connections and their properties
- **Retrieving and updating the results of a query**
 - `ResultSet` interface
- **Standard mappings for SQL types to classes and interfaces in Java programming language**
 - `Date` class -- mapping for SQL `DATE`
 - `Time` class -- mapping for SQL `TIME`
 - `Timestamp` class -- mapping for SQL `TIMESTAMP`
 - `Types` class -- provides constants for SQL types
- **Metadata**
 - `DatabaseMetaData` interface -- provides information about the database
 - `ResultSetMetaData` interface -- provides information about the columns of a `ResultSet` object
- **Exceptions**
 - `SQLException` -- thrown by most methods when there is a problem accessing data and by some methods for other reasons
 - `SQLWarning` -- thrown to indicate a warning
 - `DataTruncation` -- thrown to indicate that data may have been truncated

<i>Interface Index</i>	<i>Class Index</i>	<i>Exception Index</i>
<i>CallableStatement</i>	<i>Date</i>	<i>DataTruncation</i>
<i>Connection</i>	<i>DriverManager</i>	<i>SQLException</i>
<i>DatabaseMetaData</i>	<i>DriverPropertyInfo</i>	<i>SQLWarning</i>
<i>Driver</i>	<i>Time</i>	
<i>PreparedStatement</i>	<i>Timestamp</i>	
<i>ResultSet</i>	<i>Types</i>	
<i>ResultSetMetaData</i>		
<i>Statement</i>		

Tabla 8.1. JDBC API

8.2.2. DriverManager

La clase `DriverManager` es la capa gestora de JDBC, trabajando entre el usuario y el controlador (driver). Se encarga de seguir el rastro de los controladores que están disponibles y establecer la conexión entre la base de datos y el controlador apropiado.

La clase `DriverManager` mantiene una lista de clases `Driver` que se han registrado llamando al método `DriverManager.registerDriver`. Un usuario normalmente no llamará al método `DriverManager.registerDriver` directamente, sino que será llamado automáticamente por el controlador (driver) cuando este se carga. El usuario lo que hace es forzar que se cargue el driver, lo cual puede hacerse de dos formas, aunque la recomendada es llamando al método `Class.forName()`. Esto carga la clase driver explícitamente. La siguiente sentencia carga la clase `sun.jdbc.odbc.JdbcOdbcDriver`, que permite usar el Puente JDBC-ODBC:

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

La clase `sun.jdbc.odbc.JdbcOdbcDriver` ha sido escrita de forma que al cargarla crea una instancia de ella y llama a `DriverManager.registerDriver` con esa instancia como parámetro y entonces es añadida a la lista de drivers de `DriverManager` y está disponible para crear una conexión a una base de datos.

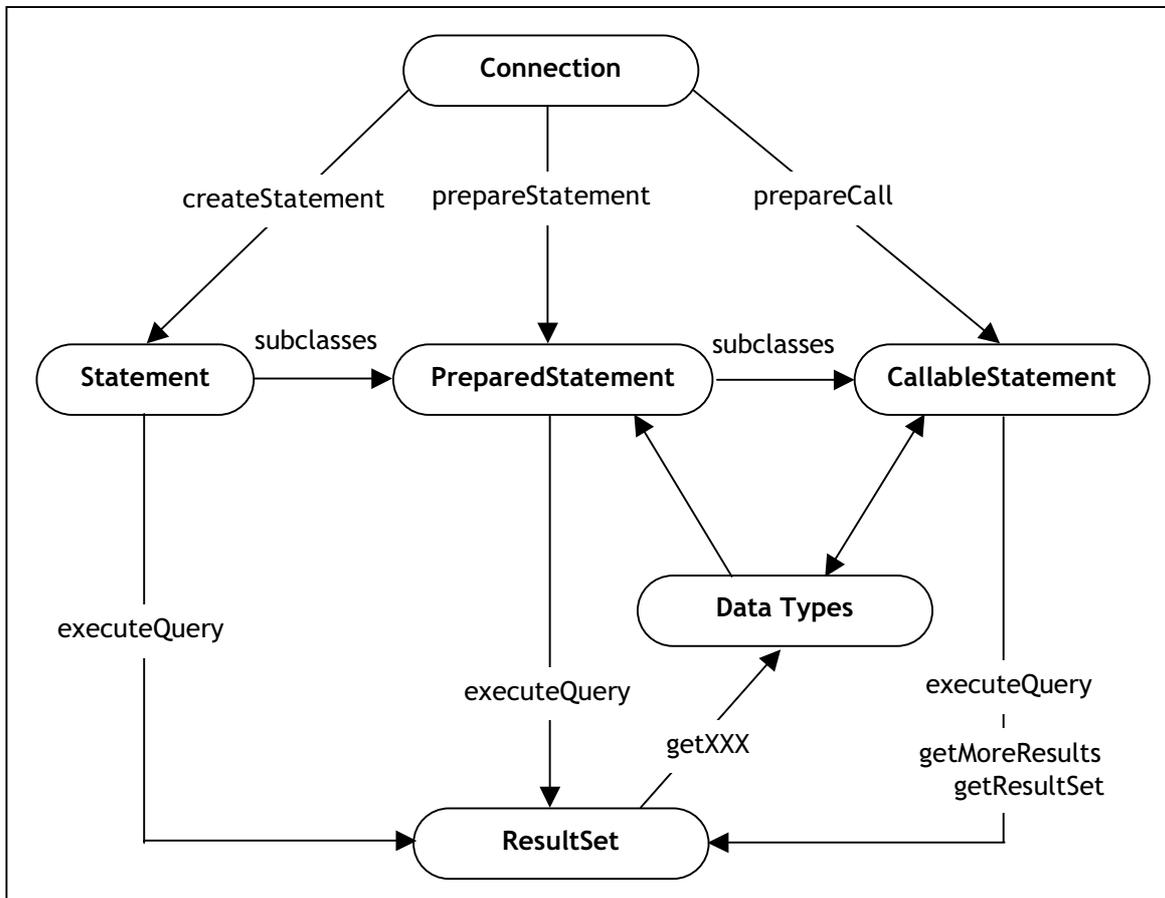


Figura 8.2. Relación entre las principales clases e interface en el paquete `java.sql`

8.2.3. Connection

Un objeto `Connection` representa una conexión a una base de datos. Una sesión con una conexión incluye las sentencias SQL que son ejecutadas y los resultados que son devueltos a través de dicha conexión. Una misma aplicación puede tener una o más conexiones con una sola base de datos o puede tener conexiones con varias bases de datos diferentes.

La forma estándar de establecer una conexión con una base de datos es llamando al método `DriverManager.getConnection`. Este método toma como parámetro una cadena de caracteres que contiene una URL. La clase `DriverManager` trata de localizar el driver que pueda conectar con la base de datos representada por esa URL.

El siguiente código ejemplifica cómo abrir una conexión a una base de datos localizada en la URL "jdbc:odbc:wombat":

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

Una URL de JDBC facilita una forma de identificar una base de datos de forma que el driver apropiado la reconozca y establezca una conexión con ella. La sintaxis estándar para URLs de JDBC es la siguiente:

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

Una URL de JDBC tiene tres partes separadas por dos puntos:

`jdbc` es el protocolo. El protocolo en una URL JDBC es siempre `jdbc`.

<subprotocolo> es usualmente el driver o el mecanismo de conectividad de la base de datos, el cual debe ser soportado por uno o más drivers. Un ejemplo de un subprotocolo es `odbc`, que ha sido reservado para URLs que especifican fuentes de datos de ODBC. Por ejemplo, para acceder a una base de datos a través del Puente JDBC-ODBC se usará una URL como la siguiente:

```
jdbc:odbc:fred
```

donde el subprotocolo es `odbc` y el subnombre es `fred`, una fuente de datos ODBC.

<subnombre> es una forma de identificar la base de datos. Puede variar dependiendo del subprotocolo y puede tener un subsubnombre con cualquier sintaxis interna que el programador del driver haya elegido. La función del <subnombre> es dar la suficiente información para localizar la base de datos.

8.2.4. Statement

Un objeto `Statement` se usa para enviar sentencias SQL a una base de datos. Una vez que se ha establecido una conexión con una base de datos particular, esa conexión puede ser usada para enviar sentencias SQL. Un objeto `Statement` se crea con el método `createStatement` de `Connection` como en el siguiente fragmento de código:

```
Connection con = DriverManager.getConnection(url);
Statement stmt = con.createStatement();
```

La sentencia SQL que será enviada a la base de datos es proporcionada como argumento a uno de los métodos para ejecutar un objeto `Statement`:

```
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table2");
```

8.2.5. ResultSet

Un `ResultSet` contiene todos los registros (filas) que satisfacen las condiciones impuestas en una sentencia SQL y proporciona acceso a los datos en dichos registros a través de un conjunto de métodos `get` que permiten acceder a los diferentes campos o atributos (columnas) del registro actual. Un `ResultSet` mantiene un cursor que apunta al registro actual. El método `ResultSet.next()` se usa para moverse al siguiente registro del `ResultSet`, haciendo el siguiente registro el registro actual.

Los métodos `getXXX` proporcionan los medios para obtener los valores de los campos, atributos o columnas del registro actual. Para cada registro, los valores de las columnas pueden ser obtenidos en cualquier orden, pero para la mayor portabilidad, se debe hacer de izquierda a derecha y leer el valor de la columna sólo una vez. Tanto el nombre de la columna como el número de esta puede ser usado para designar la columna de la cual se quiere obtener el valor. Si en el ejemplo anterior la columna "a" es de tipo entero, la "b" del tipo cadena de caracteres y la "c" de tipo coma flotante, la siguiente porción de código imprimiría los valores de todos los registros:

```
while(rs.next()){
    int i = rs.getInt("a");
    String s = rs.getString("b");
    Float f = rs.getFloat("c");
    System.out.println("ROW= " + i + " " + s + " " + f);
}
```

8.3. Empezando con JDBC

Mediante unos ejemplos vamos a introducirnos en la utilización de la API de JDBC. Para acceder a la base de datos vamos a utilizar el Puente JDBC-ODBC suministrado junto al Java Software Development Kit.

Primeramente, vamos a crear una base de datos tipo Access a la cual accederemos y un DSN (Data Source Name) en ODBC de conexión a dicha base de datos. El DSN será necesario para que el driver localice la base de datos.

8.3.1. Base de datos en formato Access

Supongamos que hemos creado una base de datos en Access con el nombre `Libros.mdb` (las bases de datos en formato Access tienen extensión `*.mdb`) que contiene una tabla denominada `Datos` que tiene como campos: `Codigo`, `Titulo` y `Autor`, todos ellos de tipo texto. Además hemos rellenado la tabla con datos de libros.

Codigo	Titulo	Autor
C001	La estructura de las revoluciones científicas	Thomas S. Kuhn
C002	Sobre la teoría de la relatividad especial y general	Albert Einstein
C003	Historia del tiempo	Stephen W. Hawking
C004	La nueva mente del Emperador	Roger Penrose
C005	Killing Time	Paul Feyerabend
N001	El perfume	Patrick Süskind
N002	Orgullo y prejuicio	Jane Austen
N003	El péndulo de Foucault	Umberto Eco
N004	El Proceso	Franz Kafka
N005	Baudolino	Umberto Eco

Figura 8.3. Forma de presentar las tablas de Microsoft Access

8.3.2. Creación de un Data Source Name (DSN)

Un Data Source Name (DSN) en ODBC es una Fuente de Datos de ODBC, que nos indica el tipo y la localización de la base de datos para que más tarde podamos acceder a ella mediante el Puente JDBC-ODBC desde nuestro programa en Java, lo cual simplifica enormemente la conexión, que queda en manos del driver.

La creación de un DSN varía según los sistemas. Nosotros vamos a ver cómo se crea en un sistema Windows. Los pasos a seguir son los siguientes:

Abrir el administrador de ODBC desde el Control Panel (Start/Settings) o mediante Start/Run... y ejecutando "odbcad32". En Windows 2000 se encuentra en Control Panel/Administrative Tools/Data Sources (DSN). En las salas de ordenadores habrá que buscar el ejecutable con el explorador (C:\WinNT\System32\Odbcad32.exe) y ejecutarlo haciendo doble clic sobre él.

Añadir un nuevo Data Source Name (DSN) mediante el botón Add y seleccionar el driver de Microsoft Access en la siguiente ventana.

Dar un nombre al DSN (en este caso *pruebaODBC*) y pulsando el botón Select asignarle la base de datos que se ha creado anteriormente (*Libros.mdb*).

El proceso completo de creación de un DSN se puede ver en la figura 6.4.

8.3.3. Ejemplo de una aplicación JDBC simple

Este primer ejemplo tiene como objetivo mostrar la utilización del paquete `java.sql` y realiza un proceso completo de registrar el Puente JDBC-ODBC, crear una conexión a una base de datos tipo Access a través de dicho driver, enviar una sentencia SQL a la base de datos a través de dicha conexión y por último leer y mostrar los resultados de la consulta SQL.

El programa recibe como primer argumento el Data Source Name (DSN) de ODBC que hemos creado antes, la sentencia SQL entre comillas que queremos enviar a la base de datos como segundo argumento y después como argumentos tercero, cuarto, etc., los nombre de los campos que queremos que nos muestre al imprimir los registros obtenidos de la base de datos.

Una vez escrito el programa en un editor de texto y después de haberlo guardado con el nombre `firstJDBC.java`, debemos compilarlo con la instrucción ya familiar:

```
javac firstJDBC.java
```

Para ejecutarlo, la instrucción que debemos usar tiene la siguiente forma:

```
java firstJDBC DataSourceName "Sentencia SQL" [Campo1 Campo2 ...]
```

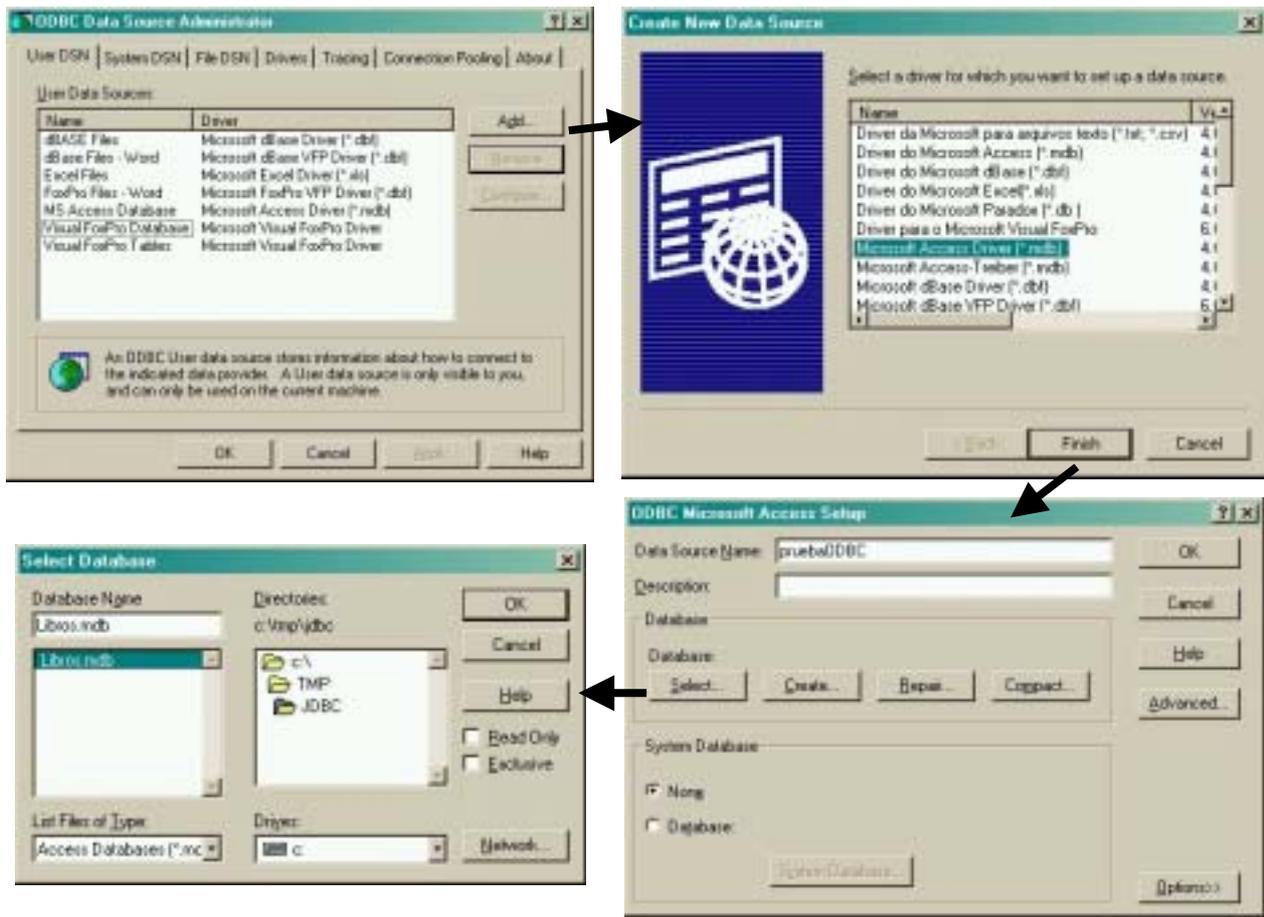


Figura 8.4. Proceso de creación de un DSN

Los corchetes indican que se trata de argumentos opcionales. Probemos a pasarle como parámetros distintas sentencias SQL para irnos familiarizando con el uso del SQL:

```
java firstJDBC pruebaODBC "SELECT * FROM Datos ORDER BY Codigo" Codigo Titulo Autor
```

La salida será la siguiente:

```
C001|La estructura de las revoluciones científicas|Thomas S. Kuhn
C002|Sobre la teoría de la relatividad especial y general|Albert Einstein
C003|Historia del tiempo|Stephen W. Hawking
C004|La nueva mente del Emperador|Roger Penrose
C005|Killing Time|Paul Feyerabend
N001|El perfume|Patrick Süskind
N002|Orgullo y prejuicio|Jane Austen
N003|El péndulo de Foucault|Umberto Eco
N004|El Proceso|Franz Kafka
N005|Baudolino|Umberto Eco
...
```

Una sentencia SQL un poco más compleja involucra selección de registros que cumplan condiciones en alguno de sus campos, como es el caso siguiente, que selecciona todos los registros cuyo código empiece por la letra "C":

```
java firstJDBC pruebaODBC "SELECT * FROM Datos WHERE Codigo Like 'C%' ORDER BY Autor" Codigo Tirulo Autor
```

La salida será la siguiente:

```
C002|Sobre la teoría de la relatividad especial y general|Albert Einstein
C005|Killing Time|Paul Feyerabend
C004|La nueva mente del Emperador|Roger Penrose
C003|Historia del tiempo|Stephen W. Hawking
C001|La estructura de las revoluciones científicas|Thomas S. Kuhn
```

Ejemplo 1:

```
import java.sql.*;

class firstJDBC {
    public static void main(String args[]) throws ClassNotFoundException,
        SQLException {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        String url="jdbc:odbc:" + args[0];

        Connection connection=DriverManager.getConnection(url);
        Statement statement=connection.createStatement();

        String sql = args[1];
        ResultSet result=statement.executeQuery(sql);

        while(result.next()) {
            for(int i=2;i<args.length;++i) {
                if(i<args.length-1)
                    System.out.print (result.getString(args[i])+"|");
                else
                    System.out.println (result.getString(args[i]));
            }
        }
        connection.close();
    }
}
```

8.3.4. Ejemplo de una aplicación JDBC con excepciones y MetaData

El siguiente ejemplo introduce más funcionalidades al ejemplo anterior. Una de ellas es que se pueden conocer las características de los campos que la base de datos devuelve a partir de una sentencia SQL en un objeto `ResultSet`. La otra característica añadida es el manejo de excepciones.

Para ejecutar este segundo ejemplo más elaborado, la instrucción que debemos usar tiene la siguiente forma:

```
java ResultAppSQL DataSourceName "Sentencia SQL"
```

Como se puede ver ya no es necesario pasar como argumentos los campos que queremos que se muestren. Dichos campos ya vienen explícitamente en la sentencia SQL que le pasamos a la base de datos y, a diferencia del ejemplo 1, hemos dotado al programa de la capacidad para reconocer dichos campos y mostrarlos con su nombre. Todo esto es posible gracias a la interface `ResultSetMetaData`, de la API de JDBC.

Probemos este nuevo programa, pues, con la siguiente sentencia y tratemos de entender lo que está haciendo el programa (es muy recomendable tener presente la ayuda de la API de Java donde también encontraremos las clases correspondientes a JDBC):

```
java ResultAppSQL pruebaODBC "SELECT * FROM Datos WHERE Codigo Like 'C%'
ORDER BY Autor"
```

La salida en este caso será la siguiente:

```
Codigo|Titulo|Autor
C002|Sobre la teoría de la relatividad especial y general|Albert Einstein
C005|Killing Time|Paul Feyerabend
C004|La nueva mente del Emperador|Roger Penrose
C003|Historia del tiempo|Stephen W. Hawking
C001|La estructura de las revoluciones científicas|Thomas S. Kuhn
```

Además, hemos dotado al programa de la capacidad para manejar excepciones lanzadas por las classes de la API de JDBC, de tal forma que cuando se produzca una excepción, el programa mostrará un mensaje como el siguiente:

Error de SQLException: Texto explicativo de la excepción (lanzado por Java)

Ejemplo 2:

```

import java.sql.*;

class ResultAppSQL {
    public static void main(String args[]) {
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        }
        catch(ClassNotFoundException ex) {
            System.out.println("Error de ClassNotFoundException" + ex);
            System.exit(0);
        }

        String url="jdbc:odbc:" + args[0];
        try {
            Connection connection=DriverManager.getConnection(url);
            Statement statement=connection.createStatement();
            String sql = args[1];
            ResultSet result=statement.executeQuery(sql);
            ResultSetMetaData rmeta=result.getMetaData();

            int numColumns=rmeta.getColumnCount();

            for(int i=1;i<=numColumns;++i) {
                if(i<numColumns)
                    System.out.print(rmeta.getColumnName(i)+"|");
                else
                    System.out.println(rmeta.getColumnName(i));
            }

            while(result.next()) {
                for(int i=1;i<=numColumns;++i) {
                    if(i<numColumns)
                        System.out.print
                            (result.getString(rmeta.getColumnName(i))+"|");
                    else
                        System.out.println
                            (result.getString(rmeta.getColumnName(i)));
                }
            }

            connection.close();
        }
        catch(SQLException ex) {
            System.out.println("Error de SQLException:" + ex);
            System.exit(0);
        }
    }
}

```

De esta forma, cuando se introduce una sentencia SQL errónea, el programa mostrará un aviso que nos dirá qué es lo que está pasando. Bien, probemos las siguientes sentencia:

```
java ResultAppSQL pruebaODBC " "
```

La respuesta del programa será la siguiente:

```
Error de SQLException:java.sql.SQLException: [Microsoft][ODBC Microsoft
Access Driver] Invalid SQL statement; expected 'DELETE', 'INSERT', 'PROCEDURE',
'SELECT', or 'UPDATE'.
```

Introduzcamos un error en la propia sentencia SQL, una tabla que no existe:

```
C:\tmp\JDBC>java ResultAppSQL pruebaODBC "SELECT * FROM OtraTabla"
```

La respuesta del programa en este caso será:

```
Error de SQLException:java.sql.SQLException: [Microsoft][ODBC Microsoft Access Driver] The Microsoft Jet database engine cannot find the input table or query 'OtraTabla'. Make sure it exists and that its name is spelled correctly.
```

Introduzcamos otro error en la propia sentencia SQL, concerniente a un campo inexistente:

```
C:\tmp\JDBC>java ResultAppSQL pruebaODBC "SELECT Precio FROM Datos"
```

La respuesta del programa en este caso será:

```
Error de SQLException:java.sql.SQLException: [Microsoft][ODBC Microsoft Access Driver] Too few parameters. Expected 1.
```