



PROGRAMACIÓN

BASES DE DATOS CON
Microsoft®
VISUAL BASIC .NET

FRANCISCO CHARTE OJEDA

ANAYA
MULTIMEDIA

Nº Reg.

00593

de 17

de 2002

17

ANAYA MULTIMEDIA

PROGRAMACIÓN

BÁSICOS DE JUEGOS

VISUAL BASIC .NET

Todos los nombres propios de programas, sistemas operativos, equipos hardware, etc. que aparecen en este libro son marcas registradas de sus respectivas compañías u organizaciones.

Reservados todos los derechos. El contenido de esta obra está protegido por la ley, que establece penas de prisión y/o multas, además de las correspondientes indemnizaciones por daños y perjuicios, para quienes reprodujeren, plagiaren, distribuyeren o comunicasen públicamente, en todo o en parte, una obra literaria, artística o científica, o su transformación, interpretación o ejecución artística fijada en cualquier tipo de soporte o comunicada a través de cualquier medio, sin la preceptiva autorización.

© EDICIONES ANAYA MULTIMEDIA (GRUPO ANAYA, S.A.), 2002

Juan Ignacio Luca de Tena, 15. 28027 Madrid

Depósito legal: M. 41.765 - 2002

ISBN: 84-415-1375-9

Printed in Spain

Imprime: Artes Gráficas Guemo, S.L.

Febrero, 32. 28022 Madrid.

índice

de contenidos

Agradecimientos	6
Introducción	18
Soluciones de acceso a datos	20
ADO.NET	20
Visual Studio .NET	21
Objetivos de este libro	21
Estructura del libro	22
Ayuda al lector	24
Parte I. Sentar las bases	25
1. Terminología y conceptos	26
Orígenes de datos	27
Bases de datos	29
DBMS y RDBMS	29
Arquitectura cliente/servidor	30
Arquitecturas n-tier	32
Servicios de acceso a datos	33
Proveedores y controladores	33
Lenguajes de comunicación	34

Estructura de los datos	36
Entidades y atributos	36
Dominios y restricciones	37
Identidad de una entidad	37
Relaciones entre conjuntos de entidades	38
Índices	39
Vistas	40
Procedimientos almacenados	40
Normalización	41
Transacciones	41
XML	43
Resumen	43
2. SQL y sus dialectos	44
¿Qué es SQL?	46
Partes de SQL	46
Derivados de SQL	47
Ejecución de sentencias SQL	47
DDL	48
Creación de una base de datos	51
Creación de tablas	51
Modificación y borrado de tablas	54
Otras operaciones de definición de datos	54
DML	56
Inserción de datos	57
Recuperación de datos	58
Alias de tablas	58
Selección de filas	59
Condicionales complejos	59
Orden de las filas	61
Expresiones y funciones de resumen	61
Agrupamiento	62
Enlaces entre tablas	62
Consultas dentro de consultas	63
Actualización de datos	64
Eliminación de datos	65
DCL	65
Derivados de SQL	66
Transact-SQL	66
Variables y tipos de datos	67
Evaluación de expresiones	68
Condicionales y bucles	68
Codificación de procedimientos almacenados	69
PL/SQL	69

Variables y tipos de datos	70
Evaluación de expresiones	70
Condicionales y bucles	71
Codificación de procedimientos almacenados	71
Resumen	72
3. Orígenes de datos	74
Orígenes locales y remotos	75
Microsoft Access	76
Definición de las tablas	77
Relación entre las tablas	79
Introducción de datos	79
Simplificar la selección de editorial	81
SQL Server	83
Creación de la base de datos	84
Definición de las tablas	85
Relación entre las tablas	87
Introducción de datos	89
Uso de la base de datos de ejemplo	90
Definición de una vista	91
Definir procedimientos almacenados	93
Ejecución de procedimientos almacenados	95
Oracle	96
Creación de la base de datos	97
Definición de las tablas	98
Introducción de datos	104
Definición de una vista	105
Definir funciones y procedimientos almacenados	106
Ejecución de funciones y procedimientos	110
InterBase	111
Creación de la base de datos	113
Definición de las tablas	115
Introducción de datos	116
Definición de una vista	117
Definir procedimientos almacenados	118
Ejecución de procedimientos almacenados	120
Microsoft Excel	120
Creación de un nuevo libro	121
Definición de la estructura	121
Introducción de datos	123
XML	125
Definición de la estructura de documento	125
Creación del documento XML	128
Directorio activo	133

Acceso al Directorio activo	133
Resumen	134
Parte II. ADO.NET	137
4. Introducción a ADO.NET	138
Objetivos del modelo ADO.NET	139
Representación interna en XML	141
Ausencia de cursores de datos	143
Cursores de lectura	144
Solución multipropósito	144
Configuración de los clientes	148
Resumen	150
5. Modelo de objetos	152
Estructura del modelo de objetos	153
Ámbitos con nombre de ADO.NET	154
Interfaces para los proveedores	155
Asociación de columnas y tablas	156
Acceso a filas de datos	156
Adaptadores de datos	159
Conexiones, comandos y transacciones	160
Detalles sobre los proveedores	161
Otras clases comunes y específicas	162
Clases independientes del origen de datos	163
Conjuntos de datos	164
Tablas	165
Filas	166
Columnas	167
Restricciones	168
Relaciones	168
Vistas de datos	169
Resumen	170
6. Conexión al origen de datos.....	172
Obtención e instalación de proveedores adicionales	173
Dónde obtener los proveedores	174
Instalación del proveedor	174
Generalidades sobre la conexión	176
Cadena de conexión	176
Apertura y cierre de la conexión	177
Propiedades informativas	178
Cadenas de conexión	178

Selección del controlador	178
Identificación del servidor u origen de datos	181
Base de datos inicial	181
Parámetros de seguridad	181
Propiedades exclusivas	183
En la práctica	183
Conexión con Microsoft Access	183
Conexión con Microsoft Excel	185
Conexión con SQL Server	186
Conexión con InterBase	188
Conexión con Oracle 8i	189
Conexiones ODBC mediante DSN	193
Tipos de DSN	194
Creación de un DSN	194
Uso del DSN con ADO.NET	196
Archivos UDL	197
Resumen	199

7. Información de esquema de la base de datos 200

¿Qué es la información de esquema?	201
Orígenes OLE DB	202
Tabla de resultados	203
En la práctica	204
Otros orígenes	206
En la práctica	206
Información sobre columnas	209
En la práctica	210
Resumen	212

8. Recuperación de datos 214

Generalidades sobre los comandos	215
Asociación entre comando y conexión	216
Definición del comando a ejecutar	217
Ejecución del comando	217
Lectura de los datos	219
Recuperar el contenido de una tabla	220
Varios conjuntos de datos	222
Ejecución de sentencias de selección	223
Sentencias con parámetros	226
Recuperación de un solo valor	228
Manipulación de datos	228
Otras operaciones	229
Recuperación de una vista	230

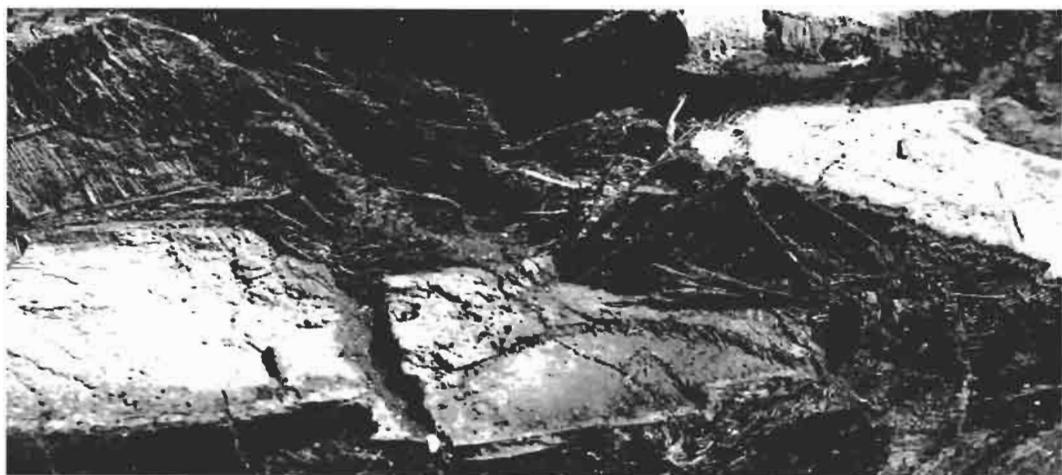
Ejecución de un procedimiento almacenado	231
Resumen	233
9. Conjuntos de datos	234
Generalidades sobre conjuntos de datos.....	235
Tablas y relaciones	236
Selección de datos	237
Generalidades sobre adaptadores de datos	237
Creación de un adaptador de datos	238
Obtención de los datos	239
Actualización de datos	241
En la práctica	243
Recuperación de datos	244
Información de esquema	246
Manipulación de los datos	250
Definición de conjuntos de datos	253
Creación de tablas, columnas y relaciones	253
Inserción de datos	255
Consulta de estructura y contenido.....	256
Almacenamiento local	258
Escritura de esquema y datos	260
DiffGrams	261
Lectura del conjunto de datos	264
Resumen	267
10. Relaciones y vistas	270
Filtrado y ordenación de un DataTable	272
Generalidades sobre DataView y DataViewManager	274
Funcionamiento de un DataView	276
Funcionamiento de un DataViewManager	277
En la práctica	278
Múltiples vistas sobre una misma tabla	278
La vista por defecto de una tabla	279
Búsqueda de datos en una vista	281
Edición de datos en la vista	284
Uso de un DataViewManager	286
Resumen	289
11. XML	290
XML y temas relacionados	291
Breve introducción a XML	292
Definiciones de tipo y esquemas	294
Ámbitos con nombre	295

Mecanismos de manipulación de documentos XML	296
Selección de datos con XPath	297
Transformación con XSLT	297
El ámbito System.Xml	298
Lectura secuencial de documentos XML	298
Manipulación de documentos XML con DOM	301
Navegación por el documento mediante XPath	305
Transformación de documentos	307
XML y ADO.NET	309
La clase XmlDocument	310
Creación del DataSet a partir del documento XML	311
Generación del documento XML a partir del DataSet.....	312
Resumen	314
Parte III. Visual Studio .NET	317
12. Capacidades de datos en Visual Studio .NET	318
Ediciones de Visual Studio .NET	319
Posibilidades de acceso	321
Productos integrados en el paquete	322
Posibilidades de diseño	322
Una visión general	324
Resumen	325
13. Herramientas visuales de datos.....	326
El Explorador de servidores	328
Definir una nueva conexión	328
Registrar un nuevo servidor	331
Apertura y cierre de conexiones	332
Creación de nuevas bases de datos	333
Edición de datos	334
Navegar por las filas	336
Selección de datos	336
Modificación, inserción y eliminación	340
Agrupación de los resultados	341
Edición de información de esquema	343
Creación y modificación de tablas	343
Diseño de vistas	346
Edición de procedimientos almacenados y funciones	347
Componentes de acceso a datos	350
Preparación de la conexión	351
Definición de comandos	352
Definición del adaptador de datos	354

Creación del conjunto de datos	355
Diseño de una sencilla interfaz	356
Creación automática de componentes	358
El asistente de configuración de adaptadores	358
Comprobación del adaptador de datos	362
Generación del conjunto de datos	362
Conjuntos de datos con tipo	363
Aún más simple	365
Resumen	366
14. Componentes con vinculación a datos	368
Tipos de vinculación	369
Vinculación simple	370
Vinculación con múltiples filas de una columna	374
Vinculación con múltiples filas y columnas	376
Enlace a datos en formularios Windows	377
Posición actual en una lista de datos	378
Control de la vinculación	379
Componentes enlazables	379
Enlace a datos en formularios Web	380
Vínculos sólo de lectura	380
Navegación con componentes simples	381
Actualización del origen	385
Resumen	386
15. Formularios de datos	388
El asistente para formularios Windows	389
Selección del DataSet	390
Definición de la conexión	391
Selección de los elementos de origen	391
Definir la relación entre las tablas	392
Selección de columnas	393
Elegir el diseño del formulario	395
Personalización del diseño	396
Análisis del código generado	397
El asistente para formularios Web	401
Análisis del código generado	402
Añadir capacidades de edición	404
Inserción de los enlaces de edición	404
Eventos y elementos de un DataGrid	405
Código asociado a los eventos	406
Actualización del origen de datos	408
Resumen	410

Parte IV. Resolución de casos concretos	411
16. Conexión genérica	412
Interfaces genéricas	413
Diseño de la interfaz de usuario	414
Implementación de funciones	415
Ejecución del proyecto	418
17. Acceso a Oracle desde Visual Basic .NET	420
Servidor, servicio y esquema	421
Instalación de Oracle9i	423
Administración del servidor	429
Identificación de servicio y esquema	429
El software cliente	430
Definición del servicio	431
Instalación del proveedor	431
18. Control de transacciones	434
Razón de ser de las transacciones	435
Transacciones en la base de datos	436
Transacciones en Visual Basic .NET	437
Creación del objeto Transaction	438
En la práctica	438
19. Resolución de problemas de concurrencia	442
Políticas de bloqueo y actualización	443
Información de retorno durante la actualización	444
Un primer acercamiento	445
Fusión de los cambios	447
20. Tablas con columnas calculadas	450
Columnas calculadas en la sentencia SQL	451
Añadir columnas a un DataTable	453
Creación de un nuevo DataColumn	453
Creación en fase de diseño	454
21. Almacenamiento y recuperación de imágenes	458
Añadir una columna para la portada	459
Columnas binarias	460
Diseño del formulario Windows	462
Recuperar la imagen de la base de datos	464
Asignar una imagen desde un archivo	465

22. Creación de proveedores ADO.NET	468
Inicio del proyecto	470
FileSystemClientConnection	471
FileSystemClientCommand	474
FileSystemClientDataReader	479
FileSystemClientDataAdapter	485
Prueba del proveedor.....	487
Ejecución de un comando.....	488
Uso del lector de datos	488
Uso del adaptador de datos.....	489
23. Application Blocks para ADO.NET	492
Obtención de Data Access Application Blocks.....	493
Compilación del ensamblado	495
Uso de los métodos SqlHelper	496
Agregar una referencia al ensamblado	496
Uso de los métodos compartidos	497
Documentación adicional.....	498
A. Glosario	500
B. Contenido del CD-ROM	506
Uso de los ejemplos	507
Atención al lector	509
Índice alfabético	511



Introducción

Independientemente de la finalidad concreta que tenga una cierta aplicación, la necesidad de almacenar y recuperar información es una constante casi invariable. En ocasiones esa información se aloja localmente, en un soporte asociado al mismo ordenador en el que se ejecuta la aplicación, mientras que en otras lo hace de forma remota, en algún tipo de servidor. A lo largo del tiempo, desde el nacimiento de los primeros lenguajes de programación hasta la actualidad, los mecanismos utilizados para transferir la información desde su origen hasta la aplicación, o viceversa, han sido diversos, si bien es cierto que en los últimos años el más usado ha sido la *base de datos*. Si bien es cierto que, en un principio, lo más importante era la posibilidad de guardar los datos para recuperarlos posteriormente, ésa es una exigencia básica que ya se da por asumida, pasando a primer plano aspectos como la seguridad e integridad de los datos, la posibilidad de acceder a ellos de manera simultánea desde múltiples puntos, el rendimiento en la ejecución de las operaciones, etc. La expansión de las tecnologías WWW en las aplicaciones actuales, tanto internas como de cara al público, también ha aportado nuevas necesidades, como el acceso a los datos a través de redes públicas, principalmente Internet, o la transformación a y desde estándares como XML.

Las necesidades de las aplicaciones se convierten, también de manera invariable, en trabajo adicional para el desarrollador. Al fin y al cabo, todas esas capacidades han de implementarse de alguna forma. Es preciso conectar con el origen de los datos, transferirlos hasta la aplicación, procesarlos internamente o bien con intervención del usuario y, finalmente, devolverlos a su origen debidamente actualizados. Para efectuar ese trabajo es necesario disponer de soluciones.

Soluciones de acceso a datos

Las soluciones o mecanismos de acceso a datos han sido y, por ahora, seguirán siendo muchos y diversos. Tenemos lenguajes que gestionan directamente sus formatos de archivo, como es el caso de los conocidos como *archivos indexados* de COBOL; motores de acceso a datos que pueden usarse desde distintos lenguajes, como DAO, ADO o BDE; bases de datos que cuentan con su propio lenguaje de programación, desde dBase y Access hasta cualquier RDBMS actual, y plataformas que cuentan con sus propios servicios de datos, como JDBC y ADO.NET.

Normalmente cada una de esas soluciones está asociada a una determinada herramienta de desarrollo o lenguaje, o bien a un modelo de objetos, sistema o plataforma. DAO y ADO, por poner un ejemplo, sólo existen en Windows y, además, sólo pueden ser usados desde lenguajes que contemplan el trabajo con objetos COM, mientras que JDBC es útil sólo en caso de que se trabaje con el lenguaje de programación Java.

Cada una de estas soluciones tiene sus particularidades, problemas, ventajas y requerimientos. La elección de una de ellas suele limitar los orígenes de datos a los que puede accederse, determinar el rendimiento y la escalabilidad de las aplicaciones. No es el objetivo de este libro entrar en los detalles específicos de cada una de ellas sino, como puede imaginar, centrarnos en el estudio de ADO.NET.

ADO.NET

La plataforma Microsoft .NET supone un nuevo universo de posibilidades para los programadores. Pueden elegir el lenguaje de programación que requieran sin, por ello, comprometer los servicios a los que tienen acceso o las posibilidades de compartir código con otros desarrolladores. La CLS representa una interoperabilidad entre lenguajes sin precedentes hasta ese momento. El código MSIL es independiente de procesador y sistema operativo, lo cual permite ejecutarlo en plataformas como Pocket PC, Linux o FreeBSD.

En la plataforma Microsoft .NET existe una biblioteca de clases que representan los servicios de la propia plataforma, entre los cuales se encuentra ADO.NET: la solución de acceso a datos de Microsoft .NET. Estos servicios pueden emplearse desde cualquier lenguaje .NET sin diferencias, si bien en este libro se ha optado por basar los ejemplos en Visual Basic .NET al considerarse el lenguaje más extendido en este momento. El código, no obstante, puede aplicarse a C# y otros lenguajes .NET previa conversión de sintaxis, manteniendo exactamente las mismas clases de objetos, métodos, propiedades y eventos.

ADO.NET es una solución global de acceso a datos diseñada como una evolución de ADO. Decimos global porque puede aplicarse tanto a casos simples, aplicaciones en las que los datos se alojan en un archivo en el propio ordenador; hasta las típicas aplicaciones cliente/servidor, con conexión permanente o no al origen

de datos; aplicaciones en varias capas o *n-tier* y, por supuesto, aplicaciones distribuidas basadas en Internet.

Con ADO.NET es posible acceder a prácticamente cualquier origen de datos, ya sea local o remoto, estructurado o jerárquico, basado en DBMS o no. El rendimiento obtenido es, en la mayoría de los casos, superior a la de otras soluciones existentes en la actualidad. En ADO.NET se ha optado por un nuevo esquema de trabajo con los datos, un esquema en el que se prescinde de elementos como la conexión continua con el servidor u origen de datos, la ejecución de un cursor de acceso a la información o la representación binaria de forma interna. El resultado es un incremento en rendimiento, escalabilidad, compatibilidad y transportabilidad, así como un menor uso de recursos en los servidores.

Como tendrá ocasión de ver en este libro, podemos usar ADO.NET para trabajar con documentos XML, bases de datos Microsoft Access, SQL Server, Oracle y, en general, cualquier origen para el que exista un controlador OLE DB u ODBC lo que, en la práctica, significa prácticamente todos los orígenes posibles.

Visual Studio .NET

Aunque es posible utilizar todos los servicios de ADO.NET sin más ayuda que un editor de textos simple y el compilador de Visual Basic .NET, ejecutable desde la línea de comandos, el entorno de Visual Studio .NET puede, en ocasiones, facilitar de forma notable nuestro trabajo. Dicho entorno cuenta con elementos que, como el **Explorador de servidores**, nos permiten establecer conexiones con orígenes de datos, seleccionar conjuntos de información, definirlos, crear automáticamente los objetos necesarios para acceder a ellos desde programa, etc.

Independientemente de que esté utilizando *Visual Basic .NET Standard* o alguna de las ediciones de *Visual Studio .NET, Professional, Enterprise Developer o Enterprise Architect*, hay ciertas operaciones básicas que siempre tendremos al alcance. Otras, por el contrario, sólo están disponibles en las ediciones superiores.

Una vez que nos hayamos familiarizado con ADO.NET, y sepamos cómo usar sus clases para efectuar todas las operaciones habituales, veremos cómo aprovechar las posibilidades del entorno de Visual Studio .NET.

Objetivos de este libro

El libro que tiene en sus manos no está pensado para aprender a programar con Visual Basic .NET, familiarizarse con el entorno de Visual Studio .NET o bien conocer los servicios básicos de la plataforma .NET para la creación de aplicaciones Windows, aplicaciones y servicios Web. Todos esos conocimientos se asume que el lector ya los tiene, lo cual nos permite centrarnos específicamente en un tema: el acceso a datos con ADO.NET desde Visual Basic .NET. No es éste, por tanto, un

libro adecuado para aquellos que desconocen el lenguaje o el entorno, a los cuales recomendamos títulos como *Programación con Visual Basic .NET* o *Guía práctica para usuarios de Visual Studio .NET*, de la misma editorial y de tipo más genérico.

A diferencia de otros títulos, en éste no se parte de que el lector conoce ADO, el mecanismo de acceso a datos empleado en Visual Basic 6.0 y otras herramientas previas a la aparición de Visual Studio .NET, ni DAO ni ninguna otra solución previa. No encontrará, por tanto, comparativas entre los objetos de ADO y ADO.NET o descripción de procesos tomando como base los de ADO. Con este libro podrá usar ADO.NET sin necesidad de conocer previamente ningún otro mecanismo de acceso a datos, tan sólo necesita saber cómo utilizar Visual Basic .NET para construir aplicaciones Windows y Web.

Para alcanzar estos objetivos es indispensable que su configuración, la del equipo donde va a probar los ejemplos propuestos, cuente con estos elementos:

- Tener instalado *Visual Basic .NET Standard* o algunas de las ediciones de *Visual Studio .NET*. El autor ha empleado la edición *Enterprise Architect* de *Visual Studio .NET*, si bien la mayor parte del contenido es aplicable a cualquier edición, incluida la citada *Visual Basic .NET Standard*.
- Contar con una unidad de CD-ROM desde la cual recuperar los ejemplos del disco que acompaña al libro, copiándolos en su sistema.
- Disponer del software cliente y servidor de datos usado en los ejemplos, entre otros Microsoft Access, Microsoft SQL Server 7/2000, Oracle 8i e InterBase. En los capítulos respectivos se tratarán las bases de estas aplicaciones y, en algunos casos, se indicará cómo obtenerlas, instalarlas y configurarlas en su sistema.

Estructura del libro

Al redactar este libro se ha tenido como primer punto de mira el contenido didáctico, por ello los fundamentos y conocimientos se van abordando de manera escalonada, capítulo a capítulo, asumiendo que ésa será la secuencia inicial de lectura. No obstante, muchos capítulos también pueden usarse a modo de consulta o referencia, una vez que haya adquirido la visión global de los temas que se tratan.

El libro está dividido en cuatro partes: *Sentar las bases*, *ADO.NET*, *Visual Studio .NET* y *Resolución de casos concretos*. La primera de ellas, compuesta de tres capítulos, nos servirá para conocer la terminología que va a utilizarse a lo largo del libro, así como unos fundamentos del lenguaje SQL y ciertas aplicaciones que pueden actuar como orígenes de datos.

La segunda parte se centra en el estudio de ADO.NET, sus clases y la forma de usarlas para conectar con un origen de datos, recuperar información, manipularla y devolverla. Aprenderá a acceder a distintos orígenes de datos, establecer relaciones y restricciones, trabajar con conjuntos de datos o usar XML con ADO.NET.

En la tercera parte se tratan las herramientas visuales de datos de Visual Studio .NET, mediante las cuales pueden facilitarse muchas de las tareas que, en la parte anterior, se han efectuado manualmente mediante la escritura de código. También se estudian los componentes con vinculación a datos que simplifican la construcción de interfaces de usuario.

Por último, en la cuarta parte, encontrará una serie de capítulos en los que, de forma específica y mediante un ejemplo, se aborda la resolución de un cierto caso: la ejecución de un procedimiento almacenado en SQL Server, la creación de una columna calculada en un conjunto de datos o el almacenamiento y recuperación de imágenes, son algunos de los temas estudiados.

Nota

Conforme vaya leyendo los diferentes capítulos de este libro, irá encontrando acrónimos o siglas diversas, como por ejemplo ADO, DBMS, RDBMS o COM. El significado de todos ellos se encuentra en un apéndice al final del libro, evitando así su introducción en el texto cada vez que aparezca el término.

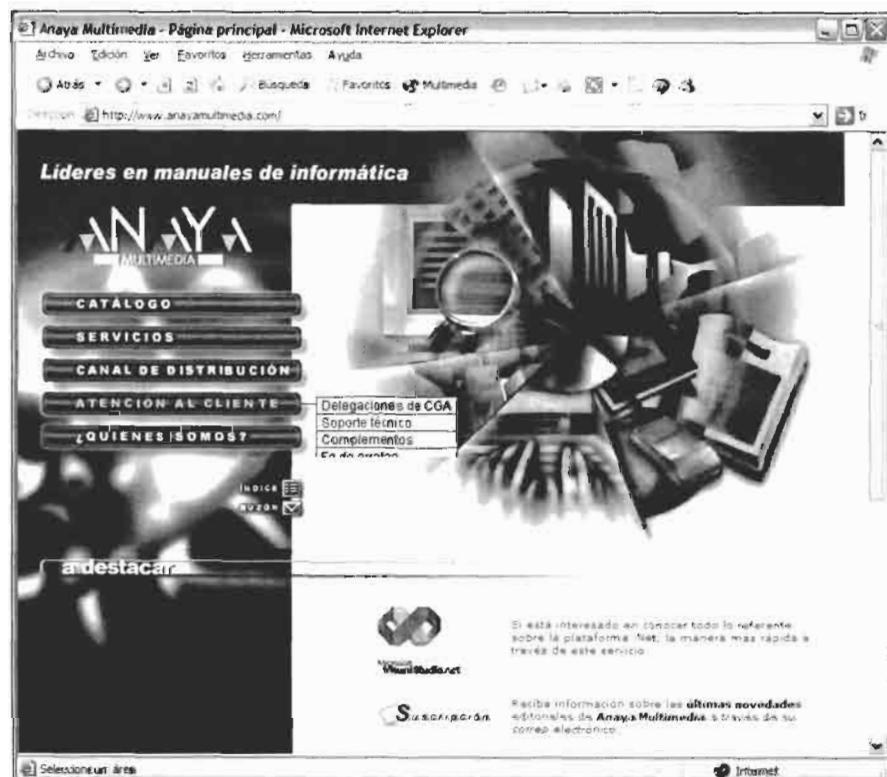


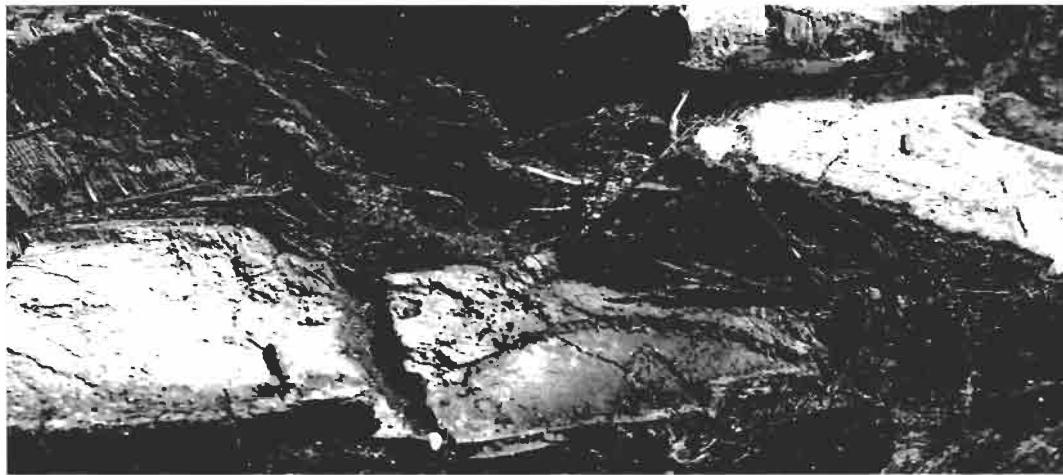
Figura I.1. Sede Web de la editorial Anaya Multimedia

Ayuda al lector

En caso de encontrar algún problema con este libro o el CD-ROM que le acompaña, el lector puede recurrir tanto a la editorial como directamente al autor.

Desde la Web de la editorial, en <http://www.AnayaMultimedia.com>, puede acceder a la sección **Atención al cliente** para acceder al soporte técnico, así como a las secciones de **Complementos** y **Fe de erratas**.

Si desea ponerse en contacto con el autor para plantear exclusivamente alguna cuestión relativa a este libro, puede hacerlo a través de la Web *Torre de Babel*, en <http://www.fcharte.com>.



1

Terminología y conceptos

Este libro está dirigido a desarrolladores que necesitan utilizar los servicios de ADO.NET para trabajar con bases de datos, documentos XML y, en general, cualquier origen de datos. Como programadores, el autor asume que ya se conocen los términos habituales en cualquier lenguaje de programación: variable, bucle, sentencia, expresión condicional, compilación, etc. No se asume, por el contrario, ningún conocimiento relativo al trabajo con datos, campo que también cuenta con sus propios conceptos y terminología específica.

A pesar de no ser éste un título dedicado a los fundamentos de tratamiento de datos, tema sobre el que encontrará libros monográficos, en este capítulo se abordan los elementos que podrían considerarse básicos: ¿qué es un DBMS o el lenguaje SQL?, ¿cómo se estructuran los datos en filas y columnas? o ¿qué es XML? son algunos de los contenidos de este capítulo.

El objetivo no es otro que facilitarle los conocimientos indispensables, conceptos y términos, para que pueda leer cómodamente los demás capítulos de este libro. Posteriormente, dependiendo de su interés y necesidades, puede recurrir a textos que traten con mayor profundidad la teoría del tratamiento de bases de datos.

Orígenes de datos

Los servicios de acceso a datos .NET, conocidos como ADO.NET, pueden ser usados para operar sobre orígenes de datos diversos, no exclusivamente sobre lo

que se conoce como *bases de datos*. Un origen de datos puede ser una base de datos SQL Server, un documento XML, una hoja de cálculo Microsoft Excel, el Directorio Activo de Windows o, en general, cualquier recurso para el que exista un proveedor de datos .NET.

El *origen de datos*, por tanto, es el recurso, ya sea local o remoto, del que va a extraerse o en el que se va a introducir información, teniendo un sentido mucho más amplio que el de *base de datos*. En el texto encontrará ambas expresiones, la primera cuando se quiera dar un sentido general y la segunda al hacer referencia concreta a un DBMS.

Utilizando el paradigma del origen de datos, sin asumir su localización, estructura ni naturaleza, ADO.NET es realmente un mecanismo de acceso a datos universal. Esto tiene como ventaja fundamental el ahorro de trabajo para el desarrollador, al no tener que recurrir a sistemas diferentes dependiendo de los datos que se pretendan manipular o recuperar.

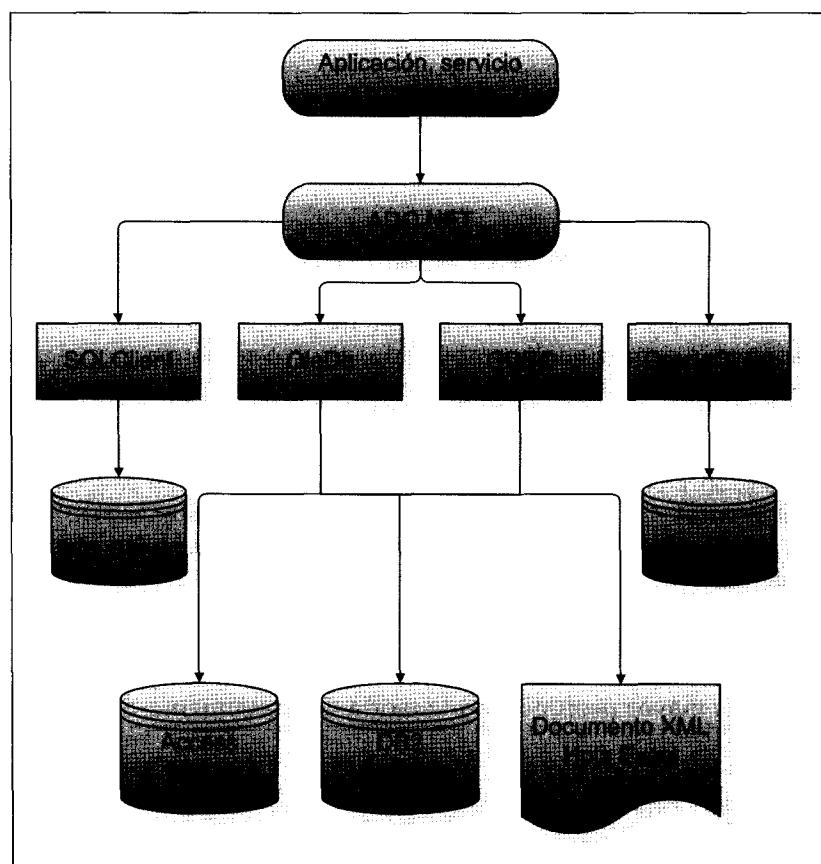


Figura 1.1. ADO.NET actúa como intermediario entre las aplicaciones y los posibles orígenes de datos, utilizando para ello diversos proveedores de acceso

Nota

Los servicios de ADO.NET pueden utilizarse desde una aplicación con interfaz de usuario, un componente que se ejecuta en un servidor, un servicio Windows o Web. En general, es posible recurrir a ADO.NET desde cualquier punto de una aplicación que se ejecute en un sistema donde esté disponible la plataforma .NET.

En el tercer capítulo se entrará a estudiar con mayor detalle la naturaleza de los orígenes de datos, describiendo básicamente aquellos más conocidos a los que podemos acceder desde Visual Basic .NET usando ADO.NET.

Bases de datos

El origen de datos por excelencia es la *base de datos*. Una base de datos puede definirse como uno o más archivos en los que se almacenan los datos y toda la información que describe su estructura, fundamental para poder operar sobre ellos. Las bases de datos pueden ser de distintas categorías, siendo el tipo más habitual el de las bases de datos relacionales. Por otra parte, las bases de datos también se dividen en dos clases: bases de datos de escritorio y servidores de bases de datos. Las primeras están dirigidas principalmente a usuarios con ciertos conocimientos que le permiten crear y mantener sus estructuras de información, razón por la cual estas bases de datos cuentan con una interfaz de usuario bastante completa y amigable. Es el caso de Microsoft Access, dBase o Paradox. También se caracterizan por no contar con un *middleware* que permita tener el software en una máquina y los datos en otra distinta, es decir, son bases de datos locales.

Se llama *servidor de datos* a una aplicación que se encarga de manipular físicamente los datos, ocupándose de su almacenamiento y recuperación de los archivos en que se encuentran. De esta forma las aplicaciones no tienen que conocer la estructura de dichos archivos, limitándose a solicitar al servidor las operaciones que quiere efectuar. También suele conocerse como *servidor de datos* al ordenador en el que se ejecuta dicha aplicación.

Aunque una base de datos de escritorio o local puede colocarse en una unidad compartida de red, facilitando el acceso a ella desde diferentes puestos, no es la configuración más adecuada para construir un acceso multiusuario. En este caso resultan mucho más eficientes y seguros los servidores de bases de datos.

DBMS y RDBMS

Los servidores de bases de datos son sistemas de administración de información o DBMS, aplicaciones cuyo objetivo no es sólo almacenar y recuperar datos,

sino también facilitar la manipulación de éstos de la forma más eficiente y segura. Un servidor de datos debe elaborar la información que va a devolver al cliente a partir de los datos recuperados del sistema de archivos, usando para ellos diferentes estructuras físicas, así como asegurar su integridad verificando restricciones y utilizando transacciones, temas que se tratarán de inmediato.

Aunque en un principio surgieron DBMS de distintos tipos, según la estructura con la que se almacenaba la información, los más extendidos y conocidos son los RDBMS, llamados así porque los datos se estructuran con ciertas relaciones entre ellos, simplificando la recuperación y el tratamiento y evitando la repetición innecesaria de información. Algunos ejemplos de RDBMS son los ya mencionados SQL Server, Oracle, DB2 o InterBase.

Nota

También se conoce a los DBMS por sus siglas en nuestro idioma, SGBD, aunque su uso es menos habitual.

Arquitectura cliente/servidor

Durante años, las bases de datos se han utilizado en sistemas que se ajustaban a una arquitectura conocida como *cliente/servidor*. En ella los datos residen en un ordenador que actúa como servidor, ejecutando el software que denominábamos antes *servidor de datos*. Los usuarios, desde ordenadores remotos, se sirven de un software *cliente* para comunicarse con el servidor de datos. Ese software cliente es específico para cada servidor de datos existente.

Supongamos que está utilizando SQL Server como servidor de datos, estando instalado dicho software en una máquina que reside en un centro de proceso de datos. Desde su ordenador, ubicado en otra planta del mismo edificio, se comunicaría con ese servidor mediante el software cliente de SQL Server que, lógicamente, debería estar instalado en su máquina. Dicho software cliente no serviría para comunicar con un servidor Oracle o DB2, por poner un caso, teniendo que disponer del software cliente que corresponda en cada caso.

Nota

Las bases de datos de escritorio, como los citados Access, dBase y Paradox, no son servidores de datos y, por tanto, no pueden utilizarse en una arquitectura cliente/servidor real, a pesar de la posibilidad indicada antes de colocar los archivos de datos en una unidad compartida de red. La diferencia está en que cada máquina accedería a sus datos directamente, como si los archivos se encontrasen en ella, mientras que en una arquitectura cliente/servidor real el

cliente nunca accede directamente a los datos, sino que delega ese trabajo en el servidor.

En la figura 1.2 puede ver representada una típica configuración cliente/servidor. Concretamente aparece un servidor de datos, en la parte inferior, y varios clientes, en la superior. Aunque en dicha imagen no se ha representado, se supone que tanto clientes como servidor tienen sus respectivos paquetes de software instalados.

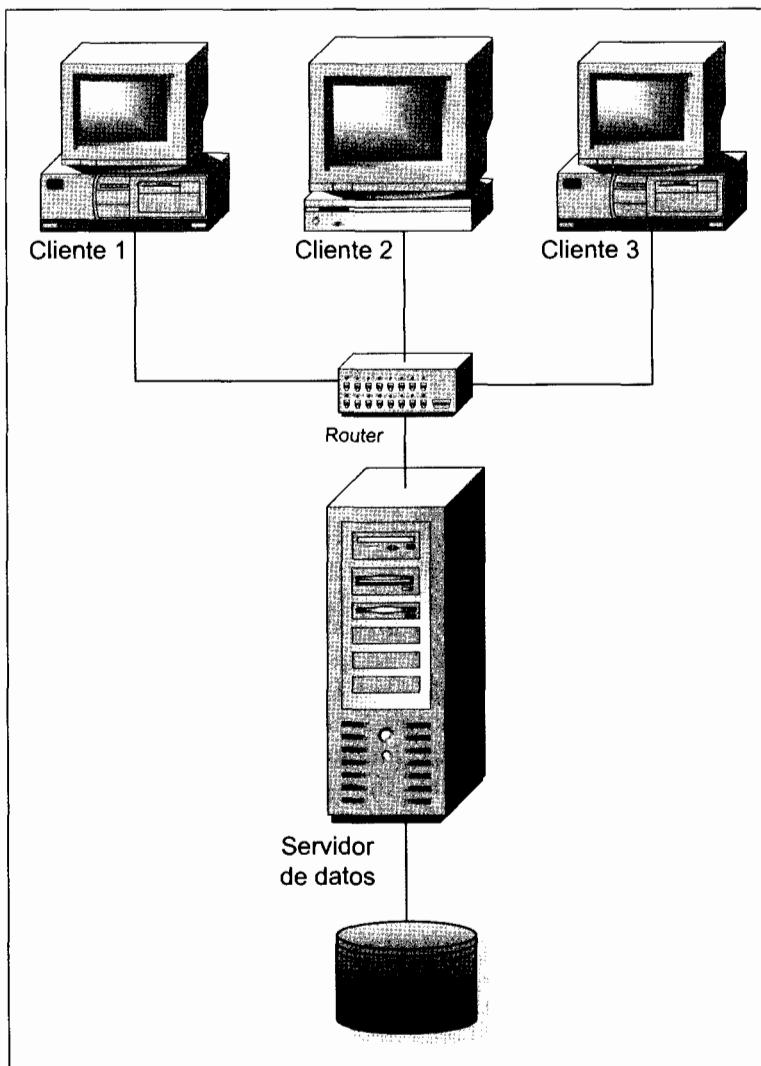


Figura 1.2. Configuración cliente/servidor de acceso a una base de datos

Arquitecturas *n-tier*

Partiendo del modelo cliente / servidor, descrito brevemente en el punto anterior, han surgido otros conocidos genéricamente como *n-tier*, siendo el más habitual el *three-tier* o de *tres capas*. En dicho modelo los clientes no se comunican directamente con el servidor de datos, sino que entre ellos se interpone un nuevo servidor: el de aplicaciones. La figura 1.3 es una representación visual de este modelo.

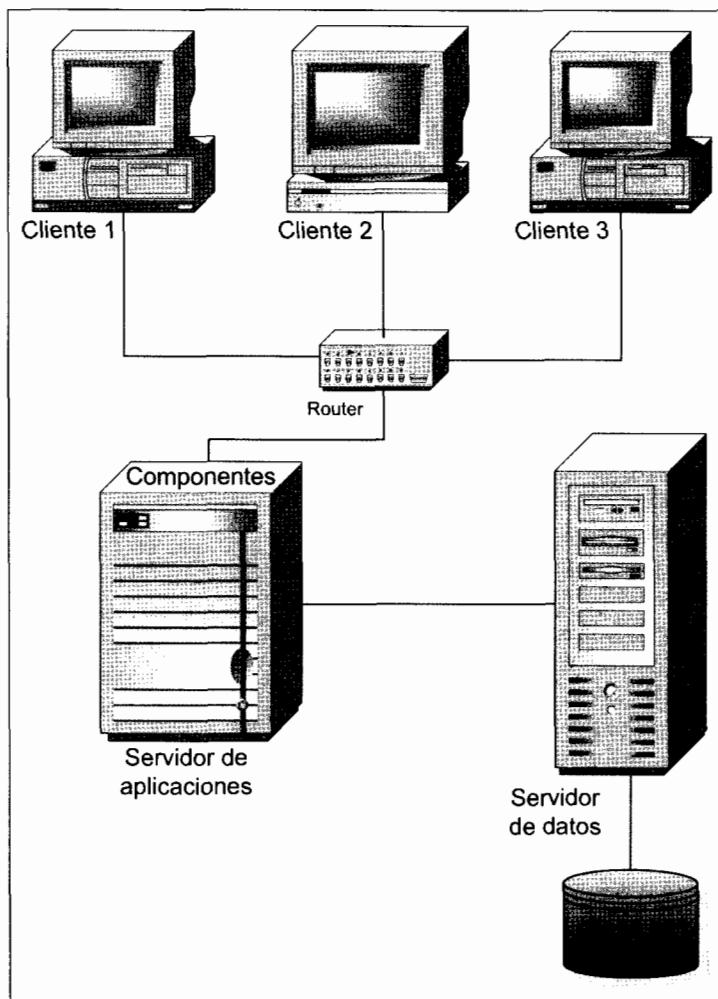


Figura 1.3. Un sistema distribuido en tres capas

En este modelo los clientes no tienen acceso directo a los datos, trabajo que queda en manos de los componentes que se ejecutan en el servidor de aplicaciones.

Dichos componentes, además, alojan la lógica de proceso de la información, lo que normalmente se conoce como *reglas de negocio*, tarea que en el modelo cliente/servidor recaía en la aplicación del cliente. De esta forma, el cliente queda relegado a una simple interfaz de usuario, ya sea nativa en un ordenador, un documento en un navegador Web o desde un dispositivo móvil.

La ventaja principal es que resulta más fácil llegar a clientes heterogéneos y dispersos, interconectados no en redes de la propia empresa sino a través de Internet. En cuanto a nosotros respecta, la única diferencia sería que la aplicación estaría dividida en dos apartados: la interfaz del usuario, por una parte, y los componentes que se ejecutarían en el servidor de aplicaciones y accederían a los datos, por otra.

Servicios de acceso a datos

Aunque ciertos tipos de información son accesibles para el usuario mediante aplicaciones especializadas, por ejemplo Microsoft Excel para las hojas de cálculo o Paradox para las bases de datos propias de Paradox, en la mayoría de los casos el usuario necesitará una solución a medida de sus necesidades, en lugar de otra general. No sería lógico, por ejemplo, que los usuarios utilizasen el software de administración para operar directamente sobre un servidor de datos como SQL Server u Oracle. Aparte de complejo para aquellos que no tienen conocimientos de bases de datos y del lenguaje SQL, esa posibilidad pondría en peligro la propia seguridad de los datos.

Gracias a esa necesidad de los usuarios tenemos trabajo los desarrolladores, creando aplicaciones según las características, conocimientos y necesidades de nuestros clientes. Para que nuestros programas puedan trabajar con los archivos donde reside la información, no obstante, necesitaremos diversas herramientas, entre ellas uno o varios servicios de acceso a datos. Por ejemplo un componente para poder operar sobre las hojas Excel, otro para comunicarse con bases de datos, un tercero para leer y escribir documentos XML, etc.

Aquí es donde entra en escena ADO.NET, nombre con el que se denomina genéricamente a los servicios de acceso a datos de la plataforma Microsoft .NET. ADO.NET, además, es una solución global para el acceso a la información, no necesitamos otros servicios según el tipo de datos con los que vayamos a trabajar. Como se indicaba anteriormente, con ADO.NET podemos tanto operar sobre archivos XML, documentos Excel, bases de datos locales y remotas, etc.

Proveedores y controladores

Nosotros contamos con los servicios de ADO.NET para acceder a los orígenes de datos, utilizando para ello un modelo de objetos y componentes que tendrá ocasión de conocer en un capítulo posterior. ADO.NET, a su vez, precisa de otros

elementos para poder efectuar su trabajo, entre ellos los proveedores y los controladores.

Un proveedor ADO.NET es una implementación específica de una serie de componentes que facilitan el acceso a un determinado origen de datos. Visual Basic .NET incorpora por defecto dos proveedores de datos .NET: uno para SQL Server y otro capaz de emplear cualquier controlador OLE DB. Hay disponibles dos más, que instalará posteriormente, uno para Oracle y otro dirigido al uso de un controlador ODBC.

Los proveedores .NET aparecen, desde el punto de vista del desarrollador, como una serie de definiciones de clases alojadas en un ámbito con nombre o *namespace*, clases que pueden usarse para acceder a un determinado origen de datos. El proveedor de datos para SQL Server y el de Oracle son específicos, comunicándose directamente con el software cliente de esas dos bases de datos sin intermediario alguno. Los proveedores OLE DB y ODBC, por el contrario, son de tipo genérico, diseñados para aprovechar todos los controladores que ya hay disponibles de esos dos tipos.

Suponga que quiere, desde una aplicación propia, recuperar datos de una hoja de cálculo Excel o comunicarse con una base de datos IBM DB2 o InterBase. No hay proveedores .NET específicos para estos tipos de datos, al menos no por el momento, y por ello hay que recurrir a las soluciones genéricas. Mediante un controlador OLE DB se puede acceder a Excel, y mediante ODBC a IBM DB2 o InterBase. En estos casos emplearíamos un módulo o componente de software adicional al proveedor OLE DB u ODBC: el controlador específico del origen al que va a accederse.

En la figura 1.4 puede ver representados dos supuestos en los que un cliente necesita acceder a dos orígenes de datos diferentes: una base de datos SQL Server y una base de datos IBM DB2. En el primer caso, puesto que existe un proveedor .NET específico, el cliente tan sólo precisa el software cliente de SQL Server y ya puede comunicarse, mediante una infraestructura de red, con el servidor. En el segundo, por el contrario, no existe ese proveedor específico, pero sí un controlador ODBC que sabe cómo *hablar* con el software cliente de DB2. Utilizamos, por tanto, el proveedor genérico ODBC de .NET que, a su vez, empleará el controlador ODBC adecuado.

Ni que decir tiene que el acceso mediante proveedores genéricos implica más carga de proceso y, por tanto, un menor rendimiento en la aplicación. En algunas ocasiones, sin embargo, puede ser la única vía para poder llegar a una cierta información.

Lenguajes de comunicación

Aparte de los componentes ADO.NET, que facilitan la comunicación con los orígenes de datos y ejecución de ciertas operaciones, en ocasiones también necesitaremos conocer algún lenguaje específico según la naturaleza del origen de datos.

Mediante XPath, por ejemplo, podría efectuarse una selección de datos en un documento XML.

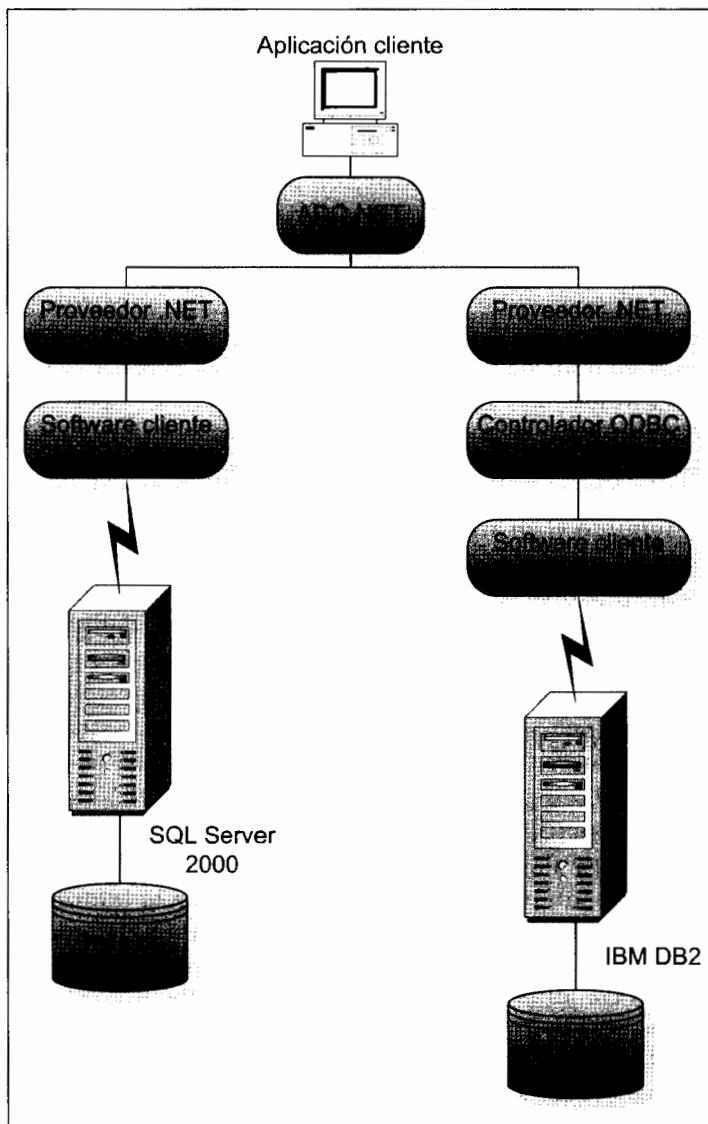


Figura 1.4. Acceso a dos orígenes de datos diferentes

El lenguaje de consulta de datos por excelencia entre los RDBMS es SQL, creado hace algo más de veinte años y conocido originalmente como SEQUEL. Realmente SQL se compone de varios sub-lenguajes, principalmente DDL y DML. El primero de ellos se utiliza para definir las estructuras de los datos que se manipulan con el

segundo. Conocer el lenguaje SQL es fundamental para poder trabajar con bases de datos, por ello en el siguiente capítulo podrá encontrar una introducción a DDL y DML.

Además de para manipular la información, la mayoría de los RDBMS también emplean SQL para otros fines como la implementación de procedimientos almacenados. En realidad, cada fabricante de un producto RDBMS cuenta con su propio derivado de SQL para efectuar esas tareas. SQL Server, por ejemplo, cuentan con el lenguaje T-SQL, mientras que Oracle utiliza PL/SQL. También encontrará en el capítulo siguiente una breve introducción a dichos lenguajes.

Estructura de los datos

Ciñéndonos ya a las bases de datos relacionales, posteriormente en otros capítulos nos ocuparemos del resto de orígenes de datos, encontramos una serie de términos y conceptos que también es preciso conocer. Los RDBMS introducen la información en tablas, que pueden ser consideradas como conjuntos de entidades cada una de las cuales tiene una serie de atributos. Los atributos almacenan datos en un cierto dominio y, en algunas ocasiones, sirven para establecer relaciones con otras entidades.

Los puntos siguientes son un recorrido rápido por los conceptos más importantes relativos a la estructura de la información en un RDBMS, independiente de cuál sea éste.

Entidades y atributos

La información almacenada en una base de datos pertenece generalmente a objetos reales que es posible identificar fácilmente. Esos objetos cuentan con una serie de propiedades. Por ejemplo, un libro tiene un título, una editorial, un autor, un ISBN, un precio, un número de páginas, etc. Toda la información de ese objeto formaría una entidad, mientras que cada dato sería un atributo de la entidad.

Los atributos de una entidad se reparten en una fila, ocupando cada atributo una columna. Al conjunto de varias filas, cada una con idénticos atributos pero diferentes valores, es a lo que se conoce como *conjunto de entidades*, con estructura de tabla bidimensional. En la figura 1.5 puede ver un conjunto de entidades formado por tres entidades, cada una de las cuales cuenta con cuatro atributos. Observe que todas las entidades que pertenecen a un conjunto tienen los mismos atributos, aunque sus valores sean diferentes. Es decir, todos los libros tienen un título, una editorial, un ISBN y un número de páginas, aunque el título de cada uno sea distinto al igual que el ISBN o el número de páginas.

En un RDBMS ese conjunto de atributos sería una *tabla*, cada entidad sería una *fila* y cada atributo una *columna*. Al trabajar con bases de datos de escritorio es habitual llamar *campo* a cada atributo y *registro* a cada entidad.

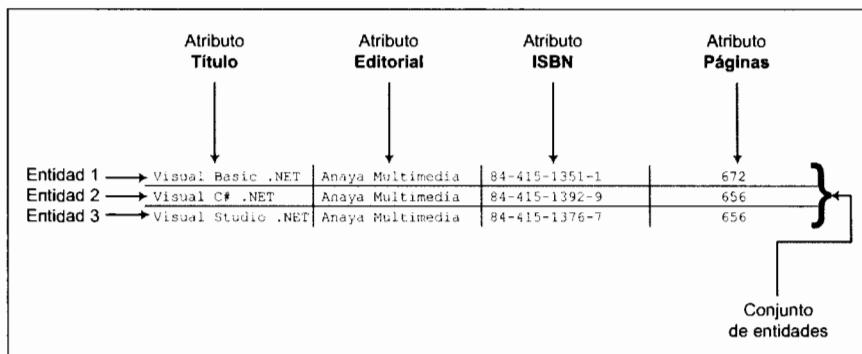


Figura 1.5. Conjunto de tres entidades con cuatro atributos cada una

Dominios y restricciones

Al conjunto de valores que puede tomar un atributo es a lo que se llama *dominio*. El atributo ISBN del ejemplo mostrado en la figura 1.5, por poner un ejemplo, tendrá un dominio que limite la introducción a 13 caracteres ajustados a un cierto formato. De manera análoga, el atributo Páginas contaría con un dominio que tan sólo permitiese la introducción de valores numéricos comprendidos entre 100 y 2000, por ejemplo.

Los dominios, cuando se definen asociados a columnas en un RDBMS, impiden la introducción de valores incorrectos en la base de datos, asegurando así la integridad de la información. Al tiempo, se evita que la aplicación tenga que estar comprobando continuamente la validez de los valores introducidos por los usuarios.

Una *restricción* se define como el conjunto de reglas de validación de un cierto atributo, reglas que pueden ser mucho más complejas que un dominio. Mediante una restricción es posible, por ejemplo, evitar que se introduzca en una tabla información relativa a un elemento que no existe en otra. En realidad, los dominios son un tipo de restricción.

Identidad de una entidad

Por norma, las entidades suelen contar con un atributo al que se aplica una restricción que le obliga a contener un valor único entre el conjunto de entidades, utilizándose como atributo de *identidad* de cada una de las entidades. Es el clásico *código* que se asocia a las filas, normalmente un número consecutivo generado por la propia base de datos, y que permite identificarlas de manera única.

Que las entidades tengan un atributo de identidad es algo indispensable en los RDBMS, ya que facilita el establecimiento de relaciones entre distintos conjuntos de entidades y hace posible la integridad referencial mediante restricciones de claves externas. Además, las reglas de normalización comentadas brevemente más adelante exigen la existencia de ese atributo de identidad.

En el ejemplo mostrado en la figura 1.5, el atributo ISBN podría actuar como identidad de cada entidad ya que es irrepetible. En el caso de que las entidades no dispongan de un atributo de este tipo se añadiría un atributo artificial que permitiese la identificación.

Relaciones entre conjuntos de entidades

Los conjuntos de entidades que conforman una base de datos, en forma de tablas, generalmente guardan ciertas relaciones entre ellos. Las reglas de normalización de las bases de datos relacionales, que conocerá básicamente en un punto posterior, persiguen evitar la repetición de datos en las tablas a fin de ahorrar espacio. No tiene sentido, por ejemplo, que en el conjunto de entidades de la figura 1.5 se repita el nombre de la editorial en cada entidad.

En la figura 1.6 se pueden ver dos conjuntos de entidades: uno relativos a libros y otro a editoriales. Observe que el segundo atributo de cada libro ahora no es el nombre de la editorial, sino un número que la identifica. Ese número es el atributo identidad de las entidades del segundo conjunto.

Visual Basic .NET	1	84-415-1351-1	672
Visual C# .NET	1	84-415-1392-9	656
Visual Studio .NET	1	84-415-1376-7	656

1	Anaya Multimedia
2	APress
3	O'Reilly

Figura 1.6. Relación entre dos conjuntos de entidades

El atributo Editorial de la primera entidad que hay en el primer conjunto indica que la editorial es el número 1, estableciendo así una relación con la primera entidad del segundo conjunto. El número 1 es el atributo identidad o *clave primaria* de cada entidad editorial, mientras que actúa como *clave externa* en la entidad correspondiente a cada libro. Estas claves también suelen conocerse como PK y FK. Mediante una restricción de integridad referencial se evitaría que el usuario pudiera introducir en una entidad de la tabla de libros un código de editorial que no existiese en la segunda tabla.

De manera análoga, otra restricción podría impedir la eliminación de una editorial del segundo conjunto mientras existiesen en el primero entidades que hicieran referencia a ella.

Las relaciones entre tablas pueden ser de diversos tipos, siendo las más habituales las conocidas como *uno-a-uno* y *uno-a-muchos*. En el primer caso a cada entidad del primer conjunto le corresponde sólo una entidad del segundo. Es lo que ocurre en el caso de la figura 1.6, en el que cada fila de la tabla de libros apunta a una sola fila de la tabla de editoriales. Si invirtiésemos la relación (véase figura 1.7), por el

contrario, nos encontraríamos con el segundo caso, por cada fila de la editorial de tablas tendríamos múltiples filas de la tabla de libros que le corresponden, en este ejemplo concreto tres.

1	Anaya Multimedia			
2	APress			
3	O'Reilly			
<hr/>				
→	Visual Basic .NET	1	84-415-1351-1	672
→	Visual C# .NET	1	84-415-1392-9	656
→	Visual Studio .NET	1	84-415-1376-7	656

Figura 1.7. Relación de tipo uno-a-muchos

Nota

Al hablar de relaciones suele usarse con cierta frecuencia el término *maestro-detalle*. Hace referencia a un tipo de relación en la que una tabla actúa como maestra o primaria y otra como tabla de detalle o secundaria, dando lugar a una relación *uno-a-muchos*.

Índices

Al trabajar con tablas de tamaño considerable, con muchos miles o, incluso, millones de filas, el proceso de recuperación de datos puede verse afectado de manera considerable. Es lógico si tenemos en cuenta que para, por ejemplo, encontrar todos los libros de una editorial, sería preciso recorrer la tabla de libros completa, de principio a fin.

Un mecanismo común para acelerar esa operación son los índices. Un índice es una lista de claves con una estructura tal que el servidor puede realizar búsquedas de forma muy rápida en ella.

Las claves pueden estar formadas por el contenido de una o varias columnas de una tabla. Además de acelerar la búsqueda de información, un índice también puede ser utilizado para establecer el orden en el que se almacenarán las filas en una tabla.

Cada vez que se inserta, modifica o elimina una tabla de la cual depende uno o varios índices, el servidor tiene que actualizar no sólo la tabla de datos sino también todos los índices que existan a fin de que sean consistentes con la información actual. Es fácil deducir que cuántos más índices existan más tiempo será necesario para efectuar cualquiera de esas operaciones.

En capítulos posteriores podrá conocer los fundamentos de la definición de procedimientos almacenados en algún RDBMS, así como su ejecución desde una aplicación propia a través de ADO.NET.

Normalización

El proceso de normalización de una base de datos se compone de varios pasos que se apoyan en un conjunto de reglas formales relativamente estrictas. De lo que se trata, básicamente, es de racionalizar tanto el contenido de las tablas como las relaciones existentes entre ellas, persiguiendo la menor repetición de información y la mayor flexibilidad.

En una base de datos normalizada lo habitual es que exista un mayor número de tablas que en otra que no lo está. En consecuencia, las tablas normalizadas suelen ser más pequeñas, con menos atributos, que las tablas no normalizadas. Dos de las reglas fundamentales de este proceso son:

- Cada entidad debe contar siempre con un atributo identidad que actúe como clave primaria y única, facilitando de esa manera la identificación inequívoca de cada una de las filas de una tabla. Tal y como antes se indicaba, el atributo identidad es un dato, real o creado a propósito, que nunca se repite entre las entidades.
- Otra norma común del proceso de normalización es evitar la duplicidad de datos, tanto dentro de una misma tabla como entre las tablas que componen la base de datos. Es lo que ocurría en el ejemplo representado en la figura 1.5 y que hemos solucionado en la figura 1.6 normalizando la base de datos.

Aunque seguramente los dos aspectos más importantes del proceso de normalización sean éstos, existen otros que también deberían considerarse para conseguir el objetivo de un diseño racionalizado y lógico. Es importante que no existan relaciones demasiado complejas entre tablas que provoque dificultades a la hora de recuperar información, en forma de consultas difíciles de expresar por parte del programador o usuario y de ejecutar por parte del RDBMS. Otra regla indica que todos los atributos que se empleen para establecer relaciones entre tablas deben ser claves o formar parte de un índice para acelerar su proceso.

Transacciones

Al operar sobre un RDBMS del tipo Oracle, SQL Server, DB2, Sybase o InterBase, los que podríamos considerar *servidores de datos*, todas las actuaciones que implican manipulación de la información, no sólo recuperación de datos, tienen lugar en el ámbito de una *transacción*. Esto es especialmente cierto en aquellos casos en los que la información manipulada se aloja en dos o más tablas y, por tanto, podría generarse una inconsistencia entre los datos.

Suponga que crea una aplicación para una entidad bancaria en la que el usuario, por ejemplo el cajero de una oficina, introducirá datos sobre operaciones bancarias. Atendiendo a un cliente que quiere efectuar una transferencia desde su cuenta a la de un proveedor, el cajero introduce los dos códigos de cuenta y el importe a transferir, entre otros datos. Al pulsar un botón su aplicación deduce el importe de la cuenta del cliente y, cuando va a sumarlo a la cuenta del proveedor, se produce un fallo en el sistema. ¿Qué ha ocurrido con la operación? En ese momento ya no está el dinero en la cuenta del cliente, pero tampoco en la del proveedor. Simplemente ha desaparecido o quizás esté en *el limbo*.

Obviamente, esa situación no debería darse nunca. Al iniciar las operaciones sobre la base de datos la aplicación debería iniciar una transacción. Ésta asegura que los datos que se manipulen no son escritos de manera inmediata en la base de datos, sino en un espacio temporal. Sólo cuando se han finalizado los cambios en todas las tablas afectadas se termina la transacción, momento en el que dichos cambios se confirman.

Para que este sistema sea realmente útil es necesario que el sistema de transacciones del RDBMS cumpla con las propiedades conocidas como ACID:

- La *atomicidad* de la transacción hace que todas las operaciones que comprende sean vistas por el RDBMS como una sola, de tal manera que o se efectúa o se rechaza completa, no existiendo la posibilidad de que el trabajo quede a medias.
- Mediante la *consistencia* se garantiza el cumplimiento de las reglas que llevan asociadas las operaciones introducidas en la transacción, tales como restricciones, ejecución de desencadenadores, etc.
- Que una transacción tenga entre sus propiedades el *aislamiento* implica que las operaciones intermedias que se efectúen, antes de finalizar la transacción, no son visibles para nosotros y, análogamente, las realizadas en esta transacción no sean visibles en otras hasta que se confirme. Así se consigue una independencia total en el trabajo de múltiples transacciones concurrentes.
- Por último tenemos la *durabilidad* o *persistencia*, la propiedad que da la garantía de que la transacción, una vez concluida, no se perderá a pesar de los problemas posteriores que pudiesen encontrarse.

En el supuesto anterior de la transferencia bancaria, si se produjese un fallo en el punto indicado la transacción no se finalizaría y quedaría en un espacio conocido como *limbo*, de la cual se recuperaría posteriormente.

Las transacciones se confirman o se rechazan. La *confirmación* de una transacción implica su finalización, así como la conversión en definitivos de los cambios provisionales que se hubiesen efectuado. De *rechazarse*, la transacción dejaría en su estado original todos los datos afectados, finalizando también la transacción.

Con ADO.NET, como veremos en su momento, generalmente no tendremos que preocuparnos de iniciar y finalizar las transacciones de forma explícita, ya que hay un mecanismo que se encarga de hacer ese trabajo por nosotros.

XML

Seguramente ya sepa qué es XML y cuál es su finalidad, ya que se trata de un lenguaje que en los últimos años ha encontrado aplicación en prácticamente todos los campos de la informática. Al trabajar con ADO.NET no sólo podremos operar sobre documentos XML sino que, además, los conjuntos de datos en ADO.NET se almacenan internamente en dicho formato.

XML es un lenguaje que se centra en la definición de la estructura de los datos, en contraposición a otros, como es el caso de HTML, que se ocupa no de los datos en sí sino de su representación visual. En XML es posible definir marcas propias según se precise, facilitando así la creación de estructuras de información a medida. El beneficio es que XML se almacena y transfiere a través de redes no en un formato binario, como venía siendo habitual en cualquier medio para el almacenamiento de datos, sino como texto simple, facilitando así la comunicación entre aplicaciones y sistemas sin importar el tipo de procesador, sistema operativo o lenguaje de programación empleados.

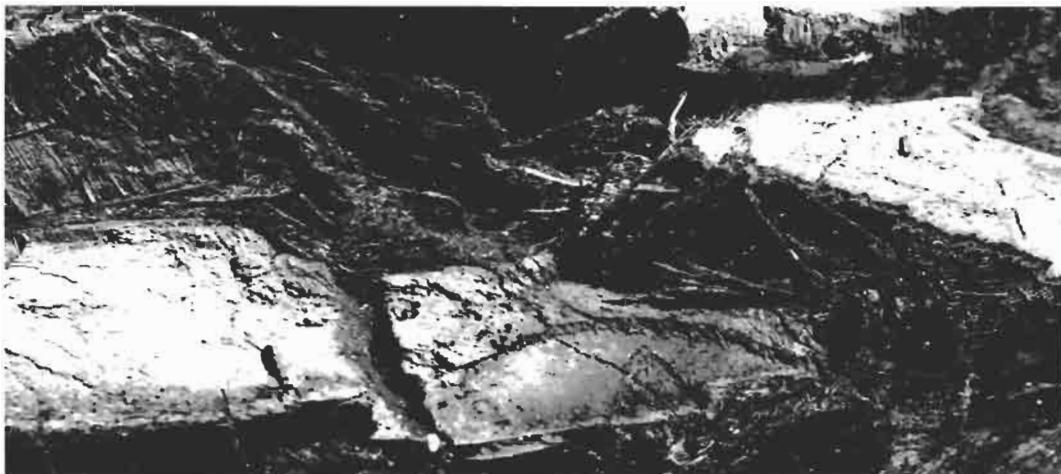
A pesar de su naturaleza, XML es un formato para el almacenamiento de datos con definición implícita, no para la presentación de esos datos. Es muy fácil, no obstante, definir una hoja de estilo XSL y aplicarla a un documento XML para obtener una representación visual de la información. De la misma forma, también se pueden definir esquemas XSD para asegurar que la estructura de los documentos es correcta.

En la plataforma .NET existen servicios específicos para trabajar con documentos XML. Como veremos posteriormente, también podemos actuar sobre ellos mediante ADO.NET, como si fuesen orígenes de datos locales.

Resumen

Al finalizar el primero de este trío de capítulos introductorios ya conoce una parte del vocabulario que usaremos en el resto del libro, así como conceptos fundamentales que puede ampliar recurriendo a otras fuentes específicas.

En la mayoría de ejemplos que se propondrán posteriormente trabajaremos sobre una arquitectura cliente/servidor o de tres capas, aunque también se abordará el uso de orígenes de datos locales como documentos XML. También en los capítulos siguientes tendrá oportunidad de conocer SQL y aprender a definir bases de datos y sus distintos elementos en ciertos sistemas RDBMS.



2

SQL

y sus dialectos

A pesar de que nuestro papel en la empresa esté más cerca, o sea puramente, del desarrollador que del administrador de bases de datos, lo cierto es que conocer el lenguaje SQL resultará totalmente imprescindible.

ADO.NET, como verá en su momento, cuenta con componentes capaces de generar las sentencias SQL necesarias para actuar sobre las bases de datos, pero serán muchos los casos en que dichas sentencias deban facilitarse desde la aplicación que estemos creando.

Actualmente todos los DBMS, incluidos aquellos que no son servidores como es el caso de Access o dBase, contemplan el uso del lenguaje SQL para efectuar cualquier tarea sobre la información, desde la definición de su estructura hasta la manipulación de los datos propiamente dichos.

Algunos RDBMS, realmente todos aquellos que contemplan la definición de *procedimientos almacenados*, utilizan también SQL como base para un lenguaje propio de tipo procedimental.

El objetivo de este capítulo es acercarle al lenguaje SQL, facilitando una explicación de sus estructuras básicas que le permitan, por ejemplo, crear una tabla, seleccionar un conjunto de datos o modificar información ya existente en la base de datos.

También se facilita, al final del capítulo, una breve introducción a dos derivados de SQL: Transact-SQL y PL/SQL. Obviamente no se intenta abordar la sintaxis completa de SQL y todas sus posibilidades. Si necesita conocer a fondo este lenguaje siempre tiene la posibilidad de recurrir a un texto específico o la referencia del lenguaje de su RDBMS en particular.

¿Qué es SQL?

Como su propio nombre indica, SQL es un lenguaje para consulta de datos. A diferencia de los lenguajes de programación que estará acostumbrado a usar, en SQL no existen ciertos elementos, como las estructuras de control, por lo que no es un lenguaje que pueda ser utilizado para crear procedimientos. Su finalidad básica es facilitar la manipulación de datos como si fuesen conjuntos, en contraposición al tratamiento que se efectúa desde los lenguajes de programación, donde cada dato se procesa de manera individual.

SQL es un estándar ANSI, lo cual significa que existe una especificación creada por dicha institución y que, en principio, deberíamos poder usar exactamente la misma sintaxis independientemente del RDBMS concreto que vayamos a utilizar. La realidad, sin embargo, es que cada RDBMS incorpora sus mejoras o extensiones particulares al SQL estándar, dando así lugar a implementaciones específicas e incompatibles con las del resto de fabricantes. Existe, no obstante, un conjunto de SQL que podríamos calificar de *mínimo común denominador*, al ser idéntico, o casi, en todos los RDBMS existentes.

Partes de SQL

El lenguaje SQL está, en realidad, compuesto de varios sub-lenguajes, entre los cuales destacaríamos los tres siguientes:

- **DML:** Es la parte más conocida del lenguaje SQL, al formar parte de él todas las sentencias de manipulación de datos: extracción de información, actualización y eliminación. Teóricamente, y asumiendo que existe un DBA que se ocupa del RDBMS, los programadores tan sólo tendrían que conocer DML para efectuar su trabajo.
- **DDL:** Con las sentencias DDL no se manipulan los datos propiamente dichos, sino la información de su estructura. Con las sentencias DDL pueden definirse las columnas de una tabla y sus atributos, eliminar o crear un índice, etc.
- **DCL:** Mucho menos conocido que los dos anteriores, este lenguaje se compone de sentencias específicas para garantizar la seguridad de acceso a los datos, facilitando la gestión de usuarios y el otorgamiento o denegación de los permisos necesarios para operar sobre cada elemento de la base de datos.

Teóricamente, la definición de las estructuras de la base de datos, así como los temas relativos a la seguridad, deberían quedar en manos del administrador del RDBMS de la empresa a la que se dirija la aplicación, de tal manera que el desarrollador sólo precisaría conocer el subconjunto DML de SQL para poder ejecutar consultas y manipular los datos según las necesidades de los usuarios de las apli-

caciones. En los puntos siguientes, no obstante, también encontrará una rápida introducción a algunas de las sentencias de DDL y DCL.

Derivados de SQL

Si bien, en un principio, la finalidad de los RDBMS era básicamente la de almacenar, recuperar y manipular la información, según las sentencias SQL facilitadas desde algún tipo de aplicación o por el usuario, con el paso del tiempo se han ido confiando a estos sistemas de tratamiento de datos otras tareas adicionales. Una de ellas es la ejecución de una cierta lógica de proceso que asegure la integridad de los datos o, simplemente, haga más fácil su tratamiento por parte de las aplicaciones. Estas operaciones se implementan en forma de desencadenadores y procedimientos almacenados, siendo preciso un lenguaje de programación para efectuar dicha implementación. SQL, como se ha dicho antes, no es un lenguaje procedimental y, por tanto, no puede ser utilizado para dar solución a estas necesidades. Por esa razón, cada fabricante de RDBMS incorpora en su producto un lenguaje propio para desempeñar esas tareas. Por regla general, estos lenguajes tienen una sintaxis basada en la de SQL, aportando los elementos necesarios para definir procedimientos, evaluar expresiones condicionales o definir una ejecución reiterada.

Dos de los lenguajes derivados de SQL más conocidos son PL/SQL, propio de las bases de datos Oracle, y T-SQL (también conocido como Transact-SQL), que podemos encontrar en SQL Server o Sybase. Lógicamente, necesitaremos aprender uno u otro dependiendo del RDBMS que utilice nuestra empresa.

Nota

Las bases de escritorio, como dBase, Access y Paradox, no disponen de un lenguaje similar a PL/SQL o T-SQL para la implementación de procedimientos almacenados, pues dichos elementos no existen en esos productos. Sí cuentan, sin embargo, con un completo lenguaje de programación que permite crear verdaderas aplicaciones, en contraposición a la lógica discreta, siempre dirigida al tratamiento de datos, para la que están pensados PL/SQL y T-SQL.

Ejecución de sentencias SQL

A partir del punto siguiente va a ir conociendo múltiples sentencias SQL para selección de datos, modificación, eliminación y definición de algunas estructuras. Estas sentencias no puede introducirlas, sin más, en el editor de Visual Basic .NET y esperar a que funcionen, puesto que su entorno de trabajo es el interior de un RDBMS. Si quiere ir comprobando los ejemplos propuestos, lo que no es totalmente necesario para comprender la sintaxis de SQL, deberá recurrir a la herramienta específica de la base de datos que pretenda emplear en sus desarrollos.

Puede usar una base de datos de escritorio, por ejemplo Microsoft Access (véase figura 2.1), ya que permiten la ejecución de un subconjunto del lenguaje SQL, o bien la utilidad de SQL interactivo del RDBMS con que cuente. Ésta puede ser el Analizador de consultas SQL de SQL Server (véase figura 2.2), el Interactive SQL de InterBase (véase figura 2.3) o bien el SQL*Plus Worksheet de Oracle 8i (véase figura 2.4). En cualquier caso, no entraremos en este momento en detalles sobre cómo usar una herramienta u otra, en el próximo capítulo conocerá algunas de ellas, sino que nos centraremos en la sintaxis de SQL, sin más.

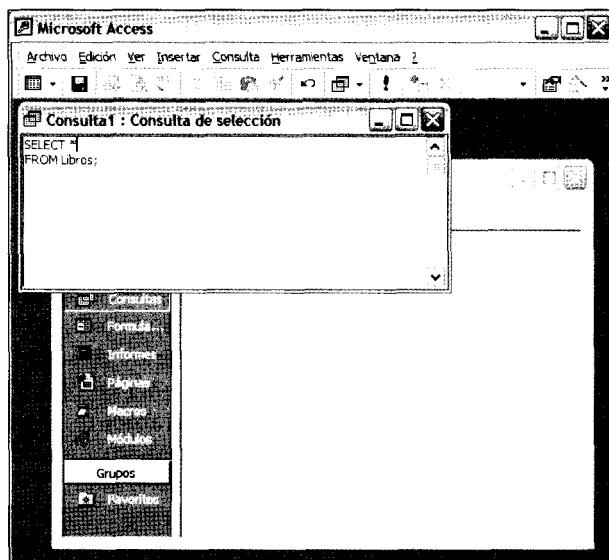


Figura 2.1. Desde Microsoft Access es posible definir consultas mediante asistentes, pero también pueden escribirse de forma manual

Nota

Debe tener en cuenta que la compatibilidad de las distintas implementaciones SQL en cada RDBMS respecto al estándar SQL, normalmente se toma como referencia SQL-92 y SQL3, es muy variable. Esto implica que algunas sentencias SQL puedan necesitar ligeros cambios según la base de datos sobre la que vaya a ejecutarse.

DDL

Antes de poder manipular datos mediante DML, es necesario que dichos datos existan ya en la base de datos y, para ello, es preciso definir las estructuras que los

albergarán. Estas estructuras conforman lo que se conoce habitualmente como el *catálogo* del RDBMS. Por esta razón comenzaremos introduciéndonos en DDL, el lenguaje de definición de datos.

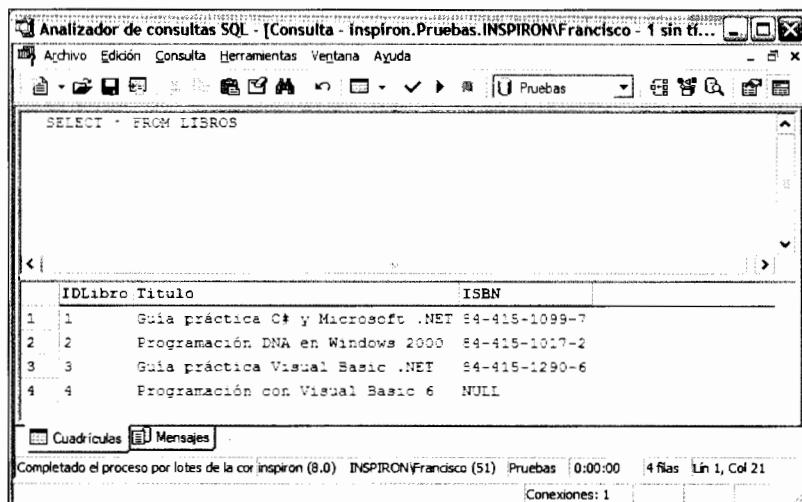


Figura 2.2. El Analizador de consultas de SQL Server facilita la ejecución de sentencias SQL sobre este RDBMS, mostrando los resultados en la parte inferior de la ventana

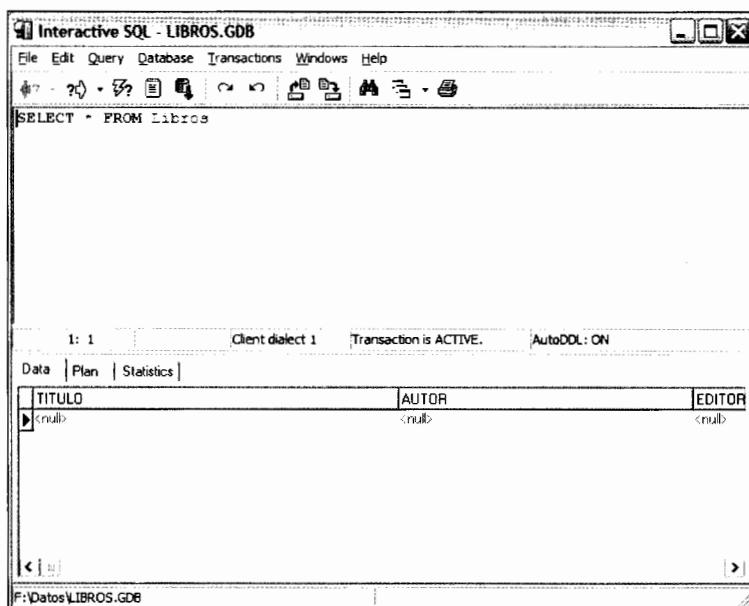


Figura 2.3. Aunque con un nombre diferente y sobre otro RDBMS, el Interactive SQL de InterBase es una herramienta similar a la de SQL Server

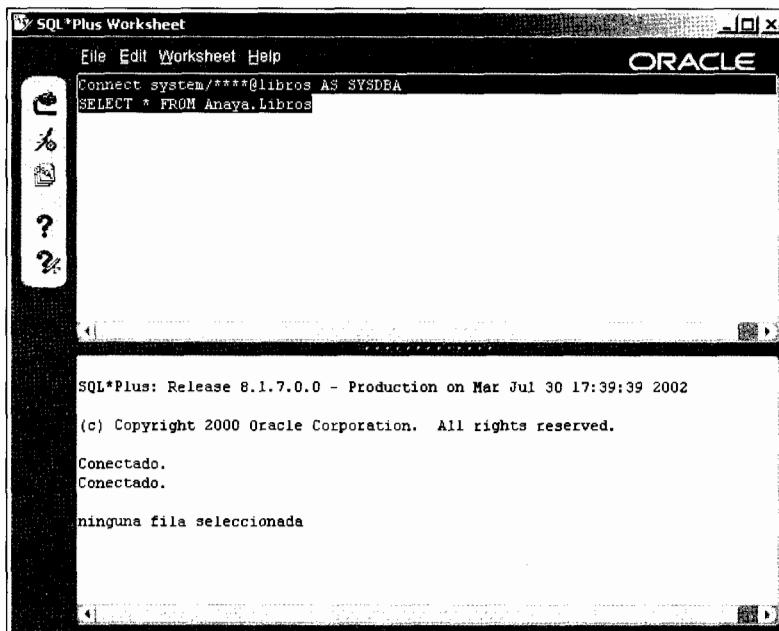


Figura 2.4. La utilidad SQL*Plus de Oracle es posiblemente la más rudimentaria, al estar prácticamente basada en texto sin elementos de interfaz que faciliten las operaciones básicas

Mediante DDL es posible crear una base de datos, crear las tablas definiendo las columnas que tendrán, crear índices y vistas y, por supuesto, modificar y eliminar todos esos elementos. Los comandos DDL son tres:

- **CREATE:** Se utiliza para crear bases de datos, tablas, índices, desencadenadores, procedimientos almacenados y vistas, según la palabra clave que se indique a continuación. Dependiendo del caso aceptará más o menos parámetros adicionales.
- **DROP:** Acepta las mismas palabras clave que el comando CREATE, si bien la finalidad es eliminar el elemento indicado en lugar de crearlo.
- **ALTER:** Con este comando es posible modificar algunas de las estructuras de una base de datos. No acepta todas las palabras clave de los dos comandos anteriores, sólo parte de ellas. Las vistas, por ejemplo, pueden crearse y eliminarse, pero no modificarse.

Dependiendo del elemento que se pretenda crear, eliminar o modificar, tras el comando irá una de las palabras clave siguientes:

- **DATABASE:** Base de datos.
- **TABLE:** Tabla.

- VIEW: Vista.
- PROCEDURE: Procedimiento almacenado.
- TRIGGER: Desencadenador.
- INDEX: Índice.

Los parámetros adicionales dependerán del comando y, en ocasiones, del RDBMS sobre el que esté trabajándose.

Creación de una base de datos

La creación de una nueva base de datos es un proceso relativamente complejo, al ser preciso definir su contenido y estructura, espacio de almacenamiento, permisos de acceso, usuarios con que cuenta y otra serie de reglas adicionales. Por ello prácticamente todos los RDBMS incorporan un asistente para efectuar este trabajo, asistente que va dirigido al DBA. Como desarrolladores, tan sólo necesitaríamos emplear la sintaxis siguiente en caso de que fuese imprescindible crear la base de datos desde nuestra propia aplicación.

La sintaxis básica para crear una nueva base de datos sería ésta:

```
CREATE DATABASE nombre_base_de_datos
```

El nombre de la base de datos puede ser un nombre lógico, caso en el cual existirán opciones adicionales para indicar la localización y nombre físico de los archivos donde será alojada, o bien ser una especificación de nombre de archivo en sí mismo, según el RDBMS con que trabajemos.

Nota

Dependiendo del RDBMS empleado, también es posible utilizar los comandos **DROP** y **ALTER** con **DATABASE** para eliminar una base de datos o modificar alguno de sus parámetros.

Creación de tablas

Una vez que ya tenemos la base de datos creada, y estamos conectados a ella, el siguiente paso lógico es crear las tablas que alojarán los datos, definiendo los atributos de cada una de dichas columnas. Al tiempo, también es posible establecer las restricciones para cada una de las columnas, así como la creación de claves primarias.

La sintaxis general para la definición de una nueva tabla en una base de datos es la siguiente:

```
CREATE TABLE nombre_tabla
( nombre_columna tipo opciones,
  nombre_columna tipo opciones,
  ...
)
```

Pueden existir parámetros adicionales, pero lo más importante es el nombre que deseamos dar a la tabla y la lista con la definición de las columnas, cada una de las cuales tendrá un nombre, un tipo que determinará la clase de información que puede contener y opciones adicionales.

Los tipos de datos suelen ser específicos de cada DBMS, si bien existe un conjunto básico definido en el estándar ANSI que más o menos es compatible en todos los sistemas. Algunos de esos tipos son:

- **INTEGER:** Un número entero, normalmente de 32 bits.
- **SMALLINT:** Un número entero de menor tamaño que el anterior, normalmente de 16 bits.
- **CHAR (N):** Una cadena de N caracteres con longitud fija.
- **VARCHAR (N):** Una cadena de hasta N caracteres, con longitud variable.
- **DATE:** Una fecha.
- **TIME:** Una hora.
- **FLOAT:** Un número en punto flotante.

En SQL Server no existe el tipo DATE y, en su lugar, se utiliza el tipo DATETIME para alojar fecha y hora. En Oracle, a pesar de que existe el tipo estándar VARCHAR, éste se encuentra en desuso y, en su lugar, se utiliza normalmente VARCHAR2. En realidad VARCHAR es, en Oracle 8i, un sinónimo para VARCHAR2. Lo mismo ocurre con el tipo INTEGER, que en Oracle es un sinónimo de NUMBER (38), al ser en dicho RDBMS el tipo NUMBER el más empleado para tipos numéricos.

Tras el tipo de cada columna, y de manera opcional, pueden aplicarse restricciones y definir claves.

Algunas de las opciones posibles son:

- **NOT NULL:** Indica que la columna no puede quedar sin valor. Hay que tener en cuenta que una cadena de caracteres vacía o un cero no es igual a NULL, por lo que una columna que tenga esta restricción podría tomar perfectamente uno de esos valores.
- **UNIQUE:** Esta restricción impide que en la columna asociada puedan existir dos valores idénticos o, dicho de otra manera, garantiza que los valores incluidos en esta columna serán únicos.
- **PRIMARY KEY:** Establece la columna a la que acompaña como clave primaria o principal de la tabla, es decir, como atributo de identidad de cada fila.

Esto suele conllevar el hecho de que los valores de dicha columna deban ser únicos.

- FOREIGN KEY: Define una columna como clave externa, que hace referencia a una clave primaria de otra tabla.

Las opciones PRIMARY KEY y FOREIGN KEY pueden tanto seguir a una columna como disponerse al final de la lista de columnas, caso en el cual se indicaría entre paréntesis, detrás de KEY, el nombre de la columna o columnas que actuarían como clave primaria o clave externa. En este último caso, hay que añadir el apartado REFERENCES (col), donde col sería la referencia a la tabla y columna externa.

A continuación tiene un ejemplo de creación de una tabla simple:

```
CREATE TABLE Libros
( IDLibro INTEGER PRIMARY KEY,
  ISBN CHAR(13) UNIQUE,
  Titulo VARCHAR(30) NOT NULL,
  Autor VARCHAR(50),
  FOREIGN KEY(Editorial)
    REFERENCES Editoriales(IDEditorial)
)
```

Tenemos una columna que actúa como clave principal, identificando a cada una de las filas que pudieran existir en la tabla. Se ha utilizado una columna INTEGER como clave única en lugar de la columna ISBN que, como puede verse, alberga valores únicos. Para los RDBMS, sin embargo, resulta mucho más fácil gestionar una clave que es un número que una cadena de 13 caracteres. Aunque parezca redundante añadir una columna como identidad de cada fila, en realidad está ahorrándose espacio y se mejora el rendimiento.

Las columnas Titulo y Autor tienen una longitud variable, aunque con un máximo, en contraposición a ISBN que siempre tendrá 13 caracteres. El uso de VARCHAR también ahorra espacio en la base de datos, al ocupar tan sólo el espacio necesario para almacenar cada valor en lugar de asignar el máximo posible.

Por último, encontramos la definición de una clave externa, llamada Editorial en esta tabla, que hace referencia a la columna IDEditorial de la tabla Editoriales. Lógicamente, dicha tabla deberá existir y la columna IDEditorial sería su clave primaria.

Nota

Debe tener en cuenta que, dependiendo del DBMS que emplee, para crear una tabla puede necesitar ciertos permisos o privilegios. De no contar con ellos, la ejecución de una sentencia como la anterior provocaría un error. En algunos DBMS, además, deberá preceder el nombre de la nueva tabla con el esquema donde quiera crearla si es que dicho esquema no es el asumido por defecto.

Nota

Otra posibilidad, a la hora de crear una tabla en una base de datos, consiste en hacerlo a partir de los datos que ya existen en otra tabla. En este caso el comando `CREATE TABLE nombre` iría seguido no de una lista de columnas, tipos y opciones, como en el ejemplo anterior, sino de una consulta SQL que recuperaría de otra tabla o tablas los datos con los que se generaría la nueva tabla.

Modificación y borrado de tablas

Las modificaciones a una tabla pueden ser de distintos tipos, desde añadir o eliminar una columna o una restricción hasta modificar la definición original de la columna para cambiar el tipo. En cualquier caso, la sentencia la iniciaremos con `ALTER TABLE NombreTabla`, tras la cual dispondremos la palabra `ADD`, para añadir un nuevo elemento; `ALTER`, para modificarlo, o `DROP` si queremos eliminarlo. Por ejemplo:

```
ALTER TABLE Libros DROP COLUMN ISBN  
...  
ALTER TABLE ADD ISBN VARCHAR(13)  
...  
ALTER TABLE Libros ALTER COLUMN Titulo VARCHAR(35)
```

Nota

Hay que tener en cuenta que la modificación del tipo de una columna puede implicar una pérdida de datos, algo que es obvio si la columna se elimina.

Si deseamos eliminar una tabla completa, no tenemos más que usar la sintaxis mostrada en el ejemplo siguiente:

```
DROP TABLE Libros
```

Otras operaciones de definición de datos

Aunque las tablas son el centro de cualquier base de datos, al ser físicamente las que alojan la información, también existen otros elementos de uso común, entre ellos los índices y las vistas.

Para crear un índice utilizaría la siguiente sintaxis:

```
CREATE INDEX NombreIndice ON NombreTabla(Columna)
```

El nombre de la tabla y la columna determinan la información a partir de la cual el DBMS generará el nuevo índice, al que llamará `NombreIndice`. El índice es en sí una estructura ordenada que acelera tareas como las búsquedas o la obtención de los datos con un cierto orden. La creación de un índice tiene sentido sobre aquellas columnas que van a emplearse con cierta frecuencia en las consultas para buscar u ordenar.

Tras el nombre de la tabla y columna, de manera opcional, es posible indicar si el orden del índice será ascendente o descendente, con las cláusulas `ASC` y `DESC`, respectivamente. Sólo puede aparecer una de ellas.

La tabla `Libros` que hemos utilizado en los ejemplos anteriores dispone de una columna que actúa como clave principal, pero es de esperar que en la mayoría de las ocasiones la búsqueda se efectúe a partir del ISBN.

Con la siguiente sentencia SQL generaríamos un índice que aceleraría dichas búsquedas.

```
CREATE INDEX IndISBN ON Libros(ISBN)
```

Nota

Un índice puede eliminarse con la sentencia `DROP INDEX NombreIndice` y, en algunos DBMS, también modificarse mediante el comando `ALTER`.

En cuanto a las vistas, podríamos definirlas como consultas SQL predefinidas, a las que se asocia un nombre, cuyo objetivo es facilitar la vida a los usuarios y programadores, por ejemplo evitando que tengan que componer complejas consultas con relaciones entre múltiples tablas para obtener los datos que precisan. Las vistas se definen una vez, normalmente es trabajo del administrador o DBA, y posteriormente pueden ser usadas tantas veces como se necesite, apareciendo a los ojos de los usuarios como si fuesen tablas.

La sintaxis para definir una nueva vista es ésta:

```
CREATE VIEW NombreVista AS Consulta
```

Para componer la consulta necesitaremos, lógicamente, conocer la sentencia DML `SELECT`, que trataremos en un punto posterior. Aunque con ciertas limitaciones, es posible definir como vista prácticamente cualquier consulta. En el ejemplo siguiente se crea una vista que facilita todas las editoriales con los libros que tienen publicados, partiendo de las hipotéticas tablas `Editoriales` y `Libros` que se han empleado en ejemplos anteriores.

```
CREATE VIEW LibrosEditorial AS
    SELECT Editorial.Nombre, Libros.Titulo
    FROM Editorial, Libros
    WHERE Editorial.IDEditorial=Libros.Editorial
```

El resultado, al ejecutar esta vista, sería una tabla temporal con dos columnas, una con el nombre de la editorial y otra con el título del libro, y tantas filas como combinaciones de editoriales y libros existan en las tablas.

La consulta asociada a una vista puede modificarse con la sentencia ALTER VIEW, por ejemplo:

```
ALTER VIEW LibrosEditorial AS  
    SELECT Editorial.Nombre, Libros.ISBN  
    FROM Editorial, Libros  
    WHERE Editorial.IDEditorial=Libros.Editorial
```

Aquí hemos modificado la vista anterior para que, en lugar del título de los libros, se incluya el ISBN.

Asimismo, la eliminación de una vista es tan simple como ejecutar la sentencia DROP VIEW NombreVista.

Nota

Según se indicaba al inicio de este punto, dedicado a DDL, los mismos comandos CREATE, ALTER y DROP pueden utilizarse también para crear desencadenadores, procedimientos almacenados y otros elementos. Las posibilidades y sintaxis, en estos casos, varían en mayor medida de un DBMS a otro. Para crear un desencadenador o procedimiento almacenado, por ejemplo, suele hacerse uso del lenguaje específico del RDBMS para efectuar la implementación de una cierta lógica.

DML

Si con DDL definimos las estructuras para el almacenamiento de los datos, con DML podremos manipular los datos propiamente dichos, efectuando consultas para recuperar datos, modificaciones, borrados, etc.

Las operaciones fundamentales de manipulación de datos son cuatro y, consecuentemente, cuatro los comandos DML que necesitaremos para poder llevarlas a cabo:

- **SELECT:** La sentencia DML por excelencia. Se utiliza para ejecutar consultas de recuperación de datos.
- **INSERT:** Su finalidad es insertar nuevas filas en una tabla.
- **UPDATE:** Con este comando es posible cambiar la información alojada en una o más filas.
- **DELETE:** Se utiliza para eliminar una o más filas.

A diferencia de lo que ocurre con las sentencias de un lenguaje de programación típico, como puede ser Visual Basic, estas sentencias DML no actúan sobre un solo elemento, sino sobre conjuntos de filas y columnas. Es importante tenerlo en cuenta a la hora, por ejemplo, de utilizar una sentencia DELETE para eliminar una fila, puesto que, inadvertidamente, podemos afectar a varias o, incluso, la tabla completa.

En los puntos siguientes va a conocer la sintaxis básica de cada una de las cuatro sentencias indicadas, así como a ver algunos ejemplos de su uso.

Inserción de datos

Si creamos una aplicación para que los usuarios puedan trabajar sobre una base de datos, una operación básica es la inserción de nuevos datos. De hecho, en principio será la única opción disponible a menos que las tablas de datos sobre las que se trabaje tengan ya información previamente. La sintaxis de la sentencia INSERT es la siguiente:

```
INSERT INTO NombreTabla VALUES (valor, valor ...)
```

Tras la palabra VALUES, y entre paréntesis, se facilitarán tantos valores como columnas existan en la tabla indicada y, además, en el mismo orden y del mismo tipo. Los valores numéricos se introducen tal cual, mientras que las secuencias de caracteres normalmente se delimitan entre comillas simples.

Suponiendo que quisiésemos añadir una fila a la tabla Libros creada como ejemplo en un paso anterior, y que contaba con cinco columnas, utilizaríamos una sentencia así:

```
INSERT INTO Libros VALUES  
(1, '84-415-1351-1',  
 'Programación con Visual Basic .NET',  
 'Francisco Charte Ojeda', 3)
```

En caso de que desee introducir datos en menos columnas de las que existen en la tabla, debería especificar las columnas de destino detrás del nombre de la tabla, entre paréntesis. Por ejemplo:

```
INSERT INTO Libros(IDLibro, ISBN) VALUES  
(1, '84-415-1351-1')
```

El resto de las columnas de esa fila quedarían con el valor NULL, siempre, por supuesto, que sus restricciones no lo impidan.

En el caso de que se viole alguna restricción, como que un valor no sea único en una columna que debe serlo o que se entregue como una clave externa un valor que no se encuentre en la tabla de destino, la inserción no se efectuará y generará un error.

Recuperación de datos

Como se apuntaba anteriormente, la sentencia SELECT es la más conocida de SQL y es que, no en vano, es generalmente la más utilizada. Posiblemente también sea la que con más opciones y variantes cuenta, así como algunas diferencias de implementación según el RDBMS. Lo habitual es que el resultado de una sentencia SELECT sea un conjunto de datos, compuesto de múltiples filas y columnas, aunque también es posible recuperar un dato concreto como resultado de la evaluación de una expresión.

La sintaxis más sencilla de esta sentencia es la siguiente:

```
SELECT Columna1, Columna2, ... FROM Tabla
```

El conjunto de datos devuelto se compondrá de las columnas seleccionadas en todas las filas de la tabla indicada tras FROM. Por ejemplo:

```
SELECT Titulo, Autor FROM Libros
```

Esta consulta retornaría el título y autor de todas las filas existentes en la tabla Libros. Si queremos obtener todas las columnas de la tabla, sin necesidad de especificar el nombre de cada una, podemos sustituir la lista que sigue a la palabra SELECT por un asterisco:

```
SELECT * FROM Libros
```

En este caso se obtendría la totalidad de la tabla Libros, es decir, todas las filas con todas las columnas. Usualmente esto no es necesario, por ello se seleccionan sólo las columnas que se precisan en cada caso y, además, se limitan las filas mediante condiciones. También es posible establecer el orden en que se facilitan las filas recuperadas, agruparlas, efectuar consultas sobre consultas, etc.

Alias de tablas

Como va a ver en los puntos siguientes, a la hora de establecer condiciones y, especialmente, cuando se usa más de una tabla para generar la consulta, es necesario hacer referencia a las columnas utilizando la notación NombreTabla.NOMBREColumna. Esto puede hacer algo farragosa la codificación de consultas complejas, repitiendo el nombre completo de la tabla una y otra vez.

Afortunadamente, la sentencia SELECT nos permite asociar un alias a cada una de las tablas participantes en la consulta. Para ello no tenemos más que poner dicho alias tras el nombre original de la tabla, a continuación de la palabra FROM, pudiendo usar el alias en toda la consulta. Por ejemplo, en la siguiente consulta se asocia el alias L a la tabla Libros:

```
SELECT L.ISBN, L.Titulo, L.Editorial  
FROM Libros L
```

En este caso concreto no tiene mucho sentido ya que, al participar una sola tabla en la consulta, no tenemos porqué usar la notación L.ISBN para seleccionar la columna ISBN, hubiera bastado con facilitar directamente el nombre de la columna. Encontrará una mayor aplicación al uso de los alias de tablas en los puntos siguientes.

Selección de filas

Indicar en la consulta qué columnas deseamos incluir en el conjunto de resultados es bastante sencillo, ya que el número de columnas con que cuenta una tabla suele ser pequeño y, por tanto, podemos indicar los nombres de cada una de ellas tras la palabra SELECT.

Para seleccionar un determinado conjunto de filas no es posible utilizar una técnica similar, primero porque las filas no tienen nombres, sino que contienen valores, y segundo porque, a diferencia de columna, una tabla puede alojar miles o incluso millones de filas. Por ello la solución tiene que ser totalmente diferente pero, como programadores, nos debe resultar relativamente fácil comprenderla ya que está basada en la evaluación de expresiones condicionales.

Aparte de la cláusula FROM, única obligatoria y con la cual se indican las tablas de las que va a recuperarse la información, la sentencia SELECT puede contar, opcionalmente, con varias más. Una de ellas es la cláusula WHERE, encargada de aplicar uno o más condicionales basándose en los cuales se filtrarán las filas a devolver. La sintaxis de la consulta sería la siguiente:

```
SELECT Columnas FROM Tablas WHERE Condición
```

La condición se asemejará bastante a los condicionales que estamos acostumbrados a utilizar en Visual Basic, con operadores relacionales como =, < y > y operadores lógicos como AND, OR y NOT. Esto permite una gran flexibilidad a la hora de seleccionar conjuntos de datos. Por ejemplo:

```
SELECT ISBN, Titulo  
FROM Libros  
WHERE Autor='Bill Gates'  
  
...  
  
SELECT ISBN, Titulo  
FROM LIBROS  
WHERE Editorial=3 AND Precio<60
```

En el primer caso obtendríamos la lista de libros escritos por un cierto autor, en este caso Bill Gates, mientras que en el segundo la lista se compondría de todos los libros de la editorial con el código 3 cuyo precio no sea superior a 60 euros.

Condicionales complejos

Además de condicionales relativamente simples, en los que se compara si el valor de una columna es igual, mayor o menor que otro valor dado, en la cláusula

WHERE pueden emplearse elementos adicionales que nos permiten crear condicionales realmente complejas. A los operadores relacionales clásicos hay que añadir otros, como IN, LIKE y BETWEEN, mucho más flexibles.

Mediante el operador IN es posible crear una expresión que será cierta en caso de que el valor de la columna indicada a la izquierda contenga uno de los valores delimitados entre paréntesis, a la derecha del operador. Por ejemplo:

```
SELECT ISBN, Titulo
  FROM Libros
 WHERE Autor
   IN ('Bill Gates', 'Paul Allen', 'Steven Jobs')
```

En esta consulta se seleccionan todos los títulos escritos por los tres autores indicados entre paréntesis, evitándonos así la construcción de tres condicionales con el operador = unidos mediante AND.

Con el operador LIKE podemos efectuar comparaciones no absolutas entre los valores de una columna y un valor dado o, dicho de otra forma, facilita la búsqueda de un cierto patrón entre los valores de una columna.

Tras el operador LIKE debemos facilitar, entre comillas simples, el patrón a buscar. Por ejemplo:

```
SELECT ISBN, Titulo
  FROM Libros
 WHERE Titulo LIKE 'Visual%'
```

En este caso obtendríamos la lista de todos aquellos libros cuyo título comience con Visual, sin necesidad de especificar el título completo.

Por último tenemos BETWEEN, mediante el cual es posible especificar un rango de valores en el cual deberá encontrarse una cierta columna para ser incluida en el conjunto de resultados. Por ejemplo, si queremos obtener todos aquellos libros cuyo precio se encuentre entre 30 y 50 euros podemos utilizar una consulta como la siguiente:

```
SELECT ISBN, Titulo
  FROM Libros
 WHERE Precio BETWEEN 30 AND 50
```

No olvide que todas estas expresiones simples pueden combinarse, mediante los operadores lógicos, dando lugar a consultas mucho más elaboradas. Sirve como ejemplo la mostrada a continuación:

```
SELECT ISBN, Titulo
  FROM Libros
 WHERE Autor
   IN ('Bill Gates', 'Paul Allen', 'Steven Jobs')
   AND Precio BETWEEN 30 AND 50
   AND Titulo LIKE 'Visual%'
```

El conjunto de datos obtenido estaría formado por todos aquellos libros que, escritos por Bill Gates, Paul Allen o Steven Jobs, costasen entre 30 y 50 euros y, además, su título comenzase con la palabra Visual.

Orden de las filas

En principio, las filas recuperadas con la sentencia SELECT no aparecerían en un orden concreto, aunque lo usual es que se devuelvan en el mismo orden en que aparecen en la tabla. Es posible que éste no sea el orden que más nos interese, un aspecto que podemos alterar fácilmente mediante la cláusula ORDER BY.

Lo único que tenemos que facilitar a ORDER BY es el nombre de la columna que actuará como referencia para la ordenación. En caso de que sean varias, separaríamos el nombre de cada una de ellas mediante comas. Por ejemplo:

```
SELECT * FROM Libros ORDER BY Titulo
```

Con esta consulta obtendríamos una lista completa de todas las filas que hay en la tabla Libros ordenadas alfabéticamente por el contenido de la columna Titulo. El orden sería de menor a mayor, es decir, de la A a la Z. Podemos alterar el orden mediante las palabras DESC y ASC, según deseemos un orden descendente o ascendente.

Expresiones y funciones de resumen

Hasta ahora, en las consultas mostradas como ejemplo en los puntos anteriores, siempre hemos seleccionado como resultados una serie de columnas existentes en las tablas. También existe la posibilidad de introducir expresiones, usando como operando el valor de una o varias columnas, y funciones de resumen o *agregación*.

Para obtener, aparte de las columnas que nos interesen, datos obtenidos a partir de algún cálculo u operación, tan sólo tenemos que incluir la expresión apropiada como si de una columna más se tratase. Por ejemplo:

```
SELECT Titulo, Precio * 1,04  
      FROM Libros
```

Con esta consulta se obtendría una lista de títulos de libros y una segunda columna que tendría el precio más un 4 por ciento. Dicho valor se obtendría como resultado de la consulta, pero no afectaría en absoluto al contenido de la tabla Libros.

Si lo que necesitamos no es operar sobre una cierta columna de una fila, como en este caso, sino sobre todas las columnas recuperadas a partir de la consulta, podemos utilizar las funciones de agregación. Las de uso más corriente son:

- COUNT(): Cuenta el número de filas obtenidas como resultado.
- SUM(): Suma los valores de la columna indicada de todas las filas del conjunto.

- **AVG ()**: Efectúa una media sobre los valores de la columna indicada de todas las filas del conjunto.
- **MAX ()**: Halla el valor máximo de la columna indicada entre todas las filas del conjunto.
- **MIN ()**: Halla el valor mínimo de la columna indicada entre todas las filas del conjunto.

Cuando en una sentencia SELECT aparece una o más funciones de agregación, las únicas columnas adicionales que pueden incluirse son las utilizadas para efectuar agrupamientos, según se verá en el punto siguiente. El siguiente es un ejemplo de uso de estas funciones:

```
SELECT COUNT(Precio), MIN(Precio), MAX(Precio)
  FROM Libros
```

El resultado obtenido sería una sola fila conteniendo el número total de filas de la tabla Libros, puesto que no se ha añadido una cláusula WHERE, así como el precio del libro más barato y el del más caro.

Agrupamiento

Las funciones de agregación encuentran su mayor utilidad cuando se emplean de manera combinada con la cláusula de agrupamiento de la sentencia SELECT: GROUP BY. Ésta puede ir seguida del nombre de una o más columnas, de tal forma que las filas de la consulta se agrupan según los valores de ellas. Por ejemplo:

```
SELECT Editorial, COUNT(Precio),
       MIN(Precio), MAX(Precio)
  FROM Libros
 GROUP BY Editorial
```

En este caso no se obtendría una sola fila de resultados, como en el punto previo, sino una fila por editorial. Además, cada fila contendría el número de títulos de esa editorial, así como los precios mínimo y máximo de los títulos de esa editorial. Un resultado así, que es ejecutado en el RDBMS y devuelto a la aplicación, puede ahorrarnos muchas sentencias de programa Visual Basic.

Enlaces entre tablas

En todos los ejemplos previos se ha operado exclusivamente sobre una tabla de la base de datos. No es extraño, sin embargo, que sea preciso recuperar columnas de dos o más tablas, por ejemplo una lista de editoriales y los libros correspondientes a cada una, la información de un pedido y las líneas asociadas, etc.

Tras la cláusula FROM puede aparecer cualquier número de tablas, y tras SELECT todas las columnas que se necesiten de esas tablas. Para poder asociar los datos de las distintas tablas, y componer adecuadamente cada fila del conjunto de datos re-

sultante, es preciso crear un vínculo entre las distintas tablas. Con este fin se utilizan las claves primarias y externas que se definieron previamente, enlazándolas mediante un condicional en el apartado WHERE.

Suponga que desea obtener una lista con el nombre de cada editorial y los títulos que le corresponden. La consulta podría ser la siguiente:

```
SELECT E.Nombre, L.Titulo
  FROM Editorial E, Libros L
 WHERE E.IDEditorial=L.Editorial
```

Obtendríamos un conjunto de datos resultante de una relación uno-a-muchos, es decir, por cada editorial aparecerían todos los títulos con que cuenta. Podrían añadirse enlaces a varias tablas, más de dos, utilizando exactamente la misma técnica, si bien a medida que se incluyen más relaciones la consulta va complicándose tanto para nosotros como para el RDBMS, que tardará más tiempo en elaborarla.

Como puede suponer, a los datos resultantes de un enlace entre varias tablas también podemos aplicar condicionales, agrupamientos y el orden que nos interese. Todo lo visto en los puntos previos puede ser aplicado también cuando se opera sobre múltiples tablas.

Consultas dentro de consultas

Las sentencias SELECT pueden anidarse, dando lugar a lo que se conoce como *subconsultas*. Primero se ejecuta la consulta que está en el nivel más interior (aparece después en la sentencia completa), obteniéndose un conjunto de resultados que, en lugar de devolverse directamente, se entrega a la consulta del nivel exterior para que efectúe el proceso apropiado. Este proceso se repite las veces que sea preciso hasta llegar al SELECT más exterior, el que aparece en primer lugar en la sentencia, momento en que se devuelven los resultados.

```
SELECT Columnas FROM Tabla
  WHERE Condición
    (SELECT Columnas FROM Tabla)
```

Suponga que desea obtener una lista de todos aquellos libros cuyo precio es superior a la media. ¿Cómo se expresaría esa consulta? Primero es necesario obtener la media del precio y, a continuación, utilizarla como resultado para ejecutar otra consulta. En lugar de utilizar una variable intermedia y ejecutar dos consultas, resulta mucho más efectivo utilizar una sentencia como la siguiente:

```
SELECT Titulo, Precio FROM Libros
  WHERE Precio >
    (SELECT AVG(Precio) FROM Libros)
```

El resultado de la subconsulta no tiene que aplicarse necesariamente a un condicional, pudiendo utilizarse directamente como origen de datos para la consulta superior.

Por ejemplo:

```
SELECT Titulo, Precio*1.04 FROM
  (SELECT Titulo, Precio, Editorial
   FROM Libros
   WHERE Editorial=
     (SELECT IdEditorial FROM Editoriales
      WHERE Nombre='Anaya Multimedia')
  )
```

En este caso se ejecuta primero la sentencia `SELECT IdEditorial ...` con el fin de obtener el identificador de una cierta editorial. Ese identificador se utiliza para recuperar el título, precio y editorial de todos los libros de esa editorial, resultado que, a su vez, es usado por otro `SELECT` para obtener el título y un cálculo sobre el precio.

Nota

No es aconsejable abusar de la anidación de consultas. Aunque en ocasiones resulte la única vía posible para obtener un cierto resultado, en otras pueden sustituirse por condicionales más o menos simples en una sola consulta.

Actualización de datos

Una vez insertados en una tabla, los datos pueden consultarse tantas veces como se precise e, igualmente, es posible modificarlos si es necesario. En este caso la sentencia a utilizar será `UPDATE`, con la sintaxis siguiente:

```
UPDATE NombreTabla
  SET Columna=Valor, Columna=Valor ...
```

Hay que tener en cuenta que, de no facilitar una cláusula `WHERE` con un condicional, la sentencia de actualización afectará a la tabla completa. La ejecución de la sentencia siguiente, por ejemplo, asignaría el valor 30 al precio de todos los libros existentes en la tabla `Libros`:

```
UPDATE Libros
  SET Precio=30
```

Lo habitual es que todas las sentencias de actualización sean siempre condicionales:

```
UPDATE Libros
  SET Precio=30
  WHERE ISBN='84-415-1351-1'
```

Eliminación de datos

La eliminación de filas de una tabla seguramente sea la operación más simple, ya que no tienen que indicarse columnas al no ser posible el borrado de éstas individualmente, sino el de las filas completas. La sintaxis de la sentencia DELETE es la que sigue:

```
DELETE FROM Tabla
```

Al igual que ocurre con la sentencia UPDATE, si no se aplica condicional alguno la eliminación afectará a toda la tabla, es decir, se eliminará todo su contenido, por lo que hay que tener un especial cuidado al utilizar esta sentencia. También debe tenerse en cuenta que es posible encontrar problemas si, por ejemplo, intenta borrar una fila a la que se hace referencia desde otro punto, ya sea de la misma tabla o bien de una externa. De esta forma el DBMS asegura la integridad referencial, evitando, por ejemplo, que se borre una cierta editorial si en la tabla de libros hay filas que hacen referencia a ella.

Suponiendo que quisiera eliminar la fila correspondiente a un cierto libro, la sentencia podría ser la mostrada a continuación:

```
DELETE Libros  
WHERE ISBN='84-415-1351-1'
```

DCL

Como se indicaba al inicio del capítulo, DCL resulta mucho menos conocido para los desarrolladores que DML y DDL, especialmente porque las tareas para las que se utiliza recaen siempre en un DBA, encargado de gestionar los usuarios que pueden acceder a la base de datos y otorgarles o denegarles la operación sobre sus tablas.

Las dos sentencias más conocidas de este grupo son GRANT y REVOKE. Con la primera se otorga un cierto permiso, mientras que con la segunda se deniega. Los permisos pueden ser globales o muy concretos, por ejemplo la inserción o borrado de una tabla o la modificación de ciertas columnas. El destinatario del otorgamiento o la denegación es un perfil o una cuenta de usuario, según los casos. Estos perfiles y cuentas también pueden crearse mediante DCL, con sentencias del tipo CREATE USER y CREATE ROLE, si bien la sintaxis suele diferir entre distintos DBMS.

Suponiendo que estuviésemos trabajando con SQL Server y deseásemos permitir la selección e inserción, pero no la modificación ni el borrado, sobre la tabla Libros a un usuario llamado Operador, podríamos usar las siguientes sentencias DCL:

```
GRANT SELECT, INSERT  
ON Libros TO Operador
```

```
REVOKE UPDATE, DELETE  
ON Libros FROM Operador
```

Derivados de SQL

El lenguaje SQL, como ha podido ver desde el inicio de este capítulo, está orientado al trabajo con conjuntos de datos, facilitando la selección y modificación de manera flexible y potente. Efectuar un trabajo similar desde Visual Basic, procesando los datos de manera individual, conllevaría una carga mucho mayor para nuestras aplicaciones.

A pesar de todo, SQL tiene sus limitaciones o, simplemente, no estaba pensado originalmente para las necesidades que han surgido con posterioridad. Una de esas necesidades es la incorporación en el propio DBMS de parte de la lógica de proceso de los datos, en forma de desencadenadores y procedimientos almacenados, con el fin de aliviar la carga de trabajo de los clientes y, al tiempo, centralizar las reglas de negocio en un servidor, facilitando su mantenimiento.

Por esta razón, cada fabricante de DBMS ha creado su propio lenguaje que, partiendo de la sintaxis y naturaleza de SQL, añade posibilidades como el trabajo con variables, evaluación de expresiones condicionales y repetición de sentencias. Dos de esos lenguajes, quizás los más representativos, son Transact-SQL, conocido originalmente como T-SQL y utilizado por Sybase y SQL Server, y PL/SQL, empleado en Oracle.

Tanto Transact-SQL como PL/SQL son lenguajes lo suficientemente complejos como para escribir varios libros específicos sobre ellos, puede encontrarlos en su librería habitual, por lo que en los puntos siguientes lo único que se pretende es facilitar un acercamiento muy superficial a ellos.

Nota

El código escrito en lenguajes como Transact-SQL y PL/SQL se ejecuta en el interior de una base de datos, no directamente sobre el sistema operativo o la plataforma .NET como lo haría una aplicación Visual Basic. Con ellos es posible crear bloques de código, llámeselos desencadenadores, procedimientos almacenados o funciones, capaces de procesar datos y tomar decisiones actuán sobre el contenido de la base de datos. No podemos utilizarlos para crear una interfaz de usuario en una aplicación estándar.

Transact-SQL

Provenientes de una misma raíz, Sybase y SQL Server cuentan con un lenguaje llamado Transact-SQL que, con el tiempo, ha evolucionado de manera distinta en

cada DBMS. La versión de la que nos ocupamos en este apartado es la implementada en SQL Server, el RDBMS de Microsoft que acompaña a las ediciones superiores de Visual Studio .NET y que, en contraposición a otros productos como DB2 u Oracle, va ganando cuota de mercado en lugar de perderla.

Este lenguaje es aplicable también a MSDE, la solución de bases de datos de Microsoft para pequeñas instalaciones que no requieren la potencia de SQL Server. En realidad MSDE es el núcleo de SQL Server pero con menos capacidad de conexiones y proceso.

Variables y tipos de datos

Para poder procesar los datos, evaluando expresiones y efectuando operaciones, es preciso contar con un medio de almacenamiento temporal con el que no cuenta SQL, ya que éste opera siempre sobre datos almacenados en una tabla de la base de datos. En Transact-SQL podemos definir variables de distintos tipos, como haríamos en Visual Basic.

La sintaxis para declarar una variable es la siguiente:

```
DECLARE @Identificador Tipo
```

Observe que el identificador va precedido del símbolo @, que utilizaremos también al efectuar una asignación a la variable o recuperar su valor. En cuanto al tipo, tenemos a nuestra disposición tipos numéricos enteros y con parte decimal de distintas precisiones (int, smallint, bigint, decimal, money y float, entre otros), fechas (datetime y smalldatetime), cadenas de caracteres ASCII (char, varchar y text), cadenas de caracteres Unicode (nchar, nvarchar y ntext) y secuencias de datos binarios (binary, varbinary e image). Dos tipos especialmente interesantes son table y cursor, permitiendo el almacenamiento de un conjunto de resultados, en el primer caso, o el recorrido secuencial, fila a fila, de un conjunto de datos, en el segundo.

Declarada la variable, la asignación de un valor se efectúa mediante la sentencia SELECT. Dicho valor puede ser un valor constante, otra variable, el resultado de una expresión o, incluso, el resultado de la ejecución de una consulta. A continuación tiene un par de ejemplos:

```
DECLARE @NumLibros int  
SELECT @NumLibros = 5  
...  
SELECT @NumLibros = COUNT(ISBN) FROM Libros
```

En la primera asignación damos a NumLibros el valor 5, mientras que en la segunda contamos el número de filas que hay en la tabla Libros y guardamos el resultado en la variable.

Nota

Las variables también pueden ser de los tipos VARCHAR, INTEGER y demás tipos indicados anteriormente como estándar en SQL.

Evaluación de expresiones

Para componer expresiones, Transact-SQL cuenta prácticamente con los mismos operadores aritméticos y relacionales que encontramos en Visual Basic. Tenemos los cuatro operadores aritméticos básicos, +, -, * y /, además del operador % para obtener el resto de división entera. Los relacionales son los mostrados en la tabla 2.1.

Tabla 2.1. Operadores relacionales de Transact-SQL

Operador	Relación a evaluar
=	Igualdad
<	Menor que
<=	Menor o igual que
!<	No menor que
>	Mayor que
>=	Mayor o igual que
!>	No mayor que
<>	Desigualdad

El resultado de la expresión puede almacenarse en una variable, asignarse como contenido de una columna de una tabla o ser utilizado en otra expresión o sentencia condicional.

Condicionales y bucles

Las expresiones condicionales pueden utilizarse para ejecutar o no una sentencia, dependiendo del resultado, o bien controlar los ciclos de un bucle.

Las sentencias para codificar este tipo de lógica son IF, ELSE, WHILE, CONTINUE y BREAK. La forma de utilizarlas no difiere, en esencia, de las homónimas en Visual Basic.

IF, ELSE y WHILE afectan, por defecto, sólo a la sentencia que les siga. En caso de que nos interese que no sea una, sino múltiples las sentencias que se vean afectadas, delimitaremos entonces el bloque mediante las palabras BEGIN, al inicio, y END, al final.

Codificación de procedimientos almacenados

Una de las razones fundamentales de la existencia de Transact-SQL, según se apuntaba anteriormente, es la posibilidad de crear procedimientos almacenados, bloques de código que, almacenándose con un nombre en la base de datos, pueden ser ejecutados a demanda de la aplicación con el fin de obtener resultados mucho más elaborados que los que podrían tenerse con las consultas más complejas.

Para crear un procedimiento almacenado se utiliza la sentencia CREATE PROCEDURE, tras la cual dispondríamos el nombre del procedimiento y la palabra AS. Antes de ésta, y de manera opcional, se declararían los parámetros necesarios para invocar al procedimiento. Las sentencias de implementación se introducirían en un bloque BEGIN/END, devolviéndose el resultado mediante la sentencia RETURN.

El siguiente código sería un procedimiento almacenado muy básico pero que cuenta con algunos elementos importantes:

```
CREATE PROCEDURE NumTitulos
    @Editorial INTEGER = NULL
AS
BEGIN
    IF @Editorial IS NULL
        RETURN (SELECT COUNT(ISBN) FROM Libros)
    ELSE
        RETURN (SELECT COUNT(ISBN) FROM Libros
                WHERE Editorial = @Editorial)
END
```

Al invocar a este procedimiento, llamando NumTitulos, es posible facilitar un parámetro, concretamente el código de una editorial. De no hacerlo, el procedimiento devuelve el número total de títulos que hay en la tabla, mientras que en caso contrario se totalizan los títulos de la editorial indicada. Al ejecutarse este procedimiento almacenado, desde Visual Basic, se obtendría como resultado un número.

Obviamente, es posible crear procedimientos mucho más complejos, que retor-
nen uno o varios conjuntos de resultados o, incluso, que se encarguen de efectuar sobre la base de datos las operaciones adecuadas. Podría, por ejemplo, crearse un procedimiento almacenado al que facilitando una serie de parámetros se encarga-
se de insertarlos en sus respectivas tablas.

PL/SQL

El lenguaje PL/SQL es a Oracle lo que Transact-SQL a SQL Server, es decir, el lenguaje mediante el cual pueden codificarse procedimientos almacenados, desen-
cadenadores y funciones. Aunque la sintaxis, como va a ver brevemente en los
puntos siguientes, es distinta, lo cierto es que sus posibilidades son básicamente
las mismas.

Mediante PL/SQL es posible declarar variables, operar sobre ellas, evaluar expresiones, codificar condicionales y bucles, etc. Vamos a seguir exactamente el

mismo esquema del punto dedicado a Transact-SQL, prácticamente con los mismos ejemplos, pero con la sintaxis de PL/SQL.

Variábles y tipos de datos

En lugar de una sentencia `DECLARE` para la declaración de cada una de las variables, como en Transact-SQL, PL/SQL divide el código en dos secciones bien diferenciadas: una para declaraciones y otra para implementación. La primera se inicia, precisamente, con la palabra `DECLARE`, mientras que la segunda irá delimitada entre las palabras `BEGIN` y `END`.

Las variables, por tanto, se declararían tras la palabra `DECLARE` y antes de que se inicie el bloque de código ejecutable con la palabra `BEGIN`. La sintaxis de declaración es así de simple:

```
Identificador Tipo;
```

Observe el punto y coma al final de la declaración, es un elemento habitual de PL/SQL para indicar el final de las sentencias.

En cuanto a los tipos de datos posibles, básicamente son los que pueden emplearse en cualquier columna de una tabla Oracle. Tenemos números enteros (`NATURAL`, `POSITIVE`, `SIGNTYPE`), y decimales (`DECIMAL`, `DOUBLE PRECISION` y `FLOAT`), caracteres (`CHAR`) y cadenas de caracteres (`VARCHAR` y `STRING`), fechas (`DATE`) y algunos tipos más específicos como `ROWID` o `BFILE`, que actúan como punteros a una fila en una tabla o un archivo, respectivamente.

Los valores se asignan a las variables mediante el operador `:=` o bien, en caso de que el valor a asignar se obtenga de una consulta, con la cláusula `INTO` de la sentencia `SELECT`. Por ejemplo:

```
DECLARE
    NumLibros INTEGER;
BEGIN
    NumLibros := 5;
    ...
    SELECT COUNT(ISBN) INTO NumLibros
    FROM Libros;
    ...
END;
```

Primero declaramos la variable `NumLibros`, de tipo `INTEGER`, y, a continuación, efectuamos dos asignaciones. La primera asigna una constante y, por ello, se utiliza el operador `:=`. En la segunda el valor se recupera mediante una consulta.

Evaluación de expresiones

En lugar de un valor constante, como en el ejemplo anterior, una variable puede recibir el resultado de la evaluación de una expresión. Ésta se compondría de operandos, constantes y/o variables, y operadores aritméticos y/o relacionales.

Los operadores aritméticos son los habituales `+`, `-`, `*` y `/`, a los que habría que añadir el operador `**` que efectúa la potenciación. Los aritméticos `=`, `<`, `<=`, `>`, `=>` y `!=`. Los primeros son equivalentes a los indicados en la sección de Transact-SQL, mientras que el último comprueba la desigualdad entre dos operandos. Los operadores lógicos son `AND`, `OR` y `NOT`.

Utilizando estos operadores, la composición de expresiones se efectúa de la manera habitual y puede almacenarse el resultado, para un uso posterior, o bien emplearse directamente en un condicional o sentencia similar.

Condicionales y bucles

Si deseamos ejecutar una o varias sentencias dependiendo de que una cierta expresión sea cierta o no, utilizaremos la sentencia `IF/THEN/ELSE`, prácticamente idéntica a la que existe en Visual Basic. La expresión a evaluar o condicional irá tras la palabra `IF`, mientras que las sentencias a ejecutar si dicha expresión es cierta se pondrán tras la palabra `THEN`. Puede ser sólo una o múltiples sentencias, sin delimitar el bloque en forma alguna puesto que el final vendrá marcado por un `END IF`, un `ELSE` o bien un `ELSIF` en caso de que necesitemos evaluar expresiones adicionales:

```
IF Condición THEN
    sentencias;
ELSIF Condición THEN
    sentencias;
ELSE
    sentencias;
END IF;
```

En cuanto a los bucles, podemos usar dos tipos: `FOR` y `WHILE`. El primero se asemeja bastante al típico bucle por contador de Visual Basic, mientras que el segundo es un bucle condicional corriente. La sintaxis de ambos es la siguiente:

```
FOR Contador IN N .. M LOOP
    Sentencias;
END LOOP;
...
WHILE Condición LOOP
    Sentencias;
END LOOP;
```

La `N` y `M` del primer bucle se sustituirían por los valores extremos del bucle, por ejemplo 1 y 10, valores que irá tomando `Contador` mientras se ejecutan las sentencias que hay en el interior del bucle. En el segundo caso tan sólo tendría que sustituir `Condición` por la expresión condicional a evaluar en cada ciclo del bucle.

Codificación de procedimientos almacenados

PL/SQL es aplicable a la implementación de desencadenadores, funciones, bloques de código anónimos y, por supuesto, procedimientos almacenados, bloques

de código que podemos ejecutar a demanda desde las herramientas de administración o bien desde aplicaciones externas.

La creación de un procedimiento almacenado se inicia con CREATE PROCEDURE seguido del nombre del procedimiento y, opcionalmente y entre paréntesis, la lista de parámetros que recibirá el procedimiento. Éstos pueden tener un valor por defecto, especificado mediante la palabra clave DEFAULT. A continuación se dispone la palabra IS, el bloque de declaraciones de variables (sin el apartado DECLARE ya que AS actúa como punto de partida de dicho bloque) y el código entre los habituales BEGIN y END.

En caso de que necesitemos devolver un resultado tenemos dos opciones: definir un parámetro de salida en la lista de argumentos del procedimiento o, en su lugar, implementarlo como función, cambiando PROCEDURE por FUNCTION e indicando al final de la cabecera, antes de la palabra IS, el tipo a devolver mediante RETURN.

Suponiendo que deseásemos codificar el mismo procedimiento almacenado NumTitulos usado como ejemplo anteriormente, en PL/SQL lo haríamos así:

```
CREATE FUNCTION NumTitulos(
    IDEditorial IN INTEGER DEFAULT 0)
RETURN NUMBER
IS
    N NUMBER;
BEGIN
    IF IDEditorial = 0 THEN
        SELECT COUNT(ISBN) INTO N FROM Libros;
    ELSE
        SELECT COUNT(ISBN) INTO N FROM Libros
            WHERE Editorial = IDEditorial;
    END IF;
    RETURN N;
END;
```

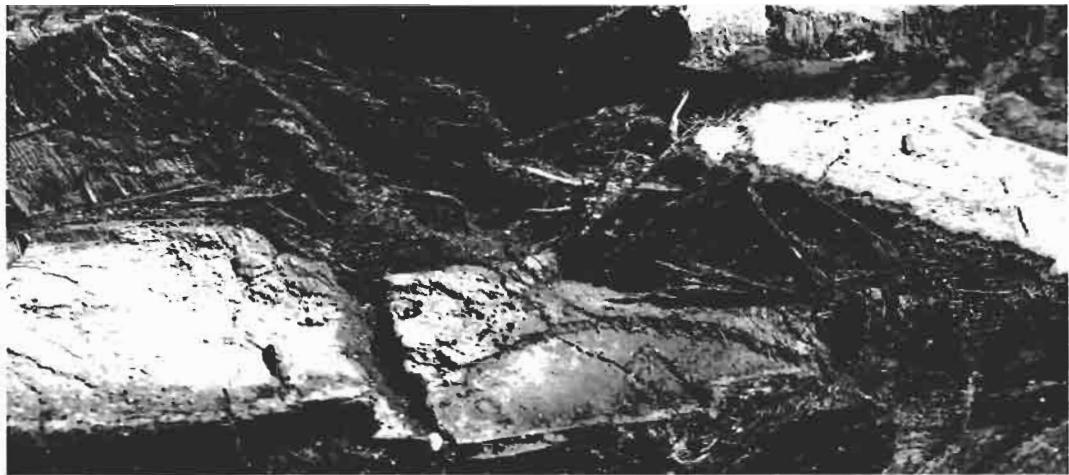
El funcionamiento y resultado de la ejecución de esta función sería como el descrito antes para el procedimiento almacenado Transact-SQL. Si facilitamos un código de editorial se obtiene el número de títulos con que cuenta, de lo contrario se retorna el número total de títulos.

Resumen

Como ha podido ver en este capítulo, el lenguaje SQL, en general, y los derivados de cada RDBMS, en particular, suponen un extenso campo de estudio y aplicación sobre el tratamiento de datos en sistemas como SQL Server u Oracle. Nuestro objetivo era introducirle en ese campo, aunque apenas arañando el mundo de posibilidades existentes en él. Ha aprendido, no obstante, a utilizar sentencias DDL para definir estructuras de datos básicas, como las tablas; DML para manipular esos datos y DCL para otorgar y denegar privilegios a los usuarios.

Aunque muy brevemente, también se han descrito algunas de las características de Transact-SQL y PL/SQL, dos de los lenguajes más importantes para programación de bases de datos al ser los productos asociados, SQL Server y Oracle respectivamente, dos de los más extendidos en el mercado.

Parte de los conocimientos que ha adquirido en este capítulo le serán útiles en el próximo, donde los pondrá en práctica al usar algunas herramientas de DBMS, y, por supuesto, en los posteriores, al introducirnos en ADO.NET.



3

Orígenes de datos

En el primer capítulo, dedicado a terminología y conceptos generales, conocimos lo que era un origen de datos y, a modo de enumeración, se citaron algunos de los orígenes a los que puede accederse mediante ADO.NET, como las bases de datos, de escritorio y RDBMS, documentos XML o bien hojas de cálculo Microsoft Excel. No entraremos, entonces, en mayores detalles.

Para poder trabajar desde Visual Basic con ADO.NET, en los capítulos posteriores, es indispensable contar con los orígenes de datos que van a utilizarse. Por ello, en este capítulo va a conocer algunas de las herramientas y aplicaciones que le permitirán crear y configurar dichos orígenes de datos, así como introducir en ellos algunos datos de partida.

Tras algunos conceptos adicionales, aprenderá a utilizar Microsoft Access, SQL Server, Oracle, InterBase y Excel con el fin de preparar algunas bases de datos y documentos. Las indicaciones dadas serán las suficientes para cada necesidad. Recurra a la documentación de cada producto, o a bibliografía adicional, si le interesa profundizar en alguna de esas aplicaciones. También conocerá algunas bases sobre XML y el Directorio activo.

Orígenes locales y remotos

Un origen de datos, siempre hablando respecto a la aplicación, componente o servicio que va a acceder a él, puede ser *local* o bien *remoto*. Hablamos de origen de

datos local cuando tanto la información como el *middleware* preciso para acceder a él se encuentran en el mismo ordenador, conjuntamente con la aplicación. Es el caso típico en configuraciones *monousuario*, en el que el ordenador aloja programas y datos e, incluso, no está conectado a otros equipos en red.

En ocasiones, el acceso a esa información local puede efectuarse directamente desde ADO.NET, sin necesidad de intermediario alguno. Es lo que ocurre, por ejemplo, al trabajar con documentos XML tratándolos como si fuesen conjuntos de datos. En otras, por el contrario, que la información esté en el mismo equipo donde se ejecuta ADO.NET no implica que pueda manipularse sin más, precisándose ese intermediario en forma de controlador. Un caso así se encuentra al operar sobre una base de datos Microsoft Access o dBase que está en el mismo ordenador.

Cuando el origen de datos es remoto, por ejemplo una base de datos alojada en un servidor y gestionada por un RDBMS, siempre es preciso contar en el ordenador donde se ejecuta la aplicación, el ordenador del usuario final, con un *software cliente* dependiente del RDBMS que se emplee. ADO.NET, o el controlador que corresponda, se comunicará con ese software cliente que, a su vez, se comunicará con el servidor a través de la infraestructura de red correspondiente. Lógicamente, es necesario tener instalado el software cliente del RDBMS y configurados los parámetros de comunicación entre él y el servidor. Éstos son aspectos que quedan fuera del ámbito de este capítulo, en el que se asume que utilizará las herramientas indicadas desde el mismo ordenador en que está instalado el servidor y se encuentran los datos aunque, en la práctica, las operaciones indicadas pudieran efectuarse también remotamente.

Microsoft Access

Este producto, que forma parte de la *suite* Microsoft Office, es uno de los gestores de bases de datos más populares del mercado, especialmente en instalaciones en las que el propio usuario opera sobre la información, utilizando la interfaz de esta aplicación, o bien utiliza una aplicación desarrollada con VBA que se ejecuta directamente dentro de Access.

Desde una aplicación Visual Studio .NET podemos necesitar recuperar la información que el usuario gestiona habitualmente utilizando Access o, incluso, crear una aplicación que se encargue de actuar como interfaz para ese usuario a fin de que no tenga que aprender a usar Access. En cualquier caso, en este apartado vamos a conocer algunos elementos de esta aplicación, los necesarios para crear una base de datos y editar su contenido.

Comience por iniciar Microsoft Access, en este caso suponemos que la versión es la 2000, se encontrará con un cuadro de diálogo similar al de la figura 3.1. Seleccione la opción **Base de datos de Access en blanco** y pulse el botón **Aceptar**.

Introduzca el camino y nombre de la base de datos a crear. La que va a describirse a continuación la encontrará en la carpeta **Ejemplos** del CD-ROM con el nombre **Libros.mdb**.

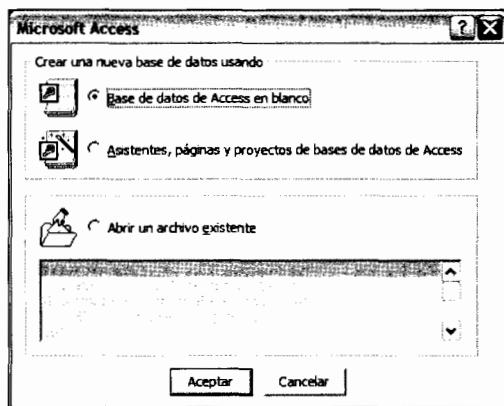


Figura 3.1. Creamos una base de datos Access en blanco

Definición de las tablas

Dado el paso anterior, creando la base de datos, encontrará una ventana similar a la de la figura 3.2. En el margen izquierdo aparecen una serie de botones que dan acceso a las distintas categorías de elementos que pueden existir en una base de datos Access, mientras que a la derecha se enumeran las opciones que en ese momento hay disponibles en la categoría seleccionada.

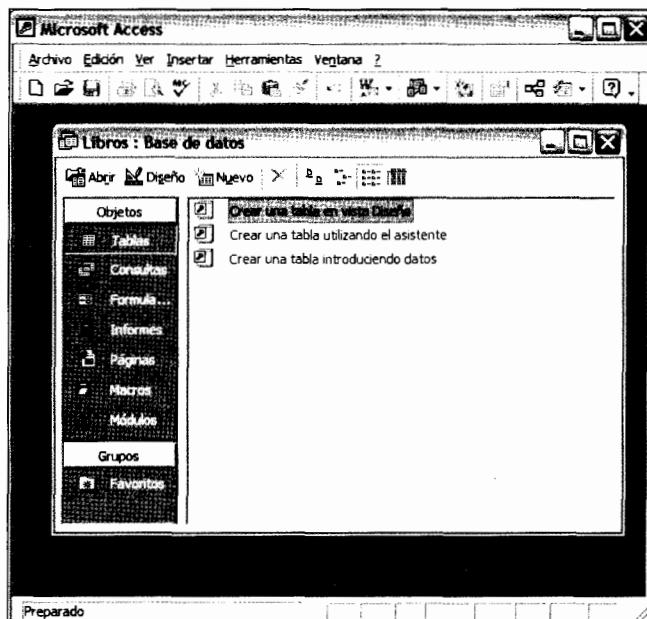


Figura 3.2. Ventana de trabajo sobre la base de datos

Seleccione la opción **Crear una tabla en vista Diseño**, la primera que aparece en la lista de la sección **Tablas**, simplemente haciendo doble clic sobre ella. Aparecerá una nueva ventana, con una especie de cuadrícula, en la que deberá introducir el nombre y tipo de cada una de las columnas de la tabla que va a crearse. Comenzaremos con las columnas de la tabla **Editoriales**, como se aprecia en la figura 3.3. Las columnas y sus atributos serán los siguientes:

- **IDEditorial**: Será el identificador de cada editorial. Seleccione el elemento **Autonumérico** de la lista de tipos de datos y pulse el botón **Clave principal** que hay en la parte superior. De esta forma la columna obtendrá automáticamente un valor numérico incrementado para cada fila y que, además, actuará como clave primaria, generándose para ella un índice en el que no se permitirán duplicados.
- **Nombre y Dirección**: Dejaremos estas columnas con su tipo y atributos por defecto, sirviendo para introducir el nombre y dirección de la editorial, respectivamente.

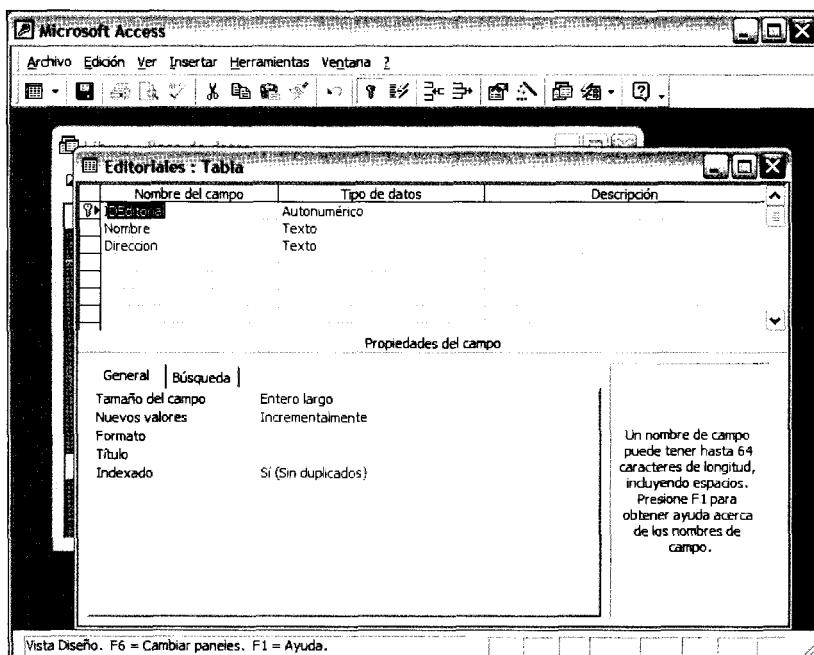


Figura 3.3. Definimos la estructura de la primera tabla

Al cerrar la ventana aparecerá un pequeño cuadro de diálogo en el que debe introducirse el nombre de la tabla, tras lo cual volverá a la ventana de la figura 3.2. Seleccione de nuevo la opción **Crear una tabla en vista Diseño** a fin de definir las columnas de la tabla **Libros**, que serán éstas:

- **IDLibro:** Identificador de cada libro, de tipo Autonumérico y clave primaria de la tabla.
- **ISBN:** El tipo será **Texto**. Ajuste el **Tamaño del campo** a 13 caracteres, en la parte inferior, y dé el valor **Sí** a la opción **Requerido**.
- **Titulo:** Queda con los atributos por defecto ofrecidos por Access.
- **Autor:** Lo dejamos con el tipo **Texto** pero ajustando el **Tamaño del campo** a 30 caracteres.
- **Editorial:** Seleccionamos el tipo **Numérico** de la lista desplegable y, en la parte inferior, damos el valor **Sí** al apartado **Requerido**.
- **Precio:** Seleccionamos el tipo **Moneda** como tipo de dato.

Con esto finalizamos la creación de la segunda tabla con que constará nuestra base de datos. Cerramos la ventana y llamamos **Libros** a la tabla, volviendo de nuevo al cuadro de diálogo de la figura 3.2.

Relación entre las tablas

Está claro que las dos tablas que hemos creado en el punto anterior guardan una relación. Concretamente, la columna **Editorial** de la tabla **Libros** alojará el identificador de una entrada de la tabla **Editoriales**. En cierta manera, la tabla **Editoriales** actuará como primaria en una relación maestro/detalle o uno-a-muchos respecto a la tabla **Libros**.

Encontrándose en el cuadro de diálogo de la figura 3.2, en el que ya aparecerán las dos tablas recién creadas, pulse sobre el botón **Relaciones** que hay en la paleta principal. Se abrirá un primer cuadro de diálogo del que podrá elegir las tablas a añadir a la relación. Seleccione las dos creadas, añadiéndolas y después cerrando el cuadro de diálogo. Accederá a otra ventana, similar a la de la figura 3.4, en la que aparecen representadas las dos tablas. Sitúe el punto del ratón sobre la columna **Editorial** de la tabla **Libros**, pulse el botón principal del ratón y, sin soltarlo, muévase hasta la columna **IdEditorial** de la tabla **Editoriales**.

Una vez libere el botón del ratón, se abrirá un nuevo cuadro de diálogo (véase figura 3.5) mostrando múltiples opciones. Active las opciones **Exigir integridad referencial** y **Actualizar en cascada los campos relacionados**. Así consigue dos objetivos: impedir que en la tabla **Libros** pueda introducirse un identificador que no exista en la tabla **Editoriales** y, por otra parte, actualizar automáticamente las referencias desde la tabla **Libros** en caso de que se modifique el identificador de una editorial, algo altamente improbable. Pulse el botón **Crear**.

Introducción de datos

El siguiente paso lógico, una vez creadas las estructuras de la base de datos, es ahora introducir alguna información en las tablas. Siempre partiendo del cuadro

de diálogo mostrado en la figura 3.2, haga doble clic sobre la tabla **Editoriales**, o bien selecciónela y pulse **Intro**, para abrir la cuadrícula de introducción de datos.

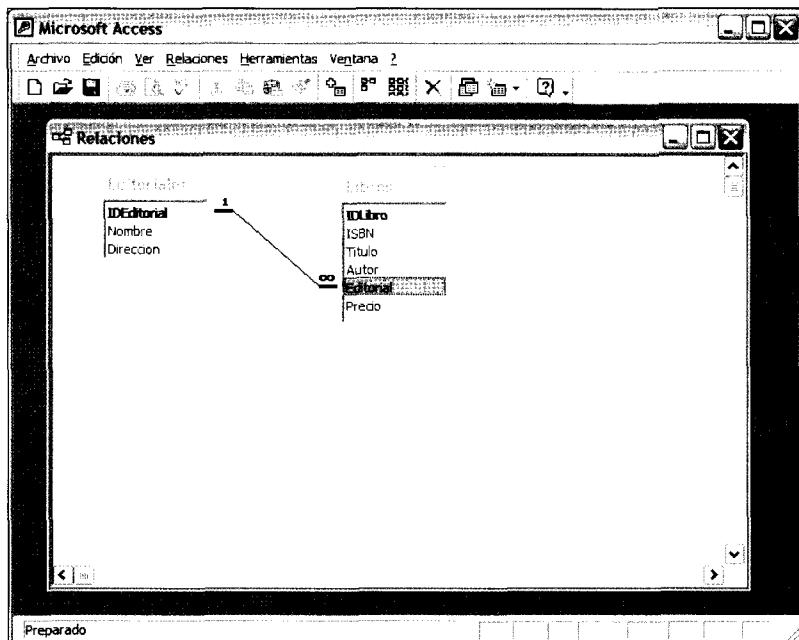


Figura 3.4. La relación entre las dos tablas una vez establecida

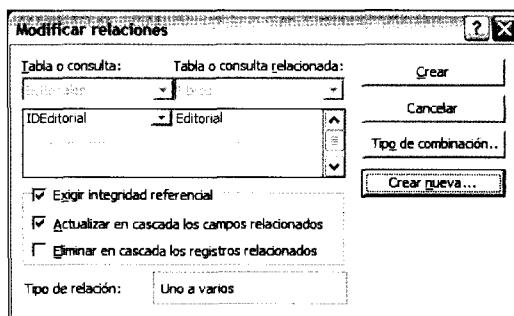


Figura 3.5. Opciones de la relación entre las dos tablas

La primera columna, **IDEitorial**, aparece con el texto **(Autonumérico)** en su interior. No es necesario introducir dato alguno en ella, en el momento en que comencemos a escribir el nombre de una editorial se establecerá automáticamente su código, evitando así un posible fallo humano de asignación a dos o más editoriales del mismo código.

En la figura 3.6 puede ver cómo se introducen varias editoriales. Tan sólo hay que ir escribiendo el nombre y dirección de cada una de ellas, pulsando la tecla

Tab para ir avanzando de una columna a la siguiente y de una fila a otra. Introducidos los datos, cerramos la ventana para retornar al cuadro de diálogo de tareas. En este momento tenemos algunas editoriales en la tabla **Editoriales**, totalmente imprescindible para trabajar con la tabla **Libros**.

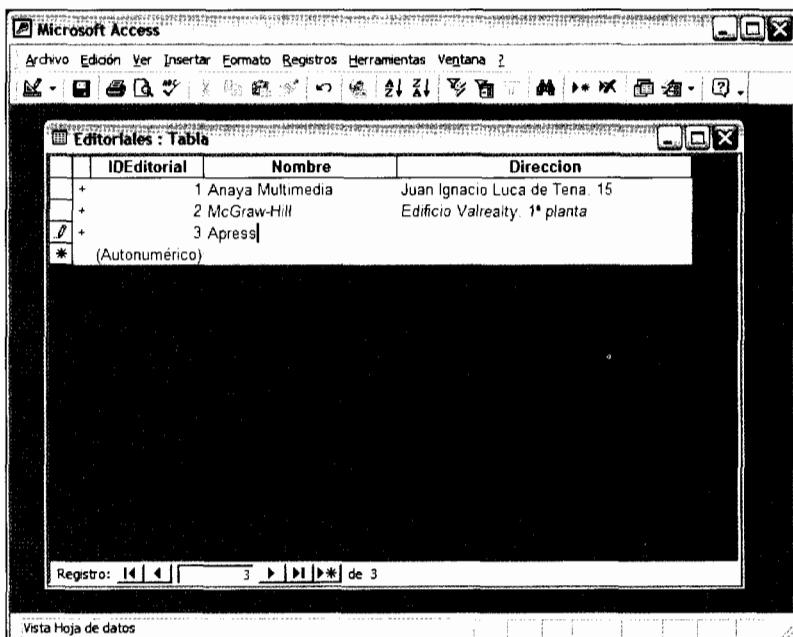


Figura 3.6. Edición del contenido de la tabla **Editoriales**

Pruebe a hacer ahora lo mismo en la tabla **Libros**, introduciendo los datos de algunos libros que tenga a mano. Compruebe que si introduce un código de editorial que no exista, por no haberse creado previamente en la tabla **Editoriales**, aparece un mensaje de error como el de la figura 3.7. Esto es la integridad referencial.

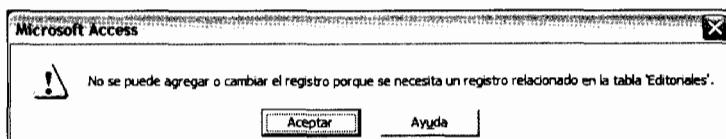


Figura 3.7. Access asegura la integridad referencial impidiéndonos introducir libros de editoriales inexistentes

Simplificar la selección de editorial

Aunque el objetivo que nos marcamos aquí no es otro que tener una base de datos Access con alguna información útil para utilizarla desde ADO.NET, creando

aplicaciones para el usuario, lo cierto es que para nosotros mismos no resulta demasiado cómodo tener que introducir el código de cada editorial de manera manual, escribiendo dicho código que o tenemos que recordar o bien comprobar en la tabla Editoriales.

Podemos simplificar esta operación de forma muy simple. Seleccione la tabla Libros, en el cuadro de diálogo de la figura 3.2, y pulse sobre el botón Diseño, abriendo así de nuevo la ventana de diseño de la tabla. Seleccione la columna Editorial y, en la parte inferior, abra la página Búsqueda. En ella encontrará opciones que permiten asignar a esta columna un control de búsqueda. Seleccione los valores que se aprecian en la figura 3.8 para cada uno de los apartados. Opcionalmente puede establecer el ancho de la columna y otros parámetros.

Ahora, al ir a introducir los datos de un libro, observará que la columna Editorial tiene asociada una lista desplegable. Puede escribir el código de la editorial o, si lo prefiere, abrir dicha lista y elegirlo, como se hace en la figura 3.9. No cabe duda de que resulta mucho más cómodo, especialmente si son muchas las editoriales existentes y no resulta fácil recordar el identificador de cada una de ellas.

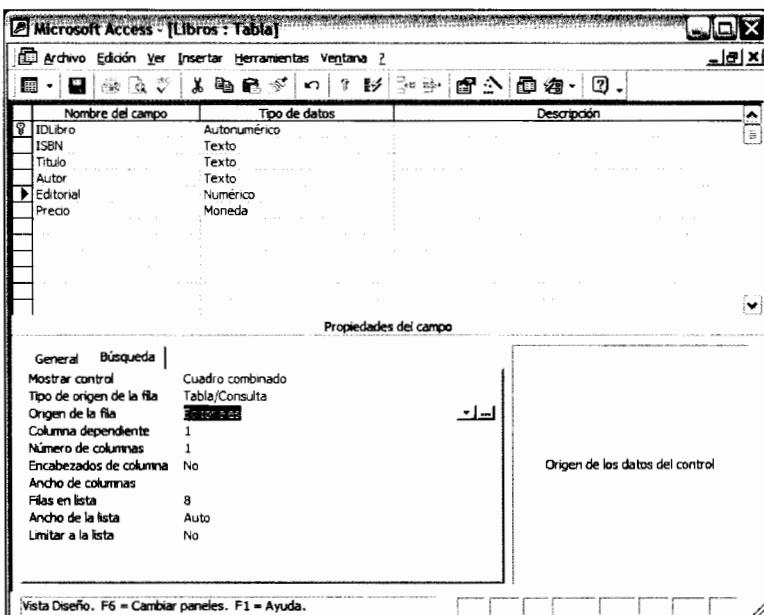


Figura 3.8. Configuramos la columna con un control de búsqueda

Nota

Introduzca algunos datos en las tablas para tener información con la que trabajar en capítulos posteriores o, si lo prefiere, tome la base de datos que se facilita en el CD-ROM, en la que existen algunas entradas.

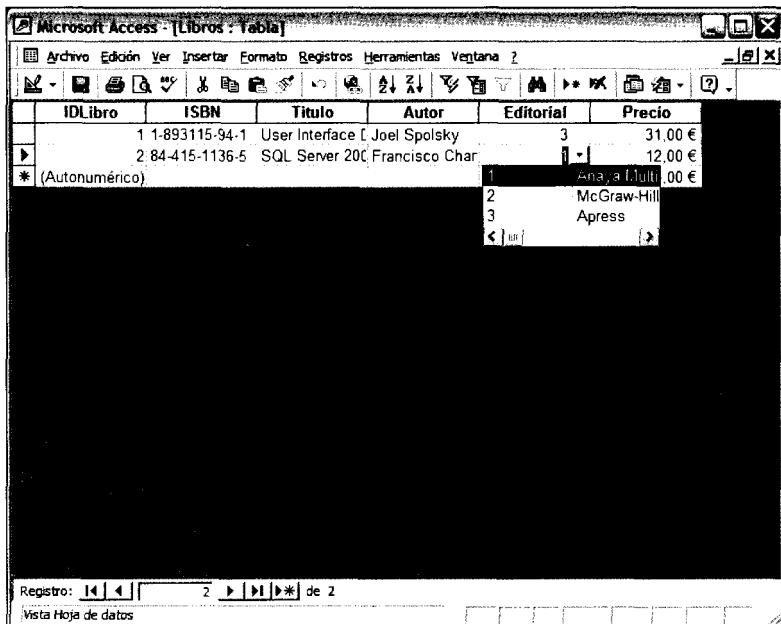


Figura 3.9. Podemos buscar el código de la editorial en lugar de escribirlo manualmente

SQL Server

A diferencia de Microsoft Access, SQL Server es un completo RDBMS, un servidor disponible en varias ediciones en el que es posible utilizar un lenguaje, Transact-SQL, para programar procedimientos almacenados y desencadenadores, aparte de poder definirse tablas, índices, vistas, etc. Visual Basic .NET cuenta con un proveedor ADO.NET nativo para trabajar con SQL Server, ofreciendo el mejor rendimiento posible. Como se indicó en el capítulo previo, MSDE es el motor de SQL Server pero, a diferencia de éste, no dispone de las herramientas de administración que vamos a usar en los puntos siguientes. Para operar sobre MSDE, en caso de que no disponga de SQL Server, tendría que usar el propio Microsoft Access como interfaz de usuario.

Nota

Puede solicitar una versión de evaluación de Microsoft SQL Server, válida durante 120 días, en <http://www.microsoft.com/spain/servidores/sql/productinfo/evaluate.asp>. Es una buena opción si desea probar este producto. En este ejemplo se utiliza la versión 2000 del producto operando sobre un sistema Windows XP Professional.

Suponiendo que vamos a operar sobre el mismo servidor en el que se encuentra instalado SQL Server, abra la lista del menú **Programas** y localice el grupo **Microsoft SQL Server**, seleccionando la opción **Administrador corporativo**. Se encontrará con una interfaz similar a la de la figura 3.10. El único elemento disponible en la carpeta **Grupo de SQL Server** es el servidor **(local)**, en cuyo interior encontramos múltiples carpetas con distintos objetos.

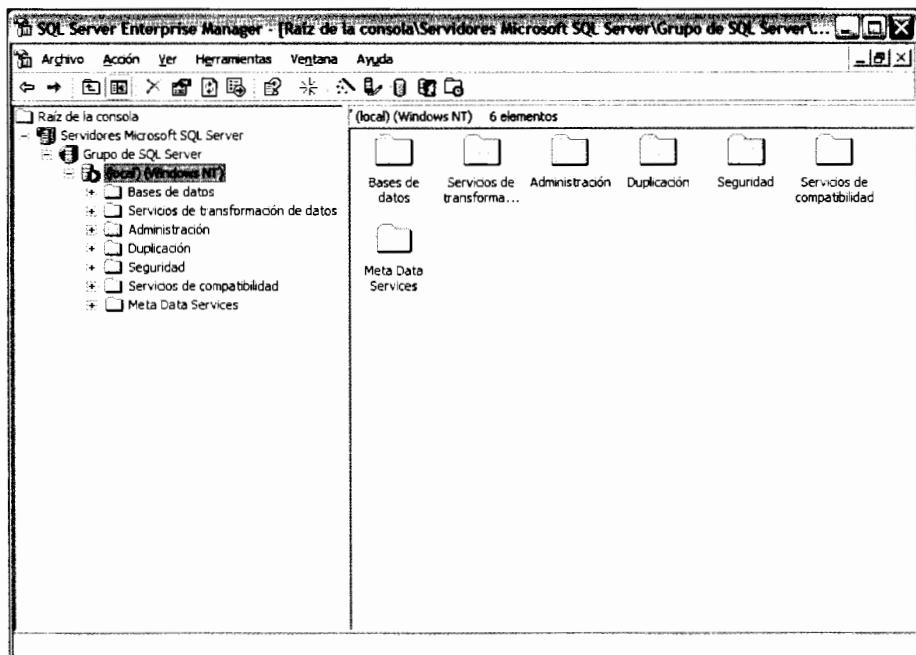


Figura 3.10. Interfaz del Administrador corporativo

Creación de la base de datos

Haga clic con el botón secundario del ratón sobre la carpeta **Bases de datos** y elija **Crear nueva base de datos**. Se abrirá la página **Propiedades de la base de datos**, en la que debemos introducir el nombre interno de la base y los datos de los archivos en los que se almacenará físicamente. En la primera página (figura 3.11) facilite el nombre de la base de datos, *Libros*, y deje el resto de opciones como están.

Notes

Aunque en este caso vamos a utilizar un asistente para efectuar el proceso de creación de la base de datos, también podríamos abrir el Analizador de consultas SQL e introducir las sentencias SQL de creación de la base y las tablas.

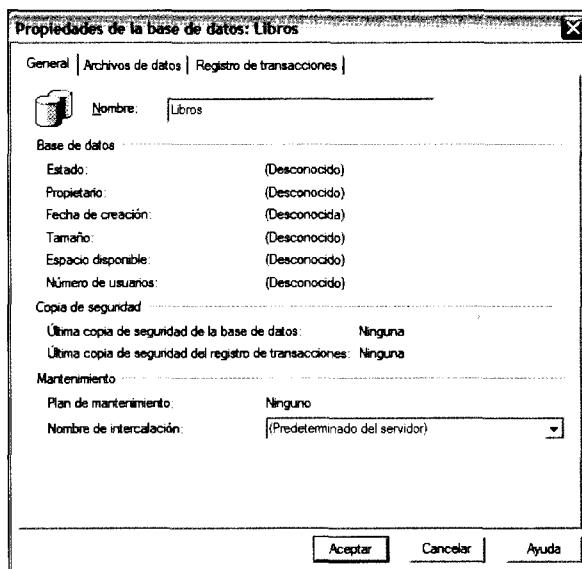


Figura 3.11. Introducimos el nombre lógico de la base de datos

Las otras dos páginas del asistente se utilizan para configurar los archivos de trabajo de la base de datos. Ésta puede contar con uno o más archivos para datos, que se configuran en la página **Archivos de datos**, y uno o más archivos de registro de transacciones, a configurar en la página **Registro de transacciones**. Estos archivos tienen un tamaño inicial, pueden crecer automáticamente y contar con un límite.

Dado que nuestra base de datos no va a contener una gran cantidad de información, ni va a experimentar un volumen de transacciones importante, dejaremos los parámetros por defecto para ambos archivos. Puede, si lo desea, modificar el camino y nombre de los archivos, como se ha hecho en la figura 3.12. En este caso se ha aceptado el nombre por defecto, si bien se ha modificado el camino en el que residirán. En este momento la base de datos ya está creada y aparece en la carpeta **Bases de datos** del servidor SQL Server. Hemos dado el primer paso.

Definición de las tablas

A diferencia de lo que ocurrió con Access, que justo tras la creación de la base de datos nos encontrábamos con un recipiente vacío, la base de datos SQL Server recién creada cuenta con una serie de tablas de sistema, diversas funciones y usuarios. Es decir, no está vacía. Tendremos, no obstante, que definir las estructuras de datos que necesitamos, comenzando por las tablas donde se alojará la información. El proceso es similar al descrito previamente para Access dado que la interfaz de usuario es parecida. Abra la carpeta **Bases de datos** y también la base de datos **Libros**. Haga clic con el botón secundario del ratón sobre el elemento **Tablas** y seleccione la opción **Nueva tabla**.

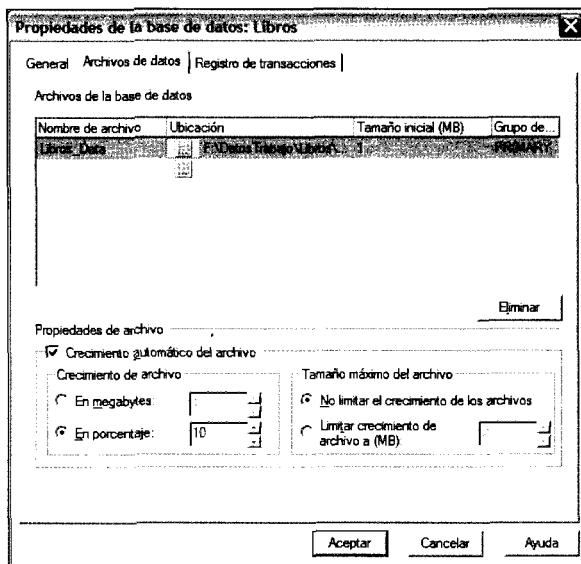


Figura 3.12. Opciones de los archivos de datos

Deberá introducir información para las tres columnas siguientes:

- **IDEditorial:** Seleccione el tipo `int` de la lista de datos y, en la parte inferior, despliegue la lista adjunta al apartado **Identidad** para seleccionar el valor **Sí** (no disponible para duplicación): Por último, pulse el botón **Establecer clave principal**.
- **Nombre y Dirección:** Seleccione el tipo `varchar` de la lista **Tipo de datos**, dejando el resto de parámetros con sus valores por defecto.

Los atributos de la columna `IDEditorial` le convierte en el atributo de identidad de cada fila. Observe que en la parte inferior (véase figura 3.13) aparece el valor inicial que tomará esta columna, así como el incremento que se aplicará a cada fila sucesiva. En definitiva, tenemos un campo autoincrementado que actúa como identificador único, que es lo que nos interesa en este momento.

Seleccione de nuevo la opción **Nueva tabla** para proceder con la creación de la tabla `Libros`. En este caso las columnas y sus atributos serán:

- **IDLibro:** Tendrá los mismos atributos que la columna `IDEditorial` de la tabla anterior, actuando como atributo identidad de cada fila y clave primaria.
- **ISBN:** Elegimos el tipo `char` y le damos 13 caracteres de longitud. En este caso el tipo `varchar` no aporta beneficio alguno ya que un ISBN siempre tiene esa longitud. Desactive la opción **Permitir valores nulos**, obligando así a introducir siempre el ISBN del libro.
- **Título y Autor:** Seleccione el tipo `varchar` y deje la longitud por defecto.

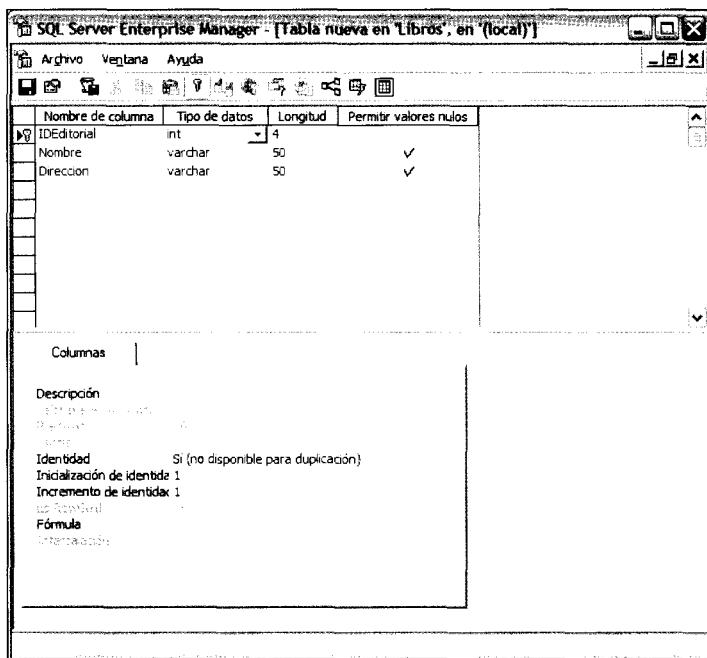


Figura 3.13. Creación de la tabla Editoriales

- **Editorial:** Su tipo será int y desactivamos la opción Permitir valores nulos. Contendrá el identificador de la editorial, para lo cual estableceremos después la relación adecuada.
- **Precio:** Elija el tipo de dato money.

En este momento tenemos definidos todos los atributos de la tabla Libros, tal y como se aprecia en la figura 3.14, pero no cierre aún la ventana.

Relación entre las tablas

Encontrándose aún en la ventana mostrada en la figura 3.14 (puede volver a ella, en caso de haberla cerrado, haciendo clic con el botón secundario del ratón sobre la tabla Libros, en el panel derecho del Administrador corporativo, y seleccionando luego la opción Diseñar tabla) pulse el botón Administrar relaciones. Se abrirá el cuadro de diálogo Propiedades con todas las propiedades de la tabla, abriéndose inicialmente la página Relaciones.

Pulse el botón Nueva que hay debajo de la lista combinada Relación seleccionada, procediendo así a la creación de una nueva relación. Se crea automáticamente un nombre para esta relación. En la parte inferior seleccione la columna IDEditorial de la tabla Editoriales, a la izquierda, y la columna Editorial de la tabla que está creándose, que es Libros. Active la opción Actualizar en cascada los

campos relacionados. El resto de las opciones de integridad referencial están marcadas por defecto, tal como puede verse en la figura 3.15, a excepción del borrado en cascada. Al hacer clic sobre el botón **Cerrar** se establecerá la relación entre las dos tablas. Ahora puede cerrar la ventana en la que estaba definiendo la tabla **Libros** asignándole dicho nombre.

Nombre de columna	Tipo de datos	Longitud	Permitir valores nulos
IdLibro	int	4	
ISBN	char	13	
Título	varchar	50	✓
Autor	varchar	50	✓
Editorial	int	4	
Precio	money	8	✓

Identity: Sí (no disponible para duplicación)
Inicialización de identidad: 1
Incremento de identidad: 1
Fórmula: 1

Figura 3.14. Enumeración de las columnas de la tabla **Libros** y sus atributos

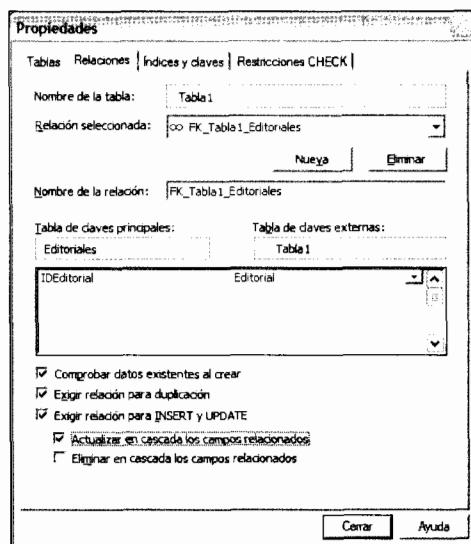


Figura 3.15. Definimos la relación existente entre las dos tablas

Nota

También podríamos haber definido la relación posteriormente, tras haber completado la creación de las dos tablas, ya sea volviendo al cuadro de diálogo de diseño o bien creando un diagrama visual de la estructura de la base de datos.

Con este paso hemos finalizado, por ahora, la definición de estructuras en la base de datos.

Introducción de datos

A diferencia de Microsoft Access, la herramienta de administración de SQL Server no está pensada para facilitar la introducción de datos por parte del usuario porque se supone que éste usará alguna aplicación a medida para efectuar su trabajo. Podemos introducir datos en las tablas que acabamos de crear, pero sin las comodidades de poder contar, por ejemplo, con un control de búsqueda para las editoriales y otras ayudas. Seleccione de la carpeta Tablas la tabla Editoriales, haga clic con el botón secundario del ratón y elija la opción Abrir tabla>Devolver todas las filas. En ese momento SQL Server generará una consulta para recuperar las filas existentes de la tabla, en este momento está vacía, y mostrará el resultado en una cuadrícula. En esa misma cuadrícula puede introducir los datos, como se hace en la figura 3.16, dejando que el administrador vaya generando las instrucciones SQL necesarias para que SQL Server inserte los datos. Deje en blanco la columna IDEditorial, a medida que vaya pasando de una fila a otra SQL Server se encargará de darle un valor consecutivo y único que actuará como clave principal.

The screenshot shows the Microsoft SQL Server Enterprise Manager interface with the title bar 'SQL Server Enterprise Manager - [Datos en tabla 'Editoriales' en 'Libros' en '(local)']'. The menu bar includes 'Archivo', 'Ventana', and 'Ayuda'. Below the menu is a toolbar with various icons. A data grid displays the 'Editoriales' table with three rows of data:

IDEditorial	Nombre	Direccion
1	Anaya Multimedia	Juan Ignacio Luca de Tena, 15
2	McGraw-Hill	Edificio Valerealty, 1 ^a planta
3	Apress	901 Grayson Street
*		

Figura 3.16. Introducimos los datos de varias editoriales

De manera análoga puede introducir los datos de algunos libros, teniendo en cuenta que el código de editorial deberá ser uno de los previamente definidos en la tabla `Editoriales`. Como ocurría con Access, de introducirse un código inexistente obtendremos un mensaje de error (véase figura 3.17) y la operación de inserción no se llevará a cabo.

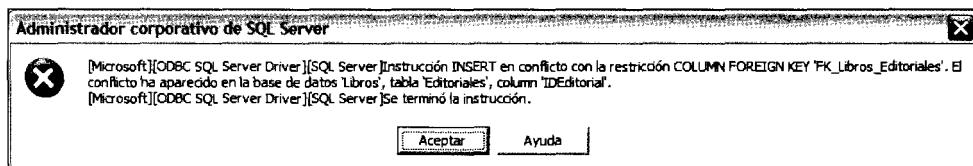


Figura 3.17. Error producido al introducir un código de editorial inexistente

Uso de la base de datos de ejemplo

Aunque puede definir la estructura de la base de datos e introducir información en ella, siguiendo las indicaciones dadas en los puntos previos, en la carpeta de ejemplo correspondiente a este capítulo encontrará los archivos MDF y LDF que componen la base de datos. Para utilizarla, sin embargo, no basta con abrirla como haría en Microsoft Access, el proceso es algo más complejo.

Abra el Administrador corporativo de SQL Server, abra el servidor local y haga clic con el botón secundario del ratón sobre la carpeta **Bases de datos**. Elija la opción **Todas las tareas>Adjuntar bases de datos**, abriendo el cuadro de diálogo que puede verse en la figura 3.18. Introduzca el camino y nombre de la base de datos, pulse el botón **Comprobar** y, finalmente, facilite el nombre con el que desea adjuntarla en su sistema y la cuenta del usuario que figurará como propietario.

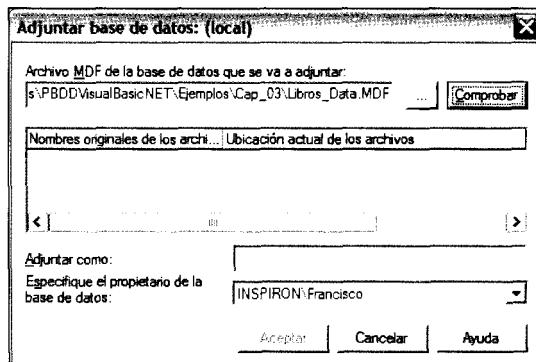


Figura 3.18. Importamos una base de datos en SQL Server

A partir de dicho momento puede utilizar la base de datos como si la hubiese creada en su sistema.

Definición de una vista

Como se apuntaba anteriormente, SQL Server es un RDBMS con todas las posibilidades que tienen estas aplicaciones, posibilidades algunas inexistentes en Microsoft Access. Una de esas posibilidades es la definición de vistas, objetos que se almacenan en la base de datos y ejecutan a demanda de las aplicaciones cuando éstas las abren como si se tratases de tablas. El uso de las vistas ahorra a las aplicaciones y usuarios la composición de consultas más o menos complejas para poder obtener los datos que necesitan.

Vamos a definir una vista que, aunque simple, nos servirá para conocer los pasos que habríamos de dar en cualquier caso. Dicha vista devolverá las editoriales y los títulos que les corresponden, estableciendo una relación entre ambas tablas. Los pasos a dar son los indicados a continuación:

1. Haga clic con el botón secundario del ratón sobre la carpeta **Vistas** de la base de datos y luego seleccione del menú emergente la opción **Nueva vista**. Aparecerá una ventana con varias secciones, como la de la figura 3.19, inicialmente vacías.

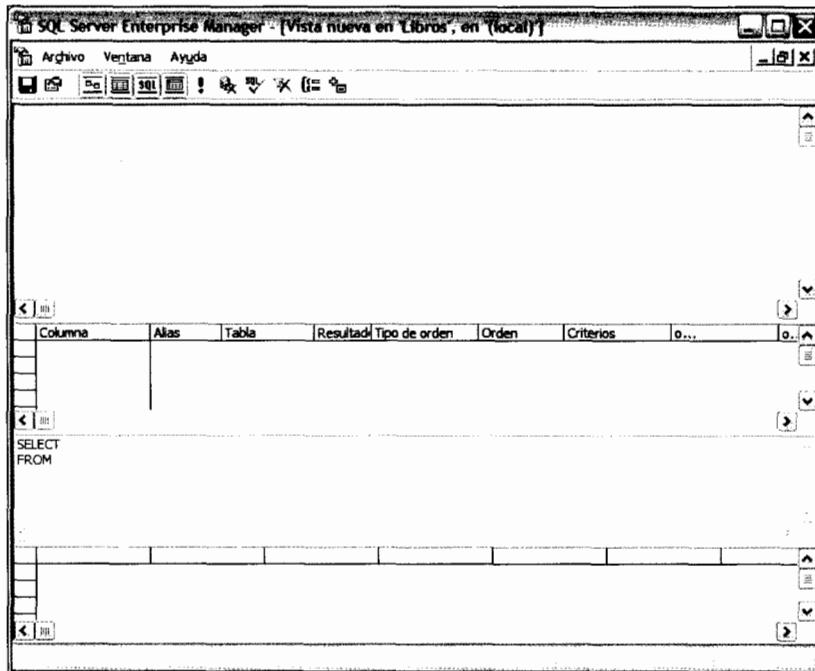


Figura 3.19. Iniciamos la creación de una nueva vista

2. Haga clic sobre el botón **Agregar tabla**, el último que aparece en la barra de botones, para abrir el cuadro de diálogo **Agregar tabla**. Seleccione las tablas

Editoriales y Libros y pulse el botón **Agregar**. Pulse el botón **Cerrar** para volver a la ventana anterior que, en este momento, mostrará el aspecto de la figura 3.20. Aparecen las dos tablas, con su relación, y una consulta SQL en la parte central.

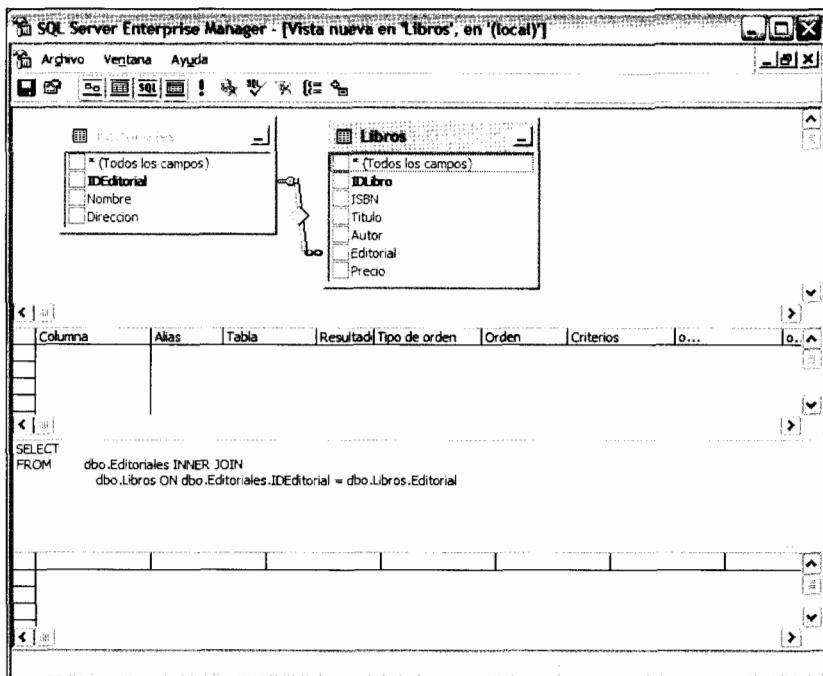


Figura 3.20. Añadimos las tablas a la vista

3. Haga clic con el botón principal del ratón sobre las casillas que aparecen a la izquierda de las columnas **Nombre**, en la tabla **Editoriales**, y **Título** y **Precio**, en la tabla **Libros**. De esta manera las selecciona como columnas para el resultado final. Observe cómo esas columnas aparecen en el panel que hay debajo del diagrama y también en la consulta SQL.
4. Utilice el panel que aparece debajo del diagrama para establecer criterios de búsqueda u ordenación. Puede, por ejemplo, desplegar la lista de la columna **Orden** asociada a la **Precio** para que los datos se ordenen ascendente-mente por el precio, es decir, de más baratos a más caros. También puede introducir condiciones de búsqueda, por ejemplo que el precio esté en un cierto rango.
5. Pulse el botón **Ejecutar**, identificado con un signo de admiración, para probar la vista y obtener, en la parte inferior, el resultado, como se ve en la figura 3.21. Puede modificar los criterios y el orden según le interese. En la base de datos de ejemplo no existen criterios y el orden es el obtenido por defecto.

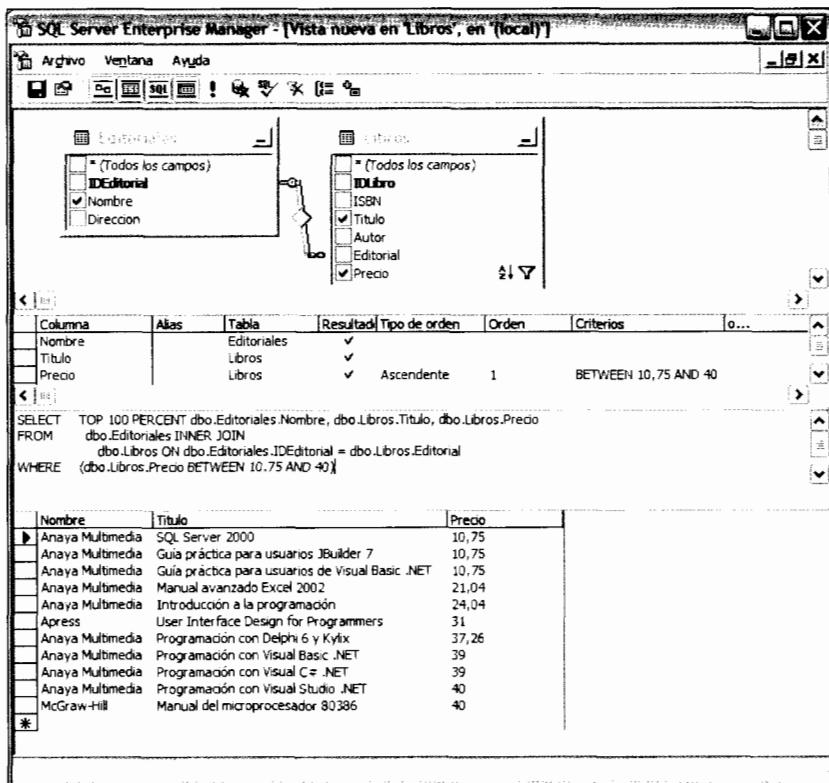


Figura 3.21. Comprobamos el resultado de ejecutar la vista

6. Salga de la ventana de composición y después guarde la vista con el nombre *LibrosEditorial*. Si hace doble clic sobre ella, en el panel derecho del Administrador corporativo, verá aparecer una ventana de propiedades con el código Transact-SQL utilizado para crear la vista.

Puede ejecutar la vista en cualquier momento, desde el propio Administrador corporativo de SQL Server, haciendo clic sobre ella con el botón secundario del ratón y seleccionando la opción Abrir vista>Devolver todas las filas.

Definir procedimientos almacenados

Hasta ahora, todos los elementos creados en la base de datos SQL Server han sido generados mediante asistentes que se han encargado de producir las sentencias SQL adecuadas, enviándolas al servidor para su ejecución.

Al crear un procedimiento almacenado, sin embargo, no tendremos más remedio que escribir el código, ya que un procedimiento almacenado es, en sí, un bloque de código con un nombre.

Seleccione la carpeta Procedimientos almacenados de la base de datos, abra el menú emergente y elija la opción Nuevo procedimiento almacenado. Se encontrará con un cuadro de diálogo en el que aparece la cabecera para el nuevo procedimiento. Debemos facilitar el nombre y, por supuesto, introducir la lógica a ejecutar que, en este caso, será la que puede ver en la figura 3.22. Se trata del procedimiento almacenado propuesto como ejemplo en el capítulo previo, cuyo objetivo es devolver como resultado el número de títulos de la editorial cuyo código se entrega como parámetro o, por defecto, el número total de títulos de todas las editoriales.

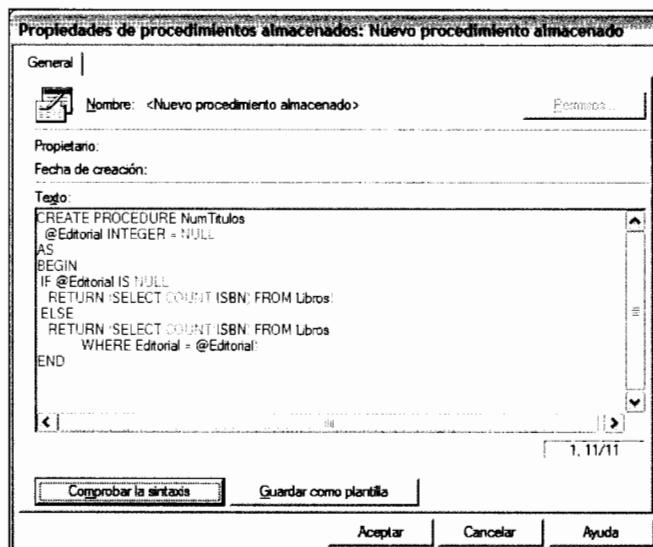


Figura 3.22. Introducimos el código del procedimiento almacenado

Tras escribir el código, pulse el botón **Comprobar la sintaxis** para asegurarse de que no ha introducido ningún error. A continuación puede hacer clic en **Aceptar** para cerrar el cuadro de diálogo y crear efectivamente el procedimiento almacenado. En cualquier momento puede hacer doble clic sobre éste para volver a abrir la ventana y editar el código.

Este procedimiento almacenado devuelve como resultado un valor único, en este caso un número entero. Vamos a crear otro que retorne un conjunto de datos, por ejemplo el nombre de editorial y el número de títulos de cada una de las editoriales que existan. Siga los pasos que se han indicado antes, eligiendo la opción **Nuevo procedimiento almacenado**, e introduzca el código siguiente en la ventana:

```
CREATE PROCEDURE NumTitulosEditorial
AS
BEGIN
    SELECT E.Nombre, COUNT(L.Titulo) NumTitulos
    FROM Editoriales E, Libros L
    WHERE E.IDEditorial = L.Editorial
```

```
GROUP BY E.Nombre  
END
```

Lo que hacemos es ejecutar una consulta, devolviendo el resultado, en la que se agrupan las filas por editorial y se obtiene el número de títulos. El identificador NumTitulos que se ha puesto tras COUNT() será el nombre de esa columna, el número de títulos, en el conjunto de resultados.

Ejecución de procedimientos almacenados

Aunque nuestro objetivo, al crear los procedimientos almacenados del punto anterior, es aprender a utilizarlos desde Visual Basic .NET, a fin de comprobar que el resultado que generan es el que esperamos, y no otro, deberíamos ejecutarlos en este mismo momento. Para hacerlo deberemos recurrir al Analizador de consultas SQL, seleccionando dicha opción del menú Herramientas del Administrador corporativo.

Un procedimiento almacenado se ejecuta mediante la sentencia EXECUTE, tras la que debe entregarse el nombre del procedimiento almacenado y, en caso que sea necesario, los parámetros apropiados. Por ejemplo, para ejecutar el procedimiento almacenado NumTitulosEditorial, el segundo que hemos creado, bastaría con introducir la sentencia EXECUTE NumTitulosEditorial y pulsar el botón Ejecutar consulta o la tecla F5. Es lo que se ha hecho en la figura 3.23, en la que puede ver, en la parte inferior, el conjunto de resultados obtenido.

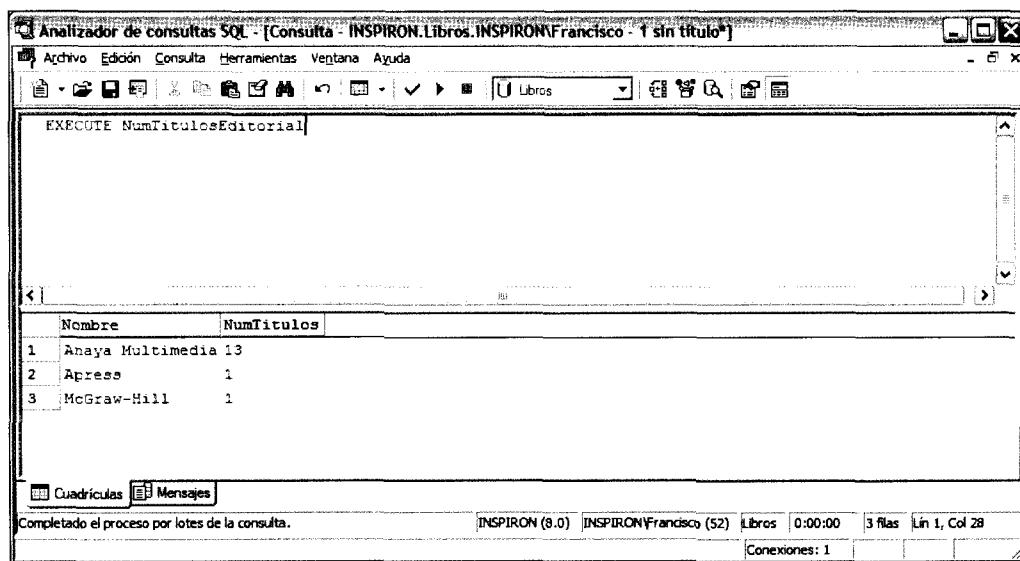


Figura 3.23. El procedimiento almacenado NumTitulosEditorial devuelve un conjunto de resultados obtenido mediante una consulta

Si ejecuta el procedimiento almacenado NumTitulos siguiendo el mismo método, verá que en la parte inferior aparece un mensaje indicando que se ha ejecutado con éxito, pero no se muestra conjunto de resultados alguno. El **Analizador de consultas SQL** recoge los valores devueltos por consultas y los muestra, como se ha visto, en la parte inferior, pero el procedimiento NumTitulos no devuelve conjunto de resultados alguno, tan sólo un número entero.

Para poder comprobar este procedimiento tendremos que recoger el valor devuelto en una variable, previamente declarada, usando a continuación la sentencia PRINT para mostrar el contenido en el panel de mensajes, como se ha hecho en la figura 3.24. Si tras NumTitulos se pone el código de una editorial, obtendrá el número de títulos con que cuenta ésta. En caso contrario se indicará el número total de títulos que hay en la tabla Libros.

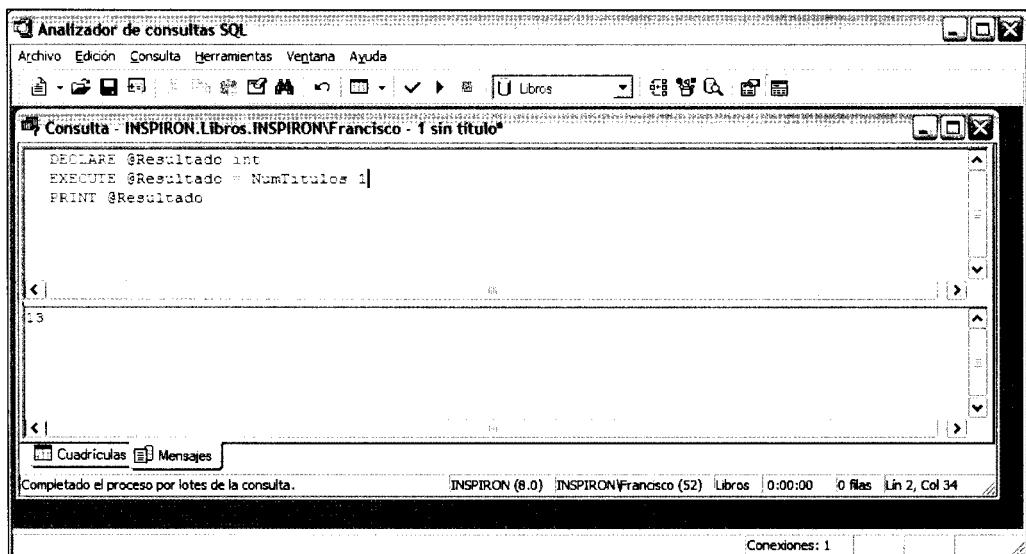


Figura 3.24. Para ejecutar el procedimiento NumTitulos y obtener el resultado tenemos que recurrir a una variable intermedia

Oracle

SQL Server es un RDBMS de Microsoft en continua expansión, especialmente desde la aparición de las versiones 7 y 2000, pero actualmente el número uno de los RDBMS sigue siendo Oracle. No es de extrañar si se tiene en cuenta que es uno de los RDBMS más antiguos, y por tanto con más experiencia, y que, además, está disponible para múltiples plataformas, desde sistemas personales con Windows o Linux hasta grandes sistemas, lo cual le otorga ventaja sobre un producto que, como SQL Server, sólo puede ejecutarse en Windows.

Oracle siempre ha tenido fama de ser un producto robusto, fiable y muy escalable, pero no precisamente fácil de instalar o administrar. Esto, por suerte, va cambiando en las últimas versiones del producto, como la 8i y 9i. Puede obtenerse el RDBMS de Oracle desde <http://otn.oracle.com/software/content.html>, en distintas versiones y ediciones. Tan sólo tiene que registrarse en OTN, es gratuito, proceder a la descarga e instalación siguiendo los pasos indicados.

En los puntos siguientes va a utilizarse *Oracle8i Release 3* (versión 8.1.7) sobre un servidor Windows .NET Enterprise Server, asumiendo que el directorio raíz de Oracle es `OraHome81`. Si en su sistema es uno distinto téngalo en cuenta por los cambios que pudieran ser necesarios.

Creación de la base de datos

Abra la carpeta Oracle - OraHome81 de la lista de programas, en el menú del botón Inicio, y seleccione la opción Database Administration>Database Configuration Assistant. Se pondrá en marcha el Asistente de configuración de bases de datos Oracle, mediante el cual, según se puede ver en la figura 3.25, podemos crear nuevas bases de datos, modificar la configuración de las ya existentes y eliminar bases de datos.

Nota

Cuando se instala Oracle se crea una base de datos de ejemplo, usualmente llamada `oracle`, que podríamos utilizar, pero vamos a crear otra para que conozca el proceso a seguir.

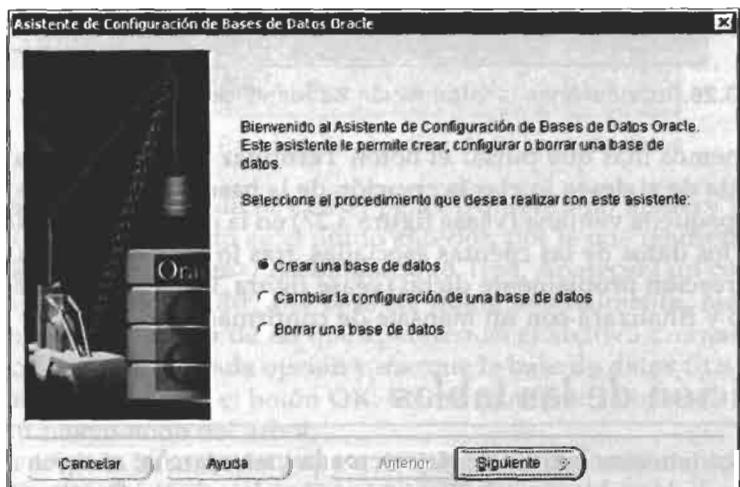


Figura 3.25. Inicio del Asistente de configuración de bases de datos Oracle

Pulse el botón **Siguiente** dejando seleccionada la opción Crear una base de datos. En el paso siguiente podrá elegir entre una base de datos Típica y otra Personalizada. Deje marcada la primera opción y pulse de nuevo el botón **Siguiente**.

Oracle puede crear una base de datos nueva copiando las estructuras básicas de un modelo ya existente, almacenado en el CD-ROM del producto, o bien creando dichas estructuras a partir de cero, lo cual siempre es más lento. Deje seleccionada la opción Copiar los ficheros de base de datos existentes del CD y pulse el botón **Siguiente** una vez más.

En el siguiente cuadro de diálogo debe introducir el nombre global para la base de datos, así como el SID o identificador lógico. El primero se suele componer del nombre de la base de datos más un dominio, formando así un identificador único, mientras que el segundo es un identificador simple. Para este ejemplo, como se ve en la figura 3.26, llamaremos **libros** a la base de datos, como en los casos previos.



Figura 3.26. Introducimos la información de identificación de la nueva base de datos

No tenemos más que pulsar el botón **Terminar** y responder afirmativamente a la pregunta de si desea iniciar la creación de la base de datos. Acto seguido aparecerá una pequeña ventana (véase figura 3.27) en la que se indica el identificador de la base y los datos de las cuentas asociadas, tras lo cual se pone en marcha el proceso de creación propiamente dicho (véase figura 3.28). Éste puede tardar un cierto tiempo y finalizará con un mensaje de confirmación.

Definición de las tablas

Una vez tenemos la base de datos creada y en marcha, el siguiente paso será la definición de las tablas que la formarán. Abra la carpeta Oracle - OraHome81 de nuevo de la lista de programas y seleccione la opción Database Administration>DBA

Studio para iniciar el Oracle DBA Studio. En el cuadro de diálogo que aparece, preguntando si desea iniciarla en modo estándar o conectando con un servidor de administración, deje seleccionada la primera opción y pulse **Intro**. Transcurrido un momento, debería encontrarse con una interfaz como la de la figura 3.29. En el panel izquierdo aparecen las bases de datos disponibles, conteniendo en su interior objetos tales como esquemas, tablas, procedimientos, etc.

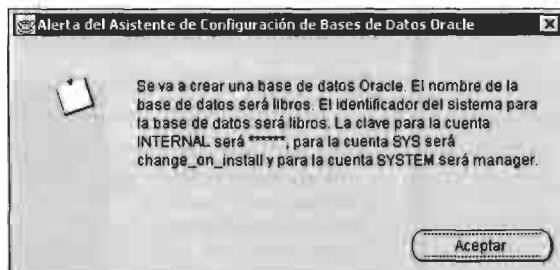


Figura 3.27. Información sobre la base de datos que va a crearse

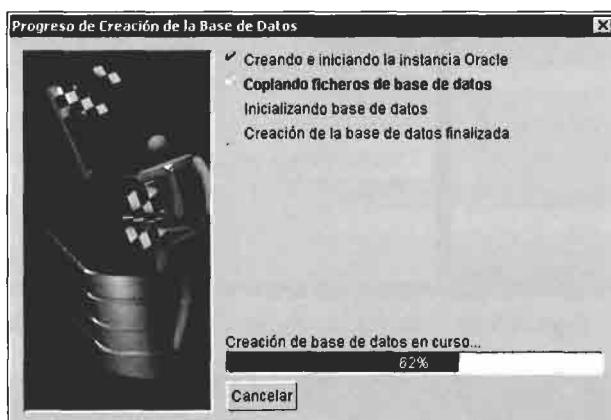


Figura 3.28. Proceso de creación de la base de datos

Como verá, en el árbol que hay en el panel izquierdo no aparece nuestra base de datos, la que hemos creado en el punto anterior, por lo que tendremos que añadirla. Seleccione la opción **File>Add Database to Tree**. Aparecerá un cuadro de diálogo con dos opciones: añadir una base de datos manualmente, facilitando sus datos, o bien seleccionar una de las que aparecen en el archivo `tnsnames.ora` de Oracle. Seleccione esta segunda opción y marque la base de datos **Libros**, haciendo clic a continuación sobre el botón **OK**. En ese momento la base de datos aparecerá como un nuevo nodo del árbol.

Haga clic sobre la base de datos para desplegar su contenido, iniciando sesión como administrador de la base de datos (SYSDBA) utilizando la cuenta **SYSTEM**. Despliegue la rama **Schema** y, dentro de ella, elija el nodo **Table**. Haga clic sobre

él con el botón secundario del ratón y elija la opción **Create Using Wizard** para poner en marcha el asistente de creación de tablas.

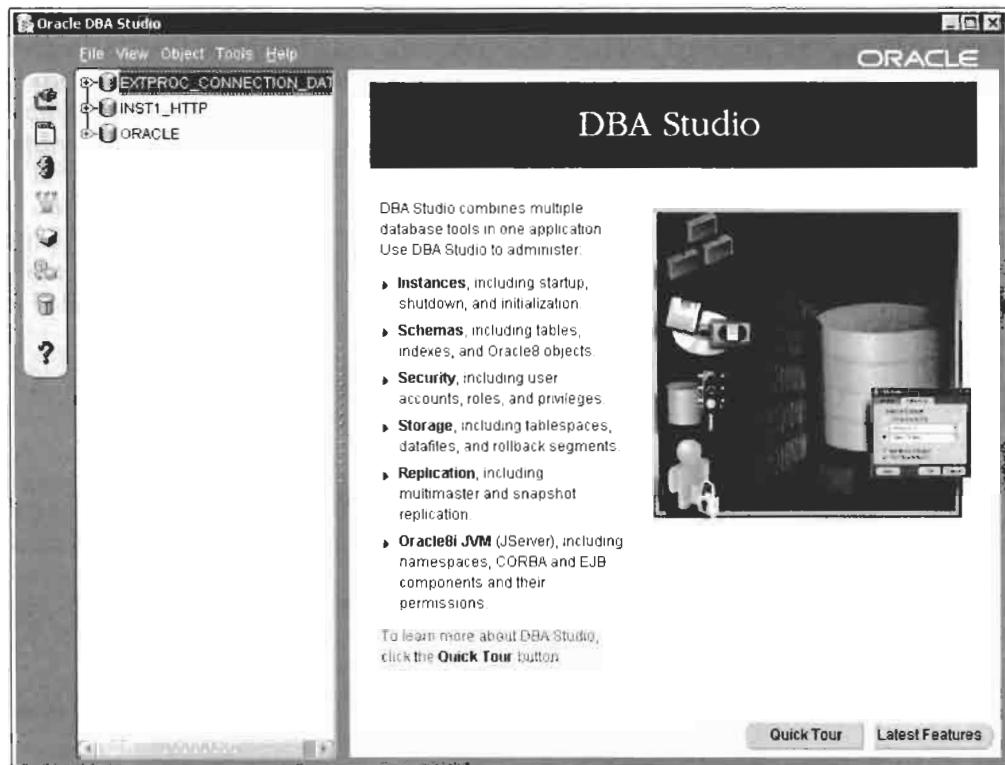


Figura 3.29. Interfaz de usuario del Oracle DBA Studio

Nota

Aunque para este ejemplo, y por inmediatez, vamos a usar la cuenta **SYSTEM** para conectar con la base de datos, lo habitual es que se creen los usuarios que van a tener acceso a ella y se utilice una de esas cuentas para operar.

Se pone en marcha el asistente para creación de tablas, compuesto de nada menos que trece pasos. En los puntos siguientes se detallan uno a uno, asumiéndose que pulsará el botón **Siguiente** a medida que vaya leyéndolos para ir de una ventana a la siguiente.

1. Introduzca el nombre de la tabla, en este caso **EDITORIALES**, y seleccione el esquema del que formará parte. Por defecto aparece el esquema **SYSTEM** porque es la cuenta con la que hemos iniciado sesión, pero puede desplegar

la lista **Which Schema do you want the table to be part of?** y elegir cualquiera de las creadas como ejemplo en la base de datos, por ejemplo el conocido esquema SCOTT.

2. El segundo paso consiste en la definición de las columnas que tendrá la tabla. A la izquierda aparece una lista, inicialmente vacía, con las columnas, mientras que a la derecha se reflejan los atributos de la columna elegida en cada momento. Introduzca los datos de las columnas siguientes, pulsando el botón **Add** para ir añadiéndolas a la lista.
 - **IEDITORIAL:** Asígnele el tipo NUMBER estableciendo 32 como tamaño. En Oracle no hay un tipo similar al autoincrementado de SQL Server y Microsoft Access, aunque podría conseguir el mismo efecto por medio de un desencadenador y una secuencia.
 - **NOMBRE** y **DIRECCION:** Selección el tipo VARCHAR2 introduciendo 50 en el apartado **Size**.

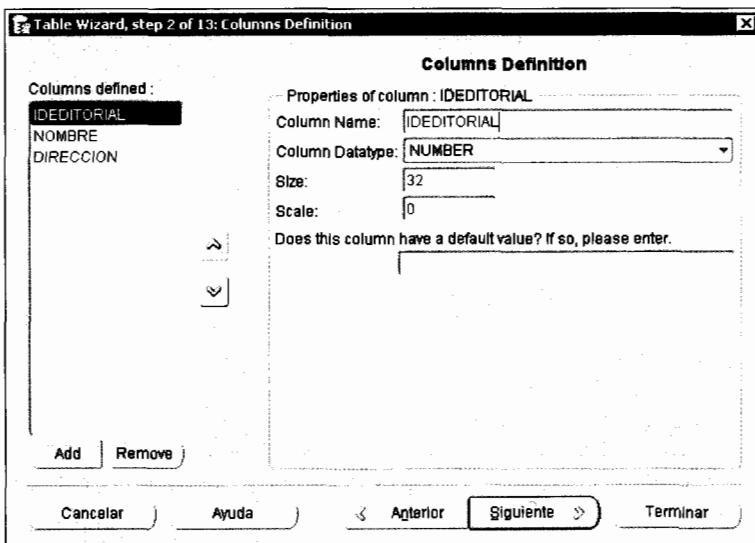


Figura 3.30. Definición de las columnas de la tabla EDITORIALES

3. En la ventana siguiente podemos establecer una clave primaria para la tabla. Active la opción **Yes, I want to create a primary key** y haga clic en la columna **Order** de **IEDITORIAL**, convirtiéndola en la única participante de la clave principal.
4. A continuación encontrará las opciones necesarias para definir algunas restricciones, concretamente las de permitir o no valores nulos y duplicación de valores. En la columna de la izquierda seleccione la columna **IEDITORIAL** y, a continuación, marque las opciones **No, it cannot be Null** y **Yes, it must be**

Unique a la derecha. A continuación podría establecer restricciones similares para las otras columnas, aunque en este caso las dejaremos con sus valores por defecto.

5. El paso 5 de 13 permite definir claves externas, algo que en el caso de la tabla EDITORIALES no es aplicable. Por lo tanto, pulsamos directamente el botón **Siguiente**.
6. Otro paso que nos saltaremos será el sexto, ya que no vamos a aplicar restricciones adicionales de comprobación a ninguna de las columnas de esta tabla. Como se aprecia en la figura 3.31, no tenemos más que elegir una columna, a la izquierda, e introducir la restricción a la derecha tras activar la opción **Yes, the column has a Check Condition**. En la parte inferior aparecen algunos ejemplos.

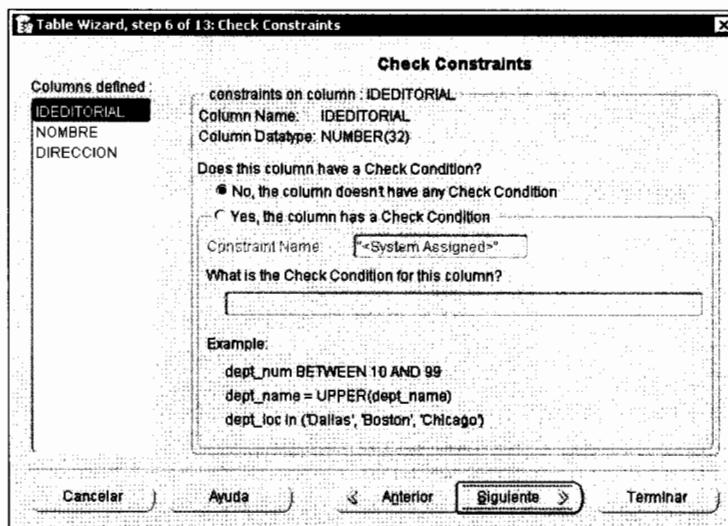


Figura 3.31. Restricciones de comprobación para las columnas

7. La ventana de este paso se emplea para introducir opciones específicas sobre almacenamiento. Dejaremos la configuración por defecto y pasaremos a la siguiente.

Dado que hemos dejado en algunos apartados los valores por defecto, el asistente pasa directamente al paso 13 y nos muestra la sentencia SQL que se utilizará para crear la tabla (véase figura 3.32). Basta con pulsar el botón **Terminar** para proceder a la creación de la tabla.

Utilizando el mismo asistente, vamos ahora a crear la tabla LIBROS. Ésta contará con las mismas columnas que ya definimos previamente en SQL Server o Access, pero con los tipos de datos de Oracle. La columna IDLIBRO, por ejemplo, será de tipo NUMBER, al igual que EDITORIAL y PRECIO; ISBN será de tipo CHAR con una

longitud de 13 caracteres y las demás columnas serán de tipo VARCHAR y 50 caracteres como límite.

Defina la columna IDLIBRO como clave primaria y aplíquele las restricciones necesarias para evitar que se introduzcan valores duplicados o que quede con un valor nulo. También puede aplicar las mismas restricciones a la columna ISBN.

En el paso cinco deberá establecer la relación entre la columna EDITORIAL de esta tabla e IDEITORIAL de la tabla EDITORIALES, tal y como se muestra en la figura 3.33. Así se asegura la integridad referencial entre ambas.

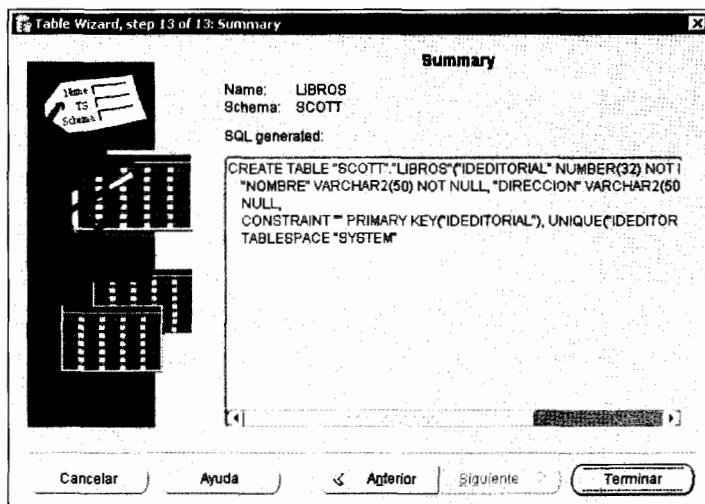


Figura 3.32. Resumen de la creación de la tabla

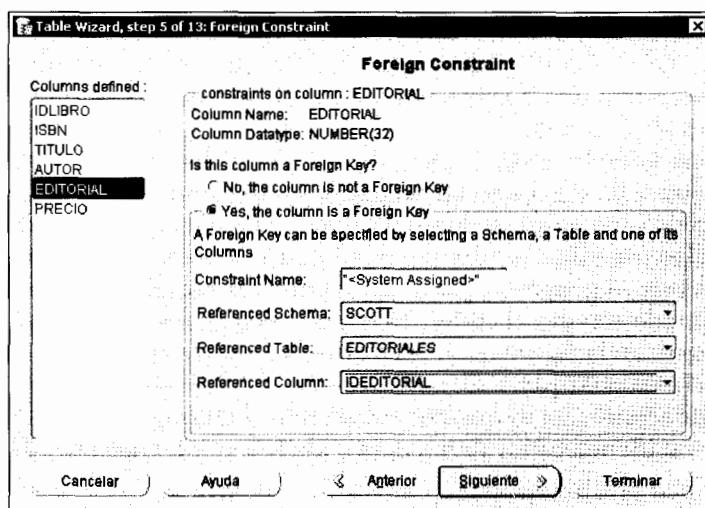


Figura 3.33. Definimos las restricciones de integridad referencial

Ahora puede pulsar directamente el botón **Terminar** para ir al último paso del asistente, revisar la sentencia SQL y, pulsando otra vez el botón **Terminar**, crear la tabla. En este momento ya tenemos definidas las estructuras que necesitamos.

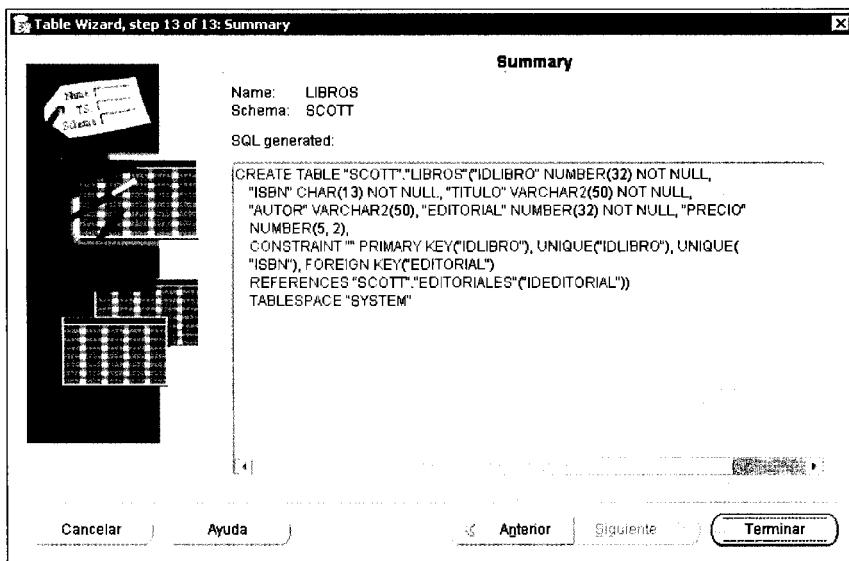


Figura 3.34. Resumen de la creación de la tabla LIBROS

Introducción de datos

El mecanismo de edición de datos de Oracle se asemeja, en funcionamiento, al visto anteriormente para SQL Server, si bien resulta algo más rudimentario ya que los cambios no van confirmándose a medida que se pasa de una fila a otra, sino que van generando una o más sentencias SQL que no se ejecutan hasta que lo confirmamos explícitamente.

Seleccione la tabla EDITORIALES en el panel derecho del Oracle DBA Studio, haga clic sobre ella con el botón secundario del ratón y elija la opción **Table Data Editor**. Inicialmente aparece una rejilla vacía. Puede ir introduciendo los datos de las editoriales, incluido el identificador puesto que no hemos preparado ningún mecanismo que lo genere automáticamente. Utilizando los botones que tiene a la izquierda puede acceder al código SQL que se va generando a medida que introduce datos, como en la figura 3.35. Dichas sentencias se ejecutarán en el momento en que pulse el botón **Apply**.

A continuación haga lo mismo con la tabla LIBROS, teniendo en cuenta que debe facilitar un identificador único para cada fila y, además, introducir el código de cada editorial definida en la tabla anterior. Cualquier error de restricción provocará un mensaje de error similar al de la figura 3.36, en el que se comunica que se ha dado un código de editorial不存在.

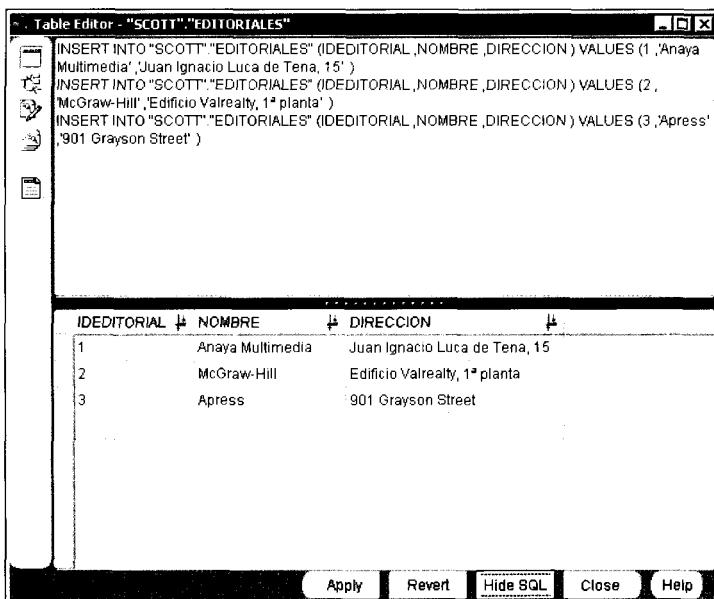


Figura 3.35. Introducimos una serie de filas en la tabla EDITORIALES

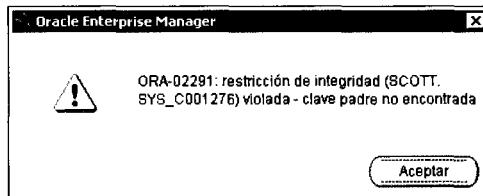


Figura 3.36. Error provocado al violarse una de las restricciones

Definición de una vista

Como hicieron en la base de datos SQL Server, vamos a definir una vista simple que nos permita obtener una lista de todas las editoriales junto con los títulos que le corresponden. Seleccione la carpeta View, en el panel izquierdo del Oracle DBA Studio, y seleccione la opción Create Using Wizard. En este caso el asistente se compone de los siguientes cinco pasos:

1. En la primera ventana introduzca el nombre de la vista que vamos a crear, LIBROEDITORIAL, y seleccione el esquema SCOTT para alojarla. Pulse el botón **Siguiente**.
2. La segunda ventana mostrará en el panel izquierdo una lista de las tablas disponibles, junto con sus columnas, a fin de que seleccionemos aquellas que deseamos incluir en la vista. En la figura 3.37 puede ver cómo se ha incluido

el nombre de la editorial, el título y el precio de cada libro. Pulse entonces el botón **Siguiente**.

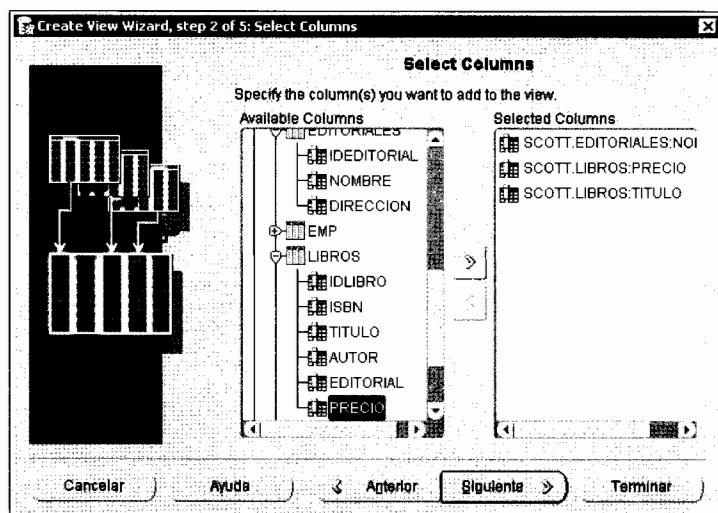


Figura 3.37. Seleccionamos las columnas a incluir en la vista

3. El paso siguiente nos permite introducir un nombre para cada columna de la vista, en sustitución de los nombres de las columnas originales. También puede alterar el orden de las columnas arrastrando y soltando.
4. La penúltima operación será definir las condiciones de selección de los datos. Arrastre desde el panel de la izquierda la columna IDEDITORIAL hasta el de la derecha. Haga clic en éste con el botón secundario del ratón y cambie a la vista SQL para poder completar la condición SCOTT.EDITORIALES.IDEDITORIAL = SCOTT.LIBROS.EDITORIAL. Pulse el botón **Siguiente**.
5. Como al crear las tablas, el último paso es simplemente de confirmación. En él podemos ver la sentencia SQL con la que va a generarse la vista. No tenemos más que pulsar el botón **Terminar** para crearla.

Para ejecutar la vista, y ver el resultado, selecciónela, en el panel derecho, haga clic sobre ella con el botón secundario del ratón y elija la opción **View Contents**. Deberá encontrarse con una ventana como la de la figura 3.39. Lógicamente, puede modificarla para añadir criterios adicionales de selección, en la cláusula WHERE, con el objetivo de filtrar las filas y obtener sólo las que cumplan ciertas condiciones.

Definir funciones y procedimientos almacenados

Como vimos en el capítulo anterior, en Oracle podemos crear procedimientos almacenados, que pueden tomar parámetros pero no devolverlos, y funciones, que

sí pueden devolver valores como resultado. Comencemos definiendo la función propuesta como ejemplo al final del capítulo previo.

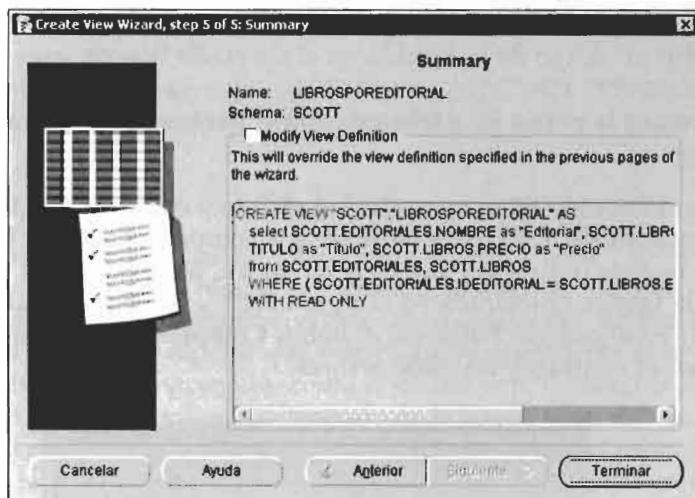


Figura 3.38. Resumen de la creación de la vista

Content Viewer - "SCOTT"."LIBROSPOREDITORIAL"			
Editorial	Título	Precio	
Apress	User Interface Design for Programmers	31	
Anaya Multimedia	SQL Server 2000	10.75	
Anaya Multimedia	Guía práctica para usuarios JBuilder 7	10.75	
Anaya Multimedia	Programación con Visual C# .NET	39	
Anaya Multimedia	Programación con Visual Studio .NET	40	
Anaya Multimedia	Programación con Visual Basic .NET	39	
Anaya Multimedia	Guía práctica para usuarios de Visual Basic .NET	10.75	
Anaya Multimedia	Guía práctica para usuarios de Visual Studio .NET	10.52	
Anaya Multimedia	Programación con Delphi 6 y Kylix	37.26	
Anaya Multimedia	Guía práctica para usuarios de Delphi 6	10.52	
Anaya Multimedia	Manual avanzado Excel 2002	21.04	
Anaya Multimedia	Guía práctica para usuarios de Excel 2002	10.52	
Anaya Multimedia	Guía práctica para usuarios de Kylix	10.52	
Anaya Multimedia	Introducción a la programación	24.04	
McGraw-Hill	Manual del microprocesador 80386	40	

Figura 3.39. Resultado de ejecutar la vista

Localice la carpeta Functions y haga clic sobre ella con el botón secundario del ratón, seleccionando la opción Create. Aparecerá un cuadro de diálogo en el que tiene que dar tres pasos:

1. Introducir en la parte superior el nombre de la función, NUMTITULOS en este caso.
2. Seleccionar el esquema donde se alojará la función, en este caso SCOTT.
3. Escribir el código de la función en el apartado Source, exceptuando la cabecera CREATE FUNCTION NUMTITULOS que ya se da por asumida. Es decir, dejaremos la pareja de paréntesis con el parámetro de entrada y el resto del código.

Al pulsar el botón **OK** se compilará el código y creará la función. Seleccione la carpeta Functions>SCOTT para localizarla y compruebe que en la columna de la derecha aparece la indicación **Valid**. De no ser así, haga doble clic sobre la función, para abrir la ventana mostrada en la figura 3.40, y corrija cualquier error que pudiera existir en el código. Pulsando el botón **Compile** podrá saber si la función es válida o, por el contrario, contiene errores.

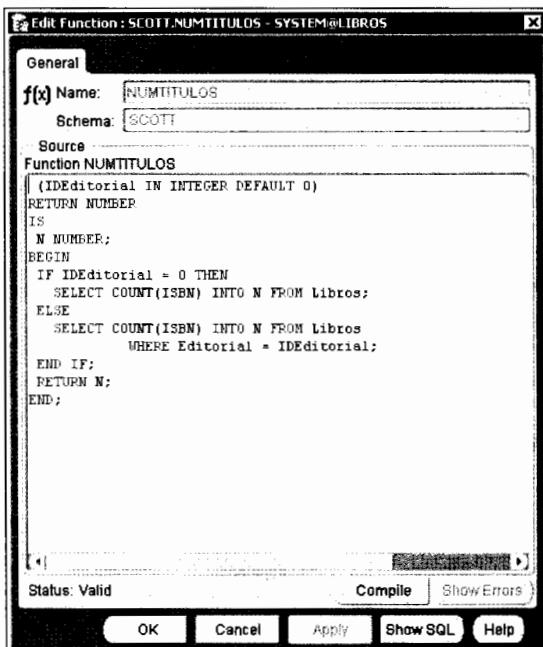


Figura 3.40. Creamos la función NUMTITULOS

Esta función devuelve un resultado único. Si queremos crear un procedimiento almacenado que haga lo mismo, lo cual es perfectamente posible, tendremos que utilizar un parámetro de salida, de tipo OUT, asignándole el valor a devolver en lugar de utilizar la sentencia RETURN.

Si lo que pretendemos devolver no es un valor único, como en este caso, sino un conjunto de datos resultante de una consulta, como hicieramos en el segundo pro-

cedimiento almacenado escrito con Transact-SQL, el tema se complica algo más, ya que en PL/SQL no podemos, sin más, ejecutar algo así:

```
CREATE PROCEDURE NUMTITULOSEDITORIAL
IS
BEGIN
    SELECT E.NOMBRE, COUNT(L.TITULO) NUMTITULOS
    FROM SCOTT.EDITORIALES E, SCOTT.LIBROS L
    WHERE E.IDEDITORIAL = L.EDITORIAL
    GROUP BY E.NOMBRE;
END;
```

A pesar de que la consulta está expresada correctamente, podemos ejecutarla en SQL*Plus y obtenemos el resultado esperado, al intentar introducirla en un procedimiento almacenado obtenemos un error de compilación. En PL/SQL no puede ejecutarse una consulta y esperar que vuelva automáticamente al proceso que ejecute el procedimiento almacenado, como sí ocurre con Transact-SQL.

La solución pasa por recoger el resultado de la consulta en una variable, una referencia a un cursor, y esperar que el proceso que invoque al procedimiento haga con él lo que le interese. Para esto es necesario dar dos pasos: crear un paquete PL/SQL en el que se defina un nuevo tipo de dato asociado a REF CURSOR, por una parte, y codificar el procedimiento almacenado de tal manera que emplee ese tipo para devolver el resultado.

Localice la carpeta **Package** en el panel izquierdo del Oracle DBA Studio, abra el menú emergente y seleccione la opción **Create**. Introduzca LIBROSPKG como nombre, elija SCOTT como esquema de destino e introduzca, como se aprecia en la figura 3.41, la definición del nuevo tipo, al que llamaremos **rs**. El paquete contendrá sólo esto, la definición del tipo.

A continuación definiremos el procedimiento almacenado, seleccionando la opción **Create** de la carpeta **Procedure**. Introducimos el nombre del procedimiento, NUMTITULOSEDITORIAL, elegimos SCOTT como esquema y escribimos el código siguiente:

```
(Resultado OUT LIBROSPKG.RS)
IS
BEGIN
    OPEN Resultado FOR
    SELECT E.NOMBRE, COUNT(L.TITULO) NUMTITULOS
    FROM SCOTT.EDITORIALES E, SCOTT.LIBROS L
    WHERE E.IDEDITORIAL = L.EDITORIAL
    GROUP BY E.NOMBRE;
END;
```

Observe cómo se indica que existe un parámetro de salida, cuyo tipo es el definido previamente en el paquete LIBROSPKG. Fíjese también en cómo se inserta el resultado de la consulta SQL en la variable **Resultado** mediante la sentencia **OPEN**.

Al cerrar la ventana habrá finalizado la creación del procedimiento almacenado que, en el punto siguiente, verá cómo utilizar.

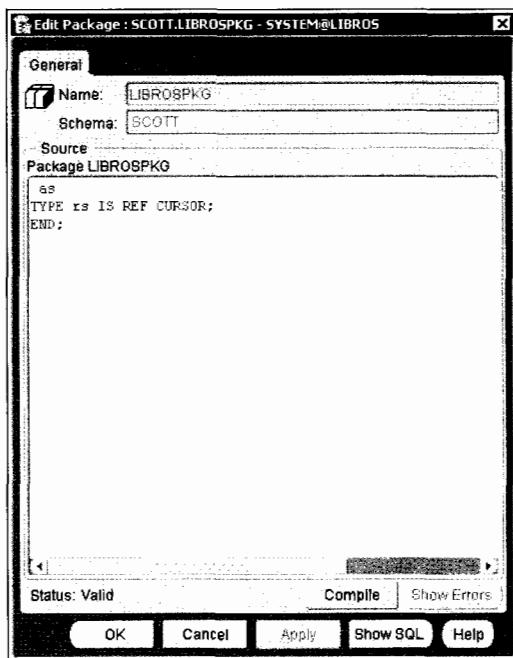


Figura 3.41. Definición del paquete con el tipo de dato rs

Ejecución de funciones y procedimientos

En este momento contamos con una función y un procedimiento almacenado que, en principio, aparecen como válidos, ya que no han generado errores al guardarse en la base de datos. No sabemos, sin embargo, cómo funcionan ni si el resultado obtenido es el que se espera, para ello tendremos que ejecutarlos. Con este fin tendremos que recurrir a la herramienta **SQL*Plus Worksheet**, seleccionándola del grupo **Database Administration** de la lista de programas de Oracle. Deberá introducir el nombre de usuario, clave de acceso y nombre del servicio con el que va a conectar, que en este caso será **LIBROS**.

Comencemos ejecutando la función **NUMTITULOS**, para lo cual tendremos que declarar una variable, guardar el resultado y después mostrarlo en el panel inferior. Esto es lo que hemos hecho en la figura 3.42, en la que hemos usado la palabra clave **VARIABLE** para declarar la variable, en vez de abrir una sección **DECLARE** seguida de un bloque **BEGIN/END**. El resultado, como se aprecia en la parte inferior, es el número de títulos devuelto por la función.

Para ejecutar el procedimiento almacenado **NUMTITULOSEditorial** hemos de tener en cuenta que, a diferencia de lo que ocurría en Transact-SQL, no bastará con llamarlo mediante la sentencia **EXECUTE**. Es preciso pasar un parámetro, concretamente una variable capaz de guardar una referencia a un cursor. Por lo tanto declararemos una variable de tipo **REFCURSOR**, entregándola como parámetro al

procedimiento almacenado. Después, no tenemos más que imprimir dicha variable para poder ver el resultado, como se aprecia en la figura 3.43.

En un capítulo posterior veremos cómo acceder a estos datos desde una aplicación escrita con Visual Basic, utilizando los servicios de ADO.NET.

The screenshot shows the Oracle SQL*Plus Worksheet interface. In the top-left menu bar, it says "SQL*Plus Worksheet". Below the menu is a toolbar with icons for file operations. The main workspace contains the following PL/SQL code:

```
CONNECT SYSTEM/********@libros AS SYSDBA
VARIABLE N NUMBER;
EXECUTE :N := SCOTT.NUMTITULOS
PRINT :N
```

When run, the output window displays:

```
Procedimiento PL/SQL terminado correctamente.

N
-----
1

Conectado.

Procedimiento PL/SQL terminado correctamente.

N
-----
15
```

Figura 3.42. Ejecución de la función NUMTITULOS en SQL*Plus Worksheet

Nota

En realidad, el procedimiento almacenado que hemos creado como ejemplo es simplemente eso, un ejemplo, ya que en la práctica, puesto que no se efectúa ningún proceso especial de la información, resultaría mucho más fácil crear una vista que generase el mismo resultado.

InterBase

Oracle es un RDBMS disponible en múltiples sistemas operativos, según se apuntaba anteriormente, pero no por ello será siempre el producto más adecuado,

especialmente si nuestras necesidades de almacenamiento y gestión de datos no son las de una gran empresa. De ser así, podemos evitar gran parte de la complejidad de Oracle optando por un producto de menor nivel. Existen muchos otros RDBMS multiplataforma, para varios sistemas operativos, aparte de Oracle, por ejemplo el popular MySQL o InterBase.

The screenshot shows the Oracle SQL*Plus Worksheet interface. The command window contains the following PL/SQL code:

```
SQL*Plus Worksheet
File Edit Worksheet Help
CONNECT SYSTEM/*****@libros AS SYSDBA
VARIABLE Resultado REFCURSOR;
EXECUTE SCOTT.NUMTITULOSEDITORIAL(:Resultado);
PRINT :Resultado
```

The output window displays the results of the executed procedure:

NOMBRE	NUMTITULOS
Anaya Multimedia	13
Apress	1
McGraw-Hill	1

Conectado.
Procedimiento PL/SQL terminado correctamente.

NOMBRE	NUMTITULOS
Anaya Multimedia	13
Apress	1
McGraw-Hill	1

Figura 3.43. Recuperamos la referencia al cursor devuelto por el procedimiento almacenado

InterBase es un producto adquirido por Borland hace años a la empresa Ashton-Tate, la misma que creó dBase. Tras diversos vaivenes, InterBase actualmente es el RDBMS de Borland y es ampliamente utilizado por aquellos que emplean las herramientas de desarrollo de dicha empresa, entre ellas el conocido Delphi, C++ Builder y Kylix.

A pesar de sus posibilidades, InterBase es un RDBMS ligero, de pequeño tamaño, y fácil de instalar, distribuir y configurar. Como todos los RDBMS, cuenta con una parte que actúa como servidor y otra que es el software a instalar en los clientes para comunicarse con ese servidor. Ambos pueden instalarse, indistintamente, en Windows, Linux y Solares.

Actualmente existen dos versiones de InterBase independientes. Una tiene una licencia de software libre, sin coste pero sin soporte alguno ni garantía por parte

de Borland, y la otra es el producto comercial que vende esta empresa. La primera es InterBase 6.0 y la segunda InterBase 6.5. Puede obtener la primera de <http://mers.com>, tan sólo tiene que pulsar sobre el sistema operativo que desea, descargarla e instalarla. Si desea probar InterBase 6.5, puede obtener una edición *trial* de http://www.borland.com/products/downloads/download_interbase.html.

En los puntos siguientes se utilizará la versión 6.5 instalada sobre un sistema Windows XP Professional. Ésta incorpora una herramienta, llamada **IBConsole**, que simplifica la mayoría de las operaciones que, en versiones previas a la 6.0, tenían que efectuarse necesariamente mediante sentencias SQL. Abra el grupo de programas **InterBase** y seleccione la opción **IBConsole**, deberá encontrarse con una interfaz como la de la figura 3.44. Ésta será el punto de partida para las operaciones descritas en los puntos siguientes.

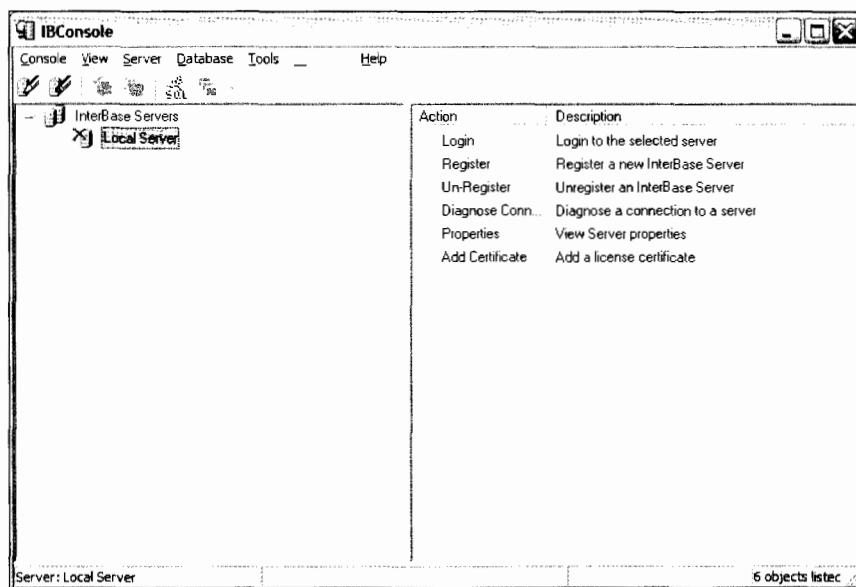


Figura 3.44. Aspecto de la herramienta IBConsole recién puesta en marcha

Creación de la base de datos

Habitualmente **IBConsole** se utiliza en el mismo ordenador donde se ha instalado InterBase y, al iniciarse, ese servidor aparece ya como **Local Server**. De no ser así, pulse el primer botón que aparece en la parte superior, el que está más a la izquierda, para registrar el servidor local o un servidor remoto en caso de que esté usando **IBConsole** desde una máquina diferente.

Teniendo seleccionado el servidor, en el panel izquierdo, haga doble clic en el derecho sobre la opción **Login** para iniciar sesión. Aparecerá un pequeño cuadro

de diálogo en el que debe introducir el nombre de usuario y la clave. Si no ha creado una cuenta específica para su trabajo, puede utilizar la cuenta SYSDBA con la clave por defecto masterkey.

Una vez se haya iniciado sesión, podrá desplegar el contenido de Local Server y ver que hay diferentes carpetas: Databases, Users, Certificates, etc. La carpeta **Databases** está inicialmente vacía ya que no hemos creado base de datos alguna. Haga clic con el botón secundario del ratón sobre dicha carpeta y elija la opción **Create database**. Aparecerá un cuadro de diálogo en el que debe introducir tres datos:

- El camino y nombre del archivo, u archivos, en el que residirá la base de datos. En nuestro caso será un solo archivo llamado **Libros.gdb**.
- El tamaño inicial del archivo, expresado en páginas. Vamos a introducir el valor 250.
- El alias o nombre con el que se conocerá la base de datos internamente, en **IBConsole**.

En la figura 3.45 puede ver los valores introducidos. Al pulsar el botón **OK** se procederá a crear la base de datos y, de inmediato, ésta se abrirá en **IBConsole** mostrando (véase figura 3.46) a la izquierda las diferentes categorías de elementos, mientras que a la derecha se enumeran las acciones posibles en ese instante.

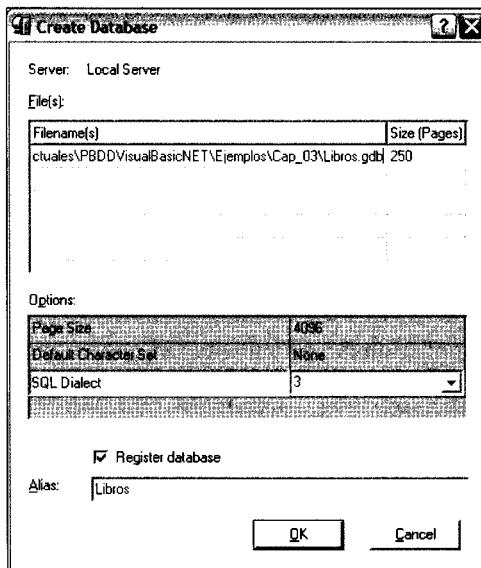


Figura 3.45. Parámetros de creación de la base de datos

En este momento ya tenemos a nuestra disposición una base de datos vacía, por lo que podemos proceder con la definición de las tablas.

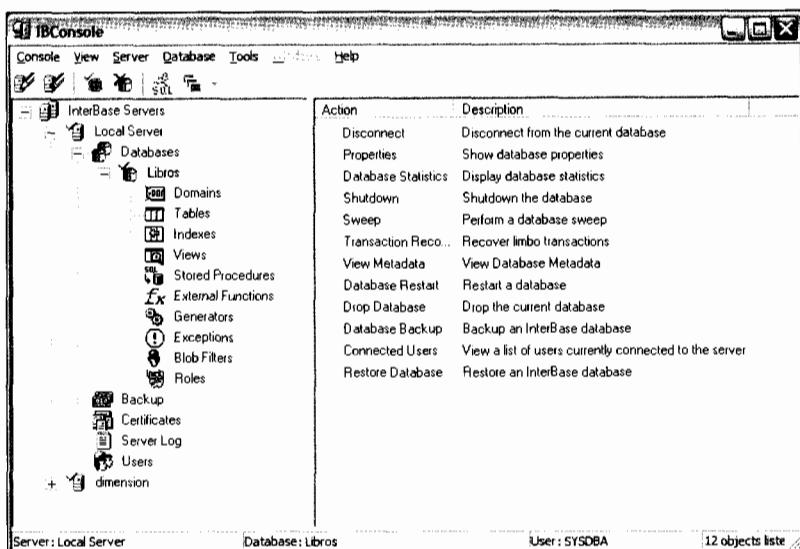


Figura 3.46. Aspecto de IBConsole tras crear la base de datos Libros

Nota

Encontrará el archivo `Libros.gdb` en la carpeta de ejemplos correspondiente a este capítulo. Para utilizarla no tiene más que copiarla a su sistema y usar la opción Register en lugar de Create Database en IBConsole.

Definición de las tablas

Para definir la estructura de las tablas tendremos que recurrir a la herramienta Interactive SQL de Interbase, que puede abrir pulsando el botón que tiene ese nombre en IBConsole. No hay disponibles asistentes, como en SQL Server u Oracle, por lo que tendremos que escribir las sentencias DDL y ejecutarlas.

Introducimos en Interactive SQL la sentencia CREATE TABLE de la tabla `Editoriales`, que puede observar en la figura 3.47, y pulsamos el botón de ejecución. Si todo va bien, la tabla aparecerá en IBConsole y el código SQL desaparecerá. A continuación escribimos la sentencia siguiente para crear la tabla `Libros`. Observa cómo se establece la relación entre ambas tablas con FOREIGN KEY.

```
CREATE TABLE Libros
(
  IDLibro INTEGER NOT NULL PRIMARY KEY,
  ISBN CHAR(13) NOT NULL UNIQUE,
  Titulo VARCHAR(50),
  Autor VARCHAR(50),
  Editorial INTEGER,
```

```
Precio DECIMAL(6,2),
FOREIGN KEY (Editorial) REFERENCES
Editoriales(IDEditorial)
)
```

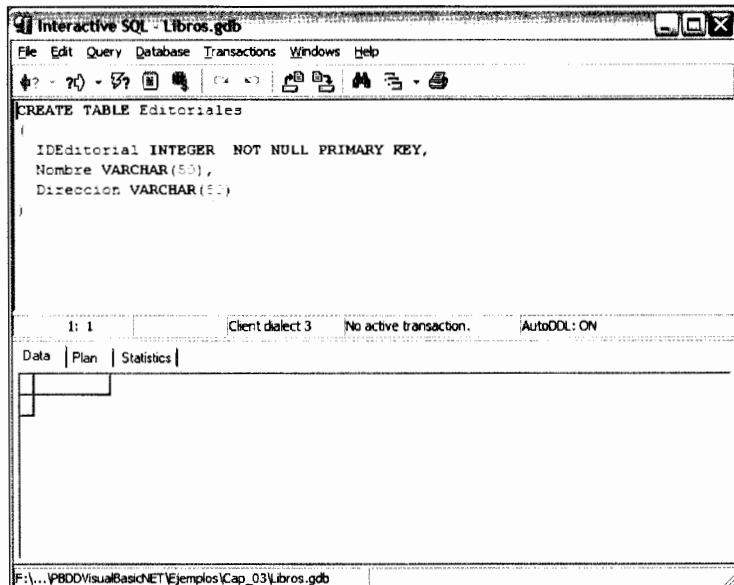


Figura 3.47. Creación de la tabla `Editoriales` desde Interactive SQL

Tras ejecutar las dos sentencias puede cerrar la ventana de Interactive SQL. Verá que las tablas aparecen en la carpeta **Tables** de IBConsole. Abriendo su menú emergente podrá efectuar diversas tareas, como extraer el código DDL o acceder a su ventana de propiedades.

Nota

En InterBase no existe un tipo de dato con incremento automático. Al igual que en Oracle, no obstante, existe la posibilidad de crear una secuencia, llamada *generador* en el caso de InterBase, y utilizarla en un desencadenador para dar valores únicos a una columna. No obstante, en este ejemplo nos limitaremos a usar el tipo `INTEGER` dejando que sea el usuario el que introduzca el código de las editoriales y libros.

Introducción de datos

Volvemos a IBConsole para proceder a la introducción de datos en nuestras tablas. Haga doble clic sobre la tabla `Editoriales`, para abrir la ventana de propie-

dades, y luego seleccione la página **Data**. En ella puede, como se aprecia en la figura 3.48, introducir los datos de las editoriales, sin olvidar asignar a cada una de ellas un código único.

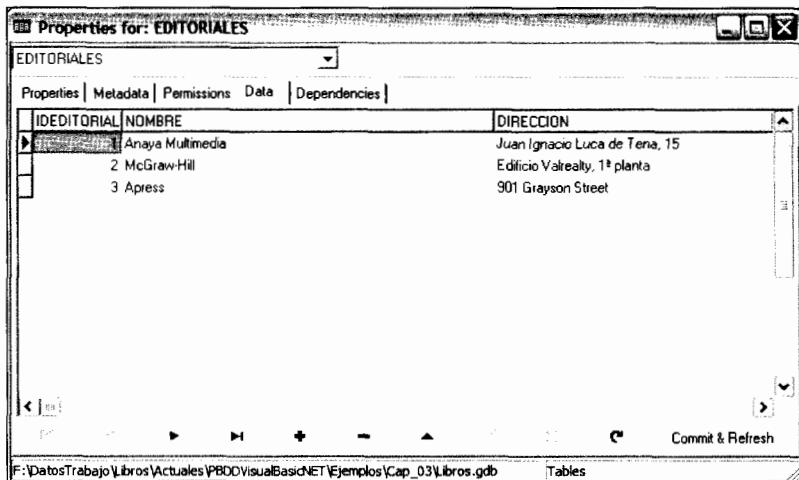


Figura 3.48. Desde IBConsole podemos acceder al contenido de las tablas y editarlo

Siguiendo el mismo procedimiento, proceda a insertar algunas entradas en la tabla *Libros*. Recuerde que puede utilizar la base de datos del CD-ROM que acompaña al libro. Si introduce un código de editorial inexistente obtendrá un mensaje de error como el de la figura 3.49. Algo similar ocurrirá si da a dos libros el mismo identificador, infringiendo otra de las restricciones.

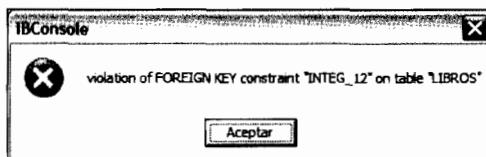


Figura 3.49. Error generado al introducir un código de editorial inexistente

Definición de una vista

Al igual que para definir las tablas, tendremos que recurrir a la herramienta **Interactive SQL** para crear la vista *LibrosEditorial*. Introduzca el código siguiente y pulse el botón de ejecución.

```
CREATE VIEW LibrosEditorial AS
SELECT E.Nombre, L.Titulo, L.Precio
FROM Editoriales E, Libros L
WHERE E.IDEditorial = L.Editorial
```

La vista es exactamente igual a la definida en los puntos previos usando Transact-SQL y PL/SQL. Como ya sabe, puede añadir condiciones adicionales, si así lo desea, para filtrar los datos y obtener sólo parte de las filas.

Para ver el resultado vuelva a IBConsole, haga doble clic sobre la vista, en la carpeta Views, y abra la página Data para ver el resultado. Será similar al mostrado en la figura 3.50.

NOMBRE	TITULO	PRECIO
Apress	User Interface Design for Programmers	31
Anaya Multimedia	SQL Server 2000	10,75
Anaya Multimedia	Guía práctica para usuarios JBuilder 7	10,75
Anaya Multimedia	Programación con Visual C# .NET	39
Anaya Multimedia	Programación con Visual Studio .NET	40
Anaya Multimedia	Programación con Visual Basic .NET	39
Anaya Multimedia	Guía práctica para usuarios de Visual Basic .NET	10,75
Anaya Multimedia	Guía práctica para usuarios de Visual Studio .NET	10,52
Anaya Multimedia	Programación con Delphi 6 y Kylix	37,26
Anaya Multimedia	Guía práctica para usuarios de Delphi 6	10,52
Anaya Multimedia	Manual avanzado Excel 2002	21,04
Anaya Multimedia	Guía práctica para usuarios de Excel 2002	10,52
Anaya Multimedia	Guía práctica para usuarios de Kylix	10,52
Anaya Multimedia	Introducción a la programación	24,04
McGraw-Hill	Manual del microprocesador 80386	40

Figura 3.50. Ejecución de la vista desde IBConsole

Nota

En lugar de volver a IBConsole, también puede ejecutar la vista desde Interactive SQL con una sentencia como `SELECT * FROM LibrosEditorial`. El conjunto de datos resultante aparecerá en la parte inferior de la ventana.

Definir procedimientos almacenados

La sintaxis para definir procedimientos almacenados en InterBase es ligeramente diferente a la de Transact-SQL o PL/SQL, como sería de esperar al tratarse de un producto RDBMS de otro fabricante.

Los procedimientos almacenados de InterBase pueden devolver valores, en realidad varios si bien tan sólo uno de ellos, con el nombre RETVAL, actúa como verdadero valor de retorno. Los parámetros de entrada se declaran entre paréntesis, detrás del nombre del procedimiento almacenado, y no pueden tener valores por defecto.

Cada una de las sentencias de un procedimiento almacenado InterBase debe finalizar con un punto y coma. Esto es también cierto en Oracle. El problema es que la herramienta Interactive SQL identifica ese mismo carácter como final de sentencia, de tal forma que, al encontrarlo, da por terminada la ejecución y, consecuentemente, el procedimiento almacenado no se crea apropiadamente. Para solucionar este problema, es necesario utilizar la orden SET TERM a fin de cambiar el finalizador ; por otro carácter, restituyéndolo al terminar.

Comencemos definiendo el procedimiento NumTitulos que, como en los casos anteriores, debe facilitar el número total de títulos o los de una cierta editorial en caso de facilitarse su código. Introduzca el código siguiente en la herramienta Interactive SQL y ejecútelo.

```
SET TERM # ;
CREATE PROCEDURE NumTitulos(CodEditorial INTEGER)
RETURNS (RETNAL INTEGER)
AS
BEGIN
  IF (CodEditorial = 0) THEN
    SELECT COUNT(ISBN) FROM Libros INTO :RETNAL;
  ELSE
    SELECT COUNT(ISBN) FROM Libros
      WHERE Editorial = :CodEditorial INTO :RETNAL;
    SUSPEND;
  END#
END#
```

```
SET TERM ; #
```

La primera sentencia cambia el carácter de fin ; por #, de tal forma que cuando Interactive SQL encuentre el ; que hay tras el primer SELECT no crea que el procedimiento almacenado finaliza ahí. La última sentencia devuelve este parámetro a su estado por defecto.

Observe que tras la ejecución de un SELECT u otro, dependiendo del valor de CodEditorial, se usa una sentencia llamada SUSPEND. Ésta hace posible la devolución del resultado desde el procedimiento almacenado al código que lo invoque.

El segundo procedimiento almacenado, NumTitulosEditorial, se crearía de forma muy similar. En este caso se declaran como valores de retorno tantas variables como columnas sigan a la cláusula SELECT, introduciendo los valores extraídos en ellas mediante INTO.

```
SET TERM # ;
CREATE PROCEDURE NumTitulosEditorial
RETURNS (Nombre VARCHAR(50), NumTitulos INTEGER)
AS
BEGIN
  FOR SELECT E.Nombre, COUNT(L.Titulo)
    FROM Editoriales E, Libros L
   WHERE E.IDEditorial = L.Editorial
     GROUP BY E.Nombre
     INTO :Nombre, :NumTitulos
  DO SUSPEND;
```

```
END#
SET TERM ; #
```

Con esto ya tenemos creados nuestros dos procedimientos almacenados. De forma similar podría crear otros más sofisticados, incluyendo, aparte de condicionales simples y consultas, bucles y otras operaciones sobre las tablas de la base de datos.

Ejecución de procedimientos almacenados

Como comprobación, puede ejecutar los procedimientos almacenados recién creados desde la propia herramienta Interactive SQL. Sólo tiene que introducir una sentencia `SELECT * FROM` seguida del nombre del procedimiento almacenado y, en el caso de `NumTitulos`, incluyendo los parámetros necesarios.

En la figura 3.51 puede ver la ejecución del procedimiento `NumTitulosEditorial`, mostrando en la parte inferior los resultados. Éstos son idénticos a los obtenidos en los ejemplos de SQL Server y Oracle.

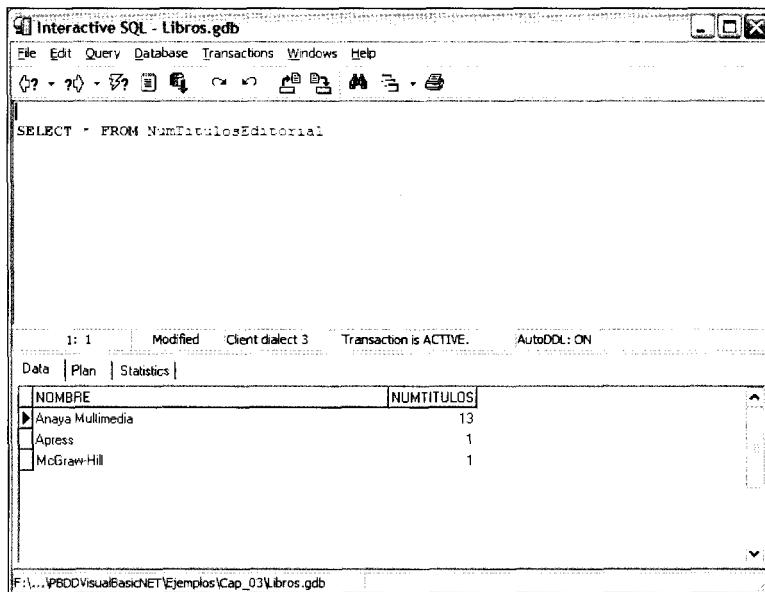


Figura 3.51. Ejecución del procedimiento almacenado `NumTitulosEditorial`

Microsoft Excel

Oracle, SQL Server e InterBase comparten el hecho de ser sistemas relacionales de gestión de datos, o bien RDBMS, guardando algunas similitudes con Microsoft Access que, a pesar de ser un sistema de datos de escritorio, no es un RDBMS, tam-

bien se usa para almacenar información estructurada y utiliza el lenguaje SQL. No toda la información con la que tiene que trabajar una aplicación está siempre estructurada como hemos visto en los puntos anteriores, en algunas ocasiones pueden ser datos más o menos organizados en documentos, por ejemplo hojas de cálculo Excel, datos en documentos XML e, incluso, información jerárquica como la que gestiona el Directorio activo de Windows 2000 y Windows .NET.

Microsoft Excel es una aplicación muy extendida, tanto o más que Access, siendo empleada por todo tipo de usuarios para gestionar sus datos, efectuar cálculos sobre ellos, representaciones gráficas, etc. Esta herramienta estructura la información en libros, cada uno de los cuales se compone de páginas que, a su vez, están formadas por celdillas. Desde ADO.NET es posible acceder a estos documentos como si de bases de datos se tratases, por lo que vamos a preparar una hoja de cálculo Excel que utilizaremos como origen de datos en capítulos posteriores.

Creación de un nuevo libro

Al iniciar Microsoft Excel se encontrará ya con un nuevo libro en blanco, mostrando la interfaz una apariencia similar a la que aparece en la figura 3.52. El libro tiene un nombre por defecto, *Libro1*, y cuenta inicialmente con tres páginas llamadas *Hoja1*, *Hoja2* y *Hoja3*, a las que puede acceder utilizando las pestañas que aparecen en la parte inferior de la ventana.

Observe que el área central tiene el aspecto de una cuadrícula, identificándose cada columna con una letra y cada fila con un número. El cruce de cada columna con cada fila es una *celdilla*, identificada por la letra de columna y número de fila, por ejemplo D5. A un conjunto de múltiples filas adyacentes, a lo largo de varias filas y/o columnas, es a lo que se llama *rango*.

Haga clic sobre el botón que muestra el icono de un disquete. Aparecerá el típico cuadro de diálogo para guardar archivos. Dé el nombre *Libros.xls* al libro. Ya tiene creado el equivalente a una base de datos vacía.

Nota

En la carpeta de ejemplos de este capítulo encontrará el archivo *Libros.xls* tal y como quedaría al final del proceso que va a describirse en los puntos siguientes.

Definición de la estructura

Con el objetivo de preparar este nuevo libro para introducir los datos que nos interesan, similares a los escritos en las bases de datos de ejemplo, vamos a hacer algunos cambios en su estructura. Los pasos, muy sencillos, son los que se describen a continuación:

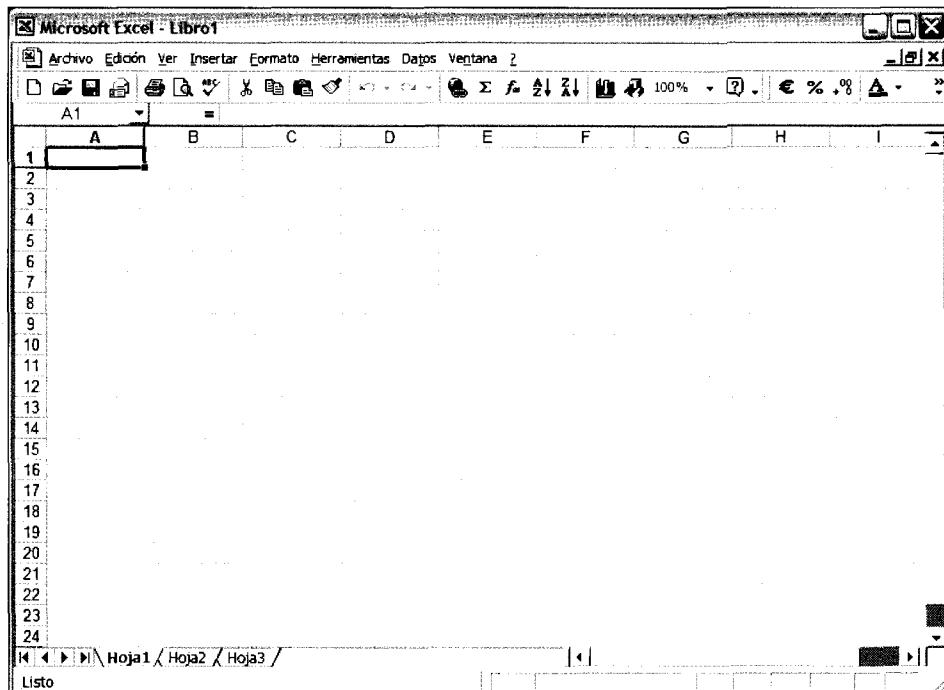


Figura 3.52. Apariencia de Microsoft Excel al iniciarse

- Haga clic con el botón secundario del ratón sobre la pestaña Hoja3, en la parte inferior del libro, y luego seleccione la opción Eliminar. Responda afirmativamente, pulsando el botón Aceptar, al aviso de que el borrado será permanente.
- Haga doble clic sobre la pestaña Hoja1. Verá que aparece un cursor que le permite modificar su nombre. Llámela Editoriales.
- Repita el paso anterior con la Hoja2 llamándola Libros.
- Desplace el cursor de la hoja Editoriales hasta la celdilla A1, en realidad ya debería estar allí si no lo movió antes, y escriba el texto IDEditorial, pulsando Intro al final.
- Muévase a la columna de la derecha, a la celdilla B1, y escriba Nombre. Repita la operación escribiendo Direccion en C1. De esta manera ha definido las tres columnas con que contará cada fila correspondiente a una editorial. En este momento el libro debería tener el aspecto que puede verse en la figura 3.53.
- Siguiendo la misma línea, escriba a continuación en las celdillas A1 a F1 de la hoja Libros los títulos IDLibro, ISBN, Titulo, Autor, Editorial y Precio.

Completados estos pasos, tenemos un libro Excel con una estructura similar a las tablas creadas previamente en Oracle o Access. Como habrá observado, no hemos indicado en punto alguno el tipo de los datos, las relaciones, claves, etc. Excel no es un RDBMS y la mayoría de estos aspectos no se contemplan. El tipo de cada una de las columnas, por ejemplo, será deducido por ADO.NET a partir del contenido que tengan las celdillas.

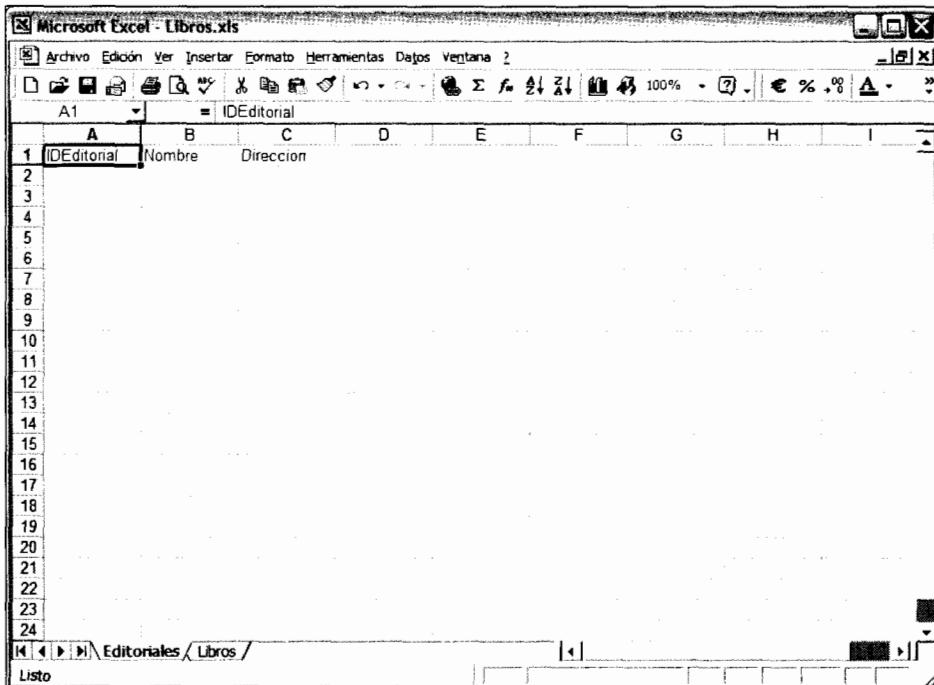


Figura 3.53. El libro Excel tras introducir los títulos de las columnas correspondientes a la hoja Editoriales

Introducción de datos

Microsoft Excel es una aplicación para usuario final y, por ello, la introducción de datos resulta muy sencilla, al contar con características como el autocompletado, la creación automática de secuencias, etc. Éste, sin embargo, no es un libro sobre Excel, por lo cual obviaremos toda esa funcionalidad y nos limitaremos a introducir los datos, sin más.

En el punto anterior, al escribir los títulos de las columnas, ya hemos introducido varios datos en cada una de las hojas del libro, aunque esos datos fuesen los títulos o nombres de las propias columnas.

Usando exactamente el mismo método escribiríamos la información de cada fila, no tenemos más que utilizar las teclas del cursor para situarnos en la celdilla

adecuada en cada caso. En la figura 3.54 puede ver el aspecto de la hoja Libros tras introducir una serie de datos.

A	B	C	D	E	F
1	IDlibro	ISBN	Título		
2		1 1-893115-94-1	User Interface Design for Programmers	Joel Spolsky	3 31
3		2 84-415-1136-5	SQL Server 2000	Francisco Charte	1 10.75
4		3 84-415-1324-4	Guía práctica para usuarios JBuilder 7	Francisco Charte	1 10.75
5		4 84-415-1392-9	Programación con Visual C# .NET	Francisco Charte	1 39
6		5 84-415-1376-7	Programación con Visual Studio .NET	Francisco Charte/Jorge Serrano	1 40
7		6 84-415-1351-1	Programación con Visual Basic .NET	Francisco Charte	1 39
8		7 84-415-1290-6	Guía práctica para usuarios de Visual Basic .NET	Francisco Charte	1 10.75
9		8 84-415-1291-4	Guía práctica para usuarios de Visual Studio .NET	Francisco Charte	1 10.52
10		9 84-415-1261-2	Programación con Delphi 6 y Kylix	Francisco Charte	1 37.26
11		10 84-415-1255-8	Guía práctica para usuarios de Delphi 6	Francisco Charte	1 10.52
12		11 84-415-1230-2	Manual avanzado Excel 2002	Francisco Charte	1 21.04
13		12 84-415-1202-7	Guía práctica para usuarios de Excel 2002	Francisco Charte/M Jesús Luque	1 10.52
14		13 84-415-1132-2	Guía práctica para usuarios de Kylix	Francisco Charte	1 10.52
15		14 84-415-1145-4	Introducción a la programación	Francisco Charte	1 24.04
16		15 84-7615-234-5	Manual del microprocesador 80386	Chris H Pappas&William H Murray. III	2 40
17					
18					
19					
20					
21					
22					
23					
24					
25					

Figura 3.54. Introducimos varias filas de datos en la hoja Libros de Excel

Nota

Para ajustar automáticamente el ancho de cada columna al apropiado para mostrar su contenido, sitúe el puntero del ratón entre la división de dos columnas en la fila de encabezado, por ejemplo entre las columnas A y B. En ese momento el cursor cambiará de apariencia, indicándole que puede pulsar el botón principal del ratón y arrastrar para modificar el ancho de la columna. Haga doble clic para dejar que sea el propio Excel quien efectúe el ajuste.

Usando las opciones de Excel podríamos, a partir de los datos introducidos, buscar automáticamente el nombre de la editorial a partir del código que hay en la hoja Libros, creando referencias en una nueva hoja que, en la práctica, funcionaría como una vista dinámica de los datos existentes en el libro. No obstante, para no extendernos en un campo que no es específicamente el que más nos interesa, nos limitaremos a guardar el libro dejándolo preparado para su uso en un capítulo posterior.

XML

Los libros de Excel se almacenan en un formato específico creado por Microsoft para este producto. A pesar de que existen filtros en algunas aplicaciones para poder utilizar su contenido, lo cierto es que el intercambio de información con otras aplicaciones y sistemas es limitado. Lo mismo ocurre con la representación interna de los datos de todos los RDBMS, sin excepción, tanto físicamente en disco como en memoria al transferir los datos desde el servidor a los clientes.

El lenguaje XML nació, precisamente, con el objetivo de facilitar el intercambio de información salvando las barreras en que se convierten en ocasiones las diferencias de plataforma, sistema operativo, lenguajes, etc. Basado en SGML, el lenguaje XML se caracteriza, como su propio nombre indica, por ser extensible, lo que significa que podemos crear nuestras propias marcas según las necesidades de cada caso.

ADO.NET utiliza XML como formato de representación interna de los conjuntos de datos, formato que también emplea a la hora de transferir la información entre máquinas, independientemente de que el origen de datos sea un RDBMS, un libro Excel o cualquier otro posible. Guardar los datos en formato XML y recuperarlos posteriormente, como veremos en su momento, es muy simple. En los dos puntos siguientes, por tanto, nos centraremos tan sólo en la aportación de algunos fundamentos sobre XML.

Definición de la estructura de documento

La estructura de los documentos XML se define mediante esquemas XML, también conocidos como esquemas XSD. Realmente, un esquema XSD es un documento XML que sigue unas reglas preestablecidas por el W3C, usando marcas específicas para la definición de tipos y elementos que deben existir en el documento.

Para crear un esquema XSD, al igual que para editar cualquier documento XML, no necesitamos más que un editor simple de texto, como podría ser el **Bloc de notas**. No obstante, como usuarios de Visual Studio .NET tenemos a nuestro alcance herramientas mucho más sofisticadas, herramientas que pueden ahorrarnos la edición manual de este tipo de archivos. Vamos a emplearlas para crear el esquema XSD del documento XML que crearemos posteriormente.

Partiendo de un proyecto vacío, abra el cuadro de diálogo **Agregar nuevo elemento y**, de la carpeta **Datos**, seleccione **Esquema XML**, como se muestra en la figura 3.55. Introduzca en la parte inferior el nombre del archivo donde se alojará el esquema, **Libros.xsd** en este caso.

En la superficie de trabajo aparecerá una indicación comunicándonos que podemos arrastrar y soltar componentes para comenzar a diseñar. Observe que en la parte inferior aparecen dos botones: **Esquema** y **XML**. Si pulsa este último verá la definición textual del esquema que, a medida que lo diseñemos visualmente, irá actualizándose. Tome del **Cuadro de herramientas** un **complexType** e insértelo en

cualquier punto del área central. Aparecerá un recuadro con dos columnas. Introduzca en la parte superior el nombre que va a darle al nuevo tipo, en este caso Editorial. A continuación vaya introduciendo, en la columna izquierda, el nombre de cada columna de datos, indicando a la derecha su tipo. En el detalle de la figura 3.56 puede ver cómo se han definido las tres columnas necesarias para registrar las editoriales.

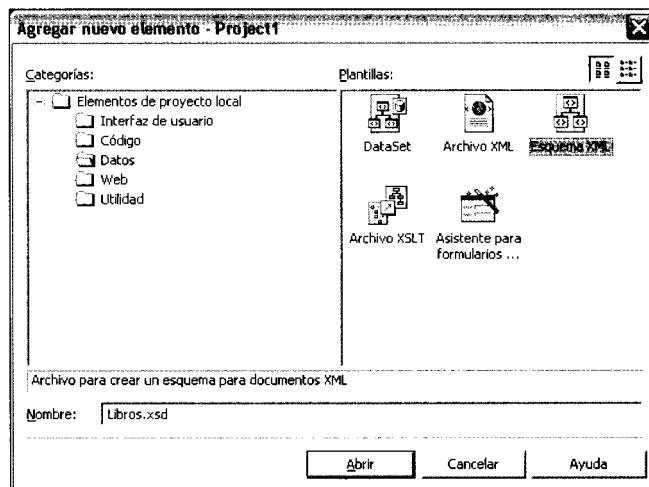


Figura 3.55. Añadimos al proyecto un esquema XML

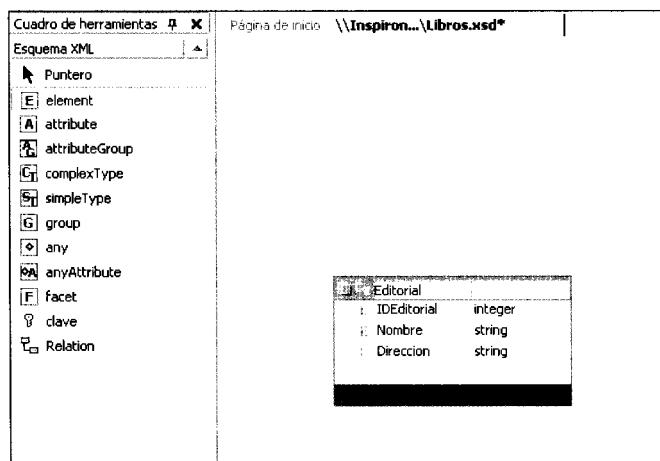


Figura 3.56. Definición del tipo complejo Editorial

Añada un nuevo tipo complejo, llamándole Libro, y defina el nombre y tipo de cada una de las columnas, tomando como referencia el libro Excel del punto anterior o cualquiera de las bases de datos creadas a modo de ejemplo.

El último paso será la inserción en la superficie de un element, al que llamaremos Libros. La diferencia entre un complexType y un element es la misma que existe entre la definición de una estructura de datos y la declaración de una variable. Editorial y Libro son tipos de datos, a partir de los cuales se pueden crear otros o bien declararse variables que, en el entorno XML, se conocen como elementos.

Nuestro elemento Libros estará compuesto de dos subelementos: Editoriales y Libros. El primero de ellos sería de tipo Editorial y el segundo de tipo Libro. De esta forma, el elemento Libros hace las veces de bases de datos, mientras que los subelementos indicados serían las tablas que habría en su interior. Visualmente el esquema quedaría como puede apreciarse en la figura 3.57. El código del esquema es el siguiente:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="Libros"
  targetNamespace="http://tempuri.org/Libros.xsd"
  elementFormDefault="qualified"
  xmlns="http://tempuri.org/Libros.xsd"
  xmlns:mstns="http://tempuri.org/Libros.xsd"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="Editorial">
    <xs:sequence>
      <xs:element name="IDEitorial" type="xs:integer" />
      <xs:element name="Nombre" type="xs:string" />
      <xs:element name="Direccion" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Libro">
    <xs:sequence>
      <xs:element name="IDLibro" type="xs:integer" />
      <xs:element name="ISBN" type="xs:string" />
      <xs:element name="Titulo" type="xs:string" />
      <xs:element name="Autor" type="xs:string" />
      <xs:element name="Editorial" type="xs:integer" />
      <xs:element name="Precio" type="xs:decimal" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Libros">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Editoriales" type="Editorial" />
        <xs:element name="Libros" type="Libro" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Guarde el documento Libros.xsd, esquema que, alojado en un archivo independiente, servirá para establecer la estructura de los documentos XML que actuarán como bases de datos, contando siempre con un elemento de primer nivel Libros que contendrá a Editoriales y Libros.

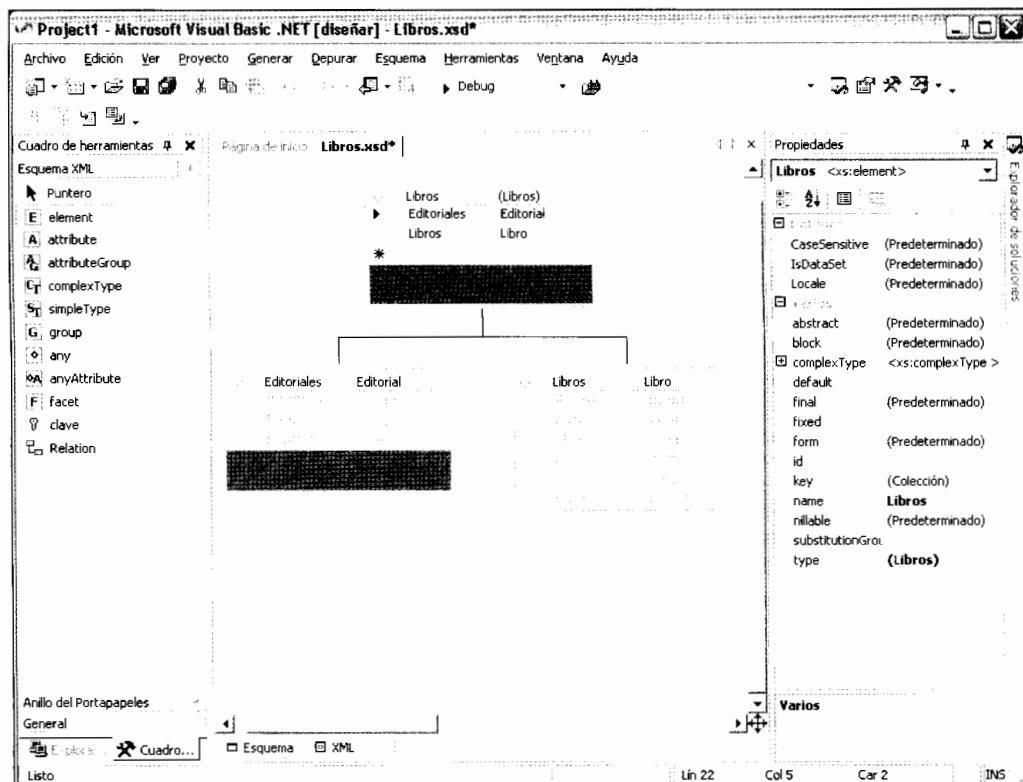


Figura 3.57. Aspecto visual del esquema XML en el entorno de Visual Studio .NET

Nota

En este ejemplo el esquema XSD utiliza el espacio de nombres `http://tempuri.org/`, usado habitualmente durante pruebas. En la práctica, habría que sustituir dicho espacio por la referencia correcta donde se encuentre el esquema.

Creación del documento XML

Partiendo del esquema anterior, crear un documento XML que se ajustase a su estructura es bastante fácil. El documento contendría un elemento raíz llamado **Libros** que, a su vez, tendría dentro los elementos **Editoriales** y **Libros**, compuestos cada uno de ellos de las columnas que contendrían los datos reales, tales como identificadores, nombre de la editorial, ISBN del libro, etc.

Abra de nuevo el cuadro de diálogo **Agregar nuevo elemento** y, en esta ocasión, seleccione de la carpeta **Datos** el elemento **Archivo XML**, introduciendo en la parte

inferior el nombre del archivo, por ejemplo `Libros.xml`. Al pulsar **Abrir** se encontrará en el editor de Visual Studio .NET con un documento compuesto de una sola línea:

```
<?xml version="1.0" encoding="utf-8" ?>
```

Esta línea indica simplemente que el archivo es un documento XML, comunicando el conjunto de caracteres en que se encuentra codificado.

Acceda a la ventana **Propiedades** y despliegue la lista que hay adjunta a la propiedad `targetSchema`. Seleccione el elemento `http://tempuri.org/Libros.xsd`, referencia al esquema que hemos creado en el punto anterior. En el editor aparecerá, bajo la anterior, la línea siguiente:

```
<Libros xmlns="http://tempuri.org/Libros.xsd"></Libros>
```

Ya tenemos la marca `<Libros>` que actúa como raíz de todo, como si fuese el inicio de la base de datos, así como la marca de cierre `</Libros>`.

Aunque podríamos ir introduciendo los datos manualmente, el editor de Visual Studio .NET se encarga de ir introduciendo de forma automática las etiquetas de cierre y, además, ofrece listas de posibles etiquetas y parámetros, es mucho más fácil pulsar el botón **Datos** que aparece en la parte inferior del editor. Al hacerlo nos encontramos con una interfaz como la de la figura 3.58, en la que podemos introducir los datos como si de una tabla de una base de datos se tratase. En el margen izquierdo seleccionamos el elemento a editar, mientras que a la derecha vamos añadiendo las filas de datos.

A medida que vamos editando visualmente los datos de cada editorial y libro, Visual Studio .NET se encarga de actualizar el documento XML para que siempre esté sincronizado. También podemos editar directamente el documento, viendo posteriormente los datos en forma de tabla. Lo interesante es que la información, el documento `Libros.xml` que puede ver completo a continuación, es compatible con todos los sistemas operativos, plataformas hardware y lenguajes de programación. Es, por tanto, el recurso ideal para el intercambio de información.

```
<?xml version="1.0" encoding="utf-8" ?>
<Libros xmlns="http://tempuri.org/Libros.xsd">
  <Libros>
    <IDLibro>15</IDLibro>
    <ISBN>84-7615-234-5</ISBN>
    <Titulo>Manual del microprocesador 80386</Titulo>
    <Autor>Chris H. Pappas&amp; William H. Murray, III</Autor>
    <Editorial>2</Editorial>
    <Precio>40</Precio>
  </Libros>
  <Libros>
    <IDLibro>14</IDLibro>
    <ISBN>84-415-1145-4</ISBN>
    <Titulo>Introducción a la programación</Titulo>
```

```
<Autor>Francisco Charte</Autor>
<Editorial>1</Editorial>
<Precio>24.04</Precio>
</Libros>
<Libros>
<IDLibro>13</IDLibro>
<ISBN>84-415-1132-2</ISBN>
<Titulo>Guía práctica para usuarios de Kylix</Titulo>
<Autor>Francisco Charte</Autor>
<Editorial>1</Editorial>
<Precio>10.52</Precio>
</Libros>
<Libros>
<IDLibro>12</IDLibro>
<ISBN>84-415-1202-7</ISBN>
<Titulo>Guía práctica para usuarios de Excel 2002</Titulo>
<Autor>Francisco Charte/M.Jesús Luque</Autor>
<Editorial>1</Editorial>
<Precio>10.52</Precio>
</Libros>
<Libros>
<IDLibro>11</IDLibro>
<ISBN>84-415-1230-2</ISBN>
<Titulo>Manual avanzado Excel 2002</Titulo>
<Autor>Francisco Charte</Autor>
<Editorial>1</Editorial>
<Precio>21.04</Precio>
</Libros>
<Libros>
<IDLibro>10</IDLibro>
<ISBN>84-415-1255-8</ISBN>
<Titulo>Guía práctica para usuarios de Delphi 6</Titulo>
<Autor>Francisco Charte</Autor>
<Editorial>1</Editorial>
<Precio>10.52</Precio>
</Libros>
<Libros>
<IDLibro>9</IDLibro>
<ISBN>84-415-1261-2</ISBN>
<Titulo>Programación con Delphi 6 y Kylix</Titulo>
<Autor>Francisco Charte</Autor>
<Editorial>1</Editorial>
<Precio>37.26</Precio>
</Libros>
<Libros>
<IDLibro>8</IDLibro>
<ISBN>84-415-1291-4</ISBN>
<Titulo>Guía práctica para usuarios de Visual Studio .NET</Titulo>
<Autor>Francisco Charte</Autor>
<Editorial>1</Editorial>
<Precio>10.52</Precio>
</Libros>
<Libros>
<IDLibro>7</IDLibro>
<ISBN>84-415-1290-6</ISBN>
```

```
<Título>Guía práctica para usuarios de Visual Basic .NET</Título>
<Autor>Francisco Charte</Autor>
<Editorial>1</Editorial>
<Precio>10.75</Precio>
</Libros>
<Libros>
    <IDLibro>6</IDLibro>
    <ISBN>84-415-1351-1</ISBN>
    <Título>Programación con Visual Basic .NET</Título>
    <Autor>Francisco Charte</Autor>
    <Editorial>1</Editorial>
    <Precio>39</Precio>
</Libros>
<Libros>
    <IDLibro>5</IDLibro>
    <ISBN>84-415-1376-7</ISBN>
    <Título>Programación con Visual Studio .NET</Título>
    <Autor>Francisco Charte/Jorge Serrano</Autor>
    <Editorial>1</Editorial>
    <Precio>40</Precio>
</Libros>
<Libros>
    <IDLibro>4</IDLibro>
    <ISBN>84-415-1392-9</ISBN>
    <Título>Programación con Visual C# .NET</Título>
    <Autor>Francisco Charte</Autor>
    <Editorial>1</Editorial>
    <Precio>39</Precio>
</Libros>
<Libros>
    <IDLibro>3</IDLibro>
    <ISBN>84-415-1324-4</ISBN>
    <Título>Guía práctica para usuarios JBuilder 7</Título>
    <Autor>Francisco Charte</Autor>
    <Editorial>1</Editorial>
    <Precio>10.75</Precio>
</Libros>
<Libros>
    <IDLibro>1</IDLibro>
    <ISBN>1-893115-94-1</ISBN>
    <Título>User Interface Design for Programmers</Título>
    <Autor>Joel Spolsky</Autor>
    <Editorial>3</Editorial>
    <Precio>31</Precio>
</Libros>
<Libros>
    <IDLibro>2</IDLibro>
    <ISBN>84-415-1136-5</ISBN>
    <Título>SQL Server 2000</Título>
    <Autor>Francisco Charte</Autor>
    <Editorial>1</Editorial>
    <Precio>10.75</Precio>
</Libros>
<Editoriales>
    <IDEEditorial>1</IDEEditorial>
```

```

<Nombre>Anaya Multimedia</Nombre>
<Direccion>Juan Ignacio Luca de Tena, 15</Direccion>
</Editoriales>
<Editoriales>
    <IDEEditorial>2</IDEEditorial>
    <Nombre>McGraw-Hill</Nombre>
    <Direccion>Edificio Valrealty, 1a planta</Direccion>
</Editoriales>
<Editoriales>
    <IDEEditorial>3</IDEEditorial>
    <Nombre>Apress</Nombre>
    <Direccion>901 Grayson Street</Direccion>
</Editoriales>
</Libros>

```

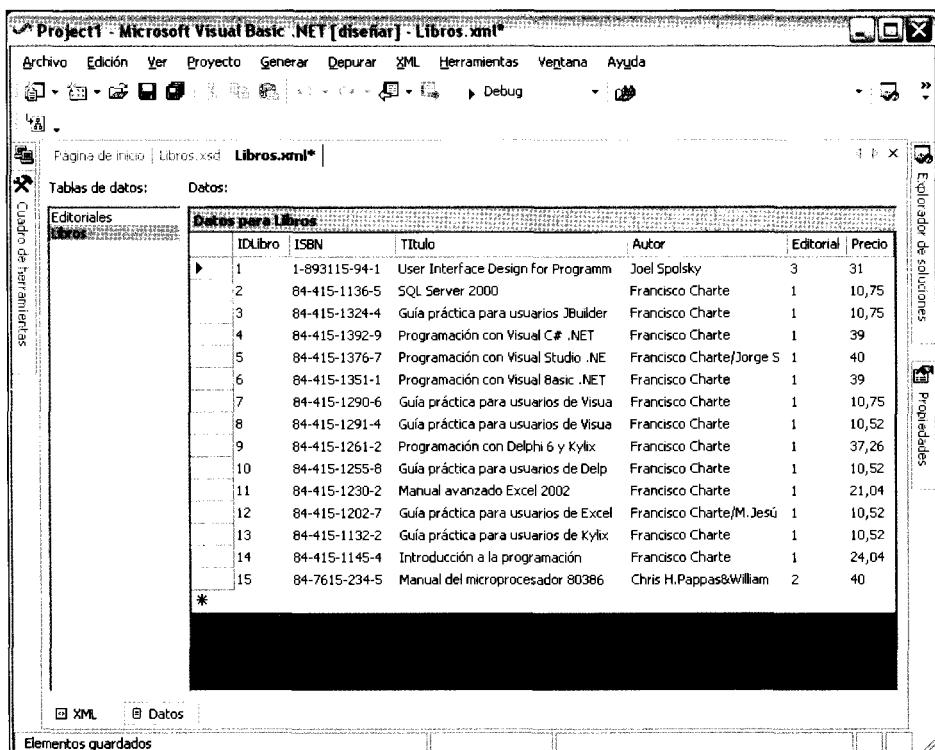


Figura 3.58. Introducción de datos en el documento XML

Note

En la carpeta de ejemplos de este capítulo encontrará los archivos `Libros.xsd` y `Libros.xml`, conteniendo el esquema y el documento XML desarrollados como ejemplo en este punto.

Directorio activo

En la mayoría de empresas actuales, especialmente de un tamaño medio a grande, existen redes heterogéneas, diferentes sistemas y todo ello distribuido geográficamente en distintas localizaciones que, incluso, se encuentran en distintos países. Gestionar los recursos de la infraestructura informática de este tipo de empresas supone un desafío para los administradores y, en ocasiones, también para los desarrolladores de aplicaciones.

Las redes se componen de ordenadores, dispositivos independientes o asociados a un ordenador, tales como impresoras o medios de almacenamiento, grupos o perfiles, cuentas de usuario, etc. Generalmente cada sistema cuenta con un servicio de directorio propio que facilita la administración de estos recursos, como el servicio de directorio de Windows NT o el de Novell Netware. Existe un protocolo, bastante aceptado, para acceder a estos servicios de directorio llamado LDAP.

Con el objetivo de simplificar la gestión de recursos en distintos tipos de directorio, Microsoft incluyó en Windows 2000 Server el Directorio activo (DA para abreviar) o *Active Directory*. Mediante DA es posible acceder a recursos de cualquier punto de la red, sin importar el servicio de directorio subyacente del sistema con que funcionen las máquinas. Gracias a DA, los administradores pueden organizar la información de forma mucho más coherente y organizada, simplificando la localización de los recursos y su gestión remota.

La información se almacena en el DA con estructura jerárquica, en lugar de relacional al estilo que hemos conocido en los puntos previos. En la posición más alta se encuentran los bosques, formados por árboles jerárquicos de dominios que, a su vez, alojan unidades organizativas. Éstas contienen máquinas, usuarios y otros elementos.

Nota

Por su complejidad, queda fuera del ámbito de este capítulo entrar en los detalles sobre el funcionamiento y los esquemas del DA. Puede encontrar más información sobre este servicio en <http://MSDN.Microsoft.es>.

Acceso al Directorio activo

Los administradores de sistemas acceden al DA mediante una interfaz de usuario típica, similar a la que puede verse en la figura 3.59. En el panel izquierdo aparece la jerarquía de contenidos, en este caso perteneciente a un dominio llamado estelar.net, mientras que a la derecha se muestran los objetos que existen en el contenido seleccionado en ese momento. Mediante operaciones de ratón, seleccionando elementos y accediendo a sus respectivos menús de opciones, puede efectuarse cualquier operación de gestión.

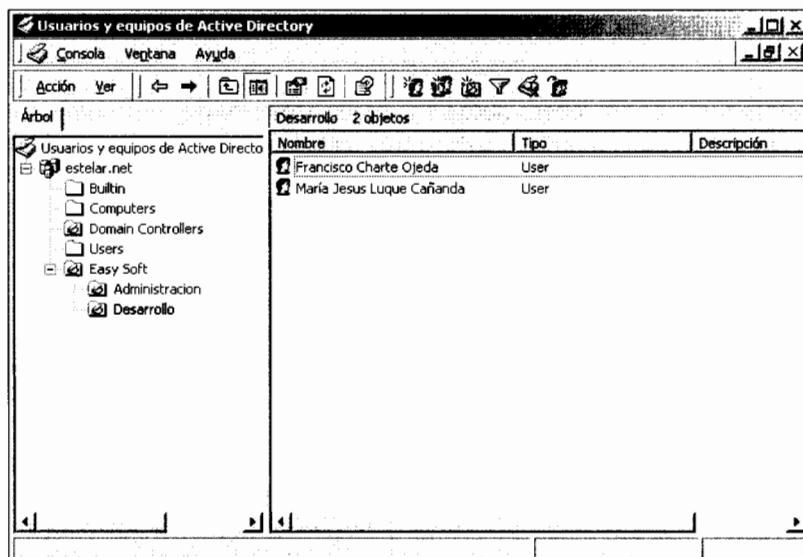


Figura 3.59. Herramienta de administración del Directorio activo

Como desarrolladores, lo que más nos interesará es el acceso al DA desde aplicaciones propias. Con ese objetivo se utilizan los servicios ADSI, una interfaz de programación compuesta de múltiples funciones a las que puede invocarse desde cualquier lenguaje.

Ya que la mayoría de desarrolladores de datos están habituados a emplear algún servicio de acceso a datos, una forma de simplificar el acceso al DA desde las aplicaciones consiste en utilizar el controlador OLE DB para DA, en lugar de llamar directamente a los servicios ADSI. Desde ADO.NET podemos configurar una cadena de conexión con cualquier controlador OLE DB, por lo que podemos usar ADO.NET para acceder al DA y, por ejemplo, localizar un recurso en el que esté interesado el usuario.

Un método alternativo, especialmente simple desde Visual Basic .NET, es usar los componentes `DirectoryEntry` y `DirectorySearcher` del ámbito `System.DirectoryServices`. Gracias a ellos es posible efectuar todas las operaciones disponibles con ADSI pero sin tener que llamar directamente a las funciones de esta interfaz.

Resumen

A lo largo de este capítulo ha creado múltiples bases de datos y documentos, recursos que nos servirán como orígenes de datos para los ejemplos a desarrollar en capítulos posteriores. Por el camino, y aunque muy vagamente, también habrá adquirido cierta familiaridad con algunos de los productos, RDBMS y no RDBMS,

que van a usarse como servidores o productores iniciales de los datos. Ha aprendido a definir estructuras de tablas en distintas bases de datos, codificar procedimientos almacenados en los lenguajes de varias de ellas, crear vistas y ejecutar algunos de esos elementos para obtener unos resultados visibles de manera inmediata.

El objetivo principal, más allá de esa familiarización con SQL Server, Oracle o Excel, ha sido definir la estructura e introducir los datos que, tal y como se ha comentado, nos servirán de base para los siguientes capítulos. Antes, no obstante, deberemos conocer también otros elementos, como la estructura de ADO.NET y sus componentes principales. A ello están dedicados los dos capítulos que encontrará a continuación de éste.



4

Introducción a ADO.NET

Con la denominación genérica ADO.NET se hace referencia a todos los servicios de acceso a datos disponibles en la plataforma Microsoft .NET, servicios que podemos usar desde cualquiera de los lenguajes de programación capaces de producir código MSIL, entre ellos Visual Basic.

Los elementos de ADO.NET están pensados para simplificar el acceso a datos, sin perder por ello flexibilidad y eficacia. Sus componentes se encuentran repartidos por varios ámbitos con nombre que, en su mayor parte, tendrá ocasión de conocer en el próximo capítulo.

Nuestro objetivo, en este capítulo, es obtener una visión general de ADO.NET desde un punto de vista bastante teórico, obteniendo los conocimientos necesarios para que, a medida que conozcamos estos servicios en los siguientes capítulos, comprendamos fácilmente su funcionamiento y razón de ser. Desde este nivel, relativamente alto, será más sencillo percibir la arquitectura de ADO.NET.

Objetivos del modelo ADO.NET

Microsoft cuenta desde hace años con un mecanismo de acceso a datos, ADO, muy asentado en el sector, disponible en la práctica totalidad de las versiones de Windows y usado por casi todas las herramientas de desarrollo disponibles. ADO, gracias a la existencia de diversos controladores OLE DB, ofrece acceso a orígenes de datos de todo tipo y facilita al desarrollador las operaciones más habituales.

Para proponer un nuevo modelo de acceso a datos, en este caso ADO.NET, era preciso alcanzar unos objetivos que le hiciesen superior a ADO y, por tanto, el cambio evolutivo de uno a otro, por parte de los desarrolladores, mereciese la pena. Dichos objetivos se han llegado a alcanzar y podrían resumirse en los siguientes puntos:

- **Modelo de objetos más simple y racional.** Partiendo de unas interfaces genéricas, y aprovechando las características de orientación a objetos con que cuentan todos los lenguajes .NET, el modelo de objetos es mucho más simple y fácil de usar, con una curva de aprendizaje más suave respecto a ADO. También resulta más sencillo extender este modelo, añadiendo nuevas clases de objetos y componentes.
- **Representación y transporte en formato XML.** A pesar de que las últimas actualizaciones de ADO permitían guardar los *recordsets* en formato XML, lo cierto es que la integración total con este formato se alcanza en ADO.NET. La transferencia de los datos se efectúa en formato XML e, internamente, los conjuntos de datos se representan como documentos XML. Esto simplifica la interoperabilidad con otras aplicaciones, al poder entregar y recoger datos en un formato entendido por todos los sistemas operativos y lenguajes.
- **Menor uso de recursos en el servidor.** Uno de los grandes problemas actuales de los RDBMS es la capacidad que tienen para atender solicitudes de conexión por parte de los clientes, puesto que cada uno de ellos abre una o más conexiones y su número se encuentra limitado físicamente por los recursos de la máquina. ADO.NET no utiliza cursos en el servidor y, además, opera usando un modelo de conjuntos de datos desconectados y comandos. Esto significa que la conexión se mantiene sólo el tiempo necesario para ejecutar un comando, ya sea de recuperación o manipulación de datos, trabajando el resto del tiempo sin consumir ningún recurso del servidor. El resultado es que éste se hace mucho más escalable, pudiendo atender a un número de clientes muy superior sin problemas y sin necesidad de ampliar los recursos.
- **Mejor rendimiento.** El rendimiento en general de las operaciones efectuadas mediante ADO.NET es superior al obtenido con otros mecanismos de acceso a datos, en parte gracias a la existencia de proveedores nativos para trabajar sobre SQL Server y Oracle, donde se obtiene el mayor beneficio, así como acceso directo a controladores ODBC, sin necesidad de pasar por el puente OLE DB-ODBC que añade una capa más y reduce la velocidad con que se efectúan las transacciones.
- **Solución global de acceso a datos.** Tal y como se indicaba en la introducción, ADO.NET es una solución global de acceso a datos, al hacer posible el trabajo con la mayoría de orígenes de datos existentes y, al mismo tiempo, ser un servicio disponible para todos los lenguajes .NET, incluidos aquellos no desarrollados por Microsoft. Es el único mecanismo que necesitaremos, in-

dependientemente de que nuestra aplicación vaya a ejecutarse sólo en un ordenador de manera aislada o en una configuración distribuida con servidores de datos y aplicaciones.

Los cambios en ocasiones son para mejor y, en ésta concretamente, seguramente lo será si comparamos las características de ADO.NET con las del sistema de acceso a datos que estemos empleando actualmente.

Representación interna en XML

Aunque ADO.NET cuenta con un mecanismo para lectura directa de datos desde el servidor al cliente, utilizando una conexión unidireccional y sólo de lectura especialmente apropiada para la elaboración de informes y tareas similares, lo más habitual será trabajar con conjuntos de datos en el cliente.

Los pasos para obtener un conjunto de datos o *DataSet* son, por regla general, los siguientes:

1. Se establece una conexión con el origen de datos.
2. Ejecución de un comando que recupera información de dicho origen.
3. Creación en memoria del conjunto de datos.
4. Cierre de la conexión.

El conjunto de datos se crea en el cliente, donde está ejecutándose la aplicación, siendo su estructura totalmente independiente del origen de datos. Esto significa que el conjunto de datos sería exactamente el mismo sin importar que los datos hayan sido recuperados de SQL Server, Oracle, Excel o un documento XML. A partir de ese momento, la transferencia de la información alojada en el conjunto de datos se efectúa siempre en formato XML. Un componente que se ejecuta en un servidor de aplicaciones, y se comunica con el RDBMS, puede enviar los conjuntos de datos a clientes remotos incluso si son aplicaciones que se ejecutan en otro sistema operativo distinto a Windows.

La figura 4.1 representa una configuración hipotética en la que el equipo central, destacado con un tamaño algo mayor, es el servidor de aplicaciones que, funcionando con Windows, utiliza ADO.NET para acceder a un RDBMS, pongamos por caso Oracle, ejecutándose en una máquina Unix. Una vez que el servidor de aplicaciones envía al de datos el comando de recuperación de datos, y éste los devuelve al primero, tenemos un conjunto de datos ADO.NET alojado en un componente. Éste se comunica mediante protocolos estándar con diferentes tipos de clientes, entregando el conjunto de datos en formato XML. El primer cliente, el que está más a la izquierda, podría ser una aplicación Windows típica, el segundo un cliente web sobre cualquier sistema y el tercero una aplicación escrita, por ejemplo, con Borland Kylix funcionando en Linux.

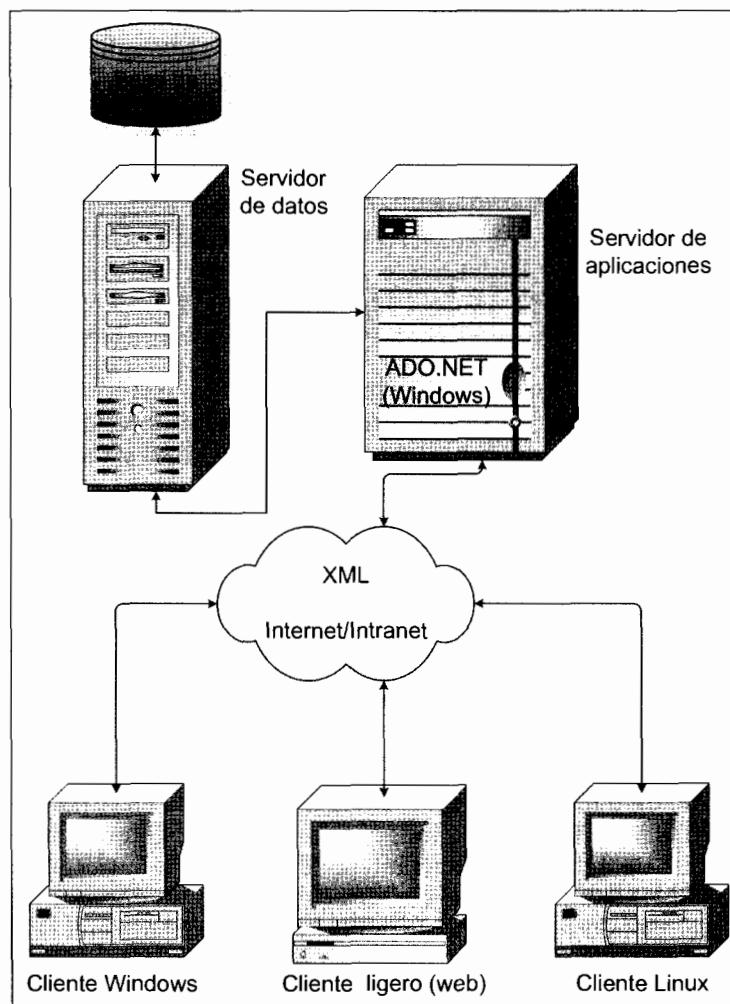


Figura 4.1. La transferencia de los conjuntos de datos en formato XML abre las puertas a la interoperabilidad con clientes heterogéneos

Los conjuntos de datos son estructuras dinámicas. Es posible insertar, modificar y eliminar información, operaciones que se registran temporalmente en forma de *DiffGrams*. Estos sirven para resolver posteriormente con el origen de datos, generando los comandos apropiados de inserción, actualización o eliminación.

Nota

En el noveno capítulo, dedicado al estudio de los conjuntos de datos, conocerá algunos detalles sobre los *DiffGrams*.

Ausencia de cursores de datos

Se indicaba en un punto previo que ADO.NET no emplea cursores en el servidor, reduciendo el uso de recursos en los servidores de datos. Si está acostumbrado a utilizar este elemento en sus aplicaciones, algo habitual en la mayoría de motores de acceso a datos actuales, seguramente se preguntará cómo va a poder efectuar su trabajo sin poder crear un cursor.

Tradicionalmente, las aplicaciones establecían una conexión con el servidor y éste creaba un cursor que facilitaba el acceso a los datos por parte del cliente. Éste, mediante operaciones simples, solicitaba la fila anterior o siguiente, la inserción o edición de datos. El cursor se encargaba de bloquear los datos que estaban en edición, evitando el acceso por parte de otros usuarios, siguiendo un cierto esquema de comportamiento. El uso de cursores, y la configuración de funcionamiento de esas aplicaciones, con conexión continua con el servidor de datos, tenían sentido especialmente en los escenarios cliente/servidor en el que los clientes conectaban directamente con el RDBMS y, además, su número era fácilmente predecible según el tamaño y necesidades de la empresa.

La extensión de las tecnologías de Internet y, especialmente, las aplicaciones que se ejecutan en un cliente Web y los servicios Web, han modificado de manera significativa las necesidades y, por tanto, es preciso buscar nuevas soluciones. Mantener una conexión continua con el servidor de datos no es lo más apropiado y, en ocasiones, ni siquiera es posible en una red donde la conexión no está garantizada. Además, las aplicaciones que emplean los clientes suelen ser meras interfaces de usuario, sin posibilidades de acceso directo a un RDBMS y que dependen de componentes alojados en un servidor de aplicaciones según la configuración esbozada en la figura 4.1.

Tampoco es fácil anticipar el número de clientes que existirán, especialmente en aplicaciones que no se ejecutan en una *intranet* corporativa sino a través de Internet y abierta a todos los potenciales usuarios. Si cada uno de ellos abriese una conexión con nuestro servidor y crease su propio cursor de datos, posiblemente nos viéramos obligados a incrementar constantemente los recursos de la máquina, entre ellos la memoria.

Frente a estos nuevos problemas, ADO.NET ofrece soluciones como el trabajo sin conexión continua y los conjuntos de datos sin conexión. La conexión con el servidor de datos es transitoria, sólo durante el tiempo preciso para ejecutar un comando y obtener el correspondiente resultado. Éste se transfiere al servidor de aplicaciones o cliente, dependiendo de dónde esté ejecutándose ADO.NET, y a partir de ese momento se opera sobre un conjunto de datos sin conexión. Las acciones típicas sobre el cursor, como el desplazamiento por las filas, se efectúa de manera inmediata en el cliente, sin tener que enviar una solicitud al servidor y esperar la respuesta ni, por supuesto, consumir recurso alguno en el servidor.

Si todas las acciones se efectúan sobre un conjunto de datos sin conexión, ¿cómo se actualiza la información en el servidor? En el modelo cliente servidor, segúin

decía antes, era el propio cursor el que se encargaba de ir bloqueando y gestionando los conflictos entre clientes. En ADO.NET se generan comandos de actualización de forma dinámica, enviándose al servidor cuando es necesario consolidar la información. En caso de conflicto, se obtiene una enumeración de los problemas que es preciso gestionar en el cliente.

En resumen, ADO.NET se adapta mejor a las configuraciones distribuidas, utilicen o no Internet como medio de comunicación, al tiempo que ofrece una alternativa al uso de cursores en configuraciones cliente/servidor.

Cursos de lectura

A pesar de lo dicho, aún queda en ADO.NET un vestigio del uso de cursores: los lectores de datos o `DataReader`. Éstos se usan para ejecutar sólo sentencias de consulta, obteniéndose un cursor ligero que sólo permite la lectura y, además, es unidireccional, facilitando tan sólo el avance de una fila a la siguiente hasta llegar al final.

Los lectores de datos son útiles en casos en los que va a recuperarse un conjunto de datos con el objetivo de preparar un informe, realizar unos cálculos o tareas similares, de tal forma que, aunque se mantenga la conexión abierta con el servidor, el proceso dure el menor tiempo posible.

La existencia de estos lectores de datos hace posible la recuperación de información sin necesidad de transferir todo el conjunto de datos completo desde el servidor al cliente en un solo paso, reduciendo así el uso de memoria en el cliente, que no tiene necesidad de almacenar todos los datos al poder procesarlos fila a fila. En el octavo capítulo conocerá los componentes `DataReader` y aprenderá a utilizarlos en aplicaciones propias.

Solución multipropósito

Los servicios de ADO.NET dan cabida a las necesidades de todo tipo de aplicaciones, desde las más sencillas de tipo monousuario hasta aquellas que deben operar con conexión esporádica al servidor. Esto demuestra la flexibilidad y versatilidad de ADO.NET respecto a otros mecanismos de acceso a datos, mucho más limitados en cuanto a posibilidades se refiere.

Las distintas combinaciones de componentes ADO.NET, proveedores y controladores dan como resultado diferentes soluciones que pueden aplicarse a múltiples escenarios, entre los cuales se encontrarían las siguientes configuraciones:

- Monousuario: Aplicaciones con necesidades básicas de tratamiento de datos, almacenamiento local de la información, en la misma máquina, y que no precisan la consolidación con un servidor. Es el modelo representado esquemáticamente en la figura 4.2, en la que el recuadro representaría al ordena-

dor del cliente y los elementos que hay en su interior los diversos componentes. La aplicación utilizaría un conjunto de datos ADO.NET que podría guardar directamente, en formato XML, o bien emplear una base de datos de escritorio o un documento Excel como depósito de la información. La primera opción es la más eficiente y simple, ya que no se precisa ninguna otra aplicación externa como sería Excel, mientras que la segunda habilitaría la posibilidad de tratar la información desde fuera de la aplicación cliente, con un producto estándar.

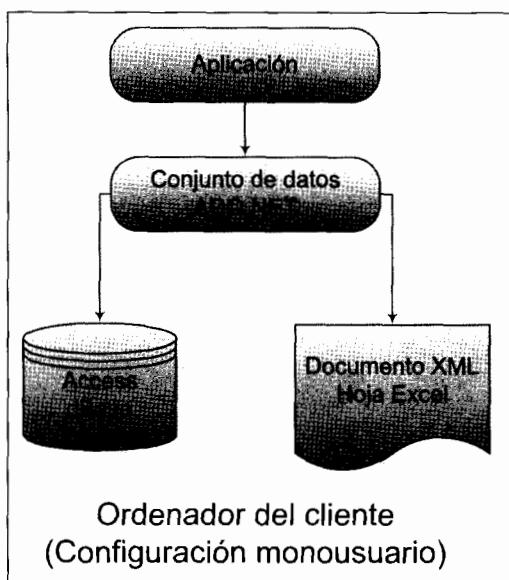


Figura 4.2. Representación esquemática de una configuración monousuario

Nota

Los conjuntos de datos o `DataSet` pueden guardarse y recuperarse localmente con una llamada a un método, siendo la alternativa más simple y rápida para el tratamiento de datos en aplicaciones sencillas.

- Cliente/Servidor: La configuración más utilizada en la última década del pasado siglo, separando la interfaz de usuario y lógica de aplicación, que se ejecutarían en el ordenador del cliente conjuntamente con ADO.NET, del almacenamiento de datos, que queda en manos de un servidor. En esta disposición aparecen en escena, como se aprecia en la figura 4.3, varios elementos más, como el adaptador y el proveedor de datos y/o controlador, según los casos. El adaptador de datos se comunica con el proveedor y hace posible la independencia de los conjuntos de datos, al separar su representación de la

que utiliza el origen de datos nativo. A pesar de que en esta configuración, típica de las redes internas de las empresas, sería posible una conexión continua con el servidor de datos, ADO.NET utiliza el mecanismo antes explicado de ejecución de comandos y creación de conjuntos de datos locales en el cliente, reduciendo la carga del RDBMS.

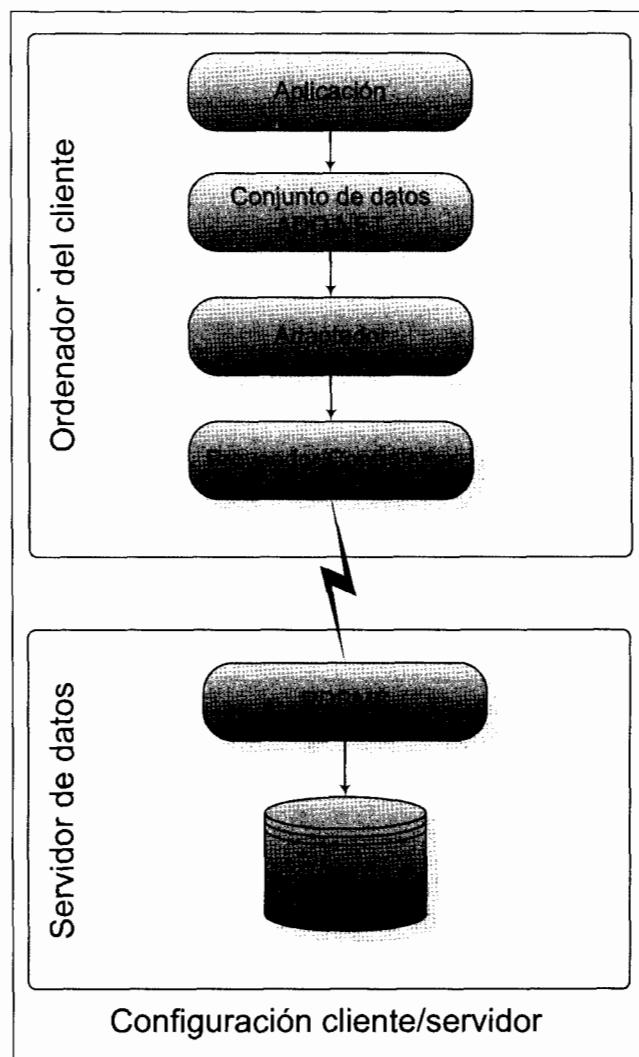


Figura 4.3. Esquema de una configuración cliente/servidor típica

- Cliente/Servidor con conexión esporádica: Similar a la anterior, se caracteriza porque el cliente en ocasiones tiene la necesidad de trabajar con los datos sin contar con una conexión con el servidor, precisando una consolidación

posterior, cuando la conexión es posible. Eso es lo que ocurre, por ejemplo, cuando se trabaja con un ordenador portátil durante un viaje, por ejemplo tomando pedidos de clientes que después es necesario transferir a la base de datos para su proceso. La configuración sería la de la figura 4.4. La aplicación cliente se sirve de un conjunto de datos que almacena localmente, en formato XML, en el mismo ordenador. Cuando la conexión con el servidor está disponible, se usan los mismos elementos cliente/servidor indicados antes para ejecutar los comandos de actualización apropiados.

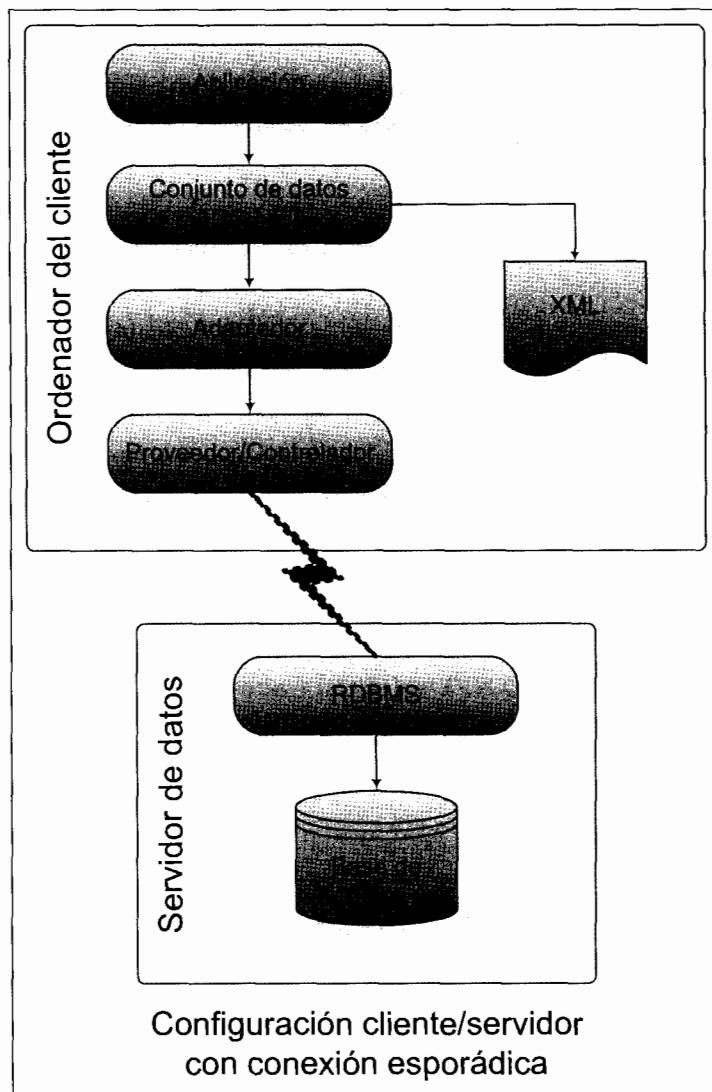


Figura 4.4. La configuración con conexión esporádica es similar a la cliente/servidor

- **Distribuida:** Cuando las empresas tienen un cierto tamaño, el mantenimiento de aplicaciones que siguen el modelo cliente / servidor se convierte en un problema, ya que la actualización de los algoritmos de proceso de la información se encuentran codificados en la aplicación que ejecuta cada ordenador cliente, lo cual implica que cualquier cambio conlleve una actualización al nivel de toda la empresa. Las aplicaciones distribuidas solucionan este problema, al aislar la lógica de proceso de datos en un servidor de aplicaciones que actúa como intermediario entre clientes y RDBMS. Es el modelo representado en la figura 4.5. En él aparecen en el servidor de aplicaciones los componentes ADO.NET necesarios para comunicarse con el RDBMS y efectuar todas las operaciones de manipulación de datos. La información se transfiere a y desde el cliente, convertido en una simple interfaz de usuario, usando el formato XML. En este caso se habla de un modelo en tres capas o *three-tier*, por ser el más habitual, pero es posible añadir capas adicionales según las necesidades de la empresa.

Nota

Los componentes que se ejecutan en el servidor de aplicaciones pueden ser componentes .NET o bien servicios Web. Éstos cuentan con algunas ventajas, al ser más fácilmente accesibles a través de redes en las cuales existen cortafuegos.

- **Web:** En realidad es una variante del modelo anterior. Básicamente se sustituye en el cliente la aplicación nativa, basada en formularios Windows, por una aplicación tipo ASP.NET, obteniéndose un *cliente ligero* ejecutable en cualquier navegador. Los elementos ADO.NET así como su distribución serían idénticos.

Como puede ver, ADO.NET solventará nuestras necesidades en todos los casos, empleando siempre los mismos elementos básicos: proveedores ADO.NET, controladores en caso de ser necesarios, adaptadores y conjuntos de datos.

Configuración de los clientes

Mientras desarrollamos una aplicación de acceso a datos con Visual Studio .NET, comprobando su funcionamiento en una máquina de desarrollo, nunca tendremos problemas con los componentes porque al instalar Visual Studio .NET se instalan también automáticamente los MDAC, concretamente la versión 2.6. Hay que tener en cuenta, sin embargo, que las aplicaciones, al finalizar la fase de desarrollo y depuración, van a instalarse en máquinas clientes en las que, lógicamente, no existirá Visual Studio .NET.

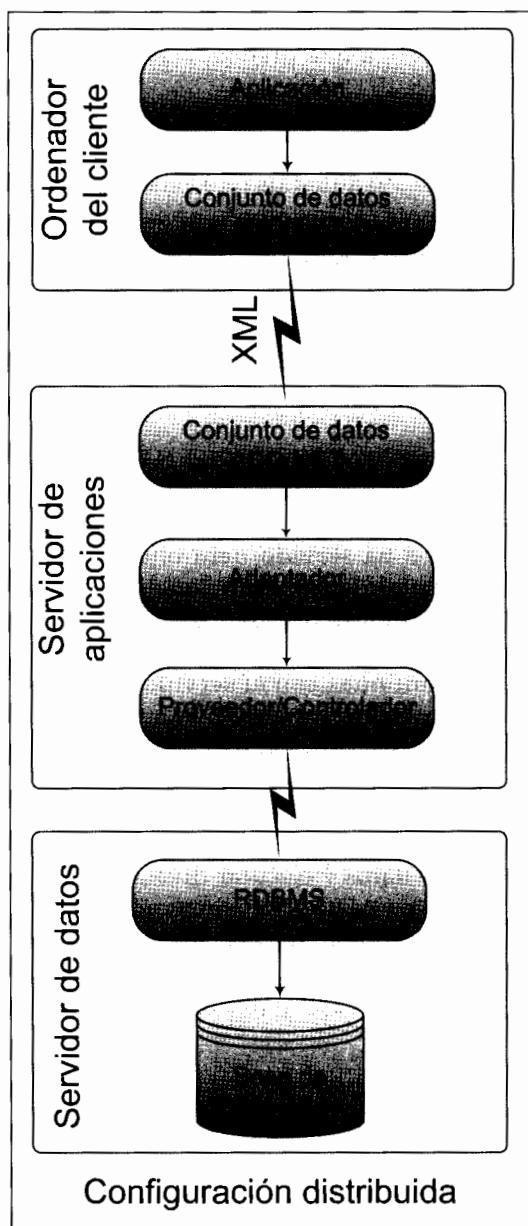


Figura 4.5. Configuración de una aplicación distribuida en tres capas

En cualquier ordenador donde vayan a utilizarse los servicios de ADO.NET, ya sea un cliente, en los esquemas de las figuras 4.2 y 4.3, o bien los componentes de un servidor de aplicaciones, en la figura 4.5, debe instalarse la versión 2.6 o posterior de los componentes MDAC, para lo cual existe un archivo redistribuible que,

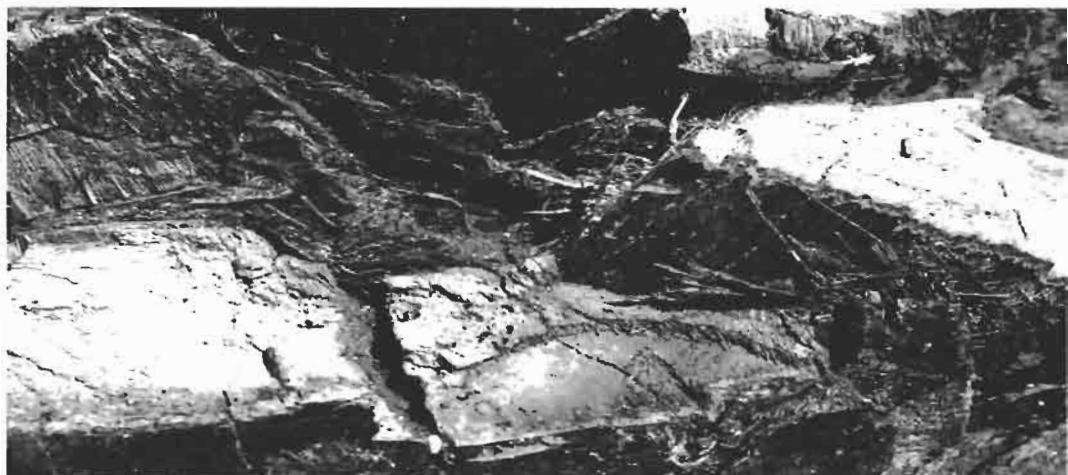
incluso, es posible combinar con la instalación de nuestra propia aplicación, de tal forma que el usuario no tenga que preocuparse de obtener e instalar dichos componentes.

Otro factor que hay que tener en cuenta son las referencias a servidores y caminos de archivos. Durante la fase de desarrollo habitualmente se utilizarán servidores de datos de pruebas, no aquellos que están dando servicio a las aplicaciones en explotación. Lógicamente, al instalar la aplicación debe pasar a utilizarse, precisamente, ese servidor de explotación. En lugar de introducir directamente las referencias en el código o los componentes, siempre existe la posibilidad de recuperar esa información de un archivo de configuración, facilitando así el cambio de un servidor a otro sin necesidad de tocar la aplicación, simplemente editando un archivo de texto.

Resumen

Este breve capítulo nos ha servido para obtener, de una manera teórica, una idea general sobre las posibilidades y funcionalidad del modelo de acceso a datos ADO.NET, un modelo compuesto por interfaces y componentes que, en su mayor parte, va a conocer en el próximo capítulo. Ahora ya sabe que ADO.NET no utiliza cursores ni conexiones continuas con el servidor para efectuar su trabajo, que los conjuntos de datos se transfieren y almacenan en formato XML y que es posible utilizar ADO.NET en cualquier configuración, desde la más sencilla hasta la más compleja.

A partir del sexto capítulo, tras conocer en el quinto los ámbitos, interfaces y clases más importantes de ADO.NET, comenzará a usar todos esos elementos para conectar con un origen de datos y ejecutar diversos comandos. Todos los ejemplos se basan en el uso directo de ADO.NET, sin ayuda de los asistentes, diseñadores y elementos de Visual Studio .NET. Una vez conozca perfectamente cómo funciona y cómo utilizar ADO.NET, pasaremos, en la tercera parte, a estudiar cómo Visual Studio .NET puede ahorrarnos una parte importante del trabajo.



5

Modelo de objetos

Aunque manteniéndonos aún en el campo teórico iniciado en el capítulo previo, en éste vamos a ir conociendo los ámbitos con nombre que componen ADO.NET y las interfaces y clases que podemos encontrar en ellos, información fundamental para, en los capítulos siguientes, ir utilizándolos para conectar con un origen de datos, recuperar información, ejecutar comandos, etc.

Recuerde que en la ayuda electrónica de Visual Studio .NET existe una referencia de toda la biblioteca de clases .NET, en la que encontrará la lista de elementos relacionados con ADO.NET.

Nuestro objetivo, en este capítulo, es introducir los elementos de mayor interés y esbozar un esquema general del modelo de objetos ADO.NET, pero recuerde que todos los detalles sobre ámbitos, clases y sus miembros se encuentran en la información de referencia de Visual Studio .NET.

Estructura del modelo de objetos

El modelo de objetos ADO.NET está construido de forma coherente y facilitando la posterior extensión, gracias a la herencia y la implementación de interfaces. Sus elementos podrían clasificarse en dos categorías: dependientes e independientes del origen de datos. En la primera entrarían todas las clases específicas para cada origen de datos concreto, como pueden ser `OleDbDataAdapter` y `SqlDataAdapter` para OLE DB y SQL Server, respectivamente. La segunda se compone de

clases que no tienen dependencias respecto al origen de datos, como `DataSet` o `DataRelation`.

Los elementos independientes del origen de datos son de uso general, independientemente de que conectemos con SQL Server, Oracle, dBase, Excel o un archivo XML. El mayor exponente es la clase `DataSet`, que representa un conjunto de datos en una aplicación sin importar de dónde se haya extraído. Las operaciones disponibles son siempre las mismas, lo cual nos facilitará el trabajo al no tener que recurrir a distintos caminos según el origen de datos empleado.

Aquellos componentes que necesitan comunicarse con el origen de datos, y entender los comandos, funciones y estructuras de éste, implementan interfaces comunes a pesar de sus diferencias. Así, es posible tratar de manera homogénea un adaptador de datos de SQL Server o de Oracle, por poner un ejemplo, ya que cuentan con similares métodos y propiedades aunque la implementación, como es lógico, difiere de un componente a otro.

Dada la independencia del origen de datos, las clases de uso genérico, como la citada `DataSet`, no necesitan ser extendidas para usos específicos. El conjunto de datos siempre será el mismo, sin importar el origen de datos del que se haya obtenido o al que se vaya a enviar. Las clases con dependencias, por el contrario, tienen una estructura que permite una ampliación fácil. Esto ha facilitado, por ejemplo, que Microsoft añada rápidamente proveedores de datos específicos para ODBC y Oracle a los ofrecidos inicialmente con Visual Studio .NET para SQL Server y OLE DB. De igual manera, terceros fabricantes, e incluso nosotros mismos, podemos crear proveedores específicos para nuestros orígenes de datos.

Ámbitos con nombre de ADO.NET

La instalación por defecto de la plataforma .NET aporta cinco ámbitos con nombre o *namespaces* relacionados con ADO.NET, a los cuales habría que añadir uno o más por cada proveedor adicional que pudiéramos instalar después. Estos cinco ámbitos son:

- `System.Data`: Aloja las clases independientes del origen de datos, así como las interfaces que deben implementar las clases que son dependientes.
- `System.Data.Common`: Contiene clases que facilitan la implementación de las interfaces existentes en `System.Data` por parte de los distintos proveedores de datos, así como otras compartidas por todos los proveedores.
- `System.Data.OleDb`: Corresponde al proveedor ADO.NET que permite utilizar cualquier controlador OLE DB para conectar con un origen de datos. En él se encuentran implementaciones específicas de clases para comunicarse mediante OLE DB.
- `System.Data.SqlClient`: Como el anterior, alberga clases específicas para operar sobre un determinado origen de datos, en este caso SQL Server.

- `System.Data.SqlTypes`: Relacionado con el anterior, en este ámbito encontramos definiciones de tipos de datos específicos para trabajar con SQL Server.

Instalando los dos proveedores adicionales que Microsoft ha publicado hasta el momento para ADO.NET, a los anteriores habría que sumar los dos ámbitos siguientes:

- `System.Data.OracleClient`: Correspondiente al proveedor nativo para acceso a bases de datos Oracle. Al igual que `System.Data.SqlClient` o `System.Data.OleDb`, contiene implementaciones específicas de clases que facilitan la conexión y comunicación con este RDBMS.
- `Microsoft.Data.Odbc`: Este ámbito corresponde al proveedor ODBC nativo de ADO.NET, conteniendo los elementos que son necesarios para poder acceder a cualquier origen de datos para el que haya disponible un controlador OLE DB.

Nota

Para poder utilizar estos dos últimos, en caso de que los necesite, tendrá que añadir una referencia en la carpeta Referencias del proyecto al ensamblado correspondiente. En un capítulo posterior se indicará de dónde puede obtener estos controladores y cómo instalarlos y usarlos.

Interfaces para los proveedores

Como se comentaba anteriormente, ADO.NET es un mecanismo de acceso a datos extensible, pudiendo crearse nuevos proveedores que faciliten el trabajo con otros orígenes de datos.

El núcleo de ADO.NET, por tanto, debe estar preparado para operar con proveedores que no conoce de antemano. Esto es posible gracias a la existencia de unas interfaces genéricas, predefinidas en el ámbito `System.Data`, que todo proveedor de datos, ya sea directa o indirectamente, debe implementar.

Las interfaces de `System.Data` pueden dividirse en dos grupos: aquellas que aplicables a cualquier origen de datos, sea o no una base de datos relacional, y las específicas para bases de datos relacionales. Las primeras inician su nombre, como es habitual en todas las interfaces, con el prefijo `I`, mientras que las segundas emplean el prefijo `IDb`.

Como se verá en los puntos siguientes, las del segundo grupo suelen estar derivadas de las del primero, contando con algunas implementaciones por defecto alojadas como clases en el ámbito `System.Data.Common`.

Asociación de columnas y tablas

Al crear un conjunto de datos local, en el ordenador en el que se ejecuta la aplicación cliente, es preciso establecer una correspondencia entre las columnas del origen de datos y las existentes en dicho conjunto local. Esas dependencias se representan mediante la interfaz `IColumnMapping`, implementada en la clase `ColumnMapping` y todas las implementaciones específicas derivadas de ésta.

`IColumnMapping` es una interfaz compuesta tan sólo de dos propiedades públicas: `DataSetColumn` y `SourceColumn`. La primera contiene el nombre de la columna en el `DataSet`, el conjunto de datos local, y la segunda el nombre que recibe en el origen de datos. Cualquier clase que implemente `IColumnMapping` está obligada a implementar estas dos propiedades. Para facilitar dicha operación, en el ámbito `System.Data.Common` encontramos la clase `DataColumnMapping`, que se limita a implementar las dos propiedades y ofrecer un constructor. Cualquier proveedor puede utilizar esta clase como base para crear las suyas propias en caso de que fuese necesario, si bien una asociación entre columnas de origen y locales no requiere un trabajo adicional.

Dado que un conjunto de datos suele estar formado por múltiples columnas, por lógica tienen que existir varios `DataColumnMapping`. La interfaz `IColumnMappingCollection`, implementada en la clase `DataColumnMappingCollection`, actúa como colección de enlaces entre columnas origen y locales.

En la figura 5.1 puede ver la relación existente entre las interfaces y clases indicadas, así como sus miembros más destacables, en este caso las propiedades `DataSetColumn` y `SourceColumn`.

Las columnas no existen en un `DataSet` de manera aislada, sino que se agrupan en tablas representadas por objetos `DataTable`. Es preciso, por lo tanto, establecer también una correspondencia entre las tablas del origen de datos y las que existirán en el `DataSet`, tarea de la que se encarga la interfaz `ITableMapping`. Ésta cuenta tan sólo con tres propiedades:

- `SourceTable`: Referencia a la tabla del origen de datos.
- `DataSetTable`: Referencia a la tabla que le corresponde en el `DataSet`.
- `ColumnMappings`: Colección con las asociaciones de columnas de esta tabla.

En el ámbito `System.Data.Common` existe una implementación de la interfaz `ITableMapping` en la clase `DataTableMapping`. De manera análoga a lo que ocurre con las columnas, también existe una interfaz `ITableMappingCollection` con su correspondiente implementación en `DataTableCollection`.

Acceso a filas de datos

Los orígenes de datos de tipo RDBMS son capaces de facilitar un cursor, un motor software que el cliente puede utilizar, en el caso de ADO.NET, para acceder a

filas de datos de manera unidireccional y con el único propósito de leer datos, nunca de modificarlos. Cada proveedor facilitará una clase cuyo objetivo es facilitar la comunicación con el cursor del servidor, clase que se conoce habitualmente como *Reader* o bien lector de datos. Dichas clases deben implementar las interfaces *IDataRecord* e *IDataReader*.

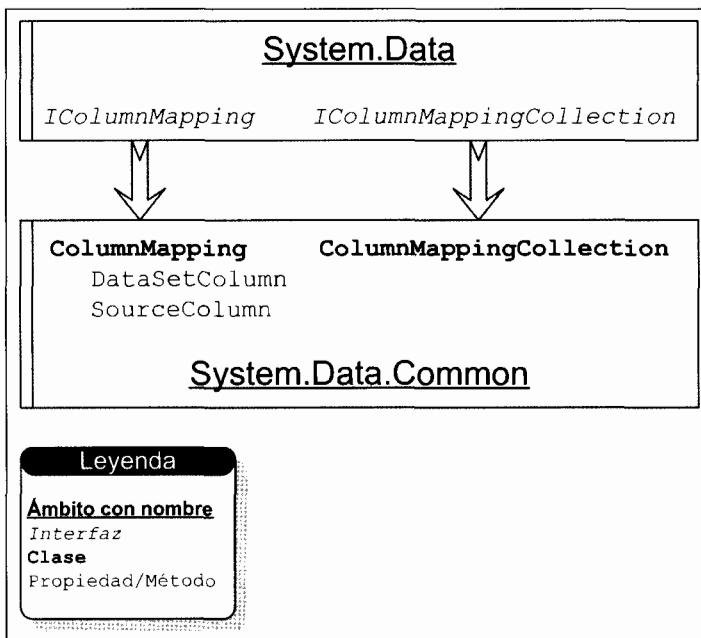


Figura 5.1. Elementos relacionados con la asociación de columnas entre conjuntos de datos locales y orígenes de datos

La interfaz *IDataRecord* define las propiedades y métodos necesarios para acceder a los datos alojados en las columnas de una fila de datos. Podemos saber el número de columnas existentes mediante la propiedad *FieldCount*, accediendo individualmente a cada una de ellas con la propiedad *Item*. Esta interfaz cuenta, asimismo, con más de una veintena de métodos del tipo *GetBoolean()*, *GetByte()*, *GetFloat()* y *GetString()*, cuyo objetivo es recuperar el valor de una determinada columna en el formato especificado.

Para poder operar sobre las columnas de una fila de datos, finalidad de *IDataRecord*, antes es necesario recuperar esa fila, así como tener la posibilidad de avanzar a la siguiente hasta llegar al final de las filas facilitadas.

Éste es el objetivo de los miembros de la interfaz *IDataReader*, entre los cuales están los siguientes:

- *RecordsAffected*: Propiedad que contiene el número de filas existentes.
- *Read()*: Método que lee la siguiente fila.

- `NextResult()`: Accede al siguiente conjunto de resultados en caso de existir varios.
- `Close()`: Cierra el lector.

Como ya sabe, las interfaces no pueden ser utilizadas directamente desde un programa propio para operar sobre los datos, siendo necesario emplear clases que las implementen. Estas dos interfaces se encuentran implementadas en las clases `DataReader` de cada proveedor, por ejemplo `SqlDataReader`, `OleDbDataReader`, `OracleDataReader` y `OdbcDataReader`. Cada una de ellas facilita una implementación específica de los miembros citados, añadiendo al tiempo otros exclusivos en caso de ser necesario. `SqlDataReader`, por ejemplo, cuenta con una serie de métodos `GetSqlByte()`, `GetSqlString()`, `GetSqlDouble()`, etc., que facilitan la recuperación de columnas con tipos de datos exclusivos de SQL Server, mientras que el proveedor de Oracle dispone de métodos tipo `GetOracleBinary()`, `GetOracleNumber()` y `GetOracleString()`, ajustados a los tipos de este RDBMS.

La figura 5.2 representa la relación entre las interfaces `IDataRecord` e `IDataReader`, definidas en el ámbito `System.Data`, y las clases `DataReader` de cada proveedor. Conociendo los miembros de las interfaces podremos trabajar con cualquier proveedor, sin tener que conocer su implementación específica.

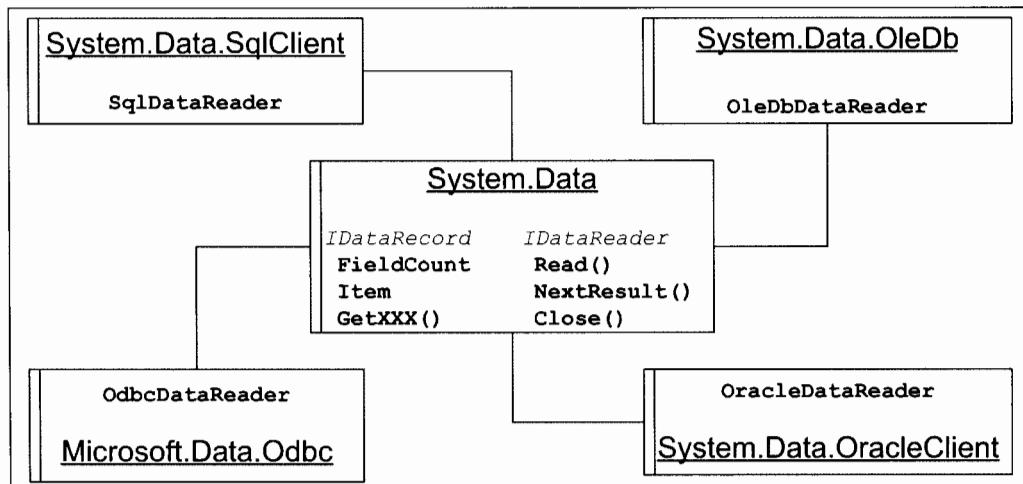


Figura 5.2. Cada proveedor de datos implementa una clase `DataReader` que implementa los miembros de las interfaces `IDataRecord` e `IDataReader`

En el octavo capítulo aprenderá, en la práctica, a usar las clases `DataReader` de los proveedores para acceder a un origen de datos y recuperar información.

Adaptadores de datos

Como se explicó en el capítulo anterior, ADO.NET no emplea cursores de servidor para tratar la información si exceptuamos los cursores unidireccionales sólo de lectura representados por las clases `DataReader`. En su lugar, siempre que es precisa una navegación más compleja o la edición de los datos, lo que se hace es ejecutar un comando y obtener un conjunto de datos local o `DataSet`. Para efectuar esa operación se necesita un adaptador de datos.

Un adaptador de datos es el nexo de unión entre un origen de datos y un `DataSet`, el eslabón que se encarga de adaptar la información de las dependencias propias del origen al carácter independiente del conjunto de datos local. Como sería de esperar, cada proveedor de datos cuenta con su propio adaptador específico, si bien todos ellos implementan una interfaz común: `IDataAdapter`.

Los miembros de la interfaz `IDataAdapter` tienen dos objetivos: recuperar filas del origen de datos convirtiéndolas en filas del `DataSet`, por una parte, y generar los comandos necesarios para enviar las modificaciones del `DataSet` de vuelta al origen de datos. Entre sus propiedades y métodos más importantes están:

- `Fill()`: Es el método encargado de recuperar filas del origen de datos incluyéndolas en el `DataSet` local.
- `Update()`: Complementario al anterior, efectuaría las operaciones necesarias en el origen de datos para transferir las acciones de edición que haya sufrido el `DataSet`.

`IDataAdapter` es una interfaz genérica, aplicable tanto a orígenes de tipo RDBMS como a los que no lo son. Existe una interfaz derivada, `IDbDataAdapter`, específica para los orígenes que son bases de datos relacionales. `IDbDataAdapter` añade a los miembros de su interfaz base cuatro propiedades: `SelectCommand`, `InsertCommand`, `UpdateCommand` y `DeleteCommand`, cuya finalidad es alojar las sentencias de selección, inserción, actualización y borrado de filas en SQL.

La interfaz `IDataAdapter` no es implementada directamente por clases específicas de los proveedores de datos, sino por las clases `DataAdapter` y `DbDataAdapter` que existen en el ámbito `System.Data.Common`. De esta forma se ofrece una implementación por defecto, genérica, que después cada proveedor hereda y puede personalizar según precise. `DataAdapter` implementa la interfaz `IDataAdapter`, mientras que `DbDataAdapter` es una clase derivada de `DataAdapter` y, por tanto, hereda la implementación de `IDataAdapter`.

Derivadas de `DbDataAdapter`, e implementando la interfaz `IDbDataAdapter`, encontramos una clase en cada proveedor: `SqlDataAdapter`, `OleDbDataAdapter`, `OracleDataAdapter` y `OdbcDataAdapter`. Estas clases cuentan con la implementación por defecto de los métodos `Fill` y `Update` facilitada por `DataAdapter` y `DbAdapter`, al tiempo que ofrecen una implementación específica de las propiedades `SelectCommand`, `InsertCommand`, `UpdateCommand` y `DeleteCommand` que, presumiblemente, diferirán de un proveedor a otro.

En la figura 5.3 se ha representado la relación existente entre las interfaces `IDataAdapter` e `IDbDataAdapter` respecto a las clases `DataAdapter`, `DbDataAdapter` y `SqlDataAdapter`, perteneciente al proveedor `SqlClient`. Ésta podría ser sustituida por `OleDbDataAdapter`, `OracleDbTypeAdapter` o bien `OdbcDataAdapter` sin cambiar más que el ámbito de `System.Data.SqlClient` al que corresponda.

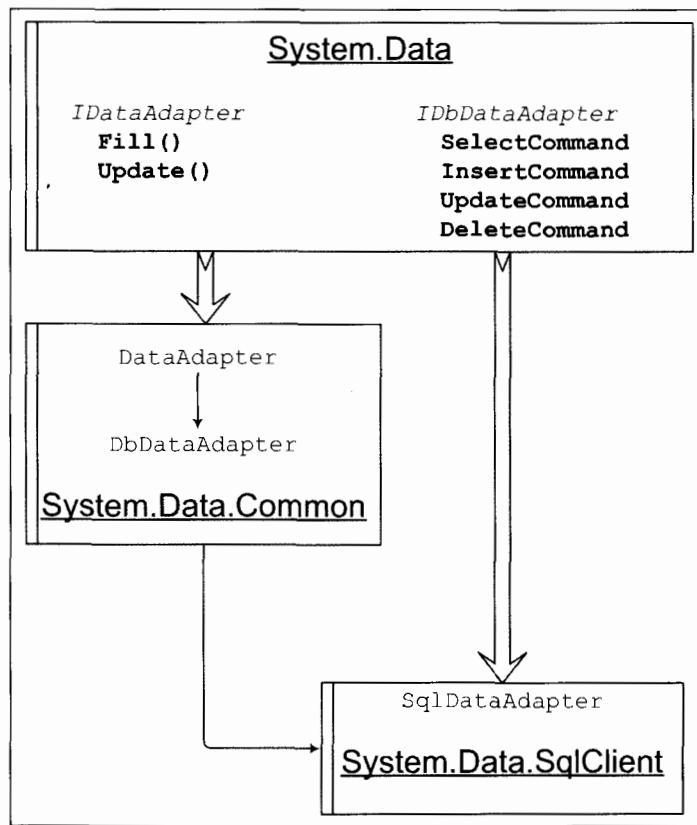


Figura 5.3. La interfaz `IDataAdapter` es implementada por la clase `DataAdapter`, base de `DbDataAdapter` que, a su vez, es base de `SqlDataAdapter`. Esta última implementa la interfaz `IDbDataAdapter`

Conexiones, comandos y transacciones

Los proveedores de datos que acceden a orígenes de tipo RDBMS o bases de datos de escritorio, la mayor parte de ellos, deben facilitar operaciones adicionales como es la conexión con la base de datos, la definición de comandos y control de las transacciones. Estas operaciones son específicas y diferentes en cada proveedor, pero con el fin de facilitar, si es que desea, una codificación independiente del

proveedor, existen las interfaces `IDbConnection`, `IDbCommand` e `IDbTransaction`. Como puede suponer, la primera contiene los miembros relacionados con la conexión a la base de datos, la segunda tiene que ver con los comandos y la tercera con las transacciones.

De la interfaz `IDbConnection` habría que destacar los miembros `ConnectionString`, `Open()`, `Close()`, `CreateCommand()` y `BeginTransaction()`, que contiene la cadena de conexión a la base de datos, utiliza esa cadena para abrir la conexión, cierra la conexión, crea un nuevo comando asociado a esta conexión e inicia una transacción, respectivamente.

En el punto anterior se indicó que la interfaz `IDbDataAdapter` contaba con cuatro propiedades: `SelectCommand`, `InsertCommand`, `UpdateCommand` y `DeleteCommand`. Todas ellas son referencias a la interfaz `IDbCommand`, en la que se definen los miembros que debe tener un comando que va a ejecutarse sobre una base de datos.

Las propiedades `CommandType` y `CommandText` de la interfaz establecen el tipo de comando y el texto, que puede ser el nombre de una tabla, una sentencia SQL, el nombre de una vista o procedimiento almacenado, etc., dependiendo del valor dado a `CommandType`.

Un comando siempre se obtiene a partir de una conexión. La relación entre ésta y el comando la fija la propiedad `Connection` de `IDbCommand`, conteniendo una referencia de tipo `IDbConnection`. Finalmente, `IDbCommand` cuenta con una serie de métodos cuya función es ejecutar el comando, ya sea obteniendo un lector de datos o sin recuperar resultado alguno.

También las transacciones están asociadas a una conexión de base de datos, por ello la interfaz `IDbTransaction` tiene una propiedad `Connection`, como `IDbCommand`. Por lo demás, tan sólo hay tres miembros en esta interfaz: `IsolationLevel`, que determina el nivel de aislamiento de la transacción; `Commit()`, que la confirma, y `Rollback()`, que la descarta.

Ninguna de estas tres interfaces cuenta con una implementación por defecto en el ámbito `System.Data.Common`, a diferencia de la mayoría de las enunciadas en los puntos previos. Son los proveedores, directamente, los responsables de implementarlas. En cada uno de ellos debe existir una clase `XXXConnection`, una `XXXCommand` y una `XXXTransaction`, donde `XXX` sería el prefijo del proveedor: `Sql`, `OleDb`, `Oracle` u `Odbc`.

La figura 5.4 resume algunos de los miembros de las tres interfaces, así como las clases que las implementan en los ámbitos de cada uno de los proveedores.

Detalles sobre los proveedores

Tras leer los puntos anteriores, ya se habrá hecho una cierta idea sobre la composición de un proveedor de datos .NET y su funcionamiento. Cada proveedor se compone de una serie de clases que implementan determinadas interfaces definidas en `System.Data`, ya sea directamente, como ocurre con las tratadas en el

punto anterior a éste, o indirectamente, al heredar la implementación de las clases que encontramos en `System.Data.Common`.

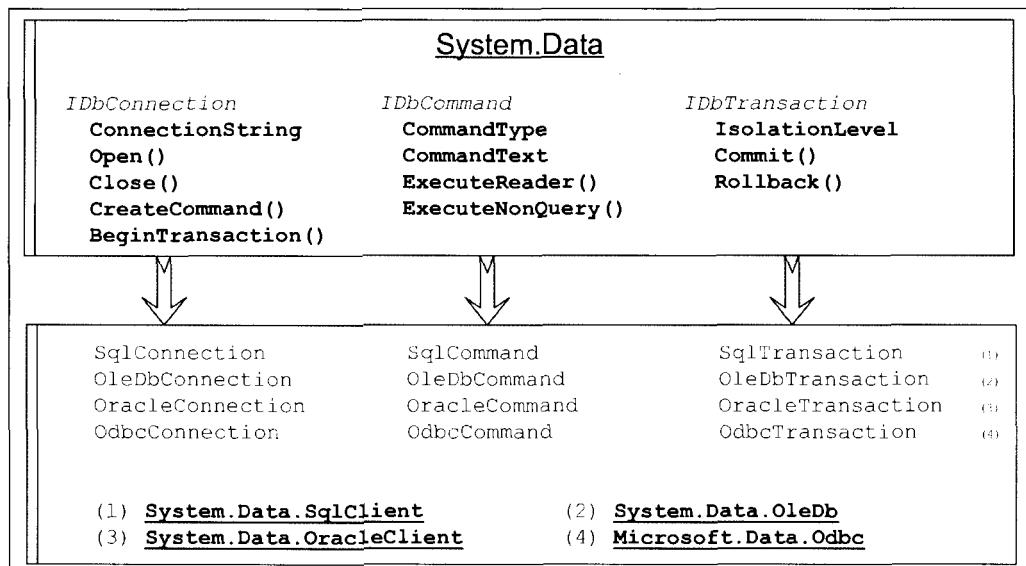


Figura 5.4. Cada proveedor cuenta con implementaciones específicas de las interfaces `IDbConnection`, `IDbCommand` e `IDbTransaction`

Todo proveedor de datos debe constar, como mínimo, de las clases necesarias para:

- Establecer una conexión con el origen de datos.
- Definir y ejecutar comandos.
- Leer de un cursor unidireccional directamente desde el origen de datos.
- Crear un `DataSet` a partir de un comando.

Estas clases se denominan siempre `Connection`, `Command`, `DataReader` y `DataAdapter`, respectivamente, precedidas de la denominación del proveedor: `Sql` en el de SQL Server, `OleDb` en el de OLE DB, `Oracle` en el proveedor de Oracle y `Odbc` en el proveedor para ODBC. Conociendo los miembros de las interfaces `IDbConnection`, `IDbCommand`, `IDataReader` e `IDataAdapter`, tratadas en los apartados previos, podremos usar las clases específicas de cualquier proveedor.

Otras clases comunes y específicas

A parte de las ya mencionadas, en el ámbito `System.Data.Common` existen otras clases de uso común entre los proveedores. Todas ellas sirven como clases

base de otras, personalizadas para cada proveedor de datos concreto. Estas clases son `DbDataPermission`, `DBDataPermissionAttribute`, `RowUpdatedEventArgs` y `RowUpdatingEventArgs`.

Otros métodos, a pesar de no contar con una interfaz genérica ni derivar de una clase común definida en `System.Data.Common`, también existen en la mayoría de proveedores, aunque su implementación no es obligatoria. Las clases `SqlCommandBuilder`, `OleDbCommandBuilder`, `OracleCommandBuilder` y `OdbcCommandBuilder`, cada una de ellas perteneciente a los cuatro proveedores que ya conoce, básicamente son idénticas de cara al programador aunque, como es lógico, internamente cada una de ellas funciona de manera distinta. Su finalidad es generar las sentencias necesarias para insertar, actualizar y eliminar información del origen de datos, para lo cual disponen de los métodos `GetInsertCommand()`, `GetUpdateCommand()` y `GetDeleteCommand()`. Lo que se obtiene es un comando de inserción, actualización o borrado específico para el proveedor en el que estemos trabajando, de tal forma que una misma llamada, por ejemplo a `GetInsertCommand()`, generaría una sentencia distinta según que el origen de datos sea SQL Server, Oracle o un controlador OLE DB u ODBC.

Algo similar ocurre con las clases `Parameter`, `ParameterCollection` y `Error`, si bien esta última no existe en el proveedor de Oracle y, en su lugar encontramos la clase `OracleException`.

Por último, cada proveedor aporta sus clases exclusivas, inexistentes en los demás proveedores, enfocadas a aprovechar las características propias de un cierto origen de datos. Su uso, no obstante, resulta mucho menos habitual puesto que, en cierta medida, limita o hace más difícil una posible transición futura a otro producto de almacenamiento de datos.

Clases independientes del origen de datos

Hasta ahora nos hemos centrado en el estudio de los elementos relacionados con los proveedores de datos ya que, al fin y al cabo, ellos son los que permiten a las aplicaciones acceder a la información, recuperarla, editarla y volver a almacenarla. Una vez que esa información se encuentra en nuestra aplicación, sin embargo, para operar sobre ella utilizaremos un conjunto de clases que son totalmente independientes del origen de datos.

No existe, por ejemplo, una clase `DataSet` para SQL Server, otra para Oracle, una tercera para OLE DB y una más para ODBC, sino que tenemos una única clase `DataSet`.

De este conjunto de clases prácticamente la única que hemos mencionado ha sido `DataSet`, lo cual es lógico, ya que es posiblemente la más importante de las existentes en `System.Data`. La clase `DataSet` es relativamente compleja, puesto que un conjunto de datos está formado por una o más tablas que, a su vez, se componen de columnas. Entre esas tablas, además, pueden existir relaciones, y cada una de las columnas puede tener activas ciertas restricciones. Cada uno de esos

elementos, la columna, tabla, relación o restricción, se representa con una clase disponible en el ámbito System.Data.

Ya que estas clases son de uso final, no tendremos que conocer después implementaciones específicas por proveedor, en los puntos siguientes se abordan con algo más de profundidad, a fin de que las conozcamos lo suficiente como para poder usarlas en los ejemplos de posteriores capítulos. Su uso en la práctica, no obstante, se dejará para más adelante.

Conjuntos de datos

Como se indica en la propia ayuda de Visual Studio .NET, el componente DataSet es un elemento fundamental de ADO.NET, una clase en torno a la que existen muchas otras de las que depende. Un DataSet puede crearse mediante código o, lo que resulta mucho más corriente, obtenerse a partir de un origen de datos mediante el correspondiente adaptador de datos. En el noveno capítulo conocerá con detalle la forma de trabajar con la clase DataSet y otras relacionadas con ella, por ahora nos limitaremos a mencionar sus miembros más relevantes. Éstos son:

- **DataSetName:** Cada conjunto de datos cuenta con un nombre que le identifica, de igual forma que una tabla, vista o procedimiento almacenado tiene un identificador único. Este nombre normalmente se establece en el momento de la creación del objeto DataSet, facilitándolo como parámetro al constructor, mientras que esta propiedad permite tanto obtenerlo como cambiarlo.
- **Tables:** En el capítulo dedicado a SQL aprendió a componer sentencias SQL en las que se relacionaban varias tablas con el fin de obtener un conjunto de datos combinado. El conjunto de datos, por tanto, puede estar compuesto de varias tablas. La propiedad Tables es una referencia a una colección DataTableCollection en la que se encuentran todas las tablas que forman el DataSet.

Nota

Mediante la propiedad Tables es posible acceder a las tablas de datos, cada una de las cuales está representada por un objeto DataTable. Las operaciones se efectúan en las filas y columnas de cada tabla, afectando al conjunto de datos.

- **Relations:** Si el conjunto de datos cuenta con varias tablas, entre ellas necesariamente existirán relaciones. Esta propiedad es una referencia a un objeto DataRelationCollection que aloja todas esas referencias.
- **HasChanges():** Devuelve True en caso de que haya algún cambio en el conjunto de datos, ya sea de inserción, modificación o eliminación. Estos

cambios se almacenan provisionalmente separados de los valores del conjunto de datos original, permitiéndose posteriormente su aceptación o rechazo.

- `AcceptChanges ()`: Provoca una llamada en cascada al método homónimo de cada una de las tablas referenciadas en la propiedad `Tables` que, a su vez, llaman al mismo método de cada una de las filas, propagando así la aceptación de todos los cambios que hubiese pendientes en el conjunto de datos.
- `RejectChanges ()`: Su funcionamiento es similar al del método anterior, pero en este caso deshaciendo los cambios en lugar de aceptándolos.
- `WriteXml () / ReadXml ()`: Guardan y recuperan el conjunto de datos localmente en formato XML, ya sea en un archivo o flujo de datos previamente preparado.
- `WriteXmlSchema () / ReadXmlSchema ()`: Similares a los anteriores, pero afectando tan sólo al esquema XML de los datos, no a los datos en sí.

La mayoría de estos métodos cuentan con múltiples versiones sobrecargadas. En la documentación electrónica de Visual Studio .NET encontrará los detalles relativos a los parámetros que acepta cada una de esas versiones y su comportamiento.

Tablas

Un conjunto de datos o `DataSet` consta, básicamente, de tablas y relaciones. Cada una de las tablas, a las que se tiene acceso mediante la propiedad `Tables` según acaba de verse, está representada por un objeto de la clase `DataTable`. Ésta cuenta con los miembros necesarios para poder acceder a las filas de datos, saber qué columnas componen cada fila, qué restricciones existen, etc. También dispone de métodos para aceptar y rechazar cambios, añadir nuevas filas, recuperar aquellas que cumplen un cierto criterio, etc.

Cada tabla se identifica con un nombre que se almacena en la propiedad `TableName` de la clase `DataTable`. La relación entre una tabla y su conjunto de datos queda reflejada en la propiedad `DataSet` del `DataTable`. Algunas de las propiedades más relevantes de `DataTable` son:

- `Rows`: Da acceso a la colección de filas de datos que componen la tabla, representada cada una de ellas por un objeto de la clase `DataRow`.
- `Columns`: Colección de todas las columnas de la tabla. Cada columna se representa con un objeto de tipo `DataColumn`.
- `Constraints`: Referencia a un objeto `ConstraintCollection` que contiene todas las restricciones de esta tabla.
- `PrimaryKey`: Identifica la columna o columnas que actúan como clave primaria de la tabla.

- `ChildRelations/ParentRelations`: Cada una de ellas contiene una referencia a un objeto `DataRelationCollection`, en el primer caso con las relaciones hijo de la tabla y en el segundo con las relaciones padre. Esta información puede utilizarse para, por ejemplo, obtener todas las filas hija de una dada en una relación maestro/detalle entre dos tablas.

Mediante la propiedad `Rows` se tiene acceso a las filas de datos actuales, pudiéndose modificar su contenido o eliminarse. Si queremos añadir filas se emplea el método `NewRow()` del propio objeto `DataTable`. Los cambios efectuados pueden aceptarse, con `AcceptChanges()`, o descartarse, llamando a `RejectChanges()`. Utilizando el método `Select()` es posible filtrar las filas de la tabla, obteniendo sólo aquellas que se ajustan a un cierto criterio.

Al igual que otras muchas clases, `DataTable` dispone de múltiples eventos. Con ellos se notifica el cambio del contenido de una columna, el cambio en una fila o la eliminación de una fila. Todos estos eventos tienen una forma en gerundio y otra en pasado, por ejemplo `RowDeleting` y `RowDeleted`, produciéndose el primero en el momento en que va a efectuarse la acción y el segundo una vez que se ha completado.

Filas

Las filas de datos de una tabla, tal y como se ha indicado en el punto anterior, se representan mediante objetos `DataRow` alojados en la colección `Rows`. Como en toda colección, es posible añadir nuevos elementos, acceder a los existentes y eliminarlos. Una vez tengamos una referencia a un `DataRow`, a una fila de datos propiamente dicha, utilizaríamos los miembros de dicha clase para manipular su contenido.

Cada fila se encontrará, respecto al estado original de la colección de filas a la que pertenece, en un determinado estado. Éste se aloja en la propiedad sólo de lectura `RowState`, indicando si se ha modificado, permanece sin cambios, ha sido añadida o eliminada. Las operaciones básicas que podemos llevar a cabo sobre la fila, y que provocan el cambio de su propiedad `RowState`, son las enumeradas a continuación:

- Añadir una fila. Tomando como base la propiedad `Rows` del `DataTable`, que contiene una referencia a una colección, no tenemos más que usar el habitual método `Add()` de cualquier colección para añadir una nueva fila.
- Eliminar una fila. Basta con llamar a su método `Delete()`.
- Modificación del contenido de una fila. La clase cuenta con una propiedad llamada `Item`, que es la propiedad por defecto, mediante la cual es posible acceder a cada columna de la fila, ya sea mediante su nombre o con el índice de posición que ocupa en la fila. Este acceso permite tanto leer el valor actual como cambiarlo.

A medida que se modifican los valores de las columnas de una fila, se irán ejecutando automáticamente todas las comprobaciones y restricciones que haya definidas en el conjunto de datos. Esto, en ocasiones, puede no ser lo más adecuado, especialmente si van a hacerse muchos cambios que tienen restricciones y, además, dichos cambios tienen interdependencias que pudieran causar estados inválidos. De ser así, antes de iniciar las modificaciones deberíamos llamar al método `BeginEdit()` del `DataRow`, inhibiendo tanto las comprobaciones como la generación de los eventos de cambio. Cuando hayamos terminado de manipular la fila, confirmaremos todos los cambios con una llamada a `EndEdit()`. Será en ese momento cuando se ejecuten todas las restricciones y produzcan los eventos. También puede utilizarse el método `CancelEdit()` para devolver la fila a su situación inicial.

Las restricciones y tipos de datos de las columnas pueden dar lugar a que, tras una modificación, una o más columnas queden en estado de error. La clase `DataRow` dispone de la propiedad `HasErrors`, que nos permite saber si existen o no errores, y el método `GetColumnsInError()`, que devuelve un arreglo con todas las columnas que tienen errores, pudiendo obtenerse la descripción del error.

Nota

A medida que van editándose, las filas de una tabla pueden contar con múltiples versiones de valor que almacenan, por ejemplo el valor por defecto, el que tenía anterior al cambio y el posterior. El método `HasVersion()` permite saber si un cierto `DataRow` tiene o no múltiples versiones del valor, en caso afirmativo puede accederse a ellos utilizando los valores de la enumeración `DataRowVersion`.

Columnas

Cada una de las columnas existentes en un `DataRow` cuenta con una serie de atributos que definen, por ejemplo, el tipo de información que puede contener, el valor por defecto si es que existe, si los valores que contendrá deben ser únicos o no, etc. Todos estos atributos son accesibles mediante los miembros de la clase `DataColumn`.

De esta clase nos interesarán básicamente una docena de sus propiedades, ya que el resto de los miembros son heredados de las clases ascendientes y no tienen que ver directamente con el trabajo con datos.

Algunas de esas propiedades son:

- `ColumnName`: Nombre de la columna.
- `DataType`: Tipo de dato de la columna.
- `MaxLength`: Longitud máxima si la columna contiene una cadena de texto.

- `DefaultValue`: Valor por defecto para la columna.
- `AllowDBNull`: Indica si puede dejarse un valor nulo en la columna.
- `Unique`: Determina si los valores de la columna deben ser únicos entre todas las filas de la tabla.
- `AutoIncrement`: Especifica si la columna contiene un valor que se incrementará automáticamente.

Como puede ver, los miembros de `DataRow` definen las características a los que tendrán que ajustarse los valores de una cierta columna. Dichos valores, no obstante, se almacenan en las filas, es decir, en objetos `DataRow`.

Restricciones

Una de las propiedades de `DataTable`, llamada `Constraints`, es la encargada de contener la colección de restricciones a aplicar a la tabla, estando definida cada una de ellas mediante los miembros de una clase derivada de `Constraint`. Ésta, clase abstracta en la que se define la propiedad `ConstraintName` que albergará el nombre de la restricción, es base de las clases `ForeignKeyConstraint` y `UniqueConstraint`, que representan los dos tipos de restricción más habituales.

La clase `UniqueConstraint`, añadida a la colección `Constraints` de la tabla, asegura que los valores que tiene una cierta columna sean siempre únicos en todas las filas. Crear un objeto `UniqueConstraints` y añadirlo a la citada colección es exactamente igual que dar el valor `True` a la propiedad `Unique` del `DataColumn` que representa a la columna cuyos valores tienen que ser únicos.

Las restricciones de clave externa, correspondientes a la clase `ForeignKeyConstraint`, son más complejas. Al crearse relacionan una columna de una tabla con una columna de otra tabla, impidiendo la inconsistencia de sus valores. Además, esta clase dispone de dos propiedades, `UpdateRule` y `DeleteRule`, que establece qué debe hacerse cuando se modifique o elimine, respectivamente, el valor de una columna en la tabla maestra que tienen dependencias en la tabla de detalle.

Nota

Las restricciones existentes en un conjunto de datos, y que se definen al nivel de cada tabla, no se comprueban a menos que la propiedad `EnforceConstraints` de la clase `DataSet` sea `True`, que es su valor por defecto.

Relaciones

Según se indicaba anteriormente, si en un `DataSet` existen múltiples `DataTable` es preciso definir la relación que guardan entre ellos. Con este fin se utiliza la clase

DataRelation, creándose un objeto a partir de ella por cada relación que exista en el conjunto de datos. Una vez se ha establecido una relación entre las tablas, se evitará cualquier actuación sobre ellas que pudiera infringirla, garantizando la integridad de la información.

Las propiedades de DataRelation suelen establecerse en el momento de la creación del objeto, facilitando al constructor los parámetros apropiados. Para ello se entrega el nombre de la tabla maestra y la columna o columnas que actúan como clave principal, así como el nombre de la tabla de detalle y el de la columna o columnas que actuarían como clave externa. Estos valores también pueden ser establecidos con las propiedades ParentTable, ParentColumns, ChildTable y ChildColumns, respectivamente.

Al crear una relación ésta genera implícitamente dos restricciones que se aplican a las tablas participantes en dicha relación. Por una parte, se aplica una restricción UniqueConstraint a la columna indicada de la tabla maestra, a fin de que no pueda repetirse su valor ya que, de ocurrir esto, tendríamos un conflicto porque no se sabría a qué fila de la tabla maestra corresponden las de la tabla de detalle. Por otra parte, también se crea una restricción ForeignKeyConstraint aplicada a la columna de la tabla de detalle, impidiendo así que se introduzcan valores inexistentes en la tabla maestra.

En la figura 5.5 puede ver un esquema de bloques en el que se representa un DataSet con dos DataTable, relacionados mediante un DataRelation, conteniendo cada uno de ellos sus DataRow, DataColumn y Constraint, accesibles mediante las propiedades comentadas en todos los puntos anteriores.

Vistas de datos

Un objeto DataTable facilita en su propiedad Rows todas las filas que contiene la tabla del origen de datos al que está asociado, ofreciendo una vista por defecto. No obstante, partiendo de ese objeto DataTable es posible crear vistas de datos más elaborados gracias a la clase DataView. Mediante un objeto DataView las operaciones de búsqueda, ordenación y manipulación de los datos resultan más sencillas, ya que dispone de las propiedades y métodos apropiadas para ello.

Los criterios de filtrado y el orden pueden fijarse durante la creación del DataView o bien posteriormente, sirviéndose de las propiedades RowFilter, RowStateFilter y Sort. La primera contendrá una cadena de caracteres con las condiciones de filtrado de filas, la segunda aplica un filtro basado en el estado de las filas y la tercera determina las columnas sobre la base de las cuales se ordenarán las filas de la tabla.

Teniendo las filas ordenadas, operaciones como la búsqueda resultan mucho más fáciles. Usando los métodos Find() y FindRows() de DataView es posible encontrar la fila o filas que coinciden con el valor clave que se indique, buscándolo en la columna por la que está ordenándose. También pueden usarse los métodos AddNew() y Delete() para añadir y eliminar filas, así como editar los valores de las existentes accediendo a ellas mediante la propiedad Item.

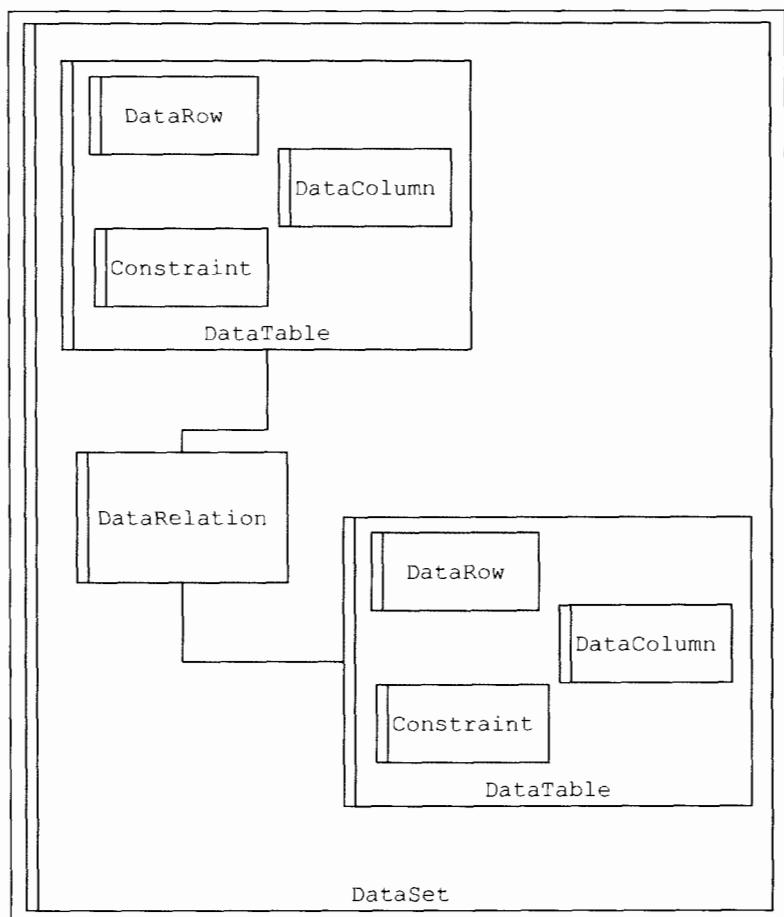


Figura 5.5. Un conjunto de datos formado por dos tablas relacionadas entre sí, compuesta cada una de ellas de filas, columnas y restricciones

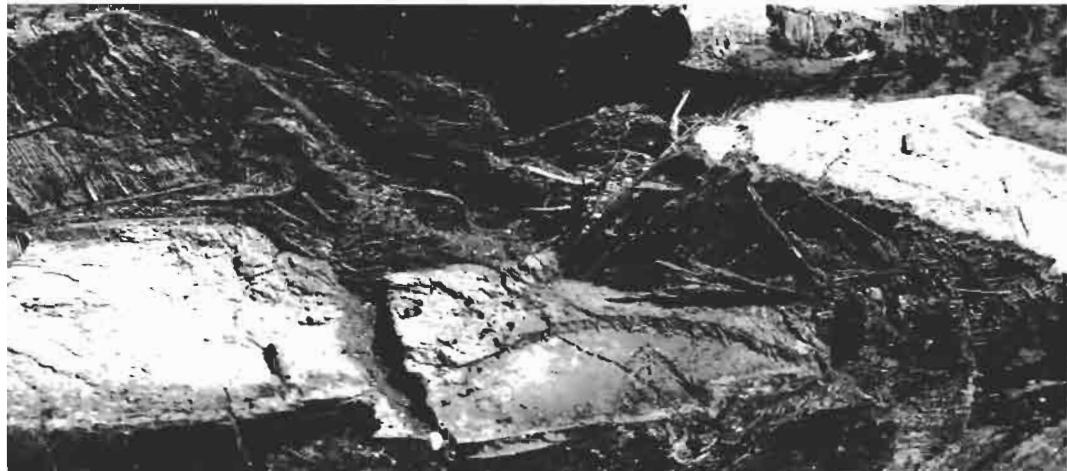
Nada impide crear múltiples objetos **DataGridView** sobre el mismo **DataTable**, obteniéndose diferentes vistas de los mismos datos según el orden y criterios de selección que interesen.

Resumen

Al finalizar la lectura de este capítulo posiblemente se sienta confundido o abrumado, lo que no es extraño teniendo en cuenta la cantidad de interfaces, clases, propiedades y métodos que se han introducido de forma rápida y exclusivamente teórica. Esto no debe preocuparle. No se trata de que sepa qué clases hay, en qué ámbito se encuentra o qué miembros cuentan, ya que toda esa información

la puede encontrar rápidamente en la referencia de la ayuda de Visual Studio .NET. El objetivo del capítulo era familiarizarle con todos los elementos mencionados, a fin de que le resulten familiares y tenga una idea aproximada de cuál es su función y relación con los demás.

Este conocimiento, que irá afianzando a medida que siga leyendo, nos servirá para, en capítulos siguientes, profundizar en el estudio de cada interfaz o clase en particular, poniéndolas ya en práctica mediante el desarrollo de algunos ejemplos simples pero demostrativos.



6

Conexión al origen de datos

Los dos capítulos previos, fundamentalmente teóricos, seguramente le habrán resultado arduos de leer, pero la teoría es necesaria, al menos en una cierta proporción, para poder abordar con un mínimo de familiaridad el desarrollo de los proyectos que van a proponerse a partir de ahora, comenzando con este capítulo.

Nuestro objetivo, en este sexto capítulo, es conocer todo lo necesario para poder conectar con distintos orígenes de datos, poniendo ese conocimiento en práctica al escribir pequeños programas que abran y cierren dicha conexión. Aprenderá a usar las diferentes clases `Connection`, una por proveedor, y también la sintaxis de las cadenas de conexión de ciertos proveedores, como los de OLE DB y ODBC.

Obtención e instalación de proveedores adicionales

Tras instalar Visual Studio .NET tenemos a nuestra disposición dos proveedores de datos: `SqlClient` y `OleDb`. El primero es específico para SQL Server, mientras que con el segundo es posible emplear cualquier controlador OLE DB instalado en el sistema, tema sobre el que volveremos después. Acceder a un cierto origen de datos mediante el proveedor `OleDb` y un controlador OLE DB supone, como es fácil deducir, una capa más de código, lo cual implica mayor uso de recursos y menor rendimiento. Por ello, siempre que sea posible es preferible el uso de proveedores de datos nativos y específicos para cada origen.

Tras la presentación de Visual Studio .NET Microsoft ha liberado dos proveedores más: OracleClient y Odbc, para operar directamente sobre bases de datos Oracle y usar controladores ODBC, respectivamente. A continuación se indica dónde puede obtener estos controladores y cómo instalarlos en su sistema.

Dónde obtener los proveedores

El primer paso será localizar los proveedores adicionales que haya publicado Microsoft, en este momento los que acaban de indicarse, y transferirlos a nuestro ordenador. Vaya a la sede *MSDN Library* de Microsoft introduciendo este URL <http://msdn.microsoft.com/library/> en su cliente Web habitual. Despliegue la sección del menú que aparece a la izquierda el apartado **Downloads** y seleccione la opción **Developer Downloads**. En el panel de la izquierda habrá aparecido una lista jerárquica con distintos nodos. Abra el nodo **.NET Framework** y allí encontrará los proveedores, por ejemplo en **ODBC .NET Data Provider** y **Microsoft .NET Data Provider for Oracle**.

Al seleccionar uno de los nodos, aparecerá un documento facilitando diversa información y un enlace para obtener el archivo de instalación del proveedor. En la figura 6.1, por ejemplo, puede ver el documento del proveedor para Oracle, con el enlace en la parte superior derecha. Obtenga el archivo y repita el paso con el proveedor ODBC.

Nota

El proveedor de ADO.NET para Oracle ocupa 1,3 megabytes, mientras que el proveedor ODBC no llega al megabyte. El archivo de instalación del primero es `oracle_net.msi` y el del segundo `odbc_net.msi`.

Instalación del proveedor

Microsoft facilita los proveedores de datos en archivos MSI autoinstalables, por lo que el proceso es realmente sencillo. Localice el archivo que contiene el proveedor, por ejemplo `oracle_net.msi`, y haga doble clic sobre él. La instalación se pondrá en marcha, mostrándose un asistente compuesto de varias páginas en las que, básicamente, puede limitarse a ir pulsando el botón **Next >** hasta llegar al último paso (véase figura 6.2) donde haríamos clic sobre **Install**.

Durante la instalación se añadirá al menos un nuevo ensamblado al GAC, conteniendo el código del proveedor, así como un nuevo grupo a la carpeta **Programas** con la documentación exclusiva del proveedor instalado.

A partir de este momento ya podemos usar el nuevo proveedor, algo queharemos en los puntos posteriores utilizando los dos indicados antes y que se asume que habrá obtenido e instalado en su sistema.

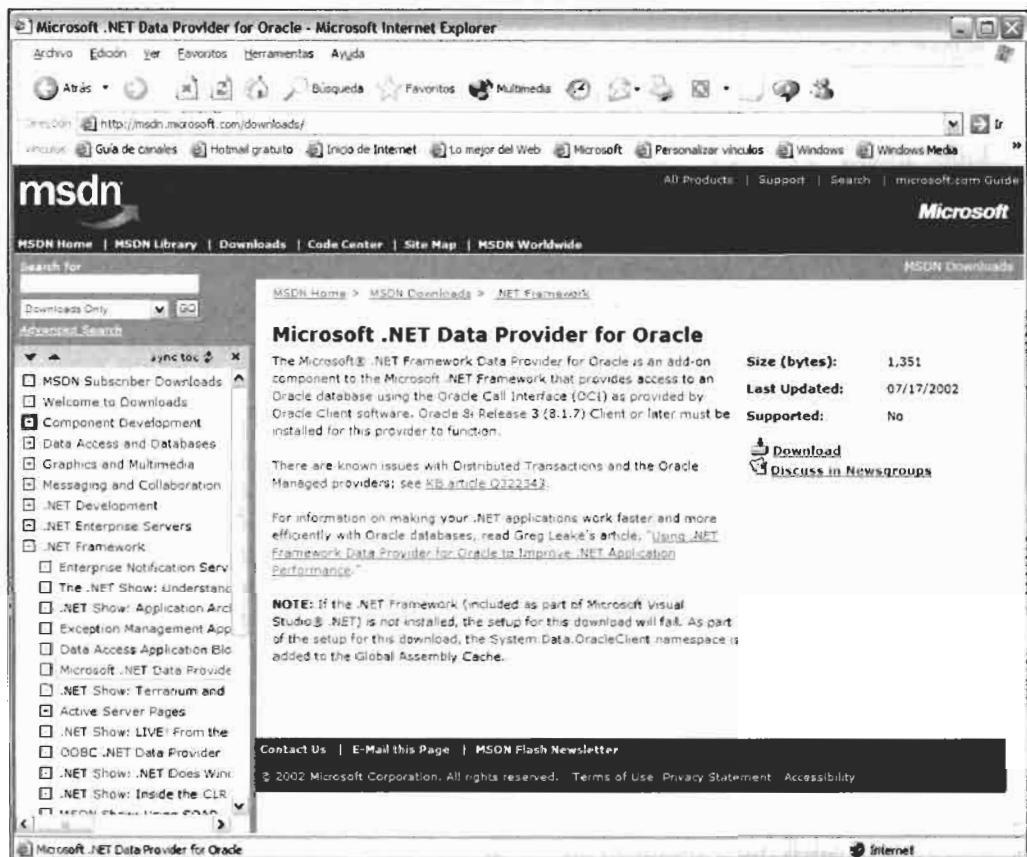


Figura 6.1. Obtenemos los proveedores de datos .NET adicionales



Figura 6.2. Instalación del proveedor ADO.NET para Oracle

Generalidades sobre la conexión

Tal como ya se indicara en el capítulo anterior, la mayoría de las clases implementadas por los proveedores tienen una clase base común o bien implementan una misma interfaz. En este caso se da lo segundo, y la interfaz que determina la lista mínima de miembros con que deben contar las clases de conexión es `IDbConnection`. La relación entre esta interfaz y el conjunto de clases que la implementan es la mostrada esquemáticamente en la figura 6.3.

Aunque, como ya sabe, no es posible crear objetos a partir de una interfaz, nada nos impide almacenar la referencia devuelta por el constructor de una de las clases `Connection` en una variable de tipo `IDbConnection`. Esto nos permitirá codificar de manera genérica los pasos de la conexión, conociendo tan sólo los miembros de esta interfaz.

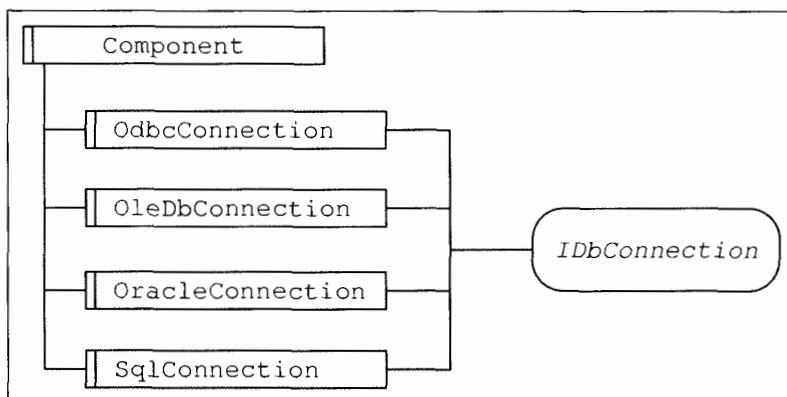


Figura 6.3. Las clases `Connection` de los distintos proveedores implementan la interfaz `IDbConnection`

Cadena de conexión

Una vez hayamos obtenido una referencia a un objeto `Connection`, el primer paso será establecer la cadena de conexión asignándola a la propiedad `ConnectionString`. También es posible facilitarla en el momento de la creación del objeto, entregándola como parámetro al constructor. En cualquier caso, esta cadena determinará el servidor con el que se comunicará el proveedor, la base de datos o archivo a abrir, los parámetros de seguridad a emplear, etc.

La estructura de la cadena de conexión depende directamente del proveedor que vayamos a utilizar y, en el ámbito de ciertos proveedores, como es el caso de `OleDb` y `Odbc`, también del controlador con el que vaya a conectarse. En puntos posteriores, en los que vamos a conectar con diversos orígenes de datos, entraremos en más detalles sobre la composición de la cadena de conexión.

Nota

Existe una sede Web, llamada CONNECTIONSTRINGS, en la que puede encontrar información rápida sobre las cadenas de conexión para acceder a los orígenes de datos más usuales mediante OLE DB y ODBC. El URL de dicha sede es <http://www.connectionstrings.com>.

Apertura y cierre de la conexión

Tras preparar la cadena de conexión, y justo antes de efectuar cualquier otra operación sobre el origen de datos, es necesario llamar al método `Open()`. En ese momento se utilizarán los parámetros de `ConnectionString` para comunicarse con el origen de datos y abrir la conexión, quedando a partir de ese momento disponible para la ejecución de comandos. Debe tener en cuenta que esta llamada es necesaria, la conexión no se establecerá automáticamente al intentar ejecutar el primero de los comandos sobre la conexión. Mantener una conexión con el origen de datos significa estar consumiendo una serie de recursos que, en ocasiones, podrían liberarse si dicha conexión va a permanecer inactiva la mayor parte del tiempo. Por regla general, es posible llamar al método `Open()` para abrirla, ejecutar los comandos necesarios y, a continuación, cerrarla con el método `Close()`.

Las llamadas a los métodos `Open()` y `Close()`, aparte de otras operaciones, afectan al estado de la conexión, provocando una transición de un estado a otro. El estado actual de la conexión podemos conocerlo consultando la propiedad `ConnectionState`, cuyos posibles valores son:

- `Closed`: La conexión está cerrada.
- `Connecting`: Se está conectando con el origen de datos.
- `Open`: La conexión está abierta.
- `Executing`: Se está ejecutando un comando.
- `Fetching`: Se están recuperando datos desde el origen.
- `Broken`: Se ha perdido la conexión tras haber llamado satisfactoriamente a `Open()`.

En realidad los dos únicos estados que podemos encontrar actualmente en la propiedad `ConnectionState` son `Open` y `Closed`, ya que el resto, aunque definidos, no tienen uso en esta versión de la biblioteca de clases .NET.

Nota

Tras abrir la conexión, con el comando `Open()`, se conecta con una base de datos o archivo que dependerá de los parámetros facilitados en la cadena de

conexión. Puede cambiar de una base de datos a otra, sin necesidad de cerrar la conexión y volver a abrirla, mediante el método `ChangeDatabase()`.

Propiedades informativas

A parte de `ConnectionString` y `State`, en la interfaz `IDbConnection` existen dos más cuyo único objetivo es facilitar el nombre de la base de datos que está utilizándose, en el caso de `Database`, y el tiempo máximo de espera antes de cancelar un intento de conexión, en `ConnectionTimeout`. Éstos son datos que, normalmente, se facilitan en la cadena de conexión, aunque la base de datos también puede especificarse mediante el método `ChangeDatabase`. Estas dos propiedades son sólo de lectura.

Los otros dos miembros de la interfaz `IDbConnection`, `CreateCommand` y `BeginTransaction`, no nos interesan en este momento ya que no tienen que ver con el establecimiento de la conexión propiamente dicha.

Cadenas de conexión

En este momento conocemos el procedimiento que habríamos de seguir para conectar con un origen de datos, utilizando para ello el objeto `Connection` de cualquiera de los cuatro proveedores mencionados, pero aún no sabemos cuál es la sintaxis que, en cada caso, debería seguir la cadena de conexión, algo totalmente imprescindible.

Una cadena de conexión se compone de pares de propiedades y valores, en el formato `propiedad=valor`. Los distintos pares de la cadena se separan entre sí utilizando puntos y coma. Las propiedades posibles dependen de cada proveedor, aunque existen ciertas coincidencias entre ellos que, hasta cierto punto, simplificarán nuestro trabajo. Veamos, caso por caso, cómo compondríamos una cadena de conexión.

Selección del controlador

Los proveedores `SqlClient` y `OracleClient` son específicos de un RDBMS, proveedores que se comunican utilizando directamente el API o conjunto de funciones del software cliente de SQL Server y Oracle, respectivamente. `OleDb` y `Odbc`, por el contrario, son proveedores mucho más genéricos, pensados para aprovechar todos los controladores OLE DB y ODBC que hay actualmente disponibles y que, en la práctica, facilitan el acceso a cualquier origen de datos.

Al usar uno de estos dos proveedores, es necesario indicar en la cadena de conexión qué controlador es el que va a utilizarse, algo innecesario con `SqlClient` y `OracleClient` porque, como se ha dicho, son específicos. En el caso de OLE DB

el controlador se asigna a la propiedad `Provider`, mientras que con ODBC la propiedad se llama `Driver`.

El nombre de los controladores OLE DB suele estar compuesto de varias partes, indicando el nombre del fabricante, identificación del controlador, versión, etc. El nombre del controlador OLE DB para acceder a una base de datos Access, por ejemplo, es `Microsoft.Jet.OLEDB.4.0`, el de Oracle `OraOLEDB.Oracle`, etc. También existen controladores con nombres abreviados, por ejemplo `sqloledb` para identificar al controlador OLE DB para SQL Server o `msdaora` para identificar al de Oracle.

Nota

Debe tener en cuenta que la identificación de los controladores OLE DB diferirá según el fabricante que los facilite. El controlador para Oracle ofrecido por Microsoft se llama `msdaora`, mientras que el ofrecido por la propia empresa Oracle se identifica como `OraOLEDB.Oracle`. Según el que tengamos instalado en el sistema, será necesario usar un nombre u otro.

En la tabla 6.1 se enumeran los proveedores OLE DB que pueden encontrarse habitualmente en cualquier sistema Windows, indicándose el nombre con el que se conoce al proveedor y el identificador que habría que utilizar con la propiedad `Provider` en la cadena de conexión.

Tabla 6.1. Proveedores OLE DB

Nombre del proveedor	Identificador
<i>Microsoft Data Shaping Service for OLE DB</i>	<code>MSDataShape</code>
<i>Microsoft OLE DB Persistence Provider</i>	<code>MSPersist</code>
<i>Microsoft OLE DB Provider for Internet Publishing</i>	<code>MSDAIPP.DSO</code>
<i>Microsoft OLE DB Provider for Microsoft Active Directory Service</i>	<code>ADSDSOObject</code>
<i>Microsoft OLE DB Provider for Microsoft Indexing Service</i>	<code>MSIDX</code>
<i>Microsoft OLE DB Provider for Microsoft Jet</i>	<code>Microsoft.Jet.OLEDB.4.0</code>
<i>Microsoft OLE DB Provider for ODBC</i>	<code>MSDASQL</code>
<i>Microsoft OLE DB Provider for Oracle</i>	<code>MSDAORA</code>
<i>Microsoft OLE DB Provider for SQL Server</i>	<code>SQLOLEDB</code>
<i>Microsoft OLE DB Simple Provider</i>	<code>MSDAOSP</code>

Los nombres de los controladores ODBC suelen ser mucho más descriptivos que los anteriores. Para utilizar el controlador para Access u Oracle, por continuar con los mismos ejemplos propuestos, usaríamos los identificadores {Microsoft Access Driver (*.mdb)} y {Microsoft ODBC for Oracle}, respectivamente. Observe que se han utilizado unas llaves para delimitar a los identificadores, llaves que hay que incluir en la cadena de conexión.

A continuación, suponiendo que ConnOleDb tiene una referencia a un objeto OleDbConnection, puede ver varios ejemplos de selección de controlador. Tenga en cuenta que estas cadenas de conexión no son válidas, ya que además del controlador son precisos parámetros adicionales antes de poder llamar a Open() para establecer la conexión.

```
' Controlador OLE DB para SQL Server  
ConnOleDb.ConnectionString = "Provider=sqloledb ...  
  
' Controlador OLE DB para Access/Excel  
ConnOleDb.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0 ...  
  
' Controlador OLE DB de Microsoft para Oracle  
ConnOleDb.ConnectionString = "Provider=msdaora ...
```

Si en lugar de un objeto OleDbConnection hubiésemos creado un OdbcConnection, para servirnos de un controlador ODBC, el formato de la cadena de conexión sería similar al mostrado en los ejemplos siguientes:

```
' Controlador ODBC para SQL Server  
ConnOdbc.ConnectionString = "Driver={SQL Server} ...  
  
' Controlador ODBC para Access  
ConnOdbc.ConnectionString =  
    "Driver={Microsoft Access Driver (*.mdb)} ...  
  
' Controlador ODBC de Microsoft para Oracle  
ConnOdbc.ConnectionString = "Driver={Microsoft ODBC for Oracle} ...
```

Nota

Además de los controladores OLE DB y ODBC que se instalan por defecto con Windows y ciertos productos, como Visual Studio .NET, existen otros, desarrollados por terceros fabricantes, para facilitar el acceso a sus productos. En:

http://ourworld.compuserve.com/homepages/Ken_North/oledbVen.htm

puede encontrar una lista de productos relacionados con OLE DB, principalmente controladores, y en:

http://ourworld.compuserve.com/homepages/Ken_North/ODBCVend.HTM

una lista similar relacionada con ODBC.

Identificación del servidor u origen de datos

El siguiente paso, el primero si vamos a usar `SqlClient` u `OracleClient`, es identificar el origen de datos que, en ocasiones, será un determinado servidor en el que se encuentra ejecutándose el RDBMS o bien un nombre de archivo. El nombre de la propiedad es `Data Source` o `Server`, dependiendo del proveedor. La primera existe en `OleDb`, `SqlClient` y `OracleClient`, mientras que la segunda aparece en `OdBC` y también `SqlClient` y `OracleClient`.

Si la cadena de conexión que estamos estableciendo es de un objeto `SqlClient` u `OracleClient`, la propiedad `Data Source` contendrá el nombre de la máquina en que está ejecutándose el servidor de datos. En mi caso, por poner un ejemplo, el ordenador en el que está instalado Visual Studio .NET, y desde el que se ejecutan las aplicaciones, es un equipo llamado `Dimension`, mientras que SQL Server está ejecutándose en otro con el nombre `Inspirón`. La cadena de conexión de un objeto `SqlConnection` sería la siguiente:

```
Data Source=inspiron
```

Lo mismo valdría para el proveedor OLE DB, anteponiendo `Provider=sqloledb`, o el proveedor `OracleClient`. En el caso del proveedor ODBC, tan sólo hay que cambiar `Data Source` por `Server`.

En caso de usar los proveedores OLE DB u ODBC para acceder a un origen de datos no RDBMS, como puede ser una base de datos Access o una hoja de cálculo Excel, el dato a facilitar es el camino y nombre completos del archivo. A causa de ello, el parámetro `Server` del proveedor ODBC se sustituirá por `Dbq`, por ejemplo:

```
ConnOdbc.ConnectionString =  
    "Driver={Microsoft Access Driver (*.mdb)};Dbq=\camino\archive.mdb"
```

Base de datos inicial

Para acceder a una base de datos Access o una hoja de cálculo Excel basta con conocer el nombre del archivo en que se encuentra, pero al trabajar con un RDBMS, como Oracle o SQL Server, no basta con conocer el nombre del ordenador en el que está ejecutándose el servidor. Además, es preciso que indiquemos sobre cuál de las bases de datos existentes en él deseamos trabajar.

Los proveedores `OleDb` y `SqlClient` admiten en la cadena de conexión la propiedad `Initial Catalog`, a la que se asignaría el nombre de la base de datos. Si utilizamos el proveedor `OdBC`, la propiedad a usar es `Database`.

Parámetros de seguridad

En ocasiones, para poder conectar con un determinado origen de datos es preciso identificarse adecuadamente ya sea directa o indirectamente. La seguridad es

un aspecto especialmente importante en los sistemas RDBMS, depósitos de información de las empresas y que deben estar resguardados de acciones, accidentales o intencionadas, y la revelación de datos confidenciales a terceros.

Como acaba de decirse, la identificación ante el origen de datos puede ser directa o indirecta. En el primer caso se entregan de manera explícita un nombre de usuario y una clave, mientras que en el segundo se deja que sea el sistema, en el que previamente hemos iniciado sesión, el que presente nuestras credenciales ante el servidor de datos. La propiedad `Integrated Security` determinará qué método va a utilizarse. Presente en tres de los cuatro proveedores, hay que exceptuar `Odbc`, esta propiedad puede tomar los valores `yes` o `no`. En el primer caso se utilizará la seguridad integrada del sistema para identificarnos, mientras que en el segundo habrá que entregar un nombre de usuario y clave.

Nota

Al usar el proveedor `SqlClient`, o bien un controlador SQL Server mediante OLE DB, es habitual la asignación a la propiedad `Integrated Security` del valor `SSPI`, equivalente a `yes`.

El nombre de usuario o esquema y la clave se facilitan siempre en las mismas propiedades, `User ID` y `Password`, en todos los proveedores con excepción, una vez más, de `Odbc`, en el que dichas propiedades se llaman `UID` y `PWD`. Con esto, una cadena de conexión completa podría quedar como alguna de las siguientes:

```
' Controlador OLE DB para SQL Server
ConnOleDb.ConnectionString =_
"Provider=sqloledb; Integrated Security=SSPI;" &_
"Data Source=inspiron; Initial Catalog=Libros"

' Proveedor SqlClient
ConnSql.ConnectionString =_
"Integrated Security=SSPI; Data Source=inspiron"

' Proveedor Oracle
ConnOracle.ConnectionString =_
"Data Source=ek400; User ID=scott; Password=tiger"

' Controlador ODBC para Access
ConnOdbc.ConnectionString =_
"Driver={Microsoft Access Driver (*.mdb)};" &_
"DBQ=C:\PBddVisualBasicNET\Ejemplos\Cap_03\Libros.mdb"
```

Nota

El valor asignado a la propiedad `User ID` en la cadena de conexión del proveedor `OracleClient` determina el esquema inicial de trabajo al conectar con la base de datos.

Propiedades exclusivas

Además de todas las anteriores, que son las de uso más habitual y general entre proveedores y controladores, también existen propiedades que son exclusivas de un cierto proveedor o controlador. Estas propiedades se introducen en la cadena de conexión siguiendo exactamente el mismo procedimiento, especificando el nombre de la propiedad y el valor separados por un signo =. Para poder conocer las propiedades exclusivas de cada proveedor o controlador tendrá que consultar su documentación.

En la práctica

Al final del punto anterior, tras revisar los distintos parámetros que pueden formar parte de una cadena de conexión, se han propuesto varias asignaciones distintas para la propiedad `ConnectionString` de varios objetos `Connection`. Sólo con esto ya estaríamos en disposición de conectar con el origen de datos, que es lo que perseguimos, llamando al método `Open()` inmediatamente después de la asignación.

Si la cadena de conexión tiene el formato correcto y, además, la configuración de red, el estado del servidor y otros parámetros permiten conectar con el origen de datos, la propiedad `State` debería cambiar de `Closed` a `Open`. Cualquier incidencia que impida la conexión provocará una excepción, por ejemplo `ArgumentException` si hay un parámetro incorrecto en la cadena de conexión.

En los puntos siguientes va a tratarse, paso a paso, el proceso para conectar con algunos de los orígenes de datos creados en un capítulo previo, comprobando esa conexión y cerrándola finalmente. El resultado no será muy espectacular, pero es el primer paso en el tratamiento de datos con ADO.NET.

Conexión con Microsoft Access

Vamos a comenzar por uno de los casos más simples, escribiendo un pequeño ejemplo en Visual Basic .NET que conecte con la base de datos Access que habíamos creado en el tercer capítulo. Para ello no es necesario que en el ordenador donde está la base de datos, y va a ejecutarse este ejemplo, se encuentre instalado Microsoft Access, basta con la instalación de los MDAC.

Por simplicidad, estos primeros ejemplos van a crearse como aplicaciones de consola. Simplemente genere una aplicación de consola vacía, utilizando el asistente **Aplicación de consola**, y a continuación introduzca el código indicado que, en este caso, sería el siguiente:

```
' Necesitamos las clases del proveedor OLE DB
Imports System.Data.OleDb
```

```

Module Module1

Sub Main()
    ' Creamos el objeto Connection
    Dim ConnOleDb As New OleDbConnection()

    ' y establecemos la cadena de conexión
    ConnOleDb.ConnectionString =
        "Provider=Microsoft.Jet.OLEDB.4.0; " &
        "Data Source=\PBddVisualBasicNET\Cap_03\Libros.mdb"

    Try ' Intentamos abrir la conexión
        ConnOleDb.Open()
        ' Si está abierta
        If ConnOleDb.State = ConnectionState.Open Then
            ' hemos tomado éxito, así que mostramos algunos datos
            Console.WriteLine("Se ha establecido la conexión")
            Console.WriteLine("ConnectionString=" & _
                ConnOleDb.ConnectionString & "")
            Console.WriteLine("ConnectionTimeout=" & _
                ConnOleDb.ConnectionTimeout)
            Console.WriteLine("Database=" & ConnOleDb.Database)

            ConnOleDb.Close() ' y la cerramos
        Else ' si no está abierta, lo indicamos
            Console.WriteLine("La conexión no está abierta")
        End If
    Catch ' si se produce un error lo indicamos
        Console.WriteLine("*** Error al intentar la conexión ***")
    End Try

    End Sub

End Module

```

Nota

Los proyectos de ejemplo de éste y los demás capítulos puede encontrarlos en las distintas subcarpetas de la carpeta Ejemplos que hay en el CD-ROM que acompaña al libro.

Tras crear un objeto OleDbConnection, observe cómo se establece la cadena de conexión asignándola a la propiedad `ConnectionString`. Habría conseguido exactamente el mismo resultado facilitando esa cadena como parámetro al constructor de `OleDbConnection`.

Fíjese en cómo se indica el camino en el que está el archivo, así como el nombre completo de éste, incluyendo la extensión. Lógicamente, si en su sistema el archivo está en otro punto deberá modificar el valor de `Data Source`.

El resto del código está incluido en un bloque `Try/Catch`, recogiendo así cualquier excepción que pudiera generarse al llamar al método `Open()`. Dependiendo

del valor de State, o de que se produzca o no esa excepción, veremos salir un mensaje indicativo u otro por pantalla. Si todo va bien, como debería, el resultado tendría que ser el mostrado en la figura 6.4. Cualquier otro mensaje significará que algo no ha ido bien.

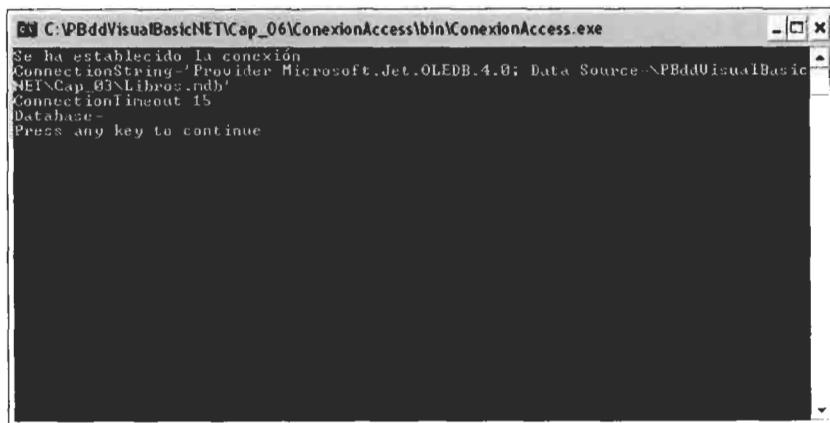


Figura 6.4. Conseguimos conectar sin problemas con una base de datos Access

Conexión con Microsoft Excel

El mismo controlador OLE DB que hemos utilizado para conectar con Microsoft Access, Microsoft.Jet.OLEDB.4.0, nos servirá también para acceder a cualquier hoja de cálculo Excel. Lo único que tenemos que hacer es cambiar la cadena de conexión, facilitando el camino y nombre del libro.

Además, tendremos que usar el parámetro Extended Properties, utilizado para enviar al controlador parámetros exclusivos, indicando que lo que va a abrirse es una hoja de Excel y, opcionalmente, si la primera fila de cada hoja contiene títulos de columnas o no.

Tomando como base el código anterior, cambie la asignación a la propiedad ConnectionString para que quede como se muestra a continuación:

```
Conn OleDb.ConnectionString =  
    "Provider=Microsoft.Jet.OLEDB.4.0;" &  
    "Data Source=\PBddVisualBasicNET\Cap_03\Libros.xls;" &  
    "Extended Properties='Excel 8.0; HDR=Yes'"
```

Observe cómo las propiedades extendidas, que en este caso indican que el archivo es un libro de Excel que cuenta con una cabecera de títulos, se delimitan entre comillas simples. De no hacerlo así, el parámetro HDR=Yes pasaría al proveedor OLE DB de ADO.NET, que intentaría interpretarlo y, al no reconocerlo, provocaría un error. Con el entrecomillado conseguimos que ese parámetro, junto con Excel 8.0, pase directamente al controlador OLE DB de Microsoft Jet.

Al ejecutar el proyecto, tras efectuar las modificaciones indicadas, el resultado debería ser como el de la figura 6.5. La única diferencia, como puede verse, es el valor de ConnectionString.

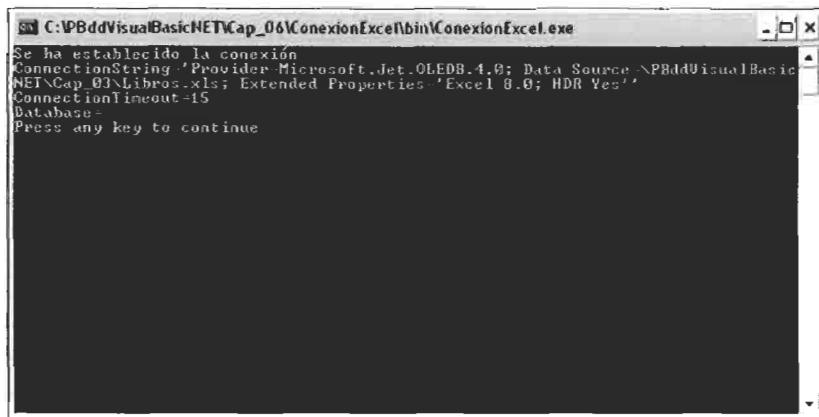


Figura 6.5. Conectamos con un libro de Microsoft Excel

Nota

Para acceder mediante el controlador OLE DB a una hoja de cálculo Excel no es necesario tener instalado Microsoft Excel en el equipo.

Conexión con SQL Server

Tanto Access como Excel son orígenes de datos a los que se accede localmente, sin necesidad de utilizar un software cliente para, a través de protocolos de red, comunicarse con un servidor que se encargue de efectuar físicamente la manipulación de los datos. De ahí que la conexión sea bastante sencilla, como acaba de verse en los dos puntos previos.

Para poder conectar con nuestra base de datos SQL Server deben darse los siguientes requisitos:

- El servidor SQL Server, ya sea 7, 2000 o MSDE, debe estar en funcionamiento, bien en el mismo ordenador donde va a ejecutarse la aplicación o en un ordenador remoto al que se tenga acceso mediante una infraestructura de red.
- En el equipo donde va a ejecutarse la aplicación debe haberse instalado el software cliente de SQL Server. Esto no es necesario si es el mismo equipo donde está ejecutándose el RDBMS o bien se trata de un ordenador donde se ha instalado la plataforma .NET y la última versión de los MDAC, que será nuestro caso.

- Hay que conocer el nombre del servidor donde está ejecutándose el RDBMS, así como disponer de una cuenta, nombre de usuario y clave, para poder conectar con él en caso de que no vaya a utilizarse la seguridad integrada.

En caso de que vayamos a ejecutar el ejemplo siguiente en el mismo equipo donde está ejecutándose el servidor, podemos cambiar el valor de Data Source por localhost. En mi caso SQL Server se ejecuta en una máquina distinta, de ahí que sea preciso indicar su nombre. Además, va a utilizarse la seguridad integrada de Windows, empleando nuestra cuenta de acceso al sistema cliente para identificarnos ante el servidor y poder acceder al RDBMS.

Nota:

Si tiene problemas con el acceso al servidor a causa de la configuración de seguridad, facilite explícitamente un nombre de usuario y clave o, en su defecto, consulte con su administrador de red para que configure su cuenta de tal manera que pueda acceder a SQL Server.

Asumiendo que se cumplen todos los requisitos previos, y que hemos comprobado que es posible el acceso desde el cliente al servidor utilizando el software cliente de SQL Server, los pasos para conectar con la base de datos, partiendo del código del punto anterior, serían los indicados a continuación:

- Sustitución de System.Data.OleDb por System.Data.SqlClient en la instrucción Imports, al inicio del módulo de código.
- Modificación de la declaración de la variable de conexión, que quedaría así:

```
Dim ConnSql As New SqlConnection(_  
    "Data Source=inspiron; Integrated Security=SSPI; " & _  
    "Initial Catalog=Libros")
```

- Observe que la cadena de conexión se facilita directamente al constructor de SqlConnection, en lugar de con una asignación posterior a ConnectionString.
- Cambio de las referencias a la variable ConnOleDb por ConnSql en las líneas siguientes del programa.

Llevados a cabo estos cambios, la ejecución del programa debería producir un resultado similar al de la figura 6.6, en el que puede verse la cadena de conexión y el nombre de la base de datos con la que se ha conectado. Si obtiene un error, comience por comprobar que es posible conectar con la base de datos Libros desde el propio servidor, utilizando la herramienta de administración usada en el tercer capítulo. A continuación verifique que hay comunicación entre el cliente, donde está ejecutándose la aplicación, y el servidor y, finalmente, compruebe la configuración de seguridad.

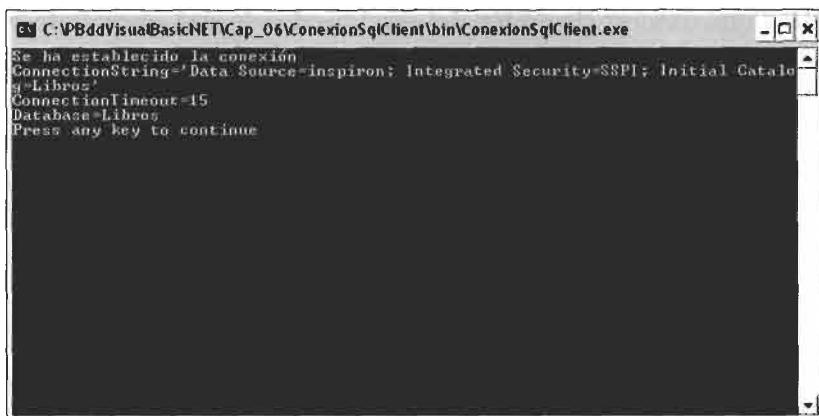


Figura 6.6. Establecemos conexión con un servidor SQL Server

Conexión con InterBase

InterBase, y su hermano Firebird, son sistemas RDBMS poco conocidos si se les compara con Oracle, SQL Server, DB2 o MySQL, si exceptuamos a la comunidad de usuarios de herramientas de desarrollo Borland, relativamente grande. A diferencia de lo que ocurre con SQL Server u Oracle, actualmente no existe un proveedor de datos ADO.NET para InterBase. El controlador Microsoft Jet, que hemos usado anteriormente mediante el proveedor OLE DB para acceder a Access y Excel, tampoco es capaz de conectar directamente con InterBase, así que deberemos buscar una alternativa que pasa, necesariamente, por obtener un controlador OLE DB u ODBC de terceros, ya que no se facilita ninguno con el propio RDBMS.

En Internet es posible encontrar varios controladores ODBC para InterBase/Firebird, tanto de uso libre como comerciales, algunos de los cuales cuentan con ediciones de prueba durante un tiempo limitado. En la Web *InterBase Installation Info - FAQ*, cuyo URL es <http://ibinstall.defined.net/faq.htm>, puede encontrar un apartado dedicado, precisamente, a los controladores ODBC para InterBase. Algunos de ellos, como el de IBPhoenix, pueden utilizarse gratuitamente, mientras que otros, como el Gemini o EasySoft, son comerciales. Para los ejemplos de este libro se ha empleado la versión de evaluación del controlador de Gemini, concretamente la versión 2.1, que puede obtener desde <http://www.ibdatabase.com/> en un archivo auto instalable de tan sólo 330 kilobytes.

Nota

También tiene la alternativa de usar un controlador OLE DB en lugar de ODBC. El proceso sería, básicamente, el mismo: localizar el controlador, hay varios de tipo shareware y comerciales, e instalarlo. En este punto se asume que empleará el controlador ODBC indicado, pero si prefiere usar un controlador OLE

DB tendrá que modificar el código para acceder a él mediante el proveedor OleDb en lugar de Odbc.

Por lo demás, la configuración para poder conectar con nuestra base de datos InterBase, creada en el tercer capítulo, no difiere de la explicada en el punto anterior para SQL Server, siendo preciso tener el RDBMS en funcionamiento, ya sea en el equipo local en un servidor, el software cliente instalado en el ordenador donde vaya a utilizarse la aplicación y la configuración correcta de red para comunicar cliente con servidor.

Nota

El ensamblado Microsoft.Data.Odbc.dll no se añade automáticamente al crear un proyecto Visual Basic, como sí ocurre con el que contiene los anteriores proveedores de datos, así que tendrá que abrir el menú emergente del proyecto, seleccionar la opción Agregar referencia y añadir una referencia a dicho ensamblado para que el código siguiente funcione.

Los cambios a efectuar en el código serían los siguientes:

- Importar el ámbito Microsoft.Data.Odbc, donde se encuentra el proveedor ADO.NET para ODBC.
- Dejar la declaración de la variable de conexión como se muestra aquí:

```
Dim ConnOdbc As New OdbcConnection( _  
    "Driver={Gemini InterBase ODBC Driver 2.0}; " & _  
    "Server=localhost; " & _  
    "Database=\PBddVisualBasicNET\Cap_03\Libros.gdb;" & _  
    "UID=SYSDBA; PWD=masterkey")
```

- Sustituir las referencias a la variable de conexión introduciendo ConnOdbc en lugar de la empleada en el ejemplo anterior.

En este caso, en la configuración utilizada para desarrollar este ejemplo, InterBase 6.5 está ejecutándose en el mismo ordenador donde se ejecuta el programa, de ahí que se haya asignado el valor localhost al parámetro Server. Fíjese en que se utiliza la cuenta SYSDBA con la clave de acceso que tiene por defecto. Deberá cambiar esto si modificó la clase o desea usar una cuenta distinta para conectar.

Asumiendo que la configuración es la correcta, el programa debería producir una respuesta como la mostrada en la figura 6.7.

Conexión con Oracle 8i

A pesar de que la facilidad de uso del RDBMS Oracle ha ido mejorando de versión a versión, hasta llegar a las actuales 8i y 9i, lo cierto es que resulta un producto

bastante más complejo que cualquiera de los tratados en los puntos previos. El proceso de conexión, desde el código, no difiere del utilizado en los ejemplos anteriores, pero, para que funcione, es precisa una preparación preliminar configurando los servicios en el software cliente.

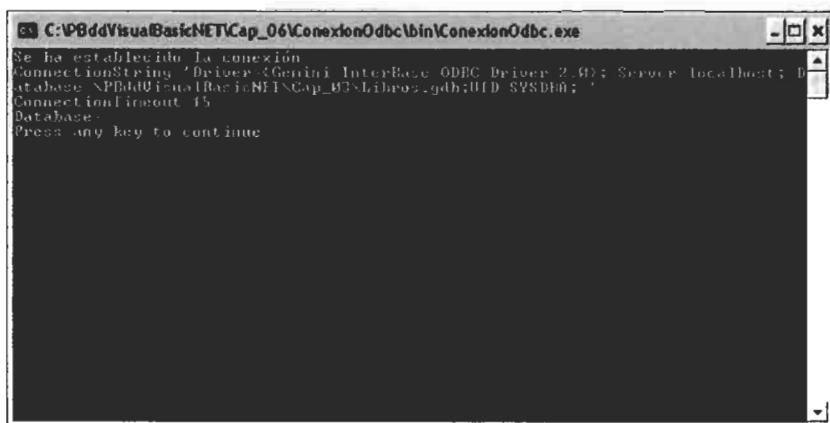


Figura 6.7. Logramos conectar con la base de datos InterBase

Como se indicó en el tercer capítulo, la base de datos creada a modo de ejemplo reside en un servidor Oracle 9i funcionando sobre Windows .NET Server. La aplicación cliente, el programa de ejemplo que vamos a desarrollar, se ejecutará desde otra máquina, en la misma red, que funciona con Windows XP Professional. En dicha máquina es preciso instalar el software cliente de Oracle, debidamente configurado para comunicarse con el servidor.

Durante la instalación del software cliente, en mi caso se ha instalado el de Oracle 9i sobre Windows XP, tendrá opción a crear un servicio de acceso a una base de datos. Si no lo hizo entonces, tendrá que elegir la opción Configuration and Migration Tools>Net Configuration Assistant para crear ese servicio. Aparecerá un asistente en el que, como se aprecia en la figura 6.8, puede configurar distintos elementos. Hay que elegir la opción Configuración del Nombre del Servicio de Red Local y pulsar el botón **Siguiente**.

En siguiente página del asistente elija la opción **Agregar**, pulsando de nuevo el botón **Siguiente**. Después tendrá que elegir la versión de la base de datos a la que va a acceder. Deje marcada la opción que aparece elegida, **Base de datos o servicio Oracle8i o posterior**, y pulse el botón **Siguiente** una vez más.

A continuación, en la cuarta ventana del asistente, tendrá que introducir el nombre del servicio con el que se conoce a la base de datos en el servidor. En nuestro caso, durante el proceso de creación, utilizamos como nombre global **libros.fcharte.com**, siendo ese el valor que tendríamos que introducir en este caso.

La ventana siguiente le permite seleccionar entre diferentes medios de comunicación entre cliente y servidor. Deje seleccionado el elemento **TCP** en la lista de posibilidades y pulse el botón **Siguiente** nuevamente. Los parámetros más impor-

tantes, introducidos en estos dos últimos pasos, se completan con el siguiente, en el que debe facilitarse el nombre del servidor donde está ejecutándose el RDBMS, así como seleccionar el puerto de comunicación en caso de que no fuese el usado por defecto por Oracle. La figura 6.9 corresponde a mi configuración particular. Tendrá que sustituir el nombre del servidor por el del ordenador en el que tenga instalado Oracle.

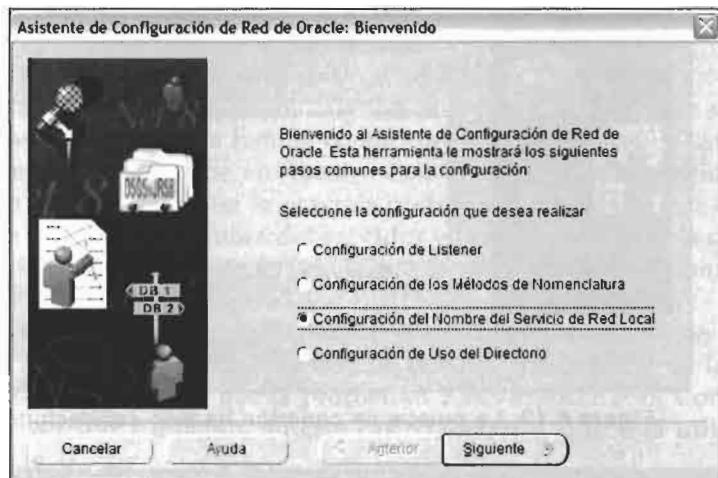


Figura 6.8. Asistente para la configuración de red de Oracle

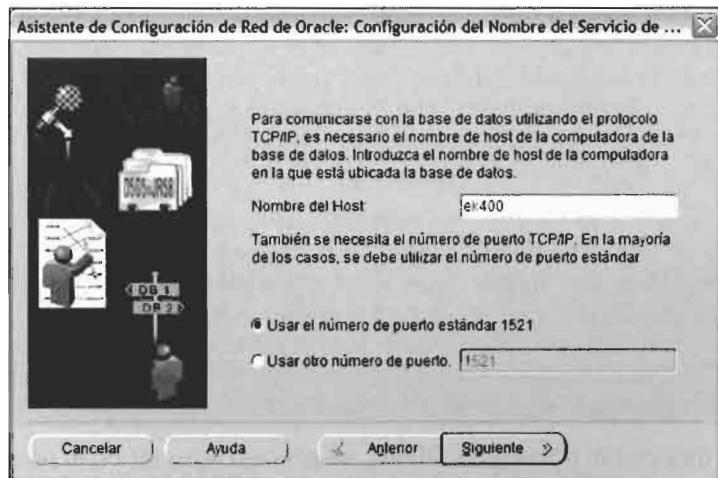


Figura 6.9. Indicamos el servidor en el que está el RDBMS Oracle y, si es necesario, el número de puerto por el que nos comunicaremos con él

Una vez que el asistente conoce todos los parámetros necesarios, en el paso siguiente se ofrece la posibilidad de efectuar una prueba de conexión. Seleccione la

opción Si, realizar una prueba y pulse el botón **Siguiente**. El resultado obtenido debería ser el de la figura 6.10. Si obtiene un error revise toda la configuración previa pulsando el botón **Anterior**.

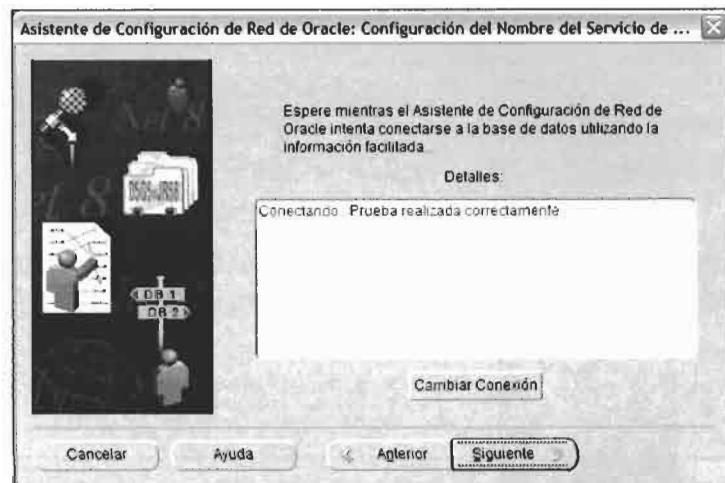


Figura 6.10. La prueba de conexión ha sido satisfactoria

Nota

Para efectuar la prueba de conexión, el asistente utiliza la cuenta y clave creadas a modo de ejemplo en todas las bases de datos Oracle, scott y tiger. Puede pulsar el botón **Cambiar Conexión**, en la ventana de resultado de la prueba de conexión, para cambiar el esquema y clave y probar con los que haya definido o le haya indicado su administrador de bases de datos.

Por último, antes de volver al inicio del asistente, tendrá que introducir el nombre de servicio local para acceder a esta configuración. Introduzca Libros y pulse el botón **Siguiente**. Salga del asistente, ahora ya está en disposición de introducir el código Visual Basic .NET para conectar con la base de datos.

Nota

Al igual que con el proveedor ODBC, es preciso agregar en el proyecto una referencia al ensamblado que contiene el proveedor para Oracle, el ensamblado System.Data.OracleClient.dll.

El código base utilizado en todos los ejemplos anteriores tendrá que sufrir las modificaciones siguientes:

- Introducir `System.Data.OracleClient` tras la sentencia `Imports`.
- La variable de conexión ahora sería de tipo `OracleConnection` y la cadena de conexión, entregada al constructor, sería `"Data Source=Libros; User ID=scott; Password=tiger"`.
- Habría que eliminar el acceso a las propiedades `Database` y `ConnectionTimeout` que mostrábamos más abajo, ya que el proveedor Oracle carece de ellas, pudiendo mostrar en su lugar el contenido de la propiedad `ServerVersion`.

Al crear la cadena de conexión, asignamos a la propiedad `Data Source` el nombre de servicio local que hemos creado previamente con el asistente de configuración de Oracle. Aunque en nuestro caso dicho nombre coincide con el SID original de la base de datos, en la práctica podría ser otro. Dado que el nombre de servicio lleva implícito el nombre del servidor y la base de datos a la que va a accederse, los únicos parámetros adicionales que se necesitan son el nombre de usuario y la clave.

Si en la comprobación de conexión efectuada con el asistente no tuvo problemas, al ejecutar este programa el resultado debería ser como el de la figura 6.11. Observe los datos devueltos por la propiedad `ServerVersion`. Con ellos puede identificar la versión y edición, en este caso *Enterprise*, que está utilizándose del RDBMS Oracle.



Figura 6.11. La conexión con Oracle ha sido satisfactoria

Conexiones ODBC mediante DSN

En los cuatro ejemplos del punto anterior, conectando con diversos orígenes de datos, hemos introducido siempre los parámetros de conexión, tales como nombre de servidor, usuario y clave, directamente en la propia cadena de conexión. Esto,

en ocasiones, puede resultar un inconveniente, ya que cualquier cambio que hubiese que efectuar, por ejemplo por cambiar el RDBMS de un servidor a otro, implicaría una modificación del código, recompilación y redistribución de la aplicación.

Hay disponibles distintas alternativas para evitar este potencial problema, comenzando por la posibilidad de que el usuario configure, en la misma aplicación, parámetros como el servidor para lo cual, lógicamente, tendríamos que habilitar esa posibilidad.

Si estamos utilizando el proveedor Microsoft ODBC para .NET, una alternativa es la configuración de un DSN, un mecanismo por el que toda la información de conexión, exceptuando generalmente el nombre de usuario y clave, se asocia con un identificador. Éste se almacena en el registro de Windows o bien en un archivo, pudiendo utilizarse posteriormente desde código en sustitución de los parámetros a los que representa.

Tipos de DSN

Los DSN siempre tienen el mismo objetivo y cuentan con los mismos parámetros, pero la forma en que se almacenan hace que pueda hablarse de tres categorías diferentes de DSN:

- De usuario. La información se almacena en el registro de Windows, concretamente en la rama perteneciente a un determinado usuario. De esta forma, ese DSN sólo puede utilizarlo el usuario al que pertenece.
- De sistema. Similar al anterior, dado que se almacena en el registro, pero no asociado a un usuario en particular, sino en la sección general del sistema. Un DSN de sistema puede ser utilizado por cualquier usuario.
- De archivo. En vez de en el registro de Windows, la información de conexión se almacena en un archivo, lo cual permite transportarla a los equipos en los que pueda ser necesaria sin precisar la modificación del registro en todos ellos.

Elegir un tipo u otro dependerá directamente de las necesidades de cada proyecto. En cualquier caso, recuerde que un DSN puede utilizarse tan sólo desde ODBC, no resultando útil con los otros proveedores ADO.NET.

Creación de un DSN

Para crear un nuevo DSN tendrá que utilizar el **Administrador de orígenes de datos ODBC**. Éste se encuentra en el **Panel de control** de Windows, en el caso de las versiones servidor, y en Windows XP Professional, lo encontrará en la carpeta **Herramientas administrativas** del **Panel de control**. Haga doble clic sobre él, deberá aparecer un cuadro de diálogo similar al de la figura 6.12. La página **Controladores** enumera todos los controladores ODBC instalados en el sistema, mientras

que en las páginas **DSN de usuario**, **DSN de sistema** y **DSN de archivo** encontrará los DSN de cada tipo existentes en este momento.

Seleccione la página **DSN de sistema** y pulse el botón **Agregar**. Elija de la ventana que aparece, con una lista de todos los controladores, el controlador Microsoft Access Driver (*.mdb), haciendo doble clic sobre él. Entonces verá la ventana **Configuración de ODBC Microsoft Access** (véase figura 6.13) en la que debe introducir el nombre que va a dar al origen de datos y todos los demás parámetros. En este caso bastará con pulsar el botón **Seleccionar** y facilitar el camino y nombre del archivo de Microsoft Access que ya habíamos empleado anteriormente como ejemplo.

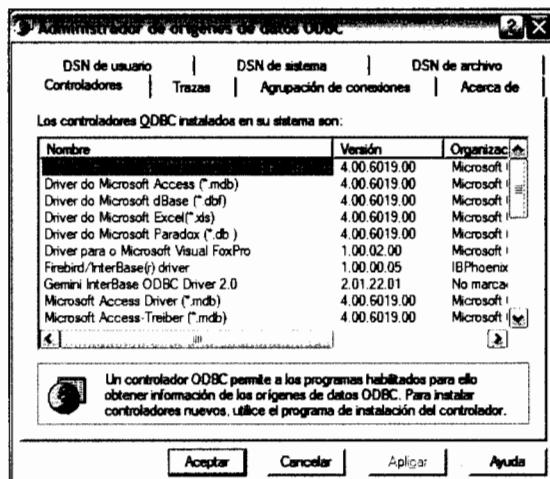


Figura 6.12. El Administrador de orígenes de datos ODBC cuenta con múltiples páginas

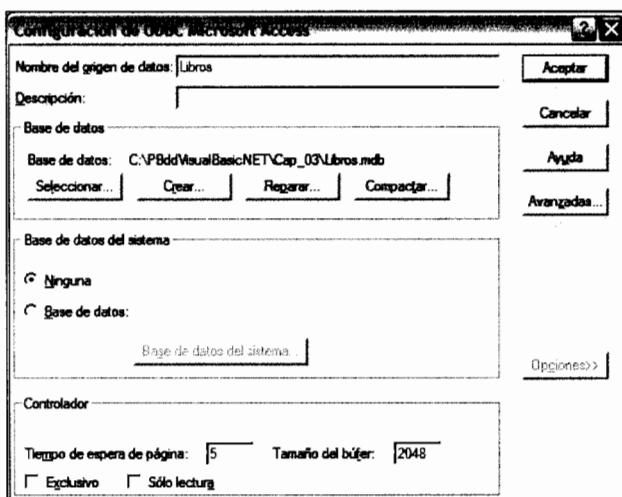


Figura 6.13. Configuración del nuevo DSN

Nota

En caso que desee asociar una cuenta de usuario y clave con el DSN, evitando que la aplicación o el propio usuario tenga que facilitar esta información, pulse el botón **Avanzadas** de la ventana Configuración de ODBC Microsoft Access.

Para terminar, pulse el botón **Aceptar** del cuadro de diálogo anterior. Volverá a la página **DSN de sistema** del Administrador de orígenes de datos ODBC, en cuya lista debe aparecer el DSN recién creado.

Uso del DSN con ADO.NET

Una vez que tenemos creado el DSN en el equipo donde va a ejecutarse la aplicación, utilizarlo desde ésta, con el proveedor `Odbc`, es realmente sencillo. Tome el ejemplo anterior en el que usábamos este proveedor para acceder a una base de datos InterBase, cambie la cadena de conexión facilitada al constructor dejándola como "`DSN=Libros`" y ejecute el programa. El resultado debería ser el de la figura 6.14. Ha conectado con Microsoft Access mediante ODBC facilitando sólo el nombre de un DSN.

Observe que ahora la propiedad `Database` facilita el camino en el que se encuentra el archivo, así como su nombre sin extensión.



Figura 6.14. Confirmación de la conexión mediante el DSN `Libros`

Nota

Si en lugar de un DSN de sistema o usuario hubiésemos creado uno de archivo, tendríamos que modificar la cadena de conexión dejándola como "`FILEDSN=Archivo.dsn`".

Archivos UDL

Los DSN son un recurso de ODBC para almacenar la información de conexión, una solución exclusiva que no puede utilizarse si, posteriormente, en nuestro código no vamos a emplear el proveedor `Odbc`. OLE DB cuenta con un mecanismo similar, si bien en este caso la información siempre se almacena en un archivo con extensión `udl`.

Para efectuar una prueba, sencilla y rápida, dé los pasos que se indican ahora:

- En la carpeta correspondiente al proyecto Visual Basic .NET en el que conectábamos, mediante el proveedor `OleDb`, con Microsoft Excel, haga clic con el botón secundario del ratón sobre el contenido de la carpeta (en el panel derecho del Explorador de Windows), seleccione la opción **Nuevo>Documento de texto** y llame al archivo `Conexión.udl`. Responda afirmativamente a la pregunta de si desea cambiar la extensión del archivo.
- Haga doble clic sobre el archivo recién creado, abriendo la ventana **Propiedades de vínculo de datos**.
- En la página **Proveedor** (véase figura 6.15) seleccione el proveedor **Microsoft Jet 4.0 OLE DB Provider**. Pulse el botón **Siguiente >>**.

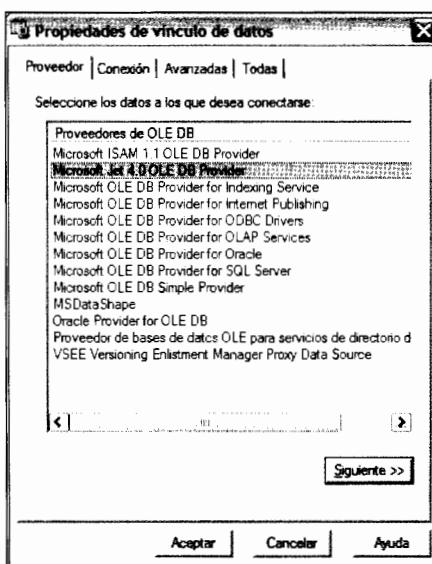


Figura 6.15. Seleccionamos el proveedor OLE DB a utilizar

- En la página **Conexión** pulse el botón ... que hay asociado a la primera opción, facilitando el camino y nombre del documento Microsoft Excel usado anteriormente como ejemplo.

- Vaya a la página **Todas** y haga doble clic sobre la propiedad **Extended Properties**, en la lista que aparece en el área central de la ventana.
- La ventana **Modificar valor de propiedad** nos permite introducir el valor que deseamos dar a la propiedad **Extended Properties**. En este caso, como puede verse en la figura 6.16, escribimos la secuencia **Excel 8.0;HDR=Yes** y pulsamos el botón **Aceptar**.

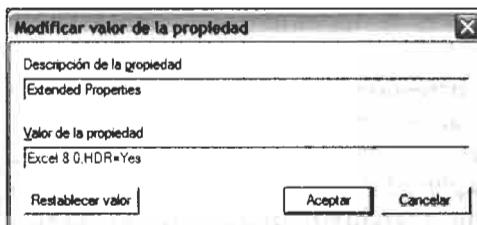


Figura 6.16. Propiedades extendidas de la configuración de conexión

- Finalmente, vuelva a la página **Conexión** y pulse el botón **Probar conexión** que hay en la parte inferior. Debería aparecer el mensaje **La prueba de conexión fue satisfactoria**. De lo contrario, revise los pasos comprobando que todos los parámetros son correctos.

Utilizar el archivo **udl** desde una aplicación propia es tan fácil como usar un DSN. Abra el proyecto en el que, mediante el proveedor **OleDb**, conectábamos con una hoja de cálculo Excel. Cambie entonces la cadena de conexión, asignada en este caso a la propiedad **ConnectionString**, dejándola como "**File Name=..\Conexión.udl**". Al ejecutar el programa se obtendría una conexión satisfactoria, si bien en este caso la propiedad **Database** no nos indica el origen de datos al que estamos conectados.

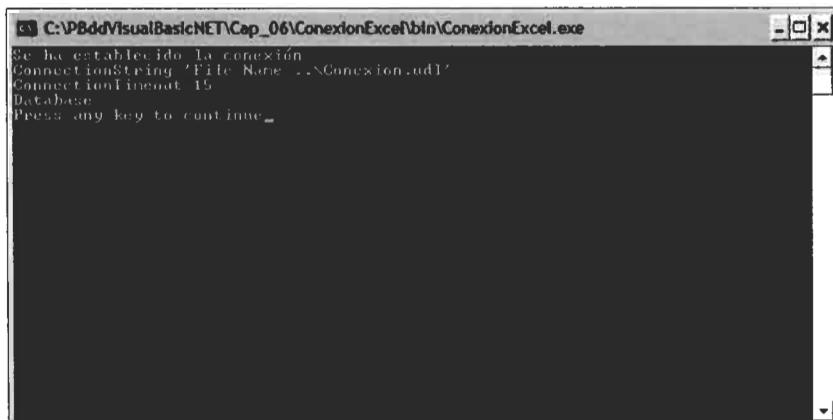
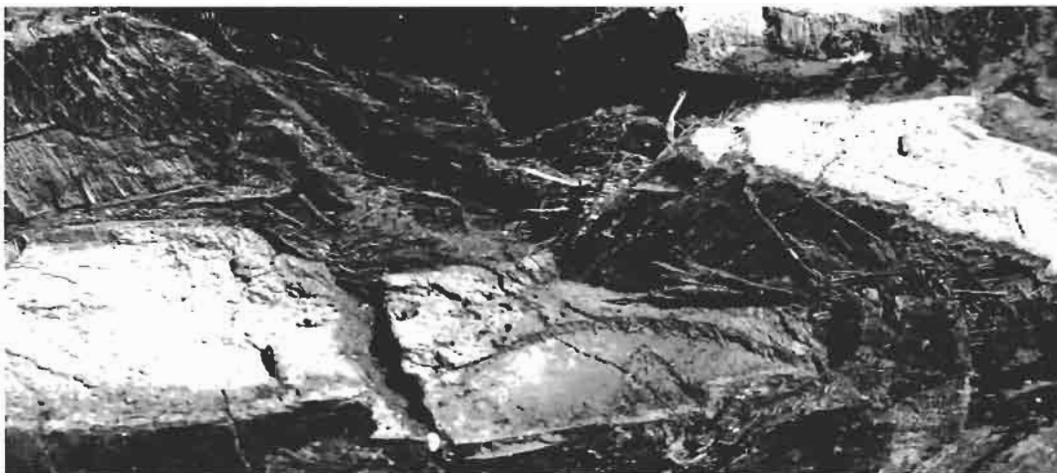


Figura 6.17. Conexión al documento Excel mediante la configuración del archivo **udl**

Resumen

Como ha podido ver en este capítulo, mediante las clases `Connection` de cada uno de los proveedores ADO.NET es posible conectar virtualmente con cualquier origen de datos, tanto RDBMS como no RDBMS. Por una parte tenemos dos proveedores específicos, `SqlClient` y `OracleClient`, capaces de comunicarse directamente con SQL Server y Oracle, respectivamente. Por otra, los proveedores `OleDb` y `Odbc` abren las puertas al uso de cualquier controlador OLE DB u ODBC que pudiera existir, facilitando el acceso a Microsoft Excel, Microsoft Access o InterBase, pero también a dBase, Paradox, Sybase, DB2, MySQL y, en general, cualquier base de datos.

Ahora que ya conocemos la sintaxis general de las cadenas de conexión, así como los miembros de la interfaz `IDbConnection` que permite abrir, cerrar y comprobar la conexión, estamos en disposición de empezar a recuperar datos desde el origen al que se haya conectado. Éste será el objetivo del próximo capítulo.



7

Información de esquema de la base de datos

La mayoría de las aplicaciones que tratan con datos, una vez que han conectado con el origen solicitan directamente la información que precisan o ejecutan los comandos pertinentes. Estas aplicaciones, por regla general, ya conocen de antemano la estructura de la base de datos, ya que ésta, habitualmente, estaba disponible cuando el equipo de desarrollo planificó la aplicación. Este escenario podríamos denominarlo como *normal*. No obstante, las excepciones también existen en este campo y es posible que al crear nuestra aplicación desconozcamos los nombres de las tablas o columnas. Es un caso que se da, sobre todo, cuando el programa no está construido específicamente para operar sobre un cierto origen de datos sino que, por el contrario, debe facilitar el acceso a cualquier estructura de información. Es el caso típico al escribir alguna utilidad general para un RDBMS o, como en este capítulo, cuando quiere simplemente experimentarse para conocer cómo conseguir una determinada información.

El objetivo de este capítulo es describir el proceso a seguir para obtener información de esquema de una base de datos, utilizando para ello los servicios del proveedor ADO.NET o, si éste carece de ellos, los específicos del origen con el que vaya a tratarse.

¿Qué es la información de esquema?

Salvo excepciones, como es el caso de las hojas de cálculo Excel, todos los orígenes de datos almacenan, aparte de los datos propiamente dichos, información

sobre la estructura de éstos. Todos los RDBMS, las bases de datos locales e, incluso, los archivos XML utilizan esta información de esquema para saber cuáles son las tablas existentes, con qué columnas cuentan, qué tipo de dato corresponde a cada una de ellas, etc.

Si conectamos con un origen de datos cuya estructura desconocemos, la recuperación de la información de esquema, también conocida como *meta-information*, nos permitirá adaptar dinámicamente el funcionamiento del programa para que trate los datos apropiadamente.

Los proveedores ADO.NET no cuentan con servicios que faciliten la recuperación de información de esquema, ya que su finalidad es ser lo más simples y eficientes posible, facilitando así también el desarrollo de nuevos proveedores.

Para recuperar la información de esquema de la base de datos, por tanto, tendremos que recurrir a medios alternativos. Existen básicamente dos: utilizar los servicios del controlador OLE DB, si es que vamos a usar el proveedor ADO.NET OleDb, o servirnos de las instrucciones que el propio origen de datos facilite para obtener esa información. En los puntos siguientes vamos a tratar ambos casos.

Orígenes OLE DB

El proveedor OleDb de ADO.NET no accede directamente a un origen de datos, como sí hacen SqlClient u OracleClient, sino que utiliza a modo de intermediario un controlador OLE DB existente. Los controladores OLE DB, como los proveedores ADO.NET, se ajustan a un estándar definido por Microsoft, contando todos ellos con métodos que permiten recuperar información de esquema del origen al que han conectado. Por eso el objeto OleDbConnection dispone de un método, llamado GetOleDbSchemaTable(), que hace uso del método original del controlador OLE DB, limitándose a devolver el resultado.

En caso de que no tuviésemos una alternativa mejor con el proveedor original del origen de datos al que vayamos a acceder, siempre existe la alternativa de usar el proveedor OleDb para recuperar la información de esquema y, posteriormente, emplear el proveedor nativo para el resto del trabajo. Conociendo el funcionamiento de dicho origen de datos, sin embargo, esto no sería preciso.

El método OleDbConnection.GetOleDbSchemaTable() necesita dos parámetros y devuelve como resultado un objeto DataTable con la información solicitada. El primero de los parámetros selecciona la información de esquema a obtener: lista de las tablas, procedimientos almacenados, vistas, etc. Los valores posibles son los definidos en la clase OleDbSchemaGuid como campos compartidos, algunos de los cuales se enumeran en la tabla 7.1.

Como segundo parámetro, el método GetOleDbSchemaTable() necesita un arreglo de objetos Object, contenido cada uno de ellos una restricción aplicable a una columna del resultado que se obtenga. Las restricciones posibles dependerán del valor entregado como primer parámetro. Si éste es OleDbSchemaGuid.Tables, por poner un ejemplo, el arreglo a facilitar como segundo parámetro de-

berá constar de cuatro elementos: nombre del catálogo a consultar, nombre del esquema, nombre de la tabla y tipo de la tabla. La mayoría de los valores pueden ser nulos. Si no se indica un catálogo, por ejemplo, se retornan todas las tablas de todos los catálogos. Para obtener todas las claves primarias de una tabla, otro ejemplo, los parámetros serían tres: el nombre del catálogo, el del esquema y el de la tabla cuya clave primaria deseamos recuperar.

Tabla 7.1. Valores a usar como primer parámetro del método
GetOleDbSchemaTable()

Valor	Información a recuperar
OleDbSchemaGuid.Tables	Tablas de la base de datos
OleDbSchemaGuid.Columns	Columnas de las tablas
OleDbSchemaGuid.Table_Constraints	Restricciones definidas en las tablas
OleDbSchemaGuid.Procedures	Procedimientos existentes en la base de datos
OleDbSchemaGuid.Procedure_Parameters	Parámetros de entrada de los procedimientos almacenados
OleDbSchemaGuid.Views	Lista de las vistas que haya definidas

Tabla de resultados

El valor devuelto por cada llamada a la función GetOleDbSchemaTable() es un objeto `DataTable`, cuyas bases conocimos de manera superficial en un capítulo previo y que aprenderemos a usar con mayor detalle en uno posterior. La estructura de ese `DataTable` dependerá de la información que hayamos solicitado, contando con más o menos columnas y filas.

Si no sabe qué columnas puede encontrar en el `DataTable`, no tiene más que recorrer la colección a la que apunta su propiedad `Columns` para obtenerlas de manera individual y recuperar, por ejemplo, el nombre y tipo de cada una de ellas. Conociendo las columnas, podrá recorrer las filas, mediante la propiedad `Rows` para acceder al nombre de cada tabla, tipo de cada parámetro, etc.

En general, si quiere mostrar todo el contenido del `DataTable`, podría utilizarse como patrón un bloque de código similar al siguiente:

```

Dim ConnOleDb As New OleDbConnection()
Dim TblResultado As DataTable
Dim Fila As DataRow, Columna As DataColumn
...
For Each Fila In TblResultado.Rows
    For Each Columna In TblResultado.Columns

```

```

        Console.WriteLine(Fila(Columna) & vbTab)
Next
        Console.WriteLine()
Next

```

Si conocemos los datos que van a devolverse, en lugar de mostrarlos todos podemos obtener sólo aquellos que más nos interesan. Al recuperar una lista de tablas existentes en el origen, por ejemplo, recuperaríamos el nombre de cada tabla con `Fila("TABLE_NAME")`, sin necesidad de recorrer todas las columnas.

En la práctica

No necesitamos saber mucho más de lo ya explicado para, sirviéndonos del proveedor `OleDb`, recuperar información de esquema de cualquier origen de datos y mostrarlos en pantalla. Vamos a codificar un pequeño ejemplo que haga precisamente eso, mostrando por la consola la información recuperada a partir del documento Microsoft Excel y la base de datos Microsoft Access, orígenes a los que ya en el capítulo previo habíamos conectado mediante el proveedor `OleDb`.

Para evitar la repetición de código, crearemos una función genérica capaz de mostrar información de cualquier origen OLE DB. Esa función toma dos parámetros: un título a mostrar por consola y la cadena de conexión a emplear para acceder al origen. Con esos datos se establece la conexión, recupera la lista de todas las tablas existentes y se muestra por consola.

```

' Este módulo es el encargado de abrir el origen
' de datos y recuperar información sobre su estructura
Sub MuestraEsquema(ByVal Titulo As String, _
ByVal CadenaConexion As String)
    ' Declararemos las variables necesarias
    Dim ConnOleDb As New OleDbConnection(CadenaConexion)
    Dim TblResultado As DataTable
    Dim Fila As DataRow, Columna As DataColumn

    ConnOleDb.Open() ' Abrimos la conexión

    ' Y recuperaremos todas las tablas del origen
    TblResultado = ConnOleDb.GetOleDbSchemaTable( _
        OleDbSchemaGuid.Tables, _
        New Object() {Nothing, Nothing, Nothing, Nothing})

    ConnOleDb.Close() ' Cerramos la conexión

    ' Unos encabezados
    Console.WriteLine("**** " & Titulo & " ****")
    Console.WriteLine("Tipo" & vbTab & "Nombre")
    Console.WriteLine("=====***=====")

    ' Recorremos todas las filas del DataTable
    For Each Fila In TblResultado.Rows
        ' mostrando el tipo y nombre de cada tabla

```

```

Console.WriteLine("TABLE_TYPE").ToString() & vbTab)
Console.WriteLine(Fila("TABLE_NAME").ToString())
Next
Console.WriteLine(vbCrLf & vbCrLf)
End Sub

```

Disponiendo de esta función, lo único que tenemos que hacer es añadir al método Main() las dos sentencias siguientes:

```

' Muestra Esquema de la base de datos de Excel
MuestraEsquema("Hoja de cálculo Excel",
  "Provider=Microsoft.Jet.OLEDB.4.0; " &
  "Data Source=\PBddVisualBasicNET\Cap_03\Libros.xls; " &
  "Extended Properties='Excel 8.0; HDR=Yes'")

' Muestra Esquema de la base de datos de Access
MuestraEsquema("Base Microsoft Access",
  "Provider=Microsoft.Jet.OLEDB.4.0; " &
  "Data Source=\PBddVisualBasicNET\Cap_03\Libros.mdb")

```

Con ellas solicitamos la visualización de información de esquema de los dos orígenes antes mencionados, simplemente facilitando la cadena de conexión adecuada. Su ejecución produciría el resultado que ve en la figura 7.1. Observe que en el caso de Excel no existen más que dos tablas, en realidad las dos páginas del libro, mientras que en el caso de Access aparecen, además, varias tablas de sistemas.

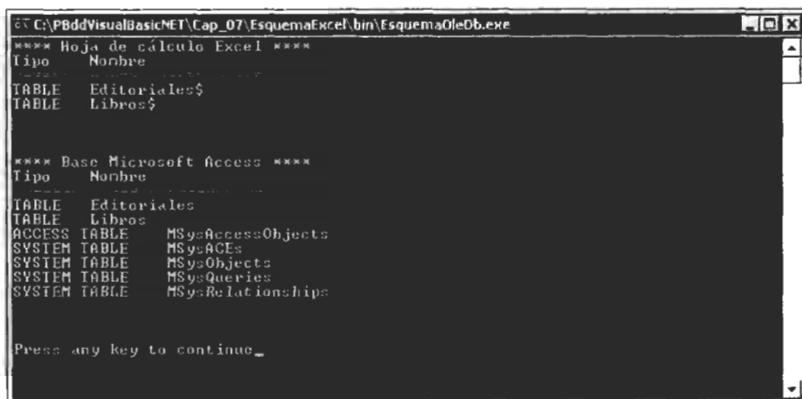


Figura 7.1. Enumeración de las tablas existentes en los dos orígenes de datos

Nota

En lugar de mostrar los nombres de las tablas en pantalla, podríamos utilizarlos para componer una sentencia SQL de cualquier tipo, eliminando la tabla, obteniendo su contenido, etc.

Otros orígenes

Como se indicaba antes, ninguno de los proveedores ADO.NET cuenta con medios propios para recuperar la información de esquema del origen de datos si exceptuamos lo que acabamos de ver en el punto anterior. Para recuperar la lista de tablas de una base de datos SQL Server u Oracle, utilizando sus respectivos proveedores, tendríamos que utilizar las sentencias Transact-SQL o PL/SQL, respectivamente, que procediesen.

Tendríamos que definir la cadena de conexión con la base de datos, ejecutar una sentencia SQL y recoger el resultado que, como en el caso de `GetOleDbSchemaTable()`, sería un objeto `DataTable`. Como intermediario tendríamos que usar un adaptador de datos, como si ejecutásemos cualquier sentencia SQL de recuperación de información. Éste es un tema del que nos ocuparemos más adelante, concretamente en el noveno capítulo, pero en el siguiente ejemplo usaremos sendos adaptadores para poder recoger la lista de tablas que hay en las bases de datos SQL Server y Oracle.

Ambos RDBMS, SQL Server y Oracle, cuentan con una serie de tablas especiales en las que almacenan la información de esquema de la base de datos. Estas tablas pueden ser consultadas, siempre que se cuente con los permisos adecuados, como cualquier otra tabla, mediante una sentencia `SELECT`.

En el caso de SQL Server existe un esquema, llamado `INFORMATION_SCHEMA`, en el que existen diversas tablas, como `TABLES`, `COLUMNS`, `ROUTINES`, etc. Podemos ejecutar la sentencia `SELECT * FROM INFORMATION_SCHEMA.TABLES`, por ejemplo, para recuperar todas las tablas de la base de datos. Algunas de las columnas de datos devueltas, relativas a las tablas, son `TABLE_CATALOG`, `TABLE_NAME` y `TABLE_TYPE`, que pueden ser usadas, en caso necesario, para filtrar el resultado.

Oracle dispone de tablas similares, llamadas `USER_TABLES`, `USER_VIEWS`, `USER_CATALOG`, etc., mediante las cuales podemos recuperar diversa información de la base de datos. En el caso de `USER_TABLES`, algunas de las columnas existentes son `TABLESPACE_NAME`, `TABLE_NAME`, `PCT_FREE` y `PCT_USED`.

Nota

La obtención de información de esquema de otros RDBMS, como InterBase, MySQL, DB2, etc., se efectúa de manera similar. Recurra a la documentación del producto particular en el que esté interesado.

En la práctica

Veamos cómo podemos crear un pequeño programa que nos muestre por consola la lista de tablas existentes en las bases de datos SQL Server y Oracle que creamos en el tercer capítulo. A fin de tener un único ejemplo, como en el caso de Excel

y Access, codificaremos una función que, recibiendo un DataTable, se encargue de mostrar su contenido por consola. En este caso, al tener que usar proveedores distintos y, además, emplear un adaptador de datos para recuperar la información, el método MuestraEsquema() será más sencillo y, a cambio, una gran parte del código estará en Main(), como puede verse en el siguiente programa completo.

```
' Necesitamos las clases de ambos proveedores
Imports System.Data.SqlClient
Imports System.Data.OracleClient

Module Module1

    Sub Main()
        ' Preparamos el DataTable
        Dim TblResultado As New DataTable()

        ' A partir de la conexión con SQL Server
        Dim AdaptadorSql As New SqlDataAdapter(_
            "SELECT * FROM INFORMATION_SCHEMA.TABLES", _
            New SqlConnection("Data Source=inspiron; " & _
                "Initial Catalog=Libros; User ID=sa; Password="))

        AdaptadorSql.Fill(TblResultado) ' lo rellenamos
        ' y enviamos para visualización por consola
        MuestraEsquema("** SQL Server **", TblResultado)

        ' A partir de la conexión con Oracle
        Dim AdaptadorOracle As New OracleDataAdapter(_
            "SELECT * FROM USER_TABLES", _
            New OracleConnection("Data Source=Libros; " & _
                "User ID=scott; Password=tiger"))

        TblResultado = New DataTable()
        AdaptadorOracle.Fill(TblResultado) ' lo rellenamos
        ' y enviamos para visualización por consola
        MuestraEsquema("** Oracle **", TblResultado)
    End Sub

    ' Este método recibe el título y el DataTable a mostrar
    Sub MuestraEsquema(ByVal Titulo As String, _
        ByVal TblResultado As DataTable)

        ' Variables para operar sobre el DataTable
        Dim Fila As DataRow, Columna As DataColumn

        ' Unos encabezados
        For Each Columna In TblResultado.Columns
            Console.Write(Columna.ColumnName & vbTab)
        Next
        Console.WriteLine(vbCrLf & New String("=", 60))

        ' Recorremos todas las filas del DataTable
        For Each Fila In TblResultado.Rows
            ' mostrando todas sus columnas
```

```

For Each Columna In TblResultado.Columns
    Console.WriteLine(Fila(Columna) & vbTab)
Next
Console.WriteLine()
Next

Console.WriteLine(vbCrLf & vbCrLf)
End Sub

End Module

```

Como puede ver, creamos sendos adaptadores de datos, un `SqlDataAdapter` y un `OracleDataAdapter`, entregándoles la consulta específica y un componente de conexión también exclusivo. El adaptador de datos, al llamar al método `Fill()`, se encargará de abrir la conexión, ejecutar un comando con la consulta entregada como primer parámetro, recuperar el resultado y cerrar la conexión. Al terminar, los datos se encontrarán en el `DataTable` dado como parámetro al método `Fill()`.

Nota

No se preocupe en este momento por el funcionamiento de los adaptadores de datos, volveremos sobre ellos en el noveno capítulo.

Al ejecutar el programa, dado el ancho limitado de la consola, probablemente le resultará algo confusa la información mostrada. La lista de tablas de SQL Server cuenta con pocas columnas, y es fácil verlas, pero en el caso de Oracle el número de columnas es muy grande. Pruebe a redirigir la salida del programa a un archivo de texto y después ábralo en cualquier editor de texto, como el Bloc de notas o el propio de Visual Studio .NET, le resultará mucho más fácil analizar los datos.

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE
<hr/>			
Librería	dbo	ctproperties	BASE TABLE
Librería	dbo	Editoriales	BASE TABLE
<hr/>			
Librería	dbo	L1	BASE TABLE
Librería	dbo	LlibrosEditorial	VIEW
Librería	dbo	sysconstraints	VIEW
Librería	dbo	syssegments	VIEW
<hr/>			
TABLE_NAME	TABLESPACE_NAME	CLUSTER_NAME	IOT_NAME
ACCOUNT	SYSTEM	10	40
BONUS	SYSTEM	10	40
DEPT	SYSTEM	10	40
EDITORIALES	SYSTEM	10	40
EMP	SYSTEM	10	40
LIBROS	SYSTEM	10	40
RECEIPT	SYSTEM	10	40
SALGRADE	SYSTEM	10	40
<hr/>			
PCT_FREE	PCT_USED	INI_TRANS	MAX_TRANS
65536	65536	1	2147483645
65536	65536	1	2147483645
65536	65536	1	2147483645
65536	65536	1	2147483645
65536	65536	1	2147483645
65536	65536	1	2147483645
65536	65536	1	2147483645
65536	65536	1	2147483645
<hr/>			
INI			
1	1	1	YES
1	1	1	YES
1	1	1	YES
1	1	1	YES
1	1	1	YES
1	1	1	YES
1	1	1	YES
1	1	1	YES
1	1	1	YES
1	1	1	YES

Figura 7.2. Lista de tablas SQL Server y Oracle mostradas en el Bloc de notas

Información sobre columnas

Aunque en los ejemplos previos nos hemos limitado a recuperar una lista de las tablas existentes en el origen de datos, en el caso de Excel y Access no hay más elementos disponibles puesto que no los creamos en su momento, en la práctica podríamos utilizar el mismo procedimiento para, por ejemplo, conocer cuáles son las vistas predefinidas, si hay procedimientos almacenados, etc. Tomando como base el ejemplo previo, que conecta con SQL Server y Oracle, pruebe usted mismo con los valores alternativos mencionados antes: INFORMATION_SCHEMA.VIEWS, USER_VIEWS, etc., y así conocer cuáles son los datos devueltos.

Si conociésemos tan sólo el nombre de cada tabla o vista, sin más, tampoco podríamos hacer mucho. La siguiente necesidad será conocer las columnas de cada una de esas entidades: nombre, tipo, etc. Aunque podríamos seguir empleando métodos específicos de cada proveedor u origen de datos para obtener esta información, existe una alternativa más fácil y, sobre todo, aplicable a cualquier origen con el que se haya conectado.

La interfaz `IDataReader`, implementada por las clases `DataReader` de cada proveedor, dispone de un método, llamado `GetSchemaTable()`, que facilita un `DataTable` con los datos de cada una de las columnas del conjunto de resultados que va a leerse.

Este conjunto de resultados viene definido por un comando ejecutado previamente con el método `ExecuteReader()` de `IDbCommand`, interfaz implementada por todos los comandos.

Descrito paso a paso, el proceso para obtener las columnas de cualquier tabla o vista sería el siguiente:

- Creación de un nuevo comando, `OleDbCommand`, `SqlCommand`, `OracleCommand` u `OdbcCommand`, con una sentencia del tipo `SELECT * FROM Tabla`, donde Tabla sería el nombre de una de las tablas previamente obtenidas en un `DataTable`. Al crear el comando éste se asocia directamente con la conexión ya abierta con el origen de datos.
- Llamada al método `ExecuteReader()` del comando, facilitando como parámetro `CommandBehavior.SchemaOnly`. Así solicitamos sólo la recuperación de información de esquema, sin datos. El resultado devuelto por este método es una referencia `IDataReader` que tendremos que almacenar en una variable, ya sea de ese tipo o del `DataReader` específico del proveedor que esté utilizándose.
- Llamada al método `GetSchemaTable()` del `DataReader`, recuperando así en un `DataTable` toda la información de columnas sobre la tabla o vista indicada.

Si consulta la información de referencia del método `GetSchemaTable()`, verá que el `DataTable` devuelto se compone de casi una veintena de campos en los

que encontramos, por ejemplo, el nombre de cada columna, su tipo original y tipo .NET, precisión numérica, si es o no único, si es sólo de lectura, etc.

Note

Como en el caso de los adaptadores, no se preocupe en este momento del funcionamiento de comandos y lectores de datos, es un tema que conocerá en un capítulo posterior. La teoría vista en el quinto capítulo será suficiente por ahora.

En la práctica

Tomando como punto de partida el primer ejemplo desarrollado en este capítulo, en el que se enumeraban las tablas del documento Excel y la base de datos Access, vamos a introducir algunos cambios para obtener, asimismo, el nombre y tipo de todas las columnas de cada tabla.

El método Main(), desde el que invocábamos dos veces a MuestraEsquema(), permanecerá sin cambios. El método MuestraEsquema(), al que pertenece el código siguiente, sí ha sufrido algunos retoques.

```
Sub MuestraEsquema(ByVal Titulo As String, _
    ByVal CadenaConexion As String)
    ' Declaramos las variables necesarias
    Dim Conn OleDb As New OleDbConnection(CadenaConexion)
    Dim TblResultado As DataTable
    Dim Fila As DataRow, Columna As DataColumn

    Conn OleDb.Open() ' Abrimos la conexión

    ' Y recuperaremos todas las tablas del origen
    TblResultado = Conn OleDb.GetOleDbSchemaTable( _
        OleDbSchemaGuid.Tables,
        New Object() {Nothing, Nothing, Nothing, "TABLE"}) 

    ' Unos encabezados
    Console.WriteLine("**** " & Titulo & " ****")
    Console.WriteLine("Tipo" & vbTab & "Nombre")
    Console.WriteLine("=====")

    ' Recorremos todas las filas del DataTable
    For Each Fila In TblResultado.Rows
        ' mostrando el tipo y nombre de cada tabla
        Console.WriteLine(Fila("TABLE_TYPE").ToString() & vbTab)
        Console.WriteLine(Fila("TABLE_NAME").ToString())

        ' Por cada tabla enumeramos sus columnas
        MuestraColumnas(Fila("TABLE_NAME"), Conn OleDb)
    Next
```

```

' una pequeña separación
Console.WriteLine(vbCrLf & vbCrLf)

ConnOLEDb.Close() ' Cerramos la conexión
End Sub

```

Para empezar, al llamar a `GetOleDbSchemaTable()` para obtener la lista de tablas facilitamos como cuarto elemento del segundo parámetro el valor "TABLE". De esta forma obtendremos sólo tablas reales de usuario, no tablas de uso interno del origen de datos. La conexión establecida al principio no la cerramos justo después de recuperar la lista de tablas, ya que para ejecutar el comando de creación del Reader la necesitamos abierta. Por eso la sentencia `ConnOLEDb.Close()` la hemos llevado al final del método. Por último, fíjese en que dentro del bucle que enumera las tablas se invoca a un nuevo método, `MuestraColumnas()`, facilitándole el nombre de cada tabla y la referencia a la conexión utilizada.

La parte más interesante, como cabría esperar, es el nuevo método `MuestraColumnas()`. Su implementación es la mostrada a continuación:

```

' Método para mostrar las columnas de una tabla
Public Sub MuestraColumnas(ByVal NombreTabla As String, _
    ByVal Conexion As OleDbConnection)

    ' En caso de que el nombre de la tabla termine en $
    If NombreTabla.EndsWith("$") Then
        ' lo introduciremos entre corchetes para evitar problemas
        NombreTabla = "[" & NombreTabla & "]"
    End If

    ' Preparamos el comando de selección de todas
    ' las columnas de la tabla
    Dim Comando As New OleDbCommand(
        "SELECT * FROM " & NombreTabla, Conexion)

    ' Creamos el DataReader, recuperando sólo esquema
    Dim Lector As IDataReader =
        Comando.ExecuteReader(CommandBehavior.SchemaOnly)

    ' Obtenemos la tabla de esquema en un DataTable
    Dim TblResultado As DataTable = Lector.GetSchemaTable()

    ' Para recorrer las filas
    Dim Fila As DataRow

    ' Preparamos una cabecera
    Console.WriteLine(vbCrLf & "--- Columnas de la tabla " &
        NombreTabla & " ---")
    Console.WriteLine(vbTab & "Nombre".PadRight(15) &
        vbCrLf & "Tipo")
    Console.WriteLine(vbTab & New String("-", 40))

    ' Recorremos el resultado
    For Each Fila In TblResultado.Rows

```

```

' Mostrando el nombre y tipo de cada columna
Console.WriteLine(vbTab & New
    String(Fila("ColumnName") -_
    Console.WriteLine(Fila("DataType"))
Next

Lector.Close() ' Cerramos el DataReader

Console.WriteLine(vbCrLf & vbCrLf) ' Una separación
End Sub

```

El primer condicional comprueba si el nombre de la tabla finaliza con el símbolo \$, habitual al operar sobre documentos Excel, en cuyo caso lo delimitamos entre corchetes para evitar problemas en la ejecución de la consulta. Acto seguido implementamos los pasos antes descritos: creamos el comando con la consulta, obtenemos el `IDataReader` y llamamos a su método `GetSchemaTable()`. A partir de ahí el código es similar al del método `MuestraEsquema()`, recorriendose todas las filas para mostrar el nombre y tipo de cada columna.

Al ejecutar el programa debería obtener un resultado similar al de la figura 7.3. En ella pueden verse todas las columnas de las dos tablas que habíamos definido en Excel y Access. Observe los tipos asociados a las columnas de Excel. Al crear este documento no se indicó tipo de dato de cada columna, por lo que el proveedor lo deduce a partir de la información que contienen las celdillas. En cualquier caso, con esta información podríamos crear una interfaz que, por ejemplo, permitiese conectar con cualquier origen de datos y enumerarse sus tablas, vistas, procedimientos, etc., accediendo a la lista de columna cuando el usuario seleccionase una tabla, vista o procedimiento en concreto.

Nota

Puede utilizar el mismo camino para enumerar las columnas de las tablas SQL Server, Oracle o bien InterBase. Una vez ejecutado el comando y obtenido el `DataReader`, todo el proceso es idéntico.

Resumen

Los proveedores de datos ADO.NET están diseñados con una arquitectura que es relativamente simple, facilitando su desarrollo y, sobre todo, potenciando el rendimiento sobre los demás aspectos. Ésta es la razón de que en ellos no existan métodos para acceder a la información de esquema de los orígenes de datos, siendo preciso utilizar medios más específicos y, por lo tanto, dependientes de cada origen en particular.

En este capítulo ha conocido los procedimientos para obtener información de esquema mediante controladores OLE DB y consultando tablas específicas de SQL

Server y Oracle. Los demás RDBMS cuentan con mecanismos similares, no tenemos más que consultar la documentación de referencia y localizar las tablas, vistas o procedimientos almacenados que entregan esa información de esquema.

```

C:\PBddVisualBasicNET\Cap_07\EsquemaExcel\bin\EsquemaOleDb.exe
*** Hoja de cálculo Excel ****
Tipo Nombre

TABLE Editoriales
--- Columnas de la tabla [Editoriales$] ---
Nombre      Tipo
IDEditorial System.Double
Nombre      System.String
Direccion   System.String

TABLE Libros
--- Columnas de la tabla [Libros$] ---
Nombre      Tipo
IDLibro     System.Double
ISBN        System.String
Titulo      System.String
Autor       System.String
Editorial   System.Double
Precio      System.Double

*** Base Microsoft Access ***
Tipo Nombre

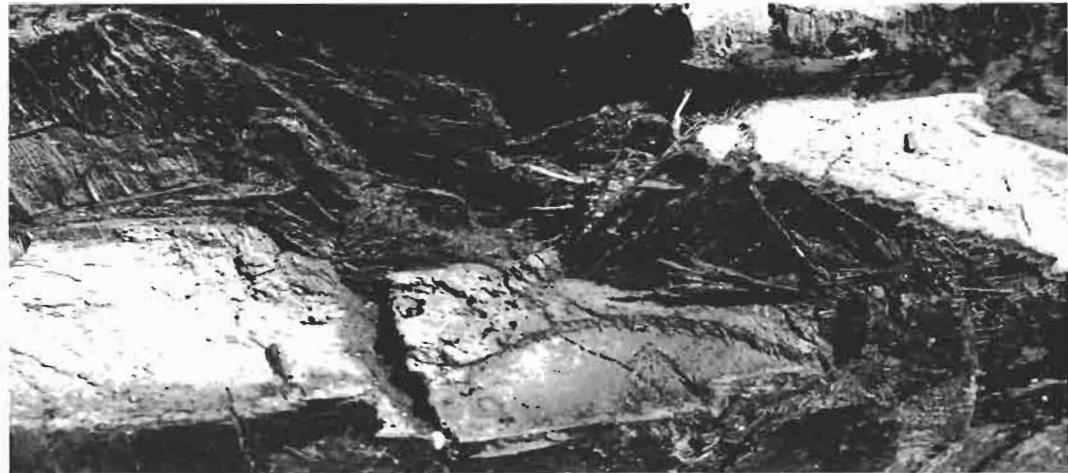
TABLE Editoriales
--- Columnas de la tabla Editoriales ---
Nombre      Tipo
IDEditorial System.Int32
Nombre      System.String
Direccion   System.String

TABLE Libros
--- Columnas de la tabla Libros ---
Nombre      Tipo
IDLibro     System.Int32
ISBN        System.String
Titulo      System.String
Autor       System.String
Editorial   System.Int32
Precio      System.Decimal

```

Figura 7.3. Enumeración de tablas y columnas existentes en Excel y Access

Una vez se conoce el nombre del objeto a consultar, la recuperación de información sobre sus columnas resulta mucho más fácil gracias al método `GetSchemaTable()` de la interfaz `IDataReader`. En el capítulo siguiente conoceremos con más detalle ésta interfaz y las implementaciones que efectúan los diversos proveedores.



8

Recuperación de datos

Conociendo la estructura que tiene el origen de datos al que va a accederse, bien porque lo hayamos diseñado nosotros mismos o lo hayamos recuperado según las indicaciones del capítulo previo, el siguiente paso que daremos será recuperar los datos que contiene dicho origen. Para ello tendremos que preparar comandos, ejecutarlos y obtener lectores de datos, operación que ya hemos efectuado en el ejemplo final del anterior capítulo.

Nuestro objetivo es conocer las interfaces `IDbCommand` e `IDataAdapter`, sus implementaciones particulares en cada proveedor y la forma de utilizarlas para obtener cursos de datos sólo de lectura y unidireccionales. Este método de recuperación de datos es el apropiado cuando va a generarse un informe, efectuar cálculos a partir de los datos o cualquier otro escenario en el que no se requiera la manipulación de la información y, además, el proceso vaya a efectuarse de forma relativamente rápida, ya que exige el mantenimiento de una conexión abierta durante todo el tiempo que dure el trabajo. El conocimiento adquirido en este capítulo también nos será útil en el próximo, cuando utilicemos comandos similares para recuperar no un lector de datos sino un `DataSet`.

Generalidades sobre los comandos

Abierta la conexión con el origen de datos, todas las operaciones se efectúan mediante la ejecución de comandos. En el quinto capítulo, de manera muy breve,

se apuntó la existencia de la interfaz `IDbCommand`, en la cual se definen los miembros con que debería contar la clase de ejecución de comandos de cada proveedor. En la figura 8.1 se puede ver la relación existente entre esta interfaz y las clases `OdbcCommand`, `OleDbCommand`, `OracleCommand` y `SqlCommand`, todas ellas derivadas de `System.ComponentModel.Component`.

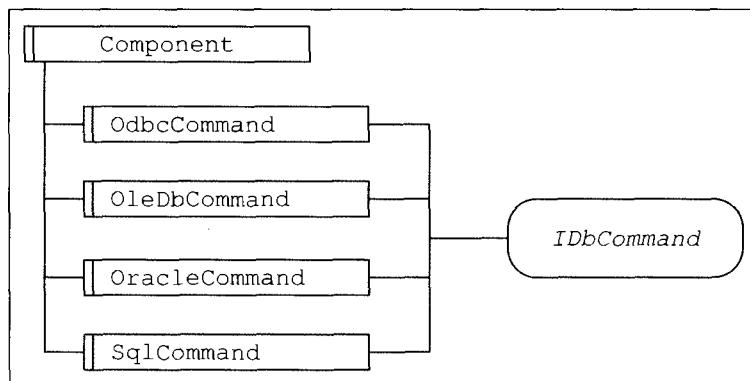


Figura 8.1. Las clases `Command` de cada proveedor implementan los miembros de la interfaz `IDbCommand`

Mediante el método `CreateCommand()`, con el que cuentan todas las clases `Connection` al estar definido en la interfaz `IDbConnection`, es posible crear un comando vacío asociado a una conexión ya existente. Ese comando será del tipo que corresponda al proveedor ADO.NET empleado, pero la referencia devuelta es de tipo `IDbCommand` y, por tanto, puede almacenarse en una variable de ese tipo.

Asociación entre comando y conexión

Si creamos un comando mediante el método `CreateCommand()`, según acaba de decirse, la asociación entre comando y conexión ya se habrá establecido automáticamente. `CreateCommand()` no toma parámetro alguno, por lo que el comando en sí deberá establecerse posteriormente, mediante las propiedades `CommandType` y `CommandText` tratadas más adelante.

```
 IDbCommand Comando = ConnOleDb.CreateCommand()
```

En el capítulo previo, con el fin de crear un lector de datos, vimos cómo creábamos un comando directamente con el constructor de una de las clases `Command`, concretamente con `OleDbCommand`. Dicho constructor acepta un `OleDbConnection` como segundo parámetro, asociando el comando con la conexión. Lógicamente, ésta deberá haberse creado previamente.

```
 Dim Comando As New OleDbCommand(Texto, ConnOleDb)
```

Otra posibilidad es que creamos el comando sin facilitar ese segundo parámetro, sólo con el texto del comando a ejecutar:

```
Dim Comando As New OleDbCommand(Texto)
```

En este caso, antes de llamar a ninguno de los métodos de ejecución, habría que asignar la conexión a la propiedad `Connection`. También podemos leer esta propiedad, al obtener un comando, para así poder acceder a los parámetros de la conexión. Dicha propiedad es de tipo `IDbConnection` en la interfaz `IDbCommand`, pero en la implementación de cada proveedor el tipo es el exclusivo de ese proveedor por lo que, en la práctica, no podríamos asignar a la propiedad `Connection` de un comando una conexión de tipo distinto.

Definición del comando a ejecutar

Dos de las propiedades más importantes de `IDbCommand` son `CommandType` y `CommandText`. Con ellas se define el tipo de comando y el texto, respectivamente, que va a enviarse al origen de datos para ser procesado.

El valor predeterminado de `CommandType` es `CommandType.Text`, indicando que el contenido de `CommandText` es una sentencia SQL. Si es éste el caso, basta con facilitar la cadena con la sentencia al constructor del objeto `Command` que vamos a usar.

Los otros dos posibles valores de `CommandType` son `CommandType.StoredProcedure` y `CommandType.TableDirect`. El primero se usa cuando es necesario ejecutar un procedimiento almacenado, facilitando el nombre de éste en la propiedad `CommandText`. El segundo permite recuperar directamente el contenido de una o más tablas, simplemente facilitando sus nombres en `CommandText`.

En caso de que el comando, ya sea procedimiento almacenado o sentencia SQL, cuente con parámetros de entrada, se utilizará la propiedad `Parameters` para acceder a la colección de parámetros y, mediante el método `Add()`, se añadirán los pertinentes, siempre antes de ejecutar.

Nota

Puede crearse un parámetro y, al tiempo, añadirlo a la colección de parámetros del comando mediante el método `CreateParameter()` de éste.

Ejecución del comando

Una vez tenemos el comando preparado, podemos ejecutarlo recurriendo a uno de los tres métodos con que cuenta `IDbCommand` o bien a algún método específico del proveedor que estemos utilizando.

Estos tres métodos son los siguientes:

- `ExecuteReader()`: Es adecuado si ejecutamos una consulta SQL, vamos a abrir una tabla o ejecutar un procedimiento almacenado que devuelve un conjunto de resultados. Devuelve un `DataReader` del que podemos recuperar la información.
- `ExecuteScalar()`: Se utiliza para recuperar sólo el dato devuelto en la primera fila de la primera columna del conjunto de resultados. Es el método a usar en caso de que ejecutemos un procedimiento almacenado o sentencia SQL que devuelve sólo un valor, no un conjunto de resultados.
- `ExecuteNonQuery()`: Como su propio nombre indica, su finalidad es ejecutar comandos que no devuelven resultados, por ejemplo sentencias o procedimientos almacenados que manipulan datos. El único valor devuelto por `ExecuteNonQuery()` es el número de filas afectadas por la operación.

De los tres procedimientos, el único que toma parámetros, de forma opcional, es `ExecuteReader()`. Se trata de un solo parámetro que determina el comportamiento del comando al ejecutarse. En el capítulo previo, por ejemplo, utilizábamos el valor `CommandBehavior.SchemaOnly` para recuperar sólo información de esquema, no datos.

Es posible emplear cualquier otro de los valores de la enumeración `CommandBehavior`, resumidos en la tabla 8.1. Si no se facilitan parámetros se asume el valor `CommandBehavior.Default`.

Tabla 8.1. Elementos de la enumeración `CommandBehavior`

Elemento	Comentario
<code>CloseConnection</code>	Cerrar la conexión utilizada para ejecutar el comando cuando se cierre el <code>DataReader</code>
<code>Default</code>	La ejecución de la consulta puede retornar múltiples conjuntos de resultados
<code>KeyInfo</code>	Se devolverá tan sólo información de la clave primaria
<code>SingleResult</code>	Tan sólo se devolverá un conjunto de resultados
<code>SingleRow</code>	Tan sólo se devolverá una fila de datos del conjunto de resultados
<code>SchemaOnly</code>	Se devolverá sólo información de esquema del conjunto de datos

El valor de retorno de `ExecuteReader()` es un `DataReader`, el de `ExecuteNonQuery()` un `Integer` y el de `ExecuteScalar()` un `Object` con el dato obtenido que podemos convertir al tipo que interese.

Los proveedores `SqlClient` y `OracleClient` disponen, además de los ya mencionados, de otros métodos `Execute` asociados a sus comandos específicos. En el caso `SqlCommand`, podemos usar el método `ExecuteXmlReader()` para obtener un `XmlDataReader` en lugar de un `SqlDataReader`. En el caso de `OracleCommand`, tenemos a nuestra disposición los métodos específicos `ExecuteNonQuery()` y `ExecuteScalar()` que, básicamente, funcionan como `ExecuteNonQuery()` y `ExecuteScalar()`, si bien retornan un identificador único de fila en caso de que se usen para afectar a una fila de datos que cuente con este elemento.

Lectura de los datos

Recuperar los datos devueltos por `ExecuteScalar()` o `ExecuteNonQuery()` no requiere ninguna explicación adicional, pero el caso de `ExecuteReader()` es distinto. Este método devolverá una referencia a un objeto `XXXDataReader` que, como se indicó en el quinto capítulo, implementa las interfaces `IDataReader` e `IDataRecord`. En realidad, la interfaz `IDataReader` está derivada de `IDataRecord` y, por tanto, hereda todos los miembros cuya implementación es obligada en las clases específicas de cada proveedor. En la figura 8.2 puede ver la relación entre estos elementos.

En la figura 5.2 del quinto capítulo encontrará un esquema de las cuatro clases `DataReader`, así como la indicación de los miembros más importantes de `IDataReader` e `IDataRecord`, descritos en el punto *Acceso a filas de datos* de dicho capítulo. Con esa información tenemos suficiente, en principio, para comenzar a recuperar datos desde cualquiera de los orígenes de que disponemos.

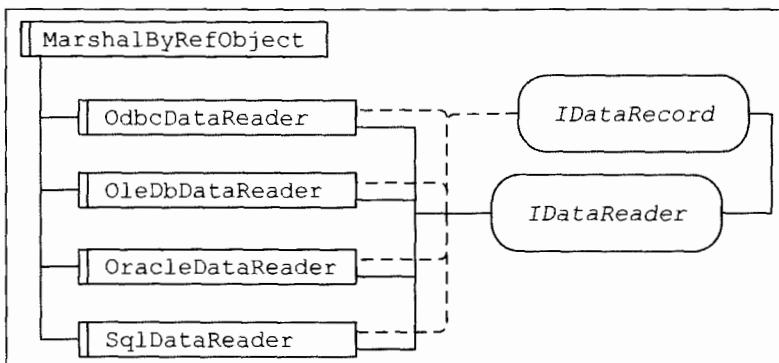


Figura 8.2. Las clases `DataReader` de cada proveedor implementan las interfaces `IDataRecord` e `IDataReader`

Recuperar el contenido de una tabla

Partamos, ya en la práctica, con uno de los supuestos más simples: recuperar todo el contenido de una tabla. Con este fin ejecutaríamos un comando, en este caso un OleDbCommand, con el valor CommandType.TableDirect en la propiedad CommandType y el nombre de la tabla en CommandText. Ejecutado el comando, con el método ExecuteReader(), no tendríamos más que recuperar las filas en el interior de un bucle controlado por el valor de retorno del método Read() del OleDbDataReader.

Suponiendo que deseamos mostrar el contenido de las tablas de nuestra base de datos Access, podríamos servirnos del código siguiente:

```
Imports System.Data.OleDb

Module Module1

    Sub Main()
        ' Mostraremos el contenido de ambas tablas
        MuestraTabla("Editoriales")
        MuestraTabla("Libros")
    End Sub

    Sub MuestraTabla(ByVal NombreTabla As String)
        ' Cadena de conexión a la base de datos Access
        Dim Conexion As New OleDbConnection(
            "Provider=Microsoft.Jet.OLEDB.4.0; " &
            "Data Source=\PBddVisualBasicNET\Cap_03\Libros.mdb")

        ' Creamos un comando asociado con la conexión
        Dim Comando As OleDbCommand = Conexion.CreateCommand()

        Conexion.Open() ' Abrimos la conexión
        ' Indicando el tipo y el nombre de la tabla
        Comando.CommandType = CommandType.TableDirect
        Comando.CommandText = NombreTabla

        Dim Lector As OleDbDataReader = Comando.ExecuteReader()

        Dim Indice As Byte
        ' Recorremos la lista de columnas
        For Indice = 0 To Lector.FieldCount - 1
            ' para mostrar su nombre
            Console.WriteLine(Lector.GetName(Indice) & vbTab)
        Next
        Console.WriteLine(vbCrLf & New String("=", 60))

        ' Mientras haya datos en la tabla
        While Lector.Read()
            ' vamos mostrando el contenido
            For Indice = 0 To Lector.FieldCount - 1
                Console.WriteLine(Lector.GetValue(Indice) & vbTab)
            Next
        End While
    End Sub
End Module
```

```

    Console.WriteLine()
End While

Lector.Close() ' Cerramos el lector
Conexion.Close() ' Cerramos la conexión.

Console.WriteLine(vbCrLf & vbCrLf)
End Sub

End Module

```

La cadena de conexión es la misma utilizada en ejemplos previos. Creamos un `OleDbCommand` a partir del `OleDbConnection`, mediante el método `CreateCommand()`, estableciendo a continuación el tipo de comando y el texto. El nombre de la tabla se recibe como parámetro desde el método `Main()`, facilitando así la visualización de las dos tablas de la base de datos.

A continuación ejecutamos el comando y obtenemos el `OleDbDataReader`. Sirviéndonos de su propiedad `FieldCount` recorremos todas las columnas del conjunto de datos, mostrando el nombre de cada una de ellas. El siguiente paso es la lectura propiamente dicha, invocando al método `Read()` en un bucle que finalizará cuando no queden más filas a leer. Por cada fila mostramos el valor de todas sus columnas.

Finalmente, cerramos el `OleDbDataReader` y la conexión. Al ejecutar el programa debería obtener un resultado similar al mostrado en la figura 8.3. Simplemente modificando la cadena de conexión, y añadiendo el carácter \$ tras el nombre de las tablas, podría efectuar la misma operación sobre el documento Excel, puesto que éste también se lee con el proveedor OLE DB.

	IPLibro	ISBN	Título	Autor	Editorial	Precio
1	1-893115-94-1	User Interface Design for Programmers	Joel Spolsky	3	31	
2	84-415-1136-5	SQL Server 2000	Francisco Charte	1	10,75	
3	04-415-1324-4	Guía práctica para usuarios jBuilder 7	Francisco Charte	1		
4	94-415-1392-9	Programación con Visual CM .NET	Francisco Charte	1	39	18,75
5	94-415-1376-7	Programación con Visual Studio .NET	Francisco Charte/Jorge Serrano	1		40
6	84-415-1351-1	Programación con Visual Basic .NET	Francisco Charte	1	39	
7	84-415-1290-6	Guía práctica para usuarios de Visual Basic .NET	Francisco Charte	1		
8	84-415-1291-4	Guía práctica para usuarios de Visual Studio .NET	Francisco Charte	1		1
9	84-415-1261-2	Programación con Delphi 6 y Kylix	Francisco Charte	1	32,26	
10	84-415-1255-0	Guía práctica para usuarios de Delphi 6	Francisco Charte	1		18,52
11	84-415-1230-2	Manual avanzado Excel 2002	Francisco Charte	1	21,04	
12	84-415-1322-7	Guía práctica para usuarios de Excel 2000	Francisco Charte/Jesús Luque	1		1
13	94-415-1132-2	Guía práctica para usuarios de Kylix	Francisco Charte		10,52	
14	84-415-1135-4	Introducción a la programación	Francisco Charte	1	24,04	
15	84-7615-234-5	Manual del microprocesador 08036	Chris H.Pappas&William H.Harary	111	2	4

Press any key to continue...

Figura 8.3. Contenido de las tablas de Microsoft Access

Nota

El uso de `CommandType.TableDirect`, entregando el nombre de la tabla en la propiedad `CommandText`, sería equivalente a ejecutar una sentencia `SELECT * FROM NombreTabla`. Aunque en este ejemplo se ha mostrado cómo utilizar la posibilidad de recuperar el contenido de una tabla con ese sistema, es mucho más compatible la segunda opción, ya que no todos los proveedores ADO.NET contemplan el uso de `CommandType.TableDirect`.

Varios conjuntos de datos

Al ejecutar un procedimiento almacenado, o un lote de sentencias SQL, es posible que la ejecución de un comando devuelva varios conjuntos de datos como resultado. En principio el `DataReader` apuntaría al primero de ellos, facilitando el acceso a los demás mediante el método `NextResult()`. Éste puede ser utilizado como condicional de un bucle, en caso de que deseemos recorrer todos los conjuntos devueltos, pero teniendo en cuenta que, a diferencia de `Read()`, que con la primera llamada nos lleva a la primera fila, la primera llamada a `NextResult()` nos llevaría directamente al segundo conjunto de datos.

Veamos cómo conseguir un efecto similar al del ejemplo del punto anterior, recuperando todo el contenido de las dos tablas, pero en este caso utilizando un lote de sentencias SQL sobre SQL Server, gestionando los conjuntos de datos devueltos con los métodos del `SqlDataReader`.

El código sería el mostrado a continuación:

```
Imports System.Data.SqlClient

Module Module1

Sub Main()
    ' Cadena de conexión a la base de datos sql server
    Dim Conexion As New SqlConnection(
        "Data Source=inspiron; Initial Catalog=Libros;" &
        "User ID=sa; Password=")

    ' Creamos un comando asociado con la conexión
    Dim Comando As SqlCommand = Conexion.CreateCommand()

    Conexion.Open() ' Abrimos la conexión
    ' Facilitando un lote de varias sentencias
    Comando.CommandText = "SELECT * FROM Editoriales;" &
        "SELECT * FROM Libros; SELECT COUNT(*) FROM LIBROS"

    ' Ejecutamos el comando
    Dim Lector As SqlDataReader = Comando.ExecuteReader()

    Dim Indice As Byte
```

```

Do
    ' Recorremos la lista de columnas
    For Indice = 0 To Lector.FieldCount - 1
        ' para mostrar su nombre
        Console.WriteLine(Lector.GetName(Indice) & vbTab)
    Next
    Console.WriteLine(vbCrLf & New String(" ", 60))

    ' Iniciamos nueva fila en la tabla
    While Lector.Read()
        ' vamos mostrando el contenido
        For Indice = 0 To Lector.FieldCount - 1
            Console.WriteLine(Lector.GetValue(Indice) & vbTab)
        Next
        Console.WriteLine()
    End While
    Console.WriteLine(vbCrLf & vbCrLf)
    ' pasar al siguiente grupo de resultados
Loop While Lector.NextResult()

Lector.Close() ' Cerramos el lector
Conexion.Close() ' Cerramos la conexión
End Sub

End Module

```

Observe que, salvo la conexión y ejecución del comando, la mayor parte del código se encuentra en el interior del bucle Do/Loop While Lector.NextResult(). Así procesaremos los tres conjuntos de datos obtenidos. Dentro de éste tenemos otro bucle, el equivalente al del ejemplo anterior, para recorrer todas las filas de cada conjunto. El resultado, como se aprecia en la figura 8.4, es prácticamente idéntico. A diferencia del programa anterior, sin embargo, tan sólo hemos conectado con el origen una vez y tan sólo hemos ejecutado un comando.

Nota

No todos los proveedores ADO.NET contemplan la posibilidad de ejecutar lotes de sentencias SQL, como se ha hecho en este ejemplo con SqlClient.

Ejecución de sentencias de selección

Mediante un objeto Command es posible ejecutar cualquier tipo de comando SQL, entre ellos, por supuesto, los comandos de selección. En el ejemplo anterior nos hemos limitado a obtener todo el contenido de dos tablas y un contador, pero igualmente podríamos ejecutar una sentencia más compleja, por ejemplo combinando tablas, ordenando los resultados, etc. Además, en los ejemplos anteriores enumeramos todas las columnas de todas las filas, cuando lo habitual es que sepamos qué columnas de datos vamos a recuperar y las tratemos según interese.



Figura 8.4. Contenido de los múltiples conjuntos de datos obtenidos

Suponga que desea obtener el título de cada uno de los libros que hay en la tabla *Libros*, junto con el nombre de su editorial y el precio. Estos datos podría utilizarlos para generar un informe impreso, un documento HTML o cualquier otro tipo de resultado.

En el código siguiente nos limitamos a mostrarlos por consola, pero el proceso sería básicamente el mismo.

```

Imports System.Data.OleDb

Module Module1
    Sub Main()
        ' Primero creamos la conexión con la base de datos
        Dim Conexion As New OleDbConnection(
            "Provider=Microsoft.Jet.OLEDB.4.0; " &
            "Data Source=\PBddVisualBasicNET\Cap_03\Libros.xls; " &
            "Extended Properties='Excel 8.0; HDR=Yes'")
        ' Segundo creamos el comando que ejecutará la consulta
        Dim Comando As OleDbCommand = Conexion.CreateCommand()
        Conexion.Open() ' Abrir la conexión
        ' y finalizar la sentencia de búsqueda
        Comando.CommandText = "SELECT L.Titulo, E.Nombre, L.Precio " &
            "FROM [Editoriales$] E, [Libros$] L " &
            "WHERE E.IDEditorial = L.Editorial " &
            "ORDER BY L.Titulo"
        ' Tercer paso es leer los resultados
        Dim Lector As OleDbDataReader = Comando.ExecuteReader()

```

```

' Unos encabezados
Console.WriteLine("Título".PadRight(50) & vbTab & _
    "Editorial".PadRight(25) & vbTab & "Precio" & _
    vbCrLf & New String("-", 100))
' Mientras haya datos en la tabla
While Lector.Read()
    ' mostramos cada una de las tres columnas
    Console.Write(Lector.GetString(0).PadRight(50) & vbTab)
    Console.Write(Lector.GetString(1).PadRight(25) & vbTab)
    Console.WriteLine("{0,5}", Lector.GetDouble(2))
End While

Lector.Close() ' Cerramos el lector
Conexion.Close() ' Cerramos la conexión
End Sub

End Module

```

Ya que conocemos el tipo de las columnas seleccionadas en la consulta, podemos utilizar directamente los métodos GetXXX() del DataReader, por ejemplo, GetString() para recuperar el título y nombre de la editorial y GetDouble() para leer el precio. Observe cómo en la consulta se han facilitado los nombres de las tablas entre corchetes, puesto que el carácter \$ no puede introducirse directamente en la consulta.

En la figura 8.5 puede ver el resultado de la ejecución de este ejemplo. Lógicamente, en lugar de utilizar el documento Excel como origen puede usar cualquier otro. Tan sólo tiene que utilizar la conexión adecuada, según se explicó en el sexto capítulo, y eliminar de la sentencia de consulta los corchetes y el símbolo \$ al final del nombre de cada tabla.

Título	Editorial	Precio
Guía práctica para usuarios de Delphi 6	Anaya Multimedia	10.52
Guía práctica para usuarios de Excel 2002	Anaya Multimedia	10.52
Guía práctica para usuarios de Kylix	Anaya Multimedia	10.52
Guía práctica para usuarios de Visual Basic .NET	Anaya Multimedia	10.75
Guía práctica para usuarios de Visual Studio .NET	Anaya Multimedia	10.52
Guía práctica para usuarios JBuilder 7	Anaya Multimedia	10.75
Introducción a la programación	Anaya Multimedia	21.04
Manual avanzado Excel 2002	Anaya Multimedia	21.04
Manual del microprocesador 80386	McGraw Hill	40
Programación con Delphi 6 y Kylix	Anaya Multimedia	37.26
Programación con Visual Basic .NET	Anaya Multimedia	39
Programación con Visual C# .NET	Anaya Multimedia	39
Programación con Visual Studio .NET	Anaya Multimedia	40
SQL Server 2000	Anaya Multimedia	10.50
User Interface Design for Programmers	Apresu	31
Press any key to continue...		

Figura 8.5. Resultado de la consulta SQL sobre el documento Excel

Nota

Al acceder a Excel, el proveedor deduce el tipo de los datos a partir del contenido de las celdillas, como se indicó anteriormente. Por eso el precio se trata como un Double. Al conectar con un origen distinto podría tener que cambiar GetDouble() por GetDecimal().

Sentencias con parámetros

Las sentencias SQL usadas en los ejemplos anteriores son constantes, lo que significa que siempre se ejecutan igual y obtienen los mismos resultados. Una sentencia, no obstante, puede contener parámetros cuyo valor se desconoce en el momento de expresar la propia consulta, lo que se conoce normalmente como parámetros sustituibles. Observe, por ejemplo, la siguiente consulta:

```
SELECT * FROM Libros  
WHERE Editorial=:CodEditorial
```

En el condicional se compara la columna Editorial de la tabla Libros con un valor, CodEditorial, que habrá que definir en algún momento antes de ejecutar la sentencia. El uso de este tipo de parámetros puede permitirnos adaptar nuestra consulta a las indicaciones del usuario sin, por ello, tener que reconstruir la sentencia SQL según cada caso.

Nota

La sintaxis para indicar la existencia de un parámetro difiere de un proveedor a otro. En la sentencia anterior se ha precedido CodEditorial con dos puntos, sintaxis adecuada para OracleClient. El proveedor SqlClient, por el contrario, usa el carácter @ para denotar la existencia de un parámetro sustituible.

Los objetos que implementan IDbCommand disponen de la propiedad Parameters, comentada brevemente antes, que mantiene la lista de parámetros necesarios para la ejecución del comando. En principio esa lista está vacía, pudiendo añadirse los parámetros que se necesiten mediante el método Add(). Éste necesita como argumento el objeto que representa al parámetro, un objeto que será dependiente del proveedor, por ejemplo SqlParameter u OracleParameter.

Una alternativa es llamar al método CreateParameter() del objeto Command que estemos usando. Éste devuelve un objeto del tipo adecuado, no teniendo más que asignar, usando sus propiedades, el nombre y valor del parámetro, datos que usando el método indicando antes se entregarían al constructor de la clase Parameter adecuada.

Suponga que quiere crear una consulta de libros pertenecientes a una cierta editorial, permitiendo al usuario que sea él quien seleccione dicha editorial. Asumiendo que utilizaremos la base de datos creada anteriormente en Oracle, el código necesario sería el siguiente:

```
Imports System.Data.OracleClient

Module Module1

Sub Main()
    ' Cadena de conexión a la base de datos SQL Server
    Dim Conexion As New OracleConnection(
        "Data Source=Libros; User ID=scott; Password=tiger")

    ' Creamos un comando asociado con la conexión
    Dim Comando As OracleCommand = Conexion.CreateCommand()

    Conexion.Open() ' abrimos la conexión
    ' Facilitando una consulta con parámetros
    Comando.CommandText = "SELECT ISBN, Titulo FROM Libros " & _
        "WHERE Editorial=:CodEditorial"

    Dim NumEditorial As Integer
    ' Solicitamos el código de editorial por teclado
    Console.WriteLine("Introduzca un código de editorial : ")
    NumEditorial = Integer.Parse(Console.ReadLine())

    ' Con ese código creamos un OracleParameter
    ' y lo añadimos a la colección de parámetros
    Comando.Parameters.Add(New OracleParameter( _
        "CodEditorial", NumEditorial))

    ' Ejecutamos el comando
    Dim Lector As OracleDataReader = Comando.ExecuteReader()

    Console.WriteLine("ISBN".PadRight(13) & vbTab & _
        "Titulo" & vbCrLf & New String("=", 70))

    ' Mientras haya datos en la tabla
    While Lector.Read()
        Console.WriteLine(Lector("ISBN") & vbTab & Lector("Titulo"))
    End While

    Lector.Close() ' Cerramos el lector
    Conexion.Close() ' Cerramos la conexión
End Sub

End Module
```

Observe cómo se define el comando y, tras solicitar el código de editorial por la consola, se añade un elemento a la colección Parameters creando un nuevo OracleParameter. El resultado de ejecutar el programa, uno de los posibles, sería el de la figura 8.6. Podría añadir, antes de solicitar el código de editorial al usuario,

el código necesario para mostrar el código y nombre de cada una de ellas, facilitando así la selección.

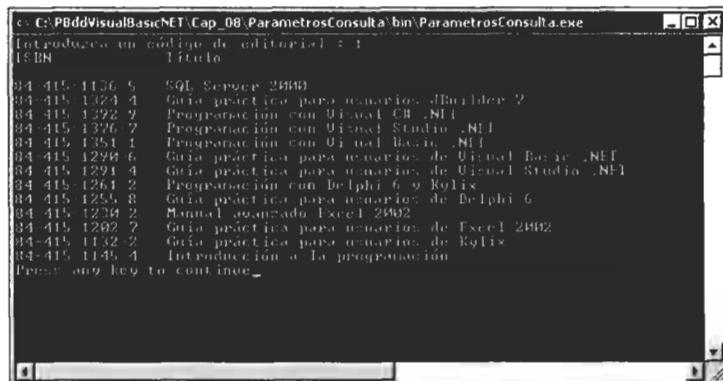


Figura 8.6. Lista de los títulos pertenecientes a la editorial seleccionada

Recuperación de un solo valor

Hasta ahora hemos usado reiteradamente el método `ExecuteReader()` para acceder a uno o varios conjuntos de datos. En ocasiones, el resultado de una sentencia de consulta será único, por ejemplo una suma, contador o algún otro cálculo. En casos así, aunque podríamos utilizar `ExecuteReader()`, resulta más fácil y eficiente utilizar el método `ExecuteScalar()`. Éste devuelve directamente el resultado, sin necesidad de recorrer filas ni columnas.

Partiendo del ejemplo del punto anterior, modifique la consulta dejándola así:

```
Comando.CommandText = "SELECT COUNT(*) FROM Libros " & _
    "WHERE Editorial=:CodEditorial"
```

A continuación se solicita el código de editorial al usuario e introduce como parámetro en el comando, que ahora ejecutaremos así:

```
Dim Resultado As Integer = Comando.ExecuteScalar()
Console.WriteLine("Esa editorial tiene {0} títulos", Resultado)
```

Hemos eliminado la declaración del `DataReader` y la llamada al método `Close()`. Si ejecuta ahora el programa, e introduce el código de una editorial, verá aparecer el número de títulos que corresponden a ese código.

Manipulación de datos

Aunque el título de este capítulo hace referencia expresa a la recuperación de datos, y a este tema se dedica la mayor parte del mismo, hay que apuntar que un

comando preparado en un objeto `Command` puede contener cualquier tipo de sentencia SQL, incluidas las que añaden filas, modifican las existentes o las eliminan.

Este tipo de sentencias no devuelven conjuntos de resultados, sino un entero que indica el número de filas que se han visto afectadas por la operación realizada. Por ello se ejecutan con el método `ExecuteNonQuery()` en vez de `ExecuteReader()`.

Conociendo la sintaxis SQL del origen con el que vayamos a conectar, efectuar cualquier operación de manipulación de datos no implica mayor complejidad. El siguiente ejemplo añade una fila de datos a la hoja `Libros` del documento Excel. La operación se efectuaría de manera análoga en cualquier otro origen y, de forma similar, podrían modificarse o eliminarse filas.

```
Imports System.Data.OleDb

Module Module1
    Sub Main()
        ' Cadena de conexión al documento Excel
        Dim Conexion As New OleDbConnection(
            "Provider=Microsoft.Jet.OLEDB.4.0; " &
            "Data Source=\PBddVisualBasicNET\Cap_03\Libros.xls; " &
            "Extended Properties='Excel 8.0; HDR=Yes'")

        ' Creamos un comando asociado con la conexión
        Dim Comando As OleDbCommand = Conexion.CreateCommand()

        Conexion.Open() ' abrimos la conexión
        ' Y definimos la sentencia de inserción
        Comando.CommandText = "INSERT INTO [Libros$] " &
            "VALUES (16, '1-893115-50-X', " &
            "'Wireless Java', 'Jonathan Knudsen', 3, 34.95)"

        ' Ejecutamos el comando
        Dim Resultado As Integer = Comando.ExecuteNonQuery()

        ' y mostramos el número de filas afectadas
        Console.WriteLine("{0} filas afectadas", Resultado)

        Conexion.Close() ' Cerramos la conexión
    End Sub

End Module
```

Para ver el resultado puede ejecutar alguno de los ejemplos previos, recuperando el contenido del documento y mostrándolo por la consola, o bien abrir directamente el documento en Microsoft Excel para observar el cambio.

Otras operaciones

Dado que con un comando podemos ejecutar cualquier sentencia SQL sobre el origen de datos, podemos virtualmente extraer y manipular todo su contenido.

Sin embargo, en ocasiones el origen de datos, en caso de ser un RDBMS, mantiene estructuras *prefabricadas* como las vistas y procedimientos almacenados, cuyo objetivo es facilitar nuestro trabajo al efectuar operaciones relativamente complejas, eximiéndonos a nosotros de codificarlas en SQL.

En las bases de datos que estamos empleando a modo de ejemplo, concretamente en las de SQL Server, Oracle e InterBase, introdujimos la definición de una vista y dos procedimientos almacenados. A continuación va a utilizarse el proveedor `SqlClient` para operar con SQL Server, pero el procedimiento sería el mismo con cualquiera de los otros orígenes, simplemente habría que cambiar de proveedor, poco más.

Recuperación de una vista

Las vistas, una vez definidas en la base de datos, no se diferencian demasiado, en cuanto a tratamiento, respecto a las tablas. Podemos consultarlas con una sentencia SQL corriente, seleccionando los datos que nos interesen. Dependiendo de la vista y del RDBMS, posiblemente también pueda utilizarse para manipular la información.

En nuestra base de datos SQL Server tenemos definida una vista, llamada `LibrosEditorial`, que devuelve una lista de las editoriales existente junto con los títulos que le corresponden a cada una, ordenando la lista alfabéticamente por el nombre de la editorial. Para recuperar esta vista, ejecutando la sentencia SQL y obteniendo su resultado, crearemos una nueva aplicación de consola con el código siguiente:

```
Imports System.Data.SqlClient

Module Module1

    Sub Main()
        ' Cadenas de conexión a la base de datos SQL Server
        Dim Conexion As New SqlConnection(
            "Data Source=inspiron; Initial Catalog=Libros; " &
            "User ID=sa; Password=")

        ' Creamos un comando asociado con la conexión
        Dim Comando As SqlCommand = Conexion.CreateCommand()

        Conexion.Open() ' abrimos la conexión
        ' Seleccionamos la vista
        Comando.CommandText = "SELECT * FROM LibrosEditorial"

        ' Ejecutamos el comando
        Dim Lector As SqlDataReader = Comando.ExecuteReader()

        Dim Indice As Byte

        ' Un encabezado
```

```

Console.WriteLine("Nombre".PadRight(20) & vbTab & _
    "Título".PadRight(50) & vbTab & "Precio" & _
    vbCrLf & New String("=", 90))

' Muestra los datos en la consola
While Lector.Read()
    ' Muestra los datos
    Console.WriteLine("{0,-20}" & vbTab & "{1,-50}" &
        vbTab & "{2,5}", Lector("Nombre"), -
        Lector("Título"), Lector("Precio"))
End While

Lector.Close() ' Cierra el lector
Conexion.Close() ' Cierra la conexión
End Sub

End Module

```

Como puede ver, la sentencia SQL asignada a CommandText bien podría ser la de consulta a una tabla llamada `LibrosEditorial`. Sabemos, sin embargo, que tal tabla no existe, puesto que los datos de las editoriales y los libros se encuentran separados. `LibrosEditorial` es una vista que, al ejecutarse, devuelve la lista de datos que puede verse en la figura 8.7.

Nombre	Título	Precio
Apress	User Interface Design for Programmers	31
Anaya Multimedia	SQL Server 2000	10.75
Anaya Multimedia	Guía práctica para usuarios JBuilder 7	10.75
Anaya Multimedia	Programación con Visual C# .NET	39
Anaya Multimedia	Programación con Visual Studio .NET	40
Anaya Multimedia	Programación con Visual Basic .NET	39
Anaya Multimedia	Guía práctica para usuarios de Visual Basic .NET	10.75
Anaya Multimedia	Guía práctica para usuarios de Visual Studio .NET	10.52
Anaya Multimedia	Programación con Delphi 6 y Kylix	32.26
Anaya Multimedia	Guía práctica para usuarios de Delphi 6	10.52
Anaya Multimedia	Manual avanzado Excel 2002	21.04
Anaya Multimedia	Guía práctica para usuarios de Excel 2002	10.52
Anaya Multimedia	Guía práctica para usuarios de Kylix	10.52
Anaya Multimedia	Introducción a la programación	24.04
McGraw-Hill	Manual del microprocesador 80386	40

Figura 8.7. Conjunto de datos devuelto por la vista `LibrosEditorial`

Ejecución de un procedimiento almacenado

Ejecutar un procedimiento almacenado es, si cabe, más fácil que abrir una vista. Básicamente tenemos que dar tres pasos, asumiendo que tenemos la conexión ya preparada:

- Asignar el valor `CommandType.StoredProcedure` a la propiedad `CommandType` del comando.
- Asignar el nombre del procedimiento almacenado a la propiedad `CommandText`.
- Añadir a la propiedad `Properties` los parámetros que el procedimiento almacenado pudiera necesitar.

Hecho esto, ejecutaríamos el comando con `ExecuteReader()`, `ExecuteScalar()` o `ExecuteNonQuery()`, según devuelva un conjunto de datos, un solo valor o nada. En este último caso, como ya sabe, obtendría el número de filas afectadas por la operación. Hemos usado previamente los tres métodos, por lo que no necesitamos saber nada nuevo para ejecutar uno de los procedimientos almacenados que tenemos en nuestra base de datos.

Tomando como base el ejemplo previo, mantenemos la primera parte, definiendo la conexión, creando el comando y abriendo la conexión, y modificamos a partir de la asignación de la sentencia SQL, introduciendo el código siguiente:

```

Comando.CommandType = CommandType.StoredProcedure
Comando.CommandText = "NumTitulosEditorial"

' Ejecutamos el comando
Dim Lector As SqlDataReader = Comando.ExecuteReader()

Dim Indice As Byte

' Un encabezado
Console.WriteLine("Nombre".PadRight(25) & vbTab &
    "Titulos" & vbCrLf & New String("=", 50))

' Mientras haya datos en la tabla
While Lector.Read()
    ' vamos mostrando el contenido
    Console.WriteLine("{0,-25}" & vbTab &
        vbTab & "{1,5}", Lector("Nombre"), _
        Lector("NumTitulos"))
End While

```

En este caso usamos el procedimiento almacenado `NumTitulosEditorial` para obtener un conjunto de datos formado por el nombre de cada editorial y el número de títulos con que cuenta. La lectura de ese conjunto de datos, como puede verse, no difiere de lo que ya conocíamos.

Nota

Intente ejecutar el procedimiento almacenado `NumTitulos`, facilitando los parámetros necesarios y recogiendo el resultado único que devuelve.

The screenshot shows a Windows command-line interface window titled 'C:\PBddVisualBasicNET\Cap_08\ProcedimientosAlmacenados\bin\ProcedimientosAlmacenados.exe'. The window displays the following table:

NOMBRE	TÍTULOS
Anaya Multimedia	13
Opress	1
McGraw-Hill	1

At the bottom of the window, the text 'Press any key to continue...' is visible.

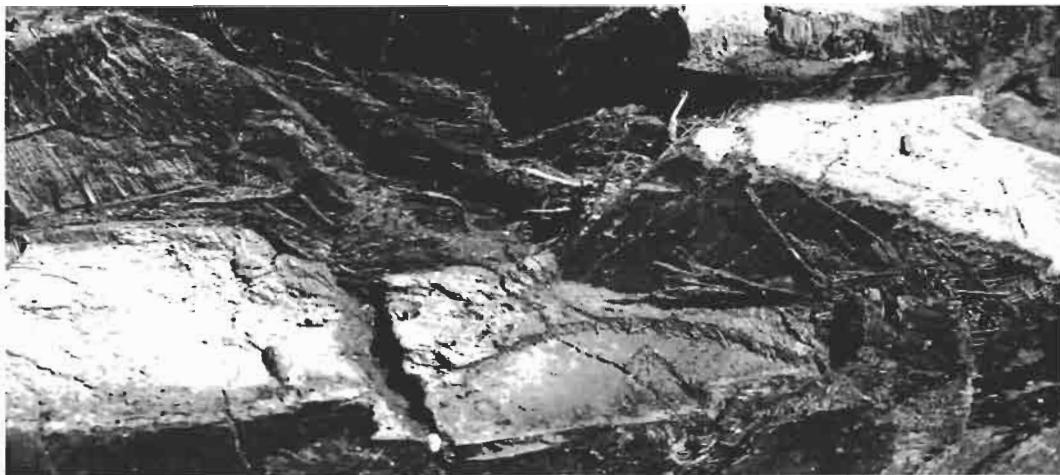
Figura 8.8. Conjunto de resultados devuelto por la ejecución del procedimiento almacenado

Resumen

Al finalizar este capítulo ya está familiarizado con la definición y ejecución de comandos, así como en el uso de las clases `DataReader` para recorrer los conjuntos de datos obtenidos a partir de esos comandos. Utilizando esta técnica puede recuperar datos y ejecutar cualquier tipo de sentencia, incluidas las de actualización, vistas y procedimientos almacenados.

Debe tener en cuenta que mientras trabaja con un `DataReader` está manteniendo abierta la conexión con la base de datos, por lo que el proceso de la información no debería requerir demasiado tiempo a fin de evitar que esa vía esté abierta indefinidamente. En los ejemplos mostrados en este capítulo ha visto cómo se abría la conexión, leían los datos para mostrarlos en consola y se cerraba la conexión.

En caso de que necesitemos alguna operativa más compleja sobre los datos, por ejemplo una visualización en una rejilla para permitir que el usuario los edite, los `DataReader` no son el medio más apropiado. En su lugar deberíamos utilizar los conjuntos de datos que vamos a conocer en el capítulo siguiente.



9

Conjuntos de datos

El sistema de trabajo mostrado en el capítulo anterior, utilizando comandos y lectores de datos, tiene sus aplicaciones concretas, pero presenta algunos inconvenientes si intenta usarlo como sistema general para todas las aplicaciones de acceso a datos. Estará obligado a mantener una conexión persistente con el servidor todo el tiempo que esté operando sobre los datos. Tendrá que conocer la sintaxis específica de cada origen de datos, por ejemplo a la hora de componer sentencias SQL, puesto que no cuenta con ninguna ayuda al respecto. Además, las tareas de edición de los datos prácticamente tendrá que codificarlas a mano, recuperando las filas y después generando las sentencias necesarias para enviar los cambios al origen.

ADO.NET dispone de un conjunto de clases, alojadas en su mayor parte en el ámbito `System.Data`, que pueden facilitarnos mucho nuestro trabajo diario. Éstas se introdujeron brevemente en el quinto capítulo, concretamente en el punto *Conjuntos de datos* y los siguientes. Las más importantes de ellos son `DataSet` y los distintos `DataAdapter`, cuyo uso vamos a conocer en este capítulo con diversos ejemplos.

Generalidades sobre conjuntos de datos

A diferencia de la mayoría de elementos que hemos conocido en los tres capítulos previos, que se caracterizan por implementar alguna interfaz común pero con

implementaciones específicas para cada proveedor, los conjuntos de datos tan sólo cuentan con una clase: `DataSet`. En esta clase pueden existir, según se apunta en el quinto capítulo, múltiples tablas, con sus filas y columnas, restricciones y relaciones.

Un conjunto de datos puede obtenerse a partir de un origen de datos, mediante un adaptador, o bien definirse mediante código. Independientemente de esto, el contenido del conjunto de datos puede almacenarse y recuperarse localmente, existiendo también, como es lógico, la posibilidad de resolver los cambios con el origen de datos del que se extrajo la información.

Si bien en principio no existen clases derivadas de `DataSet`, nada nos impide crearlas a fin de que se ajusten a la estructura concreta de los conjuntos de datos sobre los que vamos a operar. Estas clases, derivadas de `DataSet`, serían conjuntos de datos con comprobación de tipos que facilitarían el acceso a los datos comprobando, al tiempo, que éstos sean de los tipos apropiados. Aunque podríamos crear estas clases manualmente, Visual Studio .NET dispone de asistentes que se ocupan de hacerlo. Los conocerá en un capítulo posterior, al tratar las herramientas del entorno para el trabajo con datos.

En este capítulo utilizaremos siempre conjuntos de datos genéricos, es decir, objetos de la clase `DataSet`.

Tablas y relaciones

Las dos propiedades fundamentales de un `DataSet` son `Tables` y `Relations`. Cada una de ellas mantiene una referencia a una colección de objetos, en el primer caso de la clase `DataTable` y en el segundo de `DataRelation`. La primera es importante ya que nos permite recuperar y modificar la información almacenada en el conjunto de datos, mientras que la segunda define la relación entre tablas en caso de que el conjunto esté compuesto por varias.

Cuando se llena un `DataSet` a partir de un origen de datos, mediante un adaptador, las colecciones de tablas y relaciones se definen automáticamente. Nada nos impide, sin embargo, añadir nuevos elementos a las colecciones, es decir, crear nuevas tablas y relaciones en el interior del `DataSet`. Es algo que veremos en la práctica posteriormente.

Cada tabla se compone, como ya sabe, de filas y columnas, representadas por las colecciones `Rows` y `Columns`, respectivamente, de la clase `DataTable`. Mediante la colección `Rows` podemos acceder a los datos que contienen las filas, mientras que con la colección `Columns` tenemos a nuestro alcance la información de esquema de cada columna: nombre, tipo, etc. La tabla también puede contar con una colección de objetos `Constraint` especificando las restricciones aplicables a los datos.

Mediante objetos `DataRelation`, alojados en la propiedad `Relations`, se establecen los enlaces entre las tablas que forman parte del `DataSet`. Éstas, al igual que el contenido del resto del `DataSet`, pueden recuperarse mediante un adaptador de datos o bien definirse con código.

En el quinto capítulo, en la segunda mitad, se comentaron las propiedades más importantes de los objetos `DataTable`, `DataRow`, `DataColumn` y `DataRelation`. Puede recurrir de nuevo a él si es que necesita recordar algún punto concreto.

Selección de datos

Mediante la propiedad `Rows` de cualquier `DataTable` es posible acceder a la totalidad de las filas que componen la tabla en ese momento. Utilizando el método `Select()` de la clase `DataTable`, no obstante, es posible establecer filtros de selección para recuperar un arreglo compuesto tan sólo por aquellas filas que los cumplen.

Observe que el método `Select()` no afecta a la propiedad `Rows`, es decir, no provoca que en la colección aparezcan o desaparezcan filas, sino que es el mismo método el que devuelve un arreglo de objetos `DataRow` con el resultado.

Al efectuar búsquedas con el método `Select()`, por defecto, no se distingue entre mayúsculas y minúsculas. Este comportamiento está controlado por la propiedad `CaseSensitive` del `DataSet` que, por defecto, tiene el valor `False`. Puede asignarle `True` si necesita que al establecer un filtro se distinga entre mayúsculas y minúsculas.

Nota

El contenido del arreglo devuelto por el método `Select()` son las referencias a los objetos `DataRow` de la colección `Rows` que cumplen con el filtro. No se trata de copias, sino de referencias a los mismos datos. Cualquier modificación en un `DataRow` obtenido con `Select()`, por tanto, tendrá su reflejo inmediato en la colección `Rows` del `DataTable`.

Generalidades sobre adaptadores de datos

Aunque, como se verá más adelante en este capítulo, es posible almacenar y recuperar conjuntos de datos usando archivos locales, al trabajar con objetos `DataSet` asociados a orígenes de datos, que es el caso más habitual, resultan imprescindibles los adaptadores de datos, introducidos también en el quinto capítulo. Un adaptador de datos actúa como intermediario entre el origen de datos, con dependencias de cada producto particular, y el `DataSet`, que tiene una estructura totalmente independiente.

La relación entre los distintos elementos relacionados con adaptadores de datos, interfaces, clases genéricas y clases de proveedores, es la que puede apreciarse en la figura 9.1. Como puede ver, la clase DataAdapter implementa la interfaz `IDataAdapter`, sirviendo a su vez como base para la clase genérica `DbDataAdapter`. Ésta sirve como base de `OdbcDataAdapter`, `OleDbDataAdapter`, `OracleDataAdapter` y `SqlDataAdapter`, adaptadores de datos de cada proveedor que, además, también tienen en común la implementación de la interfaz `IDbDataAdapter`.

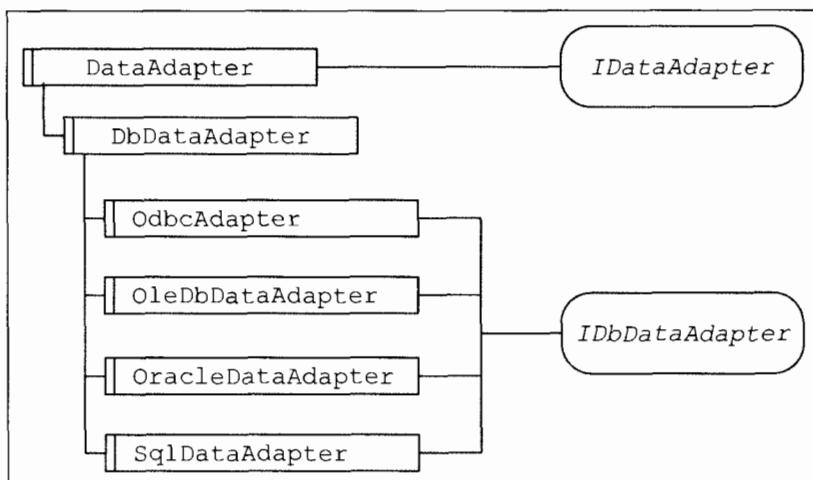


Figura 9.1. Clases e interfaces relacionadas con los adaptadores de datos

Del esquema de la figura 9.1 es fácil deducir que la clase `DataAdapter` de cualquiera de los proveedores, `OleDbAdapter` u `OracleAdapter` por ejemplo, implementa todos los miembros de las interfaces `IDataAdapter` e `IDbDataAdapter`, bien sea por sí mismo o por haberlos heredado de las clases genéricas `DataAdapter` o `DbDataAdapter`.

Nota

La clase `DbDataAdapter` es abstracta, es decir, no pueden crearse objetos directamente a partir de ella, sino que está diseñada específicamente para servir como base de otras.

Creación de un adaptador de datos

Los adaptadores de datos se crean siempre a partir de la clase `DataAdapter` específica de un proveedor dado, existiendo, en principio, las cuatro que aparecen en la figura 9.1. En el momento de la creación, es posible facilitar al constructor un

parámetro, dos o ninguno. El adaptador que obtendríamos, según el caso, tendría las siguientes características:

- **Ningún parámetro.** Se obtiene un adaptador que no está asociado a una conexión ni tiene un comando de recuperación de datos. Sería preciso asignar posteriormente un objeto `Command` a la propiedad `SelectCommand`, facilitando un comando de selección ya asociado a una conexión.
- **Un objeto `Command`.** El adaptador se crea ya asociado al comando que se entrega como parámetro.
- **Una cadena de caracteres y un objeto `Connection`.** El propio adaptador se encarga de utilizar la cadena de caracteres para crear el comando de selección, asociándolo con la conexión que se entrega como segundo parámetro.
- **Dos cadenas de caracteres.** La primera sería el comando de selección y la segunda la cadena de conexión al origen de datos. Con esto el adaptador se encargaría de crear los objetos `Connection` y `Command` del proveedor apropiado, estableciendo todas sus propiedades.

A diferencia de lo que ocurría con los lectores de datos, un adaptador no requiere que la conexión con el origen de datos esté abierta de antemano. Él mismo se ocupa de abrirla y cerrarla cuando es necesario.

Obtención de los datos

Una vez que tenemos el adaptador preparado, con su comando de selección definido, para ejecutarlo y obtener los datos en el `DataSet` no tenemos más que invocar al método `Fill()`. Éste puede tomar distintas listas de parámetros, aunque la sintaxis más habitual es la siguiente:

```
AdaptadorDatos.Fill(ConjuntoDatos)
```

`ConjuntoDatos` sería el `DataSet` destinatario de la información, en cuyo interior se crearía, con el resultado de la sentencia de selección, un `DataTable` llamado `Table`.

Si deseamos que el objeto `DataTable` tenga otro nombre, no tenemos más que facilitarlo como segundo parámetro al mismo método `Fill()`.

El comportamiento del método `Fill()` viene determinado, en parte, por el valor que tengan las propiedades `MissingMappingAction` y `MissingSchemaAction` del adaptador. La primera determina qué ocurre cuando se obtienen del origen de datos tablas o columnas que no existen aún en el `DataSet`. Los posibles valores son los enumerados en la tabla 9.1.

Mediante la segunda se especifica qué hacer con la nueva información de esquema que pudiera derivarse del origen, pudiendo tomar uno de los cuatro valores indicados en la tabla 9.2.

Tabla 9.1. Valores posibles de la propiedad MissingMappingAction

Valor	Comentario
MissingMappingAction.Ignore	Los datos no coincidentes se ignoran
MissingMappingAction.Error	Se genera un error en caso de encontrar datos no coincidentes
MissingMappingAction.Passthrough	Todo dato no coincidente se agrega al DataSet

Tabla 9.2. Valores posibles de la propiedad MissingSchemaAction

Valor	Comentario
MissingSchemaAction.Ignore	Se ignora la información de esquema adicional
MissingSchemaAction.Error	Producir un error en caso de que se encuentre información adicional
MissingSchemaAction.Add	La nueva información se añade a la ya existente en el DataSet
MissingSchemaAction.AddWithKey	Lo mismo que el anterior, pero añadiendo además información sobre claves de la tabla

El valor por defecto de MissingMappingAction es Passthrough y el de MissingSchemaAction es Add. Así, al llamar al método Fill() se añadirán al DataSet todos los datos del origen, tengan o no una correspondencia en ese momento con el contenido del DataSet, incorporando, además, información de esquema relativa a las columnas que forman la tabla.

Nota

Si se llama al método Fill() facilitando un DataSet vacío, algo muy habitual, es imprescindible que el adaptador añada todos los datos del origen y la información de esquema, de ahí que los valores por defecto de las dos propiedades sean éstos y no otros. Podría ocurrir, sin embargo, que tuviésemos un DataSet con una cierta estructura ya definida, caso en el cual podría interesarnos recuperar información en los DataTable ya existentes, sin añadir nada. En un caso así asignaríamos a ambas propiedades los valores Ignore o Error, según deseemos simplemente ignorar el hecho u obtener una excepción para conocerlo.

Si además de la información de esquema de las columnas, que es lo que se recupera por defecto, deseamos también que se obtenga la relativa a las columnas clave, antes de llamar al método `Fill()` tendríamos que asignar el valor `AddWithKey` a la propiedad `MissingSchemaAction`. Debe tener en cuenta, sin embargo, que al recuperar datos teniendo esta información se sustituirán automáticamente aquellas filas en las que el valor de la columna clave coincide con las recuperadas del origen.

Actualización de datos

El objeto `DataSet` dispone de la capacidad suficiente como para facilitar la edición local de los datos, almacenando tanto la información original como los cambios que se hayan ido produciendo en el conjunto de datos. Cada una de las filas de cada tabla mantiene en la propiedad `RowState` un indicador con el que es posible saber si se ha cambiado o no, si se trata de una fila nueva o bien si se ha eliminado.

Partiendo de esos cambios, el método `Update()` del adaptador de datos, al que hay que facilitar el `DataSet` como parámetro, sabe si tiene que ejecutar una sentencia de inserción, actualización o borrado. Estas sentencias se encontrarán almacenadas en comandos a los que hacen referencia las propiedades `InsertCommand`, `UpdateCommand` y `DeleteCommand`, respectivamente, a las que tendríamos que asignar un objeto `Command` con la información de comando y conexión apropiada.

Siempre que la sentencia de selección utilizada en el adaptador no contenga más de una tabla, podemos servirnos de un objeto `CommandBuilder` para generar automáticamente los comandos de inserción, actualización y borrado. Como se aprecia en la figura 9.2, los componentes `CommandBuilder` de cada proveedor no guardan, aparte de ser derivados todos de `Component`, ninguna relación especial entre sí. No obstante, funcionan de manera prácticamente idéntica y cuentan con los mismos miembros, entre ellos tres llamados `GetInsertCommand()`, `GetUpdateCommand()` y `GetDeleteCommand()` que, como puede suponer, devuelven las referencias a los `Command` ya preparados.

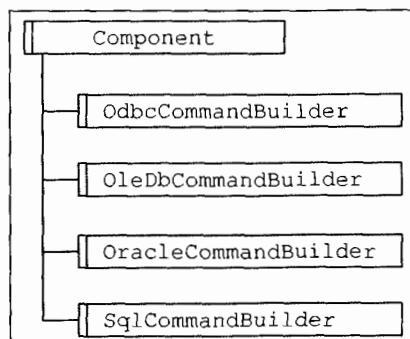


Figura 9.2. Objetos `CommandBuilder` de cada proveedor

Puede ver cómo funciona un objeto `CommandBuilder` con un ejemplo tan simple como el siguiente, en el cual se crea un `SqlCommandBuilder` a partir de un adaptador de datos ya existente. Una vez creado, podemos utilizar los métodos antes citados para acceder a los comandos y obtener las sentencias SQL. El resultado de la ejecución sería el de la figura 9.3. Cambie de proveedor y verá cómo las sentencias se ajustan, en consecuencia, a la sintaxis del nuevo origen de datos.

```

Imports System.Data.SqlClient

Module Module1

Sub Main()
    ' ...
    Dim Adaptador As New SqlDataAdapter(_
        "SELECT * FROM Editoriales", _
        "Data Source=lnapiro; Initial Catalog=Libros;" &_
        "User ID=sa; Password=")

    ' ...
    Dim Comandos As SqlCommandBuilder =_
        New SqlCommandBuilder(Adaptador)

    ' ...
    Console.WriteLine("INSERT='{0}'" & vbCrLf & vbCrLf &_
        "UPDATE=''{1}''" & vbCrLf & vbCrLf & "DELETE=''{2}''",_
        Comandos.GetInsertCommand().CommandText,_
        Comandos.GetUpdateCommand().CommandText,_
        Comandos.GetDeleteCommand().CommandText)

End Sub

End Module

```

```

C:\PbddVisualBasicNET\Cap_09\Comandos\bin\Comandos.exe
INSERT:'INSERT INTO Editoriales< Nombre , Direccion > VALUES < @p1 , @p2 >'
UPDATE:'UPDATE Editoriales SET Nombre = @p1 , Direccion = @p2 WHERE < @IDEitorial = @p3>
AND <@Nombre IS NULL AND @p4 IS NULL> OR <Nombre = @p5>> AND <@Direccion IS NULL AND @p6 IS
NULL> OR <Direccion = @p7>> '
DELETE:'DELETE FROM Editoriales WHERE < @IDEitorial = @p1> AND <@Nombre IS NULL AND @p2>
IS NULL> OR <Nombre = @p3>> AND <@Direccion IS NULL AND @p4 IS NULL> OR <Direccion = @p5>> '
Press any key to continue...

```

Figura 9.3. Sentencias de inserción, actualización y borrado generadas por el `CommandBuilder`

En la figura 9.4 se ha representado de forma esquemática, y simplificada, el modelo de objetos empleado para acceder a un origen de datos, llenarlo con información y después actualizarla. Observe que el `DataSet` no forma parte del proveedor de datos, mientras que el resto de elementos sí. Establecidos los parámetros de la conexión, crearíamos un `DataAdapter` y un `CommandBuilder` asociándolo al primero. A continuación llamaríamos al método `Fill()` para obtener los datos, con los que trabajaríamos internamente o mediante una interfaz de usuario. Al llamar al método `Update()`, el `DataAdapter` haría uso del `CommandBuilder` para enviar al origen las sentencias necesarias.

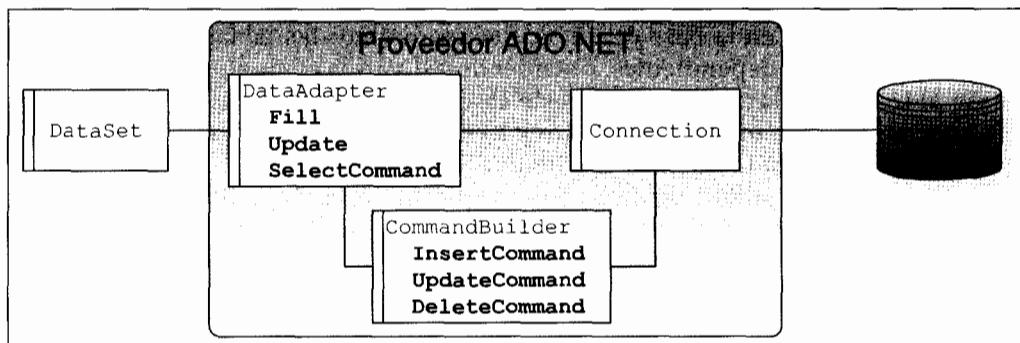


Figura 9.4. Relación entre conjuntos de datos, adaptadores, generadores de comandos y conexiones

Nota

Debe tener en cuenta que el método `Update()` del adaptador se sirve del estado de cada `DataRow` de la tabla para determinar si necesita o no ser actualizada. No debe llamar al método `AcceptChanges()` del `DataSet` antes de invocar a `Update()`, ya que, de hacerlo, todos los `RowState` volverían a su estado por defecto y el adaptador no encontraría nada que actualizar. Tras llamar a `Update()`, a fin de mantener sincronizado el `DataSet`, puede llamar a `AcceptChanges` o bien volver a invocar al método `Fill()` a fin de ver no sólo los cambios propios sino también los de terceros.

En la práctica

Ahora que conocemos teóricamente el funcionamiento de un adaptador y, en parte, el de un `DataSet`, vamos a ir poniendo en práctica algunas de las operaciones posibles usando conjuntamente ambos elementos. Primero conectaremos con el origen de datos y generaremos un `DataSet` a partir de los datos de uno de los orígenes, recorriendo después las tablas para conocer su estructura y contenido.

Después veremos cómo efectuar algunas operaciones sobre él y cómo devolverlas al origen de datos.

Una vez que nos hayamos familiarizado con estas operaciones básicas, continuaremos profundizando en los detalles de los DataSet y la realización de otras operaciones.

Recuperación de datos

Utilizando el proveedor `SqlClient`, para acceder a SQL Server, vamos a preparar un adaptador que nos devuelva en un `DataSet` las tablas `Editoriales` y `Libros` que hay en la base de datos. Recuperaremos las tablas por separado, con dos sentencias independientes, obteniendo así varios conjuntos de datos según se vio al tratar los `DataReader`. En este caso, sin embargo, no tendremos que pasar explícitamente de un conjunto a otro y leer las filas de manera individual, sino que obtendremos toda la información de una vez.

Creado el adaptador de datos y el propio `DataSet`, nos limitamos a llamar al método `Fill` para ejecutar los comandos y recuperar la información de esquema y datos propiamente dichos. El resto del programa, como puede verse a continuación, es una sucesión de bucles con los que recorremos todas las columnas de cada fila de cada tabla que contenga el `DataSet`.

```
Imports System.Data.SqlClient

Module Module1

    Sub Main()
        ' Definimos el adaptador de datos
        Dim Adaptador As New SqlDataAdapter(
            "SELECT * FROM Editoriales; SELECT * FROM Libros",
            "Data Source=inspiron; Initial Catalog=Libros; " &
            "User ID=sa; Password=""")
        ' Creamos el DataSet
        Dim Datos As New DataSet("MisDatos")
        ' y lo llenamos con la información devuelta
        Adaptador.Fill(Datos)

        Dim Tabla As DataTable
        Dim Columna As DataColumn, Fila As DataRow

        ' recorremos todas las tablas del origen
        For Each Tabla In Datos.Tables
            ' mostrando el nombre
            Console.WriteLine("Tabla '" & Tabla.TableName & "'")
            ' por cada tabla todas las columnas
            For Each Columna In Tabla.Columns
                Console.Write(Columna.ColumnName & vbTab)
            Next
            Console.WriteLine(vbCrLf & New String("=", 70))
            ' todas las filas
        Next
    End Sub
End Module
```

```

For Each Fila In Tabla.Rows
    ' con los valores de cada columna
    For Each Columna In Tabla.Columns
        Console.WriteLine(Fila(Columna))
        Console.WriteLine(vbTab)
    Next
    Console.WriteLine()
Next
Console.WriteLine()
Next

End Sub

End Module

```

La llamada al método `Fill()` provoca el envío del comando al origen de datos, en este caso SQL Server, que procesa las dos sentencias SQL facilitadas de manera independiente, devolviendo dos conjuntos de resultados. Éstos son recogidos por el `DataAdapter`, que se encarga de crear un `DataTable` por cada conjunto de datos recibido.

Puesto que no hemos especificado de manera explícita nombre alguno para las tablas, el propio adaptador les dará un nombre por defecto. A continuación, dentro de cada `DataTable`, se definirán los `DataColumn` que se precisen, utilizando para ello la información de esquema facilitada por SQL Server. De esta forma se establecerán, entre otros datos, el nombre y tipo de cada columna. Por último, se añadirán a cada `DataTable` tantos `DataRow` como filas de datos compongan cada conjunto.

Al ejecutar el programa debería obtener un resultado similar al de la figura 9.5. Observe los nombres asignados a las tablas, `Table` y `Table1`.

C:\PBdVisualBasicNET\Eap_09\UlenarDataSet\bin\UlenarDataSet.exe					
	Editorial	Título	Autor	Dirección	
1	Omega Multimedia	Juan Ignacio Lugo de Tena, 15			
2	McGraw Hill	Edificio University, 1 ^a planta			
3	Oreilly	981 Grayson Street			
Tabla 'Table'					
1					
2					
3					
Tabla 'Table1'					
1	Editorial ISBN	Título	Autor	Editorial	Precio
2	3-893115 94 1	User Interface Design for Programmers	Joel Spolsky	3	31
3	84-315 1136 5	SQL Server 2000 Francisco Charte	1	18,75	
4	84-315 1324-4	Guía práctica para usuarios JBuilder 7	Francisco Charte	1	18,75
5	84-315 1392 9	Programación con Visual C# .NET Francisco Charte	1	39	
6	84-315 1393 7	Programación con Visual Studio .NET Francisco Charte/Jorge Sipriano	1	39	
7	84-315 1351-1	Programación con Visual Studio .NET Francisco Charte	1	19	
8	84-315 1298 6	Guía práctica para programar de Visual Basic .NET Francisco Charte	1	19	
9	84-315 1291 4	Guía práctica para usuarios de Visual Studio .NET Francisco Charte	1	19,7	
10	84-315 1261-2	Programación con Delphi 6 y Kylix Francisco Charte	1	27,26	
11	84-315 1255 8	Guía práctica para usuarios de Delphi 6 Francisco Charte	1	19,52	
12	84-315 1230 2	Manual avanzado Excel 2002 Francisco Charte	1	21,04	
13	84-315 1292 2	Guía práctica para usuarios de Excel 2002 Francisco Charte/H.Jesús Lugo	1	19,5	
14	84-315 1132 2	Guía práctica para numeros de Kylix Francisco Charte	1	19,52	
15	84-315 1145-4	Introducción a la programación Francisco Charte	1	24,04	
16	84-7615 234 5	Manual del microprocesador 80386 Cheis H.Pappa/William H.Murray	III	2	40

Figura 9.5. Contenido del DataSet

La ejecución por lotes, introduciendo varios comandos en la misma sentencia, es una característica que no contemplan la mayoría de los orígenes de datos, excepción hecha de SQL Server. El mismo resultado anterior podría conseguirse usando dos adaptadores de datos independientes, uno para cada comando, y sendas llamadas a sus métodos Fill(). Puede comprobarlo eliminando la primera parte del programa anterior, hasta la declaración de la variable Tabla, y sustituyéndola por el código siguiente:

```

' Definimos una conexión común
Dim Conexion As New SqlConnection(
    "Data Source=inspiron; Initial Catalog=Libros; " &
    "User ID=sa; Password=")

' Para los dos adaptadores de datos
Dim AdaptadorEditoriales As New SqlDataAdapter( _
    "SELECT * FROM Editoriales", Conexion)
Dim AdaptadorLibros As New SqlDataAdapter( _
    "SELECT * FROM Libros", Conexion)

' Creamos el DataSet
Dim Datos As New DataSet("MisDatos")

' y lo llenamos con la información devuelta
' dando nombre a los DataTable creados
AdaptadorEditoriales.Fill(Datos, "Editoriales")
AdaptadorLibros.Fill(Datos, "Libros")

```

Observe que en este caso hemos facilitado como segundo parámetro del método Fill() el nombre que deseamos dar a los DataTable. Otra opción sería añadir un elemento a la colección TableMappings de cada adaptador, indicando el nombre de la tabla origen, que suele ser Table si en la sentencia SQL no se ha establecido ninguno, y el nombre que deseamos dar al DataTable. En cualquier caso, al ejecutar el programa tras estos cambios el resultado será prácticamente idéntico, tan sólo diferirán los nombres de los DataTable existentes en el DataSet.

Información de esquema

En el ejemplo anterior, por la consola se da salida a una combinación de información de esquema (el nombre de las columnas) y de datos contenidos en las tablas. Vamos ahora a centrarnos en la información de esquema que, en cierta manera, restringirá las operaciones que podemos efectuar con los datos de cada fila de cada tabla.

En principio, al llamar al método Fill() de un adaptador de datos, la única información de esquema que se recupera es la de las columnas: nombre, tipo, longitud máxima, valor por defecto, etc. No se obtienen, por el contrario, las restricciones aplicables a las columnas que actúan como clave primaria, ni se indica que dichas columnas deben mantener valores únicos o que se generan automáticamente como una secuencia.

Antes de recuperar los datos propiamente dichos, podemos preparar el DataSet obteniendo toda la información de esquema, utilizando para ello el método FillSchema(). Éste toma distintas listas de argumentos, siendo una de las versiones sobrecargadas la siguiente:

```
Adaptador.FillSchema(DataSet, SchemaType, NombreTabla)
```

El primer y tercer parámetro corresponden a los que hemos facilitado al método Fill en el último fragmento de código del punto anterior. El segundo determina si la información de esquema creada en el DataSet es la del origen de datos, SchemaType.Source, o, por el contrario, es necesario establecer algún tipo de asociación, SchemaType.Mapped.

Como en el ejemplo anterior, en el que recuperábamos datos, habría que hacer una llamada a FillSchema() por cada conjunto de datos a recuperar, generando en el DataSet un DataTable con toda la información, incluidas las restricciones de claves primarias. Al añadirse de manera independiente, no obstante, lo que ni los adaptadores ni el DataSet pueden determinar es la relación entre los distintos DataTable creados, por lo que no se generan las restricciones apropiadas de clave externa.

Sin embargo, esta información puede generarse fácilmente creando una relación, un DataRelation, entre las tablas, añadiéndola a la colección Relations del DataSet.

Esto es, precisamente, lo que se hace en el código siguiente:

```
' Definimos una conexión común
Dim Conexion As New SqlConnection(
    "Data Source=inspiron; Initial Catalog=Libros; " &
    "User ID=sa; Password=")

' Para los dos adaptadores de datos
Dim AdaptadorEditoriales As New SqlDataAdapter(
    "SELECT * FROM Editoriales", Conexion)
Dim AdaptadorLibros As New SqlDataAdapter(
    "SELECT * FROM Libros", Conexion)

' Creamos el DataSet
Dim Datos As New DataSet("MisDatos")

' Recuperamos información de esquema
AdaptadorEditoriales.FillSchema(
    Datos, SchemaType.Source, "Editoriales")

' De las dos tablas
AdaptadorLibros.FillSchema(Datos,
    SchemaType.Source, "Libros")

' Y creamos la relación que hay entre ellas
Datos.Relations.Add(New DataRelation("FK_EdLibro",
    Datos.Tables("Editoriales").Columns("IDEEditorial"),
    Datos.Tables("Libros").Columns("Editorial")))
```

Tras las dos llamadas a `FillSchema()`, observe cómo se añade la relación entre las tablas creando un objeto `DataRelation`. Facilitamos a su constructor el nombre que deseamos darle a la relación, una referencia a la columna de la tabla principal y otra a la columna de la tabla de detalle. Con esto se generan automáticamente las restricciones apropiadas. Al ejecutar el código anterior, tendremos un `DataSet` sin datos pero con toda la información de esquema. Podemos mostrar ésta por consola con el código siguiente, que producirá el resultado mostrado en la figura 9.6. Observe los atributos de las columnas `IDEditorial` e `IDLibro`, así como las restricciones definidas en cada tabla.

```

Dim Tabla As DataTable, Restriccion As Constraint
Dim Columna As DataColumn, Fila As DataRow
' recorremos todas las tablas del origen
For Each Tabla In Datos.Tables
    ' mostrando el nombre
    Console.WriteLine("Tabla '" & Tabla.TableName & "' " & _
        vbCrLf & New String("-", 40))
    Console.WriteLine("***** COLUMNAS *****" & vbCrLf)
    ' por cada tabla todos sus columnas
    For Each Columna In Tabla.Columns
        ' arreglo con los datos que queremos mostrar
        Dim DatosColumnas() As String = { _
            {"Nombre", Columna.ColumnName}, _
            {"Tipo", Columna.DataType.ToString()}, _
            {"Admite nulo", Columna.AllowDBNull}, _
            {"Autoincremento", Columna.AutoIncrement}, _
            {"Valor único", Columna.Unique}, _
            {"Longitud", Columna.MaxLength}}
        Dim Contador As Byte ' recorremos el arreglo
        For Contador = 0 To 5
            ' mostrando cada dato
            Console.WriteLine("{0,-15}...: {1}", _
                DatosColumnas(Contador, 0), _
                DatosColumnas(Contador, 1))
        Next
        ' separación entre columnas
        Console.WriteLine(New String("=", 60))
    Next
    ' separación entre tablas
    Console.WriteLine("***** RESTRICCIONES *****" & vbCrLf)
    ' Recorremos las restricciones
    For Each Restriccion In Tabla.Constraints
        ' mostrando su nombre
        Console.WriteLine("Nombre ...: {0}", Restriccion.ConstraintName)
        ' y tipo
        If TypeOf Restriccion Is ForeignKeyConstraint Then
            ' si es de clave externa mostramos las columnas relacionadas
            Console.WriteLine("Clave externa : {0}-{1}", _
                CType(Restriccion, ForeignKeyConstraint).Columns(0), _
                CType(Restriccion, _
                    ForeignKeyConstraint).RelatedColumns(0))
        Else ' si de unicidad mostramos la columna afectada
            Console.WriteLine("Valor único : {0}", _
                CType(Restriccion, UniqueConstraint).Columns(0))
        End If
    Next
End Sub

```

```

End If
Next
' Imprime el resultado en la consola
Console.WriteLine(vbCrLf & vbCrLf)
Next

```

The screenshot shows a Windows application window titled "InformacionEsquema.exe". Inside the window, there are two sections of schema information:

Tabla 'Editoriales'

***** COLUMNAS *****

Nombre	:: IDEditorial
Tipo	:: System.Int32
Admite nulo	:: False
Autoincremento	:: True
Valor único	:: True
Longitud	:: -1
Nombre	:: Nombre
Tipo	:: System.String
Admite nulo	:: True
Autoincremento	:: False
Valor único	:: False
Longitud	:: 50
Nombre	:: Direccion
Tipo	:: System.String
Admite nulo	:: True
Autoincremento	:: False
Valor único	:: False
Longitud	:: 50

***** RESTRICCIONES *****

Nombre	:: Constraint1
Valor único	:: IDEditorial

Tabla 'Libros'

***** COLUMNAS *****

Nombre	:: IDLibro
Tipo	:: System.Int32
Admite nulo	:: False
Autoincremento	:: True
Valor único	:: True
Longitud	:: -1
Nombre	:: ISBN
Tipo	:: System.String
Admite nulo	:: False
Autoincremento	:: False
Valor único	:: False
Longitud	:: 13
Nombre	:: Titulo
Tipo	:: System.String
Admite nulo	:: True
Autoincremento	:: False
Valor único	:: False
Longitud	:: 50
Nombre	:: Autor
Tipo	:: System.String
Admite nulo	:: True
Autoincremento	:: False
Valor único	:: False
Longitud	:: 50
Nombre	:: Editorial
Tipo	:: System.Int32
Admite nulo	:: False
Autoincremento	:: False
Valor único	:: False
Longitud	:: -1
Nombre	:: Precio
Tipo	:: System.Decimal
Admite nulo	:: True
Autoincremento	:: False
Valor único	:: False
Longitud	:: -1

***** RESTRICCIONES *****

Nombre	:: Constraint1
Valor único	:: IDLibro
Nombre	:: FK_EdLibro
Clave externa	:: Editorial IDEditorial

Figura 9.6. Información de esquema del conjunto de datos

Teniendo el DataSet en este estado, con toda la información de esquema y las relaciones definidas, llamaríamos al método `Fill()` de cada adaptador para recuperar los datos. En ese momento no sólo se almacenarían los valores de las filas en los correspondientes `DataTable` sino que, además, se comprobarían las restricciones definidas, evitando, por ejemplo, que existiesen valores duplicados en las columnas que actúan como clave o un código editorial inexistente en un libro.

Nota

Debe tener en cuenta que, una vez se han definido las restricciones, el orden en el que invoque a los métodos `Fill()` de cada adaptador es importante. Si tras las dos llamadas a `FillSchema()` del ejemplo anterior intenta ejecutar la sentencia `AdaptadorLibros.Fill(Datos, "Libros")`, vería que se produce una excepción. Esto es así porque, al no haber todavía datos en la tabla de editoriales, los libros están haciendo referencia a editoriales que no existen, violando una de las restricciones.

Manipulación de los datos

Partiendo de que tenemos un `DataSet` con toda la información de esquema de nuestras tablas, y todas sus filas, manipular el contenido podría parecer, en principio, muy sencillo. Basta con recurrir a la propiedad `Rows` de cada tabla para encontrar una fila, cambiar sus datos, eliminarla o añadir nuevas filas. Por el camino, sin embargo, podemos encontrarnos algún obstáculo que otro.

Suponga, por ejemplo, que necesita añadir una nueva fila a la tabla `Libros`, correspondiente a un título perteneciente a una editorial que no está definida en la tabla `Editoriales`. No puede introducir el libro directamente, puesto que al no existir la editorial obtendría un error al violar la restricción de integridad referencial generada a partir de la relación entre ambas tablas.

Tendría, por tanto, que crear primero la nueva editorial en la tabla `Editoriales`. El identificador lo asigna automáticamente la base de datos en algunos casos, mientras que otros se establecería mediante código. En el primer caso, para obtener el código asignado a la editorial por el origen de datos, tendríamos que interceptar el evento `RowUpdated` del adaptador, utilizando una sentencia `SELECT` para leer la variable `@@IDENTITY`, en la que se almacena el último valor de la columna identidad. En el segundo, debería obtener el último valor existente en la tabla, incrementarlo y usarlo como identificador, lo cual no nos garantiza que otro cliente, en la misma situación, efectúe idéntica operación y emplee el mismo código. Por ello lo mejor es usar un generador, columna de identidad, secuencia o procedimiento almacenado, según el tipo de origen de datos que se use.

En cualquier caso, tras crear la nueva editorial habría que actualizar el `DataSet` a fin de que refleje el nuevo estado de la tabla, no impidiendo el uso de ese nuevo identificador en la tabla de libros. Para ello bastaría con una nueva llamada al mé-

todo Fill(). Por último, insertaríamos el nuevo libro. El proceso completo queda reflejado en el programa siguiente:

```
Imports System.Data.SqlClient

Module Module1
    ' ID de la última editorial añadida
    Dim IDEditorial As Integer

    Sub Main()
        ' Definimos una conexión común
        Dim Conexion As New SqlConnection(
            "Data Source=inspiron; Initial Catalog=Libros; " &
            "User ID=sa; Password=")

        ' Para los dos adaptadores de datos
        Dim AdaptadorEditoriales As New SqlDataAdapter( _
            "SELECT * FROM Editoriales", Conexion)
        Dim AdaptadorLibros As New SqlDataAdapter( _
            "SELECT * FROM Libros", Conexion)

        ' Controlamos el evento de actualización de filas
        ' de la tabla Editoriales
        AddHandler AdaptadorEditoriales.RowUpdated,
                    New SqlRowUpdatedEventHandler(AddressOf ActualizaEditoriales)

        ' Creamos los generadores de comandos para las tablas
        Dim ComandosEditoriales As New SqlCommandBuilder( _
            AdaptadorEditoriales)
        Dim ComandosLibros As New SqlCommandBuilder( _
            AdaptadorLibros)

        ' Creamos el DataSet
        Dim Datos As New DataSet("MisDatos")

        ' Recuperamos información de esquema
        AdaptadorEditoriales.FillSchema( _
            Datos, SchemaType.Source, "Editoriales")
        ' de las dos tablas
        AdaptadorLibros.FillSchema(Datos, _
            SchemaType.Source, "Libros")

        ' y creamos la relación que hay entre ellas
        Datos.Relations.Add(New DataRelation("FK_EdLibro",
            Datos.Tables("Editoriales").Columns("IDEitorial"),
            Datos.Tables("Libros").Columns("Editorial")))

        ' Llenamos el DataSet
        AdaptadorEditoriales.Fill(Datos, "Editoriales")
        AdaptadorLibros.Fill(Datos, "Libros")

        ' Recuperamos una referencia a las filas
        ' de la tabla de editoriales, por comodidad
        Dim Filas As DataRowCollection =
            Datos.Tables("Editoriales").Rows
```

```

' Añadimos una nueva fila
Filas.Add(New Object()
          {0, "Wiley", "605 Third Avenue, New York"})
' y modificamos una de las existentes
Filas.Find(1)("Direccion") = "Telémaco, 43"

' Actualizamos la tabla
AdaptadorEditoriales.Update(Datos, "Editoriales")

' y renovamos su contenido
AdaptadorEditoriales.Fill(Datos, "Editoriales")

' Recuperamos una referencia a las filas
' de la tabla de libros
Filas = Datos.Tables("Libros").Rows

' añadimos un nuevo libro usando el ID de la editorial
' recién creada
Filas.Add(New Object() {0, "0-471-37523-3",
                           "Assembly Language Step-by-Step", "Jeff Duntemann",
                           IDEditorial, 60.5})

' Actualizamos la tabla de libros
AdaptadorLibros.Update(Datos, "Libros")
End Sub

' Este método es invocado al actualizarse la tabla
' de editoriales
Private Sub ActualizaEditoriales(ByVal sender As Object, ByVal args As SqlRowUpdatedEventArgs)

' si el comando a ejecutar es una inserción
If args.StatementType = StatementType.Insert Then
    ' recuperamos el valor dado a la columna identidad
    IDEditorial = New SqlCommand("SELECT @@IDENTITY", _
        args.Command.Connection).ExecuteScalar()

    ' lo mostramos en consola por comprobación
    Console.WriteLine("Nuevo código de editorial: {0}", _
        IDEditorial)
End If
End Sub
End Module

```

Además de añadir una nueva editorial, se utiliza el método `Find()` de la colección `Rows` de la tabla de editoriales para encontrar la que corresponde al identificador 1 y modificar su dirección.

Observe cómo se añaden nuevas filas a las tablas, mediante el método `Add()`, facilitando como argumento un arreglo con todos los valores. Fíjese también en la codificación del método `ActualizaEditoriales()`, que se ejecutará cada vez que se actualice una fila de la tabla `Editoriales`. En su interior comprobamos si lo que se produce es una inserción, caso en el cual recuperaremos el valor del identificador y lo almacenamos en una variable.

Si ejecuta este programa varias veces, se añadirán múltiples editoriales y libros a las respectivas tablas, con exactamente la misma información pero distintos identificadores, ya que éstos los genera automáticamente SQL Server.

Nota: Además del método `Find()`, que se emplea para encontrar la fila que tiene un cierto valor en la columna que actúa como clave principal, también puede usar el método `Select()` para recuperar todas aquellas filas que cumplen un cierto criterio de búsqueda. Consulte la información de referencia de la clase `DataTable` para ver las diferentes versiones que existen de dicho método y saber cómo se usa.

Definición de conjuntos de datos

Aunque la estructura de un `DataSet` puede extraerse de un origen de datos, según se ha visto en los ejemplos previos, esto no resulta imprescindible, siendo posible definirla también mediante código, creando objetos `DataTable` y `DataColumn` y definiendo sus propiedades. Es una técnica que resulta apropiada, por ejemplo, cuando no va a utilizarse un origen de datos tipo RDBMS o base de datos, sino que se quiere almacenar la información localmente en un archivo para trabajar de forma independiente.

Las colecciones `Tables` y `Relations` de un `DataSet` son dinámicas, por lo que pueden añadirseles elementos, nuevas tablas y relaciones. Lo mismo ocurre con las colecciones `Columns` y `Rows` de un `DataTable`. Sólo sabiendo esto, no tendríamos mucho problema para definir dos tablas con varias columnas, indicando su tipo y otras propiedades, y una relación entre esas dos tablas.

Después vendría la inserción de datos, para lo cual son vigentes las mismas normas vistas en los puntos anteriores. En los siguientes vamos a crear un `DataSet` con dos tablas relacionadas, insertando algunas filas en ellas.

Creación de tablas, columnas y relaciones

Suponga que desea crear un conjunto de datos para almacenar información relativa a pacientes y sus historiales, en un supuesto proyecto para una pequeña clínica privada que dará como resultado una aplicación instalada en un solo ordenador y con un solo usuario. En lugar de instalar, configurar y utilizar un RDBMS, decide generar el `DataSet` mediante código de la siguiente manera:

```
' Creamos el DataSet
Dim Datos As New DataSet("Hospital")

' Conteniendo dos tablas
Dim Pacientes As DataTable = Datos.Tables.Add("Pacientes")
Dim Historial As DataTable = Datos.Tables.Add("Historial")

' Definimos la estructura de la primera
With Pacientes.Columns
    ' el campo clave
    With .Add("IDPaciente", Type.GetType("System.Int32"))
        ' se autoincrementará
        .AutoIncrement = True
        .AutoIncrementSeed = 1
        .AutoIncrementStep = 1
        .ReadOnly = True
    End With
    ' añadimos el resto de columnas
    With .Add("Nombre", Type.GetType("System.String"))
        .MaxLength = 40
        .AllowDBNull = False
    End With
    With .Add("Direccion", Type.GetType("System.String"))
        .MaxLength = 50
    End With
    With .Add("Telefono", Type.GetType("System.String"))
        .MaxLength = 12
    End With
End With
' establecemos la clave primaria
Pacientes.PrimaryKey = New DataColumn() _
    {Pacientes.Columns("IDPaciente")}

' Definimos la estructura de la segunda tabla
With Historial.Columns
    ' la columna clave
    With .Add("IDEntrada", Type.GetType("System.Int32"))
        ' se autoincrementará
        .AutoIncrement = True
        .AutoIncrementSeed = 100
        .AutoIncrementStep = 5
        .ReadOnly = True
    End With
    ' añadimos el resto de columnas
    .Add("Paciente", Type.GetType("System.Int32"))
    .Add("Fecha", Type.GetType("System.DateTime"))
    With .Add("Descripcion", Type.GetType("System.String"))
        .MaxLength = 1024
        .AllowDBNull = False
    End With
End With
' y definimos la clave primaria
Historial.PrimaryKey = New DataColumn() _
    {Historial.Columns("IDEntrada")}
```

```
' Establecer la relación entre ambas tablas
Datos.Relations.Add(New DataRelation("FK_HistorialPaciente", _
    Pacientes.Columns("IDPaciente"), _
    Historial.Columns("Paciente")))
```

Creamos un DataSet vacío a cuya propiedad Tables añadimos dos elementos, dos tablas llamadas Pacientes e Historial, cuyas referencias almacenamos en sendas variables. A continuación tenemos dos bloques de código similares, en los cuales vamos añadiendo elementos a las colecciones Columns de cada uno de esos DataTable. Para ello usamos el método Add() que, como puede ver, toma dos parámetros: el nombre del nuevo DataColumn y el tipo de dato. No podemos entregar directamente, como valor, Integer o System.Int32, ya que éstos son tipos y lo que necesita el constructor es un valor que identifique al tipo. Lo obtenemos mediante el método compartido GetType() de la clase Type.

A medida que van añadiéndose columnas, se usa la referencia devuelta por el método Add() para establecer algunas propiedades adicionales, como MaxLength o AllowDBNull. Las columnas que actúan como clave, IDPaciente e IDEntrada, se definen sólo de lectura y con incremento automático. Fíjese también cómo se establecen como claves principales asignándolas a la propiedad PrimaryKey de sus respectivas tablas. Dicha propiedad necesita un arreglo de objetos DataColumn, ya que la clave principal puede componerse de múltiples columnas. En este caso creamos el arreglo con un único elemento: la columna IDPaciente o IDEntrada, según la tabla.

Finalmente, añadimos una relación al DataSet creando una clave externa en la tabla Historial, concretamente en la columna Paciente, enlazada con la clave primaria de la tabla Pacientes. Como sabe, esta relación generará automáticamente otras restricciones.

Inserción de datos

En este momento tenemos un DataSet con información de esquema pero vacío, el mismo estado que obteníamos tras las llamadas a FillSchema() de un ejemplo previo. Para llenar el DataSet con datos no vamos a conectar con un origen, definir un comando y llamar al método Fill() de un adaptador, sino que iremos añadiendo fila a fila e introduciendo los datos apropiados en cada una de las columnas.

La primera columna de cada tabla, clave primaria definida como sólo de lectura y con autoincremento, tomará su valor automáticamente a medida que vayamos añadiendo filas. Podemos recuperar su valor leyendo la columna de la fila recién añadida. Es lo que se hace en el código siguiente para enlazar las entradas de historial con el paciente que corresponda:

```
' Añadimos un paciente
Dim Paciente As DataRow = Pacientes.NewRow()
Paciente("Nombre") = "Francisco Charte"
Paciente("Direccion") = "Apdo. 54"
```

```

Paciente("Telefono") = "763 376 321"
Pacientes.Rows.Add(Paciente)

' y una entrada en su historial
Dim Entrada As DataRow = Historial.NewRow()
Entrada("Paciente") = Paciente("IDPaciente")
Entrada("Fecha") = DateTime.Now
Entrada("Descripcion") = "Cólico renal"
Historial.Rows.Add(Entrada)

' Añadimos un nuevo paciente
Paciente = Pacientes.NewRow()
Paciente("Nombre") = "Filipino Fino"
Paciente("Direccion") = "Pez, 12"
Paciente("Telefono") = "123 456 789"
Pacientes.Rows.Add(Paciente)

' con dos entradas en el Historial
Entrada = Historial.NewRow()
Entrada("Paciente") = Paciente("IDPaciente")
Entrada("Fecha") = DateTime.Now
Entrada("Descripcion") = "Gripe corriente"
Historial.Rows.Add(Entrada)

Entrada = Historial.NewRow()
Entrada("Paciente") = Paciente("IDPaciente")
Entrada("Fecha") = DateTime.Now
Entrada("Descripcion") = "Alergia"
Historial.Rows.Add(Entrada)

```

Si durante este proceso infringimos alguna de las restricciones definidas en las tablas, implícita o explícitamente, se producirá la correspondiente excepción. Ocurrirá esto si, por ejemplo, intenta establecer el valor de las columnas IDPaciente o IDEntrada duplicando un valor existente o si añade una entrada que haga referencia a un paciente inexistente.

Consulta de estructura y contenido

Independientemente de cómo se haya creado la información de esquema del DataSet, y facilitado el contenido de las tablas que lo componen, el método para obtener esa información y contenido son los mismos que ya conocemos. Para verlo en la práctica, vamos a añadir al programa compuesto por los dos bloques de código previos, introducidos en el método Main(), dos métodos adicionales. El primero, al que llamaremos InformacionEsquema(), recibe como parámetro una referencia a un DataSet y muestra por la consola su estructura. El código de este método, muy similar al usado en un ejemplo anterior, sería el mostrado aquí:

```

' Este método muestra el esquema del DataSet
' facilitado como parámetro
Sub InformacionEsquema(ByVal Datos As DataSet)

```

```
Dim Tabla As DataTable, Restriccion As Constraint
Dim Columna As DataColumn, Fila As DataRow

' recorremos todas las tablas del origen
For Each Tabla In Datos.Tables
    ' mostrando el nombre
    Console.WriteLine("Tabla '" & Tabla.TableName & "' " & _
        vbCrLf & New String("-", 40))

    Console.WriteLine("***** COLUMNAS *****")
    ' por cada tabla todas las columnas
    For Each Columna In Tabla.Columns
        ' arreglo con los datos que queremos mostrar
        Dim DatosColumnas(,) As String = {_
            {"Nombre", Columna.ColumnName}, _
            {"Tipo", Columna.DataType.ToString()}, _
            {"Admite nulo", Columna.AllowDBNull}, _
            {"Autoincremento", Columna.AutoIncrement}, _
            {"Valor único", Columna.Unique}, _
            {"Longitud", Columna.MaxLength}}
        
        Dim Contador As Byte ' recorremos el arreglo
        For Contador = 0 To 5
            ' mostrando cada dato
            Console.WriteLine("{0,-15}..: {1}", _
                DatosColumnas(Contador, 0), _
                DatosColumnas(Contador, 1))
        Next
        ' separación entre columnas
        Console.WriteLine(New String("=", 60))
    Next

    Console.WriteLine("***** RESTRICCIONES *****")
    ' Recorremos las restricciones
    For Each Restriccion In Tabla.Constraints
        ' mostrando su nombre
        Console.Write("Nombre ..: {0}" & vbTab, _
            Restriccion.ConstraintName)
        ' y tipo
        If TypeOf Restriccion Is ForeignKeyConstraint Then
            ' si es de clave externa mostramos las columnas relacionadas
            Console.WriteLine("Clave externa : {0}-{1}", _
                CType(Restriccion, ForeignKeyConstraint).Columns(0), _
                CType(Restriccion, _
                    ForeignKeyConstraint).RelatedColumns(0))
        Else ' si de unicidad mostramos la columna afectada
            Console.WriteLine("Valor único : {0}", _
                CType(Restriccion, UniqueConstraint).Columns(0))
        End If
    Next

    ' separación entre tablas
    Console.WriteLine(vbCrLf)
Next
End Sub
```

El segundo método, llamado `MuestraContenido()`, también recibe como parámetro un `DataSet`. En este caso lo que se hace es recorrer todas las tablas, y filas de cada tabla, para mostrar su contenido, indicando el nombre de cada columna. El código sería el siguiente:

```
' Este método da salida por la consola al
' contenido de las tablas que componen un
' DataSet entregado como parámetro
Sub MuestraContenido(ByVal Datos As DataSet)
    Dim Tabla As DataTable
    Dim Columna As DataColumn, Fila As DataRow

    ' recorremos todas las tablas del origen
    For Each Tabla In Datos.Tables
        ' mostrando el nombre
        Console.WriteLine("Tabla '" & Tabla.TableName & "'")
        ' por cada tabla todas las columnas
        For Each Columna In Tabla.Columns
            Console.Write(Columna.ColumnName & vbTab)
        Next
        Console.WriteLine(vbCrLf & New String("=", 70))
        ' Todas las filas
        For Each Fila In Tabla.Rows
            ' con los valores de cada columna
            For Each Columna In Tabla.Columns
                Console.Write(Fila(Columna))
                Console.Write(vbTab)
            Next
            Console.WriteLine()
        Next
        Console.WriteLine()
    Next
End Sub
```

Para completar el programa, añadiríamos al final del método `Main()` sendas llamadas a estos dos métodos, facilitando como único parámetro el `DataSet Datos` cuya estructura y contenido hemos preparado previamente. La ejecución debería producir un resultado como el de la figura 9.7. Observe cómo se han generado los valores de las columnas `IDPaciente` e `IDEntrada` según el valor de inicio e incremento que habíamos definido. Fíjese también en las restricciones que han surgido a partir de haber definido las claves primarias y la clave externa.

Almacenamiento local

Tras la inserción de datos efectuada en los `DataTable` de nuestro `DataSet`, cada `DataRow` mantiene una indicación de su estado, en este caso todas las filas son nuevas, de tal manera que podría abrirse una conexión de datos con un origen, prepararse un adaptador y enviarse la información a un RDBMS. Para ello, lógica-

mente, en el origen de datos debería existir exactamente la misma estructura o esquema que hemos definido nosotros en el DataSet.

```

C:\PBddVisualBasicNET\Cap_09\DefinirDataSet\bin\DefinirDataSet.exe
Tabla 'Pacientes'
***** COLUMNAS *****
Nombre    :: IDPaciente
Tipo      :: System.Int32
Admite nulo :: False
Autoincremento :: True
Valor único :: True
Longitud   :: -1
Nombre    :: Nombre
Tipo      :: System.String
Admite nulo :: False
Autoincremento :: False
Valor único :: False
Longitud   :: 40
Nombre    :: Direccion
Tipo      :: System.String
Admite nulo :: False
Autoincremento :: False
Valor único :: False
Longitud   :: 50
Nombre    :: Telefono
Tipo      :: System.String
Admite nulo :: True
Autoincremento :: False
Valor único :: False
Longitud   :: 12
***** RESTRICCIONES *****
Nombre :: Constraint1 Valor único : IDPaciente

Tabla 'Historial'
***** COLUMNAS *****
Nombre    :: IDEntrada
Tipo      :: System.Int32
Admite nulo :: False
Autoincremento :: True
Valor único :: True
Longitud   :: -1
Nombre    :: Paciente
Tipo      :: System.Int32
Admite nulo :: True
Autoincremento :: False
Valor único :: False
Longitud   :: -1
Nombre    :: Fecha
Tipo      :: System.DateTime
Admite nulo :: True
Autoincremento :: False
Valor único :: False
Longitud   :: -1
Nombre    :: Descripcion
Tipo      :: System.String
Admite nulo :: False
Autoincremento :: False
Valor único :: False
Longitud   :: 1924
***** RESTRICCIONES *****
Nombre :: Constraint1 Valor único : IDEntrada
Nombre :: FK_HistorialPaciente Clave externa : Paciente-IDPaciente

Tabla 'Pacientes'
IDPaciente    Nombre     Direccion      Telefono
1             Francisco Charte  Avda. 54        263 376 321
2             Filipino Fino   Pz. 12 123 456 289

Tabla 'Historial'
IDEntrada    Paciente    Fecha       Descripcion
100          1           21/08/2002 1:48:17  Cálculo renal
105          2           21/08/2002 1:48:17  Gripe corriente
410          2           21/08/2002 1:48:17  Alergia

```

Figura 9.7. Estructura y contenido del DataSet definido mediante código

Cuando el esquema de un `DataSet` se define como hemos hecho en los puntos previos, mediante código, normalmente es porque no existe un origen de datos del que pueda recuperarse y, lógicamente, en el que después pueda almacenarse la información. Ésta, sin embargo, sí puede almacenarse y recuperarse localmente, en un archivo.

Escritura de esquema y datos

La clase `DataSet` cuenta con los métodos `WriteXmlSchema()` y `WriteXml()`, siendo la finalidad del primero guardar el esquema de los datos, en forma de esquema XML, y la del segundo almacenar los datos propiamente dichos, con o sin información de esquema según interese.

El almacenamiento de la información de esquema, de manera independiente, se puede aplicar a diversos escenarios, no sólo a la conservación y recuperación de datos locales. En uno de los primeros ejemplos del capítulo vimos cómo, después de recuperar información de esquema desde el origen con el método `FillSchema()`, teníamos que definir manualmente la relación entre las tablas `Editoriales` y `Libros` generando las correspondientes restricciones de integridad referencial. En lugar de hacer esto en cada ordenador cliente, y cada vez que se inicie la aplicación, podríamos guardar el esquema en un archivo XSD y distribuirlo junto con la aplicación, recuperando toda la información de esquema de dicho archivo. De esta forma, tras conectar con el origen de datos tan sólo habría que recuperar los datos.

Se asume que la estructura de las tablas en el origen no van a cambiar, ya que en ese caso la información de esquema que hay en el archivo XSD podrían no coincidir con los datos recuperados por el adaptador.

Llame al final del método `Main()` del ejemplo del punto anterior al método `WriteXmlSchema()` del `DataSet`, facilitando como único parámetro el nombre del archivo en el que se almacenará el esquema, por ejemplo `EsquemaHospital.xsd`. Puede verlo con el Bloc de notas, tendrá que conocer la sintaxis de los esquemas XSD, o bien abrirlo directamente en Visual Studio .NET para ver una representación (véase figura 9.8). Posteriormente, en cualquier otro punto de este mismo programa o en otra aplicación, bastaría una llamada al método `ReadXmlSchema()` de un `DataSet` para recuperar toda la información de esquema.

Si la información de esquema no va utilizarse nunca de forma independiente, sino asociada siempre a los datos que contiene el propio `DataSet`, podemos almacenar conjuntamente esquema y filas de datos. Para ello utilizaríamos el método `WriteXml()` facilitando dos parámetros: el nombre del archivo donde va a escribirse y el indicador `XmlWriteMode.WriteSchema`. El resultado sería un archivo XML conteniendo el esquema XSD, al principio, y los datos, detrás. Si no entregá-

mos el segundo parámetro, el archivo contendría sólo los datos como puede apreciarse en la figura 9.9.

Al recuperar el archivo de datos, mediante el método `ReadXml()`, también tendríamos que indicar si queremos leer la información de esquema o sólo los datos.

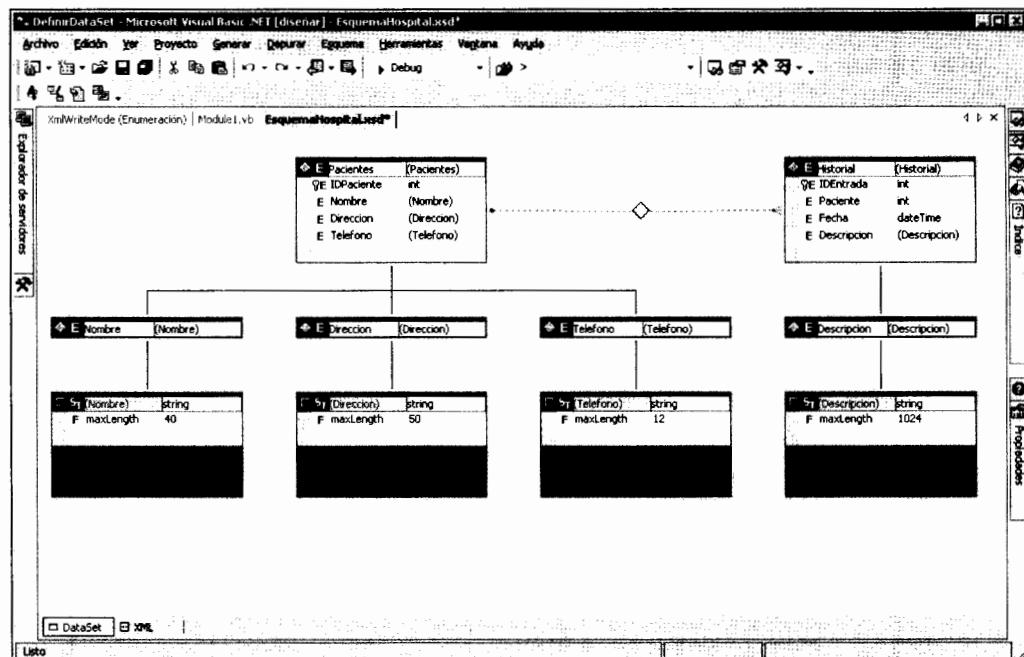


Figura 9.8. Podemos usar Visual Studio .NET para obtener una representación del esquema

DiffGrams

A medida que se efectúan cambios en las tablas de un `DataSet`, las colecciones `DataRow` de cada `DataTable` alojan no sólo la nueva información, sino también el estado y los datos que contenían previamente. Esta información, relativa a los cambios producidos desde que se creó o recuperó el contenido del `DataSet`, permanece ahí hasta que se llama al método `AcceptChanges()` del `DataSet` que, como sabe, invoca al método homónimo de cada `DataTable`.

Cuando se almacena la información en un archivo XML, mediante el método `WriteXml()`, lo que se almacena son los valores actuales de las filas, tras efectuar todos los cambios, a pesar de que no se haya invocado a `AcceptChanges()`. Al recuperar el archivo, por tanto, tendríamos un `DataSet` con datos pero sin información de cómo se operó sobre éstos. La pérdida del estado de las filas, y la información previa a la modificación, no será importante si la aplicación está diseñada para trabajar siempre localmente.

The screenshot shows a Microsoft Internet Explorer window with the title bar 'E:\IEK400\SistemaC\PBddVisualBasicNET\Cap_09\DefinirDataSet\bim\Hospital.xml'. The menu bar includes 'Archivo', 'Edición', 'Ver', 'Favoritos', 'Herramientas', and 'Ayuda'. Below the menu is a toolbar with icons for Back, Forward, Stop, Refresh, Home, Favorites, Multimedia, and Help. The address bar shows the path 'E:\IEK400\SistemaC\PBddVisualBasicNET\Cap_09\DefinirDataSet\bim\Hospital.xml'. The main content area displays the XML code:

```

<?xml version="1.0" standalone="yes" ?>
- <Hospital>
- <Pacientes>
  <IDPaciente>1</IDPaciente>
  <Nombre>Francisco Charte</Nombre>
  <Direccion>Apdo. 54</Direccion>
  <Telefono>763 376 321</Telefono>
</Pacientes>
- <Pacientes>
  <IDPaciente>2</IDPaciente>
  <Nombre>Filipino Fino</Nombre>
  <Direccion>Pez, 12</Direccion>
  <Telefono>123 456 789</Telefono>
</Pacientes>
- <Historial>
  <IDEntrada>100</IDEntrada>
  <Paciente>1</Paciente>
  <Fecha>2002-08-21T11:07:59.788Z304+02:00</Fecha>
  <Descripcion>Cólico renal</Descripcion>
</Historial>
- <Historial>
  <IDEntrada>105</IDEntrada>
  <Paciente>2</Paciente>
  <Fecha>2002-08-21T11:07:59.818Z2736+02:00</Fecha>
  <Descripcion>Gripe corriente</Descripcion>
</Historial>
- <Historial>
  <IDEntrada>110</IDEntrada>
  <Paciente>2</Paciente>
  <Fecha>2002-08-21T11:07:59.818Z2736+02:00</Fecha>
  <Descripcion>Alergia</Descripcion>
</Historial>
</Hospital>

```

At the bottom of the window, there are buttons for 'Listo' and 'Intranet local'.

Figura 9.9. Estructura del archivo XML contenido sólo los datos, sin información de esquema

Suponga, por el contrario, que la aplicación que está diseñando obtiene el esquema y los datos a partir de una conexión con un origen de datos. A partir de ahí, sin embargo, tiene que trabajar sin conexión porque el programa, pongamos por caso, se ejecuta en un portátil y el usuario viaja de un punto a otro sin posibilidad de conexión permanente con el servidor de datos.

Todas las ediciones, por lo tanto, deberán almacenarse localmente a fin de que puedan posteriormente, en algún momento en que se tenga conexión, resolverse con el origen de datos.

En un caso así es imprescindible conservar no sólo los datos nuevos, sino la propia información de los cambios que han ido efectuándose. Éstos pueden almacenarse en un documento XML con formato *DiffGram*, una estructura en la que se determina el estado de cada fila y, además, se indica cuál era el estado anterior a

los cambios y los posibles errores que pudieran haber surgido. En este caso habría que facilitar al método `WriteXml()`, como segundo parámetro, el indicador `XmlWriteMode.DiffGram`.

Nota

Al guardar el estado de un `DataSet` en formato `DiffGram` no se guarda la información de esquema, por lo que ésta debería mantenerse separadamente, con una llamada a `WriteXmlSchema()`, recuperándose posteriormente, en el momento de volver a crear el conjunto de datos, antes de recuperar los datos propiamente dichos.

Puede efectuar una prueba, partiendo del código del ejemplo anterior, a fin de ver la estructura del `DiffGram`.

Dé los pasos siguientes:

- Después de añadir los dos pacientes a la tabla `Paciente`, invoque al método `AcceptChanges()` del `DataSet`. De esta forma los cambios quedarán consolidados y las filas no mantendrán estado ni valores previos.
- Modifique el teléfono del segundo de los pacientes, simplemente por modificar un dato que ya estaba aceptado para ver cómo lo refleja el `DiffGram`. Puede hacerlo con una sentencia como la siguiente:

```
Pacientes.Select("Nombre='Filipino Fino'")(0)("Telefono") = _  
"987 654 321"
```

- Llame al final del código, tras añadir las entradas del historial del segundo paciente, al método `WriteXml()` pasando el indicador `XmlWriteMode.DiffGram` como segundo parámetro.
- Haga doble clic sobre el archivo generado para abrirlo en Internet Explorer y poder apreciar, como en la figura 9.10, la descripción del estado y datos del `DataSet`.

Observe que el primer paciente y la entrada que hay en su historial no cuentan con la propiedad `hasChanges`, ya que tras introducirlos en el `DataSet` se llamó a `AcceptChanges()`. En las demás entradas podemos apreciar el estado `modified` o `inserted`.

Fíjese en la parte inferior del documento, en la rama `<diffgr:before>`. Allí encontramos los valores que tenía la segunda fila de la tabla `Pacientes` antes de efectuar el cambio indicado más arriba. Aquí aparecerían, de igual forma, los contenidos de las filas que pudieran haberse eliminado.

Fíjese en cómo se identifica de manera inequívoca cada fila de cada tabla con un entero asignado a la propiedad `rowOrder`. El número de fila del paciente modificado coincide en la rama de datos y en la de valores previos, como puede verse.

```

<?xml version="1.0" standalone="yes" ?>
<diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
  xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
  - <Hospital>
    - <Pacientes diffgr:id="Pacientes1" msdata:rowOrder="0">
      <IDPaciente>1</IDPaciente>
      <Nombre>Francisco Charte</Nombre>
      <Direccion>Apdo. 54</Direccion>
      <Telefono>763 376 321</Telefono>
    </Pacientes>
    - <Pacientes diffgr:id="Pacientes2" msdata:rowOrder="1" diffgr:hasChanges="modified">
      <IDPaciente>2</IDPaciente>
      <Nombre>Filipino Fino</Nombre>
      <Direccion>Pez, 12</Direccion>
      <Telefono>987 654 321</Telefono>
    </Pacientes>
    - <Historial diffgr:id="Historial1" msdata:rowOrder="0">
      <IDEntrada>100</IDEntrada>
      <Paciente>1</Paciente>
      <Fecha>2002-08-21T11:35:05.7362288+02:00</Fecha>
      <Descripcion>Cólico renal</Descripcion>
    </Historial>
    - <Historial diffgr:id="Historial2" msdata:rowOrder="1" diffgr:hasChanges="inserted">
      <IDEntrada>105</IDEntrada>
      <Paciente>2</Paciente>
      <Fecha>2002-08-21T11:35:05.9865888+02:00</Fecha>
      <Descripcion>Gripe corriente</Descripcion>
    </Historial>
    - <Historial diffgr:id="Historial3" msdata:rowOrder="2" diffgr:hasChanges="inserted">
      <IDEntrada>110</IDEntrada>
      <Paciente>2</Paciente>
      <Fecha>2002-08-21T11:35:05.9865888+02:00</Fecha>
      <Descripcion>Alergia</Descripcion>
    </Historial>
  </Hospital>
  - <diffgr:before>
    - <Pacientes diffgr:id="Pacientes2" msdata:rowOrder="1">
      <IDPaciente>2</IDPaciente>
      <Nombre>Filipino Fino</Nombre>
      <Direccion>Pez, 12</Direccion>
      <Telefono>123 456 789</Telefono>
    </Pacientes>
  </diffgr:before>
</diffgr:diffgram>

```

Figura 9.10. Estructura del *DiffGram* generado por *WriteXml()*

Lectura del conjunto de datos

Complementariamente a *WriteXml()* y *WriteXmlSchema()*, la clase *DataSet* dispone de *ReadXml()* y *ReadXmlSchema()*. El primero de ellos recupera un archivo XML y genera un *DataSet*, mientras que el segundo obtiene sólo la información de esquema para definir la estructura del conjunto de datos.

El comportamiento de `ReadXml()` difiere según el contenido del archivo XML que se recupere, leyendo también el esquema, si está disponible en el mismo documento; deduciéndo el esquema si no está disponible, recuperando en formato *DiffGram* si ésta es la estructura del documento, etc. Ese comportamiento por defecto puede modificarse mediante los indicadores enumerados en la tabla 9.3.

Tabla 9.3. Elementos de la enumeración XmlReadMode

Identificador	Comentario
Auto	Determina la operación a efectuar según el contenido del documento XML
ReadSchema	Recuperar el esquema junto con los datos
InferSchema	Deducir el esquema a partir de la estructura de los datos
IgnoreSchema	Ignorar la información de esquema que pudiese existir en el archivo
DiffGram	Lee el documento como un <i>DiffGram</i> , manteniendo la información de cambios
Fragment	Menos habitual, lee documentos generados en el formato FOR XML de SQL Server

Como puede ver, `ReadXml()` siempre puede obtener el esquema, si éste se encuentra embebido en el documento XML, o bien deducirlo a partir de la estructura de los datos. La única excepción se produce cuando va a recuperarse un *DiffGram*, ya que éste no incorpora el esquema en el documento y tampoco es posible deducirlo a partir de los datos. Si vamos a recuperar un documento de tipo *DiffGram*, por tanto, es imprescindible que antes hayamos obtenido el esquema con una llamada a `ReadXmlSchema()`.

En el ejemplo desarrollado antes, en el que se definía un `DataSet` y se añadían datos a dos tablas, añada las sentencias siguientes al final:

```
Datos.WriteXmlSchema("EsquemaHospital.xsd")
Datos.WriteXml("Hospital.xml")
Datos.WriteXml("HospitalConEsquema.xml", XmlWriteMode.WriteSchema)
Datos.WriteXml("HospitalCambios.xml", XmlWriteMode.DiffGram)
```

De esta forma tendrá cuatro archivos diferentes con información del `DataSet`:

- `EsquemaHospital.xsd`: Tendrá el esquema XML, sin datos.
- `Hospital.xml`: Contendrá los datos actuales, sin esquema y sin información de modificaciones.
- `HospitalConEsquema.xml`: Almacenará el esquema junto con los datos.
- `HospitalCambios.xml`: Contendrá el *DiffGram* con los datos y modificaciones, sin información de esquema.

En un programa distinto, tras ejecutar el anterior, podríamos recuperar el contenido del `DataSet` usando diversas variaciones del método `ReadXml()`, utilizando los mismos métodos `InformacionEsquema()` y `MuestraContenido()` del ejemplo anterior para apreciar el esquema y los datos.

Si lee el archivo `Hospital.xml`, sin más, verá que obtiene el nombre y tipos de las columnas, pero no la información de claves primarias ni externas, ni tampoco las restricciones. Éstos son elementos que `ReadXml()` no puede deducir a partir de la información.

Pruebe a leer el archivo `HospitalConEsquema.xml`, o bien a recuperar el esquema `EsquemaHospital.xsd` con una llamada a `ReadXmlSchema()` y luego recuperar el mismo `Hospital.xml` anterior. En este caso sí se cuenta con toda la información de esquema, ya que ésta se almacenó, en el mismo archivo y de manera independiente, a fin de poder recuperarla.

Por último, recupere la información del *DiffGram* almacenada en el archivo `HospitalCambios.xml`, tal y como se hace en el siguiente fragmento de código. Observe que antes recuperamos el esquema ya que, de lo contrario, obtendríamos una excepción.

```
Sub Main()
    ' Creamos el DataSet
    Dim Datos As New DataSet("Hospital")

    ' Recuperamos esquema
    Datos.ReadXmlSchema("EsquemaHospital.xsd")

    ' y el DiffGram
    Datos.ReadXml("HospitalCambios.xml")

    ' Mostramos la estructura y contenido
    InformacionEsquema(Datos)
    MuestraContenido(Datos)
End Sub
```

A fin de poder apreciar la información recuperada por el `ReadXml()`, podemos modificar el bucle final del método `MuestraContenido()` dejándolo así:

```
For Each Fila In Tabla.Rows
    ' con los valores de cada columna
    For Each Columna In Tabla.Columns
        Console.WriteLine(Fila(Columna))
        Console.WriteLine(vbTab)
    Next
    Console.WriteLine("< " & Fila.RowState.ToString & " >")
Next
```

De esta forma, al ejecutar cualquiera de las variaciones vería si las filas mantienen su estado o, por el contrario, lo han perdido. En la parte inferior de la figura 9.11 puede ver cómo se indica que la segunda fila de la tabla `Pacientes` está modificada, y las dos últimas de la tabla `Historial` son nuevas. Estos datos no los vería

si lee los documentos Hospital.xml u HospitalConEsquema.xml, en los que todas las filas aparecerían como insertadas.

```

C:\PBddVisualBasicNET\Cap_09\RecuperacionLocal\bin\RecuperacionLocal.exe

Nombre      :: Descripcion
Tipo        :: System.String
Admite nulo :: False
Autoincremento :: False
Valor único  :: False
Longitud     :: 1024

***** RESTRICCIONES *****
Nombre :: Constrainti Valor único : IDEntrada
Nombre :: FK_HistorialPaciente Clave externa : Paciente-IDPaciente

Tabla 'Pacientes'
IDPaciente    Nombre      Direccion      Telefono
1   Francisco Charte Apdo. 54      763 376 321  < Unchanged >
2   Filipino Fino Pez, 12 987 654 321  < Modified >

Tabla 'Historial'
IDEntrada    Paciente      Fecha      Descripcion
100          1            21/08/2002 11:35:05  Cólico renal  < Unchanged >
105          2            21/08/2002 11:35:05  Gripe corriente < Added >
110          2            21/08/2002 11:35:05  Alergia < Added >

Press any key to continue...

```

Figura 9.11. Indicación del estado de las filas tras la recuperación

Nota

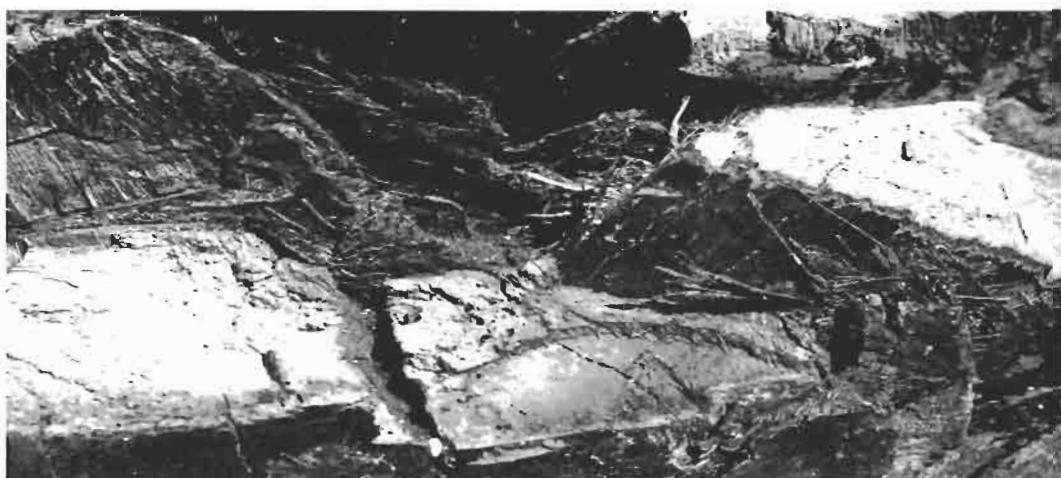
En caso de que deseáramos, desde el código, recuperar los antiguos valores de las filas que se han modificado, podemos hacerlo utilizando la propiedad `Item` de `DataRow` indicando como segundo parámetro uno de los valores de la enumeración `DataRowVersion`. Previamente, podríamos comprobar si esa información está disponible usando el método `HasVersion()` del propio `DataRow`.

Resumen

Los conjuntos de datos, representados por objetos de la clase `DataSet`, son un medio que permite a las aplicaciones trabajar con datos, obtenidos o no desde un cierto origen, sin necesidad de mantener una conexión persistente. Esto reduce el uso de recursos en los servidores y, por tanto, incrementa la escalabilidad al facilitar el acceso de un mayor número de clientes. Esta naturaleza *desconectada* nos permite, además, crear aplicaciones que no cuentan con una conexión permanente al servidor, por ejemplo al operar en sistemas móviles, consolidando los datos a posteriori, cuando dicha conexión es posible.

Al finalizar este capítulo tiene un conocimiento mucho más profundo de las características de los conjuntos de datos, así como del uso de las clases `DataSet`, `DataTable`, `DataColumn`, `DataRow`, `DataAdapter`, etc. Ha visto en la práctica cómo obtener datos, visualizarlos, manipularlos y devolver los cambios al origen. También cómo usar conjuntos de datos creados dinámicamente, mediante código, almacenándolos y recuperándolos de un archivo XML local.

A pesar de todo, aún quedan ciertos detalles que irá conociendo con la práctica y también en algunos de los capítulos posteriores, como los problemas de concurrencia que se tratarán más adelante.



10

Relaciones y vistas

En el capítulo previo se ha visto cómo, mediante el uso de la clase `DataSet`, es posible recuperar un conjunto de datos completo, formado incluso por múltiples tablas, y operar sobre él localmente, sin necesidad de tener una conexión persistente con el servidor. Las filas de datos recuperadas, alojadas en el interior de objetos `DataTable`, aparecen siempre en el orden en que se obtuvieron. Además, el `DataTable` siempre facilita el conjunto total de filas, aunque puede utilizarse el método `Select()` para obtener una lista de aquellas que cumplen un cierto criterio.

No es extraño que, durante la ejecución de una aplicación, se precise el mismo conjunto de datos pero con las filas en un orden diferente, o con un subconjunto de la lista de filas obtenidas inicialmente. Lo que se hacía en estos casos, habitualmente, era enviar una nueva sentencia SQL al servidor para recuperar un conjunto de resultados a medida. Lógicamente, una operación de este tipo consume recursos y tiempo del servidor de datos.

Con ADO.NET existe una alternativa a esa técnica, consistente en utilizar objetos `DataView` y `DataViewManager` para obtener diferentes vistas del mismo conjunto de datos. Un `DataView` filtra y ordena los datos existentes en un `DataSet`, sin necesidad de conectar con el origen de datos y ejecutar un nuevo comando. Se usan, por tanto, menos recursos y, además, el resultado es inmediato al no tener que esperar la respuesta del servidor.

Nuestro objetivo en este capítulo es conocer el funcionamiento básico de `DataView` y `DataViewManager`, creándolos mediante código para obtener varias vistas diferentes de los mismos datos. Estos objetos, al igual que los adaptadores y los propios conjuntos de datos, pueden generarse de una forma mucho más simple

usando el entorno de Visual Studio .NET como tendrá ocasión de ver en capítulos posteriores.

Filtrado y ordenación de un DataTable

La clase `DataTable` dispone de un método, llamado `Select()`, mediante el cual es posible extraer todas aquellas filas que cumplen un cierto criterio y/o estado con el orden que se prefiera. En realidad, `Select()` no ordena las filas de la tabla, sino que crea un arreglo de referencias a objetos `DataRow` en el orden apropiado, incluyendo en él sólo las referencias de aquellas filas que cumplen el criterio de filtrado.

Podemos invocar al método `Select()` facilitando cuatro listas de parámetros distintas, según las cuales se devolverán todas las filas del `DataTable`, sólo aquellas que cumplen un cierto criterio, sólo las que cumplen el criterio y en un cierto orden o bien sólo las que cumplen el criterio y tienen un cierto estado y en un cierto orden:

- `Select()`: Facilita la lista de todas las filas que forman parte de la tabla.
- `Select("Criterio")`: En el arreglo se hace referencia sólo a las filas que cumplen con el criterio indicado. Éste, una cadena de caracteres, puede ser realmente complejo pues se permite el uso de operadores de diversos tipos. En su forma más básica, sigue las reglas habituales de las consultas QBE. Pueden crearse criterios como "Precio < 30" o "Titulo LIKE Prog*".
- `Select("Criterio", "Orden")`: Aparte del criterio de selección, como primer parámetro, se entrega también una cadena indicando el nombre de la columna por la que se ordenará. Opcionalmente, pueden incluirse uno de los modificadores `ASC` o `DESC` para comunicar de manera explícita si el orden debe ser ascendente o descendente.
- `Select("Criterio", "Orden", Estado)`: Los dos primeros parámetros coinciden con los del formato anterior, mientras que el tercero será uno de los elementos de la enumeración `DataRowViewState` de la tabla 10.1. Con él seleccionaremos las filas que se encuentran en un cierto estado.

Tabla 10.1. Miembros de la enumeración `DataRowViewState`

Identificador	Comentario
<code>Added</code>	Filas que han sido añadidas nuevas
<code>CurrentRows</code>	Todas las filas que componen actualmente la tabla a excepción de las eliminadas
<code>Deleted</code>	Las filas que se han eliminado

Identificador	Comentario
ModifiedCurrent	Las filas actuales tras las modificaciones que se hubiesen efectuado
ModifiedOriginal	Las filas con sus valores originales
OriginalRows	Las filas originales, incluidas las que se han eliminado
Unchanged	Las filas que no han sufrido cambios
None	Ninguna fila

Puede comprobar el funcionamiento del método `Select()` partiendo de cualquiera de los ejemplos del capítulo previo, accediendo a una tabla del `DataSet` para recuperar tan sólo ciertas filas, según criterio o estado, o bien las filas en un cierto orden. Recuerde que el resultado obtenido por una llamada a `Select()` siempre es un arreglo de `DataRow` que, dependiendo de los criterios de selección, podría estar vacío. En el ejemplo siguiente puede ver cómo, partiendo de la tabla `Libros` de SQL Server, se muestran diversos resultados: todas las filas, sólo aquellas en las que el título del libro comienza con la palabra `Programación` y sólo las que corresponde a la primera editorial ordenándolas, además, por la columna `Precio`. La figura 10.1 es una muestra del resultado que se obtendría.

```

Imports System.Data.SqlClient

Module Module1

Sub Main()
    ' Definimos una conexión
    Dim Conexion As New SqlConnection(
        "Data Source=inspiron; Initial Catalog=Libros; " &
        "User ID=sa; Password=")

    ' Y un adaptador de datos
    Dim AdaptadorLibros As New SqlDataAdapter(
        "SELECT * FROM Libros", Conexion)

    ' Creamos el DataSet
    Dim Datos As New DataSet("MisDatos")

    ' y lo llenamos con la información devuelta
    AdaptadorLibros.Fill(Datos, "Libros")

    Dim Filas() As DataRow ' arreglo de filas

    ' seleccionamos todas las filas de la tabla
    Filas = Datos.Tables("Libros").Select()
    MuestraFilas("Todas las filas", Filas)

    ' seleccionamos sólo aquellas con un
    ' título ajustado a un patrón

```

```

Filas = Datos.Tables("Libros").Select(
    "Titulo LIKE 'Programación*'")
MuestraFilas("Libros de programación", Filas)

' sólo los libros de una editorial y
' por orden de precio
Filas = Datos.Tables("Libros").Select(
    "Editorial=1", "Precio ASC")
MuestraFilas(
    "Títulos de la editorial 1 ordenados por precio", Filas)

End Sub

' Este método enumera las filas indicadas
Sub MuestraFilas(ByVal Titulo As String,
    ByVal Filas As DataRow())
    ' mostramos el título
    Console.WriteLine(Titulo & vbCrLf & New String("-", 70))
    Dim Fila As DataRow

    ' recorremos todas las filas
    For Each Fila In Filas
        ' mostrando título, precio y editorial
        Console.WriteLine("{0,-50} {1,6} {2,3}",
            Fila("Titulo"), Fila("Precio"), Fila("Editorial"))
    Next

    ' una pausa antes de continuar
    Console.WriteLine(vbCrLf & "Pulse <Intro> para continuar")
    Console.ReadLine()
End Sub

End Module

```

Generalidades sobre DataView y DataViewManager

El método `Select()` de la clase `DataTable` tiene ciertas limitaciones, especialmente cuando no se controla el acceso a datos mediante código sino que se pretenden utilizar componentes de interfaz de usuario. El resultado devuelto por este método, un arreglo de `DataRow`, no puede vincularse directamente con una rejilla u otros controles de datos.

La clase `DataView` ofrece un mecanismo más flexible para obtener subconjuntos de los datos alojados en un `DataTable`, facilitando la vinculación directa con componentes de interfaz. Podemos usar un `DataView` para obtener parte de las filas de un `DataTable`, sobre la base de un criterio de selección o estado, en el orden que nos interese. Es posible crear múltiples `DataView` sobre un mismo `DataTable`, recuperando diferentes conjuntos filas y en diferentes órdenes, sin por ello duplicar la información.

C:\PBddVisualBasicNET\Cap_10\SelectDataTable\bin\SelectDataTable.exe		
Todos los filas		
User Interface Design for Programmers	31	3
SQL Server 2000	10,25	1
Guía práctica para usuarios JBuilder 7	10,25	1
Programación con Visual C# .NET	39	1
Programación con Visual Studio .NET	40	1
Programación con Visual Basic .NET	39	1
Guía práctica para usuarios de Visual Basic .NET	10,25	1
Guía práctica para usuarios de Visual Studio .NET	10,52	1
Programación con Delphi 6 y Kylix	32,26	1
Guía práctica para usuarios de Delphi 6	10,52	1
Manual avanzado Excel 2002	21,04	1
Guía práctica para usuarios de Excel 2002	10,52	1
Guía práctica para usuarios de Kylix	10,52	1
Introducción a la programación	24,04	1
Manual del microprocesador 80386	40	2
Assembly Language Step by Step	60,5	10
Pulse <Intro> para continuar		
Libros de programación		
Programación con Visual C# .NET	39	1
Programación con Visual Studio .NET	40	1
Programación con Visual Basic .NET	39	1
Programación con Delphi 6 y Kylix	32,26	1
Pulse <Intro> para continuar		
Títulos de la editorial 1 ordenados por precio		
Guía práctica para usuarios de Visual Studio .NET	10,52	1
Guía práctica para usuarios de Delphi 6	10,52	1
Guía práctica para usuarios de Excel 2002	10,52	1
Guía práctica para usuarios de Kylix	10,52	1
SQL Server 2000	10,25	1
Guía práctica para usuarios JBuilder 7	10,25	1
Guía práctica para usuarios de Visual Basic .NET	10,25	1
Manual avanzado Excel 2002	21,04	1
Introducción a la programación	24,04	1
Programación con Delphi 6 y Kylix	32,26	1
Programación con Visual C# .NET	39	1
Programación con Visual Basic .NET	39	1
Programación con Visual Studio .NET	40	1
Pulse <Intro> para continuar		

Figura 10.1. Conjuntos de filas devueltos por el método Select()

Un objeto DataView sólo puede operar sobre las filas de un DataTable. En caso de que tengamos múltiples tablas, relacionadas entre sí, en el interior de un DataSet, usaremos la clase DataViewManager en lugar de DataView. Un objeto DataViewManager consiste, básicamente, en una colección de DataView, pudiendo aplicarse cada uno de ellos a un DataTable del DataSet.

Ambas clases, DataView y DataViewManager, implementan la interfaz IBindingList que hace posible la vinculación con componentes de interfaz. Gracias a ella es posible, por ejemplo, la notificación de cambios por parte de la vista de datos a los componentes de interfaz, actualizando la información mostrada al usuario. En la figura 10.2 se ha representado esquemáticamente la base de estas dos clases y la implementación de la citada interfaz.

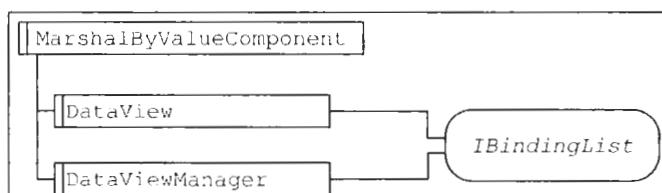


Figura 10.2. Relación de las clases DataView y DataViewManager

Los objetos `DataView` pueden crearse y personalizarse de manera visual en el entorno de Visual Studio .NET, una característica no disponible para los objetos de la clase `DataViewManager`.

Funcionamiento de un `DataView`

Los objetos `DataView` se asocian con objetos `DataTable`, ya sea facilitando la referencia a este último al constructor de la clase `DataView` o usando la propiedad `Table` del `DataView` para vincularlo al `DataTable`. En cualquier caso, a partir de ese momento podemos usar la propiedad `Item` del `DataView` para acceder a las filas de la vista, que serán todas o parte de las existentes en la tabla. Cada uno de los elementos de la propiedad `Item`, que actúa como indexadora de la clase, es un objeto de la clase `DataRowView`. Los objetos de esta clase ofrecen una vista de un `DataRow` en un estado en particular, es decir, no mantienen los diversos estados que puede tener una fila de datos durante la edición.

El conjunto de filas que forman la vista dependerá del valor que tenga la propiedad `RowFilter`. Éste puede facilitarse como segundo parámetro del constructor de `DataView` o bien establecerse posteriormente mediante una asignación a dicha propiedad. El filtro se define como una cadena de caracteres en cuyo interior se introducen los parámetros de selección, con la misma sintaxis empleada con el método `Select()` de la clase `DataTable`.

Si no deseamos que las filas aparezcan en la vista con el orden por defecto, no tenemos más que indicar en la propiedad `Sort` el nombre de la columna o columnas por las que deben ordenarse. Esta columna se convertirá en clave de un índice, facilitando las operaciones de búsqueda. Opcionalmente, puede indicar tras cada nombre de columna si el orden debe ser ascendente o descendente. Esta cadena, indicando las columnas por las que se ordenará, también puede facilitarse como parámetro al constructor de `DataView`, concretamente como tercer argumento.

Por último, en lo que respecta a las propiedades de selección de filas, podemos asignar un estado a `RowStateFilter` a fin de obtener sólo las filas que se encuentren en el estado indicado. Como se decía antes, los objetos `DataRowView` representan el valor de un `DataRow` en un cierto estado, por defecto el valor actual, pudiéndose indicar otro estado diferente mediante una asignación a `RowStateFilter` o entregando el mismo valor como último argumento del constructor.

Además de acceder a los datos, mediante la citada propiedad `Item`, también podemos efectuar operaciones de edición sobre las filas, utilizando para ello los métodos `AddNew()` y `Delete()` del `DataView` y `BeginEdit()`, `EndEdit()` y `CancelEdit()` de cada `DataRowView`. Que estas operaciones sean posibles, o no, dependerá de los valores que mantengan las propiedades `AllowNew`, `AllowDelete` y `AllowEdit` del propio `DataView`.

El valor de las propiedades AllowDelete, AllowNew y AllowEdit, que no sólo podemos leer sino también modificar, afectará, por ejemplo, al comportamiento de los componentes de interfaz que se enlacen con el DataView.

Si se ha establecido un orden en la vista, indicando una columna que actuará como clave, a partir de dicho momento pueden utilizarse los métodos Find() y FindRows() para localizar la fila o filas, respectivamente, que tienen un valor dado en esa columna clave.

Todos los DataTable cuentan con una vista por defecto, a la que puede accederse mediante la propiedad DefaultView, que puede modificarse mediante las propiedades RowFilter, Sort y RowState. De esta forma podríamos modificar la vista por defecto que nos ofrece el propio DataTable, en lugar de crear un DataView nuevo.

Funcionamiento de un DataViewManager

A diferencia de DataView, los objetos DataViewManager se enlazan directamente con un DataSet y no con un DataTable. Su finalidad es facilitar una vista global de todas las tablas que forman el conjunto de datos, respetando las relaciones que pudieran existir entre ellas. Podemos vincular el DataViewManager con el DataSet facilitando una referencia a este último al constructor del primero, o bien sirviéndonos de la propiedad DataSet con que cuenta la clase DataViewManager.

En lugar de crear un nuevo DataViewManager, también podemos obtener el que tiene por defecto el conjunto de datos mediante la propiedad DefaultViewManager de la clase DataSet.

Los dos miembros de más interés para nosotros, en la clase DataViewManager, serán DataViewSettings y CreateDataView(). El primero, una propiedad, nos permite acceder a la colección de objetos DataViewSetting que establecen las propiedades de la vista de cada tabla, mientras que el segundo, un método, es el encargado de crear nuevas vistas asociadas a tablas.

Cada elemento de la colección DataViewSettings mantiene los atributos de la vista aplicada a cada tabla. Para ello, la clase DataViewSettings cuenta con las mismas propiedades RowFilter, Sort y RowStateFilter explicadas en el punto anterior, junto con la propiedad Table que vincula la configuración con una cierta tabla.

En la práctica

Como siempre, en la documentación de Visual Studio .NET encontrará la información de referencia relativa a cada uno de los miembros de las clases mencionadas, `DataView`, `DataViewManager`, `DataRowView` y `DataViewSetting`. A partir de este punto nos centraremos en ver cómo utilizarlas en la práctica, con algunos ejemplos sencillos pero demostrativos.

Partimos, como en todos los ejemplos previos, de un nuevo proyecto de tipo **Aplicación de consola** usando el lenguaje Visual Basic .NET. Posteriormente, en los capítulos dedicados al uso de las herramientas de Visual Studio .NET, aprenderá a crear vistas y enlazarlas con componentes de interfaz sin necesidad de escribir código alguno.

Múltiples vistas sobre una misma tabla

Nuestro primer ejemplo tiene el objetivo de conseguir un resultado idéntico al del programa de la figura 10.1, pero usando múltiples vistas sobre la misma tabla en lugar de emplear el método `Select()` de ésta. Nos sirve, por tanto, la primera parte en la que se definía la conexión, el adaptador de datos y se creaba el `DataSet` con la información de la tabla `Libros`. A partir de ahí, como se ve en el código siguiente, empiezan los cambios.

```
Sub Main()
    ' Definimos una conexión
    Dim Conexion As New SqlConnection(
        "Data Source=inspirón; Initial Catalog=Libros; " &
        "User ID=sa; Password=")

    ' y un adaptador de datos
    Dim AdaptadorLibros As New SqlDataAdapter(
        "SELECT * FROM Libros", Conexion)

    ' Creamos el DataSet
    Dim Datos As New DataSet("MisDatos")

    ' y lo llenamos con la información devuelta
    AdaptadorLibros.Fill(Datos, "Libros")

    ' Creamos una vista de todas las filas
    Dim VistaTotal As New _
        DataView(Datos.Tables("Libros"))

    ' Otra sólo con las filas actuales de
    ' libros de programación
    Dim VistaProgramacion As New _
        DataView(Datos.Tables("Libros")),
        "Titulo LIKE 'Programación*' ", "Titulo", _
        DataViewRowState.CurrentRows)
```

```
' y una tercera con los títulos de la primera editorial
Dim VistaEditorial1 As New DataView(Datos.Tables("Libros"), _
    "Editorial=1", "Precio DESC", _
    DataViewRowState.CurrentRows)

' Llamamos al método MuestraFilas
MuestraFilas("Todas las filas", VistaTotal)
MuestraFilas("Libros de programación", VistaProgramacion)
MuestraFilas("Títulos de la editorial 1", VistaEditorial1)
End Sub
```

Hemos optado por establecer los criterios de selección y ordenación en el mismo momento en que se crea cada `DataView`, facilitando los métodos apropiados al constructor de dicha clase. Al final tenemos tres objetos que ofrecen vistas diferentes de la misma tabla de datos. Para trabajar con estos objetos tendremos, asimismo, que hacer algunos cambios en el método `MuestraFilas()`. Éste quedaría como se muestra a continuación:

```
' Este método muestra el contenido de una vista
Sub MuestraFilas(ByVal Titulo As String, ByVal Vista As DataView)
    ' mostramos el título
    Console.WriteLine(Titulo & vbCrLf & New String("-", 70))
    Dim Fila As DataRowView

    ' recorremos todas las filas
    For Each Fila In Vista
        ' mostrando título, precio y editorial
        Console.WriteLine("{0,-50} {1,6} {2,3}", _
            Fila("Titulo"), Fila("Precio"), Fila("Editorial"))
    Next

    ' una pausa antes de continuar
    Console.WriteLine(vbCrLf & "Pulse <Intro> para continuar")
    Console.ReadLine()
End Sub
```

Observe que la variable `Fila` ahora no es un `DataRow` sino un `DataRowView`. Asimismo, el elemento que se recorre en el bucle `For Each` es la propia vista entregada como segundo argumento, no un arreglo de objetos `DataRow`. El resultado, tal y como se aprecia en la figura 10.3, es casi idéntico al del ejemplo previo, exceptuando el orden de los datos en la segunda y tercera vista.

La vista por defecto de una tabla

Cada `DataTable` cuenta con una vista por defecto, un objeto `DataView` al que hace referencia la propiedad `DefaultView`. Esta vista, inicialmente, hace referencia a todas las filas de la tabla. Utilizando las propiedades `RowFilter`, `Sort` y `RowStateFilter`, sin embargo, podemos configurarla para que nos facilite los datos que nos interesen en cada momento.

C:\PBddVisualBasicNET\Cap_10\VariasVistas\bin\VariasVistas.exe		
Todas las filas		
User Interface Design for Programmers	31	3
SQL Server 2000	10,75	1
Guía práctica para usuarios JBuilder ?	10,75	1
Programación con Visual C# .NET	39	1
Programación con Visual Studio .NET	40	1
Programación con Visual Basic .NET	39	1
Guía práctica para usuarios de Visual Basic .NET	10,25	1
Guía práctica para usuarios de Visual Studio .NET	10,52	1
Programación con Delphi 6 y Kylix	37,26	1
Guía práctica para usuarios de Delphi 6	10,52	1
Manual avanzado Excel 2002	21,04	1
Guía práctica para usuarios de Excel 2002	10,52	1
Guía práctica para usuarios de Kylix	10,52	1
Introducción a la programación	24,04	1
Manual del microprocesador 80386	40	2
Assembly Language Step-by-Step	60,5	10
Pulse <Intro> para continuar		
Libros de programación		
Programación con Delphi 6 y Kylix	37,26	1
Programación con Visual Basic .NET	39	1
Programación con Visual C# .NET	39	1
Programación con Visual Studio .NET	40	1
Pulse <Intro> para continuar		
Titulos de la editorial 1		
Programación con Visual Studio .NET	40	1
Programación con Visual C# .NET	39	1
Programación con Visual Basic .NET	39	1
Programación con Delphi 6 y Kylix	37,26	1
Introducción a la programación	24,04	1
Manual avanzado Excel 2002	21,04	1
SQL Server 2000	10,75	1
Guía práctica para usuarios JBuilder ?	10,75	1
Guía práctica para usuarios de Visual Basic .NET	10,75	1
Guía práctica para usuarios de Visual Studio .NET	10,52	1
Guía práctica para usuarios de Delphi 6	10,52	1
Guía práctica para usuarios de Excel 2002	10,52	1
Guía práctica para usuarios de Kylix	10,52	1
Pulse <Intro> para continuar		

Figura 10.3. Conjuntos de filas ofrecidos por las distintas vistas creadas

Nota

La configuración del DefaultView afecta a las filas devueltas por la vista, pero nunca a las contenidas en la propiedad Rows del DataTable que siempre será la colección de todas las filas que haya en la tabla.

En lugar de crear múltiples vistas diferentes, lo cual en ocasiones puede no ser imprescindible, podemos modificar las propiedades de DefaultView según las necesidades de cada caso. Es lo que se hace en la siguiente versión modificada del mismo ejemplo anterior. El método MuestraFilas () se mantiene idéntico al punto anterior.

```
Sub Main()
    ' Definimos una conexión
    Dim Conexion As New SqlConnection(
        "Data Source=inspiron; Initial Catalog=Libros;" &
        "User ID=sar; Password=""")
```

```
' y un adaptador de datos
Dim AdaptadorLibros As New SqlDataAdapter(_
    "SELECT * FROM Libros", Conection)

' Creamos el DataSet
Dim Datos As New DataSet("MisDatos")

' y lo llenamos con la información devuelta
AdaptadorLibros.Fill(Datos, "Libros")

' Obtenemos una referencia a la tabla
Dim MiTabla As DataTable = Datos.Tables("Libros")

' Mostramos la vista por defecto
MuestraFilas("Todas las filas", MiTabla.DefaultView)

' Modificamos el criterio de selección
MiTabla.DefaultView.RowFilter =
    "Titulo LIKE 'Programación*!'

' y la mostramos de nuevo
MuestraFilas("Libros de programación", MiTabla.DefaultView)

' Establecemos el criterio de selección
With MiTabla.DefaultView
    .RowFilter = "Editorial=1"
    .Sort = "Precio DESC" ' y orden
    .RowStateFilter = DataViewRowState.CurrentRows
End With

' y volvemos a mostrarla
MuestraFilas("Títulos de la editorial 1", MiTabla.DefaultView)
End Sub
```

El resultado, al ejecutar este ejemplo, sería idéntico al mostrado en la figura 10.3, exceptuando el orden alfabético de las filas en la segunda vista.

Búsqueda de datos en una vista

Una vista facilita un subconjunto de las filas de una tabla, sobre la base de un filtro de selección, en un determinado orden. Éste viene determinado por el valor de una o más columnas que, en cierta manera, actúan como columnas clave en esa vista, facilitando operaciones rápidas de búsqueda.

En la clase `DataView` existen dos métodos, `Find()` y `FindRows()`, cuya finalidad es encontrar la primera fila o el conjunto de filas, respectivamente, que tienen un cierto valor en esa columna clave. El primero devuelve un entero que es el índice de la fila dentro del `DataView`, mientras que el segundo facilita un arreglo de objetos `DataRowView` con todas las filas que lo contienen.

Puede efectuar una pequeña prueba iniciando una nueva aplicación de consola e introduciendo el código siguiente:

```
Imports System.Data.SqlClient

Module Module1
Sub Main()
    ' Definimos una conexión
    Dim Conexion As New SqlConnection(
        "Data Source=inspiron; Initial Catalog=Libros; " &
        "User ID=sa; Password=")

    ' y un adaptador de datos
    Dim AdaptadorLibros As New SqlDataAdapter(
        "SELECT * FROM Libros", Conexion)

    ' Creamos el DataSet
    Dim Datos As New DataSet("MisDatos")

    ' y lo llenamos con la información devuelta
    AdaptadorLibros.Fill(Datos, "Libros")

    ' Obtenemos una referencia a la tabla
    Dim MiTabla As DataTable = Datos.Tables("Libros")

    ' Algunas variables necesarias
    Dim Columna As DataColumn
    Dim NombreColumna As String
    Dim ValorBuscado As String

    ' Mostramos la lista de columnas disponibles
    Console.WriteLine("**** Columnas disponibles ****")
    For Each Columna In MiTabla.Columns
        Console.WriteLine(vbTab & "-" & Columna.ColumnName)
    Next

    ' Solicitamos la columna por la que ordenar
    Console.Write("Introduzca el nombre de la " &
        "columna por la que quiere ordenar: ")
    NombreColumna = Console.ReadLine()

    ' y la establecemos
    MiTabla.DefaultView.Sort = NombreColumna

    ' Mostramos las filas en el orden indicado
    MuestraFilas("Todas las filas", MiTabla.DefaultView)

    ' Solicitamos el valor a buscar
    Console.Write("Introduzca el valor a buscar: ")
    ValorBuscado = Console.ReadLine()

    Dim Indice As Integer = MiTabla.DefaultView.Find(ValorBuscado)
    If Indice <> -1 Then
        Console.WriteLine("{0,-50} {1,3} {2,6}", _
            MiTabla.DefaultView(Indice)("Titulo"), _
            MiTabla.DefaultView(Indice)("Editorial"), _
            MiTabla.DefaultView(Indice)("Precio"))
    End If
End Sub
```

```

' Muestra filas mediante el método Read() de una vista
Sub MuestraFilas(ByVal Titulo As String, ByVal Vista As DataView)
    ' Pasa el valor al título
    Console.WriteLine(Titulo & vbCrLf & New String("-", 70))
    Dim Fila As DataRowView

    ' Recorre las filas de la vista
    For Each Fila In Vista
        ' Recorre las columnas de la fila
        Console.WriteLine("{0,13} {1,-50} {2,-30} {3,6} {4,3}", _
            Fila("ISBN"), Fila("Titulic"), Fila("Autor"), _
            Fila("Precio"), Fila("Editorial"))
    Next

    End Sub

End Module

```

Usamos una vez más la tabla Libros de SQL Server, aunque podríamos utilizar cualquier otro origen de datos. Tras crear el adaptador de datos y generar el DataSet, enumeramos las columnas de la tabla para que, a través de la consola, el usuario del programa pueda seleccionar una de ellas para ordenar las filas. Asignamos el nombre de columna introducido a la propiedad Sort del DataView y, acto seguido, mostramos el contenido de la vista. Aparecerán, como puede verse en la figura 10.4, las filas ordenadas por esa columna.

A continuación solicitamos un valor a buscar, mostrando algunos datos de la primera fila que lo contenga en la columna clave. Puede modificar esta parte final para, en lugar de llamar a Find(), emplear el método FindRows() y enumerar todas las filas que coinciden.



Figura 10.4. Seleccionamos el orden de las filas y un valor a buscar

En este ejemplo no se han controlado las posibles excepciones que podrían generarse, por ejemplo al introducir el nombre de una columna inexistente para ordenar por ella.

Edición de datos en la vista

Una vista de datos, siempre que contenga información suficiente como para asociar cada fila de la vista con la fila original en la tabla, puede utilizarse también para actualizar datos, modificando el contenido de las filas, añadiendo otras nuevas o bien eliminando las existentes. El proceso, en todos los casos, es igualmente sencillo:

- **Editar una fila existente:** Localizado el objeto `DataRowView` que corresponda, llamaríamos al método `BeginEdit()` para iniciar los cambios, tras lo cual modificaríamos el valor de las columnas que deseemos. El proceso terminaría con una llamada a `EndEdit()`, para confirmar los cambios, o bien `CancelEdit()`, para deshacerlos.
- **Añadir una nueva fila:** Invocamos al método `AddNew()` del `DataView` para así obtener un nuevo `DataRowView`, asignando a continuación los valores de todas sus columnas y, como en el caso anterior, confirmando los cambios o descartándolos mediante los métodos `EndEdit()` y `CancelEdit()`.
- **Eliminar una fila:** Basta con llamar al método `Delete()` del `DataRowView` a eliminar. En realidad la fila no se elimina en ese instante, sino que se modifica su estado para que aparezca como candidata a ser eliminada.

Cualquier modificación de las filas causará que en el `DataTable` del que depende la vista se añadan nuevas filas o cambie el estado de los asistentes. En cualquier caso, esos cambios no son en firme hasta en tanto no se llame al método `AcceptChanges()` del `DataSet` o, en caso de que deseemos transferirlos a un origen de datos, se utilice el método `Update()` del adaptador de datos.

Manteniendo el método `MuestraFilas()` de los ejemplos previos, cambie el método `Main()` para que quede tal y como se muestra a continuación:

```
Sub Main()
    ' Definimos una conexión
    Dim Conexion As New SqlConnection(
        "Data Source=inspiron; Initial Catalog=Libros; " &
        "User ID=sa; Password=")

    ' y un adaptador de datos
    Dim AdaptadorLibros As New SqlDataAdapter(
        "SELECT * FROM Libros", Conexion)
```

```
' Generamos los comandos de actualización
Dim Comandos As New SqlCommandBuilder(AdaptadorLibros)

' Creamos el DataSet
Dim Datos As New DataSet("MisDatos")

' y lo llenamos con la información devuelta
AdaptadorLibros.Fill(Datos, "Libros")

' Obtenemos una vista de los libros pertenecientes
' a la editorial número 1
Dim Vista As New DataView(Datos.Tables("Libros"), _
    "Editorial=1", "ISBN ASC", DataViewRowState.CurrentRows)

' mostramos las filas iniciales de la vista
MuestraFilas("Filas iniciales", Vista)

Console.WriteLine(vbCrLf & "Se añade una nueva fila" & vbCrLf)
' Creamos una nueva fila en la vista
Dim Fila As DataRowView = Vista.AddNew()

' Asignamos los datos
Fila("ISBN") = "84-415-1402-X"
Fila("Titulo") = "Guía práctica ASP.NET"
Fila("Autor") = "Óscar González"
Fila("Editorial") = "1"
Fila("Precio") = 10.75

Fila.EndEdit() ' finalizamos la edición

' Mostramos de nuevo la vista
MuestraFilas("Tras añadir la nueva fila", Vista)

' Actualizamos el origen de datos
AdaptadorLibros.Update(Datos.Tables("Libros"))
End Sub
```

Además del conjunto de datos y el adaptador, ahora creamos también un `CommandBuilder` que se encarga de generar las sentencias de actualización, inserción y borrado del origen de datos. Acto seguido creamos una vista y mostramos las filas que la componen, tras lo cual añadimos una nueva fila y confirmamos los cambios, tanto en el `DataRowView`, con la llamada a `EndInit()`, como en el propio origen de datos, llamando a `Update()`.

Si todo va bien, la ejecución de este programa debería generar el resultado que puede verse en la figura 10.5. Tras ejecutarlo, puede comprobar el contenido de la tabla `Libros`, por ejemplo desde el Administrador corporativo de SQL Server, para ver la nueva fila insertada. Como en casos anteriores, y por mantener la simplicidad del código, no se han controlado las posibles excepciones de ejecución que podrían generarse.

Aunque en este ejemplo tan sólo se inserta una fila, igualmente podrían haberse modificado datos de las existentes o haberlas eliminado. El proceso es exactamente el mismo, cambiando sólo la llamada a `AddNew()` por la acción que corresponda.

C:\PBddVisualBasic.NET\Cap_10\EditionDatosVista\bin\EditionDatosVista.exe			
Ficheros iniciales			
84-415-1132-2	Guía práctica para usuarios de Kylix	Francisco Charte	10,52
84-415-1136-5	SQL Server 2000	Francisco Charte	10,75
84-415-1145-3	Introducción a la programación	Francisco Charte	21,01
84-415-1292-7	Guía práctica para usuarios de Excel 2002	Francisco Charte/H.Jesús Lague	10,52
84-415-1293-1	Manual avanzado Excel 2002	Francisco Charte	21,01
84-415-1255-8	Guía práctica para usuarios de Delphi 6	Francisco Charte	10,52
84-415-1261-2	Programación con Delphi 6 y Kylix	Francisco Charte	32,26
84-415-1299-6	Guía práctica para usuarios de Visual Basic .NET	Francisco Charte	10,25
84-415-1291-4	Guía práctica para usuarios de Visual Studio .NET	Francisco Charte	10,52
84-415-1324-4	Guía práctica para usuarios JBuilder 7	Francisco Charte	10,25
84-415-1351-1	Programación con Visual Basic .NET	Francisco Charte	39
84-415-1376-7	Programación con Visual Studio .NET	Francisco Charte/Jorge Serrano	49
84-415-1392-9	Programación con Visual C# .NET	Francisco Charte	39
Se añadió una nueva fila.			
Tiene añadir la nueva fila.			
84-415-1132-2	Guía práctica para usuarios de Kylix	Francisco Charte	10,52
84-415-1136-5	SQL Server 2000	Francisco Charte	10,75
84-415-1145-3	Introducción a la programación	Francisco Charte	21,01
84-415-1292-7	Guía práctica para usuarios de Excel 2002	Francisco Charte/H.Jesús Lague	10,52
84-415-1293-1	Manual avanzado Excel 2002	Francisco Charte	21,01
84-415-1255-8	Guía práctica para usuarios de Delphi 6	Francisco Charte	10,52
84-415-1261-2	Programación con Delphi 6 y Kylix	Francisco Charte	32,26
84-415-1299-6	Guía práctica para usuarios de Visual Basic .NET	Francisco Charte	10,25
84-415-1291-4	Guía práctica para usuarios de Visual Studio .NET	Francisco Charte	10,52
84-415-1324-4	Guía práctica para usuarios JBuilder 7	Francisco Charte	10,25
84-415-1351-1	Programación con Visual Basic .NET	Francisco Charte	39
84-415-1376-7	Programación con Visual Studio .NET	Francisco Charte/Jorge Serrano	49
84-415-1392-9	Programación con Visual C# .NET	Francisco Charte	39
84-415-1402-8	Guía práctica ASP.NET	Oscar González	10,25
Press any key to continue			

Figura 10.5. La vista de datos antes y después de añadir una nueva fila

Nota

La clase `DataRowView` dispone de dos propiedades, `IsEdit` e `IsNew`, que nos permiten saber si la fila está en modo de edición o si se trata de una nueva fila, respectivamente.

Uso de un DataViewManager

Utilizando las propiedades de `DataView`, conjuntamente con métodos de `DataTable` como `GetChildRows()`, desde código no necesitamos más para gestionar relaciones maestro/detalle entre dos o más tablas. El caso, sin embargo, es totalmente distinto cuando van a utilizarse componentes de interfaz que es necesario vincular con orígenes de datos. Es en este contexto en el que encuentra su mayor utilidad la clase `DataViewManager`.

Suponga que quiere mostrar en una cuadrícula, en el interior de un formulario Windows, una serie de editoriales como filas primarias y ciertos títulos de cada una de ellas como filas secundarias. Tras crear una nueva aplicación Windows e insertar en el formulario, como único componente, un `DataGridView`, añadimos el código siguiente al evento `Load`:

```
' Definimos una conexión común
Dim Conexion As New SqlConnection()
    "Data Source=inspiron; Initial Catalog=Libros; " &
    "User ID=sa; Password=""
```

```

' Para los dos adaptadores de datos
Dim AdaptadorEditoriales As New SqlDataAdapter(_
    "SELECT * FROM Editoriales", Conexion)
Dim AdaptadorLibros As New SqlDataAdapter(_
    "SELECT * FROM Libros", Conexion)

' Creamos el DataSet
Dim Datos As New DataSet("MisDatos")

' Queremos recuperar información de clave
AdaptadorEditoriales.MissingSchemaAction = _
    MissingSchemaAction.AddWithKey
AdaptadorLibros.MissingSchemaAction = _
    MissingSchemaAction.AddWithKey

' al llenar los DataSet
AdaptadorEditoriales.Fill(Datos, "Editoriales")
AdaptadorLibros.Fill(Datos, "Libros")

' creamos la relación que hay entre las tablas
Datos.Relations.Add(New DataRelation("Libros de la editorial",
    Datos.Tables("Editoriales").Columns("IDEitorial"), _
    Datos.Tables("Libros").Columns("Editorial")))

DataGridView1.DataSource = Datos
DataGridView1DataMember = "Editoriales"

```

Al ejecutar el programa, la cuadrícula mostrará un aspecto similar al de la figura 10.6. Al pulsar en el nombre de la relación, la editorial seleccionada se indicará en la parte superior de la cuadrícula y aparecerán los datos de todos sus títulos.

Nota

No se preocupe en este momento por las particularidades del componente DataGridView o las propiedades de vinculación DataSource yDataMember, las estudiaremos en un capítulo posterior. En este ejemplo las usamos simplemente para mostrar la funcionalidad de la clase DataViewManager.

Suponga que quiere permitir al usuario la introducción de parte de un título, de tal forma que las editoriales sólo muestren los libros cuyo título comience por el texto dado. En principio, podríamos intentar hacer lo siguiente:

```

Datos.Tables("Libros").DefaultView.RowFilter = _
    "Titulo LIKE 'Programación*'"

```

Con esto, las editoriales tan sólo deberían mostrar los libros cuyo título comience con los caracteres "Programación". Al ejecutar el programa, sin embargo, verá que el resultado es exactamente el mismo que obtenía antes. Con la sentencia

anterior estamos modificando el filtro de la vista por defecto de la tabla Libros, pero lo que estamos mostrando en el DataGrid es el DataSet completo, no una vista en particular, así que intentamos lo siguiente:

```
DataGrid1.DataSource = Datos.Tables("Editoriales").DefaultView
```

IDEitorial	Nombre	Direccion
1	Anaya Multimedia	Telémaco, 43
2	McGraw-Hill	Edificio Valrealty, 1º planta
3	Apress	901 Grayson Street
10	Wiley	605 Third Avenue, New York

Figura 10.6. Lista de editoriales en la cuadrícula

Vinculamos el DataGrid con la vista por defecto de la tabla Editoriales, ya que queremos tener la posibilidad de seleccionar los títulos de cada editorial por separado, pero el filtro asignado a la vista por defecto de la tabla Libros sigue sin aplicarse.

Es en estos casos donde resulta útil la clase DataViewManager, ya que puede enlazarse directamente con el DataGrid facilitándole todos los parámetros de las vistas individuales de cada una de las tablas relacionadas. Sustituya las dos sentencias anteriores por el código siguiente:

```
Dim ViewManager As New DataViewManager(Datos)

ViewManager.DataViewSettings("Libros").RowFilter = -
    "Titulo LIKE 'Programación*'"

DataGrid1.DataSource = ViewManager
DataGrid1.DataMember = "Editoriales"
```

Creamos un nuevo DataViewManager asociándolo con el DataSet. A continuación usamos la colección DataViewSettings para establecer el filtro de la selección sobre la tabla Libros. Por último, enlazamos el DataViewManager directamente con el DataGrid. Ahora, al ejecutar el programa, podrá seguir seleccionando cada editorial por separado, pero sólo aparecerán los libros cuyo título comience por el prefijo indicado, como se aprecia en la figura 10.7.

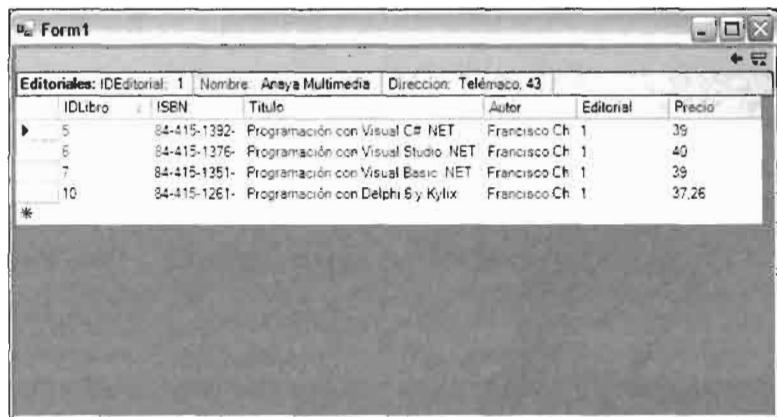


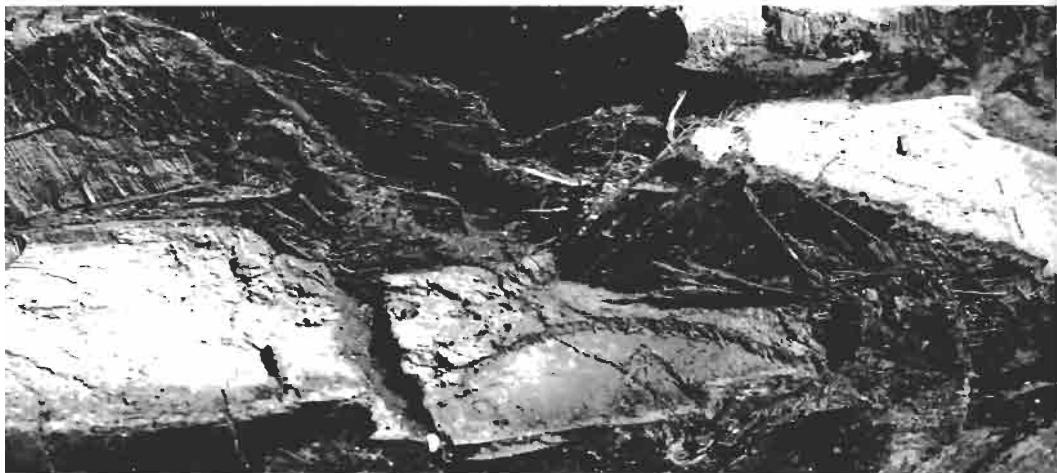
Figura 10.7. La rejilla una vez vinculada al DataViewManager

Como puede ver, gracias a la clase `DataViewManager` es posible mantener las relaciones en las vistas de datos, principalmente en componentes de interfaz como `DataGridView`, sin necesidad de gestionar por separado la selección de filas en cada `DataView` individual.

Resumen

La selección de parte de las filas de un conjunto de datos, obtenido previamente mediante una consulta, o la elección de un orden distinto, tradicionalmente ha requerido una comunicación con el servidor para facilitar el nuevo resultado. Gracias a la clase `DataView`, según ha podido verse en este capítulo, con ADO.NET es posible conseguir esas operaciones de manera mucho más rápida y sin establecer una conexión con el origen de datos.

En realidad, programando el acceso a datos desde código tal y como estamos haciendo en los ejemplos de éste y los capítulos previos, las vistas no son muy necesarias puesto que podemos conseguir resultados similares mediante el método `Select()` con el que cuentan los `DataTable`. A la hora de enlazar las filas con un componente de interfaz, un tema del que nos ocuparemos en un capítulo posterior, las cosas cambian, ya que no es posible vincular con un arreglo de `DataRow`, pero sí con un `DataView` o un `DataViewManager`.



11

XML

En el capítulo dedicado al estudio de los conjuntos de datos y clases relacionadas, `DataSet`, `DataTable`, `DataRow`, `DataColumn`, etc., se introdujeron algunos temas relativos al almacenamiento y recuperación de los datos, y su información de esquema, en forma de documentos XML y esquemas XSD. Sólo esa posibilidad nos permite, como se vio en dicho capítulo, trabajar con datos en una aplicación almacenándolos localmente, sin necesidad de ninguna base de datos ni servidor.

La integración entre ADO.NET y XML, sin embargo, puede llegar a ser mucho más estrecha como se verá en este capítulo, haciendo posible el acceso sincronizado a los mismos datos desde un `DataSet` y en forma de documento XML. Esto posibilita, por ejemplo, la transformación de los datos mediante XSLT o búsquedas más elaboradas que las que permite el método `Select()` de un `DataTable`.

También existe la posibilidad de tratar como un `DataSet` información alojada en un documento XML, sin que necesariamente tenga el formato propio de ADO.NET. Esto abre las puertas a la integración entre aplicaciones al ser XML un lenguaje adoptado como estándar mundial.

XML y temas relacionados

Antes de introducirnos en el uso de las clases .NET para la creación, lectura y manipulación de documentos XML, vamos a hacer un rápido recorrido por algunos conceptos de XML y otros temas relacionados, como DOM, SAX, XSL o XPath.

Ante todo, tenga en cuenta que este libro se centra en el tema del acceso a datos desde Visual Basic .NET usando principalmente ADO.NET, por lo que no se aporta un estudio detallado de XML como el que podría encontrar en un título más específico sobre este tema. También puede encontrar abundante información sobre XML en <http://MSDN.Microsoft.com/xml>.

El entorno de Visual Studio .NET cuenta con diseñadores y asistentes que facilitan la mayoría de las tareas que implica el trabajo con XML, como la edición de los documentos, creación de esquemas XSD u hojas de estilo XSL. Algunas de estas herramientas las conoció en el tercer capítulo, al crear los archivos *Libros.xsd* y *Libros.xml*.

Breve introducción a XML

XML es un lenguaje estándar, cuya especificación podemos encontrar en la sede del W3C (<http://www.w3.org>), cuyo objetivo es facilitar el intercambio de información entre aplicaciones, sin importar las diferencias entre plataformas hardware, sistemas operativos, lenguajes de programación ni idioma. Algunos expertos denominan a este lenguaje como el nuevo ASCII, sistema universal de codificación de caracteres reconocido por la práctica totalidad de los sistemas.

Para conseguir esta compatibilidad tan amplia, XML se describe con marcas codificadas en texto simple, nada de formatos binarios más compactos pero incompatibles, generalmente Unicode. Esto facilita el uso de caracteres de cualquier idioma, incluidos aquellos que, históricamente, han estado marginados en el campo de las tecnologías de la información. En este aspecto, podríamos comparar XML con HTML, al ser ambos formatos aplicables de manera universal.

A diferencia de HTML, sin embargo, las marcas de XML, salvo un conjunto base, no están predefinidas. Cualquiera puede definir su conjunto de marcas o etiquetas XML y aplicarlas a sus necesidades específicas. Esta capacidad es en sí una necesidad, ya que XML no se emplea para definir la apariencia de los datos que contiene, como es el caso de HTML, sino para describir su estructura.

Los documentos XML son muy fáciles de leer, ya que la estructura prácticamente se describe por sí sola. En el siguiente documento, fragmento del utilizado en el tercer capítulo como ejemplo, es fácil ver que existen dos elementos principales, *Libros* y *Editoriales*, que se repiten conteniendo información de distintos libros y editoriales. Ambos elementos están contenidos en una raíz que, en este caso, también se denomina *Libros*.

```
<?xml version="1.0" encoding="utf-8" ?>
<Libros xmlns="http://tempuri.org/Libros.xsd">
  <Libros>
    <IDLibro>14</IDLibro>
    <ISBN>84-415-1145-4</ISBN>
    <Titulo>Introducción a la programación</Titulo>
    <Autor>Francisco Charte</Autor>
    <Editorial>1</Editorial>
    <Precio>24.04</Precio>
```

```

</Libros>
<Libros>
  <IDLibro>6</IDLibro>
  <ISBN>84-415-1351-1</ISBN>
  <Titulo>Programación con Visual Basic .NET</Titulo>
  <Autor>Francisco Charte</Autor>
  <Editorial>1</Editorial>
  <Precio>39</Precio>
</Libros>
<Editoriales>
  <IDEEditorial>1</IDEEditorial>
  <Nombre>Anaya Multimedia</Nombre>
  <Direccion>Juan Ignacio Luca de Tena, 15</Direccion>
</Editoriales>
<Editoriales>
  <IDEEditorial>2</IDEEditorial>
  <Nombre>McGraw-Hill</Nombre>
  <Direccion>Edificio Valreality, 1a planta</Direccion>
</Editoriales>
</Libros>

```

The screenshot shows a Microsoft Internet Explorer window with the following details:

- Title Bar:** Extensible Markup Language (XML) - Microsoft Internet Explorer
- Menu Bar:** Archivo, Edición, Ver, Favoritos, Herramientas, Ayuda
- Toolbar:** Back, Forward, Stop, Refresh, Home, Favorites, Help, etc.
- Address Bar:** Dirección: <http://www.w3.org/XML/>
- Content Area:**
 - W3C ARCHITECTURE domain logo:** W3C logo next to the text "ARCHITECTURE domain".
 - Section Header:** Extensible Markup Language (XML)
 - Links:** Core Drafts, Developer Discussion, Events/Pubs (translations), Software, Test Suite, Bookmarks.
 - Text:** The Extensible Markup Language (XML) is the universal format for structured documents and data on the Web. [XML in 10 points](#) explains XML briefly. The base specifications are [XML 1.0](#), W3C Recommendation Feb '98, and [Namespaces](#), Jan '99. The [XML Activity Statement](#) explains the W3C's work on this topic in more detail. For related work, see: [Nearby](#).
 - Working Drafts:** XML Protocol, XML Schema, XML Query, XLink, XPointer, XML Base, DOM, RDF, CSS XSL, XHTML, MathML, SML, SVG, XML Signature and Canonicalization.
 - Other Sections:** XML Inclusions (XInclude), XML Information Set Proposed Recommendation issued 10 August 2001, XML Fragment Interchange Candidate Recommendation as of 12 February 2001.

Figura 11.1. Sede del W3C dedicada al lenguaje XML

Los documentos XML deben estar bien formados y ser válidos. Un documento está bien formado si todas las marcas de apertura cuentan con sus respectivas marcas de cierre, todos los atributos se introducen entre comillas dobles y el documento cuenta con un solo elemento raíz en el que están incluidos todos los demás. El fragmento anterior, por ejemplo, es un documento bien formado, por lo que un analizador XML podría leerlo y extraer datos sin mayores problemas. Éstas son normas del propio lenguaje XML y que, por tanto, debe cumplir cualquier documento XML.

Que un documento XML sea o no válido dependerá, primero, de que esté bien formado y, segundo, que su estructura concuerde con la definida en una DTD o esquema XSD. Estos elementos, las DTD y los esquemas XSD, se utilizan para validar documentos XML y verificar que son apropiados para la aplicación que debe procesarlos.

Definiciones de tipo y esquemas

A pesar de que a partir del contenido de un documento XML, como el mostrado en el punto anterior, es relativamente fácil deducir su estructura, si deseamos estar seguros de que los datos introducidos tienen los tipos adecuados y que la estructura es siempre la correcta, tendremos que servirnos de un documento de definición de tipos, conocido como DTD, o bien un esquema XSD.

Una DTD es un documento, no basado en XML, en el que se describe la estructura de los elementos de un documento XML, así como su tipo, si bien en este aspecto las limitaciones son bastante importantes. Un ejemplo de DTD, definiendo una estructura similar a la del documento XML previo, sería el siguiente:

```
<!ELEMENT Libreria (Libros, Editoriales)>
<!ELEMENT Libros (Libro+)
  <!ELEMENT Libro (IDLibro, ISBN, Titulo, Autor, Editorial, Precio)>
    <!ELEMENT IDLibro (#PCDATA)>
    <!ELEMENT ISBN (#PCDATA)>
    <!ELEMENT Titulo (#PCDATA)>
    <!ELEMENT Autor (#PCDATA)>
    <!ELEMENT Editorial (#PCDATA)>
    <!ELEMENT Precio (#PCDATA)>
<!ELEMENT Editoriales (Editorial+)
  <!ELEMENT Editorial (IDEitorial, Nombre, Direccion)>
    <!ELEMENT IDEitorial (#PCDATA)>
    <!ELEMENT Nombre (#PCDATA)>
    <!ELEMENT Direccion (#PCDATA)>
```

Esta definición puede colocarse al inicio del propio documento XML o bien almacenarse separadamente, en un archivo con extensión `.dtd`, haciéndose referencia a él desde el documento XML mediante la marca `<!DOCTYPE>`. El inconveniente de las DTD es que no siguen la sintaxis propia de XML y no son demasiado flexibles, por ejemplo al no contar con tipos de datos más específicos ya que en `#PCDATA` entra cualquier secuencia de caracteres, sin importar cuáles sean ni su longitud.

información en forma de documentos XML, por lo que debe existir un mecanismo para distinguir las etiquetas que, coincidiendo en nombre porque eso podría suceder, pertenezcan a aplicaciones o empresas distintas.

Aquí es donde entran en juego los ámbitos con nombre o *namespaces*, que nunca debe confundir con los ámbitos con nombre de Visual Basic. Los ámbitos con nombre se asocian siempre con un URL que, por regla general, suele apuntar a un esquema XSD, aunque esto no es imprescindible. La definición se efectúa en la marca raíz del documento utilizando el atributo `xmlns`. Éste irá seguido, tras dos puntos, del nombre del ámbito, disponiéndose el URL tras un signo `=`. A partir de ese momento, los elementos usados en el documento se precederán del nombre del ámbito al que correspondan, siendo posible emplear varios en un mismo documento.

En el esquema XSD del punto previo puede ver cómo se define el ámbito con nombre `xs`, asociándolo con el URL <http://www.w3.org/2001/XMLSchema>, y cómo se usa en el resto del documento como prefijo de las marcas propias de XSD.

Nota

Visual Studio .NET, al producir nuevos documentos XML, emplea por defecto el URL <http://tempuri.org> asociado a los ámbitos con nombre. Este URL debe modificarse por uno que identifique de manera única a ese ámbito, generalmente con el nombre de dominio de nuestra empresa y algún identificador adicional, dando lugar a un URI único.

Mecanismos de manipulación de documentos XML

A pesar de que los documentos XML, y los esquemas XSD, son relativamente fáciles de comprender y pueden ser editados manualmente, incluso con herramientas tan sencillas como el Bloc de notas de Windows, lo cierto es que están pensados para que sean aplicaciones a medida las que los manipulen. Estas aplicaciones podrían, ya que son archivos de texto simples, leerlos e intentar analizarlos por sí mismas, pero no es necesario gracias a la existencia de estándares como DOM y SAX.

SAX, como su propio nombre indica, es un conjunto simple de funciones que facilitan la extracción de datos a partir de un documento XML generando eventos a medida que lo lee, de tal forma que la aplicación, en cierta forma, es partícipe del análisis del documento. DOM es una alternativa más compleja, si bien también más flexible, que recupera un documento completo y nos ofrece la posibilidad de navegar por sus nodos, leyendo y modificando información. Para ello, a partir de la información leída del documento se genera un árbol jerárquico con todos los elementos.

En Visual Basic .NET, afortunadamente, no tenemos por qué utilizar métodos de relativo bajo nivel para operar sobre documentos XML, aunque realmente nada nos impide hacerlo. Por una parte disponemos de la clase `XmlReader` que, de

manera similar a SAX, nos permite tratar un documento XML como un flujo que va leyéndose paso a paso. Por otra tenemos la clase `XmlDocument`, que implementa los niveles 1 y 2 de DOM, mediante la cual es posible leer un documento completo y navegador por él.

Además, como ya sabemos, podemos también usar un `DataSet` para tratar un documento XML como si fuese un conjunto de datos corriente.

Selección de datos con XPath

La navegación de datos ofrecida por DOM es bastante secuencial, teniendo que ir accediendo de un nodo a sus hijos, y de uno de éstos a sus hijos repitiendo el proceso hasta llegar a la información deseada. XPath, otro estándar del W3C, es una solución más directa para la selección de datos, si bien con una sintaxis diferente, basada en caminos similares a los que componen un URI.

XPath crea internamente un árbol compuesto por los elementos y atributos del documento XML, facilitando el acceso a ellos mediante caminos como el siguiente:

```
/descendant::Libros[1@Libro="5"]/*
```

Asumiendo que tenemos una referencia a la raíz, con esta cadena buscaríamos el libro cuyo identificador sea 5 y seleccionaríamos todos los elementos hijo.

Mediante la clase `XPathNavigator` es posible utilizar XPath desde aplicaciones Visual Basic .NET. Esta clase opera sobre un documento XML representado por un objeto `XmlDocument`. Puede utilizarse la clase `XPathDocument`, que es una combinación de las dos anteriores.

Transformación con XSLT

En ocasiones es necesario generar, a partir de ciertos datos, un documento con una estructura concreta, por ejemplo una página HTML a partir de un conjunto de datos. Como veremos en un capítulo posterior, existen componentes para formularios Web que pueden conectarse con un `DataSet` y ofrecer capacidades de visualización y edición de los datos.

Un documento XML, sin embargo, no puede conectarse con esos controles y, aunque pudiéramos hacerlo, conectando el documento con un `DataSet`, puede darse el caso de que el formato de documento a generar no sea precisamente ése, sino otro distinto. Gracias a XSLT es posible generar, a partir de un documento XML, documentos distintos, por ejemplo una página HTML, un documento para dispositivos WAP o un documento XML con diferente estructura.

XSLT es un lenguaje mediante el cual se aplican plantillas a datos seleccionados con XPath. Su uso más habitual lo encontraremos en las hojas de estilo XSL, cuya finalidad, que no sintaxis, podría compararse con las hojas de estilo en cascada o bien CSS.

Desde Visual Basic .NET puede utilizar XSLT a través de la clase `XslTransform`, aplicando una hoja de estilo XSL a cualquier documento XML que tenga alojado en un `XmlDocument`, o derivado, para el que exista un `XPathNavigator`.

El ámbito System.Xml

La mayoría de las clases dirigidas a operar sobre documentos XML, esquemas XSD y emplear XPath y XSLT se encuentran en el ámbito `System.Xml` o alguno de sus subámbitos. En la figura 11.2 se han representado las clases más representativas, cada una de ellas contenido en el ámbito correspondiente. Como puede ver, el mayor número de clases, y las de utilización más habitual, están en `System.Xml`, siendo el resto de los ámbitos específicos para la gestión de esquemas, persistencia de datos, uso de XPath y transformación de documentos, respectivamente.

Como siempre, no olvide consultar la documentación electrónica de Visual Basic .NET para encontrar información detallada sobre cada uno de los miembros de éstos ámbitos. En los puntos siguientes se explica el uso de algunas clases, mostrándose ejemplos prácticos, pero no se intenta facilitar una referencia de todas las clases y sus miembros ya que ésa es una información que ya tiene a su alcance.

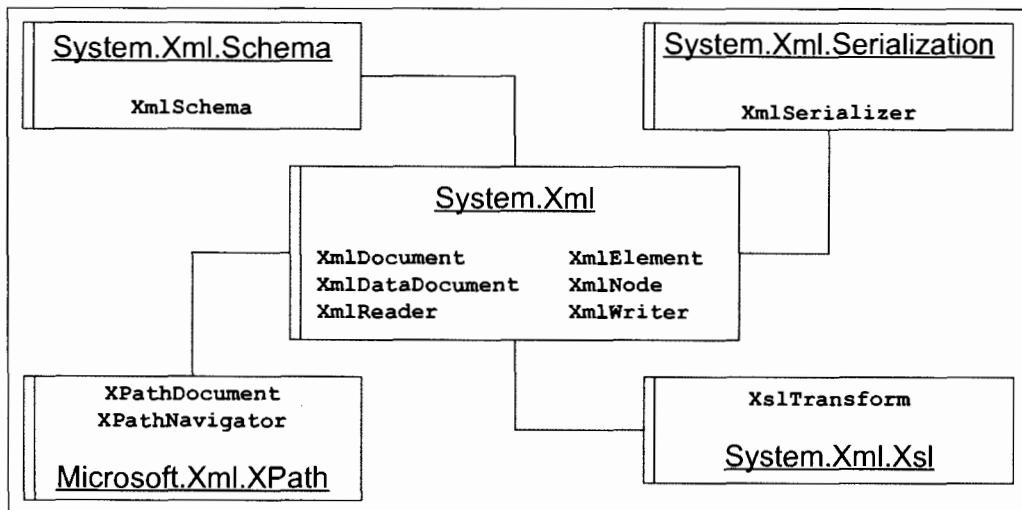


Figura 11.2. Ámbitos relacionados con XML y algunas de sus clases

Lectura secuencial de documentos XML

Para leer un documento XML de manera secuencial, nodo a nodo y elemento a elemento, tenemos a nuestra disposición las clases derivadas de `XmlReader` que, en cierta manera, son equivalentes a las clases `Reader` implementadas por los

proveedores ADO.NET, si bien sus métodos y propiedades son otros. Este método para acceder a un documento XML sería similar al uso de SAX, pero en lugar de responder a eventos que se generan a medida que se encuentran los diversos elementos, como ocurre con SAX, disponemos de un cursor sólo de lectura y unidireccional, sólo puede avanzar, para recuperar esos elementos.

En la figura 11.3 puede ver la clase `XmlReader` y sus derivadas, así como la interfaz que implementan dos de ellas y cuyo objetivo es facilitar información de posición de los elementos en el documento. `XmlTextReader` es el medio más rápido para leer un documento XML, al no efectuarse validación alguna respecto a una DTD o esquema XSD. Estas funciones sí las tiene `XmlValidatingReader`.

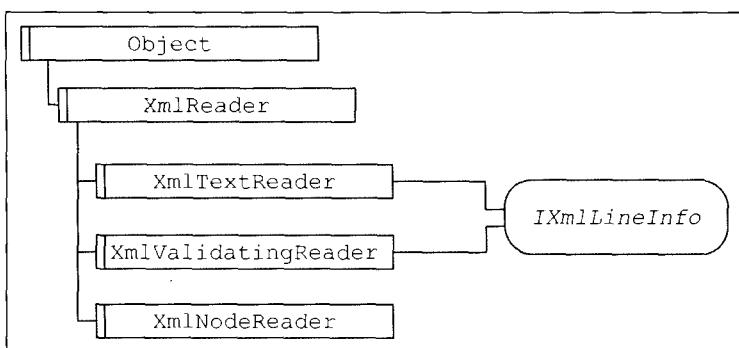


Figura 11.3. Clases derivadas de `XmlReader`

Al derivar de una base común, `XmlTextReader`, `XmlValidatingReader` y `XmlNodeReader` comparten un buen número de miembros. Centrándonos en las dos primeras, que nos permiten leer un documento XML desde un flujo de datos sin o con validación, su constructor puede tomar como primer argumento, entre otras opciones, una cadena con un camino y nombre de archivo, un `Stream` o un `TextReader`.

Teniendo acceso al documento XML a recorrer, que incluso podría ser un fragmento XML preparado en memoria, el método `Read()` nos permite ir avanzando por sus elementos. En cada momento, el cursor interno del `XmlReader` apuntará a un elemento del documento, al que tenemos acceso mediante múltiples propiedades para, por ejemplo, conocer su tipo, nombre, prefijo si está asociado a un ámbito con nombre, etc. También tenemos a nuestra disposición múltiples métodos del tipo `ReadXXXX()` para leer el texto de un elemento en distintos formatos.

Suponiendo que queremos tan sólo mostrar por la consola el contenido del documento XML `Libros.xml`, usado como ejemplo en el tercer capítulo, podríamos usar el código siguiente. Como es lógico, en la práctica efectuaríamos algún proceso de la información en lugar de imprimirla simplemente, pero el proceso de recuperación será idéntico al mostrado aquí.

```
' Ámbito donde está XmlReader y derivadas
Imports System.Xml
```

```

Module Module1
Sub Main()
    ' Obtenemos el documento XML directamente desde
    ' el archivo, asumiendo que está en nuestra carpeta
    Dim Lector As New XmlTextReader("Libros.xml")

    ' Mientras haya información a tratar
    While Lector.Read()
        ' Dependiendo del tipo de nodo lo
        ' procesaremos de una forma u otra
        Select Case Lector.NodeType
            Case XmlNodeType.Element ' si es un elemento
                ' Introducimos el sangrado y el nombre
                ' del elemento entre <>
                Console.WriteLine(New String(" ", Lector.Depth * 3) & _
                    "<{0}>", Lector.Name)

                ' En caso de que sea una marca de elemento principal
                If Lector.Name="Libros" Or Lector.Name="Editoriales" Then
                    Console.WriteLine() ' Introducir un salto
                End If

            Case XmlNodeType.Text ' si es un texto
                Console.WriteLine(Lector.Value) ' Es importante

            Case XmlNodeType.EndElement ' si es el final de un elemento
                ' En caso de que sea el último de los elementos
                ' principales, añadiremos el cierre de la lista de fino
                If Lector.Name="Libros" Or Lector.Name="Editoriales" Then
                    Console.WriteLine(New String(" ", Lector.Depth * 3))
                End If
                ' y finalmente el cierre de lista de fino
                Console.WriteLine("</{0}>", Lector.Name)
            End Select
        End While

        Lector.Close() ' Cerrando el lector
    End Sub

End Module

```

La ejecución de este código debería generar un resultado idéntico al mostrado en la figura 11.4. Básicamente es el documento XML original, pero no lo hemos leído directamente como si fuese un texto sino mediante un `XmlReader`. Esto nos permite distinguir los tipos de nodo y efectuar otras operaciones, como saber si el elemento tiene o no atributos, conocer su número, separar el prefijo de ámbito del nombre de los elementos, etc. En el ejemplo anterior se ha usado `NodeType` para saber el tipo de nodo, `Name` para recuperar su nombre, `Depth` para conocer la profundidad en el documento y `Value` para recuperar un valor de un nodo de texto.

En caso de que deseemos tratar un documento XML previa validación contra una DTD o un esquema XSD, en lugar de `XmlTextReader` emplearíamos la clase `XmlValidatingReader`, indicando en la propiedad `ValidationType` el tipo de

validación a efectuar y añadiendo el esquema a Schemas o facilitando la información necesaria en XmlResolver para acceder a la DTD.

```

<Libros>
  <Libro>
    <IDLibro>15</IDLibro>
    <ISBN>94-2615-234-5</ISBN>
    <Titulo>Manual del microprocesador 80386</Titulo>
    <Autor>Chris H.Pappas&William H.Murray, III</Autor>
    <Editorial>I</Editorial>
    <Precio>40</Precio>
  </Libro>
  <Libro>
    <IDLibro>14</IDLibro>
    <ISBN>94-415-1145-4</ISBN>
    <Titulo>Introducción a la programación</Titulo>
    <Autor>Francisco Charte</Autor>
    <Editorial>I</Editorial>
    <Precio>24.04</Precio>
  </Libro>
  <Libro>
    <IDLibro>13</IDLibro>
    <ISBN>94-415-1132-2</ISBN>
    <Titulo>Guía práctica para usuarios de Kylix</Titulo>
    <Autor>Francisco Charte</Autor>
    <Editorial>I</Editorial>
    <Precio>10.52</Precio>
  </Libro>
  <Libro>
    <IDLibro>12</IDLibro>
    <ISBN>94-415-1202-2</ISBN>
    <Titulo>Guía práctica para usuarios de Excel 2002</Titulo>
    <Autor>Francisco Charte&M.Jesús Luque</Autor>
    <Editorial>I</Editorial>
    <Precio>10.52</Precio>
  </Libro>
  <Libro>
    <IDLibro>11</IDLibro>
    <ISBN>94-415-1230-2</ISBN>
    <Titulo>Manual avanzado Excel 2002</Titulo>
    <Autor>Francisco Charte</Autor>
    <Editorial>I</Editorial>
    <Precio>21.04</Precio>
  </Libro>
  <Libro>
    <IDLibro>10</IDLibro>
    <ISBN>94-415-1255-9</ISBN>
    <Titulo>Guía práctica para usuarios de Delphi 6</Titulo>
    <Autor>Francisco Charte</Autor>
    <Editorial>I</Editorial>
    <Precio>19.52</Precio>
  </Libro>
  <Libro>
    <IDLibro>9</IDLibro>
    <ISBN>94-415-1261-2</ISBN>
    <Titulo>Programación con Delphi 6 y Kylix</Titulo>
    <Autor>Francisco Charte</Autor>
    <Editorial>I</Editorial>
    <Precio>37.26</Precio>
  </Libro>
  <Libro>
    <IDLibro>8</IDLibro>
    <ISBN>94-415-1291-4</ISBN>
    <Titulo>Guía práctica para usuarios de Visual Studio .NET</Titulo>
    <Autor>Francisco Charte</Autor>
    <Editorial>I</Editorial>
    <Precio>40.52</Precio>
  </Libro>
  <Libro>
    <IDLibro>7</IDLibro>
    <ISBN>94-415-1290-6</ISBN>
    <Titulo>Guía práctica para usuarios de Visual Basic .NET</Titulo>
    <Autor>Francisco Charte</Autor>
    <Editorial>I</Editorial>
    <Precio>10.75</Precio>
  </Libro>
  <Libro>

```

Figura 11.4. El documento XML procesado por el código de ejemplo

Manipulación de documentos XML con DOM

Las clases derivadas de `XmlReader` están dirigidas, tal como indica su propio nombre, a la lectura de información, pero en ningún caso facilitan mecanismos

para modificar el contenido de los nodos, eliminar o añadir elementos. Podríamos efectuar este trabajo manualmente en el programa, mediante código, y luego usar un `XmlWriter` para volver a generar el documento, pero existen alternativas más eficientes y simples.

Una de esas opciones es la clase `XmlDocument`, en la que se implementa el DOM nivel 1 y 2. Esta clase aloja el documento XML completo en memoria y ofrece facilidades de navegación, lectura, modificación, inserción y borrado. En la figura 11.5 puede ver que `XmlDocument` está derivada de `XmlNode`, clase que implementa la interfaz `IXPathNavigable`, actuando a su vez como base de `XmlDataDocument`, una clase que usaremos posteriormente para sincronizar la información de un documento XML con un `DataSet`.

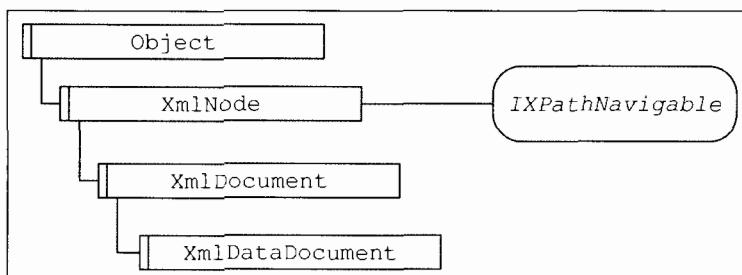


Figura 11.5. La clase `XmlDocument` y relacionadas

`XmlDocument` recupera todo el documento XML y genera un árbol jerárquico en memoria, compuesto por los nodos y elementos que existan en dicho documento. La recuperación de éste puede efectuarse desde un archivo en disco, un flujo o `Stream`, un `XmlReader`, un `TextReader`, etc. El método `Load()` cuenta con varias versiones que aceptan distintos parámetros, según nos interese en cada caso.

Disponiendo del documento en el objeto `XmlDocument`, mediante la propiedad `DocumentElement` obtendríamos el primer elemento, el que actúa como raíz de todos los demás. Éste es de una clase también derivada de `XmlNode`, por lo que, en principio, tenemos muchas de las propiedades y métodos con que cuenta `XmlDocument`, entre ellas:

- `NodeType`: Contiene el tipo de nodo.
- `Name`: Aloja el nombre del nodo.
- `Value`: Almacena el valor del nodo.
- `Attributes`: Atributos del nodo actual.
- `HasChildNodes`: Devuelve `True` si el nodo que está tratándose tiene nodos hijo.
- `FirstChild`: Devuelve el primer nodo hijo del nodo actual.
- `ChildNodes`: Recupera la lista de nodos hijo del nodo actual.

- `LastChild`: Devuelve el último nodo hijo del nodo actual.
- `ParentNode`: Devuelve una referencia al nodo que actúa como padre del actual.
- `NextSibling`: Obtiene el nodo siguiente al actual en el mismo nivel.
- `PreviousSibling`: Obtiene el nodo anterior al actual en el mismo nivel.
- `AppendChild()`: Añade un nodo hijo al actual.
- `InsertBefore() / InsertAfter()`: Inserta un nuevo nodo antes o después, respectivamente, del indicado.
- `RemoveChild()`: Elimina un nodo hijo del nodo actual.
- `ReplaceChild()`: Sustituye un nodo hijo existente por otro.
- `SelectNodes()`: Selecciona un conjunto de nodos a partir de un camino XPath.
- `CreateNavigator()`: Devuelve un `XPathNavigator` asociado al documento, para recorrerlo.

A parte de éstos, heredados de `XmlNode`, la clase `XmlDocument` cuenta con múltiples miembros adicionales que facilitan, por ejemplo, la creación de un nodo, la obtención de elementos a partir de su identificador o la escritura del documento de vuelta a un archivo o flujo de datos.

Supongamos que quiere actualizar el documento XML que contiene los datos de los libros, incrementando los precios un 10 por ciento al tiempo que se muestra por consola el título de cada libro, el antiguo precio y el nuevo. El código necesario sería el siguiente:

```
' Necesitamos las clases XML
Imports System.Xml

Module Module1
    Sub Main()
        ' Documento XML
        Dim Documento As New XmlDocument()
        Dim Nodo As XmlNode ' un nodo

        ' Dos de los datos de un nodo
        Dim Titulo As String, Precio As Decimal

        ' Recuperamos el documento
        Documento.Load("Libros.xml")

        ' y recorremos todos los nodos hijo que existan
        ' en el elemento principal
        For Each Nodo In Documento.DocumentElement.ChildNodes
            ' si el nodo es un libro
            If Nodo.Name = "Libros" Then
                ' extraemos el título y el precio
```

```

    Titulo = Nodo.ChildNodes(2).InnerText
    Precio = Nodo.ChildNodes(5).InnerText
    ' Si el título es "El Principito"
    If Titulo = "El Principito" Then
        Console.WriteLine("{0,-50} {1,5} -> {2,5}",_
            Titulo, Precio, Precio * 1.1)
        ' Actualizamos el precio en el XML
        Precio = Precio * 1.1
        ' Y lo asignamos al nodo
        Nodo.ChildNodes(5).InnerText = Precio
    End If
    Next

    ' Guardamos los cambios realizados en el documento
    Documento.Save("NuevosPrecios.xml")
End Sub

End Module

```

Tras recuperar el documento XML, mediante el método `Load()`, obtenemos la lista de nodos existentes en el elemento principal. `DocumentElement` es ese nodo principal, `<Libros>`, y `ChildNodes` nos ofrece la colección de todos los `XmlNode` que contiene. Comprobamos si el nodo corresponde a un libro y, en caso afirmativo, recuperamos el texto que hay en las columnas tercera y última, título y precio. Una vez mostrados en la consola, actualizamos el precio y lo asignamos al nodo. Finalmente, guardamos el documento en un nuevo archivo. En la figura 11.6 puede ver el resultado generado por el programa, y en la figura 11.7 aparece el documento `NuevosPrecios.xml` en Internet Explorer, con los precios ya modificados.

Manual del microprocesador 80386	40 -> 44
Introducción a la programación	24.04 -> 26.444
Guía práctica para usuarios de Kylix	10.52 -> 11.572
Guía práctica para usuarios de Excel 2002	10.52 -> 11.572
Manual avanzado Excel 2002	21.04 -> 23.144
Guía práctica para usuarios de Delphi 6	10.52 -> 11.572
Programación con Delphi 6 y Kylix	37.26 -> 40.986
Guía práctica para usuarios de Visual Studio .NET	10.52 -> 11.572
Guía práctica para usuarios de Visual Basic .NET	10.75 -> 11.825
Programación con Visual Basic .NET	39 -> 42.9
Programación con Visual Studio .NET	40 -> 44
Programación con Visual C# .NET	39 -> 42.9
Guía práctica para usuarios JBuilder 7	10.75 -> 11.825
User Interface Design for Programmers	31 -> 34.1
SQL Server 2000	10.75 -> 11.825

Figura 11.6. Resultado producido por el programa

Con XPath podemos seleccionar subconjuntos de información de un documento XML, empleando para ello caminos de selección. En la sede del W3C encontrará una definición formal de este lenguaje, realmente complejo y flexible. Básicamente, empleamos una / para indicar la raíz del documento, introduciendo a continuación el camino necesario hasta llegar a la información que necesitamos. Por ejemplo:

```
/Libros/Editoriales/*  
/Libros/Editoriales/Nombre
```

La primera línea seleccionaría todos los elementos de las editoriales, mientras que con la segunda tendríamos sólo los elementos Nombre de todas las editoriales. Es posible introducir, entre otras variantes, condicionales de selección, por ejemplo:

```
/Libros/Libro[Editorial=1]/Titulo
```

En este caso seleccionaríamos el título de todos aquellos libros cuya columna Editorial tenga el valor 1. Sería similar a la consulta SQL SELECT Titulo FROM Libros WHERE Editorial=1.

Nota

Para obtener un comportamiento correcto con XPath, el archivo XML original usado como ejemplo se ha modificado para que los nodos correspondientes a los libros se delimiten con las marcas <Libro> y </Libro>, evitando la coincidencia con la marca raíz <Libros> ya que eso confunde a XPath.

Las clases que necesitamos para operar sobre un documento con XPath, aparte de XmlDocument, son dos: XPathNavigator y XPathNodeIterator. Ambas se encuentran en el ámbito System.Xml.XPath. El método Select() de la primera acepta como parámetro la expresión XPath, un camino como los mostrados anteriormente a modo de ejemplo, devolviendo un objeto XPathNodeIterator que servirá para recorrer los nodos resultantes. Este objeto es como un cursor que mantiene un nodo actual, en la propiedad Current, y nos permite ir avanzando a los siguientes que forman el conjunto de filas: MoveNext().

Usando de nuevo el mismo documento XML de ejemplo, con los cambios indicados en la nota previa, podríamos usar el siguiente programa para extraer el resultado que se ve en la figura 11.8: el título de todos los libros pertenecientes a la editorial que tiene el código 1.

```
' Necesitamos las clases XML  
Imports System.Xml  
Imports System.Xml.XPath  
  
Module Module1  
Sub Main()
```

```

' Creamos el documento y el navegador XPath
Dim Documento As New XmlDocument()
Documento.Load("Libros.xml")
Dim Navegador As XPathNavigator = Documento.CreateNavigator()

' Seleccionamos los datos obteniendo un iterador
Dim Resultado As XPathNodeIterator =
    Navegador.Select("/Libros/Libro[Editorial=1]/TITULO")

' Mostramos el número de nodos obtenidos
Console.WriteLine("Hay {0} títulos" & vbCrLf, Resultado.Count)

' y los recorremos mostrando el valor
While Resultado.MoveNext()
    Console.WriteLine("{0}", Resultado.Current.Value)
End While
End Sub

End Module

```



Figura 11.8. Lista de títulos de la editorial con código 1

Transformación de documentos

Los documentos XML pueden ser transformados en documentos de otro tipo, según se anotaba en un punto previo, gracias a XSLT. El proceso se basa en la construcción de una plantilla XSLT que, tras ser procesada por el analizador XSLT, genera el nuevo documento a partir de la información de origen. En esa plantilla se combinan expresiones XPath, para seleccionar los datos, con instrucciones de proceso XSLT tales como template, for-each o value-of. Este lenguaje, como todo lo relacionado con XML, se encuentra definido en especificaciones del W3C y

cuenta con sentencias condicionales y de repetición, como los lenguajes de programación. El documento mostrado a continuación es una plantilla XSLT para transformación de documentos, concretamente para generar una tabla HTML con el nombre y precio de los libros del documento XML de ejemplos previos:

```
<?xml version='1.0' ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
    <xsl:template match="/">
        <html>
            <body>
                <table>
                    <xsl:for-each select="Libros/Libro">
                        <tr>
                            <td>
                                <xsl:value-of select="Titulo" />
                            </td>
                            <td>
                                <xsl:value-of select="Precio" />
                            </td>
                        </tr>
                    </xsl:for-each>
                </table>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

Las dos primeras líneas son las que identifican al documento como una plantilla XSLT u hoja de estilo XSL. Todas las instrucciones XSLT van precedidas con el prefijo `xsl`, como puede apreciarse. La marca `<template>` aplica la plantilla que hay a continuación, hasta la etiqueta de cierre `</template>`, a todos los datos del documento, ya que la expresión XPath del atributo `match` selecciona todo su contenido. A continuación encontramos marcas HTML corrientes para la creación de una tabla. La etiqueta `<for-each>` es equivalente a un bucle `For/Next` de Visual Basic, repitiéndose tantas veces como elementos `Libro` existan en la rama `Libros`. Ya sabe que esta expresión puede cambiarla para seleccionar cualquier otro conjunto de datos, por ejemplo los títulos de una cierta editorial, según se vio antes. Finalmente, en el interior de las celdillas de la tabla HTML se introduce el valor de dos elementos de cada libro, leídos con `<value-of>`.

Aunque podríamos asociar esta plantilla de transformación directamente al documento XML, viendo el resultado en Internet Explorer dado que éste es capaz de analizar y ejecutar la transformación, lo que nos interesa es ver cómo podemos hacerlo desde un programa propio. El único elemento adicional que precisamos, aparte de los que ya conocemos, es la clase `XslTransform`. Ésta deriva directamente de `Object` y no implementa interfaz alguna. El método de mayor interés es `Transform()`, encargado de aplicar la transformación al documento. Previamen-
te, sin embargo, es preciso recuperar la plantilla XSLT mediante el método `Load()`, equivalente al homónimo de la clase `XmlDocument`.

El método `Transform()` puede tomar distintas listas de parámetros, siendo el formato más fácil el que tan sólo toma dos cadenas de caracteres conteniendo el nombre, incluyendo camino si es necesario, del archivo XML de origen y el archivo de destino. Otras opciones nos permiten usar flujos de datos, `XmlPathNavigator` como origen, etc.

Asumiendo que tenemos la plantilla de transformación anterior almacenada en un archivo llamado `TablaHTML.xslt`, con el sencillo programa mostrado a continuación generaríamos el documento HTML que puede verse en la figura 11.9. No es espectacular, pero sólo sería preciso añadir algunas marcas más a la plantilla para introducir un título de página, una tabla más vistosa o más información de la contenida en el documento XML. El resultado se adaptaría automáticamente a los cambios de la plantilla, sin que tuviésemos que modificar el programa. También existe la posibilidad de crear la plantilla XSLT *al vuelo*, en el propio programa, produciendo a partir del documento XML otro formato de documento sin necesidad de procesar elemento a elemento todo su contenido.

```
' Normalizamos las salidas XML
Imports System.Xml
Imports System.Xml.XPath
Imports System.Xml.Xsl

Module Module1
    Sub Main()
        ' Creamos el Transformador
        Dim Transformador As New XslTransform()

        ' Leemos la plantilla XSLT
        Transformador.Load("../TablaHTML.xslt")

        ' Ejecutamos la transformación
        Transformador.Transform("Libros.xml", "Resultado.html")
    End Sub
End Module
```

XML y ADO.NET

Las clases que acabamos de conocer tienen utilidad por sí mismas, ya que hacen posible la lectura y manipulación de documentos XML desde cualquier aplicación desarrollada con Visual Basic .NET, sin importar su finalidad. Muchas de las operaciones descritas, no obstante, encuentran su mayor aplicación en nuestro caso, centrados en el tema de acceso a datos, al sincronizar un documento XML con un `DataSet` o viceversa.

Suponga que ha creado un `DataSet`, mediante los adaptadores de datos adecuados, y tras tener los `DataTable` con las filas de datos quiere obtener sólo aquellas que cumplan una cierta condición, o bien generar una página HTML a partir del

DataSet para su presentación a través de la Web. Podríamos usar vistas, consultas SQL y generar los documentos mediante ASP.NET pero, en ciertos casos, el uso de XPath o XSLT pueden ser mejores opciones.

El caso inverso también puede darse. Podemos obtener un documento XML desde una aplicación externa, por ejemplo a través de Internet, y, mediante la sincronización con un DataSet, manipular su contenido como si se tratase de un conjunto de tablas de datos.

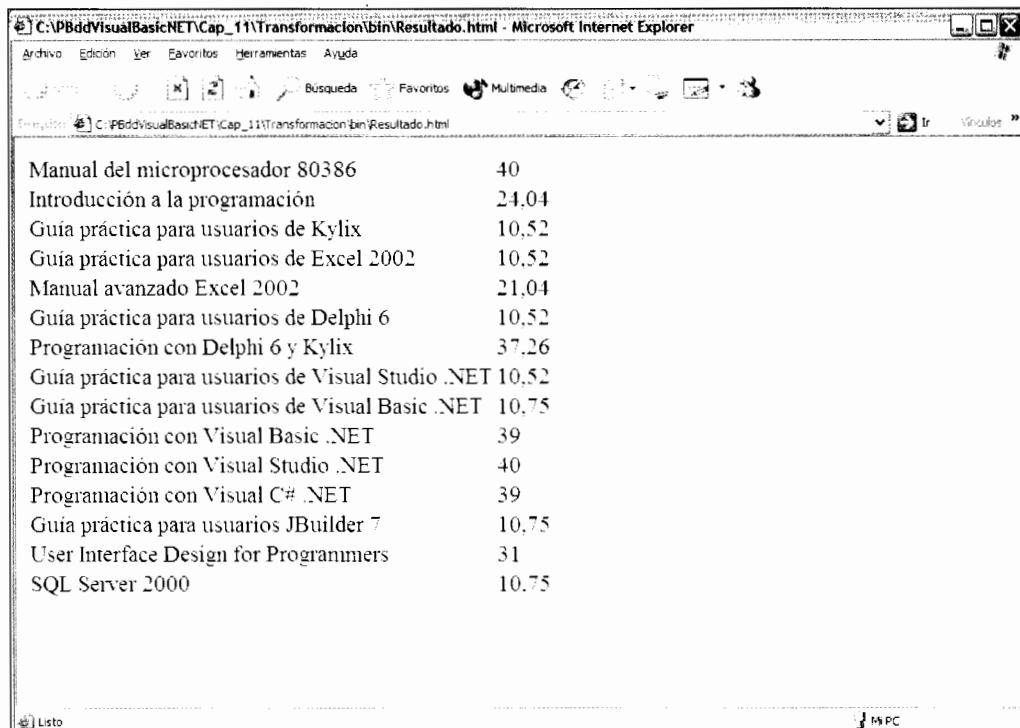


Figura 11.9. Documento HTML generado a partir de la transformación XSLT

La clase XmlDocument

Según se aprecia en la figura 11.5, XmlDocument es una clase derivada de XmlDocument por lo que, en principio, ya contamos con todos los miembros de ésta y sabemos, básicamente, cómo recuperar un documento XML y recorrerlo. A los miembros heredados, XmlDocument añade un propiedad, DataSet, y algunos métodos, como GetElementFromRow() o GetRowFromElement(), que facilitan la obtención de un elemento XML perteneciente a una cierta fila del DataSet o viceversa. El constructor de la clase también puede tomar como parámetro un DataSet, creándose la vinculación entre conjunto de datos y documento XML de manera inmediata.

El procedimiento de trabajo vinculado entre `DataSet` y `XmlDataDocument` puede desarrollarse, principalmente, de dos formas distintas:

1. Creación del `DataSet` vacío, recuperación de información de esquema, ya sea leyéndola de una definición XSD o bien creándola según se vio en un capítulo previo, y lectura de la información desde un documento XML a través del `XmlDataDocument`.
2. Creación del `DataSet` con información, ya sea obtenida mediante un adaptador o añadida por otros medios, creando a continuación el `XmlDataDocument` vinculado al `DataSet`.

Vamos a ver los dos procedimientos con un breve ejemplo en los dos puntos siguientes.

Creación del `DataSet` a partir del documento XML

El primer caso se dará cuando la información se nos facilite en formato XML, por ejemplo procedente de una aplicación externa a través de una red, y necesitemos introducirla en un `DataSet` para poder tratarla como un conjunto de tablas y, si nos interesase, incluso conectarla con componentes de interfaz de usuario.

Puede pensar que podríamos guardar la información en formato XML en un archivo y, posteriormente, recogerlo desde el `DataSet` mediante el método `ReadXml()`. Puede hacerse, pero con el inconveniente de que no se respeta el formato original que tuviese el documento XML. Si hacemos cambios y luego guardamos la información con `WriteXml()`, nada nos garantiza que el orden de los elementos, sus atributos y la estructura con que estaba escrito, por ejemplo separaciones y espacios en blanco, vayan a mantenerse.

Leyendo el documento en un `XmlDataDocument` estaremos seguros de que la estructura se mantiene, al tiempo que conseguimos el acceso desde un `DataSet`. Puede comprobarlo con el siguiente programa de ejemplo:

```
Imports System.Xml
Imports System.Data

Module Module1
    Sub Main()
        ' Creamos el DataSet
        Dim Datos As New DataSet()
        ' y recuperaremos el esquema
        Datos.ReadXmlSchema("Libros.xsd")

        ' Creamos el XmlDataDocument vinculándolo
        ' con el DataSet
        Dim Documento As New XmlDataDocument(Datos)

        ' Desactivamos la comprobación de restricciones
        Datos.EnforceConstraints = False
    End Sub
End Module
```

```
' y recuperamos el documento
Documento.Load("Libros.xml")

'Recuperamos el contenido del DataSet para mostrar todo su contenido
Dim Tabla As DataTable
Dim Fila As DataRow, columna As DataColumn

For Each Tabla In Datos.Tables
    Console.WriteLine(Tabla.TableName & vbCrLf)
    For Each Fila In Tabla.Rows
        For Each columna In Tabla.Columns
            Console.Write(Fila(columna) & vbTab)
        Next
        Console.WriteLine()
    Next
    Console.WriteLine()
Next

End Sub

End Module
```

Tras crear el DataSet vacío, recuperamos la información de esquema del archivo `Libros.xsd` creado en el tercer capítulo. De no disponer de este archivo tendríamos que obtener la estructura de algún otro modo, ya sea definiéndolo mediante código, porque lo conocemos de antemano, o bien deduciéndolo del contenido del propio documento XML.

A continuación creamos el `XmlDataDocument`. Observe cómo se pasa al constructor el `DataSet` recién creado. Acto seguido damos a la propiedad `EnforceConstraints` del `DataSet` el valor `False`, ya que durante la lectura del documento, que se efectúa en la sentencia siguiente, las restricciones de las columnas podrían no ser válidas y generar excepciones.

Finalmente, se lee el documento XML y muestra por consola el contenido del `DataSet`. El resultado será similar al de la figura 11.10. En vez de escribir los datos, como en este ejemplo, podríamos modificar, insertar, añadir, etc., y después devolver la información al documento XML con el método `Save()` de la clase `XmlDataDocument`.

Generación del documento XML a partir del DataSet

Este proceso es aún más simple si cabe, ya que basta con crear el `XmlDataDocument` vinculado al `DataSet`, entregando éste como argumento al constructor del primero, para tener automáticamente el documento XML que representa al contenido del conjunto de datos. ¿Para qué puede servirnos esto? Como se decía antes, podemos efectuar seleccionados de datos XPath sobre un `DataSet`, o aplicar una transformación a los datos para generar un nuevo documento. Son posibilidades que ha conocido brevemente en puntos anteriores.



Figura 11.10. Los datos obtenidos del DataSet a través del XmlDocument

A modo simplemente de ejemplo, el programa siguiente crea un DataSet y lo llena, mediante un adaptador de datos, con el contenido de la tabla Libros que teníamos en la base de datos SQL Server. A continuación crea el XmlDocument y lo muestra por consola. Por último, usando un XPathNavigator, se localizan todos los títulos de libros pertenecientes a la primera editorial. Observe la expresión XPath, en la que la raíz es MisDatos ya que ése es el nombre que hemos dado al DataSet, mientras que Libros es el nombre de la tabla.

```

Imports System.Xml
Imports System.Xml.XPath
Imports System.Data
Imports System.Data.SqlClient

Module Module1
    Sub Main()
        ' Definir una conexión
        Dim Conexion As New SqlConnection(
            "Data Source=inspiron; Initial Catalog=Libros; " &
            "User ID=sa; Password=")

        ' Crear comando
        Dim AdaptadorLibros As New SqlDataAdapter( _
            "SELECT * FROM Libros", Conexion)

        ' Abrir la conexión
        Datos As New DataSet("MisDatos")

        ' y la llenamos con la información devuelta
        AdaptadorLibros.Fill(Datos, "Libros")

        ' Creamos el XmlDocument vinculando con el DataSet

```

```
Dim Documento As New XmlDocument(Datos)

' Mostramos el documento XML completo
Documento.Save(Console.Out)

' Esperamos
Console.WriteLine(vbCrLf &
    "*** Pulse Intro para continuar ***")
Console.ReadLine()

' Creamos el navegador
Dim Navegador As XPathNavigator = Documento.CreateNavigator()

' Seleccionamos los datos obteniendo un iterador
Dim Resultado As XPathNodeIterator =
    Navegador.Select("//MisDatos/Libros[Editorial=1]/Titulo")

' Mostramos el número de nodos obtenidos
Console.WriteLine("Hay {0} títulos" & vbCrLf, Resultado.Count)

' y los recorriendo mostrando el título
While Resultado.MoveNext()
    Console.WriteLine("{0}", Resultado.Current.Value)
End While
End Sub

End Module
```

Al ejecutar el programa, y asumiendo que la conexión con el origen de datos es satisfactoria, verá, en un primer momento, cómo aparece toda la información por la consola en formato XML. Tras pulsar **Intro** se enumerarán los títulos de los libros pertenecientes a la primera editorial, como se aprecia en la parte inferior de la figura 11.11. Puede ampliar el programa, o basarse en él para crear otro, con el fin de que se aplique una transformación generando una tabla HTML con los datos de los libros.

Resumen

Como ha podido ver en este capítulo, especialmente en los últimos puntos, la integración entre XML y ADO.NET es tan estrecha que los conjuntos de datos pueden tratarse como documentos XML, y viceversa, con gran simplicidad. Los beneficios que se obtienen son muchos, ya que los conjuntos de datos pueden producir flujos de datos XML adecuados para ser remitidos a aplicaciones externas, que no trabajan con ADO.NET, invirtiéndose el proceso cuando es nuestra aplicación la que actúa como receptora.

La posibilidad de enviar conjuntos de datos en formato XML también es útil entre aplicaciones .NET, por ejemplo entre un servicio Web y una interfaz o entre un componente de negocio que se ejecuta en un servidor de aplicaciones y un cliente

remoto. En casos así el conjunto de datos se convierte en XML para ser transmitido por la red, invirtiéndose el proceso en el destino.

```

<?xml version="1.0" encoding="utf-8"?>
<Libros>
  <Libro>
    <IDLibro>11</IDLibro>
    <ISBN>84 415-1255-8</ISBN>
    <Titulo>Guía práctica para usuarios de Delphi 6</Titulo>
    <Autor>Francisco Charte</Autor>
    <Editorial>1</Editorial>
    <Precio>10.52</Precio>
  </Libro>
  <Libro>
    <IDLibro>12</IDLibro>
    <ISBN>84-415-1230-2</ISBN>
    <Titulo>Manual avanzado Excel 2002</Titulo>
    <Autor>Francisco Charte</Autor>
    <Editorial>1</Editorial>
    <Precio>21.04</Precio>
  </Libro>
  <Libro>
    <IDLibro>13</IDLibro>
    <ISBN>84-415 1202 7</ISBN>
    <Titulo>Guía práctica para usuarios de Excel 2002</Titulo>
    <Autor>Francisco Charte, M. Jesús Luque</Autor>
    <Editorial>1</Editorial>
    <Precio>10.52</Precio>
  </Libro>
  <Libro>
    <IDLibro>14</IDLibro>
    <ISBN>84-415-1132-2</ISBN>
    <Titulo>Guía práctica para usuarios de Kylix</Titulo>
    <Autor>Francisco Charte</Autor>
    <Editorial>1</Editorial>
    <Precio>10.52</Precio>
  </Libro>
  <Libro>
    <IDLibro>15</IDLibro>
    <ISBN>84-415-1145-4</ISBN>
    <Titulo>Introducción a la programación</Titulo>
    <Autor>Francisco Charte</Autor>
    <Editorial>1</Editorial>
    <Precio>24.04</Precio>
  </Libro>
  <Libro>
    <IDLibro>16</IDLibro>
    <ISBN>84-7615-234-5</ISBN>
    <Titulo>Manual del microprocesador 80386</Titulo>
    <Autor>Chris H. Papageorgiou, William H. Murray, III</Autor>
    <Editorial>2</Editorial>
    <Precio>40</Precio>
  </Libro>
  <Libro>
    <IDLibro>17</IDLibro>
    <ISBN>8-421-32523-3</ISBN>
    <Titulo>Assembly Language Step-by-Step</Titulo>
    <Autor>Jeff Dunteman</Autor>
    <Editorial>1</Editorial>
    <Precio>60.15</Precio>
  </Libro>
  <Libro>
    <IDLibro>18</IDLibro>
    <ISBN>84-415-1482-X</ISBN>
    <Titulo>Guía práctica ASP.NET</Titulo>
    <Autor>Isaac González</Autor>
    <Editorial>1</Editorial>
    <Precio>10.75</Precio>
  </Libro>
</Libros>
</MisDatos>
*** Pulse Intro para continuar ***

```

Hay 15 títulos

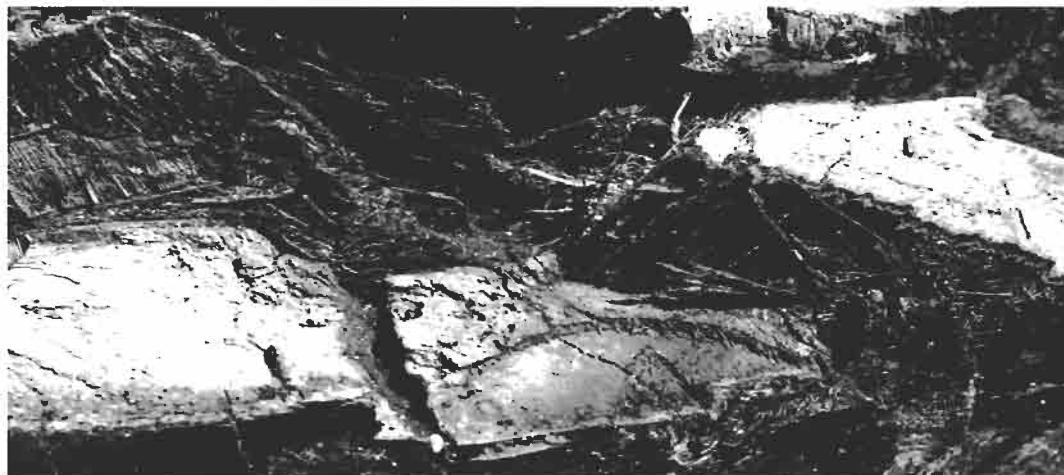
SQL Server 2000
 Guía práctica para usuarios JBuilder 7
 Programación con Visual C# .NET
 Programación con Visual Studio .NET
 Programación con Visual Basic .NET
 Guía práctica para usuarios de Visual Basic .NET
 Guía práctica para usuarios de Visual Studio .NET
 Programación con Delphi 6 y Kylix
 Guía práctica para usuarios de Delphi 6
 Manual avanzado Excel 2002
 Guía práctica para usuarios de Excel 2002
 Guía práctica para usuarios de Kylix
 Introducción a la programación
 Assembly Language Step-by-Step
 Guía práctica ASP.NET
 Press any key to continue...

Figura 11.11. Documento XML generado a partir del DataSet y resultado de la selección XPath

Por último, no hay que menospreciar la posibilidad de emplear técnicas como XPath y XSLT sobre información almacenada en un `DataSet`, como se ha demostrado en el último punto. En ocasiones una búsqueda de datos, conociendo XPath con detalle, puede ser mucho más eficiente a través de un `XmlDataDocument` que creando vistas o usando el método `Select()` de los `DataTable`.

Parte III

Visual Studio .NET



12

Capacidades de datos en Visual Studio .NET

Los capítulos de la segunda parte, que acaba de finalizar, le han servido para conocer un gran número de clases de ADO.NET, aprendiendo a utilizarlas en el código de sus programas sin recurrir a asistente o herramienta de diseño alguna. Saber qué clases son las que se emplean y cómo funcionan es importante, cuando surgen problemas imprescindible para resolverlos, pero también es cierto que las herramientas de diseño de Visual Studio .NET pueden ahorrarnos gran parte del trabajo que, hasta ahora, ha efectuando escribiendo código.

El objetivo de esta tercera parte del libro es mostrarle cómo usar el entorno de Visual Studio .NET, del cual forma parte Visual Basic .NET, para definir conexión, crear conjuntos de datos con comprobación de tipos, generar adaptadores de datos, acceder a la información desde el propio entorno, etc. Será el tema de éste y los próximos tres capítulos.

Las posibilidades o capacidades que encontraremos en el entorno, relativas al trabajo con datos, serán unas u otras dependiendo de la edición con la que contemos. Ése será el aspecto que trataremos en este capítulo, de tal forma que sepa cuáles de los elementos explicados en los capítulos siguientes tiene o no a su disposición.

Ediciones de Visual Studio .NET

Como probablemente sabrá, Visual Basic .NET es un producto que puede adquirirse por separado en su propia caja, y en cualquier comercio especializado, o

bien como parte de una suscripción a alguna de las ediciones de Visual Studio .NET, que es lo más habitual. A pesar de corresponder todos los productos a una misma versión, la primera de Visual Studio .NET presentada en febrero de 2002, existen varias ediciones distintas. Todas ellas comparten un mismo entorno, la plataforma .NET y la misma biblioteca de clases, pero difieren en ciertas características como los servidores integrados en el paquete o la funcionalidad de ciertos elementos de diseño. Visual Basic .NET, como lenguaje, podemos encontrarlo en una de las ediciones siguientes:

- *Visual Basic .NET Standard*: Es la edición más simple y única que puede adquirirse sin una suscripción de servicio. Incluye el entorno de Visual Studio .NET pero sólo con los diseñadores y elementos de Visual Basic .NET.
- *Visual Studio .NET Professional*: Aparte de Visual Basic .NET, también incluye los lenguajes Visual C# .NET, Visual C++ .NET y Visual J# .NET, así como capacidades inexistentes en la edición anterior: Crystal Reports, herramientas visuales de bases de datos, herramientas de diseño para el servidor que facilitan el acceso a servicios, bases de datos, etc. Puede obtenerse una versión de prueba de esta edición, limitada a 60 días, solicitándola en la Web de Microsoft.
- *Visual Studio .NET Enterprise Developer*: Seguramente la edición más apropiada para el desarrollo de aplicaciones con necesidades de acceso a bases de datos, ya que incorpora todas las herramientas de diseño necesarias y algunos servidores fundamentales, como SQL Server, donde la anterior sólo facilita MSDE. También se diferencia de la anterior en la incorporación de Visual SourceSafe, Visual Studio Analyzer y Application Center Test.
- *Visual Studio .NET Enterprise Architect*: Es la edición superior del producto, en la que, aparte de todo lo indicado en la anterior, encontraremos una versión específica de Visio para el modelado de bases de datos y aplicaciones y el producto Microsoft BizTalk Server para la construcción de procesos de integración con aplicaciones de clientes, proveedores o socios.

Además de éstas, que podríamos calificar de ediciones comerciales, también existe un *Visual Studio .NET Academia* dirigido a estudiantes y profesores. Sus características son similares a las de la edición *Professional*, si bien incorpora algunos elementos adicionales y específicos para la audiencia a la que se dirige.

Nota

Puede encontrar una comparativa completa entre las tres ediciones de Visual Studio .NET, así como enlaces a comparativas individuales entre Visual Studio .NET y las ediciones *Standard* de los lenguajes, en <http://msdn.microsoft.com/vstudio/howtobuy/choosing.asp>.

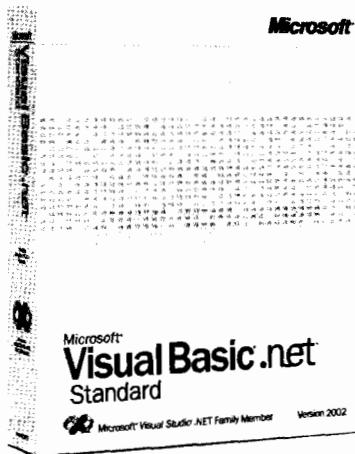


Figura 12.1. Caja de Visual Basic .NET Standard

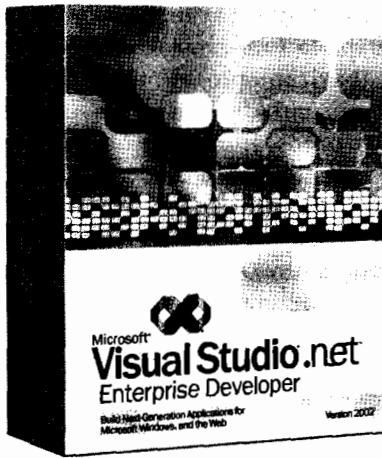


Figura 12.2. Caja de Visual Studio .NET Enterprise Developer

Posibilidades de acceso

Los mecanismos de acceso a datos de la plataforma .NET, todas las clases ADO.NET que hemos conocido en los capítulos previos, no forman parte de un lenguaje o una edición de Visual Studio .NET en particular, sino de la propia BCL que existe en cualquier instalación .NET y que, por ejemplo, encontramos ya como parte del sistema en el nuevo Windows .NET.

Las posibilidades de acceso a datos de nuestras aplicaciones, por lo tanto, son independientes de la edición de Visual Studio .NET que vayamos a emplear para

los desarrollos. Las diferencias, lógicamente, las encontraremos en el entorno de trabajo, ya que muchas de las operaciones efectuadas mediante código, o con las herramientas de administración específicas de cada origen de datos, en las ediciones superiores de Visual Studio .NET pueden realizarse desde el propio entorno.

Lo que sí resulta imprescindible es disponer de los controladores y proveedores adecuados para nuestras aplicaciones, pudiendo ser necesaria, por ejemplo, la instalación de los MDAC o los proveedores para ODBC u Oracle, en caso de que dependiéramos de ellos.

Productos integrados en el paquete

Algunas ediciones de Visual Studio .NET ofrecen, aparte del propio entorno de diseño, los compiladores y la plataforma .NET, otros productos que podríamos necesitar para el desarrollo, comprobación y depuración de nuestros proyectos. En el campo que nos interesa especialmente, el de tratamiento de datos, esos productos son MSDE, Microsoft SQL Server 2000 y Microsoft Visio.

Con la edición más baja, la *Professional*, disponemos de MSDE que, como se indicó en su momento, es, básicamente, el motor de SQL Server pero sin herramientas de administración y capacidades reducidas en cuanto a número de conexiones y tamaño de las bases de datos. El uso de MSDE, respecto a productos como Microsoft Access, es que estaremos empleando el controlador `SqlClient` y todas las posibilidades de SQL Server, aunque con MSDE. Evolucionar, cuando las necesidades lo requieran, a SQL Server será mucho más fácil.

Las ediciones *Enterprise*, tanto *Developer* como *Architect*, vienen acompañadas de varios servidores de Microsoft, entre ellos SQL Server 2000. La licencia del producto nos permite utilizarlo durante el desarrollo y comprobación, no en explotación. En cualquier caso, es una buena posibilidad ya que podemos operar sobre uno de los RDBMS más potentes y populares.

Finalmente, la edición *Enterprise Architect* también se entrega junto a la edición homónima de Microsoft Visio, un producto que nos permitirá usar técnicas de modelado basadas en estándares para construir nuestros modelos de bases de datos y negocio.

Posibilidades de diseño

Para facilitar el trabajo con bases de datos, en el sentido más amplio de la palabra, Visual Studio .NET integra una serie de herramientas que son conocidas genéricamente como *Visual Database Tools*. La mayor parte de ellas son accesibles desde el **Explorador de servidores** que, como puede verse en la figura 12.3, nos permite definir conexiones con orígenes de datos y acceder directamente a éstos, así como conectar con servidores para ver qué servicios tienen disponibles.

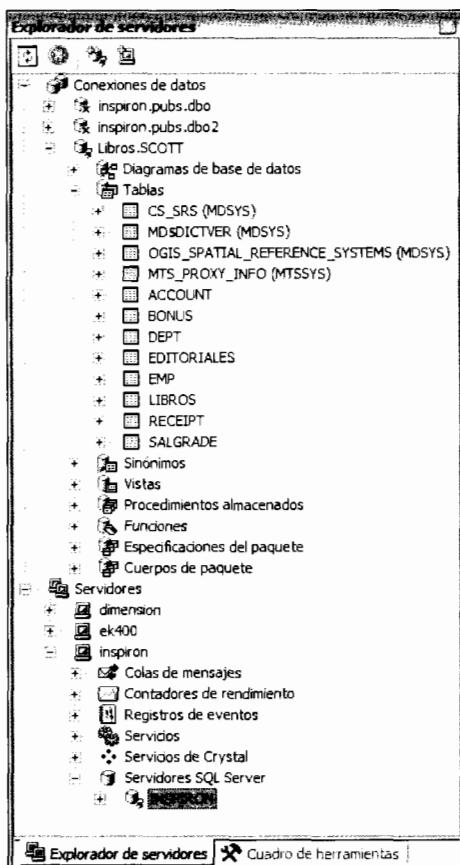


Figura 12.3. Aspecto del Explorador de servidores

Dependiendo de la edición de Visual Studio .NET con que contemos, las capacidades de estas herramientas visuales serán unas u otras. Resumidamente podríamos decir que:

- En las ediciones *Standard* podemos explorar las conexiones para ver el contenido de las tablas de bases de datos Access y MSDE.
- La edición *Professional* también nos permite explorar servidores SQL Server y cualquier origen con un controlador ODCB u OLE DB, así como diseñar tablas y vistas sobre MSDE.
- Las dos ediciones *Enterprise* permiten explorar cualquier servidor para el que exista un controlador OLE DB u ODBC, así como SQL Server de forma nativa. También facilitan el diseño de tablas, procedimientos almacenados, vistas, desencadenadores y, en general, cualquier elemento que pueda existir en un RDBMS como SQL Server u Oracle.

Nota

Los proveedores ADO.NET nativos para Oracle y ODBC aparecieron después de que Visual Studio .NET estuviese disponible, por ello desde el Explorador de servidores, en las ediciones superiores, puede accederse nativamente a un servidor SQL Server para operar sobre él, o a cualquier origen para el que exista un proveedor OLE DB, pero no puede hacerse lo mismo con Oracle. Nada nos impide, sin embargo, acceder a una base de datos Oracle usando el proveedor OLE DB correspondiente, aunque en el código después utilicemos el proveedor OracleClient.

Una visión general

Asumiendo que estamos trabajando con una de las ediciones *Enterprise* de Visual Studio .NET, en los capítulos siguientes se abordará el uso de las herramientas visuales de datos y muchos de los componentes que pueden vincularse con un DataSet, DataView o similar para facilitar la presentación y manipulación por parte del usuario final. Los puntos siguientes le ofrecen una visión general de los temas que podrá encontrar en dichos capítulos:

- Uso del Explorador de servidores para definir una conexión y acceder a los elementos de un origen de datos.
- Diseño, desde el Explorador de servidores, de nuevos elementos tales como tablas, vistas y procedimientos almacenados, observando el resultado desde el propio entorno de Visual Studio .NET sin necesidad de recurrir a la herramienta de administración propia de cada producto.
- Generación automática de conexiones a origen de datos, adaptadores de datos y objetos DataSet con comprobación estricta de tipos.
- Uso de los diversos asistentes de Visual Studio .NET para la generación semi-automática de comandos SQL para la creación y modificación de consultas, vistas, procedimientos almacenados, etc.
- Vinculación de conjuntos de datos y vistas con componentes de interfaz Windows y Web.
- Diseño de formularios de visualización y edición de datos tanto para interfaces basadas en Windows como para clientes Web, introduciendo el uso del Asistente para formularios de datos.

A medida que vaya conociendo todos los elementos, verá cómo una parte importante del código que en los ejemplos de capítulos previos introducimos manualmente, por ejemplo para definir una conexión, un adaptador de datos o un conjunto

de datos, ahora se genera de forma automática, de tal forma que tan sólo tenemos que usar los componentes creados mediante operaciones de arrastrar y soltar.

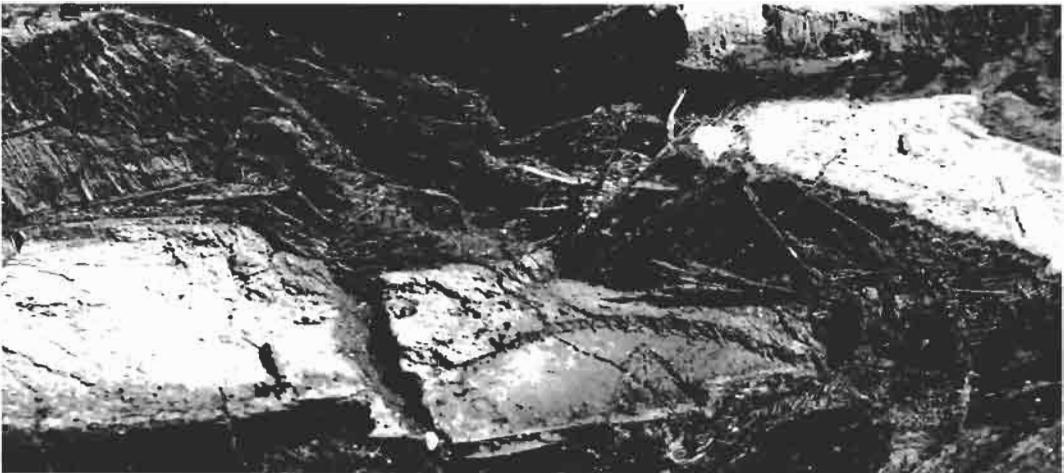
Nota

Ha de tener en cuenta que, en caso de que utilice la edición *Professional* de Visual Studio .NET, o *Visual Basic .NET Standard*, algunos de los procedimientos en los capítulos siguientes pueden no estar disponibles en su instalación del producto.

Resumen

Con este breve capítulo se han situado, en cuanto a características de acceso a datos se refiere, las diferentes capacidades de las ediciones existentes de Visual Studio .NET. Ahora ya sabe lo que podrá hacer, o no, según la edición que tenga instalada en su sistema.

También se han avanzado, en una visión rápida, algunos de los temas que se van a tratar en los capítulos de esta tercera parte del libro, a partir del capítulo siguiente.



13

Herramientas visuales de datos

En los ejemplos desarrollados en los capítulos de la segunda parte hemos conectado con diferentes orígenes de datos, recuperado información mediante objetos `DataReader`, ejecutado comandos, llenado `DataSet` con información de esos orígenes a través de adaptadores de datos, generado automáticamente los comandos de actualización, etc. Todas esas acciones las hemos implementado escribiendo código, lo cual nos ha servido para conocer muchos de los detalles de las clases implicadas, sabiendo cómo emplearlas sin necesidad de asistentes ni un entorno de diseño.

No obstante, ese entorno de diseño existe en Visual Studio .NET, y podemos aprovecharlo con el fin de ahorrar una cantidad importante de trabajo, produciendo automáticamente, mediante operaciones de arrastrar y soltar, los componentes que son necesarios para comunicarse con un origen de datos, extraer y devolver información.

El objetivo de este capítulo es mostrarle cómo puede servirse de esas herramientas de diseño para efectuar tareas como la edición directa de datos, definición de estructuras y otros elementos, creación automática de componentes de conexión, adaptadores y conjuntos de datos, etc.

Tal y como podrá observar, el contenido de este capítulo es mucho menos teórico que el de los capítulos de la segunda parte y va a indicarle directamente cómo efectuar cada tarea, por lo que es casi imprescindible que se encuentre delante de su ordenador con Visual Studio .NET abierto para poder ir siguiendo las explicaciones.

El Explorador de servidores

En versiones previas de Visual Basic, así como en otras herramientas de desarrollo, existe una ventana, conocida como **Cuadro de herramientas** o **Paleta de componentes**, desde la cual es posible tomar componentes y arrastrarlos hasta una superficie de diseño. Esa acción permite aprovechar módulos de código prefabricados, los componentes, de una manera muy simple, personalizándolos mediante propiedades y controlándolos, desde código, a través de métodos y eventos.

El **Explorador de servidores** de Visual Studio .NET es un elemento similar, si bien los objetos que encontramos en su interior no son componentes sino servicios y otras entidades de servidor tales como colas de mensajes, registros de eventos o servidores de datos. Lo interesante es que todos esos objetos pueden manipularse directamente desde el propio entorno de Visual Studio .NET, gracias al **Explorador de servidores**, y que las operaciones de arrastrar y soltar generan clases a medida según el tipo de elemento de que se trate. Si, por ejemplo, toma un servicio cualquiera y lo arrastra hasta un formulario, Visual Studio .NET creará una clase específica para poder controlarlo desde su aplicación. Si el objeto arrastrado es una tabla de un origen de datos, entonces se generarán las clases necesarias para conectar con ese origen, recoger los datos y crear un conjunto accesible desde el programa.

En la configuración por defecto, el **Explorador de servidores** aparece como una ventana con ocultación automática en el margen izquierdo del entorno, y su aspecto inicial no diferirá demasiado del mostrado en la figura 13.1. Tal y como puede apreciarse, existen dos nodos principales: **Conexiones de datos** y **Servidores**. En el primero aparecerán las conexiones de datos que tengamos definidas, ninguna en principio, mientras que en el segundo encontramos los servidores a los que tenemos acceso. Por defecto aparecerá el nombre del ordenador en el que está ejecutándose Visual Studio .NET.

Para poder trabajar sobre una conexión, o un servidor, primero tenemos que añadirlos al **Explorador de servidores**, por ello serán las dos primeras tareas de las que nos ocuparemos.

Definir una nueva conexión

El nodo **Conexiones de datos** está pensado para alojar las conexiones a orígenes de datos para las que tengamos que usar el proveedor OLE DB, existiendo una alternativa en caso de que dicho origen sea SQL Server que emplea directamente el proveedor nativo `SqlClient`. Si el origen de datos es cualquier otro, Access, Oracle, etc., tendremos que definir una conexión mediante OLE DB ya que, por ahora, los proveedores nativos para otros RDBMS, como `OracleClient`, no se encuentran incorporados en el entorno de diseño.

Haga clic con el botón secundario del ratón sobre el nodo **Conexiones de datos**, para desplegar su menú emergente, y luego seleccione la opción **Agregar conexión** (véase figura 13.2). De inmediato aparecerá el cuadro de diálogo **Propiedades de**

vínculo de datos, mostrado en la figura 13.3, que ya conoce por haberlo usado en capítulos previos, concretamente al tratar los archivos UDL. Seleccione el proveedor OLE DB que desea utilizar e introduzca los datos necesarios para identificar el origen de datos y facilitar cualquier parámetro adicional que pudiera necesitarse.

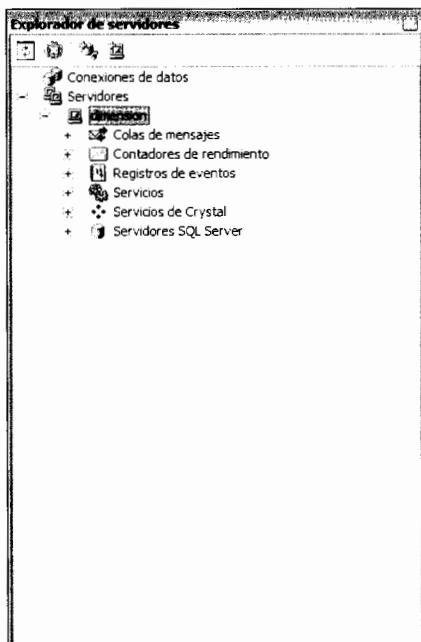


Figura 13.1. Aspecto inicial del Explorador de servidores

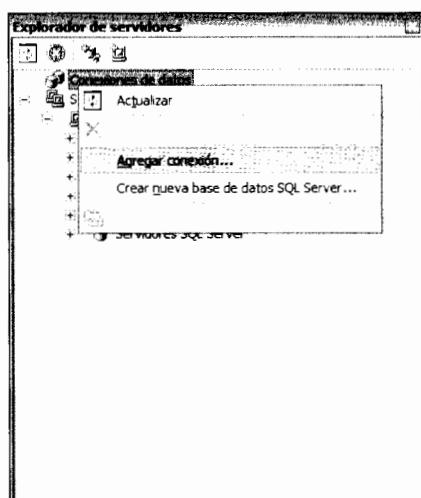


Figura 13.2. Añadimos una nueva conexión al Explorador de servidores

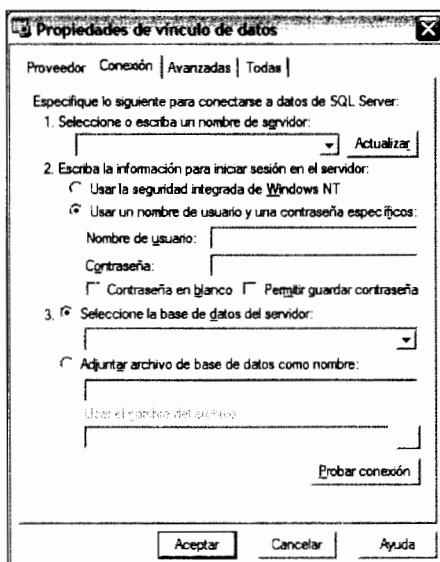


Figura 13.3. Ventana Propiedades de vínculo de datos

Seleccione de la página Proveedores el proveedor Oracle Provider for OLE DB, introduciendo en la página Conexión el nombre del servicio local Oracle que creó con el software cliente, así como el nombre de usuario y la clave. En los ejemplos de capítulos previos el servicio era Libros, el usuario scott y la clave tiger. Pulse el botón **Probar conexión** para asegurarse de que los parámetros son correctos y, finalmente, pulse el botón **Aceptar** para finalizar la definición de la conexión.

Nota

Tras crear una conexión con un RDBMS es posible que, al desplegarla en el Explorador de soluciones, aparece una ventana solicitándole la identificación. Introduzca el mismo nombre de usuario y clave que usó al definir la conexión.

El nombre con el que aparecerá la nueva conexión, en el Explorador de servidores, será una combinación del identificador de servicio y el nombre del usuario, como se aprecia en la figura 13.4. Al desplegarla ya tenemos acceso a todo el contenido de la base de datos desde el propio entorno de Visual Studio .NET, sin necesidad de usar el software cliente de Oracle.

Nota

Usando el menú emergente asociado a una conexión, tras crearla, podrá tanto modificar sus parámetros como eliminarla, en caso de que ya no le sirva.

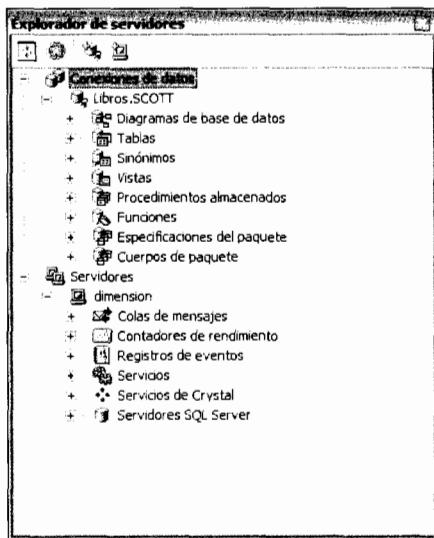


Figura 13.4. Al desplegar la conexión aparece una lista de carpetas con objetos existentes en el origen de datos

Registrar un nuevo servidor

Si el origen de datos sobre el que vamos a trabajar es SQL Server, no necesitamos definir una conexión para acceder a él. Los servidores SQL Server aparecen como elementos en el nodo **Servidores SQL Server** del equipo donde se encuentren. Si tenemos nuestro servidor ejecutándose en el mismo equipo donde estamos usando Visual Studio .NET, no tenemos más que desplegar dicho nodo para acceder a las bases de datos. En caso contrario, si SQL Server se encuentra en una máquina distinta, tendremos que registrarla como un nuevo servidor.

El proceso es realmente sencillo: haga clic con el botón secundario del ratón sobre el elemento **Servidores** del Explorador de servidores y elija la opción **Agregar servidor**. En la ventana que aparece, similar a la mostrada en la figura 13.5, simplemente introduzca el nombre de la máquina o, si lo prefiere y la conoce, directamente su dirección IP. A partir de ese momento, siempre que en dicha otra máquina tenga las credenciales apropiadas, podrá operar sobre el nuevo servidor de forma remota, aunque se encuentre a miles de kilómetros.

En mi configuración particular, Visual Studio .NET se ejecuta en un ordenador llamado Dimension, mientras que SQL Server lo hace en otro llamado Inspiron. Por ello, para usar desde Visual Studio .NET las posibilidades de diseño que me ofrece el Explorador de servidores, he de registrar Inspiron como servidor adicional. Hecho esto tengo un acceso completo al servidor SQL Server que se encuentra en esa otra máquina, según puede verse en la figura 13.6. En ella se aprecia cómo se ha conectado con el servidor y desplegado la base de datos Libros, en la que aparecen varias carpetas.

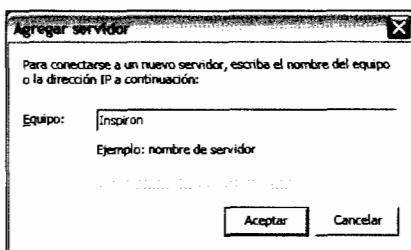


Figura 13.5. Introducimos el nombre del servidor a registrar

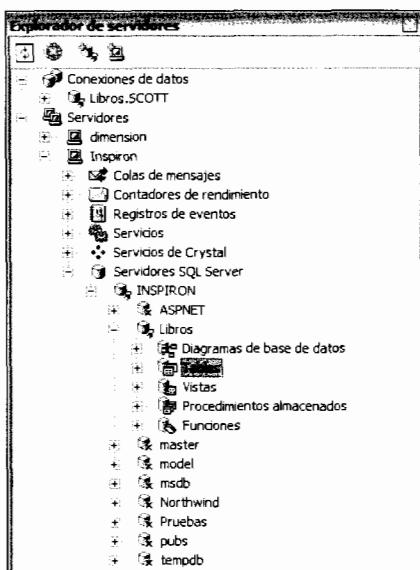


Figura 13.6. Acceso desde Visual Studio .NET a un servidor SQL Server que se ejecuta en otra máquina

Nota

En caso de que no aparezca en el nodo Servidores SQL Server la instancia del RDBMS que debería aparecer, puede utilizar la opción Registrar instancia de SQL Server, facilitando el nombre del servicio. Esto, normalmente, no suele ser necesario a menos que tenga varias instancias de SQL Server 2000, cada una con su nombre de servicio, ejecutándose en la misma máquina.

Apertura y cierre de conexiones

Para acceder desde el Explorador de servidores al contenido de un origen de datos, ya sea SQL Server o cualquier otro para el que hayamos definido una co-

nexión mediante OLE DB, es necesario establecer una conexión. Ésta se abre, normalmente, de forma automática cuando es necesario, por ejemplo al ir a acceder a las tablas de una base de datos. Dependiendo de las operaciones que efectuemos, esa conexión se mantendrá abierta durante más o menos tiempo.

¿Cómo saber si una cierta conexión está o no abierta? Tan sólo tiene que fijarse en el ícono que precede al nombre de la conexión, o la base de datos en caso de SQL Server. Si dicho ícono tiene en la parte inferior derecha un pequeño enchufe es que hay abierta una conexión. En caso de que lo que aparezca sea una cruz en rojo, no hay conexión activa. En la figura 13.7 puede observar, ampliado, un detalle de dicho ícono, con una conexión activa, a la izquierda, y cerrada, a la derecha.

Para cerrar una conexión explícitamente, cuando le interese, no tiene más que abrir el menú emergente de la conexión y seleccionar la opción Cerrar conexión.

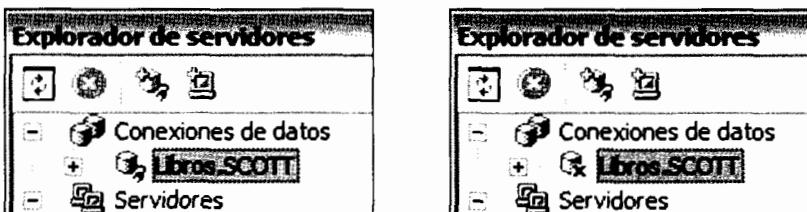


Figura 13.7. La misma conexión en los dos estados, abierta, a la izquierda, y cerrada, a la derecha

Creación de nuevas bases de datos

La integración de Visual Studio .NET con SQL Server es superior a la que tiene con el resto de orígenes de datos, existiendo opciones específicas para este RDBMS que nos están disponibles para los demás. Una de esas opciones es la que nos permite crear una nueva base de datos desde el propio Visual Studio .NET, sin necesidad de recurrir al Administrador corporativo. Puede hacerse de dos maneras:

1. Seleccionando la opción **Crear nueva base de datos SQL Server** del menú emergente asociado al elemento **Conexiones de datos**, en la ventana Explorador de servidores.
2. Seleccionando la opción **Nueva base de datos** del menú emergente asociado a una instancia de servidor de SQL Server, como se aprecia en la figura 13.8.

En ambos casos aparecerá el mismo cuadro de diálogo, si bien con la primera opción tendríamos que indicar el servidor donde va a crearse la base de datos y con la segunda no es necesaria, puesto que hemos empleado el menú emergente del servidor sobre el que recaerá la acción. Además tendremos que introducir el nombre de la nueva base de datos, que será creada y quedará como una nueva conexión disponible, si empleamos la primera alternativa, o simplemente como otra base de datos en el servidor, de haber usado la segunda.

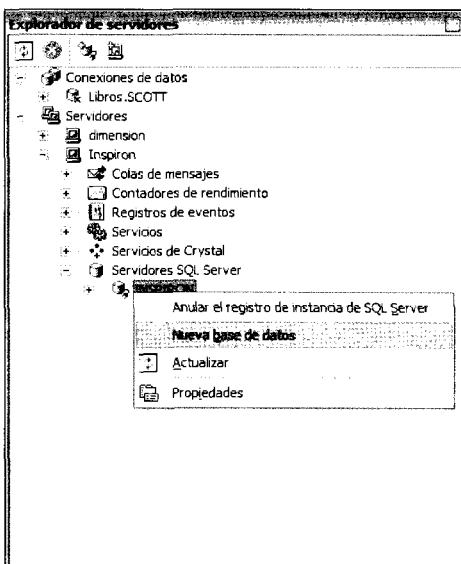


Figura 13.8. Opción para crear una nueva base de datos SQL Server

A partir de la creación, la nueva base de datos queda en el **Explorador de servidores** como una nueva conexión o una nueva base de datos, según el caso, tratándose desde ese momento como cualquier otro de los elementos que ya existiesen con anterioridad. Los procesos descritos en los puntos previos, por tanto, son aplicables de manera indistinta.

Edición de datos

Durante el desarrollo de una aplicación es habitual la necesidad de editar directamente los datos alojados en el origen, ya sea para preparar un bloque de datos de prueba, simular posibles fallos o corregir los que pudieran generarse. En cualquier caso, casi siempre se recurre a una herramienta específica del origen para efectuar ese trabajo. Así lo hicimos en el tercer capítulo, al emplear Microsoft Access para introducir información en una base de datos Access o las herramientas de administración de SQL Server y Oracle para hacer lo propio sobre esos RDBMS.

En realidad, una vez definida la estructura de la base de datos, no necesitábamos emplear ninguna de esas herramientas para editar la información puesto que, como va a ver de inmediato, es algo que puede hacerse directamente desde el entorno de Visual Studio .NET. En realidad, tampoco para definir la estructura precisamos más que este entorno si tenemos una de las ediciones superiores de Visual Studio .NET.

Si tiene acceso a un servidor SQL Server, registre el equipo en la rama **Servidores** del **Explorador de servidores**, despliegue el servidor de datos y seleccione, de

la carpeta **Tablas** de la base de datos que prefiera, una tabla cualquiera, haciendo doble clic sobre ella. Si el origen de datos no es SQL Server primero tendrá que definir una conexión, según los pasos explicados antes, tras lo cual el proceso sería idéntico.

Tras hacer doble clic, y siempre que la conexión con el origen sea satisfactoria, verá aparecer en el entorno de Visual Studio .NET una cuadrícula con los datos actuales de la tabla, siendo posible la eliminación, modificación e inserción de datos. En la figura 13.9 puede ver cómo se editan dos tablas de manera simultánea. La de la parte superior corresponde a una base de datos de ejemplo instalada con Oracle 8i, mientras que la inferior es la tabla **Libros** que creamos con Microsoft Excel en los ejemplos del tercer capítulo. En ambos casos se ha definido una conexión mediante el proveedor OLE DB.

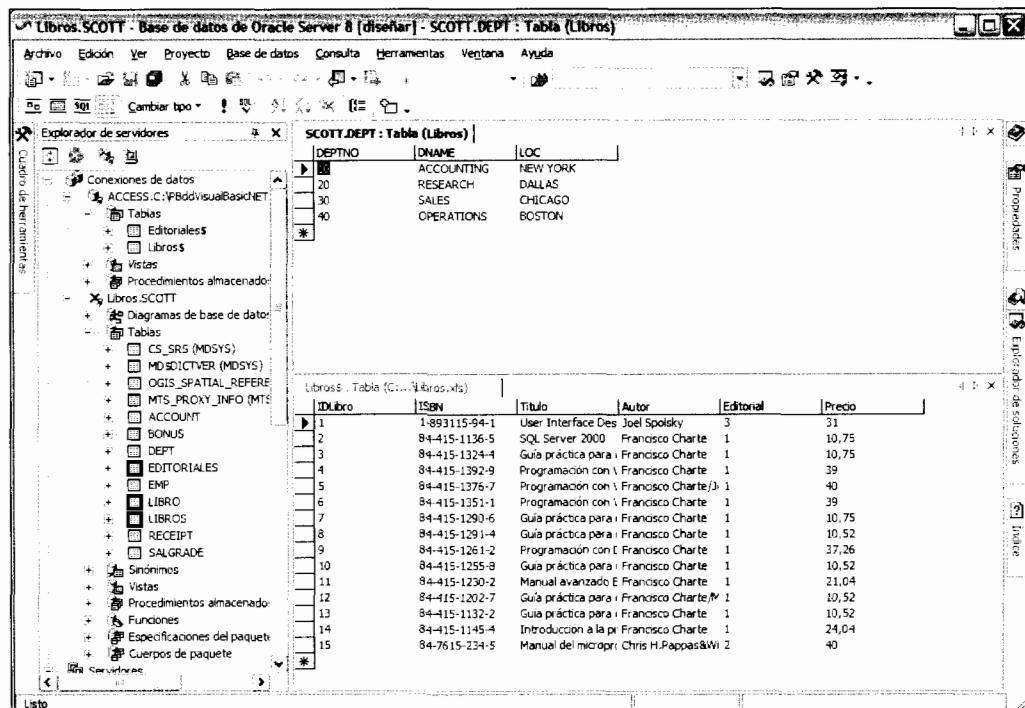


Figura 13.9. Edición de tablas de datos en el entorno de Visual Studio .NET

Nota

Observe que siempre que defina una conexión utilizando el proveedor Microsoft JET, por ejemplo para acceder al libro de Excel en este caso, en el Explorador de servidores siempre aparece el prefijo ACCESS. Si se fija, no obstante, verá que el nombre del archivo es **Libros.xls** y no **Libros.mdb**.

Navegar por las filas

Al abrir una tabla con el sistema anterior, haciendo doble clic sobre ella, obtendremos todo el contenido, lo cual puede significar tan sólo unas decenas de filas, como ocurre con la base de datos creada a modo de ejemplo para este libro, o bien miles de ellas, en caso de que conectemos con una base de datos en explotación real u obtenida como copia de una que se encuentre en explotación.

Usando las habituales teclas de desplazamiento del cursor, o bien el puntero del ratón para actuar sobre la barra de desplazamiento, nos podremos desplazar de un punto a otro fila a fila. También pueden emplearse ciertas combinaciones, como **Control-Inicio** y **Control-Fín**, para ir rápidamente a la primera de las filas o a la última, respectivamente.

Para insertar una nueva fila no tenemos más que desplazarnos debajo de la última que tiene datos, apareciendo automáticamente una fila vacía en la que podemos introducir los nuevos datos. De forma similar, podemos modificar el contenido de cualquiera de las filas mostradas. Si deseamos eliminar una de ellas tendremos que seleccionarla completa, haciendo clic en el selector que aparece a la izquierda de la primera columna, y pulsar a continuación la tecla **Supr**.

Nota

Debe tener en cuenta que ciertas columnas de datos pueden no ser editables. No podrá, por ejemplo, modificar la columna **IDLibro** de la tabla **Libros** de la base de datos SQL Server, puesto que su valor se genera automáticamente. Si añade una nueva fila, el valor de dicha columna será insertado automáticamente. Tampoco podrá editar aquellas columnas cuya precisión excede unos ciertos límites.

En lugar de efectuar todas estas operaciones mediante teclas o combinaciones de teclas, también podemos abrir el menú emergente (véase figura 13.10) y seleccionar la acción que deseamos ejecutar. La opción **Fila** nos permite ir directamente a una cierta fila introduciendo su número. Las tres anteriores van a la primera fila, a la última e insertan una nueva fila, respectivamente. Las que hay detrás facilitan la copia y el pegado de datos o filas completas, por ejemplo para llenar rápidamente una tabla.

Selección de datos

Al abrir una tabla, como hemos hecho en el punto previo, no sólo aparece una cuadrícula con las filas y columnas de datos sino que, además, se activa una paleta de herramientas que por defecto no está visible. En ella, según puede verse en la figura 13.11, existen múltiples botones con los que podemos ocultar y mostrar distintos paneles, ejecutar una sentencia SQL, añadir otras tablas, agrupar datos, etc.

Simplemente sitúe el puntero del ratón sobre cada botón para obtener una indicación de su finalidad.

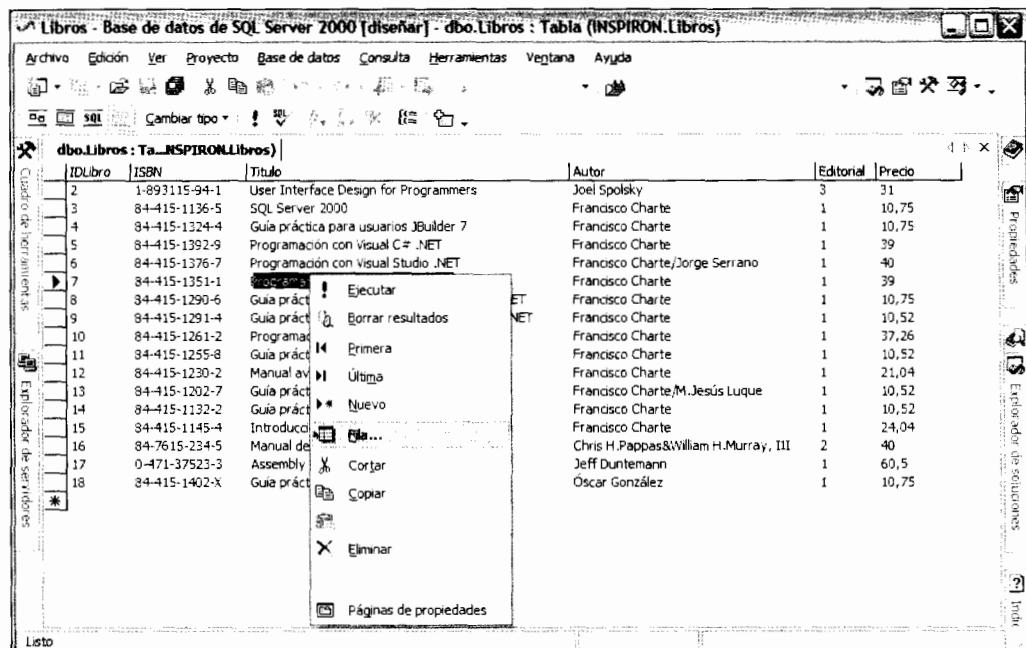


Figura 13.10. Menú en el que encontramos las operaciones más habituales



Figura 13.11. Paleta de botones activa durante la edición de datos

Tras haber hecho doble clic en la tabla `Libros`, para abrir la correspondiente cuadricula, pulse el botón **Agregar tabla** (último de la paleta indicada antes) y seleccione de la ventana que aparece (véase figura 13.12) la tabla `Editoriales`, pulsando el botón **Agregar**, primero, y el botón **Cerrar**, a continuación.

En principio lo único que notará es que los datos de la tabla `Libros`, que ya estaban visibles, aparecen en gris. Pulse el botón **Ejecutar consulta**, cuyo ícono es un signo de admiración. Verá que la tabla de datos mostrada cambia, apareciendo en cada fila, aparte de la información del libro en cuestión, todos los datos de la editorial correspondiente.

Fíjese en la figura 13.13 que la editorial mostrada a la derecha es la correspondiente al código de editorial de cada libro, a pesar de que no hayamos indicado explícitamente esa vinculación entre ambas tablas. Su estructura, sin embargo, indica que `Libros.Editorial` es una clave externa que apunta a `Editoriales.IDEditorial`, así que Visual Studio .NET no necesita saber más.

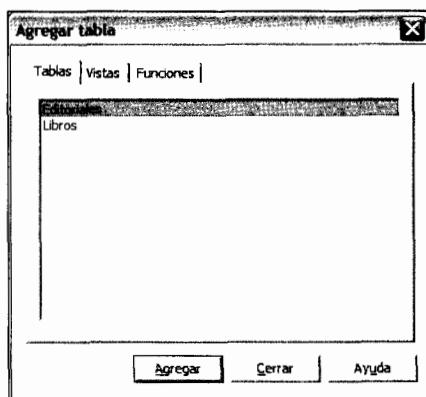


Figura 13.12. Agregamos la tabla Editoriales

Libros - Base de datos de SQL Server 2000 [diseñar] - dbo.Libros - Tabla (IMPIRON.Libros)*										
	IdLibro	ISBN	Título	Autor	Editorial	Precio	IDEditorial	Nombre	Dirección	
1	1-893115-94-1		User Interface Design for Programmers	Joel Spolsky	3	31	3	Apress	901 Grayson Street	
2	84-415-1136-5		SQL Server 2000	Francisco Charte	1	10,75	1	Anaya Multimedia	Telemaco, 43	
3	94-415-1242-4		Guía práctica para usuarios de JBuilder 7	Francisco Charte	1	10,75	1	Anaya Multimedia	Telemaco, 43	
4	84-415-1392-9		Programación con Visual C# .NET	Francisco Charte	1	39	1	Anaya Multimedia	Telemaco, 43	
5	84-415-1276-7		Programación con Visual Studio .NET	Francisco Charte/Jorge Serrano	1	40	1	Anaya Multimedia	Telemaco, 43	
6	84-415-1351-1		Programación con Visual Basic .NET	Francisco Charte	1	39	1	Anaya Multimedia	Telemaco, 43	
7	84-415-1290-6		Guía práctica para usuarios de Visual Basic .NET	Francisco Charte	1	10,75	1	Anaya Multimedia	Telemaco, 43	
8	84-415-1291-1		Guía práctica para usuarios de Visual Studio .NET	Francisco Charte	1	10,52	1	Anaya Multimedia	Telemaco, 43	
9	84-415-1261-2		Programación con Delphi 5 y Kylix	Francisco Charte	1	37,26	1	Anaya Multimedia	Telemaco, 43	
10	84-415-1255-6		Guía práctica para usuarios de Delphi 6	Francisco Charte	1	32	1	Anaya Multimedia	Telemaco, 43	
11	84-415-1207-2		Manual avanzado Excel 2002	Francisco Charte	1	21,04	1	Anaya Multimedia	Telemaco, 43	
12	84-415-1207-7		Guía práctica para usuarios de Excel 2002	Francisco Charte/M. Jesús Luque	1	10,52	1	Anaya Multimedia	Telemaco, 43	
13	84-415-1137-3		Guía práctica para usuarios de Kixx	Francisco Charte	1	10,52	1	Anaya Multimedia	Telemaco, 43	
14	94-415-1145-4		Introducción a la programación	Francisco Charte	1	24,04	1	Anaya Multimedia	Telemaco, 43	
15	84-7615-234-5		Manual del microprocesador 80386	Chris H. Papageorgiou/William H. Murray, III	2	40	2	McGraw-Hill	Edificio Valdeorras, 1ª planta	
16	94-371-3752-3		Assembly Language Step-by-Step	Jeff Duntemann	1	60,5	1	Anaya Multimedia	Telemaco, 43	

Figura 13.13. Resultado obtenido tras añadir la tabla Editoriales y ejecutar de nuevo la consulta

Si no pretendemos editar los datos, sino simplemente efectuar alguna comprobación, posiblemente no necesitemos tener visibles todas las columnas. Puede pulsar el botón **Mostrar panel de diagrama**, el primero contando desde la izquierda, para abrir un esquema visual en el que se aprecia la relación entre ambas tablas y, además, puede marcar y desmarcar las columnas que desea tener visibles. Una alternativa es pulsar el botón **Mostrar panel SQL** y editar directamente la sentencia SQL que se desea ejecutar. Para ver el resultado, como en el caso anterior, tendrá

que pulsar el botón **Ejecutar consulta**. En la figura 13.14 puede ver el **Panel de diagrama**, en la parte superior; el **Panel SQL**, en el área central, y el **Panel de resultados** en la sección inferior. Lógicamente, puede añadir a la consulta SQL cláusulas de selección de filas, ordenación, agrupamiento, etc.

Otra forma de establecer criterios de selección y ordenación consiste en abrir el **Panel de cuadrícula** (segundo de los botones de la barra) e introducir directamente las opciones deseadas. En la figura 13.15 se ve cómo se ha asignado a la columna **Nombre** de la tabla **Editoriales** el alias **Editorial**, título que aparece en el panel de resultados. Además se ha ordenado ascendente por el título del libro, aparte de seleccionarse sólo aquellos cuyo precio está por debajo de los 40 euros.

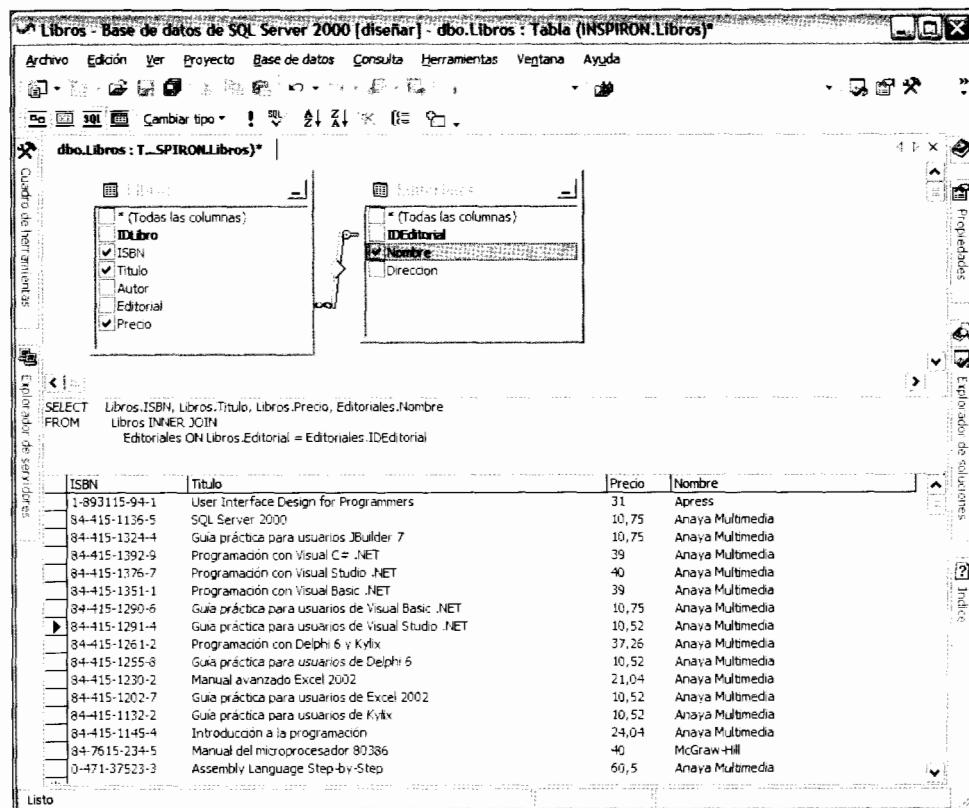


Figura 13.14. Selección de columnas mediante el Panel de diagrama y el Panel SQL

Nota

Si mantiene abierto el **Panel SQL**, a medida que haga cambios en el **Panel de diagrama** y **Panel de cuadrícula** podrá ir viendo cómo se reestructura la sentencia SQL que ejecutará al pulsar el botón **Ejecutar consulta**.

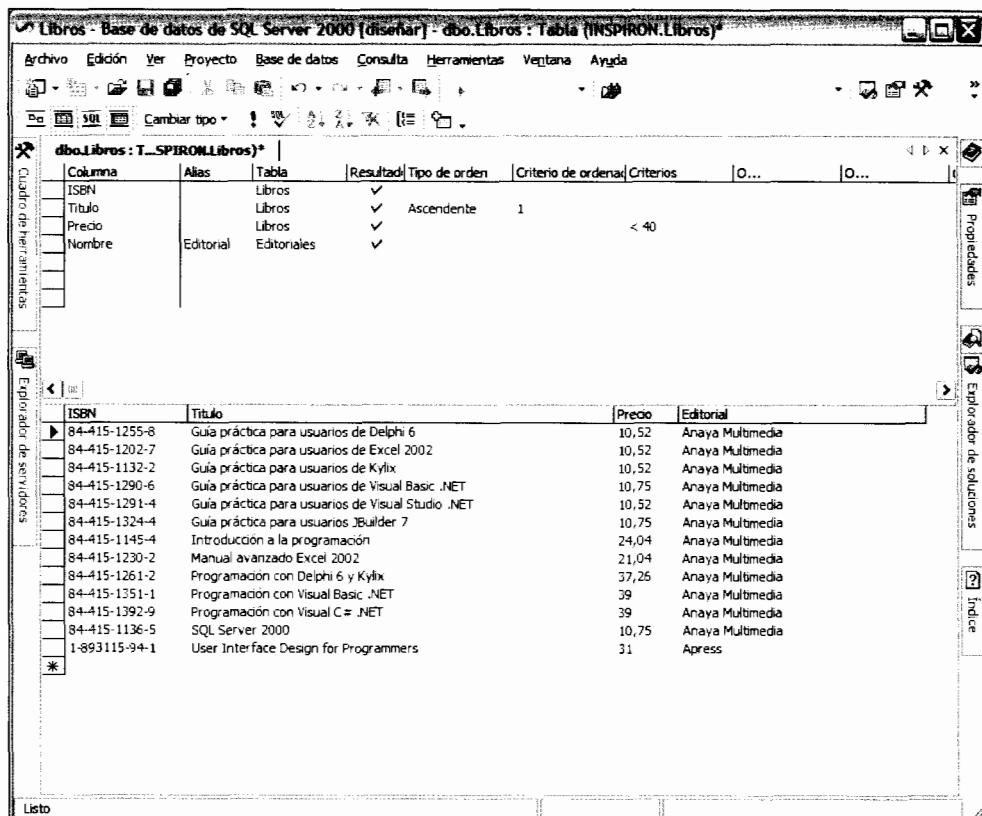


Figura 13.15. Establecemos filtros de selección de datos y criterios de ordenación

Modificación, inserción y eliminación

Al abrir una tabla desde Visual Studio .NET, haciendo doble clic sobre ella en el Explorador de servidores, se genera automáticamente una sentencia SQL de consulta para recuperar todas las filas y columnas de la tabla. Con los elementos indicados en el punto previo, los distintos paneles inicialmente ocultos, es posible seleccionar columnas y filas. La sentencia, no obstante, seguirá siendo de consulta.

Si tras abrir una tabla despliega la lista adjunta al botón **Cambiar tipo**, como se ha hecho en la figura 13.16, podrá cambiar la sentencia SQL para que en vez de ser de consulta sea de inserción de resultados, de valores, de actualización o de eliminación. Según la opción que elija, en el Panel SQL verá aparecer una sentencia `INSERT INTO`, `UPDATE` o `DELETE`.

A partir del momento en que se modifique el tipo de sentencia, las acciones que llevemos a cabo en los paneles de diagrama y cuadrícula, eligiendo columnas o estableciendo criterios de selección de filas, se aplicarán a la nueva sentencia SQL. Puede hacer pruebas manteniendo abierto el Panel SQL para ver las sentencias ge-

neradas, pero sin llegar a pulsar el botón **Ejecutar consulta** para evitar, por ejemplo, la eliminación de los datos.

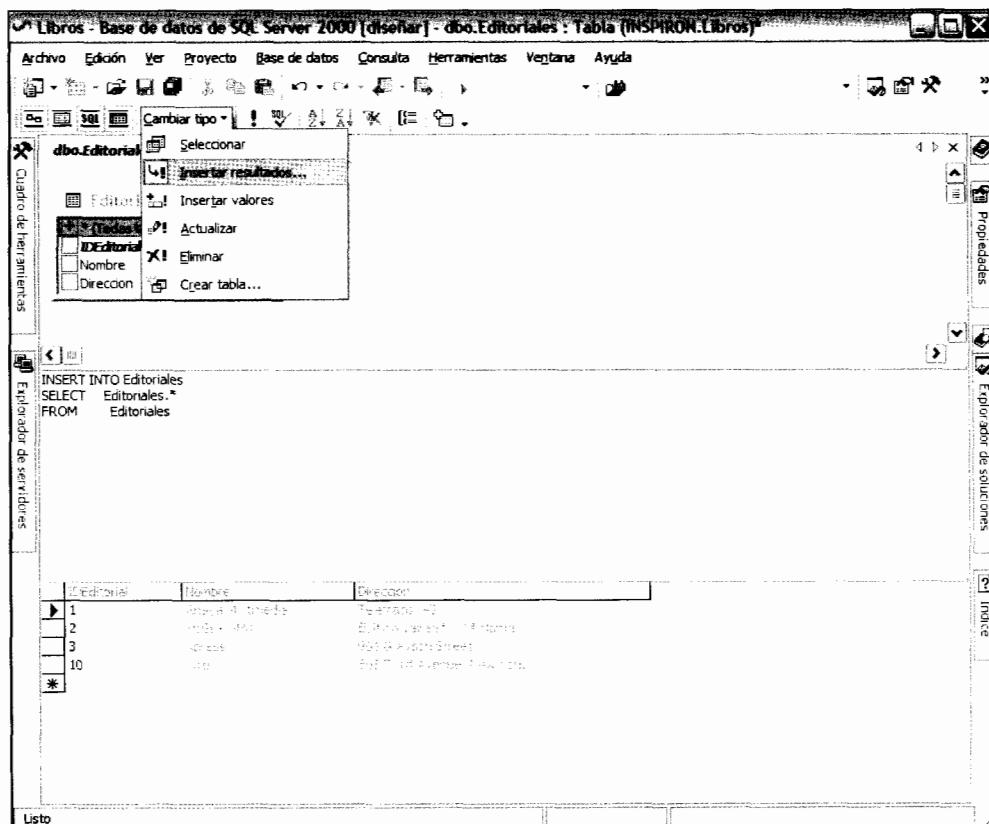


Figura 13.16. Modificamos el tipo de sentencia SQL

Agrupación de los resultados

Si tiene como parte de la consulta alguna expresión de resumen, puede agrupar las filas de datos según el valor de una cierta columna para obtener un resultado agrupado. Partiendo desde el **Explorador de servidores**, y asumiendo que aún no ha abierto ninguna tabla, puede hacer una simple prueba dando estos pasos:

- Haga doble clic sobre la tabla **Libros** para abrirla.
- Pulse el botón **Agregar tabla** y seleccione la tabla **Editoriales**.
- Abra el **Panel de diagrama** y seleccione de la tabla **Editoriales** la columna **IDEditorial**. Teniendo ésta seleccionada, pulse el botón **Orden ascendente**, estableciendo así un criterio de ordenación.

- Pulse el botón **Agrupar por**, el penúltimo de la barra, mientras mantiene la selección en la misma columna IDEditorial.
- Por último, pulse el botón **Ejecutar consulta**.

El resultado obtenido debería ser similar al de la figura 13.17. La primera columna de resultados es el número de títulos que tiene cada una de las editoriales, cuyo código aparece en la segunda columna. Puede usar el Panel de cuadrícula para asociar alias a estas columnas, así como para cambiar el orden en que aparecen en la cuadrícula de resultados simplemente arrastrando y soltando.

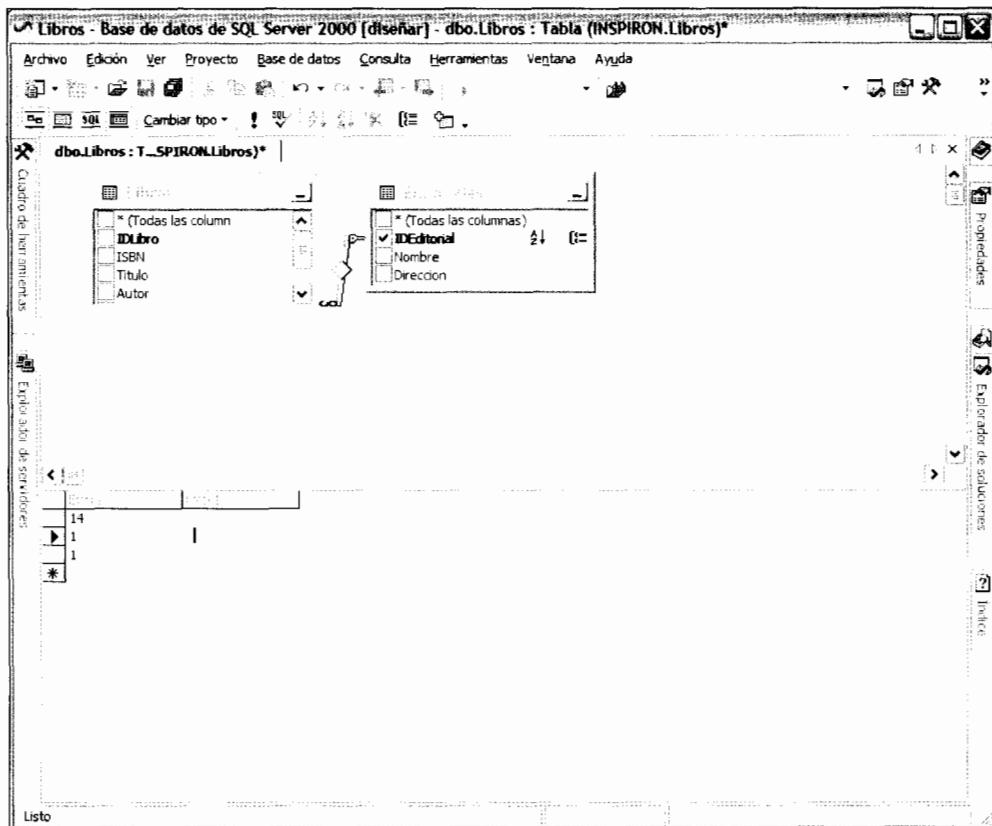


Figura 13.17. Resultado obtenido al agrupar por la columna IDEditorial

Nota

Como en los demás casos, también puede acceder a la consulta SQL y modificarla directamente, prescindiendo de los demás paneles de diseño. Si conoce el lenguaje SQL muchas veces le resultará más rápido.

Edición de información de esquema

En los puntos previos nos hemos centrado en la edición de datos, asumiendo, por supuesto, que la estructura de las bases de datos ya se encuentra establecida. Gracias a las *Visual Database Tools* de Visual Studio .NET, no obstante, también tenemos la posibilidad de editar la información de esquema si estamos usando MSDE o, en el caso de las ediciones *Enterprise*, de cualquier origen de datos tipo RDBMS, incluidos Oracle y SQL Server.

Podemos crear nuevas tablas, vistas y procedimientos almacenados, así como modificar la estructura de los elementos existentes siempre que el origen no nos lo impida, por ejemplo porque causase una pérdida de información.

Creación y modificación de tablas

Para crear una nueva tabla, abra el menú emergente de la carpeta **Tablas** de la base de datos sobre la que vaya a trabajar, eligiendo la opción **Nueva tabla**. Si lo que necesita es editar la estructura de una tabla existente, pulse el botón secundario sobre ella, en el **Explorador de servidores**, y elija la opción **Diseñar tabla**.

En cualquier caso, tanto si la tabla es nueva como si ya existía, se encontrará con una interfaz similar a la que empleó en Microsoft Access o SQL Server, en el tercer capítulo, para crear las tablas de la base de datos de ejemplo. Una cuadricula donde podrá ir introduciendo el nombre de cada columna, su tipo de dato, longitud, etc. En la figura 13.18 puede ver, a la izquierda, la edición de la estructura de la tabla *Libros* de SQL Server, mientras que a la derecha se está añadiendo una nueva tabla a la base de datos Oracle.

Nota

Inicialmente, al crear una nueva tabla, Visual Studio .NET le da un nombre provisional del tipo Tabla1. Cuando se finaliza el diseño, y se cierra la ventana, aparecerá una pequeña ventana preguntando si desea crear la tabla y solicitando un nombre definitivo para ella.

Al igual que ocurría al abrir una tabla para edición de datos, mientras diseñamos su estructura hace su aparición una paleta de botones específica. Ésta, como se aprecia en la figura 13.19, cuenta con cinco botones cuyo nombre y finalidad, de izquierda a derecha, son los siguientes:

- **Generar secuencia de comandos de cambio:** Finalizado el diseño o modificación de la tabla, al pulsar este botón se generará la secuencia de comandos SQL correspondiente. En principio dicha secuencia aparece en una ventana (véase figura 13.20) desde la que podemos guardarla en un archivo. Si lo

desea, puede activar la opción Generar automáticamente secuencia de comandos de cambio al guardar para que, sin necesidad de volver a pulsar este botón, se genere automáticamente esa secuencia de comandos cuando haya finalizado el diseño.

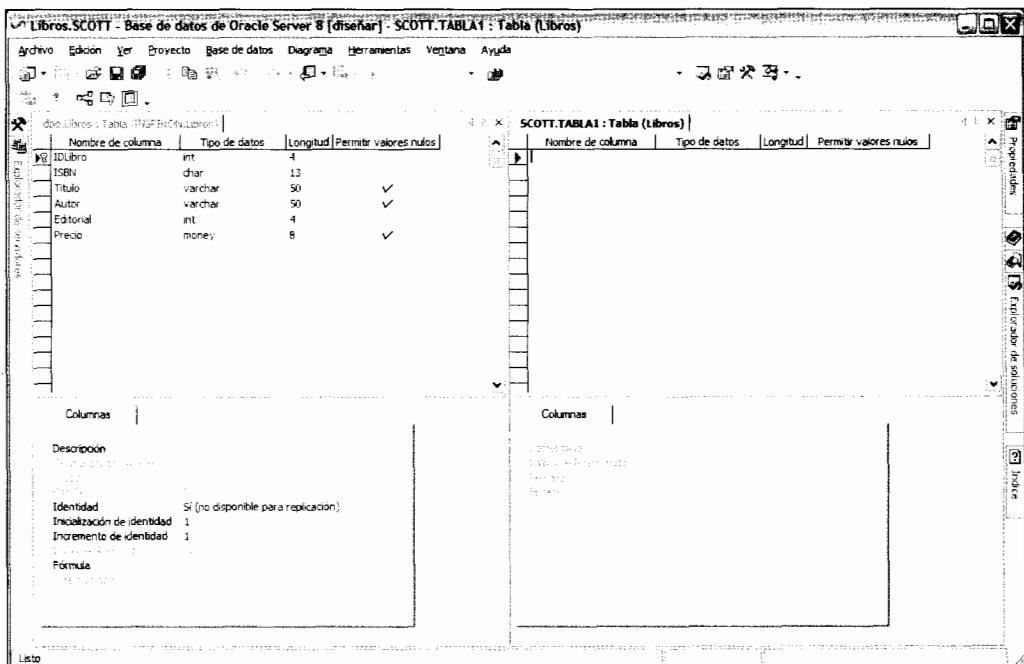


Figura 13.18. Podemos tanto modificar la estructura de una tabla existente como crear nuevas tablas



Figura 13.19. Barra de botones específica para el diseño de la estructura de una tabla

- **Establecer clave principal:** Al pulsar este botón se establece como clave principal la columna, o columnas, que tuviésemos seleccionada en ese momento. Esto produce, en la secuencia de comandos SQL, una restricción de clave primaria y otra que impide la introducción de valores nulos.
- **Relaciones:** En caso de que estemos diseñando múltiples tablas, y las relaciones existentes entre ellas, este botón nos llevará a la página Relaciones de la ventana de propiedades de la tabla. En ella podemos establecer la relación que esta tabla tenga con otra. En la figura 13.21, por ejemplo, está relacionándose la tabla TABLA3, que presumiblemente va a registrar movimientos de cuentas, con la tabla TABLA1, que contiene los datos de cada cuenta. De esta manera se crea una clave externa.

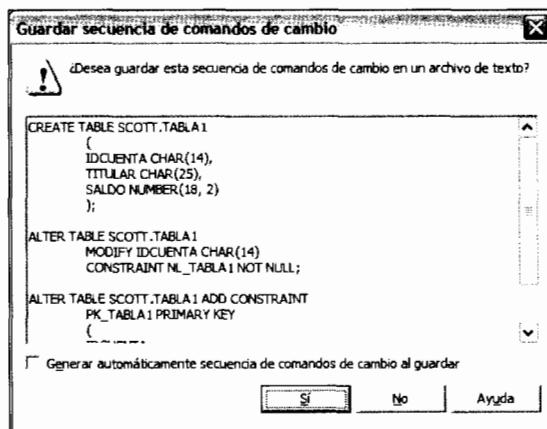


Figura 13.20. Secuencia de comandos de creación de una tabla

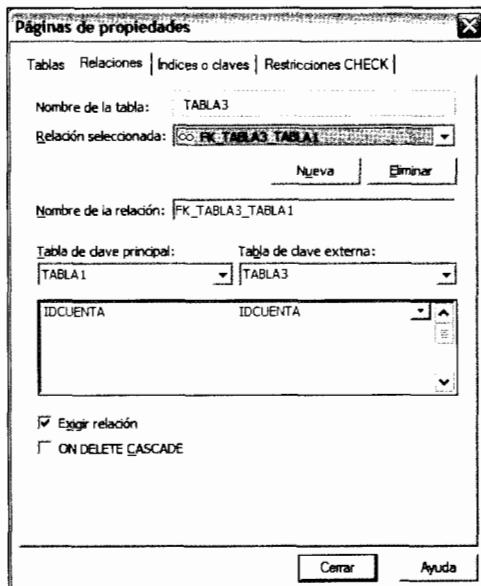


Figura 13.21. Establecemos las relaciones entre tablas

- **Administrar índices y claves:** Abre la misma ventana que se muestra en la figura 13.21, pero con la página **Índices o claves** en primer plano. Desde ella podrán crearse nuevos índices asociados a la tabla, así como establecer la clave principal en caso de que no la hubiésemos fijado previamente con el botón **Establecer clave principal**. No tiene más que ir pulsando el botón **Nuevo** de la página **Índices o claves** para ir añadiendo índices, seleccionando los nombres de las columnas participantes e introduciendo el nombre que tendrá el índice.

- **Administrar restricciones Check:** El último de los botones nos lleva a la página **Restricciones CHECK** de la misma ventana de propiedades (véase figura 13.22), en la que podemos definir restricciones adicionales que pudiéramos necesitar.

Creadas las tablas, tras cerrar las ventanas de diseño y responder afirmativamente a la pregunta de si se desean guardar los cambios, podemos editar su contenido como con cualquier otra tabla.

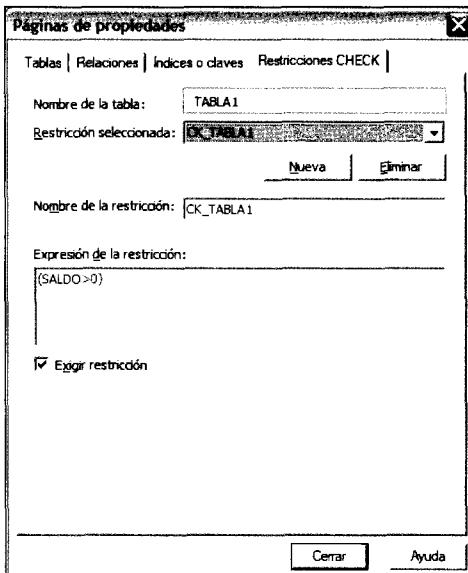


Figura 13.22. Definimos una restricción para evitar que el saldo de la cuenta pueda ser negativo

Diseño de vistas

Los datos obtenidos a partir de una vista pueden abrirse en Visual Studio .NET como si de una tabla normal se tratase. Puede desplegar la carpeta **Vistas** de la base de datos SQL Server u Oracle y hacer doble clic sobre la vista **LibrosPorEditorial** que se había creado como ejemplo en un capítulo previo. Verá que el resultado es una cuadrícula de datos, como la de una tabla, pero mostrando el resultado de la consulta que define la vista.

Si abrimos el menú emergente asociado a esa vista, o a cualquier otra, veremos dos opciones especialmente útiles: **Diseñar vista** y **Nueva vista**. La primera permite modificar la definición de una vista existente, mientras que la segunda crea una nueva vista. Recuerde que en el tercer capítulo usábamos herramientas específicas para efectuar estas tareas, mientras que ahora lo hacemos todo desde el propio entorno de Visual Studio .NET.

El diseño de una vista se efectúa a través de los paneles de diagrama, cuadrícula y SQL que conocimos en el punto dedicado a la edición de datos. La diferencia es que la sentencia SQL resultante será asociada a una vista. Seleccione la vista que teníamos como ejemplo, abra el menú emergente y luego elija la opción **Diseñar vista**. Deberá encontrarse con un entorno como el de la figura 13.23. Ya que la vista existía, aparecen las tablas que la forman, sus relaciones y selección de columnas y criterios. Podemos introducir modificaciones en cualquiera de esos apartados, por ejemplo estableciendo un criterio de selección de filas, y luego cerrar la vista respondiendo afirmativamente a la pregunta de si desea guardarla. Con esto habrá modificado la vista en el servidor de datos. Usando esos mismos elementos puede crear una nueva vista. Pruebe a añadir una vista a las bases de datos existentes para familiarizarse con los distintos paneles y la barra de herramientas asociada.

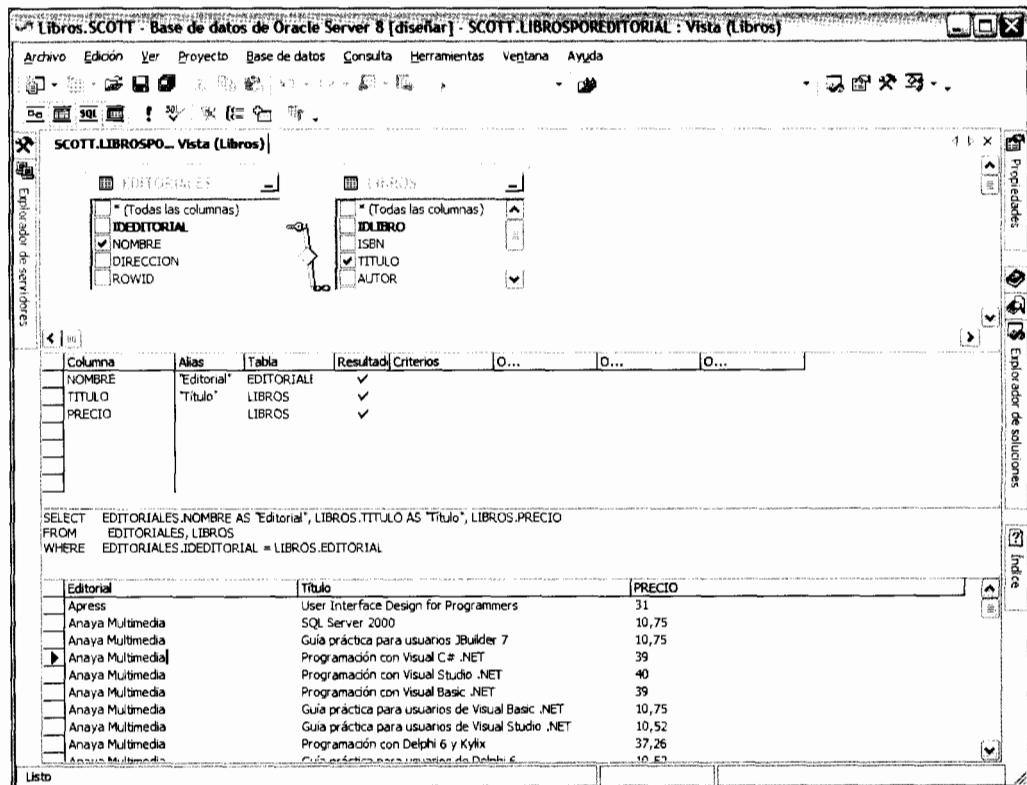


Figura 13.23. Elementos para el diseño de una vista

Edición de procedimientos almacenados y funciones

Desde el entorno de Visual Studio .NET es posible ejecutar procedimientos almacenados y funciones que residen en un RDBMS, así como editarlos, crearlos y,

en el caso de SQL Server, incluso depurarlos. Si hace clic con el botón secundario del ratón sobre un procedimiento almacenado, verá aparecer un menú emergente similar al de la figura 13.24. En él se encuentran las opciones adecuadas para ejecutarlo, editar el código, depurarlo o crear un nuevo procedimiento almacenado.

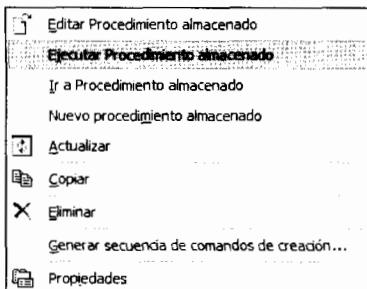


Figura 13.24. Menú de opciones asociado a un procedimiento almacenado SQL Server

Al ejecutar un procedimiento almacenado, verá aparecer los resultados en el panel **Resultados** de Visual Studio .NET, como si hubiese ejecutado cualquier aplicación. Si el procedimiento tiene parámetros de entrada, como ocurre con uno de los que definimos a modo de ejemplo en el tercer capítulo, aparecerá un pequeño cuadro de diálogo como el de la figura 13.25, mostrando una fila por cada parámetro que se precise. No tenemos más que introducir el valor y pulsar el botón **Aceptar** para ejecutar el procedimiento y ver el resultado.

Nota

Las operaciones descritas en este punto son aplicables, igualmente, a las funciones, que pueden crearse, editarse y ejecutarse desde el entorno de Visual Studio .NET.

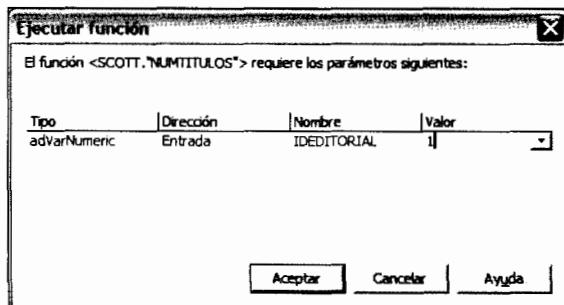


Figura 13.25. Parámetros necesarios para ejecutar un procedimiento almacenado

Los procedimientos almacenados, ya sean existentes o nuevos, se manipulan en el entorno de Visual Studio .NET en el editor de código, como si de un módulo cualquiera se tratase. Este editor, no obstante, reconoce la sintaxis del lenguaje en el que esté escrito el procedimiento almacenado, ofreciendo opciones específicas para ejecutarlo o diseñar aquellos bloques SQL que efectúan selecciones de datos.

Observe la figura 13.26, en la que está editándose el procedimiento almacenado NumTitulosEditorial de SQL Server. En la parte superior aparece el código del procedimiento almacenado. Fíjese en cómo la selección de datos se encuentra delimitada por una línea.

Mediante el menú emergente se ha ejecutado el procedimiento, obteniéndose el resultado que aparece en la parte inferior.

```

dbo.NumTitulos_SPTRONLibros]
ALTER PROCEDURE NumTitulosEditorial
AS
BEGIN
    SELECT E.Nombre, COUNT(L.Titulo) NumTitulos
    FROM Editoriales E, Libros L
    WHERE E.IDEditorial = L.Editorial
    GROUP BY E.Nombre
END

```

Nombre	NumTitulos
Anaya Multimedia	14
Apress	1
McGraw-Hill	1
No hay más resultados.	
3 filas devueltas	
@RETURN_VALUE = 0	
Ejecución de dbo."NumTitulosEditorial" finalizada.	

Figura 13.26. Edición de un procedimiento almacenado

Mediante la opción **Diseñar bloque SQL** podrá diseñar visualmente el bloque de SQL que aparece delimitado por la línea, sirviéndose para ello de los mismos paneles que utilizaría al diseñar una vista. Si trabaja con SQL Server, como en este caso, puede establecer puntos de parada en el código del procedimiento a fin de poder depurarlo, como haría en cualquier programa.

Nota

Si utiliza la opción Nuevo procedimiento almacenado para crear nuevos procedimientos, observará que el esqueleto de código que aparece en el editor se ajusta a la sintaxis propia del RDBMS con el que estemos trabajando. Esto es así, al menos, con SQL Server y Oracle.

Componentes de acceso a datos

Ahora que ya sabemos cómo emplear las herramientas visuales de bases de datos de Visual Studio .NET para preparar, o conocer, la información sobre la que vamos a trabajar, el paso siguiente será aprender a usar los componentes de acceso a datos con que contamos. En realidad, esos componentes ya los conocemos, la única novedad es que en lugar de crearlos mediante código, como en capítulos previos, usaremos operaciones de arrastrar y soltar y la ventana Propiedades.

Si inicia una aplicación típica Windows, o para la Web, verá que en la Caja de herramientas existe una sección llamada Datos. En ella aparecen, como se aprecia en la figura 13.27, los componentes OleDbConnection, OleDbDataAdapter, OleDbCommand y los equivalentes para el proveedor SqlConnection, SqlCommand, SqlDataAdapter y DataView.

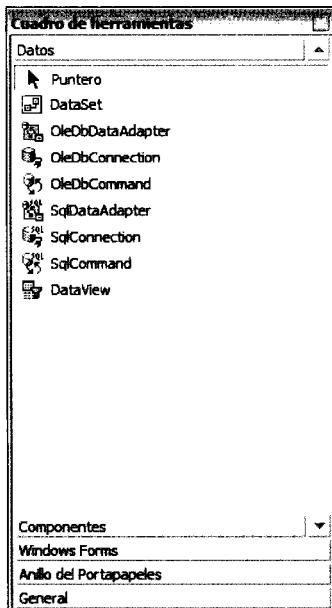


Figura 13.27. Componentes de la sección Datos en el Cuadro de herramientas

En principio no contamos con componentes específicos para los proveedores OracleClient y Odbc. Siempre que tenga dichos proveedores instalados, puede seleccionar la opción **Personalizar cuadro de herramientas** del menú emergente del **Cuadro de herramientas**, abrir la página **Componentes de .NET Framework** y activar los componentes **OracleCommand**, **OracleCommandBuilder**, **OracleConnection** y **OracleDataAdapter**, como se ha hecho en la figura 13.28, y/o los componentes **OdbcCommand**, **OdbcCommandBuilder**, **OdbcConnection** y **OdbcDataAdapter**. Tras cerrar el cuadro de diálogo, pulsando el botón **Aceptar**, verá aparecer en el **Cuadro de herramientas** todos esos componentes.

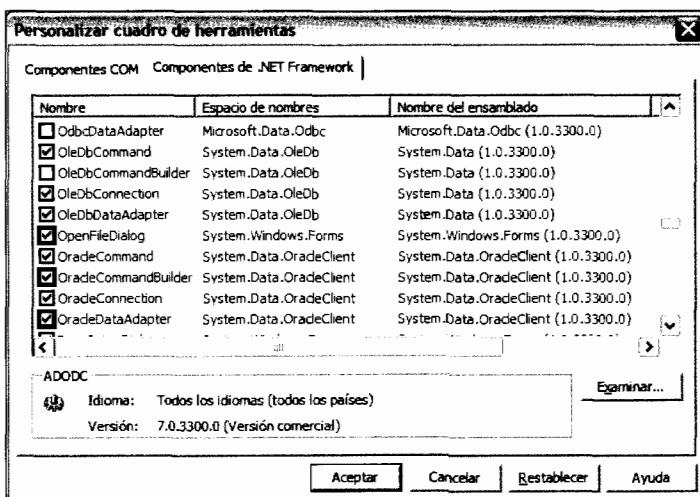


Figura 13.28. Selección de los componentes correspondientes a los demás proveedores

Teniendo todos los componentes dispuestos en el **Cuadro de herramientas**, en los puntos siguientes vamos a usarlos para establecer una conexión, definir un comando, preparar un adaptador de datos y generar un conjunto de datos, todo ello sin necesidad de escribir código alguno.

Preparación de la conexión

Dependiendo del proveedor que deseé emplear, haga doble clic sobre el componente **Connection** adecuado del **Cuadro de herramientas**, consiguiendo así su inserción en el módulo que tenga abierto en ese instante. Suponiendo que deseé acceder a los datos alojados en una base de datos Access o un libro de Excel, haría entonces doble clic sobre el componente **OleDbConnection**. Éste aparecerá en el área de diseño, en la zona inferior, y sus propiedades estarán visibles en la ventana **Propiedades**.

La única propiedad que tendremos que editar será **ConnectionString**, a fin de facilitar la información de conexión necesaria. Esta propiedad cuenta con una

lista que, al desplegarse, muestra la lista de conexiones que hay predefinidas en el Explorador de servidores, pudiendo seleccionar directamente cualquiera de ellas. Si no nos interesa ninguna de ellas, no tenemos más que seleccionar la opción Nueva conexión para abrir el cuadro de diálogo Propiedades de vínculo de datos y definir las propiedades de la conexión.

Nota

El componente `SqlConnection` se comporta prácticamente de forma idéntica a `OleDbConnection`, de tal forma que la propiedad `ConnectionString` también muestra la lista desplegable con la opción Nueva conexión. Los componentes `OracleConnection` y `OdbcConnection`, por el contrario, no disponen de esa lista ni asistente alguno, por lo que la cadena de conexión debe escribirse directamente como valor de la propiedad `ConnectionString`.

En la figura 13.29 puede ver cómo se ha insertado un `OleDbConnection` en una aplicación Windows y cómo, abriendo la lista adjunta a la propiedad `ConnectionString`, se elige una de las conexiones predeterminadas, concretamente la de acceso al libro Microsoft Excel.

Si nos interesa, podemos cambiar el nombre del componente por otro más adecuado. En este ejemplo mantendremos los nombres por defecto que tienen los componentes al ser insertados.

Tras la inserción del componente de conexión, y asignación de un valor apropiado a `ConnectionString`, para abrir la conexión usaríamos el componente como si de una variable previamente declarada se tratase, es decir, como haríamos con cualquier otro componente en Visual Basic .NET. La sentencia siguiente, por tanto, usaría el componente insertado en la figura 13.29 para abrir la conexión con el libro de Microsoft Excel.

```
OleDbConnection1.Open()
```

Definición de comandos

Con los componentes `Command`, por ejemplo `OleDbCommand`, prepararemos los comandos de selección de datos que precisemos. La única diferencia es que ahora el componente se crea al insertarlo en el diseñador, creándose automáticamente la variable, y las propiedades del comando se establecen visualmente. Nos encontraremos, no obstante, con exactamente las mismas propiedades que ya conocimos en capítulos previos, lo único que cambia es la forma de crear el objeto y personalizarlo.

Asumiendo que tiene en este momento definida la conexión del punto previo, inserte un `OleDbCommand`, simplemente haciendo doble clic sobre él en el Cuadro de herramientas, y edite en la ventana **Propiedades** las propiedades siguientes:

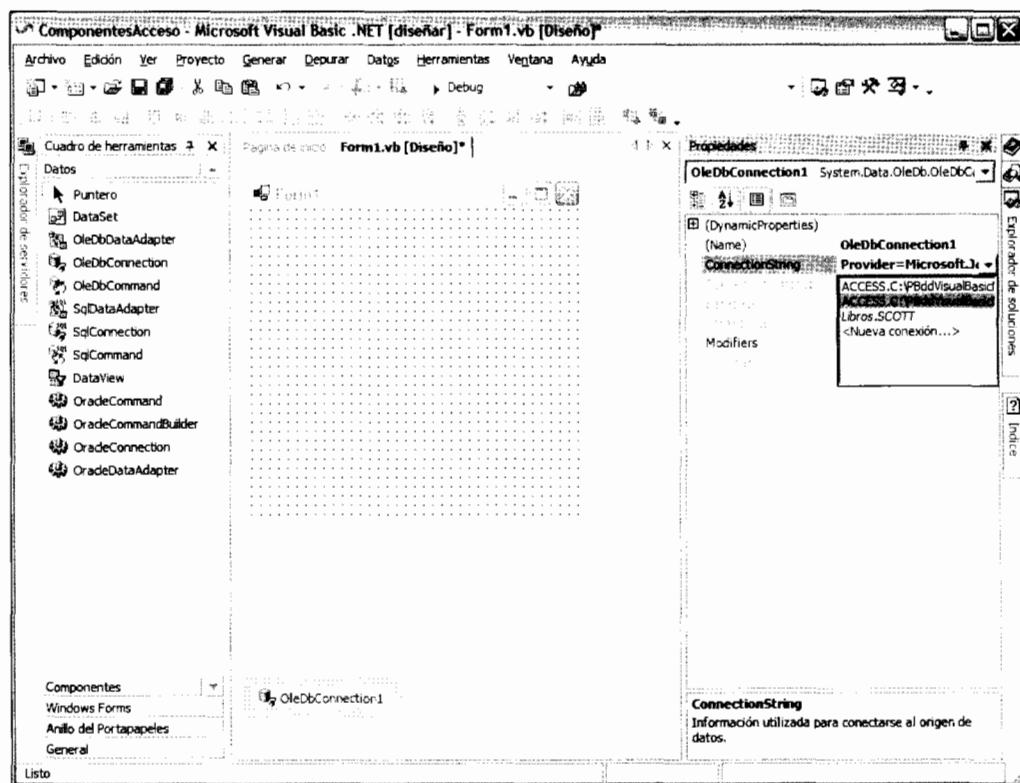


Figura 13.29. Selección de la cadena de conexión

- **Connection:** Despliegue la lista adjunta y, del nodo **Existente**, elija luego la conexión **OleDbConnection1**. De esta manera ya tiene asociado el comando con la conexión.

Nota

Si no tiene un componente de conexión, porque no lo haya insertado previamente, puede usar la opción **Nueva** de la propiedad **Connection** para definirla en ese mismo momento, generándose automáticamente el componente **OleDbConnection**.

- **CommandType:** Esta propiedad también tiene asociada una lista de opciones, de la que puede elegir si el comando recuperará directamente una tabla, contiene una sentencia SQL o el nombre de un procedimiento almacenado. Mantenemos el valor **Text** que aparece por defecto, indicando que el comando tendrá asociada una sentencia SQL.

- CommandText: Al seleccionar esta propiedad verá aparecer un botón, en el extremo derecho, con tres puntos suspensivos. Púlselo para abrir la ventana Generador de consultas (véase figura 13.30) y, usando los paneles que ya conocíó en puntos previos, diseñar visualmente la selección de datos.

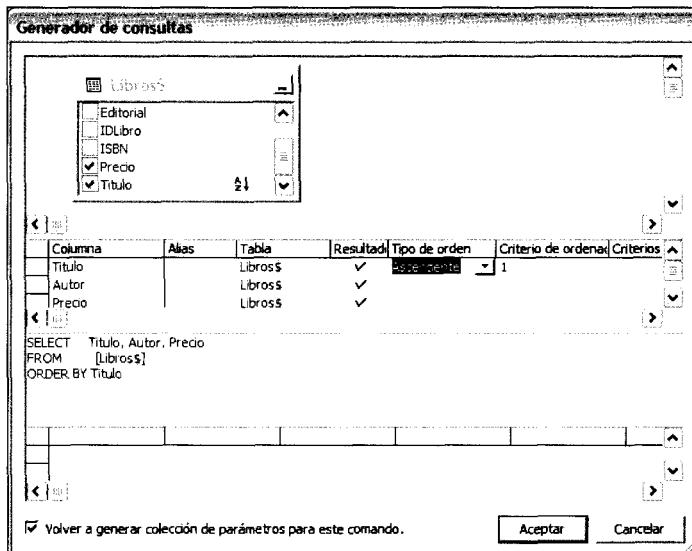


Figura 13.30. Generador visual de consultas

Nota

La ventana Generador de consultas no aparece, en este caso, como un diseñador de Visual Studio .NET sino como una ventana independiente.

Lógicamente, podemos definir tantos comandos como necesitemos para efectuar nuestro trabajo. En este caso nos quedaremos sólo con el comando que puede verse en la figura 13.30, introducido en el componente OleDbCommand1.

Definición del adaptador de datos

En este momento, tras definir el comando, podríamos ponernos a escribir código para, mediante el método `ExecuteReader()`, obtener un `OleDbDataReader` y recuperar datos. Nuestro objetivo, sin embargo, es recuperar dichos datos en un `DataSet` con el fin de vincularlo con un `DataGridView`, así que el paso siguiente será definir el adaptador de datos que, partiendo del comando, llenará el `DataSet`.

Haga doble clic sobre el elemento `OleDbDataAdapter`, en el Cuadro de herramientas, para insertar un objeto de esa clase en la aplicación. Se pondrá en marcha

un asistente de configuración que, en este caso, vamos a cerrar pulsando el botón **Cancelar**. Así accederemos directamente a las propiedades del adaptador. Si observa la ventana **Propiedades** verá algunas como **SelectCommand**, **TableMappings** o **MissingSchemaAction**, que ya conoce de capítulos previos. Despliegue la lista adjunta a **SelectCommand** y elija del nodo **Existente** el único elemento disponible. Con esto ya ha asociado el adaptador de datos con el comando definido antes.

Seleccione la propiedad **TableMappings** y luego pulse el botón que aparece tras (**colección**), accediendo a la ventana mostrada en la figura 13.31. En ella asociaremos la única tabla de entrada, procedente de la sentencia SQL definida en el comando, con una tabla que existirá en el **DataSet** a la que llamaremos **Libros**. Pulsamos el botón **Aceptar** y damos por acabada la configuración del adaptador.

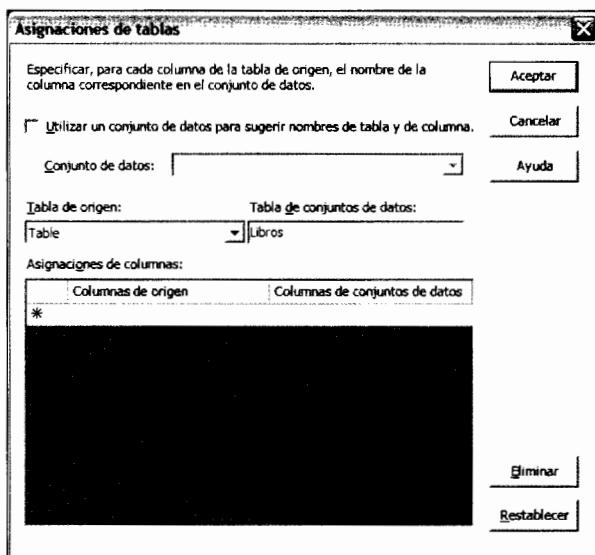


Figura 13.31. Asociaciones entre tablas de origen y tablas del conjunto de datos

Nota

En el mismo cuadro de diálogo **Asignaciones de tablas** puede también establecer una correspondencia entre las columnas del origen de datos y las que existirán en el conjunto, si es que éste cuenta con una estructura predefinida.

Creación del conjunto de datos

El penúltimo paso, en cuanto a tareas de diseño se refiere en este simple ejemplo, será la creación del conjunto de datos, para lo cual hacemos doble clic sobre el elemento **DataSet** que aparece en el **Cuadro de herramientas**. Como en el caso del

adaptador de datos, se pondrá en marcha un asistente que cerraremos pulsando el botón **Cancelar**. De esta forma tenemos un conjunto de datos simple y vacío, como los que utilizábamos en los capítulos previos.

La ventana **Propiedades** muestra todas las propiedades del **DataSet**, resultando especialmente interesantes **EnforceConstraints**, **Relations** y **Tables**, que ya conocemos. Puesto que trabajaremos sobre una sola tabla, no van a existir relaciones, pero abriendo la colección **Relations** tendremos acceso a un cuadro de diálogo desde el cual pueden ir definiéndose esas propiedades de manera visual.

Sí vamos a tener una tabla, así que seleccionamos la propiedad **Tables** y abrimos el correspondiente editor de esta colección. En principio no aparece elemento alguno, así que pulsamos el botón **Agregar** para insertar una nueva tabla. A la derecha, como se ve en la figura 13.32, aparecen todas sus propiedades. Sólo vamos a modificar **TableName**, asignándole el mismo nombre que dimos a la tabla en el adaptador de datos, es decir, **Libros**.

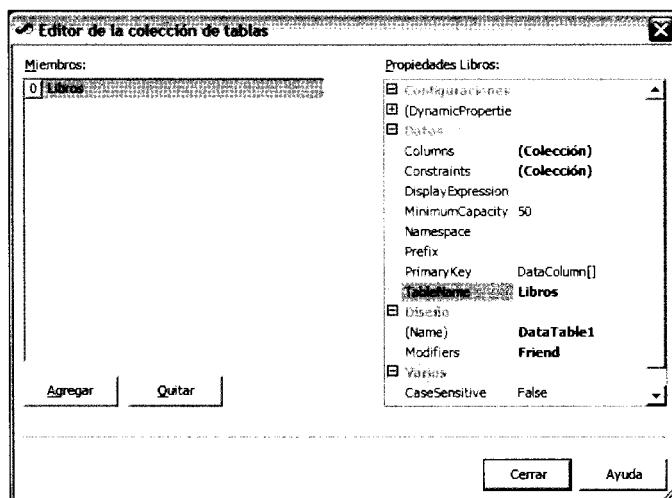


Figura 13.32. Edición de los elementos de la colección **Tables** de un **DataSet**

Nota

Observe que, a diferencia del resto de componentes, el **DataSet** no se enlaza directamente con el adaptador de datos o cualquier otro componente. Será mediante código como lo llenemos con datos.

Diseño de una sencilla interfaz

Para ver el contenido del **DataSet** en esta ocasión no usaremos la consola, sino que diseñaremos una sencilla interfaz de usuario. Abra la sección **Windows Forms**

del Cuadro de herramientas y tome un componente DataGrid, insertándolo en el formulario. Modifique la propiedad Dock para que ocupe todo el espacio disponible. Despliegue la lista adjunta a la propiedad DataSource y seleccione el elemento DataSet1, vinculando así la rejilla de datos con el conjunto de datos. Dado que éste puede contar con múltiples tablas, usaremos la propiedadDataMember para seleccionar la que deseemos. En este caso sólo hay una disponible, así que la elegimos.

La interfaz, finalizada, quedaría como se aprecia en la figura 13.33. Observe en la parte inferior del área central los componentes que se han ido insertando en los pasos previos.

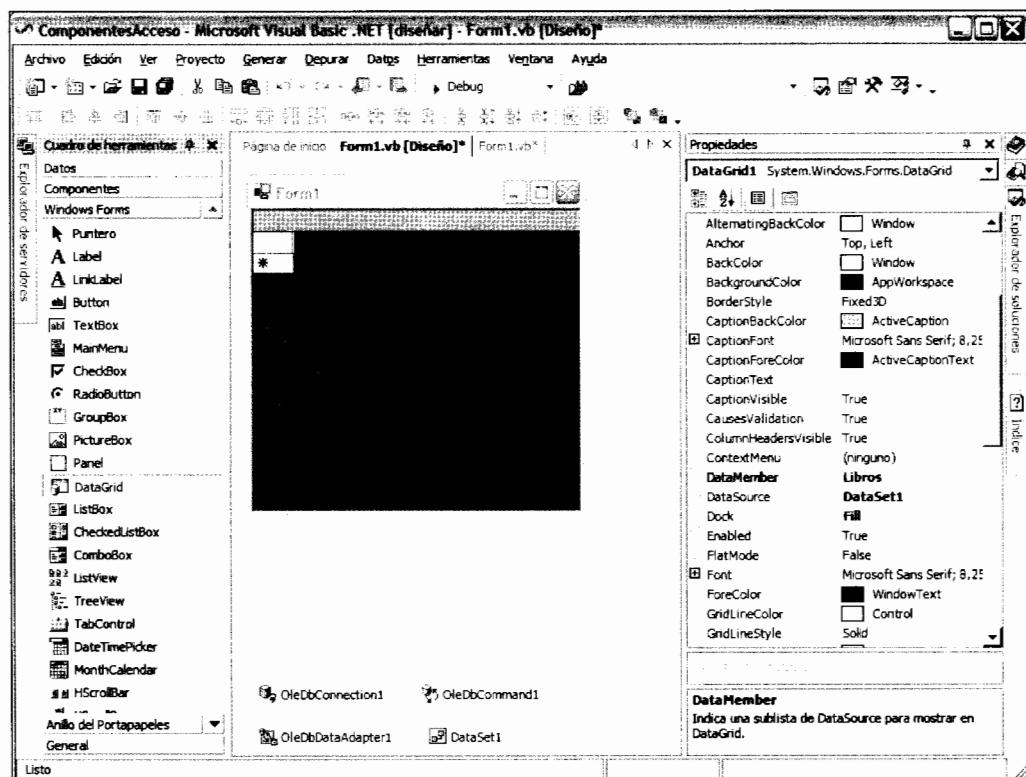


Figura 13.33. Aspecto del formulario con el DataGrid en su interior

Si ejecutásemos el programa tal y como está en este momento, veríamos que la rejilla de datos permanece vacía. Es lógico, puesto que la hemos vinculado con un DataSet que está vacío y, además, en ningún momento hemos vinculado éste con el adaptador de datos. Para hacerlo, abrimos la ventana de código y, en el evento Load del formulario, insertamos la sentencia siguiente:

```
OleDbDataAdapter1.Fill(DataSet1)
```

No es mucho código, especialmente si lo comparamos con algunos de los ejemplos de capítulos previos y el resultado obtenido que, en este caso, sería el de la figura 13.34. Una rejilla en la que podemos navegar por los datos.

Titulo	Autor	Precio
Guía práctica para usuarios de Delphi 6	Francisco Charte	10,52
Guía práctica para usuarios de Excel 2002	Francisco Charte/M. Jesús Luque	10,52
Guía práctica para usuarios de Kylix	Francisco Charte	10,52
Guía práctica para usuarios de Visual Basic .NET	Francisco Charte	10,75
Guía práctica para usuarios de Visual Studio .NET	Francisco Charte	10,52
Guía práctica para usuarios JBuilder 7	Francisco Charte	10,75
Introducción a la programación	Francisco Charte	24,04
Manual avanzado Excel 2002	Francisco Charte	21,04
Manual del microprocesador 80386	Chris H. Pappas & William H. Murray, III	40
Programación con Delphi 6 y Kylix	Francisco Charte	37,26
Programación con Visual Basic .NET	Francisco Charte	39
Programación con Visual C# .NET	Francisco Charte	39
Programación con Visual Studio .NET	Francisco Charte/Jorge Serrano	40
SQL Server 2000	Francisco Charte	10,75
User Interface Design for Programmers	Joel Spolsky	31
*		

Figura 13.34. Ventana con la rejilla de datos

Creación automática de componentes

No cabe duda, usando los componentes de conexión a datos como hemos hecho en los puntos previos, insertándolos en un diseñador y personalizándolos mediante el uso de la ventana **Propiedades**, ahorraremos una cantidad importante de trabajo respecto a la escritura de código para efectuar todo el proceso. Tan sólo hemos tenido que escribir una sentencia para conseguir una cuadrícula de datos que muestra el resultado de una consulta SQL, sin necesidad de recorrer filas, acceder a columnas y acciones similares mediante código.

No obstante, el trabajo puede simplificarse aún más mediante la creación automática de parte de los componentes que, en los anteriores puntos, hemos ido insertando y personalizando de manera individual. A continuación vamos a tratar de obtener el mismo resultado, o similar, con menos pasos y en menor tiempo. Puede partir de un nuevo proyecto o bien eliminar el código y los componentes que insertó en el ejemplo previo.

El asistente de configuración de adaptadores

Los adaptadores de datos cuentan con un asistente que facilita su configuración, asistente que anteriormente cerramos sin usarlo. Ahora, partiendo de un proyecto en el que tan sólo tenemos un formulario vacío, vamos a hacer doble clic sobre el

elemento `OleDbDataAdapter` del Cuadro de herramientas para poner ese asistente en marcha.

La primera ventana que aparece es simplemente indicativa, bastando con pulsar el botón **Siguiente >** para ir al siguiente paso. En él deberemos seleccionar la conexión asociada al adaptador, ya sea eligiendo una de las existentes, tal y como se hace en la figura 13.35, o bien pulsando el botón **Nueva conexión** para definir una nueva. En este caso nos encontraremos con el cuadro de diálogo que ya conocemos. Para seguir este ejemplo vamos a optar por la conexión `Libros.dbo` del servidor en el cual se tenga SQL Server en funcionamiento, en mi caso particular `INSPIRON.Libros.dbo`.

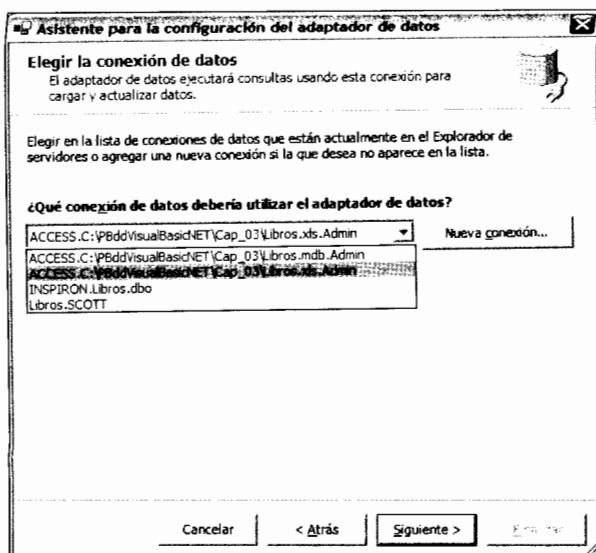


Figura 13.35. Seleccionamos la conexión que utilizará el adaptador de datos

Tras pulsar el botón **Siguiente >** nos encontramos con el apartado **Elija un tipo de consulta**. En ella, según se ve en la figura 13.36, podemos elegir entre tres opciones distintas:

- **Usar instrucciones SQL:** Para actualizar los datos, en caso de ser necesario, se emplearán sentencias SQL del tipo UPDATE, INSERT o DELETE. Éstas se generarán automáticamente para nosotros a partir de la sentencia de selección que facilitemos en el paso siguiente.
- **Crear nuevos procedimientos almacenados:** En lugar de preparar comandos con sentencias SQL, esta opción crea en el RDBMS un procedimiento almacenado para obtener los datos, otro para actualizar, otro para insertar y otro para borrar, utilizándolos cuando sea necesario. De esta forma las sentencias SQL ya se encontrarán compiladas en el RDBMS y el rendimiento será superior.

- **Usar procedimientos almacenados existentes:** En caso de que ya hubiésemos empleado la opción anterior en otro adaptador, en el RDBMS existirán los procedimientos almacenados apropiados para efectuar las tareas de edición, por lo que usaríamos esta opción con el fin de aprovecharlos en lugar de generar otros nuevos.

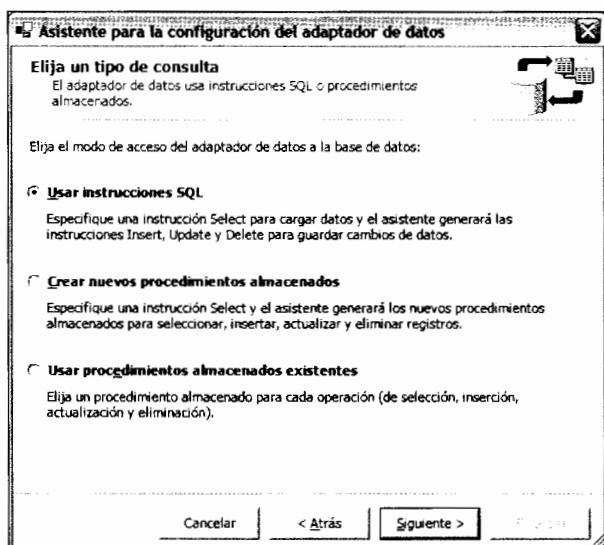


Figura 13.36. Seleccionamos el mecanismo que se utilizará para la actualización de datos

Si elegimos una de las dos primeras opciones, en nuestro caso nos quedaremos con la que aparece seleccionada por defecto, en el paso siguiente tendremos que introducir la sentencia SQL de selección de datos. Si ésta es compleja, no tan sencilla como la de la figura 13.37, puede pulsar el botón **Generador de consultas** para diseñar visualmente, utilizando la misma interfaz que ya conoce de ejemplos anteriores.

Pulsando el botón **Opciones avanzadas** de esta ventana, en la parte inferior izquierda, abrirá entonces la ventana **Opciones de generación SQL avanzadas** (véase figura 13.38). En ella aparecen tres opciones inicialmente activadas. La primera provoca que el asistente genere, aparte del comando de selección, comandos adicionales para la inserción, actualización y eliminación. Con la segunda se conseguirá que las instrucciones de actualización y eliminación empleen el mecanismo conocido como *conurrencia optimista*, consistente en asumir que los datos no van a cambiar desde que se recuperan hasta que van a modificarse, no bloqueando el acceso a ellos. Por último, la tercera opción provoca que tras cualquier actualización se efectúe una actualización inmediata del conjunto de datos, asegurando así que éste siempre se mantiene en consonancia con el contenido real del origen de datos.



Figura 13.37. Introducimos la sentencia SQL de selección de datos

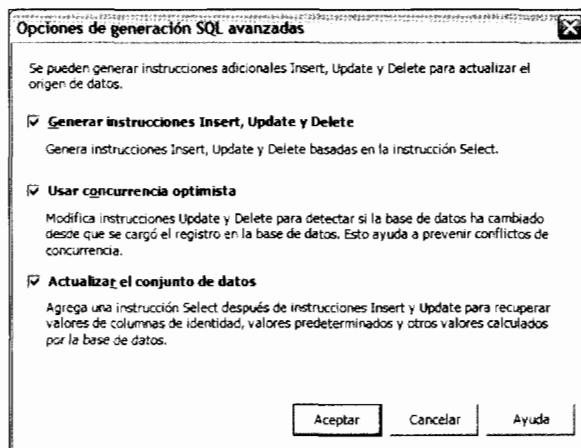


Figura 13.38. Opciones avanzadas de generación de sentencias SQL

Tras pulsar el botón **Siguiente >** una vez más, accederá a una última ventana de resumen y confirmación. En ella se detallan las acciones elegidas, bastando la pulsación del botón **Finalizar** para cerrar el asistente y concluir el proceso. Si no está conforme con algo, aún está a tiempo de pulsar el botón **< Atrás** y corregir lo que deseé. Al finalizar, aparecerá en el diseñador el componente `OleDbDataAdapter` y un `OleDbConnection`. No existen, sin embargo, componentes `OleDbCommand`, al menos aparentemente. Sin embargo, si examina el código generado por el asistente encontrará las siguientes declaraciones justo antes del método `InitializeComponent()`:

```
Friend WithEvents OleDbSelectCommand1 As _
    System.Data.OleDb.OleDbCommand
Friend WithEvents OleDbInsertCommand1 As _
    System.Data.OleDb.OleDbCommand
Friend WithEvents OleDbUpdateCommand1 As _
    System.Data.OleDb.OleDbCommand
Friend WithEvents OleDbDeleteCommand1 As _
    System.Data.OleDb.OleDbCommand
```

Existe, por tanto, un `OleDbCommand` para selección, otro para inserción, un tercero para actualización y uno más para eliminación de filas.

Busque más adelante la asignación a la propiedad `CommandText` de cada uno de los `OleDbCommand` y examine las sentencias SQL, especialmente las utilizadas para actualizar y eliminar filas existentes.

Nota

Puede cambiar la configuración de un adaptador de datos en cualquier momento, mediante el mismo asistente empleado para crearlo, usando el enlace **Configurar adaptador de datos** que aparece en la parte inferior de la ventana **Propiedades** mientras tiene seleccionado el adaptador, o bien eligiendo la opción homónima del menú emergente del `OleDbAdapter`.

Comprobación del adaptador de datos

Acabamos de configurar nuestro adaptador de datos que, una vez lo conectemos con un `DataSet`, suponemos nos facilitará la visualización y edición de todas las filas de la tabla `Libros`. Esta suposición puede convertirse en seguridad de forma muy sencilla: seleccionando la opción **Vista previa de datos** del menú emergente asociado al adaptador.

En principio se encontrará con un cuadro de diálogo en el que aparece el nombre del adaptador de datos, en la parte superior izquierda, y varias áreas vacías en el resto de la ventana.

Pulse el botón **Llenar conjunto de datos**. De inmediato verá cómo aparece el conjunto de filas resultante, así como el nombre de las tablas de datos, en este caso sólo una, y el tamaño global del conjunto de datos. La figura 13.39 muestra la vista previa del adaptador definido en el punto previo.

Puede cerrar la ventana, volviendo de nuevo al entorno de diseño a fin de proceder a la creación del conjunto de datos.

Generación del conjunto de datos

Partiendo del adaptador de datos, que ya tenemos definido y comprobado, generar el `DataSet` es un juego de niños. Bastará con abrir el menú emergente del

OleDbDataAdapter y seleccionar la opción Generar conjunto de datos, o bien hacer clic sobre ese enlace en la parte inferior de la ventana Propiedades.

Lo único que tenemos que hacer es elegir entre crear un nuevo DataSet, como en este caso, o bien introducir el conjunto de datos en uno que ya exista en la aplicación. Como se aprecia en la figura 13.40, dado que no tenemos DataSet alguno en el programa, optamos por crear uno nuevo e introducimos en él la única tabla resultante del adaptador. Dejamos marcada la opción Agregar este conjunto de datos al diseñador, de tal forma que el DataSet generado aparezca en la vista de diseño.

Si enlaza ahora el DataSet generado con el DataGrid que ya había en el formulario, observará que en la rejilla aparecen los nombres de las columnas. Aparentemente, lo único que se ha añadido al proyecto ha sido el archivo DataSet1.xsd contenido el esquema del conjunto de datos.

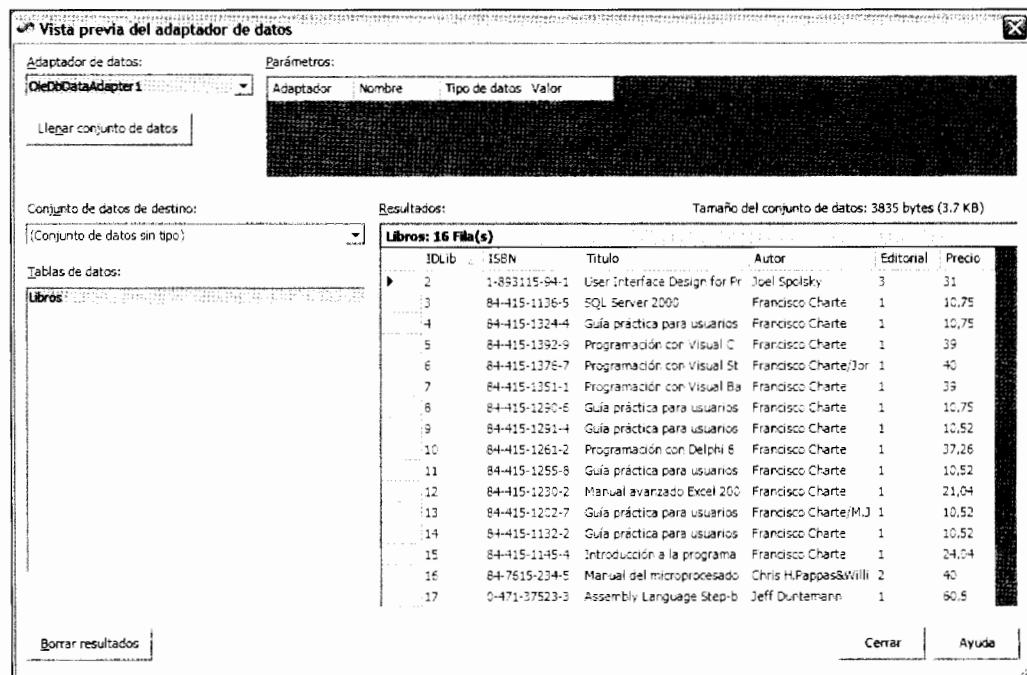


Figura 13.39. Comprobamos el resultado que genera el adaptador de datos

Conjuntos de datos con tipo

Hay ciertos casos en los que Visual Basic .NET genera módulos de código a partir de definiciones y los mantiene ocultos, evitando así su modificación accidental al tiempo que hace algo más *mágico* el funcionamiento del entorno. La creación de un tipo de datos, apuntada en el apartado anterior, es una de esas ocasiones.

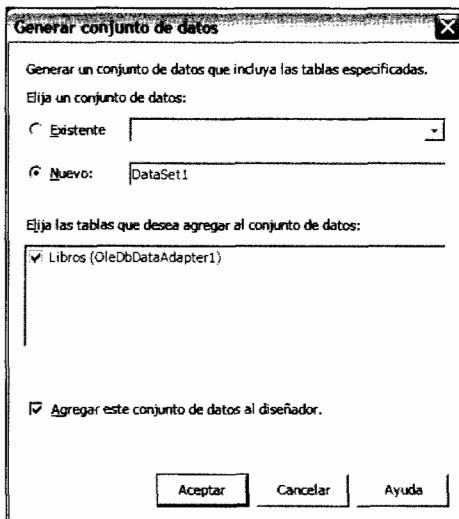


Figura 13.40. Opciones de generación del conjunto de datos

Aunque en principio parece que tan sólo se ha añadido el esquema XSD, si pulsa el botón **Mostrar todos los archivos** del Explorador de soluciones verá aparecer, asimismo, un módulo llamado `DataSet1.vb`. Ábralo y eche un vistazo a su contenido. En él encontrará una clase llamada `DataSet1` derivada de `DataSet`. Esta clase cuenta con una propiedad sólo de lectura, llamada `Libros`, de tipo `Libros-DataTable`, una clase definida más adelante en la que encontrará objetos `DataColumn` con nombres que le resultarán familiares, al coincidir con los nombres de las columnas de la tabla `Libros`.

Lo que ha hecho el asistente anterior es generar un *conjunto de datos con tipo*, una clase que, derivada de `DataSet`, implementa los elementos necesarios para facilitar el acceso a las tablas y columnas, asegurando al tiempo la validez de los tipos. La existencia de este conjunto de datos con tipo hace que sentencias como ésta:

```
DataSet1.Tables("Libros").Rows(0)("Editorial") = 3
```

en la que quiere accederse a la columna `Editorial` de la primera fila de la tabla `Libros`, puedan simplificarse así:

```
DataSet1.Libros(0).Editorial = 3
```

No es necesario emplear las colecciones `Tables`, `Rows` o `Columns`, ya que la clase `DataSet1` cuenta con propiedades que, como `Libros`, ya acceden a esos elementos por nosotros. El conjunto de datos con tipo nos ofrece, además, la integración con la tecnología *IntelliSense* del editor de código, como puede apreciarse en la figura 13.41. En la lista de miembros aparecen los nombres de las tablas y de las columnas del conjunto de datos, lo cual hace mucho más simple nuestro trabajo.

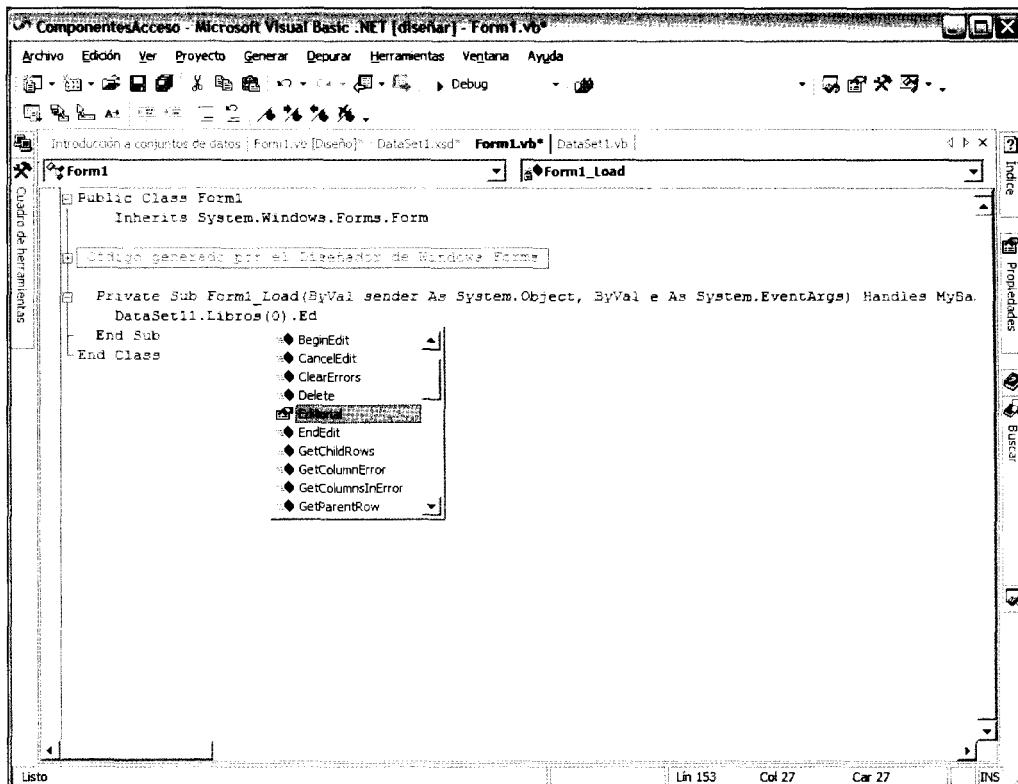


Figura 13.41. Miembros del conjunto de datos con tipo

Nota

Los conjuntos de datos con tipo evitan ciertos errores, como la asignación de valores de tipos incorrectos, produciendo los avisos durante la compilación, cuando es fácil corregirlos, en lugar de en ejecución.

Aún más simple

En los dos últimos puntos, utilizando el asistente de configuración de adaptadores de datos y el generador de conjuntos de datos, ha efectuado el mismo proceso y creado los mismos componentes que en los apartados anteriores, en mucho menos tiempo y con menos operaciones manuales. ¿Es posible hacerlo con aún menos trabajo? Pues sí, es posible.

Los elementos que aparecen en el Explorador de servidores pueden arrastrarse hasta un diseñador, y las tablas, vistas y procedimientos almacenados no son una excepción. ¿Qué ocurre si toma, por ejemplo, la tabla Editoriales desde la base

de datos SQL Server y la suelta sobre el formulario Windows? Hágalo y compruebe cómo se configura automáticamente el adaptador de datos y la conexión. Lo único que tiene que hacer es abrir el menú emergente del adaptador y elegir la opción **Generar conjunto de datos**. Ya tiene todo el trabajo anterior hecho en dos operaciones de ratón.

En lugar de una tabla puede arrastrar una vista, obteniendo el mismo resultado, e incluso un procedimiento almacenado, caso en el que se crea un `SqlCommand`, o equivalente, que facilita su ejecución.

Nota

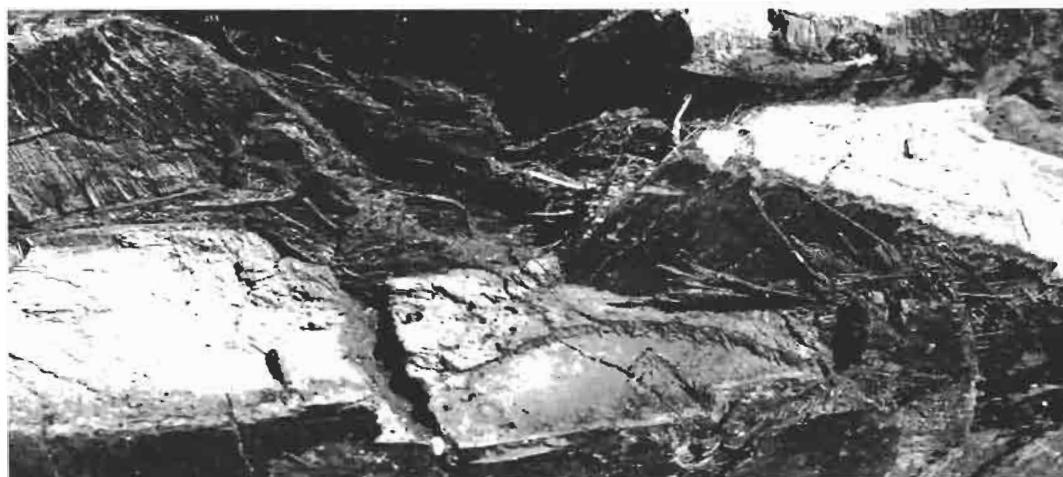
También puede arrastrar hasta un diseñador cualquiera de las conexiones que tenga predefinidas en la rama **Conexiones de datos** del Explorador de servidores, creando y configurando automáticamente un objeto `OleDbConnection`.

Resumen

Como ha podido ver en este capítulo, el entorno de Visual Studio .NET cuenta con potentes herramientas para el trabajo con bases de datos, simplificando tareas como la edición local de datos o información de esquema, incluso cuando se opera sobre un origen de datos remoto. Visual Studio .NET facilita, asimismo, la depuración de procedimientos almacenados SQL Server desde el propio entorno, lo que nos ahorra cambiar del entorno de desarrollo al de la base de datos al depurar una aplicación que tenga embebidas parte de las reglas de negocio en el servidor de datos.

Los componentes que en capítulos previos creábamos mediante código, como los adaptadores de datos, comandos y conjuntos de datos, pueden tomarse desde el **Cuadro de herramientas**, insertarse en un diseñador y personalizarse mediante la ventana **Propiedades**. En ocasiones, tal como se ha visto en los últimos puntos, la mayor parte del proceso de configuración puede efectuarse automáticamente, gracias a las opciones, asistentes y la posibilidad de arrastrar elementos, como las tablas y vistas, directamente desde el **Explorador de servidores** hasta un diseñador.

Una vez que sabemos cómo conectar con los orígenes de datos y definir adaptadores, comandos y conjuntos de datos, en el capítulo próximo veremos cómo conectar esos elementos con controles de interfaz, incluido el control `DataGridView` que hemos usado en el ejemplo de este capítulo.



14

Componentes con vinculación a datos

La presentación y solicitud de datos a través de la consola, empleada en los ejemplos de los capítulos de la segunda parte del libro, no es precisamente el mecanismo más flexible ni amigable para comunicarse con los usuarios de sus aplicaciones. Es mucho más habitual la elaboración de interfaces de usuario basadas en ventanas, ejecutándose por ejemplo sobre Windows, o bien las interfaces Web, accesibles desde un cliente como Internet Explorer. Aunque podríamos, mediante código, obtener los datos para presentarlos al usuario en componentes propios y, también mediante código, recuperar los cambios efectuados por el usuario para transmitirlos al origen de datos, lo cierto es que la mayor parte de ese proceso puede automatizarse gracias a la existencia de la vinculación a datos en ciertos componentes.

Mediante la vinculación a datos, o *data-binding*, es posible enlazar una vista, tabla o conjunto de datos con uno o varios controles de interfaz, ya sean Windows o Web, automatizando la presentación y recuperación de cambios. De esta forma nuestro trabajo se reduce en gran medida.

El objetivo de este capítulo es introducir los conceptos de vinculación a datos, así como los controles más interesantes en este aspecto. En el capítulo siguiente nos centraremos en el diseño de formularios de datos empleando estos controles.

Tipos de vinculación

La vinculación entre los orígenes de datos y los controles puede ser de varios tipos, según que el control sea Windows o Web y también dependiendo de que

pueda mostrar una sola columna de una fila o bien múltiples filas o columnas. En realidad, la vinculación puede efectuarse no sólo con una vista, una tabla o un conjunto de datos, sino también con una lista, un arreglo y, en general, con cualquier componente que implemente la interfaz `IList`.

Los controles que se utilizan habitualmente para diseñar interfaces de usuario, independientemente de que sean nativas Windows o de tipo Web, pueden clasificarse, en cuanto a su vinculación a datos, en las tres categorías siguientes:

- Controles que tan sólo precisan el contenido de una columna de una cierta fila, por ejemplo los `TextBox` y similares.
- Controles capaces de mostrar el valor que tiene una cierta columna en múltiples filas, por ejemplo un `ListBox` o `ComboBox`.
- Controles tipo cuadrícula en los que aparecen múltiples filas con múltiples columnas, como el `DataGrid` que usábamos en uno de los ejemplos del capítulo previo.

La conexión de cada una de estas categorías de control con el origen de datos se efectúa, como va a verse en los puntos siguientes, de forma distinta.

Vinculación simple

La mayoría de los componentes que estamos habituados a utilizar en el diseño de interfaces de usuario, tanto Windows como Web con ASP.NET, pueden vincular sus propiedades con un origen de datos, por regla general con una cierta columna de un conjunto de datos, vista o tabla.

Los controles Windows disponen de una propiedad, llamada `DataBindings`, que almacena la colección de vinculaciones o enlaces entre propiedades del control y una fuente de datos. Los controles ASP.NET emplean un sistema distinto, pero igualmente sencillo.

Los objetos de la colección a la que apunta `DataBindings` son de clase `Binding`, estableciendo cada uno de ellos un enlace para una propiedad del control. Esto significa, por tanto, que es posible definir múltiples vínculos con diferentes propiedades de un mismo control, a pesar de que no es lo más habitual. Cada uno de esos objetos mantiene tres datos distintos para hacer posible el enlace, datos que es preciso facilitar al constructor en el momento de la creación. Son éstos:

- **Nombre de la propiedad:** Los objetos `Binding` se añaden a la colección `DataBindings` de un cierto control, por lo que tan sólo precisan conocer, en cuanto al control de destino se refiere, el nombre de la propiedad que va a enlazarse.
- **Origen de datos:** El objeto del que va a leerse la información, normalmente un `DataSet`, `DataTable` o `DataView` aunque, como se apuntó anteriormente, también podría ser una lista o un arreglo.

- **Miembros de datos:** El origen de datos generalmente contendrá múltiples columnas, y los objetos Binding están pensados para vínculos simples, con una sola columna cuyo nombre hay que facilitar como miembro de datos.

Suponga que tiene un DataSet en el que ya existen dos DataTable, uno con libros y otro con editoriales, y que desea vincular la columna Titulo de la tabla Libros con la propiedad Text de un TextBox, para facilitar la visualización y manipulación por parte del usuario.

Asumiendo que el TextBox se llama TextBox1 y que tenemos un DataSet llamado DsLibros1 con la tabla Libros, añadiríamos el enlace con una sentencia como la siguiente:

```
TextBox1.DataBindings.Add(New  
    Binding("Text", DsLibros1.Libros, "Titulo"))
```

En realidad, usando el diseñador de formularios de Visual Studio .NET raramente tendremos que escribir sentencias de este tipo ya que, como otras, la propiedad DataBindings puede ser editada en la ventana Propiedades. Esta propiedad se encuentra en el apartado Datos, si tenemos las propiedades agrupadas por categorías, o bien como una de las primeras propiedades, si usamos el orden alfabético.

Al desplegar DataBindings nos encontramos con tres elementos, siendo dos de ellos las propiedades Text y Tag. Son las que se suelen vincular con datos. Para conseguir el mismo efecto que la sentencia anterior, por tanto, seleccionaríamos el control TextBox1 y haríamos lo que puede verse en la figura 14.1, desplegar la lista adjunta a la propiedad Text y seleccionar la columna de origen.

En caso de que la propiedad que deseemos vincular no sea una de las que nos ofrece por defecto la propiedad DataBindings, podemos pulsar el botón asociado a la entrada (Avanzado) para abrir la ventana Enlace de datos avanzado (véase figura 14.2). En ella se enumeran todas las propiedades del control que pueden ser vinculadas, pudiendo seleccionar distintos orígenes para cada una de ellas.

Los controles ASP.NET, a pesar de que en la fase de diseño parecen idénticos a sus equivalentes para Windows, cuentan con un mecanismo de enlace a datos diferente. Si inserta un TextBox en un formulario Web y, en la ventana Propiedades, accede a su propiedad DataBindings, se encontrará con una ventana como la que muestra la figura 14.3. Su funcionamiento es similar a lo explicado en el párrafo anterior, pudiendo seleccionar para cada propiedad una vinculación de acceso a datos.

Si tras establecer el vínculo examina el código HTML del documento, sin embargo, verá que TextBox1 en realidad no dispone de la propiedad DataBindings, sino que se ha asignado a su propiedad Text lo que se conoce como una *expresión de vinculación*.

En este caso dicha expresión es la siguiente:

```
Text='<%# DataBinder.Eval(DsLibros1,  
    "Tables[Libros].DefaultView.[0].Titulo") %>
```

Las expresiones de enlace se delimitan entre los caracteres <%# y %>, marcas que provocan la resolución en el servidor y no en el cliente. DataBinder es una clase con la que cuentan las páginas ASP.NET, cuyo objetivo es facilitar el enlace con orígenes de datos típicos, como conjuntos de datos y vistas. Lo único que hay que hacer es facilitar a su método Eval() la referencia al origen, en este caso un DataSet, y una cadena para seleccionar de él una cierta columna.

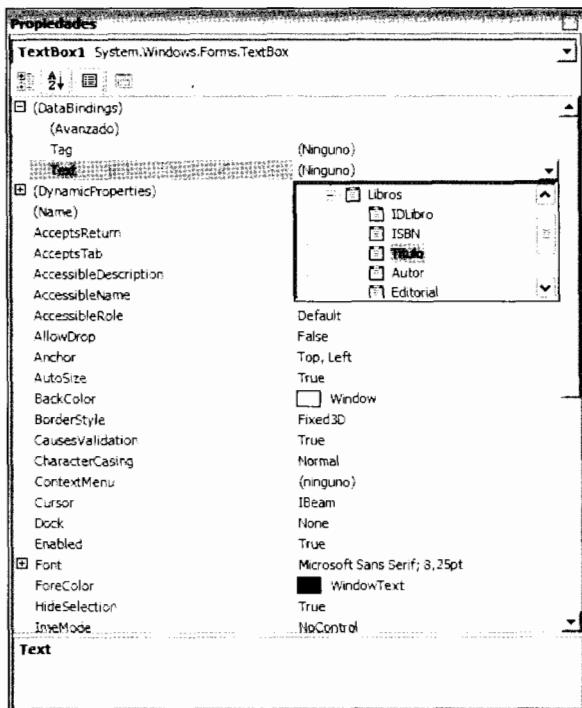


Figura 14.1. Edición visual de la propiedad DataBindings para vincular la propiedad Text de un TextBox

En lugar de utilizar DataBinder, que es lo que hace por defecto el editor de la propiedad DataBindings, podemos introducir una expresión de vinculación personalizada haciendo referencia directa al conjunto de datos que hayamos creado previamente. En la figura 14.4, por ejemplo, puede observar cómo se ha activado la opción **Expresión de enlace personalizada** e introducido la expresión `dsLibros1.Libros(0).Titulo`, vinculando así el TextBox con el título de la primera fila de la tabla de libros.

Independientemente del método que empleemos para establecer la vinculación, para que ésta sea efectiva es necesario siempre llamar al método `DataBind()`, que se encarga de resolver las expresiones una vez que el componente de origen ya dispone de los datos a utilizar. Si son múltiples los controles vinculados, en lugar de llamar al método `DataBind()` de cada uno de ellos, que sería válido pero rei-

terativo, puede invocarse al mismo método de la página, encargándose ésta de todos los controles hijo que contiene.

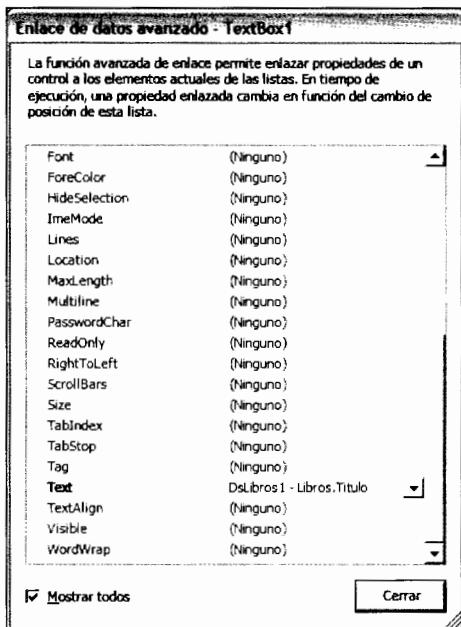


Figura 14.2. Vinculación de múltiples propiedades del mismo control

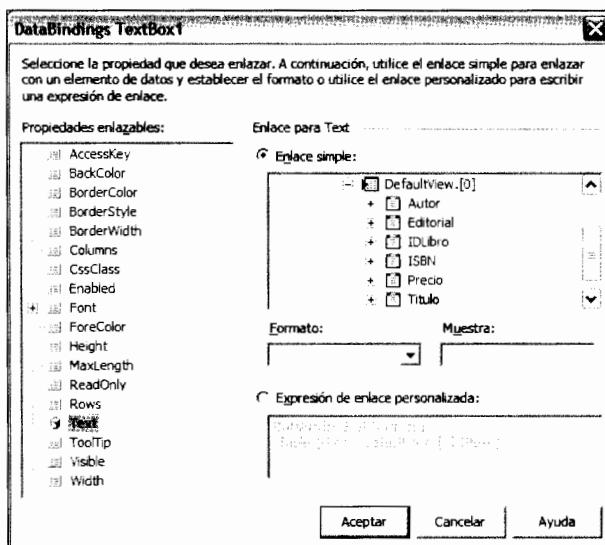


Figura 14.3. Ventana de la propiedad DataBindings correspondiente a un TextBox de ASP.NET

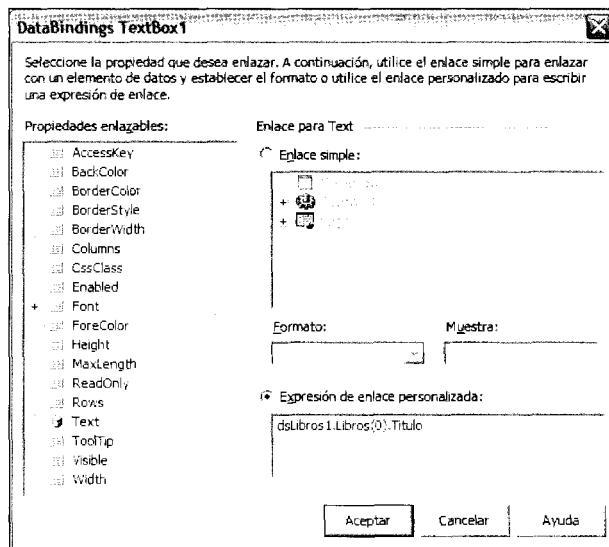


Figura 14.4. Introducimos una expresión personalizada de vinculación a datos

Vinculación con componentes más de una columna

Son varios los componentes capaces de mostrar el valor que una cierta columna tiene en múltiples filas, siendo los más habituales `ListBox` y `ComboBox`. Qualquiera de estos dos componentes puede ser usado para ofrecer al usuario una lista de valores a elegir, tomados de una columna de una tabla, asociando dichos elementos con el valor que tenga, en la misma fila, cualquier otra columna. El usuario podría, por ejemplo, elegir el nombre de una editorial de un `ComboBox` y el programa obtener el identificador de dicha editorial.

Aunque estos controles también disponen de la propiedad `DataBindings`, cuentan, además, con propiedades específicas para el comportamiento específico que tienen. En el caso de los controles para formularios Windows dichas propiedades son:

- **DataSource:** Contendrá una referencia a la tabla, vista u objeto que vaya a actuar como origen de datos.
- **DisplayMember:** Alojará el nombre de la columna cuyo contenido se mostrará en la lista.
- **ValueMember:** Indica la columna cuyo contenido se obtendrá como valor asociado a cada uno de los elementos de la lista.

Suponiendo que deseásemos mostrar en un control `ListBox` los títulos de todos los libros y que, al elegir cualquiera de ellos, se obtuviese el identificador del seleccionado, asignaríamos entonces el valor `Titulo` a `DisplayMember` e `IDLibro` a

bro a ValueMember, mientras que DataSource podría ser DsLibros1.Libros, suponiendo que tenemos un DataSet llamado DsLibros1 conteniendo la tabla Libros. En caso de que usemos los controles Web equivalentes, las propiedades a utilizar serán las cuatro siguientes:

- DataSource: La referencia al conjunto de datos u objeto que contiene la información.
- DataMember: En caso de que el objeto anterior sea un DataSet, con esta propiedad seleccionaríamos una de las tablas disponibles.
- DataTextField: Equivalente a DisplayMember, conteniendo el nombre de la columna que se mostrará en el control.
- DataValueField: Equivalente a ValueMember, indicando la columna de donde se tomará el valor a asociar a cada elemento.

Puede hacer una prueba simple insertando un ListBox en una página Web ASP.NET, junto con un TextBox, y enlazando la lista con la columna Titulo de la tabla Libros. Dé el valor True a la propiedad AutoPostBack del ListBox y, tras hacer doble clic sobre él, introduzca la siguiente sentencia:

```
TextBox1.Text = ListBox1.SelectedItem.Value
```

Al ejecutar el programa debería obtener un resultado similar al de la figura 14.5, donde ve cómo al seleccionar un título su identificador aparece en la caja de texto. Esto es así porque hemos dado el valor IDLibro a la propiedad DataValueField.

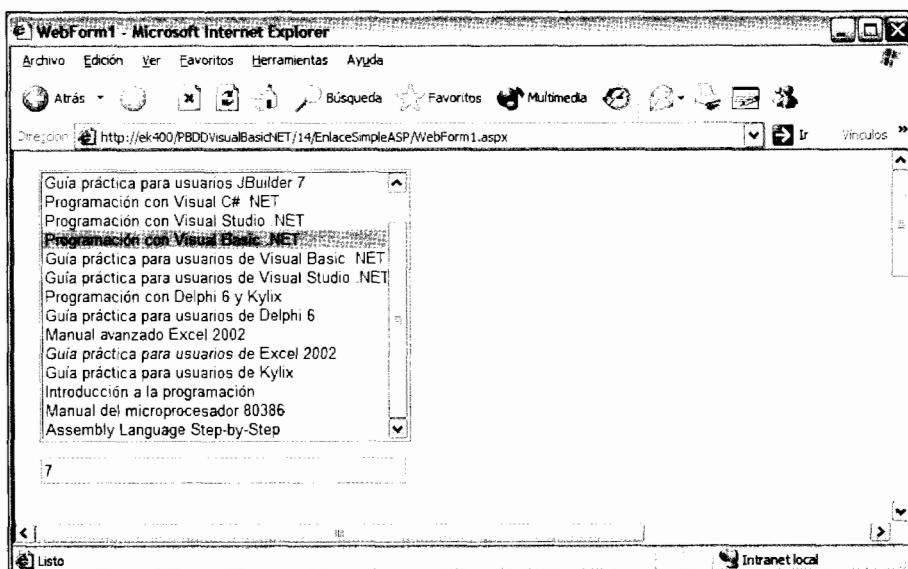


Figura 14.5. Vinculación de una lista ASP.NET con los valores de una columna

Nota

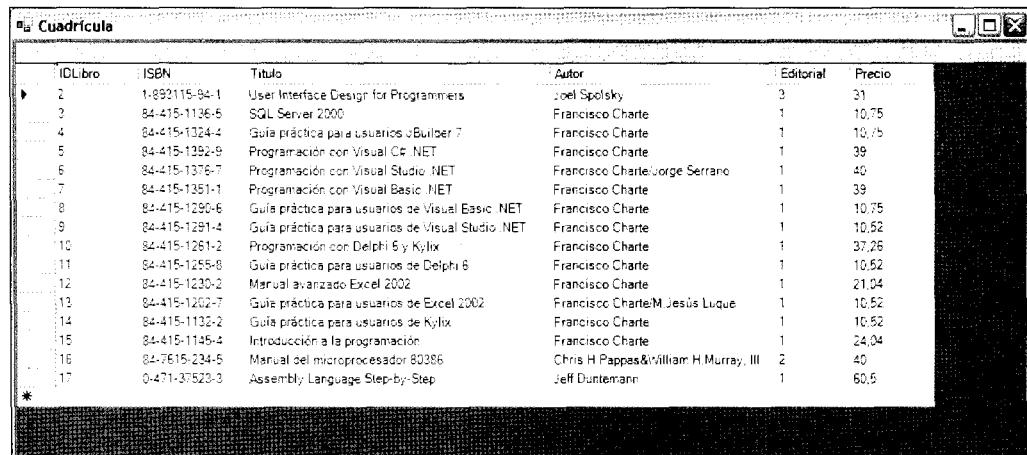
Este mismo ejemplo puede aplicarse con un **ListBox** en un formulario Windows, tan sólo tiene que cambiar las propiedades de vinculación de la lista, manteniendo el resto igual.

Vinculación con múltiples filas y columnas

La tercera categoría de componentes está compuesta exclusivamente por el control **DataGrid** que, a diferencia de los anteriores, tiene la capacidad de mostrar múltiples filas y columnas, pudiendo facilitar el acceso a tablas completas de información. Estos controles, por tanto, no se enlazan con una columna en particular, sino con una vista o una tabla completa.

Independientemente de que usemos el **DataGrid** de formularios Windows o formularios Web, las dos propiedades de enlaces a emplear son las mismas: **DataSource** y **DataMember**. La primera hará referencia al origen de datos y, en caso de que éste sea un **DataSet**, la segunda seleccionará entonces una de sus tablas o vistas.

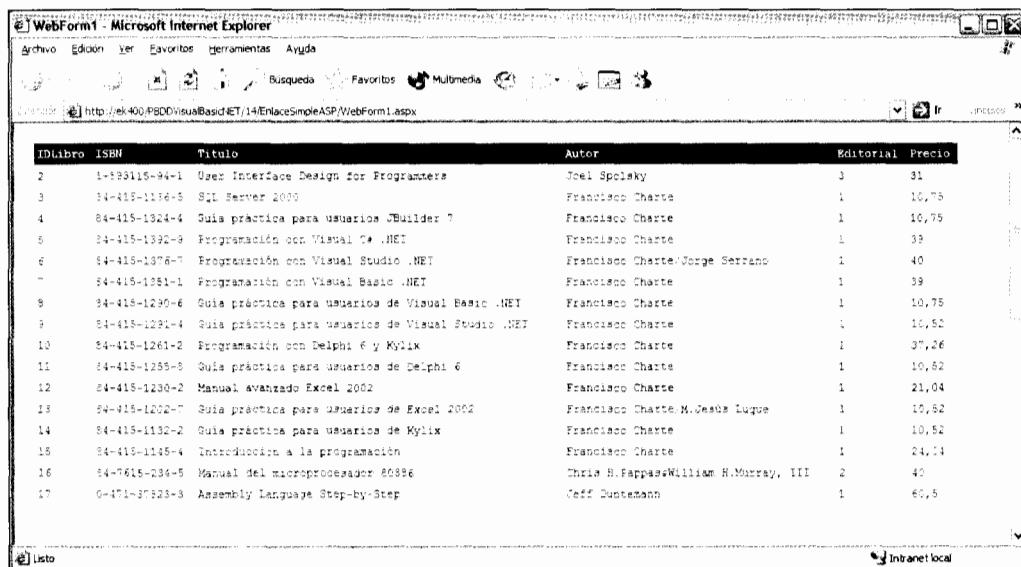
Aunque el comportamiento de ambos controles **DataGrid** difiere bastante, visualmente hablando, como se aprecia en las figuras 14.6 y 14.7, son muy similares, mostrando, como es lógico, la misma información al vincularse al mismo origen de datos.



IDLibro	ISBN	Título	Autor	Editorial	Precio
2	1-892115-84-1	User Interface Design for Programmers	Joel Spolsky	3	31
3	84-415-1136-5	SQL Server 2000	Francisco Charle	1	10,75
4	84-415-1324-4	Guía práctica para usuarios JBuilder 7	Francisco Charle	1	10,75
5	84-415-1392-9	Programación con Visual C# .NET	Francisco Charle	1	39
6	84-415-1376-7	Programación con Visual Studio .NET	Francisco Charle Jorge Serrano	1	40
7	84-415-1351-1	Programación con Visual Basic .NET	Francisco Charle	1	39
8	84-415-1290-6	Guía práctica para usuarios de Visual Basic .NET	Francisco Charle	1	10,75
9	84-415-1291-4	Guía práctica para usuarios de Visual Studio .NET	Francisco Charle	1	10,52
10	84-415-1261-2	Programación con Delphi 6 y Kylix	Francisco Charle	1	37,25
11	84-415-1255-8	Guía práctica para usuarios de Delphi 6	Francisco Charle	1	10,52
12	84-415-1230-2	Manual avanzado Excel 2002	Francisco Charle	1	21,04
13	84-415-1202-7	Guía práctica para usuarios de Excel 2002	Francisco Charle M. Jesús Luque	1	10,52
14	84-415-1132-2	Guía práctica para usuarios de Kylix	Francisco Charle	1	10,52
15	84-415-1145-4	Introducción a la programación	Francisco Charle	1	24,04
16	84-7616-234-5	Manual del microprocesador 80386	Chris H. Pappas & William H. Murray, III	2	40
17	0-471-37523-3	Assembly Language Step-by-Step	Jeff Duntemann	1	60,5

Figura 14.6. El control **DataGrid** en un formulario Windows

El **DataGrid** de ASP.NET cuenta, además, con la propiedad **DataKeyField**, a la que podemos asignar el nombre de la columna que actúa como clave de la tabla o vista a fin de identificar de manera inequívoca cada fila de datos.



The screenshot shows a Microsoft Internet Explorer window with the title bar "WebForm1 - Microsoft Internet Explorer". The menu bar includes "Archivo", "Edición", "Ver", "Favoritos", "Herramientas", and "Ayuda". The toolbar includes "Nuevo", "Abrir", "Guardar", "Imprimir", "Recortar", "Copiar", "Pegar", "Búsqueda", "Favoritos", "Multimedia", and "Ayuda". The address bar shows the URL "http://ek400-PBDDVisualBasic/ET/14/EnlaceSimpleASP/WebForm1.aspx". The main content area displays a DataGrid control with the following data:

IDLibro	ISBN	Título	Autor	Editorial	Precio
2	1-5933115-9-1	User Interface Design for Programmers	Joel Spolsky	3	31
3	14-415-1136-5	SQL Server 2000	Francisco Charte	1	16,75
4	84-415-1324-4	Guía práctica para usuarios JBuilder 7	Francisco Charte	1	16,75
5	84-415-1392-9	Programación con Visual C# .NET	Francisco Charte	1	39
6	84-415-1376-7	Programación con Visual Studio .NET	Francisco Charte/Jorge Serrano	1	40
7	84-415-1281-1	Programación con Visual Basic .NET	Francisco Charte	1	39
8	84-415-1230-6	Guía práctica para usuarios de Visual Basic .NET	Francisco Charte	1	16,75
9	84-415-1291-4	Guía práctica para usuarios de Visual Studio .NET	Francisco Charte	1	16,51
10	84-415-1261-2	Programación con Delphi 6 y Kylix	Francisco Charte	1	37,06
11	84-415-1255-8	Guía práctica para usuarios de Delphi 6	Francisco Charte	1	16,62
12	84-415-1230-2	Manual avanzado Excel 2002	Francisco Charte	1	21,04
13	84-415-1202-7	Guía práctica para usuarios de Excel 2002	Francisco Charte/M.Jesús Luque	1	16,62
14	84-415-1132-2	Guía práctica para usuarios de Kylix	Francisco Charte	1	16,52
15	84-415-1135-4	Introducción a la programación	Francisco Charte	1	24,04
16	84-7615-134-5	Manual del microprocesador 60386	Chris H.Fappas/William H.Murray, III	2	40
17	0-471-37523-3	Assembly Language Step-by-Step	Jeff Dunemann	1	60,5

Figura 14.7. El control DataGrid en un formulario Web

Nota

Mientras que el DataGrid de Windows tiene capacidades de navegación y edición intrínsecas, desde el momento en que se inserta en un formulario, el equivalente para formularios Web se comporta como una tabla HTML estática, aunque es posible personalizarlo para incluir enlaces y botones que hagan posible la manipulación de la información.

Enlace a datos en formularios Windows

Ya sabemos cómo enlazar cualquier componente Windows con un origen de datos, usando para ello la propiedad DataBindings, DataSource y DataMember, según los casos. Los controles que se vinculan con una sola columna, mediante DataBindings, no tienen noción de la fila en la que se encuentran en el conjunto de datos, puesto que sólo conocen el valor de una columna en una determinada fila. Es el propio formulario el encargado de crear los elementos necesarios para asegurar la sincronización entre los distintos controles que puedan estar vinculados con datos, impidiendo que cada uno de ellos muestre el valor de una columna en filas distintas.

Todos los formularios Windows cuentan con un objeto BindingContext, al que podemos acceder mediante la propiedad del mismo nombre, cuya finalidad es

mantener una lista de objetos, uno por cada origen de datos que exista, que se encargarán de la sincronización de todos los componentes vinculados a un mismo origen. Los elementos de BindingContext son de tipo BindingManagerBase, una clase abstracta que cuenta con dos derivadas: CurrencyManager y PropertyManager.

La clase PropertyManager se usa para mantener la asociación entre la propiedad de un control y un origen cuando no hay necesidad de saber qué fila de ese origen es la actual, caso éste en el que se emplearía un CurrencyManager. Siempre que el origen sea un DataTable, DataView o similar, el objeto encargado de mantener la asociación será un CurrencyManager.

Nota

El objeto BindingContext mantiene una lista de los BindingManagerBase, o derivados, existentes en el formulario, mientras que cada BindingManagerBase controla todos los objetos Binding de un mismo origen de datos.

Posición actual en una lista de datos

Como ya sabe, lo ha visto en los capítulos de la segunda parte, una vez que se ha recuperado información en un DataTable éste contiene todas las filas de datos, a las que puede acceder como si de un arreglo se tratase. No existe una propiedad que indique qué fila es la actual, ya que ese concepto, muy habitual en otros sistemas de acceso a datos, no existe en ADO.NET.

Cuando ese DataTable o DataView se enlaza con controles como TextBox, CheckBox y RadioButton, sin embargo, existe la necesidad de saber cuál de las filas de esa lista de datos tiene que mostrarse en los controles, es decir, cuál de las filas es la posición actual dentro de la lista. Como ADO.NET no ofrece esa funcionalidad, lo que hacen los formularios Windows es crear un CurrencyManager para cada origen que mantiene una lista de datos, como los mencionados DataTable y DataView.

Dado que en un mismo formulario podrían existir múltiples CurrencyManager, bien porque existan varios orígenes de datos o un mismo origen con enlaces no homogéneos a varios controles, para mantenerlos todos ellos se emplea un objeto BindingContext.

Conociendo el nombre del origen de datos, es fácil recuperar el CurrencyManager asociado:

```
BindingContext(DsLibros1, "Libros")
```

Con esta expresión obtendríamos el CurrencyManager asociado con la tabla Libros del DataSet llamado DsLibros1. ¿Por qué nos interesa obtener dicho objeto? Principalmente porque es él quien mantiene la posición actual en la lista de

datos, contando con propiedades que nos permiten conocer y modificar dicha posición. La mayor parte de los miembros de esta clase son heredados de `BindingManagerBase`, si bien `CurrencyManager` sustituye algunas implementaciones por otras más específicas.

En cuanto a posición se refiere, las dos propiedades de mayor interés que tiene esta clase son `Count` y `Position`. Como puede suponer, la primera indica el número de filas existentes en el conjunto, mientras que la segunda es un índice con base 0, por lo que el valor máximo será el indicado por `Count` menos 1.

Conociendo tan sólo estas dos propiedades, es fácil añadir a un formulario los controles de navegación clásicos para que el usuario pueda ir fila a fila por el conjunto de datos.

Control de la vinculación

Todos los derivados de `BindingManagerBase` disponen de los tres métodos `Refresh()`, `SuspendBinding()` y `ResumeBinding()`, con los cuales podemos establecer un cierto control sobre la vinculación, y con los eventos `CurrentChanged` y `PositionChanged`. Cada clase derivada facilita una implementación específica de estos miembros.

El método `Refresh()` provoca la actualización de todos los controles enlazados con el origen al que representa el objeto `BindingManagerBase`, mientras que `SuspendBinding()` y `ResumeBinding()` desactivan y reactivan la vinculación. Esto tiene sentido cuando van a efectuarse tareas de edición sobre los datos que pudieran provocar una violación de las restricciones definidas en el origen, impiéndole al usuario trabajar de forma cómoda. Al suspender la vinculación los contenidos de los controles pueden modificarse sin limitaciones, efectuándose las comprobaciones en el momento en que el vínculo vuelva a establecerse.

En cuanto a los eventos `CurrentChanged` y `PositionChanged`, se producen cuando uno de los valores vinculados ha cambiado, mientras que el segundo notifica una modificación de la propiedad `Position`.

Componentes enlazables

La propiedad `DataBindings` que se citaba en un punto previo, al enlazar un `TextBox` con una columna de una tabla, está definida en la clase `Control` de `System.Windows.Forms`. Esto significa que es una propiedad heredada por todos los controles Windows y que, por tanto, podemos virtualmente vincular cualquier propiedad de cualquier control con una columna de un origen de datos.

Existe, no obstante, un conjunto de controles que, por su naturaleza, se prestan más a la vinculación. Algunos de ellos son el propio `TextBox`, ya usado previamente; todos los derivados de `ButtonBase`, tales como `CheckBox`, `RadioButton` y `Button`; los controles `PictureBox`, para la visualización de imágenes, `TrackBar` y `ScrollBar` o `DateTimePicker`.

Controles como `ListBox`, `ComboBox` y `DataGridView` también cuentan con la propiedad `DataBindings`, al fin y al cabo son clases derivadas de `Control`, pero cuentan con propiedades más específicas ya explicadas en los apartados del primer punto de este capítulo.

Enlace a datos en formularios Web

La vinculación a datos de las aplicaciones Web es muy distinta a la de las aplicaciones Windows, al ejecutarse en un entorno que, por naturaleza, es *desconectado*, es decir, los clientes no mantienen una conexión continua con el servidor. Los formularios ASP.NET no disponen de la propiedad `BindingContext` ni los objetos derivados de `BindingManagerBase`, y los controles no facilitan la navegación, edición y actualización directa de los datos como sí lo hacen en los formularios Windows.

En muchas ocasiones la vinculación a datos en una página Web se emplea para mostrar información sólo de lectura, por ejemplo generando un informe. En otros, por el contrario, sí es precisa la edición, para lo cual será necesario controlar parte de las acciones mediante código.

Nota

Debe tenerse en cuenta que los clientes que vayan a usar una aplicación Web en la que se utiliza vinculación a datos, tienen que instalar en su sistema los MDAC. Puede añadir a los propios formularios ASP.NET la información necesaria para que los obtenga e instale.

A cambio, en ASP.NET existen más componentes capaces de presentar una lista de datos que en los formularios Windows. Además del conocido `DataGridView` y el control `ListBox`, también tenemos a nuestra disposición los controles `CheckBoxList`, `RadioButtonList`, `DownList` y `Repeater`, siendo estos dos últimos especialmente interesantes. Con `DownList` es posible crear listas de datos basadas en una plantilla de apariencia definida por nosotros, mientras que `Repeater` es un contenedor en el que es posible insertar otros controles que serán repetidos por la página Web. Tanto `DownList` como `Repeater` se basan en el uso de plantillas que hay que definir directamente en el módulo HTML, con etiquetas tales como `<ItemTemplate>` o `<HeaderTemplate>`.

Vínculos sólo de lectura

En caso de que vaya a diseñar una página ASP.NET con vinculación a datos, por ejemplo para mostrar las editoriales o libros existentes en nuestra base de da-

tos de ejemplo, pero sin necesitar capacidades de edición, inserción ni borrado, lo más apropiado es usar un vínculo sólo de lectura mediante un **DataReader**.

Los vínculos sólo de lectura emplean menos recursos, ya que un **DataReader** abre la conexión y recupera los datos mediante un cursor unidireccional y sólo de lectura, sin crear estructuras en memoria como sí hace un **DataSet**.

Suponga que quiere mostrar en una página Web una tabla con una lista de los títulos existentes en nuestra base de datos, utilizando para ello un componente **DataList** y un lector de datos. Los pasos, básicamente, serían los siguientes:

- Definición de la conexión y el comando para recuperar la columna **Titulo** de la tabla **Libros**.
- Inserción en la página ASP.NET de un componente **DataList**, seleccionando alguno de los estilos predefinidos para él.
- Edición de la plantilla correspondiente al cuerpo, insertando en él un componente que se enlazaría con la columna **Titulo**.
- Codificación, en el evento **Load**, de la apertura de la conexión y obtención del **DataReader**, asignándolo a la propiedad **DataSource** del **DataList**.

En la figura 14.8 puede ver cómo se ha insertado un control **Label** en el **ItemTemplate** del componente **DataList**, y cómo se utiliza la propiedad **DataBindings** de la etiqueta de texto para vincularla con la columna **Titulo**. Se asume, por supuesto, que hemos insertado y configurado la conexión y el comando.

A continuación, tras hacer doble clic sobre el fondo de la página, introduciríamos el código siguiente:

```
SqlConnection1.Open() ' Abrimos la conexión  
' Conectamos el Datalist con el DataReader  
DataList1.DataSource = SqlCommand1.ExecuteReader()  
.DataBind() ' y vinculamos  
SqlConnection1.Close() ' cerramos la conexión
```

Con él enlazamos el **SqlDataReader** devuelto por el método **ExecuteReader()** con el control **DataList**. Éste lee todos los títulos y genera una tabla como la que puede verse en la figura 14.9. No hemos necesitado un adaptador de datos, generar un **DataSet** ni nada parecido, realizando el trabajo de manera más eficiente y con menor uso de recursos.

Navegación con componentes simples

Si a pesar de la existencia de componentes como **DataList** y **Repeater**, que pueden enlazarse a una lista de datos, decide emplear componentes con vinculación simple, seguramente precisará un medio de navegación por la lista de datos, de tal forma que los controles muestren en cada momento la información de la fila actual.

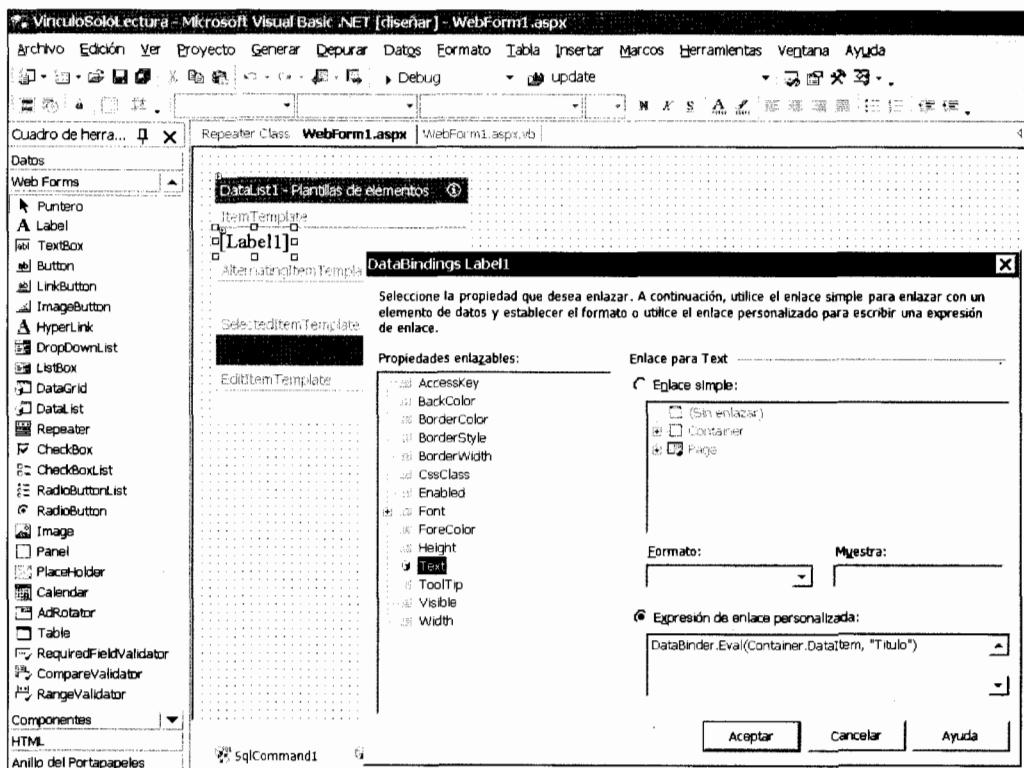


Figura 14.8. Diseño de la plantilla para el cuerpo del DataList

La mala noticia es que los formularios Web no mantienen objetos equivalentes a los CurrencyManager o BindingContext de los formularios Windows, por lo que el mecanismo para mantener la posición actual y para sincronizar los controles vinculados queda completamente en nuestras manos. La buena noticia es que codificar esa funcionalidad no resulta excesivamente complejo.

Lo primero que necesitamos es saber cuántas filas existen en la tabla a la que van a vincularse los controles, dato que podemos obtener fácilmente de la propiedad Count de la tabla.

También necesitamos un medio para almacenar ese valor, así como la posición actual, de forma que estén accesibles durante el tiempo que el usuario esté trabajando sobre la página. La solución más fácil es usar la propiedad Session para crear sendas variables y guardar dichos valores.

En segundo lugar, tenemos que asignar de alguna forma los valores de las columnas de la tabla a los controles que haya en la página. Si asignamos a la propiedad Text de un TextBox la expresión dsLibros1.Libros(0).Titulo, por ejemplo, el control mostraría siempre el título de la primera fila, independientemente del valor que tuviera la variable de posición creada anteriormente. La vinculación entre tabla y controles, por tanto, debe efectuarse *a mano*. ¿Cuándo realizar

esas asignaciones? En el momento en que el control reciba el evento DataBinding, generado cada vez que se llama al método DataBind() del control o de la página donde está insertado.

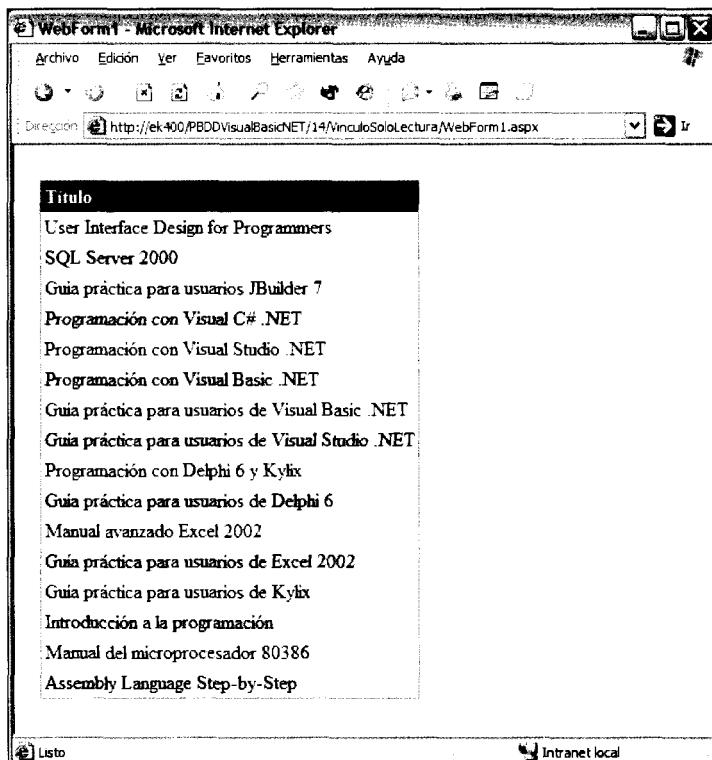


Figura 14.9. Tabla generada por el componente DataList

Puede realizar una prueba dando los pasos siguientes:

- Inicie una nueva aplicación Web ASP.NET.
- Inserte desde el Explorador de servidores la tabla Libros de uno de los orígenes que tenga disponibles, por ejemplo SQL Server.
- Genere el DataSet a partir del adaptador de datos.
- Inserte en el formulario dos controles TextBox y dos Button.
- Haga doble clic sobre el formulario para abrir el método correspondiente al evento Load, que quedaría así:

```
' Al abrirse la página
Private Sub Page_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
```

```

' Llamamos al conjunto de datos
SqlDataAdapter1.Fill(DsLibros1, "Libros")
' si no es la pulsación de uno de los botones
If Not IsPostBack Then
    ' vacaremos la posición inicial y obtenemos
    ' el número de filas
    Session("Count") = DsLibros1.Libros.Count
    Session("Position") = 0
    DataBind()
End If
End Sub

```

- El código asociado a los dos botones que ha insertado será el siguiente:

```

' Al pulsar el botón de avance
Private Sub Button1_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles Button1.Click
    Dim Position As Integer = Session("Position")
    ' Si no estamos en la última fila
    If Position < Session("Count") - 1 Then
        Position += 1 ' avanzamos
        Session("Position") = Position
        DataBind() ' y actualizamos los controles
    End If
End Sub

' Al pulsar el botón de retroceso
Private Sub Button2_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles Button2.Click
    Dim Position As Integer = Session("Position")
    ' Si no estamos en la primera fila
    If Position > 0 Then
        Position -= 1 ' retrocedemos
        Session("Position") = Position
        DataBind() ' y actualizamos los controles
    End If
End Sub

```

- Por último, abra el método correspondiente al evento DataBinding de uno de los TextBox, dejándolo como se muestra a continuación:

```

' Este método se ejecuta cada vez que es necesario
' sincronizar los controles
Private Sub TextBox1_DataBinding(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles TextBox1.DataBinding
    ' las asignamos contenido obteniéndolo del DataSet
    TextBox1.Text = DsLibros1.Libros(Session("Position")).Titulo
    TextBox2.Text = DsLibros1.Libros(Session("Position")).Autor
End Sub

```

Observe que la inicialización de las variables Position y Count se efectúa sólo la primera vez que se abre la página, y no cada vez que se pulsa uno de los botones. Éstos recuperan el valor actual de Position y lo actualizan, invocando a continuación al método DataBind() de la página que, a su vez, provoca el evento

.DataBind de los TextBox. En la figura 14.10 puede verse la página mostrando los datos de una fila.

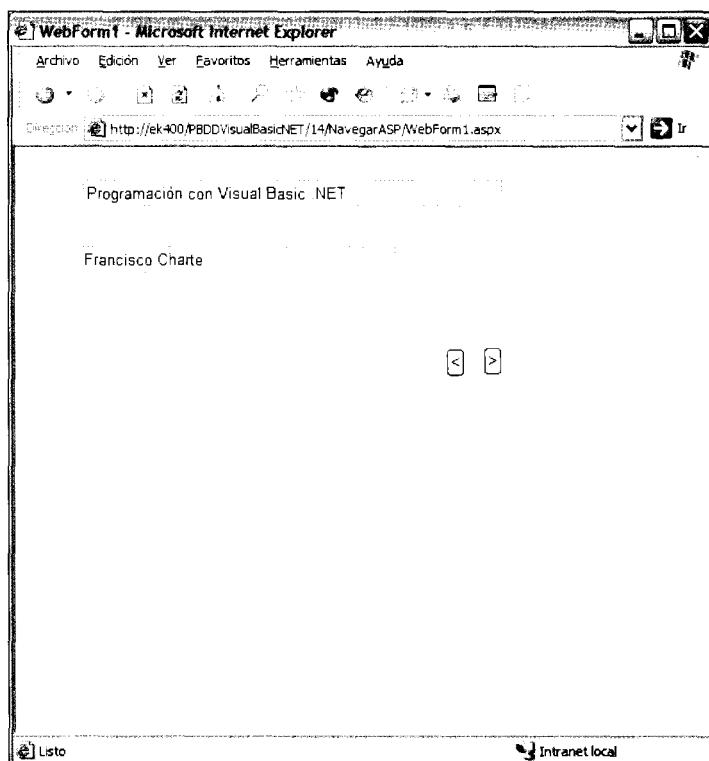


Figura 14.10. La página con las dos cajas de texto y los botones que permiten avanzar y retroceder entre filas de datos

Actualización del origen

Los controles ASP.NET tampoco disponen de un sistema de edición y actualización automática de los datos. Los cambios en la información mostrada por los controles deben detectarse, facilitando, en caso de ser necesario, los elementos para efectuar dicha edición. El componente DataGrid, por ejemplo, puede mostrar una serie de enlaces como columnas de la rejilla, haciendo posible la edición y eliminación de datos.

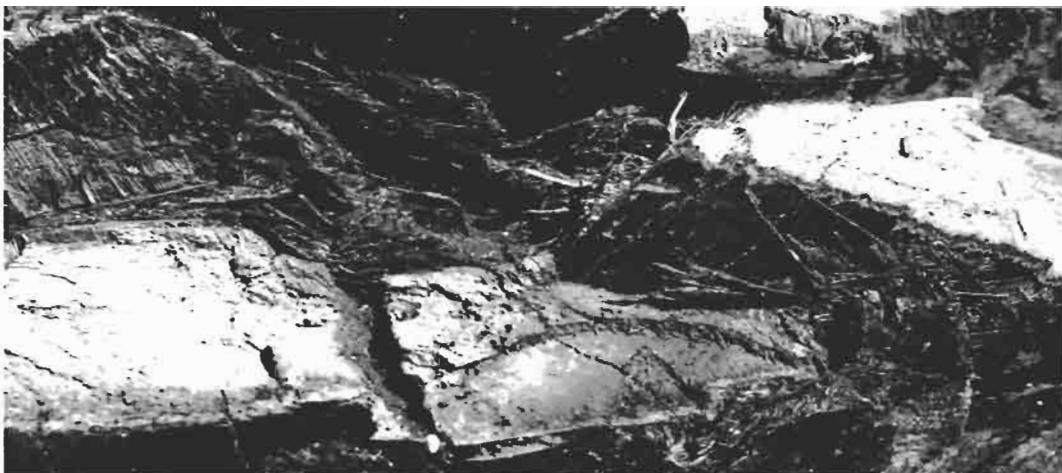
En cuanto a la actualización de los datos en el origen, necesitará disponer en el formulario algún botón o enlace que, al ser pulsado, ejecute el código de sincronización de los cambios con el servidor del que procede la información. Esto significa, por regla general, o utilizar un objeto Command con la sentencia SQL a ejecutar o, si estamos usando un adaptador de datos y un DataSet, servirnos del mecanismo habitual de actualización que ya conocemos.

Resumen

Al finalizar este capítulo ya conoce los mecanismos de vinculación que hay disponibles entre un origen de datos, típicamente un `DataSet`, `DataView` o similar, y los controles de interfaz de usuario habituales en formularios Windows y Web.

Como sería de esperar, la funcionalidad de los formularios y controles de Windows es mayor ya que se ejecutan sobre Windows como aplicaciones nativas, pudiendo asumirse la existencia de ciertos servicios. Las aplicaciones ASP.NET, por el contrario, requieren más pasos para conseguir el mismo resultado, si bien su flexibilidad y posibilidades están por encima de la mayoría de tecnologías actuales para el desarrollo de aplicaciones Web.

En este capítulo no se han explicado de manera individual los controles Windows y Web que hay disponibles. Ésta es una información de referencia que, como siempre, puede encontrar en la ayuda electrónica que acompaña a Visual Basic .NET. En su lugar, ha obtenido un conocimiento general de cómo esos componentes pueden vincularse a un origen de datos.



15

Formularios de datos

Usando los controles de las secciones **Web Forms** o **Windows Forms**, según el tipo de aplicación que se trate, pueden diseñarse formularios que permitan al usuario ver los datos, navegar por ellos, editarlos, añadir y eliminar datos. Para ello necesitará combinar gran parte de los conocimientos que ha adquirido desde el principio de este libro, desde la definición de la conexión con el origen de datos hasta la vinculación de los controles, pasando por la creación de adaptadores y conjuntos de datos o la codificación de las sentencias necesarias para actualizar el origen.

Visual Studio .NET dispone de sendos asistentes que facilitan el diseño de estos formularios, ahorrándonos gran parte del trabajo que, de otra forma, tendríamos que efectuar manualmente. El objetivo de este capítulo es mostrarle cómo servirse de estos asistentes para diseñar sus propios formularios, usando, como en los demás capítulos, alguna de las bases de datos creadas en el tercer capítulo.

El asistente para formularios Windows

Comenzaremos analizando el asistente de formularios de datos para Windows, creando una aplicación Windows típica, eliminando el formulario añadido por defecto e iniciando este asistente. Para ello seleccione la opción **Proyecto>Agregar nuevo elemento**, o pulse el botón equivalente, y seleccione del cuadro de diálogo que aparece el elemento **Asistente para formularios de datos** (véase figura 15.1). Introduzca en la parte inferior el nombre que dará al formulario y pulse el botón **Abrir**.

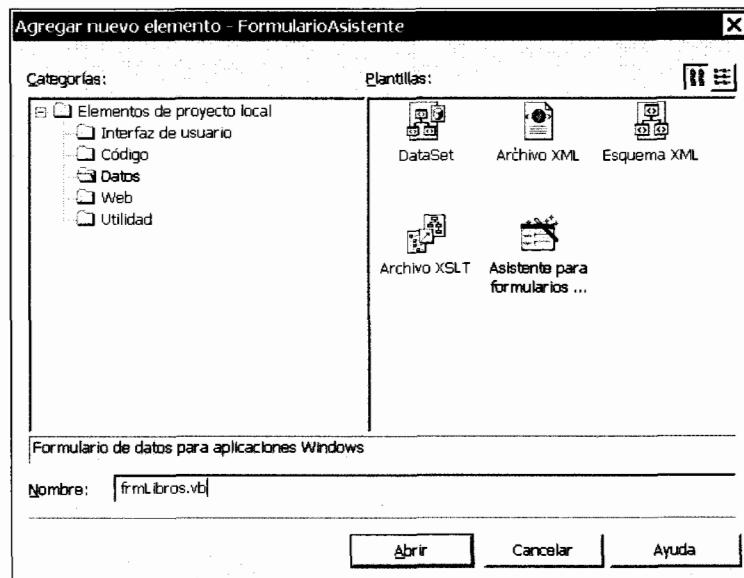


Figura 15.1. Iniciamos el asistente para creación de un formulario de datos Windows

Nuestro objetivo es crear un formulario con una relación maestro/detalle entre las tablas **Editoriales** y **Libros**, usando para ello dos controles **DataGrid**, de tal forma que, al seleccionar una editorial en el primero, aparezcan automáticamente en la rejilla de detalle los títulos que correspondan. No necesitamos definir una conexión, ni tomar las tablas desde el **Explorador de servidores** y arrastrarlas sobre un formulario, ya que todo ese trabajo lo efectuará el asistente.

La primera página del asistente es tan sólo informativa, indicándonos cuál es su finalidad, bastando con pulsar el botón **Siguiente >** para dar el primer paso.

Selección del DataSet

En nuestro caso partimos de un proyecto inicialmente vacío, pero igualmente podríamos haber invocado al asistente tras definir un **DataSet** a partir del origen de datos que nos interesase. El primer paso del asistente nos permite, tal como se aprecia en la figura 15.2, tanto crear un nuevo conjunto de datos como usar uno que ya existiese en el proyecto.

Deje elegida la primera opción, en realidad no puede activar la segunda ya que el asistente ha detectado que no hay ningún **DataSet** existente disponible, e introduzca el nombre que desea dar al conjunto de datos, por ejemplo **dsLibros**.

En este conjunto de datos, tal como veremos después, se introducirán las tablas **Editoriales** y **Libros**, así como un objeto **DataRelation** para relacionarlas. Todo ese trabajo, que en los capítulos de la segunda parte efectuábamos mediante código, será ejecutado por el asistente.

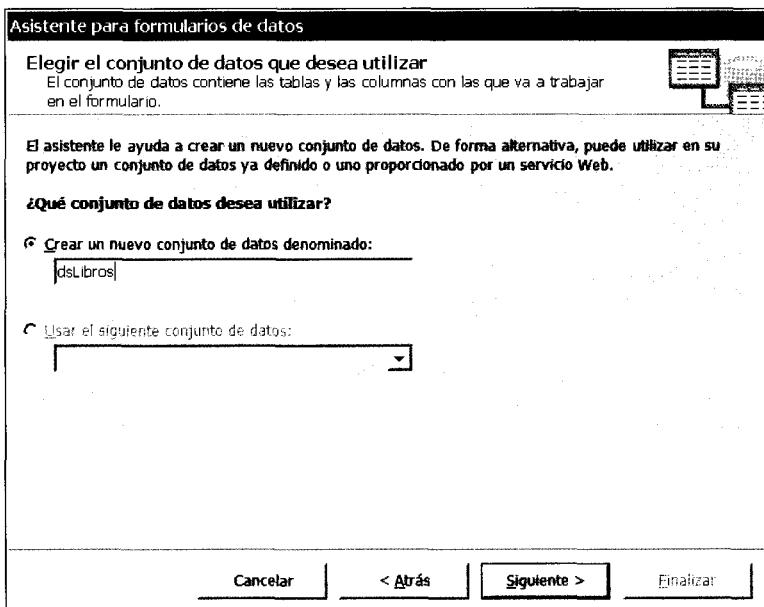


Figura 15.2. Seleccionamos crear un nuevo conjunto de datos

Definición de la conexión

El DataSet tiene que recuperar la información de un origen de datos, para lo cual es preciso facilitar una conexión. Ésta puede encontrarse predefinida, en el Explorador de servidores, o bien crearse a propósito en ese mismo momento. Es la tarea del paso siguiente del asistente, cuya ventana se muestra en la figura 15.3.

Si tenemos una conexión definida que nos sea útil, no tenemos más que desplegar la lista que aparece a la izquierda para elegirla. De no ser ése el caso, pulsaremos el botón **Nueva conexión** y definiríamos la nueva conexión con el cuadro de diálogo que ya conocemos de capítulos previos.

En este caso hemos elegido la conexión que teníamos definida para acceder a la base de datos SQL Server que se encuentra en el servidor Inspiron. En su caso, como es lógico, la conexión será otra diferente dependiendo de dónde esté ejecutándose SQL Server o bien MSDE. También puede optar por conectar con un origen de datos distinto, cualquiera de los que creó en el tercer capítulo.

Selección de los elementos de origen

Para crear nuestro formulario podemos partir de tablas, vistas, procedimientos almacenados que devuelven resultados, etc. En este paso del asistente el cuadro de diálogo está dividido en dos paneles. El de la izquierda nos muestra las tablas, vistas y, en general, todos los elementos que han podido encontrarse a través de la

conexión definida en el punto previo. En el panel de la derecha, inicialmente vacío, irán apareciendo los elementos que seleccionemos, usando para ellos los botones que hay en la parte central.

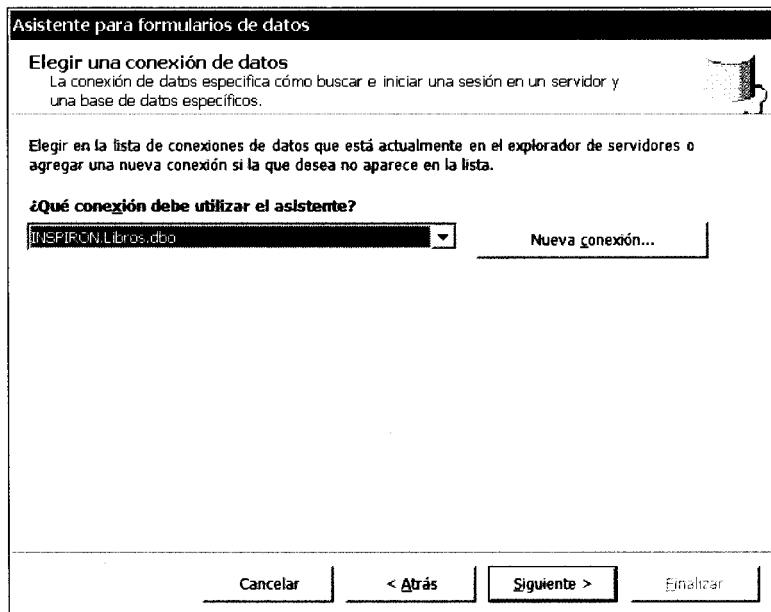


Figura 15.3. Selección de una conexión con el origen de datos

Como puede verse en la figura 15.4, hemos elegido del origen las tablas Editoriales y Libros, que ahora aparecen en el panel de la derecha en lugar de haberlo en el de la izquierda. Pulse el botón **Siguiente >** una vez más para continuar en el punto siguiente.

Nota

Ciertos pasos del asistente dependen de los elementos que se elijan aquí. Al haber seleccionado dos tablas, por ejemplo, deberemos definir una relación entre ellas, algo que no sería preciso si hubiésemos seleccionado una vista o un procedimiento almacenado. A cambio, sin embargo, podría ser necesario facilitar una lista de parámetros de entrada.

Definir la relación entre las tablas

Cuando creábamos, mediante código, un **DataSet** a partir de sendos comandos para obtener el contenido de dos tablas, un paso imprescindible era el estable-

cimiento de la relación entre dichas tablas, ya fuese creándola explícitamente o bien recuperándola con anterioridad de un esquema que hubiéramos guardado.

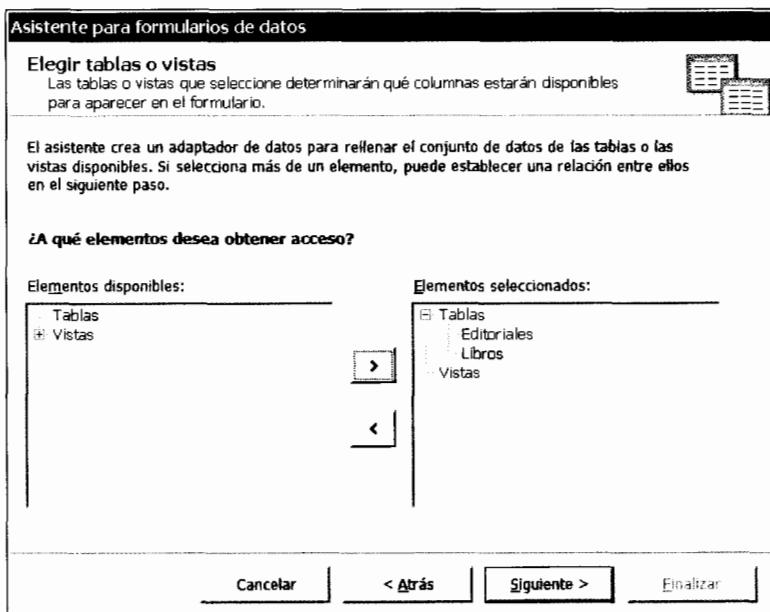


Figura 15.4. Seleccionamos las tablas con las que va a generarse el formulario

En este paso del asistente, dado que en el anterior hemos elegido como elementos dos tablas, tendremos que establecer esa relación, pero sin necesidad de crear de manera explícita el objeto `DataRelation`. En su lugar (véase figura 15.5) bastará con indicar cuál es la tabla maestra y la de detalle, seleccionando las columnas de enlace. Hay que pulsar el botón `>` para crear efectivamente la relación, momento en que aparecerá en la lista `Relaciones`.

La existencia de esta relación es fundamental a la hora de crear una relación maestro/detalle, ya que sin ella no se sabría qué libros corresponden a la editorial seleccionada en cada momento.

Selección de columnas

Que tomemos datos de dos tablas no implica que deban mostrarse todas sus columnas, aunque quizás sea éste el caso que nos interese. En cualquier caso, el paso siguiente del asistente nos permitirá elegir las columnas que deseamos tener en el formulario. Tenemos nuevamente dos paneles bien diferenciados, según se aprecia en la figura 15.6. En el de la izquierda elegimos la tabla maestra de la lista desplegable, marcando a continuación las columnas que deseamos tomar de ella. De la misma forma, en el panel derecho seleccionamos la tabla de detalle y sus columnas.

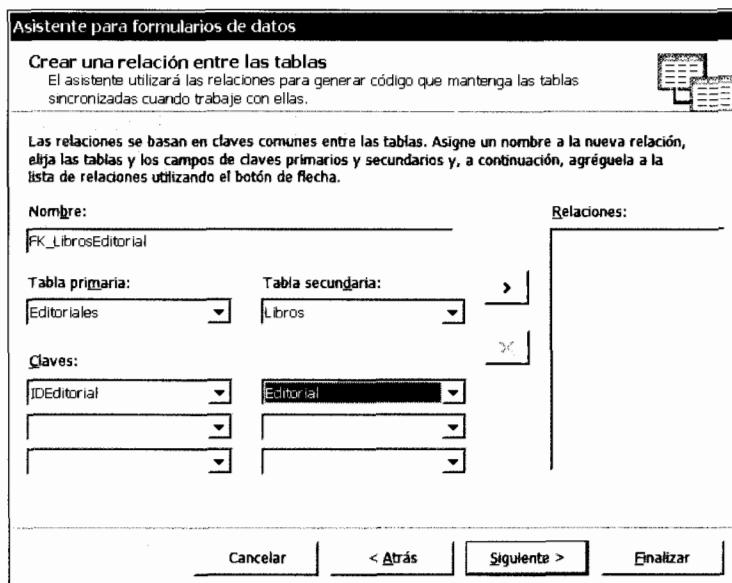


Figura 15.5. Definimos la relación que existe entre las dos tablas

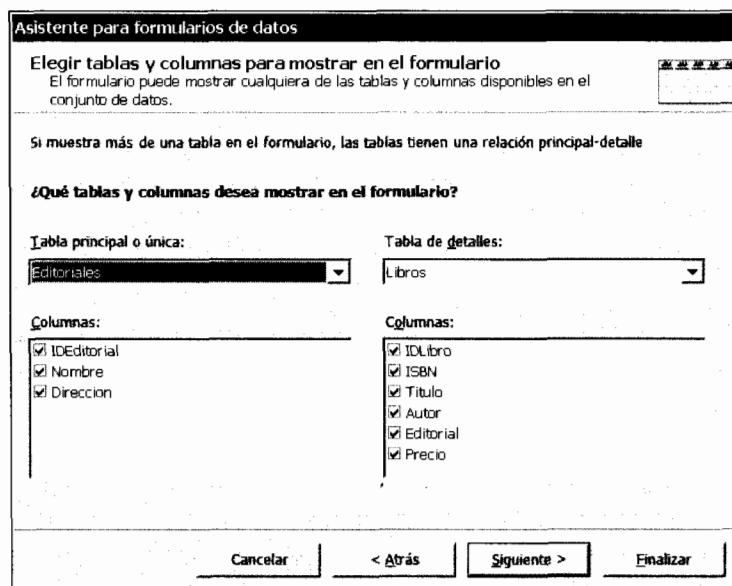


Figura 15.6. Selección de las columnas a usar en el diseño del formulario

Para nuestro formulario de ejemplo dejaremos marcadas todas las columnas aunque, en la práctica, podrían desmarcarse algunas como IDEditorial, IDLibro o Editorial.

Elegir el diseño del formulario

El último paso del asistente nos permite elegir el diseño que deseamos darle al formulario, existiendo dos opciones:

- **Todos los registros de la cuadrícula:** Emplea un control `DataGridView` para mostrar cada una de las tablas, facilitando la inserción, edición y borrado en la misma rejilla de datos.
- **Registro único en controles individuales:** Se usan controles de interfaz típicos, como `TextBox` y `CheckBox`, para mostrar la información fila a fila.

Dependiendo de la opción por la que optemos, en la parte inferior de la ventana (véase figura 15.7) estarán accesibles más o menos apartados. Si elegimos usar rejillas de datos, que es lo que nos interesa en este caso, tan sólo podemos indicar si queremos que se añada al formulario un botón para cancelar todos los cambios o no. En caso de utilizarse controles individuales, mostrando sólo una fila en cada momento, podremos elegir los botones que se añadirán para tareas de edición.

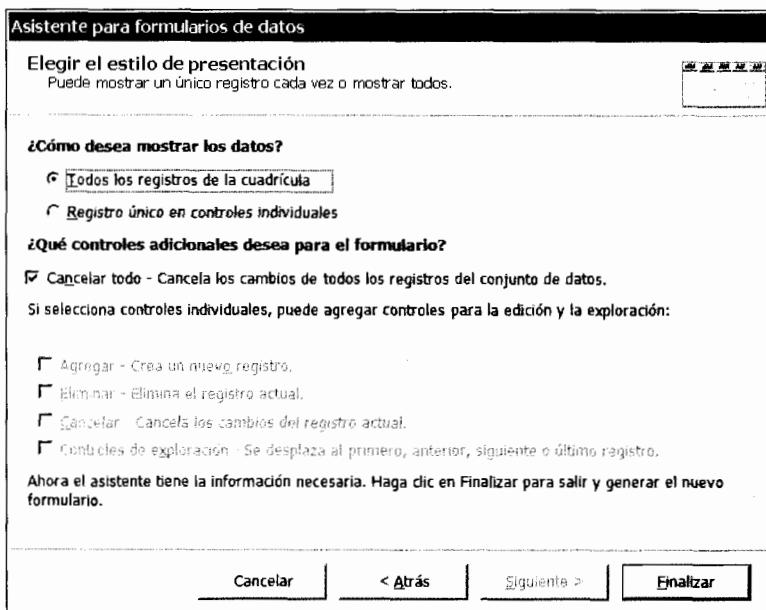


Figura 15.7. Optamos por usar controles `DataGridView` para mostrar los datos de las tablas

A parte de los botones que indiquemos explícitamente, marcando las opciones de este paso del asistente, el formulario contará también con otros que se incluyen por defecto, como un botón para recuperar los datos y otro para ejecutar la actualización.

Personalización del diseño

Al pulsar el botón **Finalizar** del asistente éste se cerrará y nos encontraremos con nuestro formulario, con su aspecto por defecto. Lógicamente puede personalizar dicho diseño, modificando títulos y disposición de los botones o los DataGrid, añadiendo nuevos elementos para realzar la interfaz, etc. En nuestro caso nos hemos limitado a modificar el ancho del segundo DataGrid, cambiando también la propiedad **Dock** de ambas cuadrículas para que se adapten automáticamente en caso de que se ajusten las dimensiones del formulario. En la figura 15.8 puede ver el formulario en el entorno de Visual Basic .NET tras los cambios.

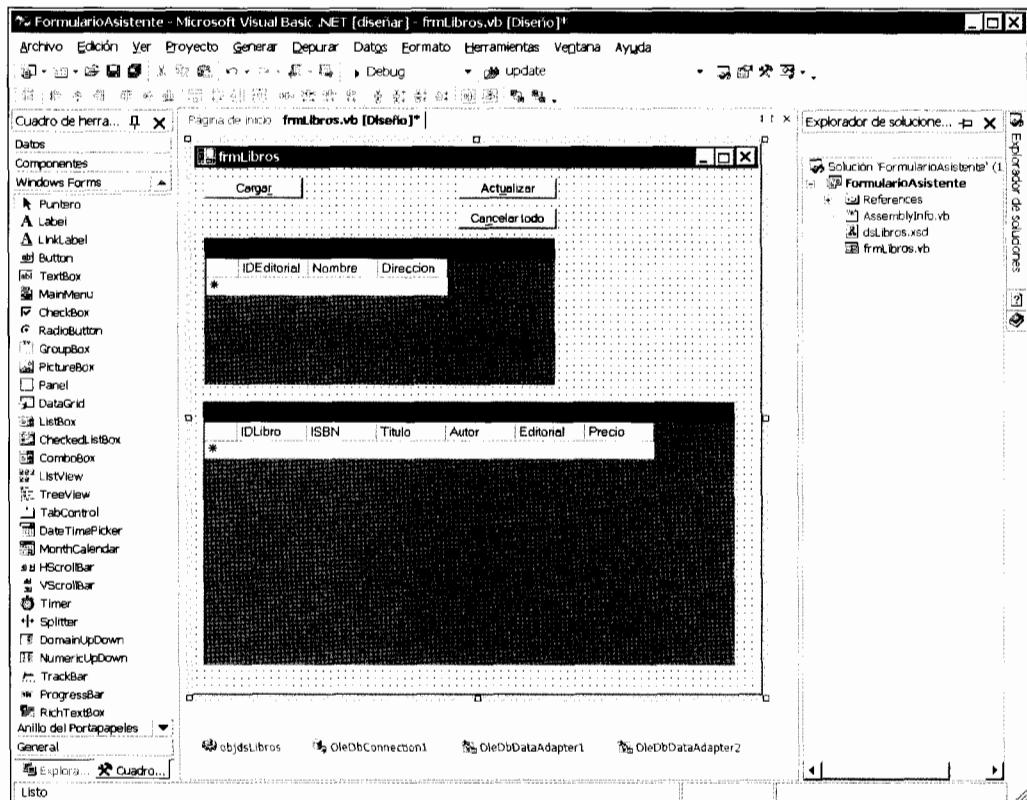


Figura 15.8. El formulario en el entorno de diseño

Nota

Observe, en la parte inferior del área de diseño, los componentes de acceso a datos que ha generado el asistente: uno para la conexión, dos adaptadores de datos y el DataSet.

Análisis del código generado

Además de todos los controles que pueden verse en el formulario, con sus propiedades debidamente establecidas, el asistente también ha generado el código necesario para recuperar los datos, actualizarlos y cancelar los cambios efectuados. Analicemos brevemente ese código para saber exactamente qué ocurrirá al ejecutar el programa.

Nota

Antes de ejecutar el programa tendrá que abrir la ventana de propiedades del proyecto y establecer el formulario recién generado como elemento de inicio.

Tras abrirse el formulario éste permanecerá vacío esperando una acción por parte del usuario. Lo habitual es que se pulse el botón **Cargar** que provoca la ejecución del código siguiente:

```
Private Sub btnLoad_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnLoad.Click
Try
    'Intento cargar el conjunto de datos.
    Me.LoadDataSet()
Catch eLoad As System.Exception
    'Añadir aquí el código de control de errores.
    'Mostrar mensaje de error, si hay alguno.
    System.Windows.Forms.MessageBox.Show(eLoad.Message)
End Try
End Sub
```

Lo único que se hace es invocar al método `LoadDataSet()` del propio formulario, controlando una posible excepción. El bloque de código que hay detrás del `Catch` está diseñado para que introduzcamos nuestro control de errores personalizado, ya que el asistente lo que hace es mostrar el posible error en una ventana.

El código del método `LoadDataSet()` es el mostrado a continuación:

```
Public Sub LoadDataSet()
    'Crear un conjunto de datos para almacenar los registros devueltos
    'de la llamada a FillDataSet.
    'Se utiliza un conjunto de datos temporal porque al realizar el
    'conjunto de datos existente.
    'implicaría que se volvieren a anclar los enlaces de datos.
    Dim objDataSetTemp As FormularioAsistente.dsLibros
    objDataSetTemp = New FormularioAsistente.dsLibros()
    Try
        'Intento llenar el conjunto de datos temporal.
        Me.FillDataSet(objDataSetTemp)
    Catch eFillDataSet As System.Exception
```

```

'Agregar aquí el código de control de errores.
Throw eFillDataSet
End Try
Try
    'Vaciar los registros obsoletos del conjunto de datos.
    objdsLibros.Clear()
    'Combinar los registros en el conjunto de datos principal.
    objdsLibros.Merge(objDataSetTemp)
Catch eLoadMerge As System.Exception
    'Agregar aquí el código de control de errores.
    Throw eLoadMerge
End Try
End Sub

```

Este método crea un objeto dsLibros, clase derivada de DataSet generada por el asistente, y lo emplea para llamar al método FillDataSet() del formulario. Observe que se usa una variable temporal, en lugar del objeto objdsLibros que ya existe en el formulario. Los controles están enlazados, lógicamente, a dicho DataSet, y no al creado aquí de forma temporal para obtener las filas de datos. De esta forma se evita que la vinculación entre controles de interfaz y DataSet se pierdan a medida que se genera el conjunto de datos.

El método FillDataSet() es el que abre la conexión y utiliza los adaptadores de datos para recuperar las filas, alojándolas en el DataSet temporal que recibe como único parámetro. Para evitar la violación de restricciones, como la de integridad referencial, se desactiva temporalmente la comprobación, reactivándose al final del método.

```

Public Sub FillDataSet(ByVal dataSet As _
    FormularioAsistente.dsLibros)
    'Desactivar la comprobación de restricciones antes de llenar
    'el conjunto de datos.
    'De esta forma los adaptadores pueden llenar el conjunto de
    'datos sin preocuparse de las dependencias entre las tablas.
    dataSet.EnforceConstraints = False
    Try
        'Abra la conexión.
        Me.OleDbConnection1.Open()
        'Intenta llenar el conjunto de datos a través de
        'OleDbDataAdapter1.
        Me.OleDbDataAdapter1.Fill(dataSet)
        Me.OleDbDataAdapter2.Fill(dataSet)
    Catch fillException As System.Exception
        'Agregar aquí el código de control de errores.
        Throw fillException
    Finally
        'Volver a activar la comprobación de restricciones.
        dataSet.EnforceConstraints = True
        'Cerrar la conexión independientemente de si se inició una
        'excepción o no.
        Me.OleDbConnection1.Close()
    End Try
End Sub

```

Una vez introducidas en el DataSet todas las filas de las dos tablas, se devuelve el conjunto de datos de nuevo al método `LoadDataSet()`. Éste limpia el DataSet existente en el formulario, llamado `objdsLibros`, e introduce en él las filas obtenidas por la llamada a `FillDataSet()`. En ese momento el formulario ya aparecería con la lista de editoriales en el primer DataGrid y la de libros en el segundo, como se aprecia en la figura 15.9. Podemos cambiar de editorial para ver sus libros, añadir entradas, eliminarlas y modificarlas.

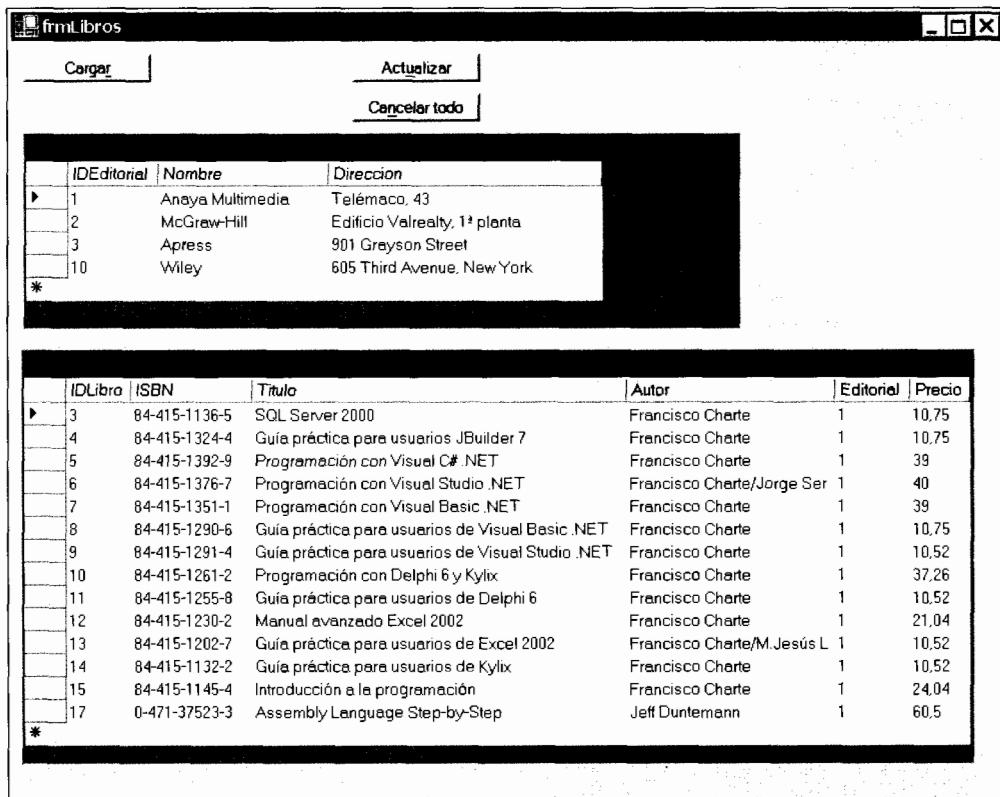


Figura 15.9. El formulario con las filas ya en las cuadriculas de datos

La pulsación del botón **Cancelar** se resuelve con una simple llamada al método `RejectChanges()` del DataSet. El botón **Actualizar** desencadena un proceso bastante más complejo que se desarrolla, inicialmente, en el método asociado al evento `Click`. Éste, básicamente, se limita a invocar al método siguiente:

```
Public Sub UpdateDataSet()
    'Crear un conjunto de datos para alojar los cambios realizados
    'en el conjunto de datos principal.
    Dim objDataSetChanges As FormularioAsistente.dsLibros =
        New FormularioAsistente.dsLibros()
```

```

'Obtener las ediciones actualizadas.
Me.BindingContext(objdsLibros, "Editoriales").EndCurrentEdit()
Me.BindingContext(objdsLibros, "Libros").EndCurrentEdit()

'Obtener los cambios realizados en el conjunto de datos
'principal.
objDataSetChanges = CType(objdsLibros.GetChanges,
                           FormularioAsistente.dsLibros)

'Comprobar si se han realizado cambios.
If (Not (objDataSetChanges) Is Nothing) Then
    Try
        'Hay cambios que necesitan aplicarse, por tanto, intenta
        'actualizar el origen de datos
        'llamando al método de actualización y pasando el conjunto
        'de datos y los parámetros.
        Me.UpdateDataSource(objDataSetChanges)
        objdsLibros.Merge(objDataSetChanges)
        objdsLibros.AcceptChanges()
    Catch eUpdate As System.Exception
        'Añadir aquí el código de control de errores.
        Throw eUpdate
    End Try
    'Añadir código para comprobar en el conjunto de datos
    'devuelto los errores que se puedan haber
    'introducido en el error del objeto de fila.
    End If
End Sub

```

En principio se llama al método `EndCurrentEdit()` del objeto `BindingManagerBase` de cada tabla, finalizando así cualquier operación de edición que pudiese haber en curso. Observe cómo se utiliza la propiedad `BindingContext` del formulario explicada en el capítulo previo.

A continuación se recuperan, en un `DataSet` temporal, los cambios que se han efectuado en el formulario, usando para ello el método `GetChanges()` del `dsLibros`. Tras comprobar que, efectivamente, hay cambios pendientes de consolidar, se llama al método `UpdateDataSource()` y combinan los cambios en el conjunto de datos al que están vinculados los controles.

El método `UpdateDataSource()`, recibiendo como parámetro la lista de cambios, se encarga de abrir de nuevo la conexión e invocar al método `Update()` de cada adaptador, facilitando la información al origen de datos.

```

Public Sub UpdateDataSource(ByVal ChangedRows As _
                           FormularioAsistente.dsLibros)
    Try
        'Sólo es necesario actualizar el origen de datos si hay
        'cambios pendientes.
        If (Not (ChangedRows) Is Nothing) Then
            'Abre la conexión.
            Me.OleDbConnection1.Open()
            'Intenta actualizar el origen de datos.

```

```
OleDbDataAdapter1.Update(ChangedRows)
OleDbDataAdapter2.Update(ChangedRows)
End If
Catch updateException As System.Exception
    'Aparecerá aquí el código de control de errores.
    Throw updateException
Finally
    'Cerrar la conexión independientemente de si se inició una
    'conexión o no.
    Me.OleDbConnection1.Close()
End Try
End Sub
```

Nota

En lugar de recuperar los cambios en un `DataSet` temporal, también podría haberse empleado el original como parámetro para el método `Update()` de los adaptadores. Sin embargo, en caso de que la actualización devolviese algún error éste se vería reflejado en las filas del conjunto de datos que hay vinculado con los controles.

El asistente para formularios Web

Si conocemos el asistente para formularios Windows, y el código que genera, el asistente para formularios Web nos resultará realmente familiar, puesto que los pasos son exactamente los mismos: selección o creación de un `DataSet`, definición de la conexión a emplear, elección de los elementos y columnas de origen y establecimiento de la relación entre las tablas. La única diferencia es que en este caso no podemos elegir entre dos diseños distintos de formulario, puesto que el asistente siempre genera uno basado en el uso de controles `DataGrid`, fácilmente vinculables a orígenes de datos.

Inicie un proyecto de aplicación Web ASP.NET, elimine el formulario que aparece por defecto y recurra al asistente, que se encuentra en el mismo lugar que el de formularios Windows, escogiendo las mismas opciones en cuanto a conjunto de datos, conexión, columnas, etc.

El resultado final deberá ser similar al de la figura 15.10, un formulario compuesto por un botón, en la parte superior, y dos `DataGrid`. Éstos se encuentran vinculados con el `DataSet` que aparece en la parte inferior.

Al ejecutar el proyecto verá que, tras pulsar el botón **Cargar**, tan sólo aparece la cuadrícula con la lista de editoriales.

Al pulsar el enlace que aparece en la primera columna de cada fila, se hará visible la segunda cuadrícula con los libros que correspondan a la editorial elegida, como se ve en la figura 15.11. Sin embargo, no obtenemos funcionalidad alguna de edición por parte de este asistente.

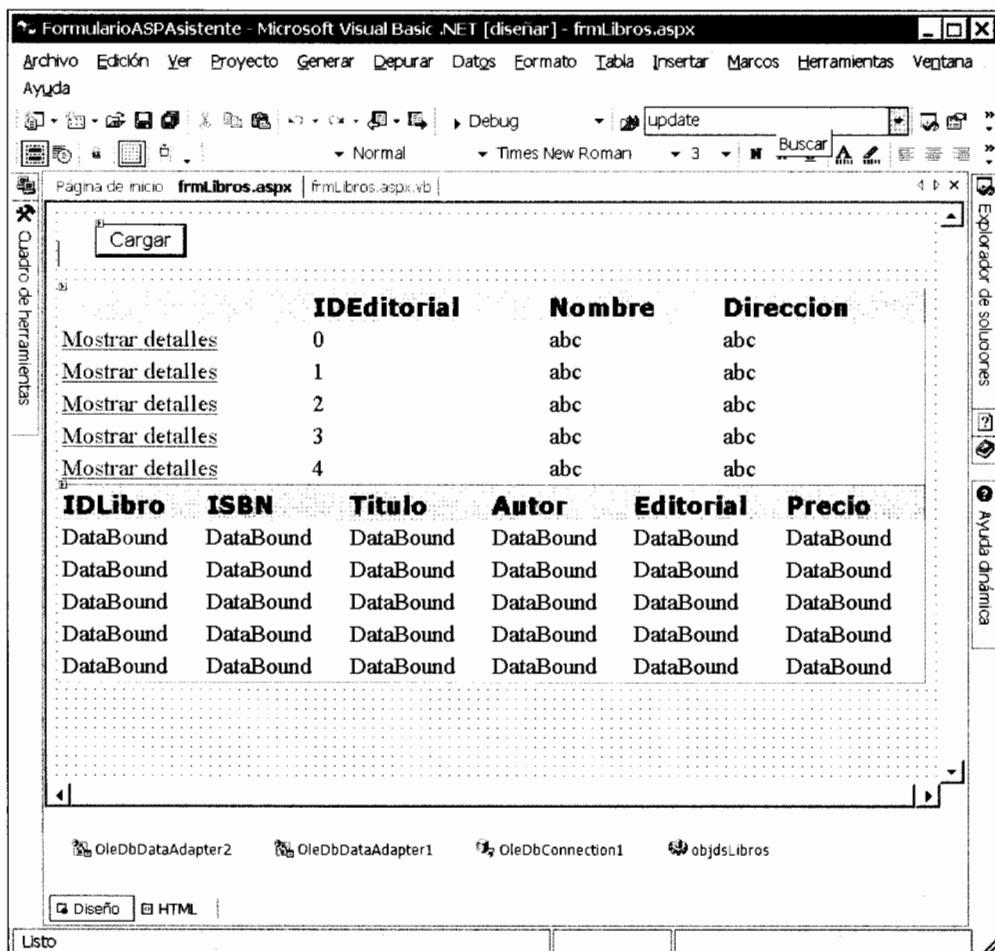


Figura 15.10. Aspecto del formulario tras cerrar el asistente

Análisis del código generado

El código de este formulario ASP.NET es también más simple que el generado para el formulario Windows, lo cual no es de extrañar ya que no existen capacidades de edición. El primer método en ejecutarse será el correspondiente al evento Click del único botón que hay en la página:

```
Private Sub buttonLoad_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles buttonLoad.Click
    Try
        Me.LoadDataSet()
        Me.masterDataGrid.SelectedIndex = -1
        Me.masterDataGrid.DataBind()
```

```

Me.detailDataGridView.Visible = False
Application("objdsLibros") = Me.objdsLibros
Catch eLoad As System.Exception
    Me.Response.Write(eLoad.Message)
End Try
End Sub

```

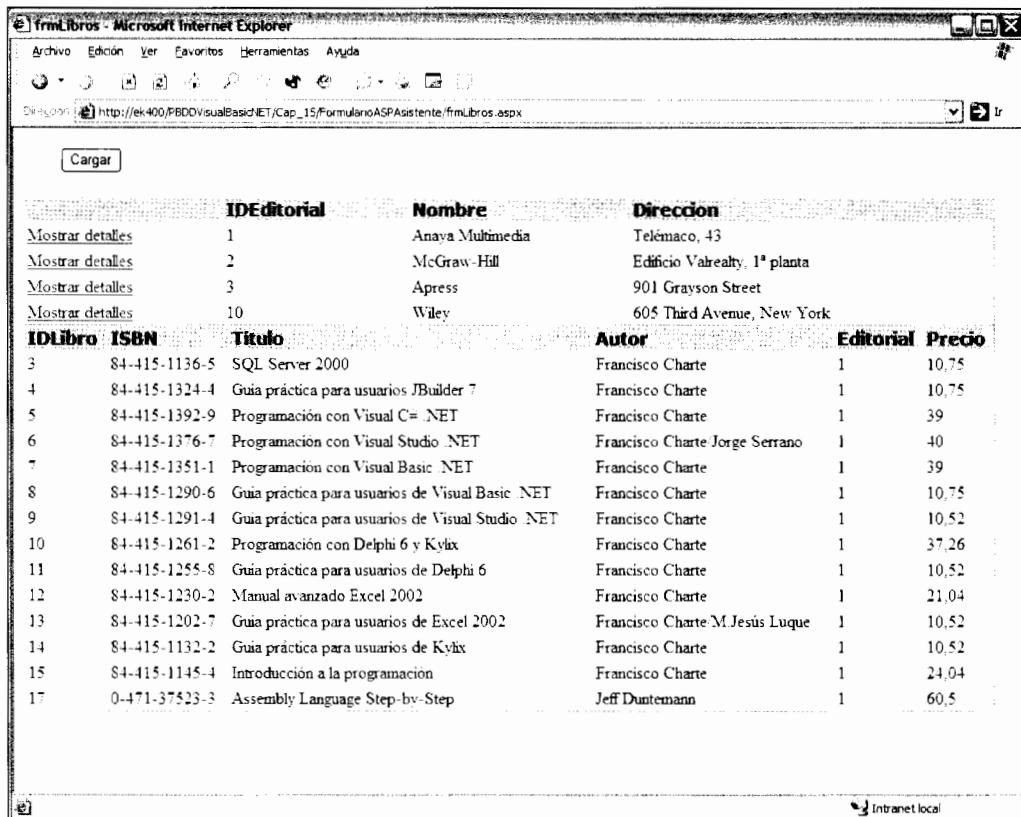


Figura 15.11. El formulario ejecutándose en Internet Explorer

Lo primero que encontramos es la invocación al método `LoadDataSet()` que, junto con `FillDataSet()`, se encargará de recuperar todos los datos en el objeto `objdsLibros` que se ha insertado en el formulario. Esos dos métodos, `LoadDataSet()` y `FillDataSet()`, son idénticos a los explicados previamente en los comentarios al código del formulario Windows.

Acto seguido se establece que no haya una fila seleccionada en el primer `DataGridView`, vinculándolo con los datos y ocultando el `DataGridView` de detalle puesto que, en principio, no tendría contenido alguno al no existir una editorial elegida.

Por último, se guarda en una variable del objeto `Application` la referencia al objeto que contiene los datos, `objdsLibros`, para facilitar el acceso a él desde otros métodos.

En este momento tendríamos el formulario con la primera rejilla mostrando las editoriales. Hasta que no se pulse el enlace **Mostrar detalles** de una de las filas la página quedará estática. Cuando eso ocurra, se generará el evento **SelectedIndexChanged** y, en respuesta, se ejecutará el código siguiente:

```
Private Sub ShowDetailGrid()
If (Me.masterDataGrid.SelectedIndex <> -1) Then
    Dim parentRows As System.Data.DataView
    Dim childRows As System.Data.DataView
    Dim currentParentRow As System.Data.DataRowView
    Me.objdsLibros = CType(Application("objdsLibros"), _
        FormularioASPAAsistente.dsLibros)
    parentRows = New DataView()
    parentRows.Table = Me.objdsLibros.Tables("Editoriales")
    currentParentRow =
        parentRows(Me.masterDataGrid.SelectedIndex)
    childRows =
        currentParentRow.CreateChildView("FK_LibrosEditorial")
    Me.detailDataGrid.DataSource = childRows
    Me.detailDataGrid.DataBind()
    Me.detailDataGrid.Visible = True
Else
    Me.detailDataGrid.Visible = False
End If
End Sub
```

A pesar de la extensión, en cuanto a número de sentencias, el proceso que ejecutado es relativamente simple, lo que ocurre es que se efectúa paso a paso. Se utiliza una vista, **parentRows**, para obtener una referencia a la fila que está elegida actualmente en el **DataGrid** maestro. Con esta información se utiliza el método **CreateChildView()** de la fila para obtener el conjunto de filas de detalle, enlazándolo con el segundo **Datagrid** y haciendo éste visible.

Añadir capacidades de edición

A la vista está que el formulario generado por el asistente para aplicaciones Web queda lejos de tener la funcionalidad que ofrecen los formularios Windows, no facilitando las operaciones de edición. Si queremos esta funcionalidad, tendremos que añadirla nosotros mismos. Para ello hay que añadir nuevos enlaces al **DataGrid** que contengan los datos a editar, por ejemplo el que muestra los libros, así como código para gestionar los eventos que generen. Es lo que vamos a hacer en los puntos siguientes.

Inserción de los enlaces de edición

El primer paso será añadir al segundo **DataGrid** los enlaces necesarios para que, en ejecución, cada fila disponga de un enlace que permita entrar en modo de edición. Haga clic con el botón secundario del ratón sobre el segundo **DataGrid**,

para abrir el menú emergente, y luego seleccione la opción **Generador de propiedades**. Se abrirá un cuadro de diálogo con múltiples apartados a los que puede acceder con los botones que aparecen a la izquierda. Pulse el botón **Columnas** (véase figura 15.12) para acceder a la lista de columnas de la cuadrícula.

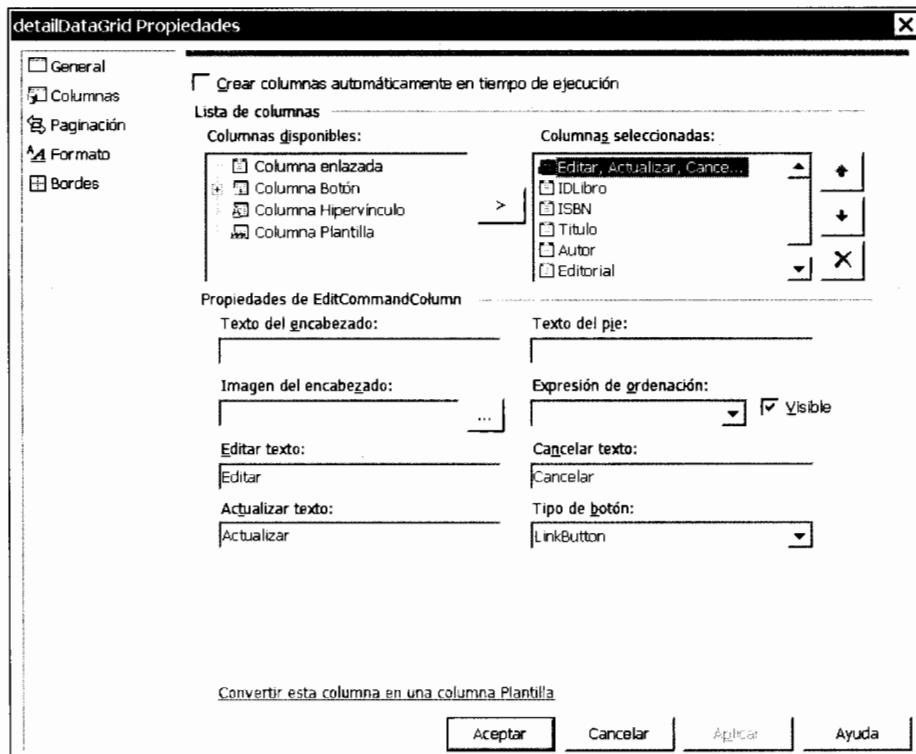


Figura 15.12. Añadimos a la rejilla una nueva columna para el botón de edición

Elija en el apartado **Columnas disponibles** la sección **Columna de botón**, añadiendo a **Columnas seleccionadas** un elemento **Editar, Actualizar, Cancelar**. Colóquelo como primer elemento, para que aparezca como la columna más a la izquierda de la cuadrícula. Si quiere puede personalizar el texto de los enlaces, así como cambiar el **Tipo de botón** que, por defecto, es **LinkButton**.

Cierre el cuadro de diálogo pulsando el botón **Aceptar**. Verá que el **DataGrid** ya muestra el enlace o botón para editar cada fila, pero si ejecuta el programa no obtendrá respuesta alguna a la pulsación sobre él.

Eventos y elementos de un **DataGrid**

Los enlaces o botones añadidos como columnas de un **DataGrid**, en nuestro caso el enlace **Editar** y los enlaces **Actualizar** y **Cancelar** que aparecerán tras pulsar el anterior, generan eventos al pulsarse sobre ellos, concretamente los eventos

`EditCommand`, `UpdateCommand` y `CancelCommand`, respectivamente. En respuesta a estos eventos deberá ejecutarse el código necesario para entrar en modo de edición, actualizar el `DataSet` o cancelar la edición.

Para responder adecuadamente a estos eventos, afectando a la fila sobre la que se ha pulsado, es imprescindible saber qué fila es esa. Cada fila de un `DataGridView` es un elemento `DataGridViewItem`, a los que podemos acceder mediante la colección `Items`. Los eventos anteriores reciben en `e.Item` una referencia al `DataGridViewItem` sobre el que se ha pulsado, y mediante sus propiedades podemos conocer el número de fila: `ItemIndex`; acceder a las celdillas que la componen: `Cells` (`NúmeroColumna`), etc.

Cada una de las celdillas puede mantener un texto o un control. Mientras se edita el contenido de una fila, por ejemplo, cada columna contiene un `TextBox`, del que podemos recuperar la información introducida para asignarla a la fila adecuada del `DataTable`.

Código asociado a los eventos

Ahora que tenemos unas nociones sobre algunas propiedades del `DataGridView`, veamos cómo codificar los eventos mencionados para así facilitar la edición en la cuadrícula. Comencemos por el evento `EditCommand`:

```
' Al pulsar el enlace de edición
Private Sub detailDataGridView_EditCommand(ByVal source As Object,
                                         ByVal e As System.Web.UI.WebControls.DataGridCommandEventEventArgs) _
Handles detailDataGridView.EditCommand
    ' establecemos el índice de la fila a editar
    detailDataGridView.EditRowIndex = e.Item.ItemIndex
    ShowDetailGrid() ' y actualizamos la rejilla de detalle
End Sub
```

Para activar el modo de edición es necesario asignar a la propiedad `EditRowIndex` el índice de la fila a editar, índice que obtenemos de la propiedad `ItemIndex` del `DataGridViewItem` sobre el que se ha pulsado. Puede parecer redundante esta asignación, pero la independencia entre el evento `EditCommand` y la propiedad `EditRowIndex` ofrece cierta flexibilidad, permitiendo, por ejemplo, la edición con controles externos, independientes del `DataGridView`.

A continuación invitamos al método `ShowDetailGrid()` ya existente, actualizando la vista de la cuadrícula de detalle. En ese momento la fila aparecerá con múltiples `TextBox` en su interior, uno por cada columna que pueda editarse.

Estando en el modo de edición, e independientemente de que se hayan hecho o no cambios en los datos de las columnas, puede pulsarse el enlace de actualización o el de cancelación. Ocupémonos de éste último en primer lugar, puesto que resulta más simple como puede verse a continuación:

```
' Si se pulsa el enlace de cancelación
Private Sub detailDataGridView_CancelCommand(ByVal source As Object,
                                         ByVal e As System.Web.UI.WebControls.DataGridCommandEventEventArgs) _
```

```

Handles detailDataGrid.CancelCommand
    ' simplemente devolvemos la rejilla
    detailDataGrid.EditRowIndex = -1
    ShowDetailGrid() ' a su estado anterior
End Sub

```

Simplemente asignamos el valor -1 a `EditRowIndex`, para finalizar la edición que estaba en curso, y llamamos a `ShowDetailGrid()` ignorando los cambios que se pudiesen haber efectuado. La cuadrícula vuelve a aparecer con sus datos originales, ya que no existe una conexión directa entre los `TextBox` que muestra el `DataGridView` y los `DataRow` del conjunto de datos subyacente.

Por último, tendremos que codificar el método del evento `UpdateCommand` que, como se puede ver, es bastante más complejo que los dos anteriores:

```

' Si se pulsa el botón de actualización
Private Sub detailDataGrid_UpdateCommand(ByVal source As Object,
ByVal e As System.Web.UI.WebControls.DataGridCommandEventEventArgs) —
Handles detailDataGrid.UpdateCommand
    ' Recuperamos la referencia al DataSet
    objdsLibros = Application("objdsLibros")
    ' y buscamos la fila correspondiente al IDLibro
    ' del libro que está editándose
    Dim Fila As dsLibros.LibrosRow =
        objdsLibros.Libros.FindByIDLibro(
            CType(e.Item.Cells(1).Controls(0), TextBox).Text)

    ' Si hemos obtenido la fila
    If Not Fila Is Nothing Then
        With Fila
            ' la asignamos los valores que hay en el DataGridView
            .ISBN = CType(e.Item.Cells(2).Controls(0), TextBox).Text
            .Titulo = CType(e.Item.Cells(3).Controls(0), TextBox).Text
            .Autor = CType(e.Item.Cells(4).Controls(0), TextBox).Text
            .Editorial = CType(e.Item.Cells(5).Controls(0),
                TextBox).Text
            .Precio = CType(e.Item.Cells(6).Controls(0), TextBox).Text
        End With
    End If
    ' finalizamos la edición
    detailDataGrid.EditRowIndex = -1
    ShowDetailGrid() ' y actualizamos la rejilla
End Sub

```

Tras recuperar la referencia al `DataSet`, almacenada como variable en la propiedad `Application` de la página, buscamos en él la fila que corresponde al libro que está editándose. Se obtiene un objeto `LibrosRow`, que no es más que un derivado de `DataRow` en el que existen propiedades específicas para el acceso a las columnas de esta tabla.

Observe cómo se recupera el control que hay en cada celdilla, mediante la expresión `Cells(n).Controls(0)`, convirtiéndolo en un `TextBox` para poder leer el texto que contiene. La primera celdilla, a la que corresponde el índice 0, es

la que contiene los enlaces de edición, así que el identificador del libro se encuentra en la segunda y los demás datos en las siguientes. Los recuperamos asignándolos a las columnas de la fila en el DataSet.

Si ejecuta el programa en este momento, verá que puede efectuar cambios y que, tras pulsar el enlace **Actualizar**, éstos permanecen ahí, en el DataSet, pero si comprueba el origen de datos, o simplemente sale del programa y vuelve a ejecutarlo, comprobará que esos cambios se pierden. Tenemos que actualizar el origen de datos.

Actualización del origen de datos

No tiene sentido actualizar la información con el origen cada vez que se modifique una columna, así que necesitaremos añadir un botón o enlace para que dicha actualización se efectúe a demanda. Como se aprecia en la figura 15.13, insertaremos ese botón a la derecha del que ya existía en la parte superior de la página.

The screenshot shows a Microsoft Internet Explorer window titled "frmlibros - Microsoft Internet Explorer". The address bar displays the URL "http://ek400.PBDDVisualBasic.NET/Cap_15/FormularioASPAsistente/frmlibros.aspx". The page contains a DataGrid control with the following columns: IDLibro, ISBN, Titulo, Autor, Editorial, and Precio. The DataGrid has 17 rows of data. At the top left of the grid, there are two buttons: "Cargar" and "Actualizar". Below the grid, there is a "Listo" button and an "Intranet local" link. The DataGrid data is as follows:

IDLibro	ISBN	Título	Autor	Editorial	Precio
1	84-415-1136-5	SQL Server 2000	Francisco Charte Ojeda	1	10,75
2	84-415-1324-4	Guía práctica para usuarios JBuilder 7	Francisco Charte	1	10,75
3	84-415-1392-9	Programación con Visual C# .NET	Francisco Charte Ojeda	1	39
4	84-415-1376-7	Programación con Visual Studio .NET	Francisco Charte Jorge Serrano	1	40
5	84-415-1351-1	Programación con Visual Basic .NET	Francisco Charte	1	39
6	84-415-1290-6	Guía práctica para usuarios de Visual Basic .NET	Francisco Charte	1	10,75
7	84-415-1291-4	Guía práctica para usuarios de Visual Studio .NET	Francisco Charte	1	10,52
8	84-415-1261-2	Programación con Delphi 6 y Kylix	Francisco Charte	1	37,26
9	84-415-1255-8	Guía práctica para usuarios de Delphi 6	Francisco Charte	1	10,52
10	84-415-1230-2	Manual avanzado Excel 2002	Francisco Charte	1	21,04
11	84-415-1202-7	Guía práctica para usuarios de Excel 2002	Francisco Charte M Jesús Luque	1	10,52
12	84-415-1132-2	Guía práctica para usuarios de Kylix	Francisco Charte	1	10,52
13	84-415-1145-4	Introducción a la programación	Francisco Charte	1	24,04
14	0-471-37523-3	Assembly Language Step-by-Step	Jeff Duntemann	1	60,5

Figura 15.13. Aspecto del DataGrid con los enlaces de edición

Haga doble clic sobre el botón e introduzca el código siguiente:

' Al pulsar el botón de actualización

```
Private Sub Button1_Click(ByVal sender As System.Object,
```

```

    ByVal e As System.EventArgs) Handles Button1.Click
        ' Recuperarímos el DataSet
        objdsLibros = Application("objdsLibros")
        Try ' y lo usamos para actualizar el origen
            ' mediante el adaptador de la tabla 'Libros'
            OleDbDataAdapter2.Update(objdsLibros)
        Catch Excepcion As Exception
            ' mostrando cualquier error que se produzca
            Page.Response.Write(Excepcion.Message)
        End Try
    End Sub

```

Podríamos haber dado todos los pasos que se ejecutaban en el código generado por el asistente de formularios Windows para efectuar la actualización, pero hemos optado por algo mucho más directo. Recuperamos la referencia al `DataSet` y lo usamos directamente como parámetro del método `Update()` del segundo adaptador de datos.

El efecto, si lo comprueba, es el mismo, se escriben los cambios en el origen de datos completándose el ciclo que le permitirá editar la información desde un navegador Web.

	IDEditorial	Nombre	Dirección			
Libros	1	Anaya Multimedia	Telemaco, 43			
Libros	2	McGraw-Hill	Edificio Valekay, 1 ^a planta			
Libros	3	Apress	901 Grayson Street			
Libros	10	Wiley	605 Third Avenue, New York			
	IDLibro	ISBN	Título	AUTOR	Editorial	Precio
Editar	3	84-415-1136-5	SQL Server 2000	Francisco Charte Ojeda	1	10,75
Editar	4	84-415-1321-4	Guía práctica para usuarios JBuilder 7	Francisco Charte	1	10,75
Actualizar	5	84-415-1392-9	Programación con Visual Studio .NET	Francisco Charte Ojeda	1	39
Editar	6	84-415-1376-7	Programación con Visual Studio .NET	Francisco Charte Jorge Serrano	1	40
Editar	7	84-415-1351-1	Programación con Visual Basic .NET	Francisco Charte	1	39
Editar	8	84-415-1290-6	Guía práctica para usuarios de Visual Basic .NET	Francisco Charte	1	10,75
Editar	9	84-415-1291-4	Guía práctica para usuarios de Visual Studio .NET	Francisco Charte	1	10,52
Editar	10	84-415-1261-2	Programación con Delphi 6 y Kylix	Francisco Charte	1	37,26
Editar	11	84-415-1255-8	Guía práctica para usuarios de Delphi 6	Francisco Charte	1	10,52
Editar	12	84-415-1230-2	Manual avanzado Excel 2002	Francisco Charte	1	21,04
Editar	13	84-415-1202-7	Guía práctica para usuarios de Excel 2002	Francisco Charte M Jesús Luque	1	10,52
Editar	14	84-415-1132-2	Guía práctica para usuarios de Kylix	Francisco Charte	1	10,52
Editar	15	84-415-1145-4	Introducción a la programación Assembly Language Step-by-Step	Francisco Charte	1	24,04
Editar	17	0-471-37523-3		Jeff Dunemann	1	60,5

Figura 15.14. Edición de una fila del DataGrid

Nota

Tomando como base todo el proceso descrito, puede añadir al mismo DataGrid la posibilidad de eliminar cualquiera de sus filas y también la de añadir filas nuevas. Le resultará un buen ejercicio para conocer más detalles sobre DataGrid. El proceso de actualización con el origen de datos le servirá tal cual está.

Resumen

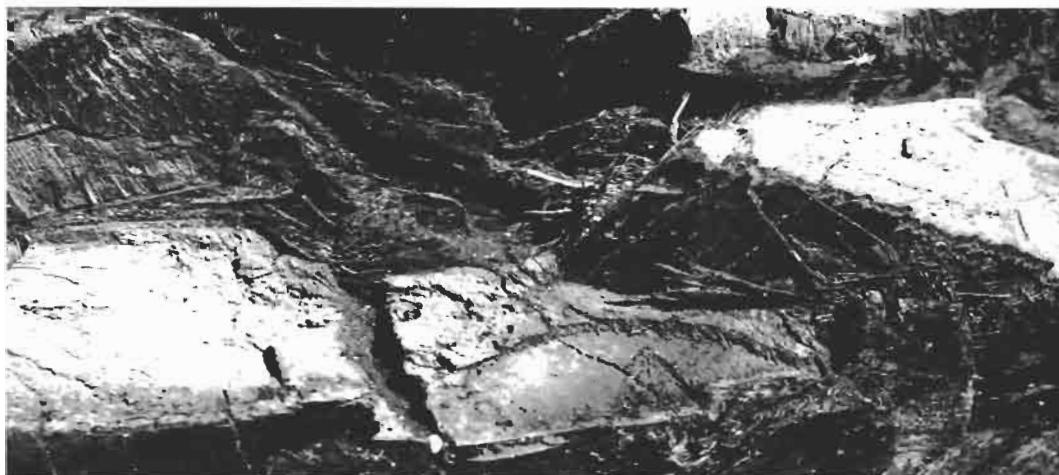
Este último capítulo dedicado al estudio de los elementos de Visual Studio .NET relacionados con el acceso a datos, le ha servido para conocer dos de los asistentes más potentes con que cuenta el entorno, mediante los cuales pueden generarse formularios con controles vinculados a datos tanto en aplicaciones Windows como en aplicaciones Web.

Ha podido ver que el asistente para formularios Web genera menos funcionalidad que el equivalente para Windows, y también cómo suplir dicha deficiencia con la personalización de las columnas del DataGrid y la escritura de algunas sentencias.

En resumen, los capítulos de esta tercera parte le han facilitado gran parte de la información que necesitará para crear interfaces de usuario, Windows o Web, con controles vinculados a los elementos que conoció en la segunda parte, principalmente conjuntos de datos, tablas y vistas.

Parte IV

Resolución
de casos
concretos



16

Conexión genérica

Los capítulos de esta última parte del libro, en general mucho más breves que los anteriores, le explican cómo abordar casos concretos, generalmente de forma directa mediante algún ejemplo.

En algunos de ellos se resume información vertida en capítulos previos, o bien se profundiza en temas que, a pesar de haberse tratado anteriormente, no se ha hecho con detalle.

En este decimosexto capítulo, que es el primero de la cuarta parte, el objetivo es codificar un ejemplo en el que se conecta con un origen de datos y se recupera información, como en los capítulos previos, pero sin emplear directamente las clases específicas de cada proveedor a lo largo de todo el código, sino tan sólo en un punto.

De esta forma conseguiremos una conexión genérica muy fácil de modificar, incluso en ejecución a demanda del usuario.

Interfaces genéricas

En el quinto capítulo, dedicado al estudio del modelo de objetos ADO.NET, conocí múltiples interfaces que eran implementadas por clases de cada proveedor de datos.

A modo de recordatorio, las interfaces que podríamos considerar más interesantes son las siguientes:

- **IDbConnection:** Implementada por `SqlConnection`, `OleDbConnection`, `OracleConnection` y `OdbcConnection`, define el conjunto de métodos base para trabajar con una conexión estableciendo la cadena de parámetros, abriéndola y cerrándola, etc.
- **IDbCommand:** Esta interfaz, implementada por las clases `SqlCommand`, `OleDbCommand`, `OracleCommand` y `OdbcCommand`, establece el conjunto de miembros necesario para definir un comando y ejecutarlo, ya sea obteniendo a cambio un lector de datos o no.
- **IDataAdapter e IDbDataAdapter:** La primera define los métodos `Fill()` y `Update()` que hemos empleado repetidamente para llenar un `DataSet` o actualizar el origen, mientras que la segunda declara las propiedades `SelectCommand`, `InsertCommand`, `UpdateCommand` y `DeleteCommand`. La clase `DataAdapter`, también genérica, se encarga de implementar los dos primeros métodos y servir como base de `DbDataAdapter` que, a su vez, es la base común de `SqlDataAdapter`, `OleDbDataAdapter`, `OracleDataAdapter` y `OdbcDataAdapter`. Éstas implementan directamente la interfaz `IDbDataAdapter`.
- **IDataReader:** Interfaz implementada por las clases `SqlDataReader`, `OleDbDataReader`, `OracleDataReader` y `OdbcDataReader`, que facilitan la lectura unidireccional de conjuntos de datos sin posibilidades de edición.

Si necesita refrescar más la memoria observe las figuras 5.2, 5.3 y 5.4, en las cuales se representaban esquemáticamente las relaciones entre estas interfaces y las clases que las implementan. En el punto siguiente las usaremos para escribir un ejemplo que pueda operar indistintamente sobre una base de datos Access, SQL Server u Oracle.

Diseño de la interfaz de usuario

Comenzaremos creando el formulario Windows en el que incluiremos todo el código. Su finalidad será permitirnos seleccionar una conexión de tres posibles, introducir un comando de selección y ejecutarlo, mostrando los resultados en una cuadrícula.

Inicie un nuevo proyecto e inserte en el formulario los elementos que puede ver en la figura 16.1: tres botones de radio en un grupo, un cuadro de texto para introducir la sentencia, un botón para recuperar los datos y un `DataGridView` ocupando toda la parte inferior. Puede asignar identificadores más coherentes a los controles, así como modificar ciertas propiedades, como `Anchor`, para que los controles se ajusten automáticamente a los cambios de tamaño del formulario.

Como puede verse, el ejemplo sólo recuperará información del origen seleccionado, pero igualmente podrían añadirse los elementos necesarios para facilitar la edición y actualización.

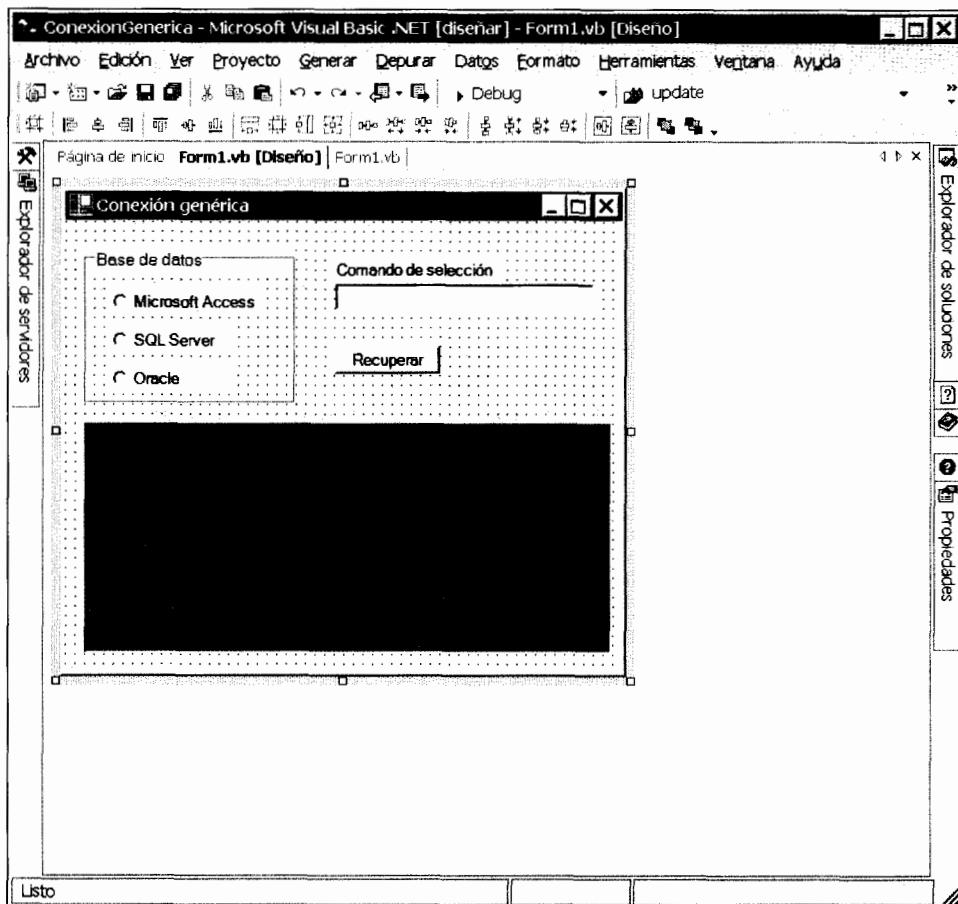


Figura 16.1. Diseño del formulario

Implementación de funciones

El paso siguiente será la codificación de los procedimientos y funciones necesarios para abrir la conexión y recuperar los datos a partir del comando de selección que se haya introducido.

Comenzaremos escribiendo la declaración de las cuatro variables siguientes al inicio del módulo:

```
' Declararemos las variables genéricas
Dim Conexion As IDbConnection
Dim Comando As IDbCommand
Dim Adaptador As IDbDataAdapter
Dim dsDatos As DataSet
```

Nota

Recuerde que es necesario añadir las sentencias Imports al inicio del módulo para importar los ámbitos System.Data.OleDb, System.Data.SqlClient y System.Data.OracleClient. Para poder usar éste último deberá añadir una referencia en el proyecto al ensamblado del proveedor Oracle.

Tenemos una conexión, un comando de selección, un adaptador de datos y un conjunto de datos, todos ellos sin ninguna vinculación específica con un proveedor ADO.NET, es decir, son objetos independientes.

Hacemos doble clic sobre el botón **Recuperar**, que ejecutará todo el código que vamos a escribir a continuación y que hemos dividido en varios métodos. La secuencia de ejecución es la siguiente:

```
' Al pulsar el botón de recuperación
Private Sub btnRecuperar_Click(ByVal sender As System.Object, _  

    ByVal e As System.EventArgs) Handles btnRecuperar.Click  

If CreaObjetos() Then ' si podemos crear los objetos  

    AbrirConexion() ' abrimos la conexión  

    RecuperarDatos() ' leemos los datos  

    VincularRejilla() ' y vinculamos el DataGridView  

Else  

    ' en caso contrario indicamos el problema  

    MessageBox.Show( _  

        "Debe elegir un proveedor y facilitar un comando")  

End If  

End Sub
```

Primero se invoca a la función **CreaObjetos()** y, si ésta devuelve el valor **True**, se ejecutan los métodos **AbrirConexion()**, **RecuperarDatos()** y **VincularRejilla()**. Si **CreaObjetos()** devuelve **False** es que no se ha seleccionado uno de los proveedores o facilitado la sentencia SQL de selección, así que se muestra un aviso y el programa queda a la espera.

La única porción de código en el que se emplean las clases propias de cada proveedor es la mostrada a continuación:

```
' Este método se encargará de crear los objetos específicos,  

' asignando las referencias a las variables genéricas
Private Function CreaObjetos() As Boolean  

    ' dependiendo del RadioButton elegido  

    ' creamos la conexión adecuada  

Select Case IIf(rbAccess.Checked, 1, _  

    IIf(rbSQLServer.Checked, 2, 3))  

Case 1 ' Access
    Conexion = New OleDbConnection(_
        "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & _  

        "C:\PBDDVisualBasicNET\Cap_03\Libros.mdb")
    Adaptador = New OleDbDataAdapter()
```

```

Case 2 ' SQL Server
Conexion = New SqlConnection(
    "Data Source=Inspiron; Initial Catalog=" & _
    "Libros; User ID=sa")
Adaptador = New SqlDataAdapter()
Case 3 ' Oracle
Conexion = New OracleConnection(
    "Data Source=Libros; User ID=scott; Password=tiger")
Adaptador = New OracleDataAdapter()
Case Else ' si no es ni ha hecho una elección
    Return False ' lo indicamos
End Select

' Creamos el comando a partir de la conexión
Comando = Conexion.CreateCommand()

' devolvemos True o False según se haya
' introducido o no un comando de selección
Return tbComando.Text <> ""

End Function

```

Como puede ver, se asigna a la variable Conexion un OleDbConnection, SqlConnection u OracleConnection, dependiendo del botón de radio seleccionado. También se crea el adaptador de datos específico. Observe que la creación de comando es genérica, pudiéndose haber efectuado en cualquier otro punto del programa sin necesidad de tener que conocer el tipo real del objeto al cual apunta Conexion.

Si este método devuelve el valor True, porque se hayan creado satisfactoriamente los objetos y, además, el cuadro de texto no esté vacío, se procederá a ejecutar el método AbrirConexion():

```

' Esta método abre la conexión
Private Sub AbrirConexion()
    Try ' intentamos abrirla
        Conexion.Open()
    Catch Excepcion As Exception
        ' notificando cualquier problema
        MessageBox.Show(Excepcion.Message)
    End Try
End Sub

```

Simplemente se usa el método Open() de IDbConnection para abrir la conexión, notificando un posible fallo que pudiera producirse.

Después se ejecutará el método RecuperarDatos() que, tal y como se puede ver a continuación, prepara el comando, lo enlaza con el adaptador y crea el conjunto de datos:

```

' Esta método recupera los datos
Private Sub RecuperarDatos()
    ' Asignamos el comando de selección
    Comando.CommandText = tbComando.Text

```

```

' y asociamos el comando con el adaptador
Adaptador.SelectCommand = Comando
' creamos el DataSet
dsDatos = New DataSet()
' y lo llenamos de datos
Adaptador.Fill(dsDatos)
' cerramos la conexión
Conexion.Close()
End Sub

```

Nota

Observe que el `DataSet` se crea nuevo cada vez que se ejecuta el método `RecuperarDatos`, de no hacerse así, dos llamadas consecutivas al método, con comandos idénticos o diferentes, causarían que el conjunto de datos fuese una combinación de ambos resultados.

Por último tenemos el método `VincularRejilla()` en el que, con dos simples asignaciones, se enlaza el `DataGrid` con la tabla creada en el `DataSet`. Ésta, al no haberse facilitado ningún nombre al llamar al método `Fill()` del adaptador, siempre se llama `Table`.

```

' Este método vincula la rejilla con
' la tabla añadida al DataSet
Private Sub VincularRejilla()
    DataGrid1.DataSource = dsDatos
    DataGrid1.DataMember = "Table"
End Sub

```

Ejecución del proyecto

No necesita ninguna preparación adicional para ejecutar este proyecto, asumiendo que tiene instalado el proveedor para Oracle y que ha añadido la correspondiente referencia a su ensamblado en el proyecto. De no ser así, elimine el código que hace uso de los objetos `OracleXXX`.

Al ejecutar el programa tendrá el formulario vacío, tal y como aparecía en la figura 16.1, debiendo seleccionarse una de las tres conexiones disponibles e introducirse un comando de selección. En la figura 16.2 puede ver el resultado de una sentencia en la que se combinan las tablas de editoriales y libros. Una vez introducida la consulta, no hay más que cambiar de proveedor y volver a pulsar el botón **Recuperar** para obtener los datos de otro origen.

De no existir las interfaces genéricas indicadas en el primer punto del capítulo, la creación de un programa como éste hubiese requerido mucho más código, prácticamente el triple, al tener que codificar la apertura y el resto de acciones de manera específica para cada proveedor.

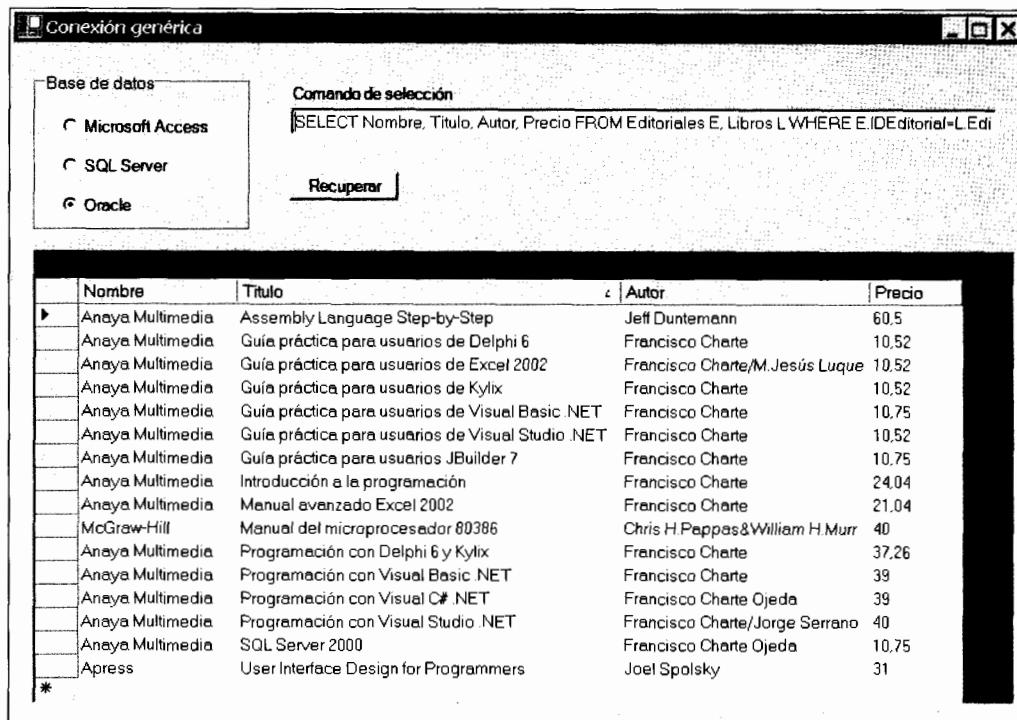


Figura 16.2. El programa en ejecución



Parte 2

Acesso a Oracle desde Visual Basic .NET

El RDBMS de Oracle es el más utilizado a escala mundial, especialmente en sistemas de tamaño medio y grande, lo cual no es de extrañar pues se trata de una de las empresas con más experiencia en el sector, una empresa que ha hecho de su producto de base de datos el centro de todo su negocio.

A pesar de las mejoras introducidas en las versiones 8i y 9i, lo cierto es que Oracle no resulta un RDBMS fácil de instalar ni administrar, tarea que suele quedar encomendada a administradores de bases de datos especializados en este producto. No obstante, como programadores acostumbrados a utilizar herramientas de distinta naturaleza, nos debe resultar más fácil usar Oracle que a un usuario que tan sólo tenga conocimientos básicos.

En los capítulos de la primera y segunda parte se han dado varios pasos para, al final, llegar a usar una base de datos Oracle desde una aplicación Visual Basic .NET. El objetivo de este capítulo es mostrarle, de forma resumida y rápida, cuál sería todo el proceso de configuración necesario.

Servidor, servicio y esquema

De partida, como es lógico, deberá contar con un servidor en el que esté ejecutándose el RDBMS de Oracle, conociendo el servicio al que quiere conectar, el nombre del esquema y la clave de acceso. Seguramente la empresa para la que vaya a trabajar ya tenga el RDBMS instalado y en funcionamiento, pero también es seguro que

no podrá usar ese servidor durante la fase de desarrollo, especialmente si ya se encuentra en explotación con otras aplicaciones.

Es posible, por tanto, que se vea obligado a instalar un servidor Oracle para utilizar durante la creación del proyecto, hasta su puesta en explotación. Si no dispone del producto, y desea probarlo como posible solución a sus necesidades, no tiene más que dirigirse a <http://otn.oracle.com>, registrarse gratuitamente si no tiene ya una cuenta OTN y descargar la versión en la que esté interesado. En la figura 17.1 puede ver cómo se opta por la *Release 2* de Oracle9i, que viene en tres archivos correspondientes a los tres CD-ROM del producto.

Tras desempaquetar cada uno de los archivos en su respectivo directorio, no es necesario grabar en un CD la información, podemos poner en marcha la instalación.

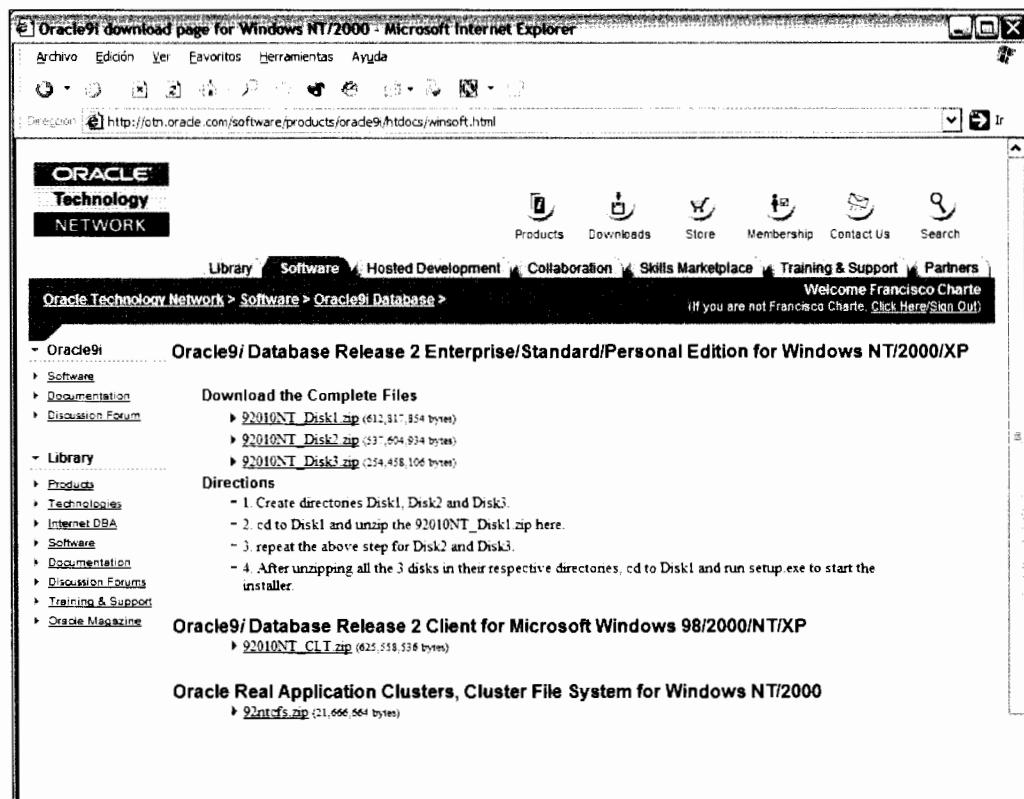


Figura 17.1. Podemos obtener Oracle9i desde la Web de OTN

A continuación se describe la instalación sobre Windows XP, los pasos pueden diferir si elige Oracle9i para otro sistema operativo.

Instalación de Oracle9i

El proceso de instalación de Oracle9i es bastante largo, si bien nuestras intervenciones en el proceso se limitarán, en la mayoría de los casos, a elegir una opción y pulsar el botón **Siguiente** para ir avanzando.

En este punto puede ver un resumen del proceso y ver, paso a paso, las páginas que debería mostrarle el asistente.

Tras iniciar el instalador, ejecutando el archivo `setup.exe` del primer CD, deberá aparecer el Instalador universal de Oracle. Éste le permite, como se aprecia en la figura 17.2, tanto instalar nuevos productos como ver los que tiene instalados o desinstalar aquellos en los que ya no esté interesado. En este caso, puesto que queremos instalar el RDBMS y los productos de los que depende, pulsaremos el citado botón **Siguiente**.

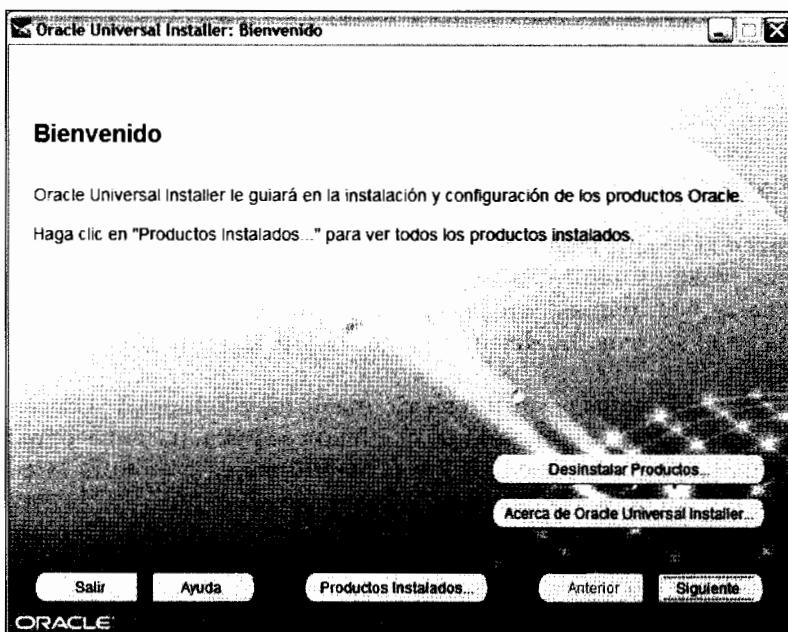


Figura 17.2. Ventana principal del instalador universal de Oracle

Lo primero que tenemos que hacer, antes incluso de elegir el producto que queremos instalar, es facilitar un nombre para el directorio raíz donde se efectuará la instalación, junto con el camino completo en el que se creará dicho directorio. De ser necesario, facilitaríamos en el apartado **Origen** el camino y nombre del archivo `.jar` donde se encuentra el índice de productos que hay en los CD. Esto habitualmente no es necesario.

El instalador propone un directorio raíz por defecto que es el que puede verse en la figura 17.3. Lo aceptamos y pulsamos de nuevo el botón **Siguiente**.

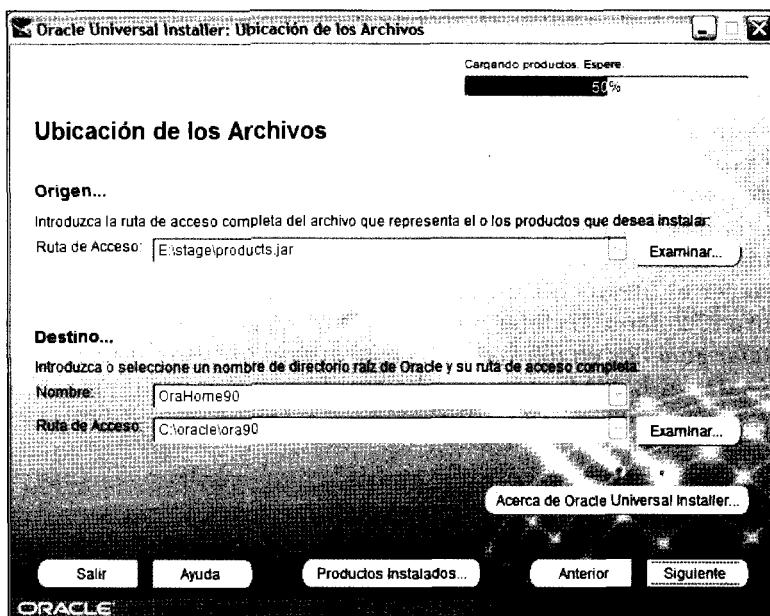


Figura 17.3. Selección del directorio raíz donde se efectuará la instalación

Finalizada la lectura del archivo `.jar`, el instalador nos mostrará la lista de opciones que tenemos a nuestra disposición. En este caso son tres (véase figura 17.4), de las cuales nos interesa, en principio, la primera, instalando la base de datos completa en el sistema. La dejamos marcada y luego pulsamos nuevamente el botón **Siguiente**.

El paso siguiente nos permitirá elegir una de las ediciones de Oracle9i, *Enterprise*, *Estándar* o *Personal*. Dependiendo de la edición se instalarán más o menos servicios y herramientas. Para el objetivo que perseguimos es indistinta la edición por la que optemos, puede ser cualquiera de las tres disponibles.

Al tiempo que se instala el RDBMS, el asistente de instalación también puede crear una base de datos evitándonos ese trabajo posterior. En la ventana que aparece en la figura 17.5 puede ver que aparecen distintas opciones. Si no quiere crear una base de datos, sino instalar el software sin más, elija **Sólo Software**. Las demás opciones crean una base de datos, optimizándola para el procesamiento masivo de transacciones, el almacenamiento de datos para análisis o un uso general. Dejamos marcada la primera opción, que aparece elegida por defecto, y pulsamos el botón **Siguiente**.

Las bases de datos Oracle se identifican con un nombre global y un SID o identificador simple dentro del servidor. Es la información que tenemos que facilitar en el paso siguiente. A medida que introduzcamos el nombre global veremos que el mismo asistente propone un SID que, normalmente, aceptaremos. En este caso (véase figura 17.6) asignamos el nombre global `Libros.AnayaMultimedia.es` a la base de datos, con el SID `Libros`.

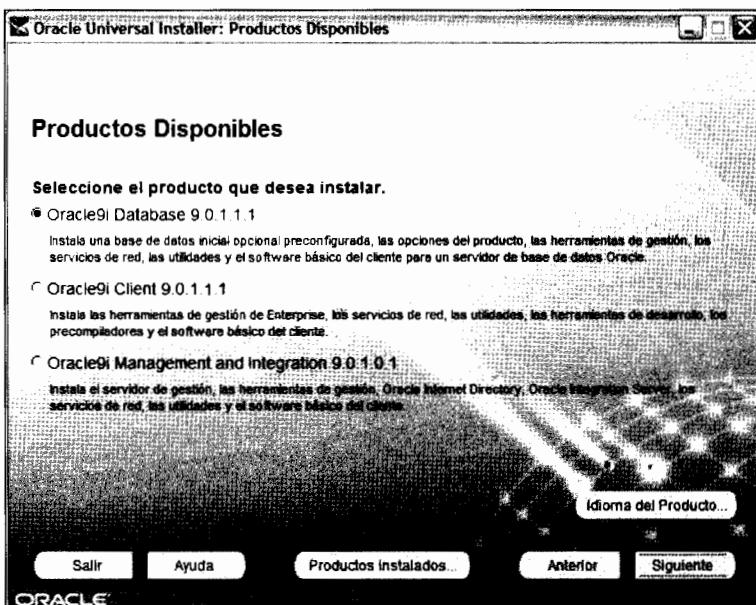


Figura 17.4. Elegimos el software que deseamos instalar

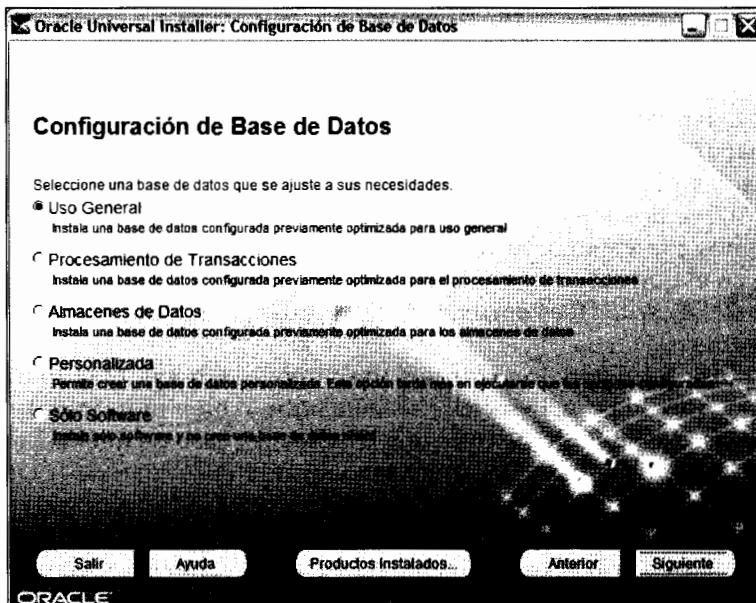


Figura 17.5. Optamos por crear una base de datos para uso general

La ubicación de las bases de datos que vayan creándose suele ser el directorio oracle\oradata de la unidad donde se haya instalado Oracle. El siguiente paso

del asistente, no obstante, nos permite especificar el directorio que deseamos emplear, pudiendo dejar el indicado por defecto o cambiarlo.

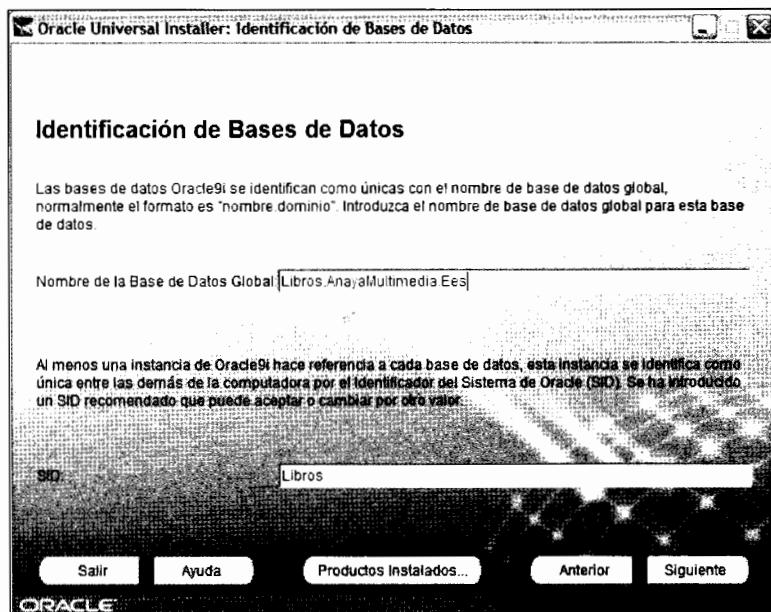


Figura 17.6. Identificamos la base de datos que va a crearse

El paso siguiente será la elección del conjunto de caracteres que se utilizará para almacenar la información en la base de datos. Tendremos tres opciones: el juego de caracteres por defecto del sistema, el juego de caracteres Unicode o bien cualquier otro que nos interese seleccionándolo de una lista.

Llegamos a la ventana resumen de la instalación, mostrada en la figura 17.7. Es el punto en el que debemos revisar todos los parámetros y a continuación pulsar el botón **Instalar**, si todo es correcto, o bien retroceder y cambiar aquellas opciones que deseemos.

Tras pulsar el botón **Instalar** nos limitaremos a ir cambiando el CD-ROM del producto, a medida que lo solicite el instalador, o bien indicar el directorio donde se encuentra el contenido de cada uno de ellos, en caso de que hayamos dejado los paquetes descomprimidos en directorios individuales.

Al final de la copia de archivos se pondrá en marcha la configuración de las distintas herramientas. Se iniciará el servidor HTTP de Oracle (una versión personalizada de Apache), el agente inteligente de Oracle y, como se aprecia en la figura 17.9, se invocará al asistente de configuración de bases de datos.

El proceso de generación de la base de datos, en el que prácticamente no tendremos que intervenir, irá abriendo algunas ventanas de consola que se cerrarán automáticamente, no tiene que actuar sobre ellas. El proceso de creación de la base de datos quedará reflejado en una pequeña ventana (véase figura 17.10). Se copia-

rán del CD-ROM los archivos de una base de datos tipo, poniendo en marcha un servicio Windows que será el que atienda las solicitudes de conexión.

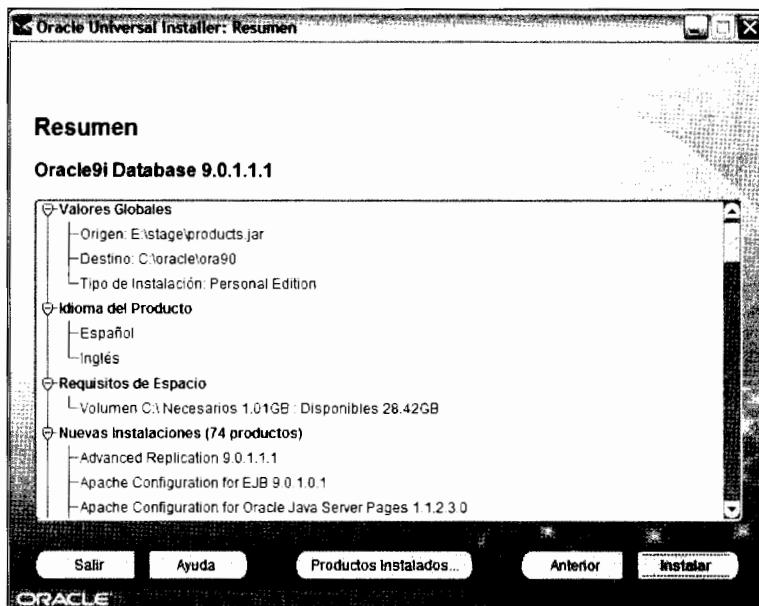


Figura 17.7. Resumen previo al inicio del proceso de instalación

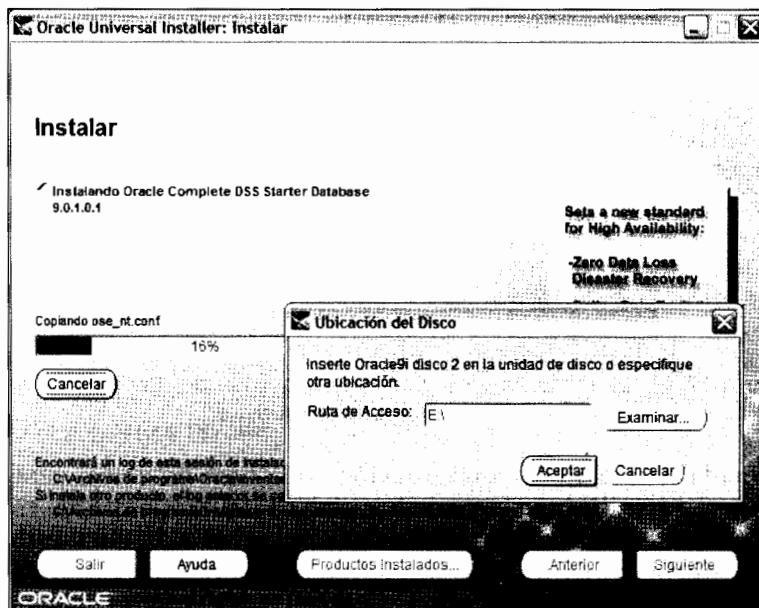


Figura 17.8. Proceso de copia de archivos y configuración

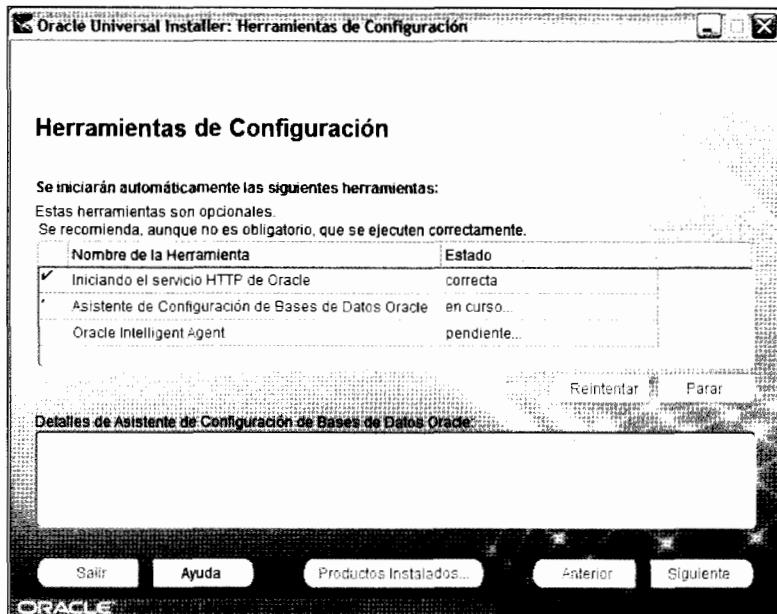


Figura 17.9. Configuración de las herramientas instaladas

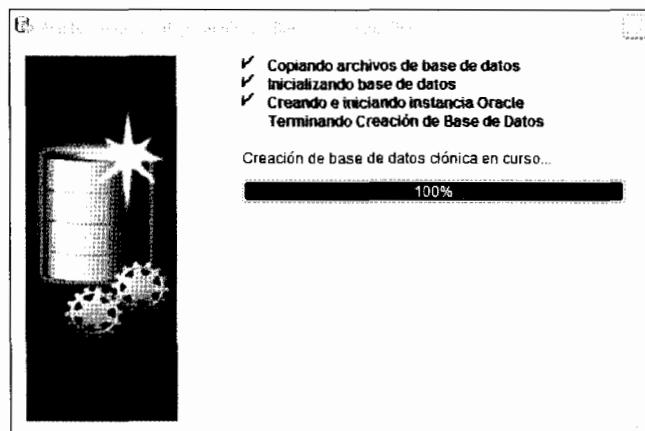


Figura 17.10. El asistente crea y configura la base de datos automáticamente sin intervención nuestra

Terminada la creación y configuración de la base de datos, verá aparecer una pequeña ventana indicándole el nombre global, el SID y la contraseña de las cuentas SYS y SYSTEM, dándole opción a modificarlas con el botón **Gestión de Contraseñas**. Pulse el botón **Salir** de esa ventana. Aparecerá la de confirmación de fin de instalación del instalador universal de Oracle. Pulse de nuevo el botón **Salir**. Ya tiene Oracle instalado en su sistema.

Administración del servidor

Una vez instalado el RDBMS, seguramente necesitará configurar parámetros de red, seguridad, etc. Oracle instala un gran número de asistentes y herramientas, en contraposición a productos como SQL Server en los que prácticamente todas las operaciones se encuentran centralizadas. Todos ellos puede encontrarlos en la carpeta Oracle - HoraHome90 de Inicio>Todos los programas. En la figura 17.11 puede ver algunos de esos elementos, concretamente la subcarpeta correspondiente a herramientas de configuración y conversión.

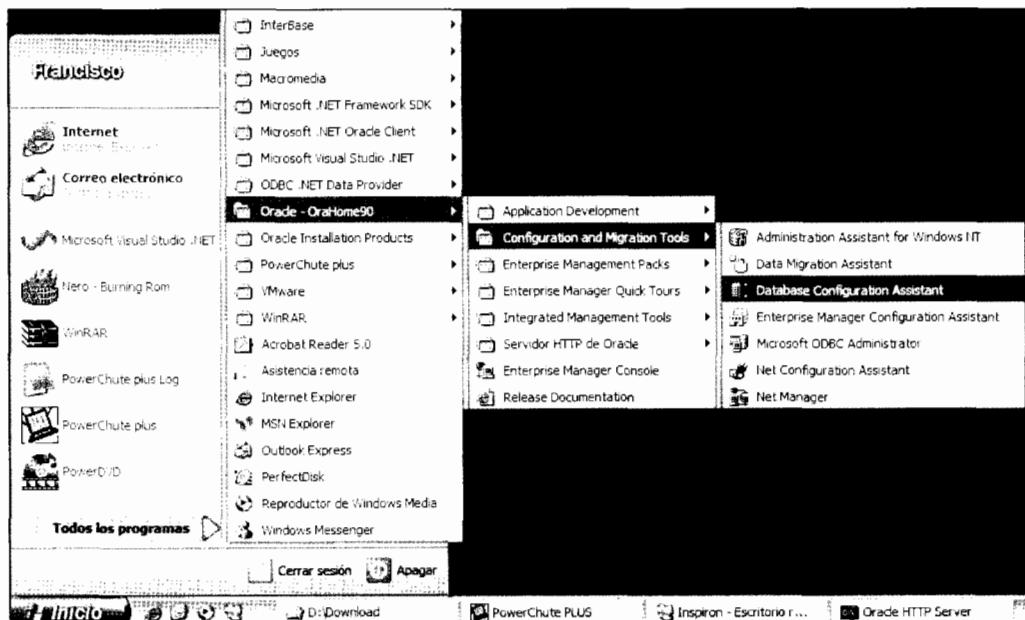


Figura 17.11. Oracle9i instala multitud de herramientas y asistentes

Muchos de los aspectos relacionados con cuentas, perfiles y administración de la base pueden realizarse con el **Oracle Administration Assistant for Windows NT** (véase figura 17.12). Abra el servidor, la rama **Bases de datos** y seleccione el SID de la base creada durante la instalación. Haga clic sobre ella con el botón secundario del ratón y elija la opción de conexión. A partir de ese momento podrá acceder a las listas de usuarios, perfiles y administradores.

Identificación de servicio y esquema

Para poder conectar con una cierta base de datos Oracle, es indispensable conocer el nombre del servicio. Éste se define en el momento de la creación en el servidor, siendo necesaria también una definición en los clientes. Nuestra base de datos

inicial, generada por el asistente, es reconocida en el propio servidor con el nombre de servicio *Libros*.

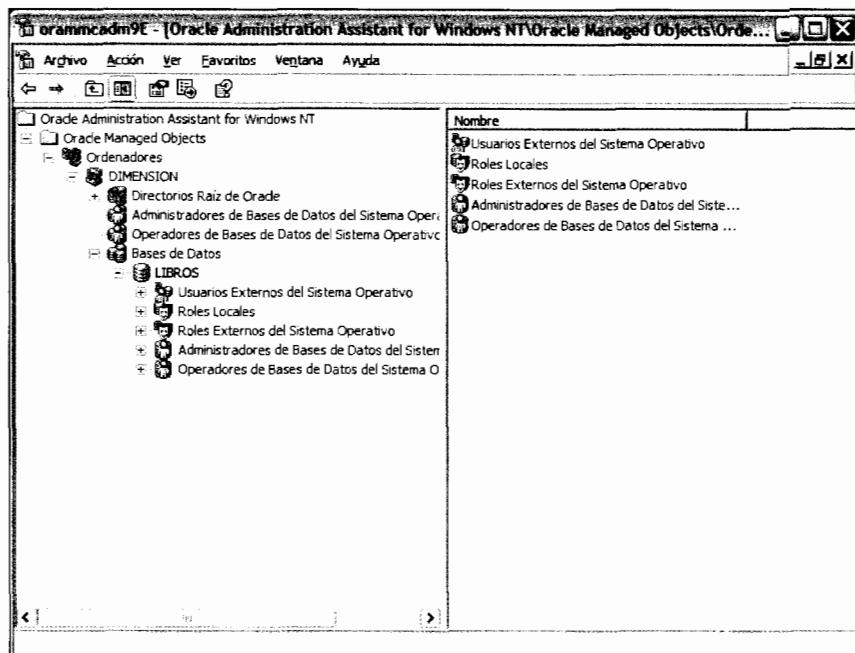


Figura 17.12. Consola de administración de Oracle

Cada base de datos cuenta con una serie de esquemas en los que se alojan los distintos elementos, como tablas, vistas o procedimientos almacenados. Existe un esquema por defecto, *scott*, que hemos usado en todos los ejemplos de los capítulos previos en los que se utilizaba la base de datos Oracle creada en el tercer capítulo. Puede crear su propio esquema y personalizarlo, si no quiere emplear éste.

El software cliente

Desde un sistema en el que se instale la plataforma Microsoft .NET, indispensable para ejecutar cualquier aplicación creada con Visual Basic .NET, es posible acceder a múltiples orígenes de datos sin necesidades adicionales, ya que también se instalan los MDAC. Esto significa que podemos usar una base de datos Access, una hoja de cálculo Excel o comunicarnos con un servidor SQL Server sin necesitar ningún software adicional.

No es posible, sin embargo, una comunicación directa con todos los productos RDBMS disponibles en el mercado, especialmente cuando son de diversos fabricantes y, por lo tanto, emplean distintos medios de comunicación entre cliente y

servidor. Por ello es necesario instalar en cada ordenador cliente el software que facilite el fabricante del RDBMS, en este caso el software cliente de Oracle9i.

Nota

El software cliente debe instalarlo en la máquina de desarrollo tan sólo si no es la misma en que esté ejecutándose el RDBMS, caso en el que no es necesario. Si debe instalarlo en todos los ordenadores cliente donde vaya a utilizarse la aplicación.

La instalación de dicho software cliente se inicia con el mismo proceso explicado en el punto dedicado a la instalación de Oracle9i, simplemente eligiendo la opción Oracle9i Client en lugar de Oracle9i Database en el paso del asistente que aparece en la figura 17.4. Al finalizar el proceso, tendrá en el ordenador los elementos necesarios para comunicarse con el RDBMS Oracle9i remoto.

Definición del servicio

Con algunos RDBMS, como es el caso de SQL Server, basta con instalar el software cliente para, desde una aplicación, conectar con una base de datos que se encuentra en el servidor. Para ello ese cliente debe indicar el nombre o dirección del ordenador que actúa como servidor, así como la base de datos que se desea abrir. Estos datos corresponden con los parámetros Data Source e Inicial Catalog de la cadena de conexión.

Sin embargo, al usar Oracle el cliente no tiene porqué conocer ni el servidor donde se ejecuta el RDBMS ni la base de datos, el valor asignado al parámetro Data Source es el nombre de servicio local con el que se conoce a la base de datos. Ese nombre de servicio hay que definirlo, usando para ello el Asistente de Configuración de Red de Oracle que se instala como parte del software cliente. Este proceso se describió en el sexto capítulo, puede recurrir a él para recordarlo.

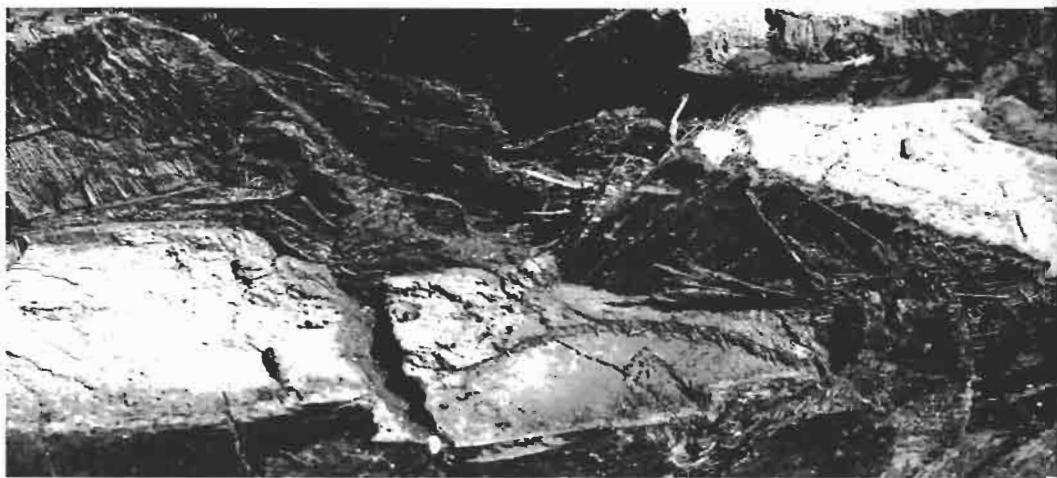
Instalación del proveedor

Ya tenemos en el equipo del cliente el software cliente instalado y el nombre del servicio definido. Para que nuestra aplicación funcione, sin embargo, es necesario un elemento más: el proveedor ADO.NET OracleClient. Éste no forma parte de la instalación estándar actual de la plataforma Microsoft .NET, a diferencia de SqlClient y OleDb, por lo que debe ser instalado en cada sistema donde vaya a utilizarse para acceder a una base de datos Oracle.

Podemos redistribuir el archivo oracle_net.msi como un módulo adicional que es necesario al instalar nuestra aplicación, de manera que esté ahí disponible

cuando haga falta. Nuestra aplicación usaría los objetos de este proveedor, según se ha descrito en los capítulos de la segunda parte, facilitando en la cadena de conexión no el nombre del servidor RDBMS o del servicio remoto, sino el nombre de servicio local que ha sido definido para acceder a esa base de datos. Además puede ser necesario facilitar el nombre de esquema/usuario, por ejemplo scott, y la clave de acceso correspondiente.

Con esto ya tendría su aplicación Visual Basic .NET, funcionando sobre Windows y accediendo a un RDBMS Oracle9i, sin importar que éste se ejecute también sobre Windows o cualquier otro sistema operativo. Como puede ver, el proceso para acceder a la base de datos más usada desde el lenguaje de programación más popular no es demasiado complejo.



18

Control
de transacciones

Al operar sobre sistemas RDBMS, una parte de los orígenes de datos que hemos usado en capítulos previos lo son, es habitual utilizar transacciones que aseguren la integridad de los datos, especialmente cuando se actúa sobre múltiples tablas.

En ninguno de los ejemplos propuestos hasta ahora nos hemos ocupado de las transacciones, dejando el aseguramiento de esa integridad en manos de ADO.NET y el RDBMS con el que se conectaba. En ocasiones, sin embargo, puede ser preciso un control explícito del inicio y fin de la transacción, momento en el cual necesitaremos emplear el objeto `Transaction` del proveedor adecuado tal y como se explica en los puntos siguientes.

Razón de ser de las transacciones

Todos los sistemas RDBMS considerados como tales, Oracle y SQL Server son dos ejemplos, contemplan el uso explícito de transacciones. Una transacción asegura que todas las operaciones efectuadas en su ámbito, desde que se inicia hasta que se cierra, se ejecutan de forma satisfactoria o no se ejecuta ninguna de ellas. Esto nos garantiza la integridad de la información almacenada en la base de datos.

Imagine que, en una aplicación Visual Basic .NET, codifica una serie de adaptadores de datos y `DataSet` para actuar sobre dos tablas: una que contiene encabezados de factura y otra que almacena las filas de detalle. Al efectuar parte de la actualización, luego de ejecutar el método `Update()` del primer adaptador que,

pongamos por caso, es el de las líneas de detalle, se produce un fallo de alimentación o una interrupción en las comunicaciones, de tal forma que se han enviado al servidor las líneas de detalle pero al intentar el `Update()` para enviar la cabecera se produce un error. En este momento en la base de datos existe una inconsistencia patente, dado que no es posible encontrar los datos del cliente y la factura a la que pertenecen las líneas. Para evitar esta situación debe utilizarse una transacción. Ésta se iniciaría antes de la primera actualización y terminaría tras la última. En ese ámbito las operaciones se registran de forma temporal, confirmándose si todas se reciben satisfactoriamente o descartándose si se produce cualquier evento que impida mantener la integridad.

الآن، يُمكنكم تجربة تطبيق **Smart Home** على هواتفكم الذكية.

Una forma de asegurar la integridad de los datos en operaciones como la descripta, y que se dan constantemente en la mayoría de aplicaciones, consiste en emplear el sistema de transacciones de la propiedad base de datos. Para ello lo habitual es codificar procedimientos almacenados en la base de datos, de tal forma que las aplicaciones, cuando necesitan efectuar alguna operación, invocan al procedimiento almacenado facilitándole los parámetros necesarios, ocupándose éste de realizar la manipulación de las tablas.

El control de transacciones en la base de datos se efectuará con las sentencias propias del lenguaje de cada RDBMS, por ejemplo PL/SQL en Oracle o T-SQL en SQL Server. En el caso de T-SQL las sentencias a conocer son estas tres: BEGIN TRANSACTION, COMMIT TRANSACTION, ROLLBACK TRANSACTION. La primera inicia una nueva transacción, la segunda la confirma y la tercera la revoca. Dado que en SQL Server las transacciones pueden anidarse, cada una de estas sentencias irá seguida de un identificador propio.

Suponga que quiere tener en la base de datos SQL Server creada como ejemplo en el tercer capítulo, un procedimiento almacenado al que facilitándole el código de una editorial elimine todos los libros que le pertenecen y la propia editorial. Esto requiere dos sentencias DELETE y, para evitar que la operación pudiese interrumpirse entre ellas, las introducimos en una transacción. El procedimiento almacenado podría ser el siguiente:

```
CREATE PROCEDURE EliminaEditorial
    @editorial INTEGER
AS
BEGIN TRANSACTION Transaccion
    DELETE FROM Libros WHERE Editorial=@editorial
    DELETE FROM Editoriales WHERE IDEditorial=@editorial
ROLLBACK TRANSACTION Transaccion
```

Observe que al final del método no se utiliza una sentencia COMMIT, que sería lo habitual para confirmar la operación, sino ROLLBACK, de tal manera que las dos

sentencias anteriores quedan canceladas aunque, en apariencia, parezca que se ejecutan. Puede comprobarlo ejecutando el procedimiento desde el **Analizador de consultas SQL** de SQL Server, como se ha hecho en la figura 18.1. Fíjese en el panel de resultados, donde se indica el número de filas afectadas en ambas tablas. Si las abre, sin embargo, comprobará que no se ha eliminado fila alguna.

Desde una aplicación Visual Basic .NET podríamos ejecutar este procedimiento almacenado, dejando el control de las transacciones directamente en manos del servidor de datos.

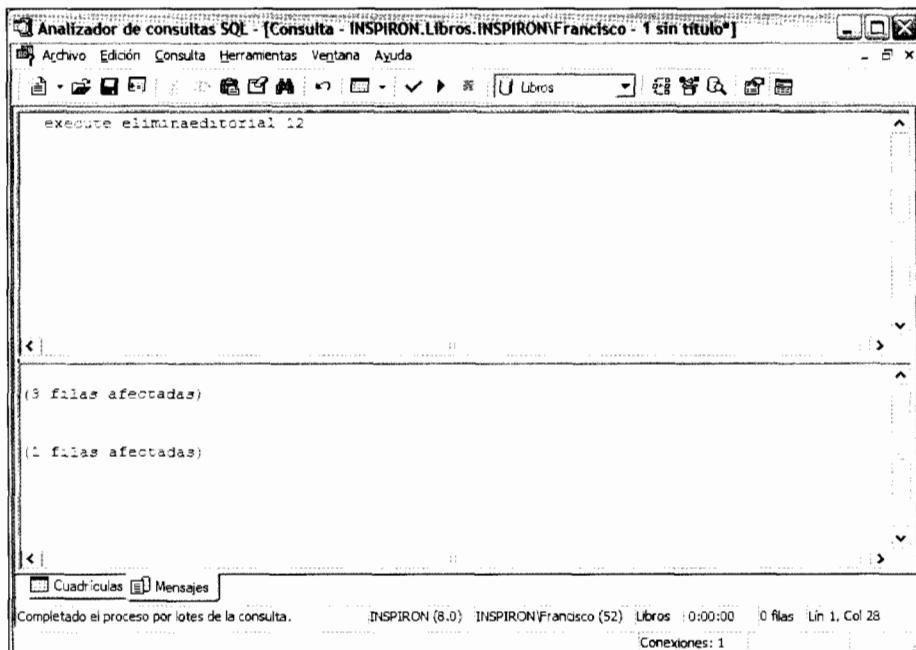


Figura 18.1. Aparentemente el procedimiento almacenado ha eliminado filas en ambas tablas

Transacciones en Visual Basic .NET

Las transacciones también pueden ser controladas de manera explícita desde el código de un programa Visual Basic .NET, empleando para ello el objeto `Transaction` específico del proveedor que vayamos a utilizar para acceder al origen de datos: `SqlTransaction`, `OleDbTransaction`, `OracleTransaction` u `OdbcTransaction`.

Las cuatro clases mencionadas tienen en común la implementación de la interfaz `IDbTransaction`. Ésta cuenta tan sólo con cuatro miembros, dos propiedades y dos métodos:

- **Connection:** Establece el vínculo entre este objeto `Transaction` y un objeto `Connection`.
- **IsolationLevel:** Fija el nivel de aislamiento entre esta transacción y las demás que pudieran existir en el RDBMS sobre los mismos elementos.
- **Commit():** Confirma la transacción.
- **Rollback():** Revoca la transacción.

No necesitamos más para controlar desde un método Visual Basic un proceso similar al explicado en el punto previo a modo de ejemplo.

Creación del objeto `Transaction`

Para crear una transacción asociada a una cierta conexión, como puede deducirse de los miembros existentes en la interfaz `IDbTransaction`, bastaría con asignar a la propiedad `Connection` la referencia a la conexión. Una alternativa es emplear el método `BeginTransaction()` de la interfaz `IDbConnection`, implementado en el objeto `Connection` de cada proveedor.

Ese objeto tiene que asociarse también con cada uno de los comandos que vayan a ser ejecutados y que se desea que queden bajo el ámbito de la transacción. Con este fin, en la interfaz `IDbConnection` hay definida una propiedad llamada `Transaction`.

Creada la transacción y enlazada tanto con la conexión como con los comandos, usaríamos el método `Commit()` o `Rollback()`, tras ejecutar todas las operaciones de actualización, para confirmar o revocar la transacción y, por tanto, todas las operaciones ejecutadas durante su tiempo de vida.

En la práctica

Veamos en la práctica cómo crear y controlar una transacción desde un programa Visual Basic .NET, empleando para ello el proveedor de SQL Server.

El método `BeginTransaction()` de `SqlConnection`, así como los métodos `Commit()` y `Rollback()` de `SqlTransaction`, en realidad hacen uso de la gestión de transacciones propia del RDBMS, mediante las sentencias comentadas anteriormente.

Comenzaremos insertando en un formulario los componentes que pueden verse en la figura 18.2. Hemos arrastrado la tabla `Libros` de la base de datos SQL Server desde el Explorador de servidores hasta el diseñador, utilizando el menú emergente de `SqlDataAdapter1` para generar el conjunto de datos `DsLibros1`. Los principales elementos de interfaz son un cuadro de texto para introducir el código de una editorial, un botón para seleccionar las filas, que aparecerán en el `DataGridView` que hay debajo, y un segundo botón que eliminaría esas filas del origen de datos.

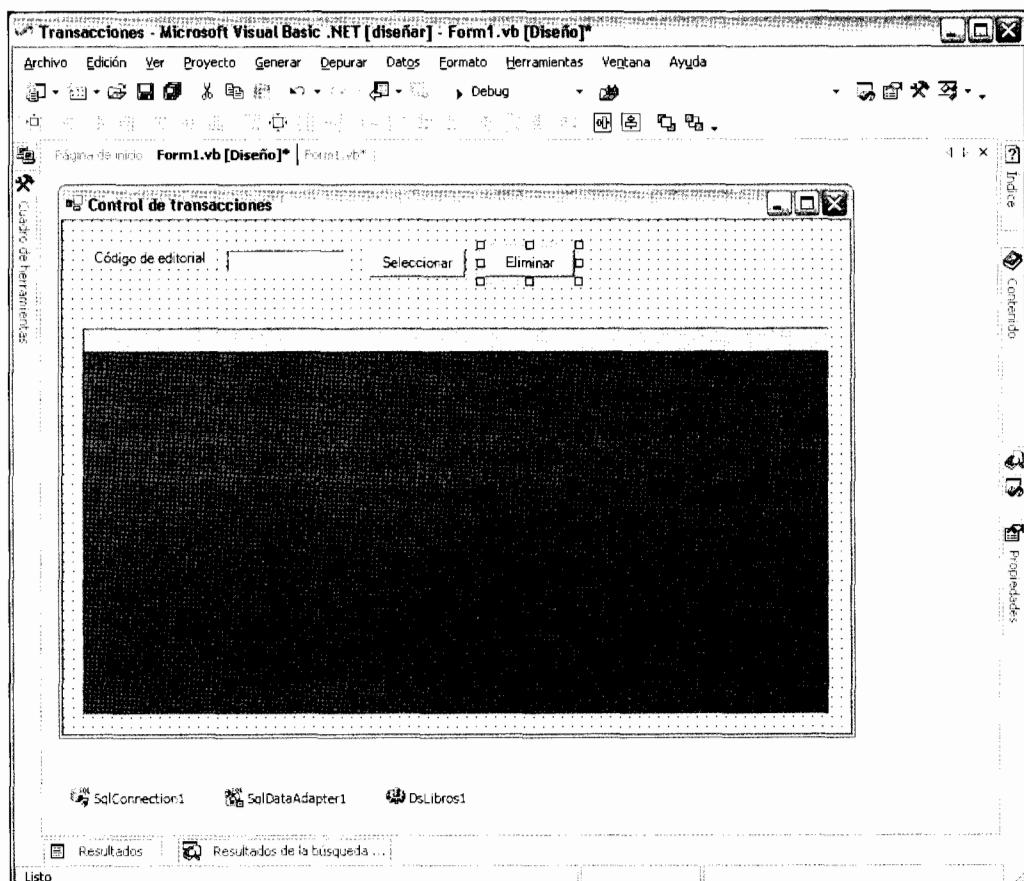


Figura 18.2. Diseño del formulario

En principio no existe ningún vínculo entre los elementos de interfaz y los componentes de acceso a datos, vínculo que se establecerá al pulsar el botón **Seleccionar**. Éste ejecuta el código siguiente:

```
' Al pulsar el botón Seleccionar
Private Sub btnSeleccionar_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles btnSeleccionar.Click

    ' Recogemos los datos actuales de la tabla libros
    SqlDataAdapter1.Fill(DsLibros1, "Libros")

    ' y creamos una vista con los títulos de la editorial
    ' que se haya indicado en el TextBox
    Dim Vista As DataView = New _
        DataView(DsLibros1.Libros, "Editorial=" & _
        tbCodigoEditorial.Text, "Titulo", _
        DataViewRowState.CurrentRows)
```

```
' Enlazamos el DataGrid con la vista  
dgLibros.DataSource = Vista  
End Sub
```

Como puede ver, llenamos el DataSet usando el adaptador de texto, creando a continuación una vista en la que aparezcan sólo los títulos de la editorial cuyo código se haya introducido en el TextBox. Finalmente, enlazamos esa vista con el DataGrid, que mostraría las filas que van a eliminarse en caso de que pulsemos el botón **Eliminar**.

El proceso de eliminación de las filas es sencillo, basta con recorrer la vista e ir llamando al método `Delete()` de cada uno de los elementos y llamar al método `Update()` del correspondiente adaptador de datos. Eso es, básicamente, lo que vamos a hacer al pulsar el botón **Eliminar**, pero delimitando la operación dentro de una transacción y confirmándola o revocándola según lo que responda el usuario a una última pregunta de confirmación. El código asociado al evento `Click` del botón sería el siguiente:

```
' Al pulsar el botón eliminar  
Private Sub btnEliminar_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles btnEliminar.Click  
    Dim IndFila As Integer  
    ' Obtenemos la referencia a la vista vinculada con el DataGrid  
    Dim Vista As DataView = CType(dgLibros.DataSource, DataView)  
    Dim Transaccion As IDbTransaction  
  
    ' Abrimos la conexión  
    SqlConnection1.Open()  
    ' e iniciamos la transacción  
    Transaccion = SqlConnection1.BeginTransaction()  
  
    ' Eliminamos todas las filas de la vista  
    While Vista.Count > 0  
        Vista(0).Delete()  
    End While  
  
    ' Establecemos la transacción en el comando de borrado  
    SqlDataAdapter1.DeleteCommand.Transaction = Transaccion  
    ' y actualizamos el origen de datos  
    SqlDataAdapter1.Update(Vista.Table)  
  
    ' Preguntamos si realmente se quieren borrar estas filas  
    If MessageBox.Show("¿Realmente desea eliminar estas filas?", _  
        "Transacciones", MessageBoxButtons.YesNo) = _  
        DialogResult.Yes Then  
        ' en caso afirmativo confirmamos la transacción  
        Transaccion.Commit()  
    Else ' en caso negativo la revocamos  
        Transaccion.Rollback()  
    End If  
  
    ' Limpiamos el DataSet  
    DsLibros1.Clear()
```

```
' y la actualizamos con los datos de la tabla
SqlDataAdapter1.Fill(DsLibros1, "Libros")

' Cerramos la conexión
SqlConnection1.Close()
End Sub
```

Obtenemos una referencia a la vista que se creó previamente para vincularla al DataGridView, efectuando la conversión de tipo necesaria. Luego abrimos la conexión e iniciamos la transacción, borramos las filas, asociamos la transacción con el DeleteCommand del adaptador e invocamos al método Update() de éste. En ese momento las filas desaparecerán del DataGridView e, incluso, no podría acceder a ellas desde otra aplicación o desde el Administrador corporativo de SQL Server. Se encuentran bloqueadas a la espera de que la transacción se confirme o descarte.

Que llamemos al método Commit() o Rollback() dependerá de la respuesta del usuario. Independientemente de cuál se ejecute, terminamos refrescando la información del DataSet y cerrando la conexión. Si se ha confirmado, el DataGridView permanecerá vacío ya que las filas se habrán borrado, mientras que de revocarse el DataGridView volverá a mostrar las mismas filas.

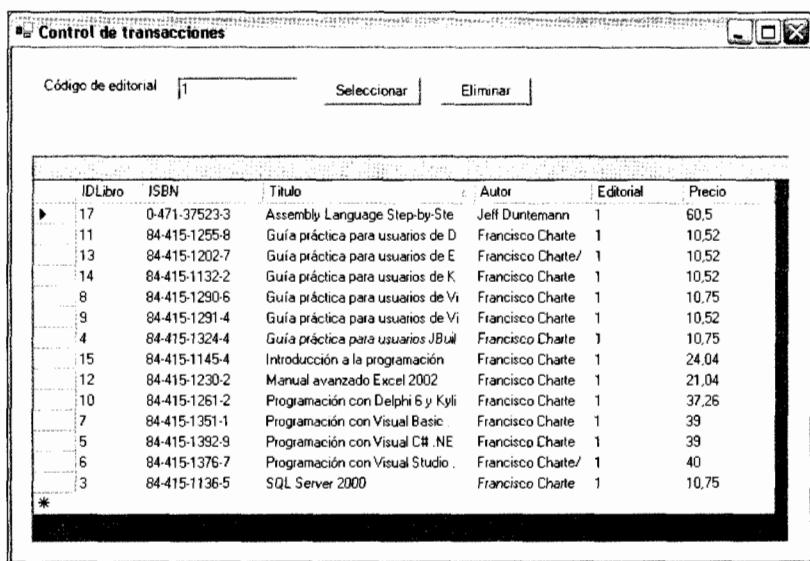
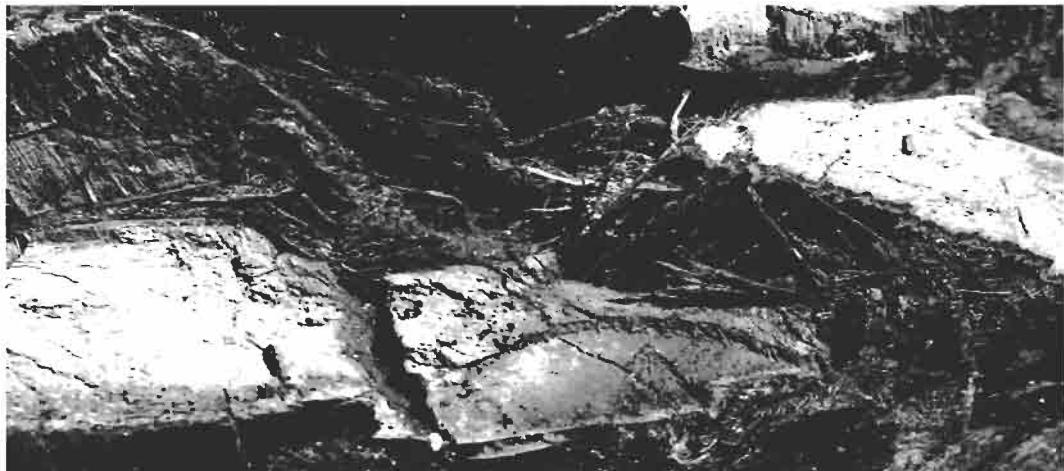


Figura 18.3. El programa mostrando la lista de filas que van a eliminarse

De manera análoga podríamos controlar con transacciones explícitas cualquier otra situación, generalmente en la que se encontrasen implicadas dos o más tablas que deban mantener una integridad referencial, ya que en este caso hubiese bastado con pedir la confirmación antes de eliminar las filas y actualizar con el adaptador de datos, sin necesidad de usar una transacción.



19

Resolución de problemas de concurrencia

El empleo de transacciones asegura la integridad de los datos, según ha podido verse en el capítulo previo, pero no resuelve por sí solo los problemas de concurrencia que pueden presentarse. Por defecto, ADO.NET utiliza una política de bloqueos optimista, lo cual significa que los datos se recuperan en un DataSet pero cualquier otro usuario puede acceder a ellos de manera simultánea. Puede darse el caso, por tanto, de que varias aplicaciones cliente, generalmente en distintos ordenadores distantes entre sí, estén mostrando la misma información e, incluso, permitan al usuario modificarlos.

¿Qué ocurre si un usuario modifica o elimina alguna de esas filas y, a continuación, otro usuario intenta modificar esa fila, ya cambiada o inexistente? Por defecto se genera una excepción, impidiendo al segundo usuario hacer cambios en una fila cuyo estado ha cambiado desde que la recuperó en el DataSet.

El objetivo de este capítulo es mostrarle, con un ejemplo, cómo responder a esta situación para poder resolverla de forma satisfactoria, sin perder directamente los cambios que haya efectuado el usuario pero sin escribir tampoco los cambios sobre el origen perdiendo la información que otro cliente haya introducido.

Políticas de bloqueo y actualización

A la hora de codificar una aplicación en la que van a obtenerse datos que, posteriormente, deben ser actualizados, hay que elegir una política de bloqueo y de

actualización. Básicamente existen dos opciones: bloqueo optimista y bloqueo pesimista. Hay una tercera, aunque generalmente no es aceptable en la mayoría de aplicaciones, a la que se hace referencia como *el último gana*.

El bloqueo pesimista consiste en bloquear las filas que presumiblemente van a actualizarse desde el mismo momento en que leen de la base de datos, lo cual significa que ningún otro cliente puede tener acceso a ellas hasta que el proceso de modificación y actualización termine. Es una política que requiere mantener una conexión abierta con el servidor todo el tiempo que dure ese proceso, porque de lo contrario no podría mantenerse el bloqueo sobre las filas. Sus inconvenientes son muchos, siendo el más importante que sólo un usuario puede tener acceso a las filas en cada momento, lo cual no suele ser aceptable en aplicaciones que se ejecutan en multitud de clientes contra una misma base de datos.

Nota

En ADO.NET no existe la opción de usar una política de bloqueo pesimista ya que, por naturaleza, se opera sin una conexión persistente con el servidor. Es posible simular ese comportamiento, no obstante, iniciando una transacción antes de recuperar las filas y manteniéndola hasta después de la actualización.

La política de bloqueo optimista sólo bloquea las filas que van a ser modificadas o eliminadas y sólo en el momento de la actualización. Así, en caso de que un programa, como los usados a modo de ejemplo en capítulos previos, recupere una lista de los títulos existentes en la tabla *Libros*, esto no implicará que ningún otro cliente pueda obtener esos mismos datos. No es necesario mantener una conexión continua con el servidor, ya que el bloqueo se establecería sólo en el momento de ejecutar la sentencia UPDATE, no al seleccionar datos. El bloqueo optimista tiene el problema antes planteado: ya que varios usuarios pueden acceder a las mismas filas, hay que controlar la posibilidad de que dos o más de ellos efectúen cambios en la misma fila. ADO.NET nos facilita los eventos y excepciones necesarias para ello.

En cuanto a la política de *el último gana*, es tan simple como asumir que en la base de datos sólo permanecerán los datos que se escriban en último lugar, es decir, si varios usuarios modifican una misma fila, se perderán todos los cambios excepto los del último en enviarlos al RDBMS. Como se decía antes, suele ser una opción inaceptable en la mayoría de los casos.

Información de retorno durante la actualización

Cuando se crea un adaptador de datos, ya sea directamente al arrastrar una tabla desde el Explorador de servidores o tal y como se describió en los capítulos de

la segunda parte, se utiliza un objeto `CommandBuilder` que se encarga de generar las sentencias de actualización y eliminación de filas. Éstas contienen todas las condiciones necesarias para garantizar el bloqueo optimista, comprobando que los valores que tenía la fila originalmente son los que tiene en el momento de la actualización, no habiéndose modificado ninguno de ellos porque se haya producido otra actualización en paralelo.

En el momento en que se llama al método `Update()` de un adaptador de datos, facilitando como parámetro la tabla que tiene las filas modificadas, se recorren las filas una a una y se ejecuta una sentencia `UPDATE` cada vez que se encuentre una cuyo estado indique que ha sufrido cambios. El RDBMS ejecuta la actualización y confirma a ADO.NET el éxito o el fracaso, generándose a continuación el evento `RowUpdated` del adaptador de datos.

El segundo parámetro que acompaña a ese evento contiene un indicador de estado, mediante el cual puede saberse si se ha efectuado la actualización o producido un error; un mensaje de error, una referencia a la fila afectada, otra al comando ejecutado, etc. En caso de que la propiedad `RecordsAffected` sea cero y `Status` contenga el error `UpdateStatus.ErrorOccurred`, sabremos que se ha producido un error que ha impedido la actualización de la fila. Razón: los valores `DataRowVersion.Original` de la fila que tenemos en el programa no coinciden con los valores que existen en la base de datos.

Si no modificamos el valor de la propiedad `Status`, y dejamos que la ejecución continúe, se producirá una excepción. Podemos notificar el problema al usuario y modificar dicha propiedad, impidiendo la generación de la excepción.

Un primer acercamiento

Veamos cómo aprovechar el evento `RowUpdated` para ofrecer información sobre el proceso de actualización, aunque sin resolver el problema puesto que los cambios efectuados por el usuario que no se hayan podido escribir en el origen se perderán, al menos en este primer acercamiento a la solución final.

Partimos diseñando un formulario como el que aparece en la figura 19.1, con un `DataGridView` y un botón como únicos elementos de interfaz. En la parte inferior del diseñador puede ver la conexión, adaptador y conjunto de datos, generados con el Explorador de servidores. Vinculamos el `DataGridView` con la única tabla del `DataSet`.

Al abrirse el formulario se ejecutará la sentencia `SqlDataAdapter1.Fill(ds.Libros1, "Libros")`, mostrando así las filas de la tabla en la cuadrícula desde un primer momento.

Cuando se pulse el botón **Actualizar** procederemos a llamar al método `Update()` del adaptador, refrescando el contenido del `DataSet` para obtener los cambios que pudiesen haberse ejecutado desde otros puestos:

```
' Al pulsar el botón Actualizar
Private Sub btnActualizar_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnActualizar.Click
```

```

' ejecutamos la actualización
SqlDataAdapter1.Update(DsLibros1.Libros)
DsLibros1.Clear() ' eliminamos el contenido del DataSet
' y lo refreshcamos
SqlDataAdapter1.Fill(DsLibros1, "Libros")
End Sub

```

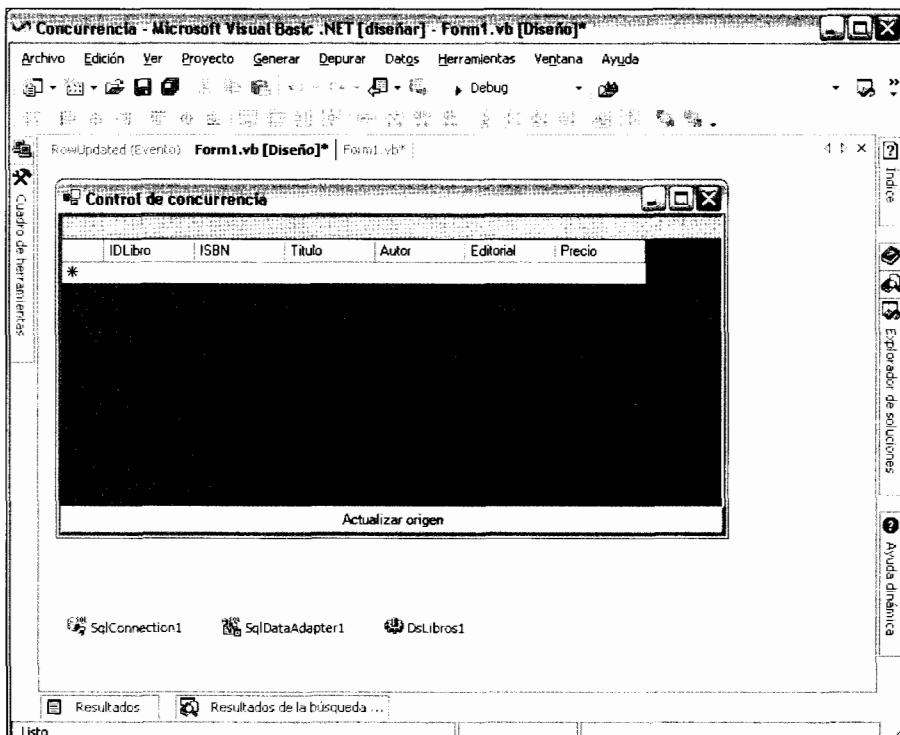


Figura 19.1. Aspecto del formulario con la cuadrícula y el botón

Si no hemos cambiado ningún dato, los datos de la cuadrícula mostrarán cambios que otros clientes puedan haber efectuado, pero no se desencadenará en ningún momento el evento RowUpdated. Esto tan sólo ocurrirá una vez por cada fila modificada, ejecutándose el código siguiente:

```

' Cada vez que se actualice una fila
Private Sub SqlDataAdapter1_RowUpdated(ByVal sender As Object, _
 ByVal e As System.Data.SqlClient.SqlRowUpdatedEventArgs) _
 Handles SqlDataAdapter1.RowUpdated
' comprobamos si hay errores
If e.Status = UpdateStatus.ErrorsOccurred Then
' en caso afirmativo los mostramos
MessageBox.Show(e.Errors.Message & vbCrLf &
e.Row.Item("Autor", DataRowVersion.Original) & _
 vbCrLf & e.Row.Item("Autor", DataRowVersion.Current))
End If

```

```

' y saltamos la fila actual continuando con la
' actualización de otras que pudieran haber sido cambiadas
e.Status = UpdateStatus.SkipCurrentRow
End If
End Sub

```

Como puede ver, comprobamos si el estado indica la existencia de un error, caso en el que mostramos por la consola el mensaje, el valor original que tenía la fila y el que nosotros pretendíamos darle en una de las columnas, que es la que pretendemos modificar a modo de ejemplo. Por último asignamos el valor `UpdateStatus.SkipCurrentRow` a la propiedad `Status`, impidiendo la generación de la excepción `DBConcurrencyException` al tiempo que se ignoran los cambios de la fila.

Si ejecuta el programa, efectúa cualquier cambio y pulsa el botón **Actualizar** todo irá bien. Para provocar un error tendrá que ejecutar dos copias del programa y cambiar la misma fila, o bien hacer alguna modificación desde el Administrador corporativo. En esos casos nos encontraremos con una situación como la de la figura 19.2. Observe el mensaje de error.

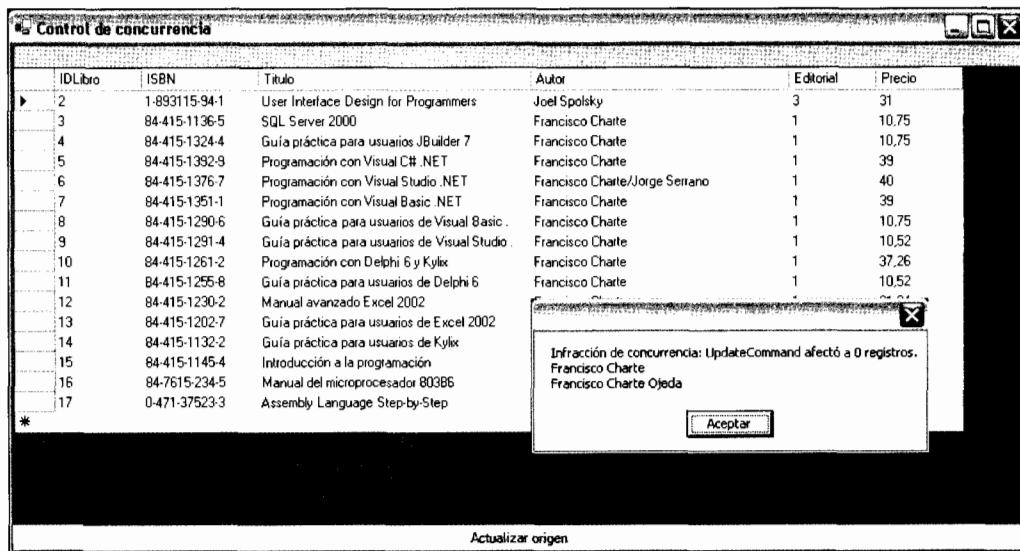


Figura 19.2. Al actualizar se ha encontrado un problema de concurrencia

Fusión de los cambios

El ejemplo anterior es útil porque notifica al usuario que ha surgido un problema de concurrencia y, por lo tanto, los cambios que ha efectuado se perderán. No obstante, se le obliga a, tras haber actualizado los datos que tenía en el `DataGridView`,

volver a introducirlos si se quiere ejecutar la modificación. Podemos mejorar este comportamiento dando los pasos siguientes en caso de que se genere un error:

- Obtención en un nuevo DataSet con la información actual que hay en la base de datos.
- Informar al usuario del valor actual, el que había originalmente y el que ha introducido él, preguntándole si quiere sobrescribir la información que hay actualmente en la base de datos.
- En caso afirmativo, se utilizaría el método Merge() del conjunto de datos original para unirlo con el que acaba de recuperarse, facilitando como segundo parámetro el valor True para preservar los cambios que haya hecho el usuario.
- Nueva llamada al método Update() del adaptador para provocar la nueva actualización.

Tomando como base el ejemplo del punto anterior, modifique el método asociado al evento RowUpdate para que, en lugar de mostrar el mensaje de error, invoque al siguiente método Resolucion() facilitando como parámetro e.Row, es decir, la fila que ha provocado el error.

```
' Método de resolución de problemas de concurrencia
Private Sub Resolucion(ByVal Fila As DataRow)
    ' Creamos un nuevo DataSet
    Dim DatosActuales As New dsLibros()

    ' Para recuperar los datos actuales de la base de datos
    SqlDataAdapter1.Fill(DatosActuales, "Libros")
    ' buscamos la fila que ha provocado el error
    Dim FilaOriginal As dsLibros.LibrosRow =
        DatosActuales.Libros.FindByIDLibro(Fila.Item("IDLibro"))

    ' Preparamos una cadena de caracteres con el valor actual
    ' de la columna Autor en la base de datos, el valor que
    ' tenía originalmente y el que nosotros queremos darle
    Dim Mensaje As String =
        String.Format("Valor actual='{0}', " &
            "Valor original='{1}', Valor propuesto='{2}'", _
            FilaOriginal.Autor, _
            Fila.Item("Autor", DataRowVersion.Original), _
            Fila.Item("Autor", DataRowVersion.Current))

    ' Preguntamos al usuario si desea sobreescribir los cambios
    ' hechos en la base de datos con los suyos
    If MessageBox.Show(Mensaje, "¿Desea sobreescribir cambios?", _
        MessageBoxButtons.YesNo) = DialogResult.Yes Then
        ' En caso afirmativo combinamos los DataSet
        ' indicando que se conserven los cambios
        DsLibros1.Merge(DatosActuales, True)
        ' y volvemos a llamar a Update
        SqlDataAdapter1.Update(DsLibros1.Libros)
```

```

    ' aceptando los cambios de esta fila
    Fila.AcceptChanges()
Else ' en caso contrario
    Fila.RejectChanges() ' rechazamos los cambios
End If
End Sub

```

Observe que se muestra tan sólo el valor de la columna Autor, asumiendo que, para realizar pruebas, va a modificarse esa columna y no otra. En la práctica, en lugar de este simple mensaje sería mucho más adecuado preparar un formulario en el que el usuario pudiese ver todos los valores que él ha introducido y los que hay en la base de datos actualmente, pudiendo elegir qué quiere hacer.

También debe tenerse en cuenta que este método, al ser invocado desde el método RowUpdate, se ejecuta asumiendo que va a pulsarse el botón **Actualizar** tras modificar cada fila. Esto no tiene necesariamente que ser así, ya que el usuario podría modificar varias filas. Podría automatizarse la llamada al método Update() del adaptador cada vez que se detectase un cambio de fila comprobando que en ella se han efectuado cambios. Otra posibilidad es no usar el evento RowUpdate sino interceptar el evento DBCurrencyException al llamar al método Update(). Si éste se genera, podríamos recorrer el conjunto de filas que tienen un error y mostrarlas todas al usuario de una sola vez, en lugar de consultar una a una.

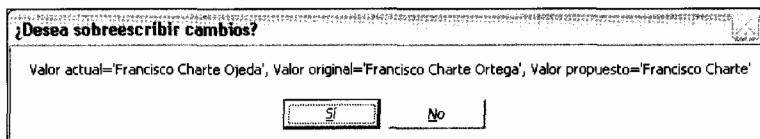
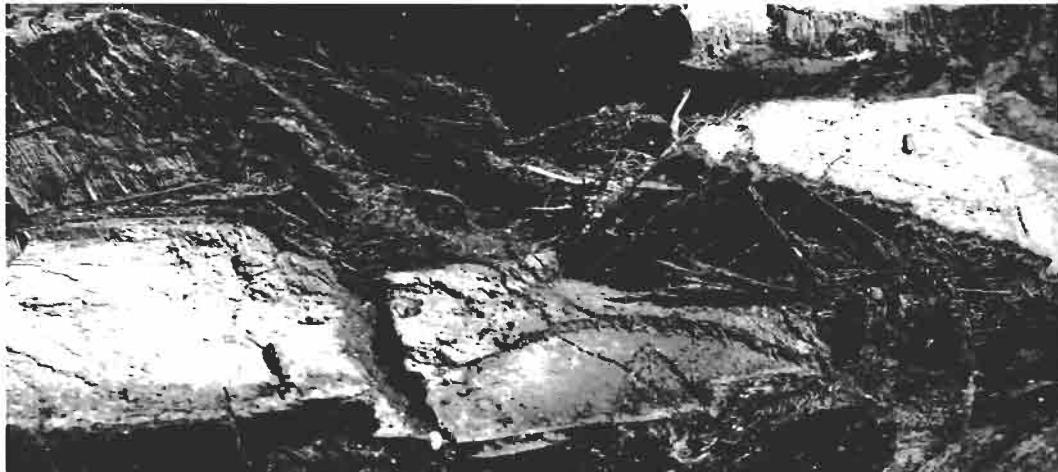


Figura 19.3. Mensaje de notificación y consulta del programa al encontrar un error de concurrencia

Nota

Este mecanismo de control de concurrencia es el adecuado cuando se opera con componentes DataSet y controles vinculados a él. Si estamos usando otros medios, como DataReader o ejecución directa de consultas, no contaremos con los comandos de bloqueo optimista generados por los asistentes, teniendo que controlar nosotros los problemas que pudiesen surgir.

En la documentación electrónica de Visual Basic .NET podrá encontrar algunas alternativas, como la inserción en todas las tablas de una columna que mantenga una referencia temporal de cuándo se modificó por última vez la fila. Al leerla se cuenta con esa referencia, y al actualizar habría que cambiarla siempre con la condición de que no se haya modificado porque ello significaría que otro usuario la ha actualizado.



20

Tablas
con columnas
calculadas

Cuando se ejecuta un comando de selección de datos, ya sea obteniendo un `DataReader` o bien mediante un adaptador para generar un `DataSet`, siempre obtenemos un conjunto de filas predefinidas en las tablas de la base de datos y con valores que están almacenados en ellas. En ocasiones, sin embargo, pueden ser necesarios datos adicionales que pueden calcularse a partir de los datos de las columnas, no siendo lógico añadir nuevas columnas en las tablas de la base de datos para almacenarlos.

Suponga que necesita mostrar en un `DataGridView` el título de cada libro, el autor, el precio, el IVA y el precio neto. En la tabla `Libros` tan sólo existen las tres primeras columnas, pero no el IVA o el precio neto. No tiene sentido añadir esas columnas a las tablas de la base de datos, ya que ocuparían un espacio innecesario al poder calcular sus valores.

En este capítulo verá cómo puede definir esas columnas calculadas, ya sea en la propia sentencia SQL o añadiendo una columna al `DataTable`, para evitar tener que calcular y añadir esos valores manualmente, mediante código.

Columnas calculadas en la sentencia SQL

La mayoría de RDBMS permiten la inclusión en la sentencia de selección de operaciones sobre columnas existentes en las filas de datos, apareciendo el resultado como si fuese otra columna. En la expresión pueden emplearse los habituales

operadores aritméticos y relacionales y, según el RDBMS, algunas funciones intrínsecas. En la siguiente sentencia de ejemplo se calcula el IVA creando una nueva columna llamada `IVA`, tomando como base la columna `Precio` ya existente:

```
SELECT Titulo, Autor, Precio, Precio * 0.04 AS IVA FROM Libros
```

Al estar definida en la propia sentencia, la columna aparece a ojos de los `DataReader` y `DataAdapter` como si de una columna más se tratase aunque, como es lógico, sería una columna sólo de lectura, al no existir físicamente en la base de datos. Para realizar una prueba sencilla y rápida, dé los pasos siguientes:

- Inicie una nueva aplicación Windows.
- Arrastre desde el Explorador de servidores la tabla `Libros` de la base de datos hasta el diseñador.
- Haga clic con el botón secundario del ratón sobre el adaptador de datos, eligiendo la opción **Configurar adaptador de datos**.
- Modifique la sentencia de selección, dejándola como puede verse en la figura 20.1. Observe la introducción de la columna `IVA`.
- Pulse el botón **Finalizar** y genere el conjunto de datos a partir del adaptador.
- Vincule el `DataSet` con un `DataGridView`.
- Introduzca en el método asociado al evento `Load` del formulario la llamada al método `Fill()` del adaptador. El formulario, al ejecutar el programa, debería aparecer como se muestra en la figura 20.2.

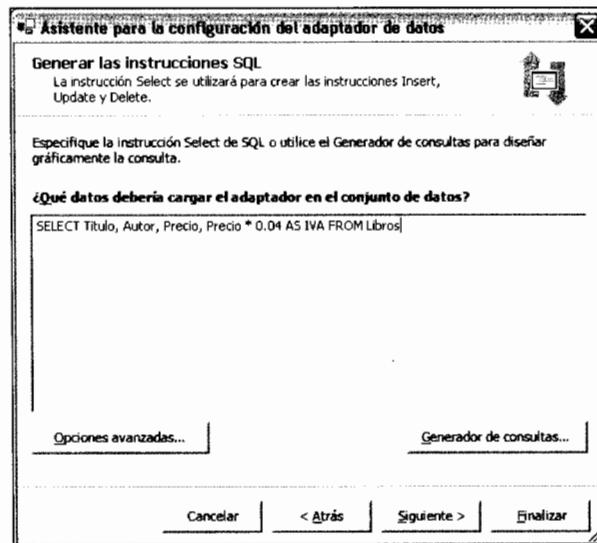


Figura 20.1. Modificamos la sentencia de selección del adaptador de datos

Título	Autor	Precio	IVA	IDLibro
Assembly Language Step-by-Step	Jeff Duntemann	60,5	2,42	17
Guía práctica para usuarios de Delphi 6	Francisco Charte	10,52	0,4208	11
Guía práctica para usuarios de Excel 2002	Francisco Charte/M.Jesús Luque	10,52	0,4208	13
Guía práctica para usuarios de Kylix	Francisco Charte	10,52	0,4208	14
Guía práctica para usuarios de Visual Basic .NET	Francisco Charte	10,75	0,43	8
Guía práctica para usuarios de Visual Studio .NET	Francisco Charte	10,52	0,4208	9
Guía práctica para usuarios JBuilder 7	Francisco Charte Ojeda	10,75	0,43	4
Introducción a la programación	Francisco Charte	24,04	0,9616	15
Manual avanzado Excel 2002	Francisco Charte	21,04	0,8416	12
Manual del microprocesador 80386	Chris H.Pappas&William H.Murray	40	1,6	16
Programación con Delphi 6 y Kylix	Francisco Charte	37,26	1,4904	10
Programación con Visual Basic .NET	Francisco Charte	39	1,56	7
Programación con Visual C# .NET	Francisco Charte	39	1,56	5
Programación con Visual Studio .NET	Francisco Charte/Jorge Serrano	40	1,6	6
SQL Server 2000	Francisco Charte Ojeda	10,75	0,43	3
User Interface Design for Programmers	Joel Spolsky	31	1,24	2

Figura 20.2. El formulario mostrando los valores de la columna calculada

Nota

Consulte la información de referencia del RDBMS que esté usando para saber qué operadores y funciones puede emplear en una sentencia de selección para crear columnas calculadas.

Añadir columnas a un DataTable

Como ya sabe, los objetos `DataTable` disponen de una colección, accesible mediante la propiedad `Columns`, compuesta de objetos `DataColumn`, cada uno de los cuales representa a una columna recuperada de las tablas del origen de datos. En múltiples ejemplos hemos recorrido esa colección para mostrar todas las columnas de una tabla o conocer sus atributos.

Lo más interesante, es que la colección a la que apunta `Columns` no es estática, siendo posible añadir nuevos objetos `DataColumn` según se necesite. Además, esos nuevos `DataColumn` no tienen necesariamente que representar a una columna de la base de datos, pudiendo, en su lugar, alojar una expresión.

Creación de un nuevo DataColumn

Al crear un `DataColumn` podemos usar distintos constructores sobrecargados que aceptan diferentes listas de parámetros. Con ellos es posible crear una nueva columna sin establecer propiedad alguna o bien, por el contrario, asignar valor a las más relevantes. Las propiedades que más nos interesarán, en este caso, son las siguientes:

- ColumnName: Nombre de la nueva columna, usado por defecto como título en controles como DataGridView.
- DataType: Tipo de dato que contendrá la columna. Al crearla nueva es necesario indicarlo.
- Expression: Si la columna no va a estar vinculada con una columna de base de datos, en esta propiedad se facilitará la expresión con la que obtendrá su valor.

Para añadir al DataSet del ejemplo anterior, concretamente al DataTable llamado Libros, una columna con el precio neto, sumando las columnas Precio e IVA, podríamos usar la sentencia siguiente:

```
DsLibros1.Libros.Columns.Add("Neto",
    GetType(System.Decimal), "Precio+IVA")
```

Puede colocar la sentencia en el mismo método asociado al evento Load del formulario. Si lo ejecuta, verá que ahora en el DataGridView aparece una columna, llamada Neto, con el precio neto, tal y como puede ver en la figura 20.3. En este caso la columna no existe en la sentencia, y por ello no aparecerá como propiedad de dsLibros.

Título	Autor	Precio	IVA	Neto
User Interface Design for Programmers	Joel Spolsky	31	1,24	32,24
SQL Server 2000	Francisco Charle Djeda	10,75	0,43	11,18
Guía práctica para usuarios JBuilder 7	Francisco Charle Djeda	10,75	0,43	11,18
Programación con Visual C# .NET	Francisco Charle	39	1,56	40,56
Programación con Visual Basic .NET	Francisco Charle/Jorge Se	40	1,6	41,6
Programación con Visual Basic .NET	Francisco Charle	39	1,56	40,56
Guía práctica para usuarios de Visual Basic	Francisco Charle	10,75	0,43	11,18
Guía práctica para usuarios de Visual Stud	Francisco Charle	10,52	0,4208	10,9408
Programación con Delphi 6 y Kylix	Francisco Charle	37,26	1,4904	38,7504
Guía práctica para usuarios de Delphi 6	Francisco Charle	10,52	0,4208	10,9408
Manual avanzado Excel 2002	Francisco Charle	21,04	0,8416	21,8816
Guía práctica para usuarios de Excel 2002	Francisco Charle/M.Jesús	10,52	0,4208	10,9408
Guía práctica para usuarios de Kylix	Francisco Charle	10,52	0,4208	10,9408
Introducción a la programación	Francisco Charle	24,04	0,9616	25,0016
Manual del microprocesador 80386	Chris H. Pappas&William H	40	1,6	41,6
Assembly Language Step-by-Step	Jeff Duntemann	60,5	2,42	62,92

Figura 20.3. El DataGridView mostrando la nueva columna añadida al DataTable

Creación en fase de diseño

Las columnas calculadas, objetos DataColumn, pueden crearse mediante código, como acabamos de hacer, o bien en fase de diseño mediante el diseñador de esquemas XSD. Al abrir el esquema asociado a un DataSet no sólo podemos ver las

tablas que lo componen, o sus columnas, sino que también podemos introducir cambios. Además, esos cambios se traducen automáticamente en modificaciones a la clase derivada de `DataSet`, con lo que las nuevas columnas aparecerían como propiedades adicionales de ésta.

Partiendo del anterior, dé los pasos siguientes para ver un sencillo ejemplo:

- Elimine la sentencia introducida en el punto anterior con la que se creaba el `DataTable` con el precio neto.
- Haga clic con el botón secundario del ratón sobre el `DataSet`, en el diseñador de formularios, y elija la opción **Ver esquema**. Debe encontrarse en el diseñador de esquemas (véase figura 20.4).
- Pulse en la última fila de la tabla, que aparece como un recuadro en el diseñador, e introduzca directamente el nombre y tipo. También puede escribir y seleccionar esos datos en la ventana **Propiedades**.
- Escriba en la propiedad **Expression** la expresión de cálculo, en este caso **Precio+IVA**.

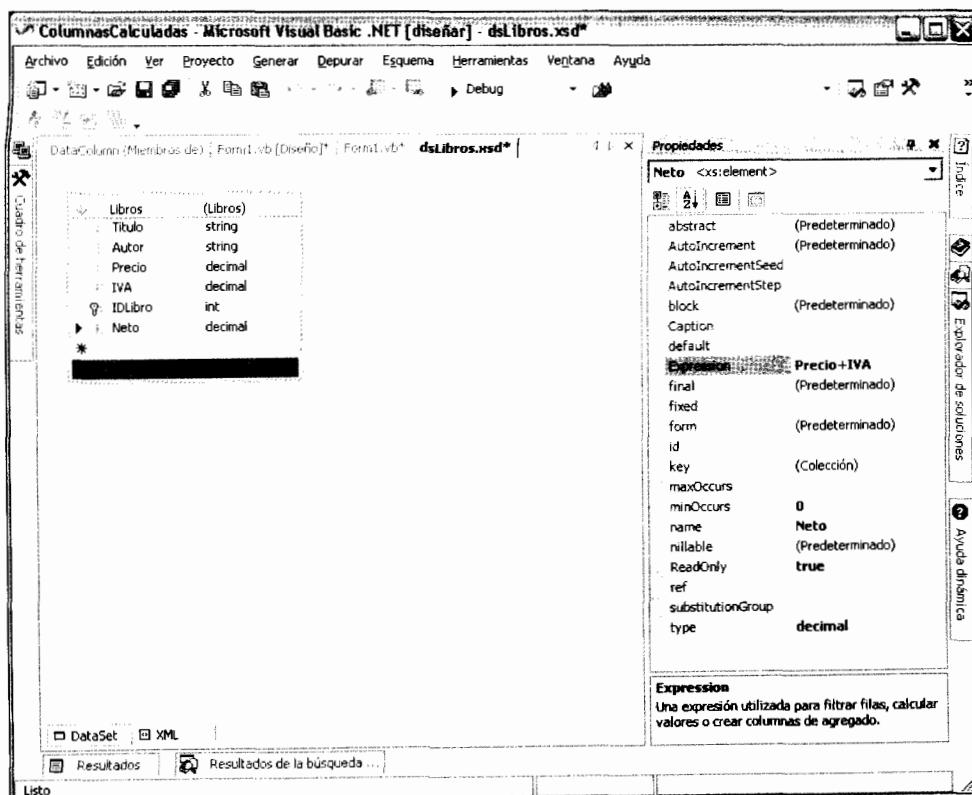


Figura 20.4. Modificamos el esquema usando la ventana Propiedades

Al compilar y ejecutar el programa verá que obtiene exactamente el mismo resultado que tenía en la figura 20.3, con la diferencia de que ahora no ha tenido que escribir código alguno. Aparte está la ventaja de que la nueva columna aparece como una propiedad de `dsLibros`, de tal manera que el editor de código la muestra como un elemento más. Esta ventaja no la teníamos al crear el `DataTableColumn` mediante código.

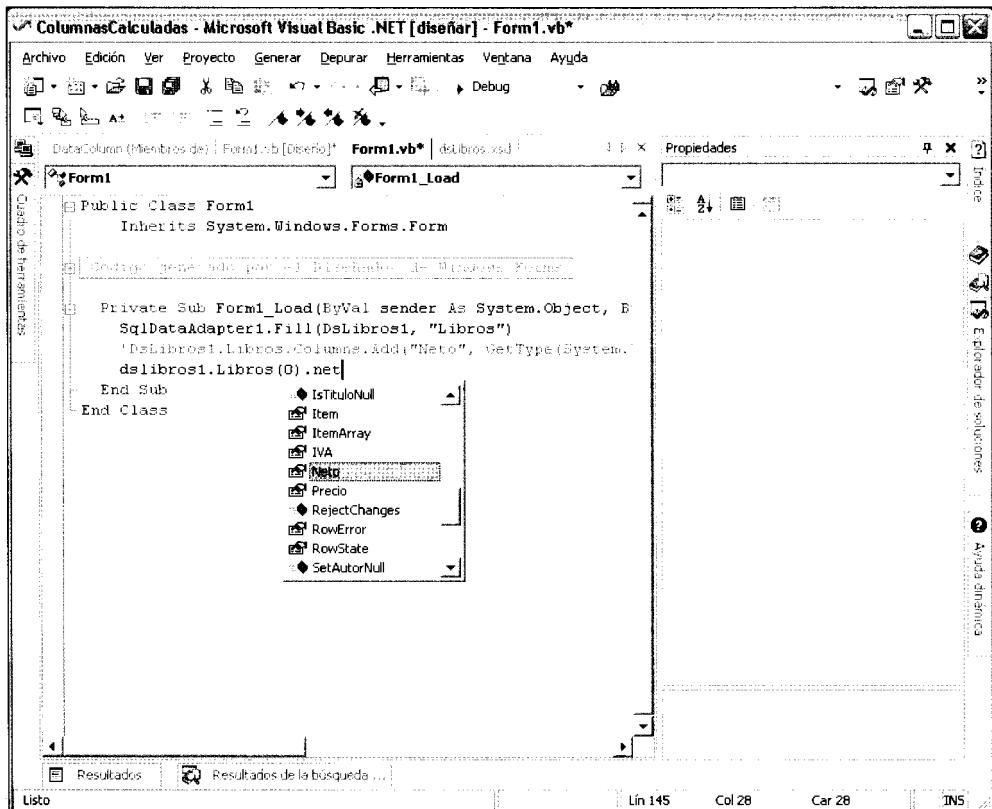
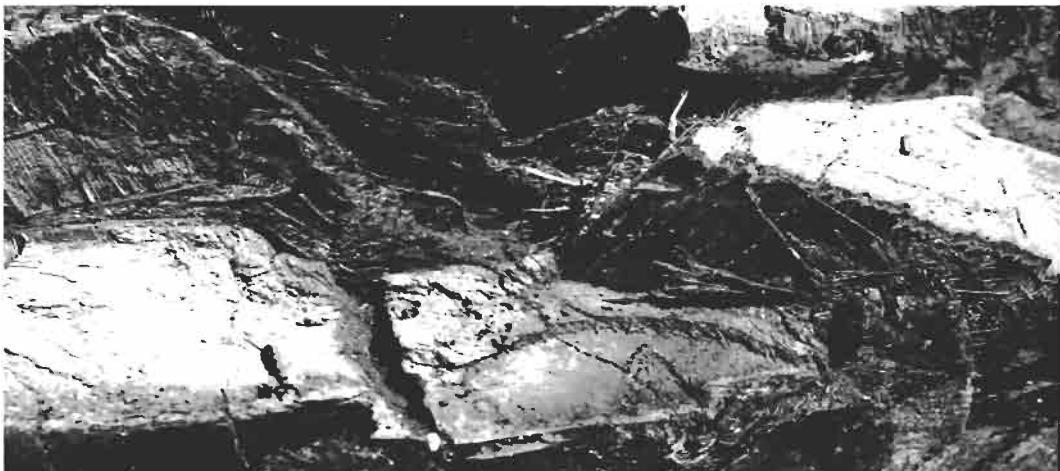


Figura 20.5. Al modificar el esquema se ha cambiado también la clase derivada de `DataSet`

Nota

Consulte la ayuda de la propiedad `Expression` de la clase `DataTableColumn` para saber qué operadores y funciones puede usar para crear sus expresiones.



21

Almacenamiento y recuperación de imágenes

Aunque en las bases de datos actuales la mayoría de las columnas contienen texto o números, no es extraño que, en ocasiones, se quiera almacenar también alguna imagen, como puede ser la portada de un libro o bien una fotografía de un producto. Esto permite, por ejemplo, que los clientes que acceden desde la Web puedan ver una imagen del objeto en que están interesados.

Casi todos los RDBMS cuentan con algún tipo de columna que facilita el almacenamiento de imágenes, pero su enlace y tratamiento de las aplicaciones no suele ser tan fácil como cabría esperar en primera instancia. SQL Server, por ejemplo, dispone de un tipo `image`, pero éste no se corresponde con ningún tipo de la plataforma .NET, y ni siquiera puede enlazarse directamente con un control `PictureBox`.

Los asistentes para creación de formularios de datos, tanto Windows como Web, no reconocen las columnas que contienen imágenes y, por tanto, no introducen controles para mostrarlas ni lógica para poder modificarlas. Por eso el objetivo de este capítulo es mostrarle cómo conseguir trabajar con imágenes desde una aplicación Visual Basic .NET, concretamente un formulario Windows con los datos de los libros.

Añadir una columna para la portada

Para empezar, tendremos que añadir a una de nuestras bases de datos una nueva columna, en la tabla `Libros`, para almacenar la portada. Ya sabe que puede

modificar la estructura de las tablas directamente desde Visual Studio .NET, por lo que no necesita recurrir a herramientas específicas del RDBMS.

En este caso vamos a tomar, una vez más, la tabla *Libros* de SQL Server, añadiéndole, como se aprecia en la figura 21.1, una columna llamada *Portada* de tipo *image*. Permitiremos los valores nulos. De hecho, al tener ya la tabla una serie de filas, el valor original de la columna *Portada* será *DBNull* en todas ellas. Si utiliza las *Visual Database Tools* para editar el contenido de la tabla, o incluso el *Administrador corporativo*, verá que la columna *Portada* aparece siempre como <Binario> (véase figura 21.2), no siendo posible ni ver su contenido ni modificarlo. La única opción, por tanto, es crear un programa a medida que facilite esas operaciones.

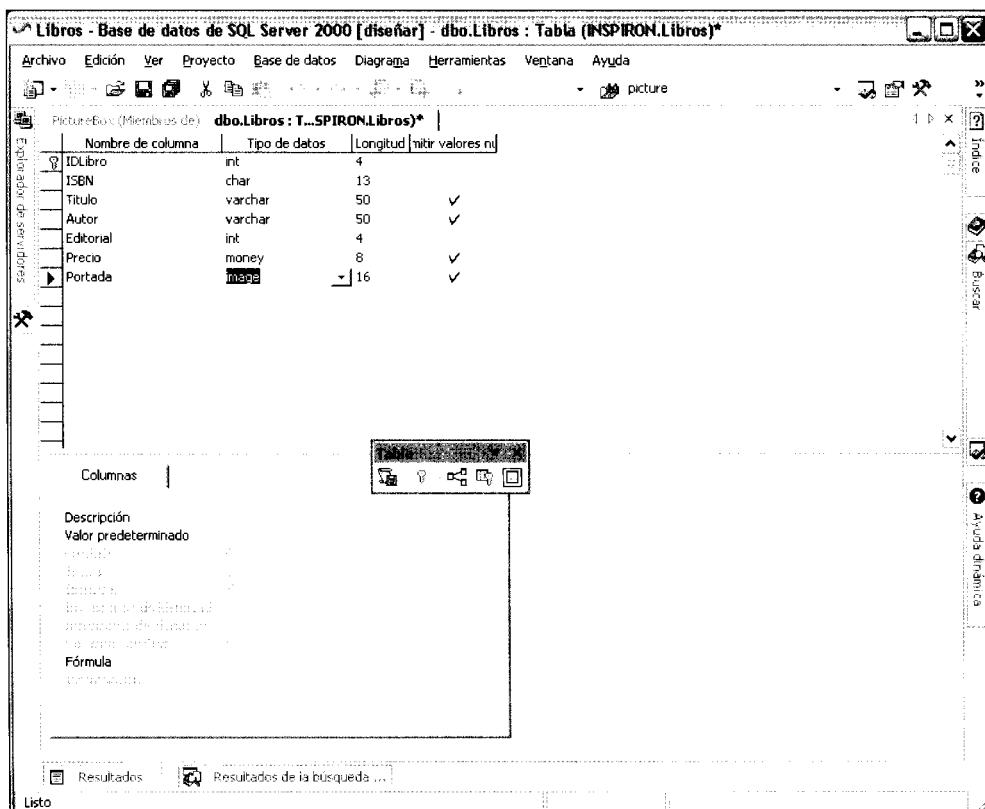


Figura 21.1. Definimos la nueva columna en la tabla *Libros*

Columnas binarias

Las columnas que almacenan imágenes, o cualquier otra información binaria, se tratan de forma similar desde Visual Basic .NET. Si crea un *DataSet* con tipo

tras añadir la columna Portada de la tabla Libros, simplemente arrastrándola desde el Explorador de servidores a un formulario y creando el conjunto de datos, verá que el tipo de la columna es base64Binary. Puede verlo en la figura 21.3, tanto en la representación visual de la tabla como en la ventana Propiedades.

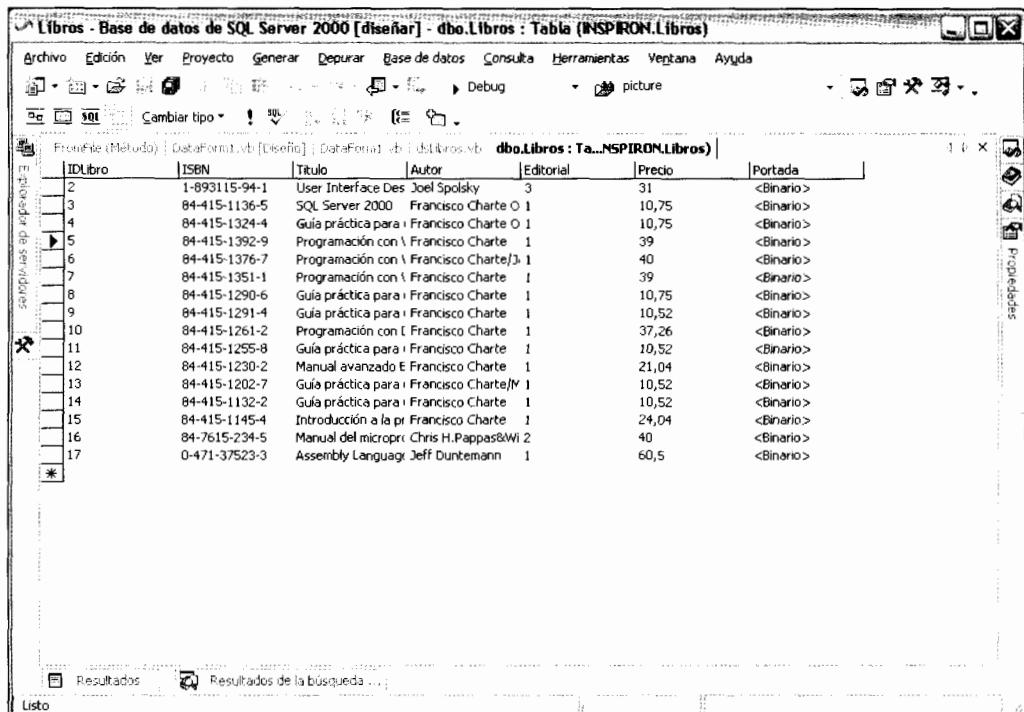


Figura 21.2. Edición de la tabla en Visual Studio .NET

A partir de este esquema, el generador de conjuntos de datos con tipo produce una serie de propiedades, una por columna, para facilitar el acceso a los datos. La definición de la propiedad Portada es la siguiente:

```

Public Property Portada As Byte()
    Get
        Try
            Return CType(Me(Me.tableLibros.PortadaColumn), Byte())
        Catch e As InvalidCastException
            Throw New StrongTypingException(
                "No se puede obtener el valor porque es DBNull.", e)
        End Try
    End Get
    Set
        Me(Me.tableLibros.PortadaColumn) = value
    End Set
End Property

```

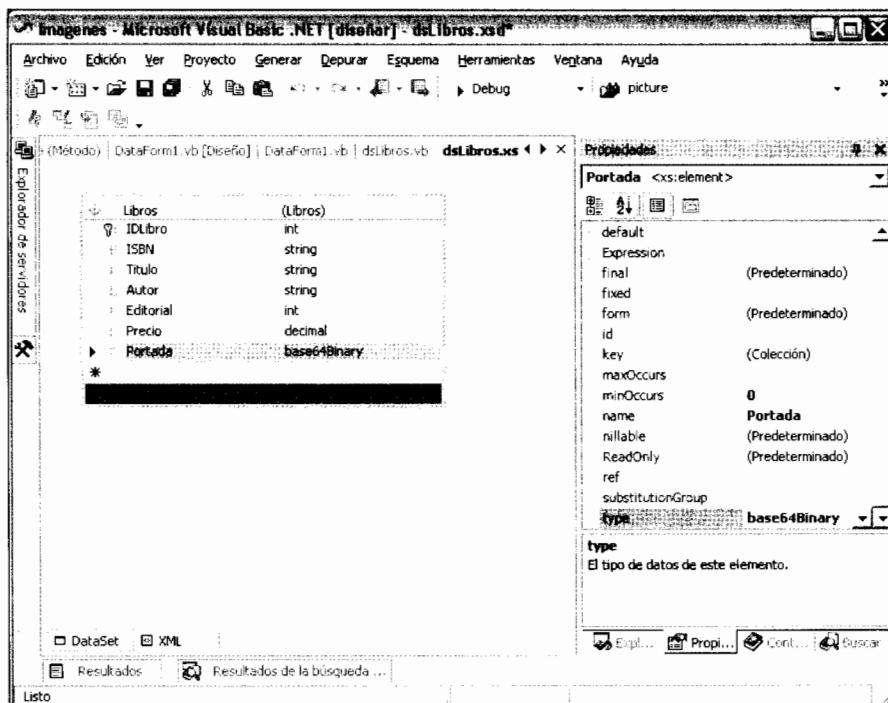


Figura 21.3. La columna en el editor de esquemas XSD

Fíjese en el tipo de dato de la propiedad, un arreglo de tipo Byte. Para obtener la portada, o modificarla, tendremos, por lo tanto, que generar la imagen a partir de una secuencia de bytes, a la hora de leer, o viceversa, si de lo que se trata es de modificar.

Diseño del formulario Windows

Diseñaremos un formulario Windows en el que podamos ver la portada asociada a cada libro, así como modificarla si es necesario. Partiendo de un proyecto estándar, elimine el formulario y emplee el asistente para formularios de datos con el fin de generar una ventana que permita editar la tabla fila a fila, con controles individuales, en lugar de usando un DataGrid. Esto nos permitirá añadir fácilmente un PictureBox en el que mostrar la imagen.

Nota

Si prefiere basar el formulario en un DataGrid, tendrá entonces que controlar la pulsación sobre la columna correspondiente a la portada a fin de abrir una

ventana secundaria con el PictureBox. También puede personalizar la cuadrícula, creando para ello una nueva clase de columna que sea capaz de mostrar la imagen.

Extienda las dimensiones del formulario creado por el asistente, haciéndolo más alto, a fin de introducir en el área inferior izquierda un control PictureBox con un botón debajo, como se puede ver en la figura 21.4. Inserte también un componente OpenFileDialog y edite su propiedad Filter para que facilite la selección de archivos con gráficos.

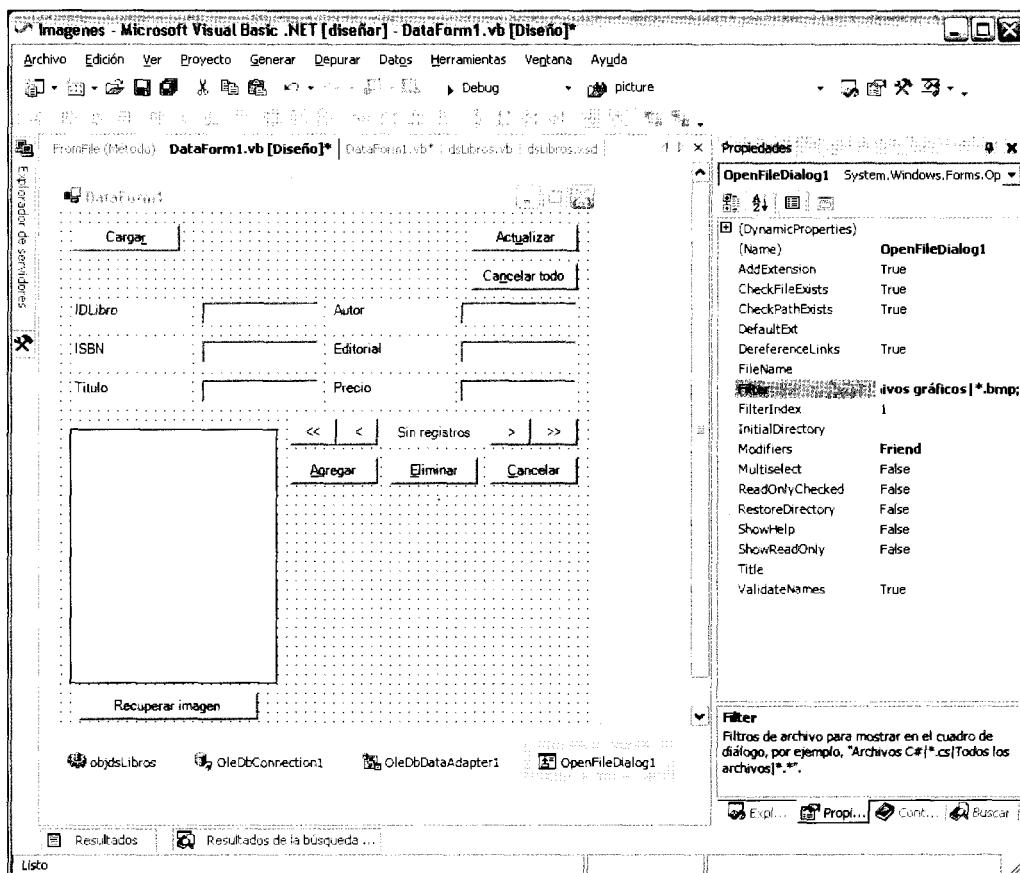


Figura 21.4. Aspecto del formulario tras introducir el PictureBox y el Button

En principio el PictureBox no contendrá ninguna imagen, pudiendo obtenerla de dos formas distintas: recuperándola de la columna Portada de una fila de la tabla Libros o, en caso de que se pulse el botón que hay debajo, directamente desde un archivo.

Recuperar la imagen de la base de datos

El código generado por el asistente de creación de formularios de datos se encarga de la vinculación de todos los controles que hay en la ventana, pero lógicamente no del PictureBox que hemos añadido nosotros. Éste no puede vincularse directamente con la columna Portada de dsLibros, puede intentarlo abriendo la ventana DataBindings del PictureBox. Tenemos, por tanto, que codificar el proceso de recuperación de la imagen contenida en la columna para mostrarla en el PictureBox.

Aprovechando que todos los métodos que causan una actualización de los datos visibles en el formulario, como los botones de navegación, invocan siempre al método `PositionChanged()`, introduciremos en él todo el código necesario.

Primero necesitamos saber cuál es la fila actual de datos que está mostrándose en el formulario, puesto que no podemos mostrar la portada de cualquier fila del DataTable, sino la indicada por el BindingManagerBase que corresponda. Teniendo la posición, comprobamos si la columna Portada de esta fila es nula o no. Si no es nula tenemos una imagen a mostrar, pero no podemos asignar directamente el valor de Portada, recuerde que es un arreglo de tipo Byte, a la propiedad Image del PictureBox, ya que los tipos no son compatibles.

El truco está en crear un flujo de datos en memoria, un objeto `MemoryStream`, que contenga la secuencia de bytes alojada en la columna Portada, usando después el método `FromStream()` de la clase `Image` para mostrarla en el PictureBox.

Por último, en caso de que la columna Portada sea nula, y por lo tanto la propiedad `IsPortadaNull` devuelva `True`, procederemos a eliminar el contenido actual del PictureBox, mostrándolo como una superficie negra.

El método `PositionChanged()` completo quedará como se muestra a continuación. La primera sentencia es la generada por el asistente para actualizar la indicación de fila actual.

```
Private Sub objdsLibros_PositionChanged()
    Me.lblNavLocation.Text = (((Me.BindingContext(_
        objdsLibros, "Libros").Position + 1).ToString + __
        " de ") + Me.BindingContext(objdsLibros, __
        "Libros").Count.ToString)

    ' Mostrar la portada si la hay

    ' Obtenemos la posición actual en la tabla
    Dim Posicion = BindingContext(objdsLibros, _
        "Libros").Position

    ' Si la portada no es nula
    If Not objdsLibros.Libros(Posicion).IsPortadaNull Then
        ' creamos un MemoryStream a partir del Byte()
        ' que nos facilita la propiedad Portada
        Dim Imagen As New MemoryStream(_
            objdsLibros.Libros(BindingContext(_
                objdsLibros, "Libros").Position).Portada)
```

```

' flujo que podemos usar para introducir
' la imagen en el PictureBox
PictureBox1.Image = Image.FromStream(Imagen)
Else ' si no hay imagen
    ' eliminamos el contenido actual del PictureBox
    PictureBox1.CreateGraphics().Clear(Color.Black)
End If
End Sub

```

Si ejecuta el programa tras introducir estos cambios, podrá navegar por las filas de datos y observar que el PictureBox siempre aparece como una superficie negra. Es lógico, ya que ninguno de los libros cuenta actualmente con una portada en la base de datos.

Asignar una imagen desde un archivo

Para llegar a tener alguna imagen en la base de datos, es nuestro objetivo, tendremos que responder al evento Click del botón que hay bajo el PictureBox con dos fines: mostrar la imagen seleccionada en el PictureBox, simplemente a modo de información de retorno porque ello no indica que la imagen exista en la base de datos, y asignar dicha imagen a la columna Portada de la actual fila de datos.

La clase Image cuenta con un método `FromFile()` capaz de recuperar la imagen del archivo y mostrarla en el PictureBox. Los `DataTable`, por el contrario, no disponen de un método similar. No podemos, sin más, asignar un nombre de archivo a la columna Portada. Ya sabemos que su tipo es `Byte()`. Por lo tanto, tendremos que recuperar la imagen en memoria, como si fuese una secuencia de bytes, y después asignarla a Portada.

El método correspondiente al evento Click quedaría tal y como se muestra a continuación:

```

' Al pulsar el botón
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    ' Abrimos el cuadro de diálogo de selección de
    ' un archivo y, si se elige uno
    If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
        ' la mostramos en el PictureBox
        PictureBox1.Image = Image.FromFile(OpenFileDialog1.FileName)

        ' Creamos un FileStream que nos permita
        ' recuperar la imagen en memoria
        Dim Archivo As New FileStream(OpenFileDialog1.FileName, _
            FileMode.Open, FileAccess.Read)
        ' concretamente en un arreglo Byte
        Dim Imagen(Archivo.Length) As Byte
        ' leemos
        Archivo.Read(Imagen, 0, Archivo.Length)
        Archivo.Close() ' y cerramos
    End If
End Sub

```

```

    ' Asignamos la secuencia de bytes a la
    ' columna Portada
    objdsLibros.Libros(Me.BindingContext(
        objdsLibros, "Libros").Position).Portada = Imagen
End If
End Sub

```

La parte más importante es la creación del FileStream y lectura de la imagen en un arreglo Byte, asignando éste a la columna Portada de la fila actual en la tabla Libros.

Ejecute de nuevo el programa y asigne imágenes a varias filas de datos. A continuación pulse el botón **Actualizar** del formulario para guardar los cambios en la base de datos. No podrá ver las portadas abriendo directamente la tabla, pero si ejecuta de nuevo el programa se recuperarán y aparecerán en el formulario. Igualmente podríamos obtenerlas desde una aplicación ASP.NET a fin de enviarlas como parte de una página Web.

En la figura 21.5 puede ver el formulario Windows mostrando una fila de datos, incluida la portada del libro.

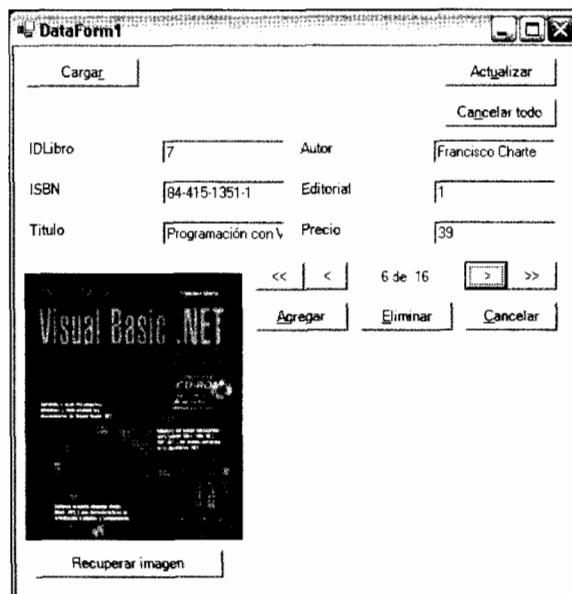
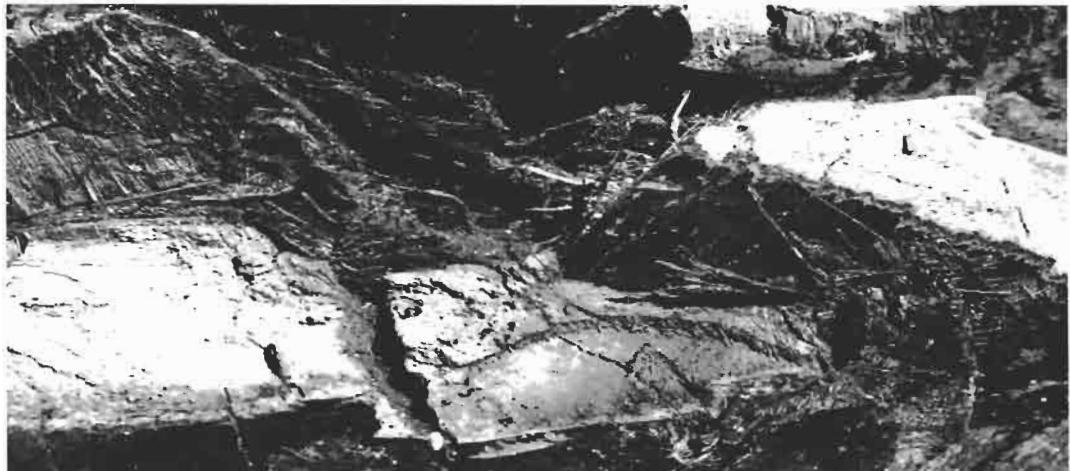


Figura 21.5. El formulario con los datos y portada de un libro



22

Creación de proveedores ADO.NET

La versión 1.0 de la plataforma .NET cuenta con dos proveedores ADO.NET: `SqlClient` y `OleDb`, a los que hay que sumar los dos que se han liberado posteriormente, `OracleClient` y `OdBC`, y que es posible instalar según se vio en el capítulo dedicado a las conexiones de datos.

Igual que se han creado proveedores específicos para SQL Server y Oracle, es posible crearlos también para cualquier origen de datos. Sería posible, por ejemplo, crear un proveedor específico para trabajar con MySql, InterBase o bien IBM DB2.

Para conseguir eso es necesario crear al menos cuatro clases: una que gestione la conexión, otra que ejecute los comandos, una tercera que facilite la lectura unidireccional y, por último, otra que haga las veces de adaptador de datos. Cada una de esas clases implementará las interfaces genéricas que ya conocemos, haciendo posible su uso desde cualquier aplicación.

Nuestro objetivo, en este capítulo, es crear un proveedor de datos ADO.NET sencillo. Su origen de datos será siempre el mismo, el sistema de archivos, pudiendo seleccionarse la carpeta de la que se desean obtener, a modo de filas, las entradas existentes, apareciendo como columnas el nombre completo de cada entrada y la fecha en que se creó.

Muchos miembros de las interfaces que es obligatorio implementar quedarán sin funcionalidad, ya que el proveedor no accede a una base de datos real y, además, no se ofrecerán funciones de actualización.

El código, no obstante, puede servirle como base para crear cualquier otro proveedor más completo.

Inicio del proyecto

Para crear el proveedor ADO.NET tendrá que iniciar un nuevo proyecto Visual Basic .NET, concretamente una **Biblioteca de clases**. Llámela `FileSystemClient`, como se ha hecho en la figura 22.1. La biblioteca obtenida, al compilar el proyecto, se llamará `FileSystemClient.dll`.

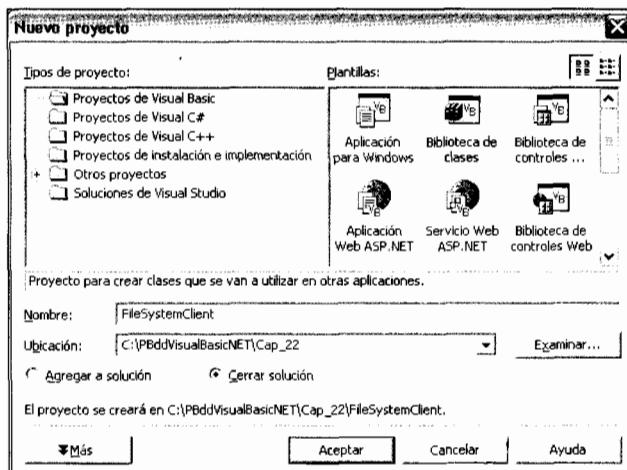


Figura 22.1. Iniciamos el proyecto de biblioteca de clases

A este proyecto añadiremos cuatro módulos de clase:

- `FileSystemClientConnection.vb`: Contendrá la clase `FileSystemClientConnection`, implementando la interfaz `IDbConnection`. Se encargará de controlar la conexión.
- `FileSystemClientCommand.vb`: En su interior definiremos la clase `FileSystemClientCommand`, que implementará la interfaz `IDbCommand`. Ejecutará el comando especificado ofreciendo un resultado.
- `FileSystemClientDataReader.vb`: Este módulo alojará la clase `FileSystemClientDataReader`. Implementará las interfaces `IDataReader` e `IDataRecord`, facilitando el acceso a los datos de una fila y el avance unidireccional por las filas.
- `FileSystemClientDataAdapter.vb`: Contendrá la clase `FileSystemClientDataAdapter`. Derivada de `DbDataAdapter` e implementando la interfaz `IDbDataAdapter`, será la clase encargada de actuar como adaptador de datos para poder llenar un `DataSet` a partir de un comando.

La biblioteca de clases generada no estará firmada con una clave o *nombre fuerte*, por lo que no podrá añadirla al GAC para emplearla desde cualquier proyecto

como hace con los otros proveedores. No obstante, no tiene más que usar la herramienta `sn` para generar la clave y firmar el módulo DLL para poder hacerlo.

FileSystemClientConnection

Cuando va a operarse con un cierto proveedor de datos, habitualmente lo primero que se hace es abrir una conexión con el origen de datos. Por ello nos vamos a ocupar en primer lugar de esta clase, la cual será la que implemente la interfaz `IDbConnection`. Su cabecera sería la siguiente:

```
' Clase que facilita la conexión
Public Class FileSystemClientConnection
    Inherits ComponentModel.Component
    ' Implementa la interfaz IDbConnection
    Implements IDbConnection
```

El hecho de implementar la interfaz `IDbConnection` nos obliga a codificar una serie de miembros, propiedades y métodos, independientemente de que su contenido sea o no funcional. Algunos de esos miembros son propiedades que tienen que mantener y facilitar datos, para los cuales definiremos las siguientes variables:

```
' Cadena de conexión
Private strConnectionString As String
' Camino al que se accederá
Private strDataBase As String
' Estado de la conexión, inicialmente cerrada
Private csState As ConnectionState = ConnectionState.Closed
```

Para crear un objeto `FileSystemClientConnection`, la clase deberá contar con uno o más constructores. Es habitual que siempre exista un constructor por defecto, que no necesita parámetros y, en este caso concreto, también un constructor que acepte como parámetro la cadena de conexión. Su implementación sería la mostrada a continuación:

```
' Constructor por defecto
Public Sub New()
    MyBase.New()
End Sub

' Constructor que acepta la cadena de conexión
Public Sub New(ByVal ConnectionString As String)
    MyBase.New()
    ' Guardamos la cadena de conexión
    strConnectionString = ConnectionString
    ' Y extraemos el camino que se ha facilitado tras 'Data Source='
    strDataBase = ConnectionString.Substring(
        ConnectionString.IndexOf("=") + 1)
End Sub
```

Observe que la cadena de conexión se guarda, tal cual, en una de las variables definidas al principio a tal efecto. A continuación se extrae de ella el valor que sigue al signo =. El único parámetro que aceptará nuestro proveedor en la cadena de conexión será Data Source, siendo su valor el camino con el que se desea conectar, por ejemplo C:\Windows.

Los parámetros establecidos con la cadena de conexión, la propia cadena de conexión y la base de datos, así como el estado en el que se encuentra la conexión, cerrada o abierta, son datos accesibles mediante propiedades definidas en IDbConnection y que implementaremos así:

```
' Propiedad para facilitar la lectura y
' modificación de la cadena de conexión
Public Property ConnectionString() As String _
    Implements IDbConnection.ConnectionString
    Get ' devolvemos la cadena de conexión
        Return strConnectionString
    End Get
    ' si se facilita una nueva cadena
    Set(ByVal Value As String)
        ' la guardamos
        strConnectionString = Value
        ' y actualizamos el camino
        strDataBase = ConnectionString.Substring( _
            ConnectionString.IndexOf("=") + 1)
    End Set
End Property

' Propiedad que facilita el acceso al
' que está accediéndole
Public ReadOnly Property Database() As String _
    Implements IDbConnection.Database
    Get
        Return strDataBase
    End Get
End Property

' Propiedad que facilita el estado actual de la conexión
Public ReadOnly Property State() As ConnectionState _
    Implements IDbConnection.State
    Get
        Return csState
    End Get
End Property

' Este método cambia la base de datos, en realidad
' el camino al que está accediéndole
Public Sub ChangeDatabase(ByVal Value As String) _
    Implements IDbConnection.ChangeDatabase
    ' Asignamos directamente el camino
    strDataBase = Value
    ' y generamos la nueva cadena de conexión
    strConnectionString = "Data Source=" & strDataBase
End Sub
```

En caso de modificarse la propiedad `ConnectionString`, o bien de llamar a `ChangeDatabase()`, se recomponen la cadena de conexión y vuelve a extraer el camino con el que debe conectarse. Con un proveedor de datos clásico, ese cambio no podría efectuarse de estar la conexión abierta, siendo precisa cerrarla y, tras cambiar la cadena de conexión, volver a abrirla. En nuestro caso, no hay problema en hacer el cambio en cualquier momento.

Además de propiedades, en la interfaz `IDbConnection` también aparecen los métodos `Open()` y `Close()`, cuya finalidad es abrir y cerrar la conexión con el origen de datos. Como se ve a continuación, nuestro proveedor realmente no abre ni cierra un archivo o una conexión con un servidor remoto, sino que se limita a cambiar el estado del objeto que representa a la conexión, simplemente.

```
' Este método abre la conexión
Public Sub Open() Implements IDbConnection.Open
    ' Simplemente cambiamos el estado
    csState = ConnectionState.Open
End Sub

' Este método cierra la conexión
Public Sub Close() Implements IDbConnection.Close
    ' Simplemente cambiamos el estado
    csState = ConnectionState.Closed
End Sub
```

El objeto `FileSystemClientConnection` no es muy útil por sí solo, y adquiere sentido al utilizarse conjuntamente con comandos. Éstos pueden crearse mediante el método `CreateCommand()` que implementaremos así:

```
' Este método crea un nuevo comando asociado con esta conexión
Public Function CreateCommand() As IDbCommand
    Implements IDbConnection.CreateCommand
    ' Devolvemos el objeto creado
    Return New FileSystemClientCommand("", Me)
End Function
```

Al facilitar como segundo parámetro al constructor del comando una referencia a la propia conexión, ya está creándose el vínculo entre conexión y comando. Esto permitirá que el objeto `FileSystemClientCommand` sepa cuál es el camino, en teoría base de datos, con el que debe conectar para obtener información.

La interfaz `IDbConnection` no define muchos más miembros que los que ya hemos implementado. Los que restan, `BeginTransaction()` y la propiedad `ConnectionTimeout`, no tienen sentido en nuestro proveedor, y por ello, aunque estamos obligados a implementarlos, no tendrán funcionalidad alguna:

```
' Miembros que hay que implementar, aunque
' sea vacíos como en este caso

Public ReadOnly Property ConnectionTimeout() As Integer
    Implements IDbConnection.ConnectionTimeout
```

```

Get
    Return 0
End Get
End Property

Public Overloads Function BeginTransaction() As IDbTransaction _
    Implements IDbConnection.BeginTransaction
    '
End Function

Public Overloads Function BeginTransaction( _
    ByVal level As IsolationLevel) As IDbTransaction ...
    Implements IDbConnection.BeginTransaction
    '
End Function

```

La propiedad ConnectionTimeout siempre devuelve 0, mientras que las dos versiones de BeginTransaction() no hacen nada.

Con esto ya tenemos la clase que permitirá conectar con el origen de datos. De querer usarla para acceder a una base de datos, o un archivo, habría que modificar el análisis de la cadena de conexión, así como la implementación de los métodos Open() y Close() para que realmente estableciesen la conexión con el origen.

FileSystemClientCommand

La siguiente clase que crearemos será FileSystemClientCommand, con la cual podremos definir y ejecutar comandos. En realidad el comando será siempre el mismo: obtener todas las entradas de directorio que existan en la carpeta establecida en el objeto FileSystemClientConnection. No será importante, por tanto, el tipo del comando o la cadena que lo defina, aunque implementaremos las propiedades necesarias para mantener esos datos.

Comenzamos por el principio, con los miembros de datos que tendrá esta clase y que son los siguientes:

```

' Clase que facilita la ejecución de comandos
Public Class FileSystemClientCommand
    Inherits ComponentModel.Component
    Implements IDbCommand

    ' Variables para guardar el texto del comando y su tipo
    Private strCommandText As String
    Private ctCommandType As CommandType

    ' Referencia a la conexión asociada al comando
    Private cnConnection As FileSystemClientConnection

    ' Éste es el origen de los datos, del que
    ' se extraerán las filas y columnas
    Friend dfOrigen As DirectoryInfo

```

Tenemos sendas variables para almacenar el tipo y texto del comando. La variable `cnConnection` mantendrá un enlace entre el comando y el objeto `FileSystemClientConnection` sobre el que actuará. Por último tenemos la variable `dfOrigen`, que actuará como origen de datos. Al ejecutar el comando, según verá después, se actuará sobre este objeto.

Observe que `dfOrigen` tiene el modificador `Friend`, facilitando el acceso a él desde la clase `FileSystemClientDataReader` que crearemos en el punto siguiente.

Acto seguido implementaremos los distintos constructores de la clase, en este caso concreto tres:

```
' Constructores

' constructor por defecto
Public Sub New()
    MyBase.new()
End Sub

' Este constructor acepta un comando
Public Sub New(ByVal CommandText As String)
    strCommandText = CommandText
End Sub

' Este constructor acepta un comando y
' una referencia a una conexión
Public Sub New(ByVal CommandText As String, _
    ByVal Connection As FileSystemClientConnection)
    strCommandText = CommandText
    cnConnection = Connection
End Sub
```

Son los tres que podemos encontrar en los demás proveedores. Como puede ver, se limitan a guardar los parámetros facilitados en las variables que corresponden, sin efectuar ninguna comprobación adicional sobre su validez.

La conexión y tanto el tipo de comando como su texto son parámetros del comando que pueden ser modificados en cualquier momento, ya que no se emplean hasta la invocación a uno de los métodos `ExecuteXXX()`.

Las tres propiedades siguientes facilitan tanto la obtención como modificación de esos parámetros:

```
' Propiedad para acceder al texto del comando
Public Property CommandText() As String
    Implements IDbCommand.CommandText
    Get
        Return strCommandText
    End Get
    Set(ByVal Value As String)
        strCommandText = Value
    End Set
End Property
```

```

' Propiedad para acceder al tipo de comando
Public Property CommandType() As CommandType 
    Implements IDbCommand.CommandType 
        Get
        Return ctCommandType
    End Get
    Set(ByVal Value As CommandType)
        ctCommandType = Value
    End Set
End Property

' Propiedad para acceder a la conexión
' asociada al comando
Public Property Connection() As IDbConnection 
    Implements IDbCommand.Connection 
        Get
        Return cnConnection
    End Get
    Set(ByVal Value As IDbConnection)
        cnConnection = CType(Value, FileSystemClientConnection)
    End Set
End Property

```

De los métodos `ExecuteXXX()` de la interfaz `IDbCommand` facilitaremos implementación útil para tres de ellos. El primero, `ExecuteNonQuery()`, debe ejecutar el comando sin recuperar en realidad datos, devolviendo tan sólo el número de filas que se obtendrían. En este caso, como puede verse a continuación, recuperamos las entradas del camino indicado con el método `GetFileSystemInfos()`, obteniendo el número de entradas con la propiedad `Length`. Antes se comprueba que tenemos un objeto de conexión asociado y, además, que dicha conexión está abierta.

```

' Este método ejecuta el comando devolviendo
' tan sólo el número de filas que se obtendrían
Public Function ExecuteNonQuery() As Integer 
    Implements IDbCommand.ExecuteNonQuery 
        ' comprobamos que tengamos una conexión
        ' asociada y que esté abierta
        If Not cnConnection Is Nothing And
            cnConnection.State = ConnectionState.Open Then
                ' en caso afirmativo creamos un DirectoryInfo
                dfOrigen = New DirectoryInfo(cnConnection.Database)
                ' y devolvemos el número de entradas que tiene
                Return dfOrigen.GetFileSystemInfos.Length
            End If
End Function

```

Observe que no usamos para nada el tipo de comando ni el texto. Podríamos modificar esta implementación, mejorando la clase `FileSystemClientCommand`, a fin de que fuese posible seleccionar sólo las entradas de archivo o sólo las entradas de directorio, o bien sólo las entradas que se ajustasen a un cierto patrón. En

esos casos habría que analizar el texto del comando y, en lugar de usar directamente la colección facilitada por `GetFileSystemInfos()`, filtrar las entradas obtenidas, por ejemplo utilizando los métodos `GetFiles()` y `GetDirectories()` en lugar de `GetFileSystemInfos()`.

El siguiente método que implementaremos será `ExecuteReader()` que, como sabe, debe facilitar un objeto que implemente las interfaces `IDataReader` e `IDataRecord`. Lo haremos así:

```
' Este método ejecuta el comando devolviendo
' un DataReader para recorrer las filas
Public Overloads Function ExecuteReader() As IDataReader _
    Implements IDbCommand.ExecuteReader
    ' comprueba si existe una conexión
    ' y que esté abierta
If Not cnConnection Is Nothing And _
    cnConnection.State = ConnectionState.Open Then
        ' Creamos el objeto DirectoryInfo
        dfoOrigen = New DirectoryInfo(cnConnection.Database)
        ' creamos un DataReader asociado al comando y
        ' lo devolvemos
        Return New FileSystemClientDataReader(Me)
    End If
End Function
```

Creamos el objeto `DirectoryInfo` utilizando el camino establecido en la conexión y, a continuación, creamos un `FileSystemClientDataReader` asociado con este comando, devolviéndolo como resultado.

El último método útil a implementar es la versión de `ExecuteReader()` que acepta un valor `CommandBehavior` como parámetro. Dicho valor indicaría un comportamiento al que tendría que ajustarse el lector de datos. Como se ve a continuación, ignoramos el valor y sencillamente ejecutamos la primera versión del método.

```
' Este método tiene que ejecutar el comando
' devolviendo un DataReader con un cierto
' comportamiento
Public Overloads Function ExecuteReader(
    ByVal behavior As CommandBehavior) As IDataReader _
    Implements IDbCommand.ExecuteReader
    ' nos limitamos a devolver el mismo
    ' resultado que ExecuteReader()
    Return ExecuteReader()
End Function
```

Como ocurriera con la clase `FileSystemClientConnection`, ésta también está obligada a implementar una serie de métodos aunque sin ninguna funcionalidad.

Es preciso añadir, por tanto, el código siguiente al módulo, aunque los métodos y propiedades queden sin código útil.

```
' Miembros que quedan sin implementar
Public Function ExecuteScalar() As Object _
    Implements IDbCommand.ExecuteScalar
    Return Nothing
End Function

Public Function CreateParameter() As IDbDataParameter _
    Implements IDbCommand.CreateParameter
    Return Nothing
End Function

Public Sub Prepare() Implements IDbCommand.Prepare
    '
End Sub

Public Property CommandTimeout() As Integer _
    Implements IDbCommand.CommandTimeout
    Get
        Return 0
    End Get
    Set(ByVal Value As Integer)
        '
    End Set
End Property

Protected ReadOnly Property Parameters() As _
    IDataParameterCollection Implements IDbCommand.Parameters
    Get
        '
    End Get
End Property

Public Property Transaction() As IDbTransaction _
    Implements IDbCommand.Transaction
    Get
        Return CType(Nothing, IDbTransaction)
    End Get
    Set(ByVal Value As IDbTransaction)
        '
    End Set
End Property

Public Property UpdatedRowSource() As UpdateRowSource _
    Implements IDbCommand.UpdatedRowSource
    Get
        Return UpdateRowSource.None
    End Get
    Set(ByVal Value As UpdateRowSource)
        '
    End Set
End Property

Public Sub Cancel() Implements IDbCommand.Cancel
    '
End Sub
```

FileSystemClientDataReader

El método `ExecuteReader()` de la clase definida en el punto anterior devuelve un objeto `FileSystemClientDataReader`, cuya clase nos disponemos a implementar a continuación. Con ella podremos ir recorriendo las filas de datos, obteniendo el valor de cada una de las columnas. La información será recuperada del objeto `DirectoryInfo` creado por el comando, al que tenemos acceso por haberse declarado `Friend` en lugar de `Private`.

Comenzamos, como en los puntos anteriores, con la declaración de miembros de datos de la clase, que será la siguiente:

```
' Clase que actúa como lector de
' datos del proveedor
Public Class FileSystemClientDataReader
    Inherits ComponentModel.Component
    Implements IDataReader
    Implements IDataRecord

    ' Posición en el conjunto de datos
    Private iPosition As Integer
    ' y número de filas existentes
    Private iRows As Integer

    ' Comando asociado al DataReader
    Private fscCommand As FileSystemClientCommand
```

Tenemos dos variables enteras, `iPosition` e `iRows`, que mantendrá la fila actual y el número total de filas. `fscCommand` almacenará la referencia al comando asociado a este lector.

Esta clase contará con un único constructor que tomará como parámetro la referencia al comando asociado. Así, no podrá nunca crearse un lector de datos sin comando asociado.

```
' El único constructor del DataReader
' necesita como parámetro el comando
' al que va a asociarse
Public Sub New(ByVal Command As FileSystemClientCommand)
    ' guardamos la referencia al comando
    fscCommand = Command
    ' obtenemos el número de filas
    iRows = fscCommand.dfoRigen.GetFileSystemInfos.Length
    ' y lijamos la posición inicial
    iPosition = -1
End Sub
```

Además de guardar la referencia al comando, damos un valor inicial a la variables `iRows` e `iPosition`. La primera contendrá el número de elementos obtenidos del método `GetFileSystemInfos()`, mientras que la segunda será -1 de tal

forma que la primera llamada al método `Read()` nos coloque realmente en la primera fila de datos.

El método `Read()`, como se ve en el código siguiente, es realmente sencillo. Incrementamos el valor de la variable `iPosition` y, a continuación, devolvemos `True` o `False` dependiendo de que queden o no más filas de datos. Para ello comparamos `iPosition` con `iRows`.

```
' Este método avanza a la fila siguiente
' y devolverá True o False dependiendo de
' que sea la última
Public Function Read() As Boolean
    Implements IDataReader.Read
    ' Avanzamos a la fila siguiente
    iPosition += 1
    ' devolvemos True si quedan filas
    Return iPosition < iRows
End Function
```

Una vez que estamos colocados en una fila de datos, el paso siguiente será recuperar el valor que almacena cada una de sus columnas. Con este fin tendremos que implementar dos versiones diferentes de la propiedad `Item`. La primera tomará como parámetro el índice de la columna, mientras que la segunda recibirá, en lugar del índice, el nombre de la columna.

Ninguno de los dos métodos accede directamente a la lista devuelta por `GetFileSystemInfos()`, sino que utilizan el método `GetValue()` de la propia clase `FileSystemClientDataReader`. Ésta siempre toma como parámetro el índice de la columna cuyo valor quiere obtenerse. Puesto que en la segunda versión desconocemos dicho índice, lo que tenemos es el nombre de la columna, lo recuperamos mediante el método `GetOrdinal()`.

```
' Implementación de la propiedad Item
' para acceder a los elementos de la
' fila actual
Default Public Overloads ReadOnly Property Item(
    ByVal i As Integer) As Object Implements IDataReader.Item
    Get
        Return GetValue(i) ' devolvemos el valor
    End Get
End Property

' La misma propiedad aceptando un nombre
' de columna en lugar de un índice
Default Public Overloads ReadOnly Property Item(ByVal name _
    As String) As Object Implements IDataReader.Item
    Get
        ' devolvemos el valor obteniendo el índice
        ' que corresponda a la columna
        Return GetValue(GetOrdinal(name))
    End Get
End Property
```

Para facilitar a las dos propiedades anteriores el índice o nombre de una columna, con el fin de obtener su valor, primero necesitaremos saber cuántas columnas hay disponibles y cuáles son sus nombres. Estos datos son los que nos facilita la propiedad FieldCount y el método GetName(), que implementamos como se ve en el código siguiente:

```
' Esta propiedad facilita el número de
' columnas que tiene la fila
Public ReadOnly Property FieldCount() As Integer _
    Implements IDataRecord.FieldCount
    Get
        Return 2
    End Get
End Property

' Función que facilita el nombre de la columna
' cuyo índice se facilita
Public Function GetName(ByVal i As Integer) As String _
    Implements IDataRecord.GetName
    Select Case i
        Case 0
            Return "FullName"
        Case 1
            Return "CreationTime"
        Case Else
            Return Nothing
    End Select
End Function
```

Como puede verse, facilitaremos tan sólo dos columnas por fila: FullName y CreationTime, es decir, el nombre del elemento y su fecha de creación. También será útil saber qué tipo de dato contiene cada una de esas columnas, información que obtendríamos con los dos métodos mostrados a continuación:

```
' Esta función devuelve una cadena con el tipo
' de dato de la columna indicada
Public Function GetDataTypeName(ByVal i As Integer) As String _
    Implements IDataRecord.GetDataTypeName
    ' ambas columnas son cadenas
    Return "System.String"
End Function

' Esta función devuelve el tipo de datos de la
' columna indicada
Public Function GetFieldType(ByVal i As Integer) As Type _
    Implements IDataRecord.GetFieldType
    ' las dos columnas son del tipo String
    Return GetType(System.String)
End Function
```

GetDataTypeName() facilita el nombre del tipo de dato, es decir, una cadena con el tipo, mientras que GetFieldType() devuelve el tipo en sí.

Las dos versiones de la propiedad `Item` hacen uso del método `GetValue()`. Éste acepta como parámetro el índice de la columna a obtener, devolviendo el valor que corresponda teniendo en cuenta que la fila actual de datos es `iPosition`.

```
' Este método devuelve el valor de la columna cuyo índice se facilita
Public Function GetValue(ByVal i As Integer) As Object _
    Implements IDataRecord.GetValue
    ' Asumiendo la referencia a la fila actual de
    ' la colección devuelta por GetFileSystemInfos()
    With fscCommand.dfOrigen.GetFileSystemInfos()(iPosition)
        Select Case i
            Case 0 ' si la columna es 0
                Return .FullName ' facilitamos el nombre
            Case 1 ' si la columna es 1
                Return .CreationTime.ToString() ' la fecha
        End Select
    End With
End Function
```

Utilizamos `iPosition` como índice para acceder a la colección devuelta por `GetFileSystemInfos()`, usando el índice facilita como parámetro a `GetValue()` para decidir si se devuelve la propiedad `FullName` o `CreationTime`.

Otro método empleado desde `Item` es `GetOrdinal()`, cuya finalidad es devolver el índice numérico correspondiente a una columna de la cual se facilita el nombre. Su implementación, en este caso, es realmente sencilla puesto que sólo tenemos dos columnas y siempre son las mismas.

```
' Este método facilita el índice de una columna a partir de su nombre
Public Function GetOrdinal(ByVal name As String) As Integer _
    Implements IDataRecord.GetOrdinal
    If name = "FullName" Then
        Return 0
    Else
        Return 1
    End If
End Function
```

El último método útil a implementar en esta clase es `GetValues()`. A diferencia de `GetValue()`, éste no recibe como parámetro un índice numérico sino un arreglo de elementos `Object` en el que debe retornarse el valor de todas las columnas de la fila actual. Además, el método devuelve un valor entero indicando el número de elementos introducidos en el arreglo. Lo implementamos así:

```
' Este método facilita los valores de las columnas
' en el arreglo facilitado como parámetro
Public Function GetValues(ByVal values() As Object) As Integer
    Implements IDataRecord.GetValues
    ' asignamos a cada elemento del arreglo el
    ' dato de la columna correspondiente
    values(0) = fscCommand.dfOrigen.GetFileSystemInfos() _
        (iPosition).FullName
```

```
values(l) = fscCommand.dfOrigen.GetFileSystemInfos() _
(iPosition).CreationTime

' indicamos el número de datos devueltos
Return 2
End Function
```

Por último, ya sin implementaciones útiles, debemos codificar los métodos mostrados a continuación. Observe que gran parte de ellos son métodos GetXXX() de la interfaz IDataRecord, métodos que recuperan el valor de una columna no con el tipo original, sino con otro especificado.

```
' Este método facilita el índice de una columna
' a partir de su nombre
Public Function GetOrdinal(ByVal name As String) As Integer _
    Implements IDataRecord.GetOrdinal
    If name = "FullName" Then
        Return 0
    Else
        Return 1
    End If
End Function

' Miembros no implementados

Public Function GetSchemaTable() As DataTable _
    Implements IDataReader.GetSchemaTable
    Return New DataTable()
End Function

Public ReadOnly Property Depth() As Integer _
    Implements IDataReader.Depth
    Get
        Return 0
    End Get
End Property

Public ReadOnly Property IsClosed() As Boolean _
    Implements IDataReader.IsClosed
    Get
        Return False
    End Get
End Property

Public ReadOnly Property RecordsAffected() As Integer _
    Implements IDataReader.RecordsAffected
    Get
        Return -1
    End Get
End Property

Public Sub Close() Implements IDataReader.Close
'
End Sub
```

```
Public Function NextResult() As Boolean _
    Implements IDataReader.NextResult
    Return False
End Function

Public Function GetBoolean(ByVal i As Integer) As Boolean _
    Implements IDataRecord.GetBoolean
    Return False
End Function

Public Function GetByte(ByVal i As Integer) As Byte _
    Implements IDataRecord.GetByte
    Return 0
End Function

Public Function GetBytes(ByVal i As Integer,
    ByVal fieldOffset As Long, ByVal buffer() As Byte,
    ByVal bufferOffSet As Integer, ByVal length As Integer) _
    As Long Implements IDataRecord.GetBytes
    Return 0
End Function

Public Function GetChar(ByVal i As Integer) As _
    Char Implements IDataRecord.GetChar
    Return ""
End Function

Public Function GetChars(ByVal i As Integer,
    ByVal fieldOffSet As Long, ByVal buffer() As Char,
    ByVal bufferOffSet As Integer, ByVal length As Integer) _
    As Long Implements IDataRecord.GetChars
    Return 0
End Function

Public Function GetGuid(ByVal i As Integer) As _
    Guid Implements IDataRecord.GetGuid
    Return CType(Nothing, Guid)
End Function

Public Function GetInt16(ByVal i As Integer) As _
    Int16 Implements IDataRecord.GetInt16
    Return CType(Nothing, Int16)
End Function

Public Function GetInt32(ByVal i As Integer) As _
    Int32 Implements IDataRecord.GetInt32
    Return CType(Nothing, Int32)
End Function

Public Function GetInt64(ByVal i As Integer) As _
    Int64 Implements IDataRecord.GetInt64
    Return CType(Nothing, Int64)
End Function

Public Function GetFloat(ByVal i As Integer) As _
    Single Implements IDataRecord.GetFloat
```

```

    Return CType(Nothing, Single)
End Function

Public Function GetDouble(ByVal i As Integer) As _
    Double Implements IDataRecord.GetDouble
    Return CType(Nothing, Double)
End Function

Public Function GetString(ByVal i As Integer) As _
    String Implements IDataRecord.GetString
    Return CType(Nothing, String)
End Function

Public Function GetDecimal(ByVal i As Integer) As _
    Decimal Implements IDataRecord.GetDecimal
    Return CType(Nothing, Decimal)
End Function

Public Function GetDateTime(ByVal i As Integer) As _
    DateTime Implements IDataRecord.GetDateTime
    Return CType(Nothing, DateTime)
End Function

Public Function GetData(ByVal i As Integer) As _
    IDataReader Implements IDataRecord.GetData
    Return Nothing
End Function

Public Function IsDBNull(ByVal i As Integer) As Boolean _
    Implements IDataRecord.IsDBNull
    Return False
End Function

```

FileSystemClientDataAdapter

La última clase que tenemos por implementar es la que actuará como adaptador de datos. A diferencia de las anteriores, esta clase no estará derivada de ComponentModel sino de DbDataAdapter.

Ésta clase base se ocupa ya de la implementación de gran parte de la funcionalidad del adaptador de datos, por lo que el trabajo que tenemos que efectuar nosotros es relativamente breve y fácil.

Comenzamos, como en los casos anteriores, con la declaración de miembros de datos:

```

' Clase que actúa como adaptador de datos
Public Class FileSystemClientDataAdapter
    Inherits DbDataAdapter
    Implements IDbDataAdapter

    ' Comando asociado al adaptador
    Private fscSelectCommand As FileSystemClientCommand

```

Una clase de adaptador de datos corriente dispondrá de cuatro objetos Command, uno para selección, otro para inserción, un tercero para actualización y el último para eliminación. En nuestro caso, al no implementarse funciones nada más que de selección, contamos sólo con una variable de tipo FileSystemClientCommand.

Contaremos con dos constructores, el constructor por defecto y otro que toma como parámetro el comando a partir del cual trabajará el adaptador.

```
' Constructores de la clase

' Constructor por defecto
Public Sub New()
    MyBase.New()
End Sub

' Constructor que acepta como parámetro el comando
Public Sub New(ByVal Command As FileSystemClientCommand)
    MyBase.New()
    ' Guardamos la referencia al comando
    fscSelectCommand = Command
End Sub
```

Por último, implementaremos la propiedad que permita obtener y modificar ese comando de selección:

```
' Propiedad que facilita el acceso al comando de selección
Public Property SelectCommand() As IDbCommand
    Implements IDbDataAdapter.SelectCommand
    Get
        Return fscSelectCommand
    End Get
    Set(ByVal Value As IDbCommand)
        fscSelectCommand = CType(Value, FileSystemClientCommand)
    End Set
End Property
```

El resto de los miembros que estamos obligados a implementar son los que aparecen a continuación, sin código útil:

```
' Miembros no implementados

Public Property InsertCommand() As IDbCommand
    Implements IDbDataAdapter.InsertCommand
    Get
        Return Nothing
    End Get
    Set(ByVal Value As IDbCommand)
        '
    End Set
End Property

Public Property UpdateCommand() As IDbCommand
    Implements IDbDataAdapter.UpdateCommand
```

```
Get
    Return Nothing
End Get
Set(ByVal Value As IDbCommand)
'
End Set
End Property

Public Property DeleteCommand() As IDbCommand
    Implements IDbDataAdapter.DeleteCommand
Get
    Return Nothing
End Get
Set(ByVal Value As IDbCommand)
'
End Set
End Property

Protected Overrides Function CreateRowUpdatedEvent(
    ByVal dataRow As DataRow, ByVal command As IDbCommand,
    ByVal statementType As StatementType, ByVal tableMapping
    As DataTableMapping) As RowUpdatedEventArgs
    Return Nothing
End Function

Protected Overrides Function CreateRowUpdatingEvent(
    ByVal dataRow As DataRow, ByVal command As IDbCommand,
    ByVal statementType As StatementType, ByVal tableMapping
    As DataTableMapping) As RowUpdatingEventArgs
    Return Nothing
End Function

Protected Overrides Sub OnRowUpdating(
    ByVal value As RowUpdatingEventArgs)
'
End Sub

Protected Overrides Sub OnRowUpdated(ByVal value =
    As RowUpdatedEventArgs)
'
End Sub
```

Con esto se ha finalizado la clase del adaptador de datos y también el proveedor de datos. Sencillo, ya que no facilita la selección de datos distintos mediante consultas o la actualización, pero suficiente para ser usado como ejemplo y como base para otros proveedores.

Prueba del proveedor

En este momento, tras compilar el proyecto, tenemos el proveedor FileSystemClient en un módulo DLL. No sabremos si funciona o no adecuadamente hasta

que lo probemos, que es lo que vamos a hacer de inmediato. Sin cerrar el proyecto del proveedor, añada un nuevo proyecto a la solución. Agregue al nuevo proyecto una referencia al primero, para poder utilizar el proveedor simplemente importando el ámbito `FileSystemClient`.

Ejecución de un comando

Comenzaremos por un ejemplo muy sencillo, con el que obtendremos el número de entradas de directorio que hay en una cierta carpeta. Para ello vamos a insertar en el formulario del proyecto añadido a la solución un botón, asociándole el código siguiente:

```
' Definimos la conexión
Dim Conexion As New FileSystemClientConnection( _
    "Data Source=C:\\")

' Ejecutamos un comando a partir de ésta
Dim Comando As FileSystemClientCommand = _
    Conexion.CreateCommand()

' Abrimos la conexión
Conexion.Open()
' y obtenemos el número de filas
Dim Resultado As Integer = Comando.ExecuteNonQuery()
' mostrándolo en un mensaje
MessageBox.Show(Resultado)

Conexion.Close()
```

Definimos una conexión y un comando, abrimos la conexión y usamos el método `ExecuteNonQuery()` para obtener el número de filas que recuperaríamos con ese comando. Al ejecutar el programa aparecería una pequeña ventana con ese número, como se ve en la figura 22.2.



Figura 22.2. Número de filas que se recuperarían con el comando

Uso del lector de datos

En vez de usar el método `ExecuteNonQuery()`, podemos invocar a `ExecuteReader()` para obtener un lector de datos. A partir de ese momento lo utilizaríamos como cualquier otro `DataReader` de los que conocimos en capítulos previos.

Inserte en el formulario un control **ListBox**. Usando la conexión y el comando definidos en el ejemplo del punto previo, añada el código siguiente antes de cerrar la conexión:

```
' Usaremos ExecuteReader() para obtener un lector
Dim Lector As IDataReader = Comando.ExecuteReader()
' Si recuperamos cada fila
While Lector.Read()
    ' añadiéndolas al ListBox
    ListBox1.Items.Add(Lector("FullName") & " - " & Lector(1))
End While
```

Al ejecutar el programa, en la lista debería aparecer el nombre y fecha de creación de cada una de las entradas de directorio existentes en la carpeta indicada por la conexión. En la figura 22.3 puede ver el ejemplo en ejecución.

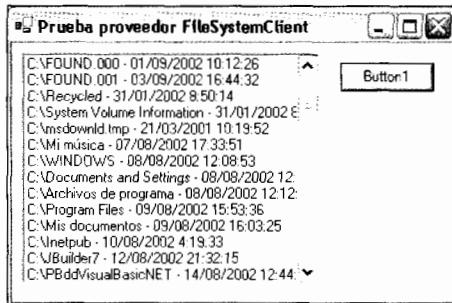


Figura 22.3. El **ListBox** con los datos de las entradas de directorio

Uso del adaptador de datos

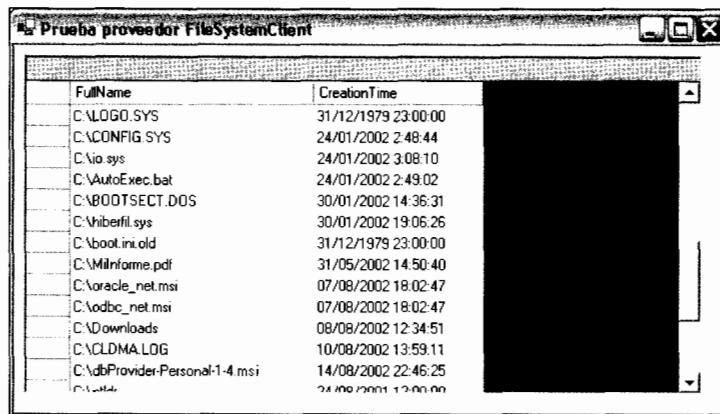
Disponiendo, como es el caso, de una clase de adaptador de datos, **FileSystemClientDataAdapter**, no es necesario usar un **DataReader** para recuperar las filas de forma individual si lo único que queremos es mostrarlas en un formulario. Puede sustituir el **ListBox** por un **DataGrid** y el código anterior por el siguiente:

```
' creamos un adaptador usando el comando
Dim Adaptador As New FileSystemClientDataAdapter(Comando)
' y un nuevo DataSet
Dim Datos As New DataSet()
' lo llenamos con el comando
Adaptador.Fill(Datos)

' Vinculamos el DataGrid con la única tabla del DataSet
DataGrid1.DataSource = Datos.Tables(0)
```

Como puede ver, creamos el adaptador de datos a partir del comando previamente definido. A continuación creamos el **DataSet** y lo llenamos con el método

`Fill()` del adaptador, como haríamos en cualquier otro caso. Por último, vinculamos el `DataSet` con el `DataGridView`, obteniendo el resultado que se puede ver en la figura 22.4.



The screenshot shows a Windows application window titled "Prueba proveedor FileSystemClient". Inside the window is a `DataGridView` control. The grid has two columns: "FullName" and "CreationTime". The data is as follows:

FullName	CreationTime
C:\LOGO.SYS	31/12/1979 23:00:00
C:\CONFIG.SYS	24/01/2002 2:48:44
C:\io.sys	24/01/2002 3:08:10
C:\AutoExec.bat	24/01/2002 2:49:02
C:\BOOTSECT.DOS	30/01/2002 14:36:31
C:\hiberfil.sys	30/01/2002 19:06:26
C:\boot.ini.old	31/12/1979 23:00:00
C:\Milinforme.pdf	31/05/2002 14:50:40
C:\oracle_net.msi	07/08/2002 18:02:47
C:\odbc_net.msi	07/08/2002 18:02:47
C:\Downloads	08/08/2002 12:34:51
C:\CLDMA.LOG	10/08/2002 13:59:11
C:\dbProvider-Personal-1-4.msi	14/08/2002 22:46:25
C:\Windows	24/08/2002 12:00:00

Figura 22.4. El `DataGridView` mostrando los datos recuperados mediante el adaptador



23

Application Blocks
para ADO.NET

No cabe duda de que los componentes ADO.NET, y los asistentes y elementos de Visual Studio .NET conocidos en capítulos previos, como el **Explorador de servidores**, facilitan en gran medida nuestro trabajo. La creación de un objeto `DataSet`, llenándolo de datos con el método `Fill()` de un adaptador, ejecuta en un solo paso una cantidad de operaciones que en otras soluciones se efectúan manualmente.

A pesar de todo, es habitual que nos encontremos, en los capítulos de la segunda parte ha ocurrido, escribiendo una y otra vez bloques de código similares: creación del objeto `Connection`, definición de un `Command` asociándolo con un `DataAdapter` e invocación a su método `Fill()` para obtener el `DataSet`. Estos fragmentos de código, de uso reiterado, son lo que Microsoft denomina *bloques de aplicación* o *Application Blocks* en su denominación original.

Para evitar que tengamos que escribir repetidamente esos bloques de sentencias, Microsoft nos ofrece un paquete de código que, al ser compilado, genera un ensamblado con múltiples métodos compartidos que son capaces de crear directamente un `DataSet`, por poner un ejemplo, a partir de los datos de conexión y comando facilitados como parámetros. El objetivo de este capítulo es mostrarle, básicamente, la funcionalidad de este paquete de ayuda para ADO.NET.

Obtención de *Data Access Application Blocks*

Puede obtener este paquete de la sección **Downloads** de MSDN, en <http://MSDN.Microsoft.com>. Despliegue la rama **.NET Framework** y luego seleccione

el elemento **Data Access Application Block**, tal como se aprecia en la figura 23.1. El archivo no llega al megabyte, por lo que su descarga es relativamente rápida.



Figura 23.1. Sección de MSDN donde se encuentra *Data Access Application Block*

El paquete obtenido es autoinstalable, por lo que bastará con hacer doble clic sobre él para poner en marcha el proceso de instalación en nuestro sistema. Aceptamos la licencia de uso, establecemos el camino de destino (véase figura 23.2) y pulsamos el botón **Next** hasta poner en marcha el proceso de copia de archivos.

La instalación no facilita el ensamblado en formato compilado, ni lo registra como ensamblado compartido, simplemente copia en la carpeta indicada el código fuente, tanto Visual Basic como C#, y la documentación, facilitada en un archivo de ayuda de Windows. También se crea en el menú del botón **Inicio** una carpeta, llamada **Microsoft Applications Block for .NET**, que facilita el acceso al código de estos bloques de aplicación, los ejemplos y la documentación, según se aprecia en el detalle de la figura 23.3.

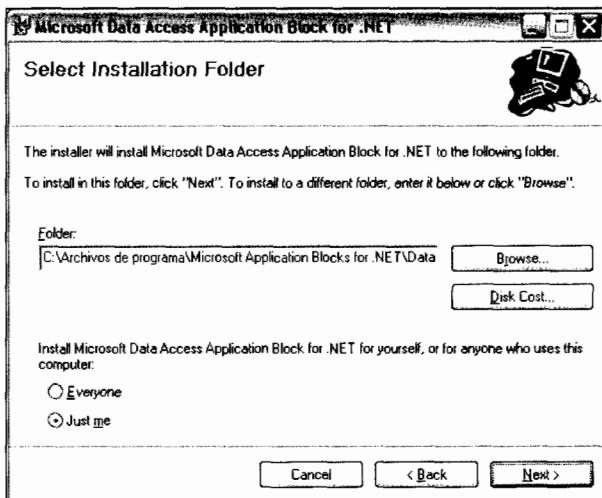


Figura 23.2. Instalación del paquete



Figura 23.3. Accesos directos al código y la documentación

Nota

Los métodos de ayuda definidos en este paquete de código son útiles para trabajar con SQL Server, no siendo aplicables al resto de proveedores .NET. Al disponer del código fuente, sin embargo, sería relativamente fácil adaptarlo para operar con el proveedor que nos interese.

Compilación del ensamblado

Para poder utilizar este código de ayuda, lo primero que tenemos que hacer es compilarlo para generar el correspondiente ensamblado. Abra el elemento **Data Access Application Block** del menú antes indicado. Da igual que use el código Visual Basic .NET o el código C#, ya que el ensamblado obtenido podrá ser utilizado indistintamente.

Se abrirá Visual Studio .NET con el proyecto elegido. Pulse la combinación de teclas **Control-Mayús-B**, o bien elija la opción **Generar>Generar solución**, para compilar el proyecto y obtener el correspondiente módulo DLL. Puede cerrar el proyecto y el entorno de Visual Studio .NET si lo desea. En este momento tenemos

un ensamblado local, que puede ser añadido a cualquier proyecto y distribuido simplemente copiando el módulo Microsoft.ApplicationBlocks.Data.dll en el equipo de destino.

Uso de los métodos SqlHelper

Todos los métodos del ensamblado que hemos obtenido aparecen como métodos compartidos, o estáticos, dentro de estas dos clases: SqlHelper y SqlHelperParameterCache, resultando especialmente interesantes los de la primera. Éstos son los siguientes:

- ExecuteNonQuery(): Ejecuta un comando devolviendo el número de filas afectadas como un entero.
- ExecuteDataSet(): Ejecuta un comando y devuelve un DataSet.
- ExecuteReader(): Ejecuta un comando y devuelve un DataReader.
- ExecuteScalar(): Ejecuta un comando obteniendo un valor único.
- ExecuteXmlReader(): Ejecuta un comando obteniendo un XmlReader.

La diferencia que hay entre estos métodos y algunos de los existentes en un objeto Command con el mismo nombre, es que éstos aceptan una cadena de conexión, una cadena con el comando y devuelven el resultado, sin tener que crear objetos intermedios como el adaptador de datos o el objeto Connection.

Por su parte, los métodos de SqlHelperparameterCache tienen el objetivo de facilitar el almacenamiento de parámetros empleados con consultas y procedimientos almacenados.

Agregar una referencia al ensamblado

Para usar estos métodos desde una aplicación propia, lo primero que tenemos que hacer es agregar una referencia al ensamblado Microsoft.ApplicationBlocks.Data.dll. Para ello seleccione la opción Agregar referencia del proyecto y, ya que no se trata de un ensamblado compartido que se encuentre registrado en la GAC, pulse el botón Examinar para localizar y añadir el ensamblado como se ha hecho en la figura 23.4.

En caso de que vaya a usar los métodos de SqlHelper y SqlHelperParameterCache en múltiples proyectos, quizás le convenga instalar el ensamblado en la GAC abriendolo en Visual Studio y añadiéndole una clave para firmarlo con un identificador único. En este caso el ensamblado aparecerá en la lista de la página .NET y podrá añadirlo directamente a sus proyectos, sin necesidad de localizar el módulo DLL.

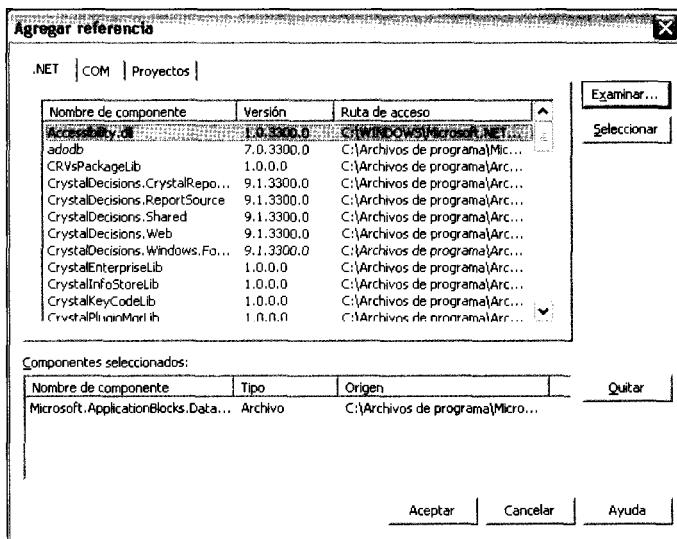


Figura 23.4. Añadimos una referencia al ensamblado

Uso de los métodos compartidos

Asumiendo que ya se ha añadido la referencia al ensamblado, para usar los métodos tendrá que importar el ámbito `Microsoft.ApplicationBlocks.Data` añadiendo luego la correspondiente sentencia `Imports` al inicio del código.

A continuación, bastaría con invocar a los métodos con los parámetros adecuados, como haríamos en cualquier otro caso. Suponiendo que hayamos insertado un `DataGridView` en un formulario, la sentencia siguiente mostraría en él la tabla `Libreros` completa.

```
DataGrid1.DataSource = SqlHelper.ExecuteDataset(
    "Data Source=Inspiron; Initial Catalog=Libros; User ID=sa",
    CommandType.Text, "SELECT * FROM Libros")
```

Observe que no hemos arrastrado elemento alguno del Explorador de servidores, insertado componentes en el formulario o creado objetos intermedios. Basta una llamada a `ExecuteDataSet()` para crear el conjunto de datos con la información deseada. De manera análoga se utilizarían los demás métodos.

Nota

Aunque en este caso se han usado cadenas de caracteres para definir el origen de datos y el comando, también pueden emplearse objetos `SqlConnection` y `SqlCommand` en caso de que los hubiésemos creado con anterioridad.

IDLibro	ISBN	Título	Auto	Editorial	Precio	Poblado
2	1-893115-94-1	User Interface Design for Programmers	Joel Spolsky	3	31	Matrix Byte[]
3	84-415-1136-5	SQL Server 2000	Francisco Ch	1	10,75	Matrix Byte[]
4	84-415-1324-4	Guía práctica para usuarios	Francisco Ch	1	10,75	Matrix Byte[]
5	84-415-1392-9	Programación con Visual C	Francisco Ch	1	39	Matrix Byte[]
6	84-415-1375-7	Programación con Visual Studio .NET	Francisco Ch	1	40	Matrix Byte[]
7	84-415-1351-1	Programación con Visual B	Francisco Ch	1	39	Matrix Byte[]
8	84-415-1290-6	Guía práctica para usuarios	Francisco Ch	1	10,75	(null)
9	84-415-1291-4	Guía práctica para usuarios	Francisco Ch	1	10,52	(null)
10	84-415-1261-2	Programación con Delphi 6	Francisco Ch	1	37,26	(null)
11	84-415-1255-8	Guía práctica para usuarios	Francisco Ch	1	10,52	(null)
12	84-415-1230-2	Manual avanzado Excel 2000	Francisco Ch	1	21,04	(null)
13	84-415-1202-7	Guía práctica para usuarios	Francisco Ch	1	10,52	(null)
14	84-415-1132-2	Guía práctica para usuarios	Francisco Ch	1	10,52	(null)
15	84-415-1145-4	Introducción a la programación	Francisco Ch	1	24,04	(null)
16	84-7615-234-5	Manual del microprocesador	Chris H.Papp	2	40	(null)
17	0-471-37523-3	Assembly Language Step by Step	Jeff Duntema	1	60,5	(null)

Figura 23.5. El DataGrid mostrando el contenido del DataSet devuelto por el método ExecuteDataSet()

Documentación adicional

Además del archivo de ayuda que se obtiene al instalar el paquete, en MSDN podemos encontrar documentación adicional sobre el uso de estas clases de ayuda, concretamente en http://www.microsoft.com/library/en-us/dnbda/html/daab_rm.asp. Se trata de un artículo (véase figura 23.6) en el que se describe esquemáticamente la relación entre los métodos y las clases, sus tipos de datos, etc. También encontrará en él fragmentos de código de ejemplo sobre cómo usar los métodos compartidos, así como otras referencias.

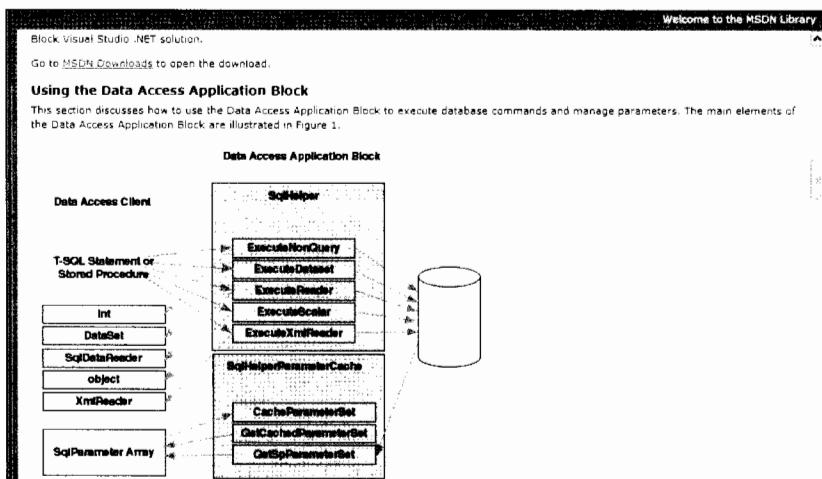
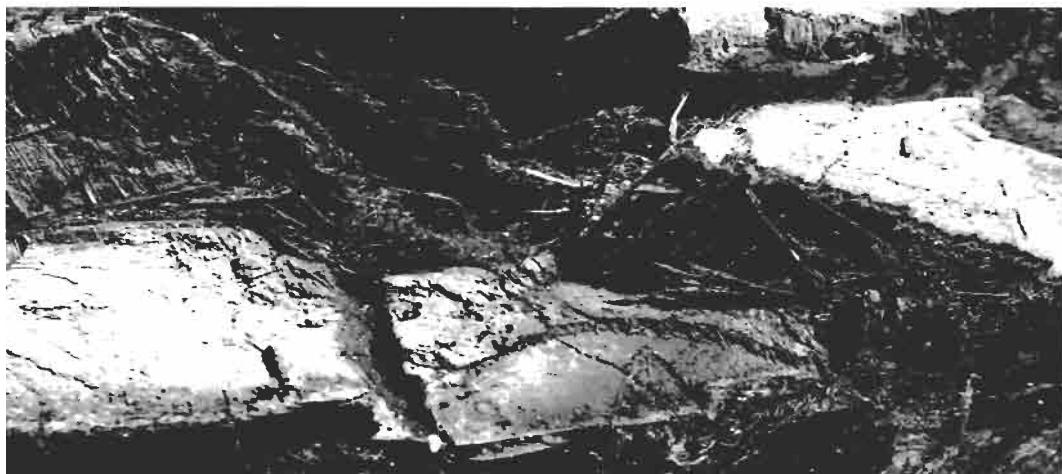


Figura 23.6. Documentación de Data Access Application Blocks for .NET



A

Glosario

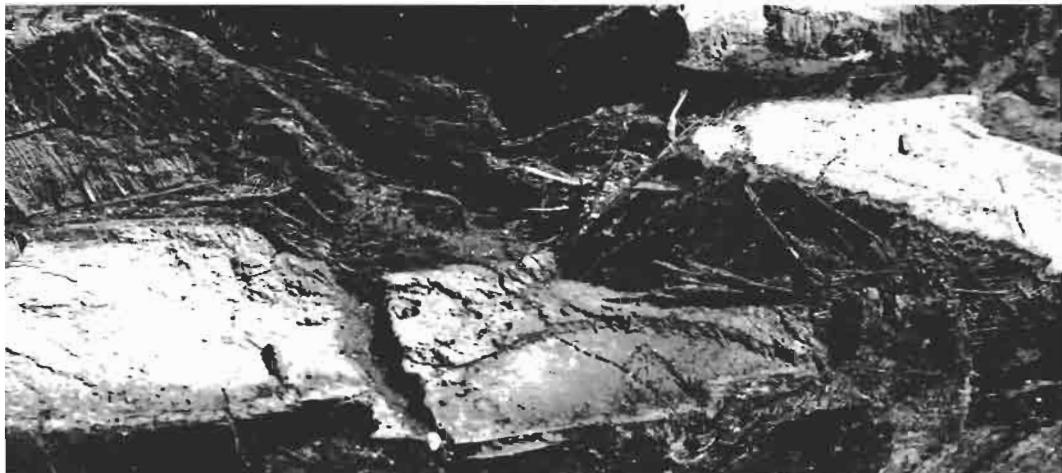
Acrónimos y significado

Acrónimo	Significado	Explicación
ACID	<i>Atomicity, Consistency, Isolation and Durability</i>	Propiedades que debe tener una transacción y que hacen referencia a su individualidad, consistencia, aislamiento y persistencia.
ADO	<i>ActiveX Data Objects</i>	Objetos ActiveX utilizados para el acceso a datos desde Visual Basic 6 y otras herramientas de desarrollo.
ADSI	<i>Active Directory Services Interfaces</i>	Nombre con el que se conoce a las interfaces que facilitan el acceso a los servicios del Directorio activo.
ANSI	<i>American National Standards Institute</i>	Institución americana encargada de promocionar y supervisar estándares.
API	<i>Application Programming Interface</i>	Conjunto de funciones que expone un sistema operativo para poder programarlo.
ASCII	<i>American Standard Code for Information Interchange</i>	Conjunto de caracteres usado como estándar mundial para el intercambio de información entre ordenadores.
BDE	<i>Borland Database Engine</i>	Motor de acceso a datos de la firma Borland.

Acrónimo	Significado	Explicación
CLS	<i>Common Language Specification</i>	Especificación creada por Microsoft para hacer posible la interoperabilidad entre distintos lenguajes de programación en la plataforma .NET.
COBOL	<i>COmmon Business Oriented Language</i>	Uno de los lenguajes de programación más antiguos, utilizado principalmente para crear aplicaciones de negocios.
COM	<i>Component Object Model</i>	Modelo de objetos usado por Microsoft desde la aparición de Windows 95 para facilitar el uso de componentes reutilizables.
DAO	<i>Data Access Objects</i>	Mecanismo de acceso a datos de Microsoft anterior a ADO.
DBA	<i>Database Administrator</i>	Persona especializada que se ocupa de la administración de la base de datos de una empresa.
DBMS	<i>Database Management System</i>	Apelativo con el que se conocen las aplicaciones para la gestión de datos, lo que coloquialmente se conoce como <i>base de datos</i> .
DCL	<i>Data Control Language</i>	Parte de SQL que comprende los comandos de control usados para otorgar permisos y tareas similares.
DDL	<i>Data Definition Language</i>	Parte de SQL compuesta de las sentencias que manipulan las estructuras de los datos.
DLL	<i>Dynamic Link Library</i>	Módulos, conocidos como bibliotecas de enlace dinámico, que contienen código compilado para ser usado desde las aplicaciones.
DML	<i>Data Manipulation Language</i>	Parte de SQL que comprende todas las sentencias de selección y manipulación de datos.
DOM	<i>Document Object Model</i>	Un modelo de objetos pensado para facilitar el acceso a documentos XML creando una imagen de ellos en forma de árbol jerárquico.
DSN	<i>Data Source Name</i>	Mecanismo de ODBC para asociar un nombre o almacenar en un archivo los parámetros de conexión.
DTD	<i>Data Type Definition</i>	Sistema utilizado para definir la estructura de un documento XML.
FK	<i>Foreign Key</i>	Clave externa. Columna de una tabla que hace referencia a una clave primaria de otra tabla.

Acrónimo	Significado	Explicación
GAC	<i>Global Assembly Cache</i>	Depósito donde se almacenan todos los ensamblados compartidos en la plataforma .NET.
HTML	<i>HyperText Markup Language</i>	Lenguaje para el diseño de documentos que se publican en lo que se conoce coloquialmente como <i>Web</i> .
HTTP	<i>Hypertext Transfer Protocol</i>	Protocolo de transferencia de documentos HTML.
JDBC	<i>Java Database Connectivity</i>	Mecanismo de acceso a datos utilizado en el lenguaje Java.
LDAP	<i>Lightweight Directory Access Protocol</i>	Protocolo estándar para el acceso a servicios de directorio.
MDAC	<i>Microsoft Data Access Components</i>	Componentes de acceso a datos de Microsoft. Un paquete compuesto de bibliotecas de enlace dinámico contenido servicios ADO y proveedores OLE DB.
MSDE	<i>Microsoft Data Engine</i>	Motor de base de datos compatible con SQL Server.
MSIL	<i>Microsoft Intermediate Language</i>	Lenguaje en el que generan código todos los compiladores de lenguajes .NET.
ODBC	<i>Open Database Connectivity</i>	Mecanismo estándar de acceso a datos, previo a DAO, ADO y ADO.NET.
OLE DB	<i>Object Linking & Embedding Database</i>	Tecnología para la creación y uso de controladores de acceso a distintos orígenes de datos.
PK	<i>Primary Key</i>	Clave primaria en una tabla.
PL/SQL	<i>Procedural Language/Structured Query Language</i>	Lenguaje procedural derivado de SQL que se emplea en las bases de datos Oracle.
RDBMS	<i>Relational Database Management System</i>	DBMS relacional, actualmente la mayoría de ellos lo son.
SAX	<i>Simple API for XML</i>	Conjunto de funciones simples para el tratamiento de documentos XML.
SGBD	<i>Sistema gestor de bases de datos</i>	Equivalente a DBMS.
SGML	<i>Standarized General Markup Language</i>	Lenguaje estándar de marcas del que se derivan otros, como HTML o XML.
SQL	<i>Structured Query Language</i>	Lenguaje utilizado en la práctica totalidad de los RDBMS para la selección y manipulación de datos y sus estructuras.

Acrónimo	Significado	Explicación
T-SQL	<i>Transact - Structured Query Language</i>	Lenguaje derivado de SQL utilizado en SQL Server.
UDL	<i>Universal Data Link</i>	Formato de archivo para el almacenamiento de vínculos a orígenes de datos.
URI	<i>Universal Resource Identifier</i>	Identificador universal de recursos, compuesto normalmente de un URL y un camino y un nombre de documento.
URL	<i>Universal Resource Locator</i>	Sistema de localización de recursos en Internet.
VBA	<i>Visual Basic for Applications</i>	Versión de Visual Basic específica para su uso embebido en aplicaciones.
W3C	<i>World Wide Web Consortium</i>	Institución que controla la mayoría de los estándares, lenguaje y protocolos, relacionados con la Web.
WWW	<i>World Wide Web</i>	Nombre completo de lo que coloquialmente se denomina <i>Web</i> .
XML	<i>Extensible Markup Language</i>	Lenguaje derivado de SGML que se caracteriza por ser extensible.
XSD	<i>XML Schema Definition</i>	Lenguaje para la creación de esquemas de documentos XML.
XSL	<i>Extensible Markup Language</i>	Lenguaje para la conversión de documentos XML en otros formatos.
XSLT	<i>XSL Transformations</i>	Lenguaje usado en las hojas XSL para efectuar las transformaciones.



B

Contenido del CD-ROM

Este libro incorpora un CD-ROM en el que podrá encontrar todos los ejemplos que se han descrito en sus capítulos, facilitándole así el acceso a todo el código sin necesidad de tener que teclearlo personalmente. En la carpeta *Ejemplos* encontrará una subcarpeta por cada capítulo, en cuyo interior se ha creado una carpeta para cada ejemplo de ese capítulo en concreto.

Además del código fuente de los proyectos también se entregan éstos ya compilados. Éstos, no obstante, no pueden ser ejecutados directamente desde Windows a menos que se tenga instalada la plataforma Microsoft .NET, entregada también en el CD-ROM. En caso de que tenga instalado en su sistema el producto Visual Studio .NET en alguna de sus versiones, ya cuenta también con la plataforma .NET.

En cualquier caso, debe tener en cuenta que los proyectos se entregan como simples ejemplos, nunca como aplicaciones de uso final, por lo que no se otorga garantía alguna ni soporte de su funcionamiento. Nuestro objetivo es facilitarle código sobre el que pueda hacer pruebas y desarrollar sus propios proyectos.

Uso de los ejemplos

Para evitar la dependencia del CD-ROM, lo más recomendable es que cree una carpeta en su disco y copie todo el contenido de *Ejemplos* a ella. Apenas le ocuparán unos diez megabytes. Compruebe que tras la copia los archivos no tienen activo el atributo de sólo lectura.

Los proyectos de consola y formularios Windows puede usarlos directamente, abriéndolos desde Visual Studio .NET. Los de formularios Web, por el contrario, tendrá que copiarlos en carpetas dentro de la raíz que esté utilizando *Internet Information Server*. Esa raíz suele ser C:\InetPub\wwwroot, cambiando la letra por la que corresponda. De no hacerlo así no podrá acceder a los ejemplos Web desde Internet Explorer correctamente.

Nota

También puede acceder a la consola de administración de IIS y crear un nuevo directorio virtual de aplicación que apunte directamente a la carpeta en la que están los ejemplos, en lugar de copiar éstos a la raíz del sitio Web por defecto.

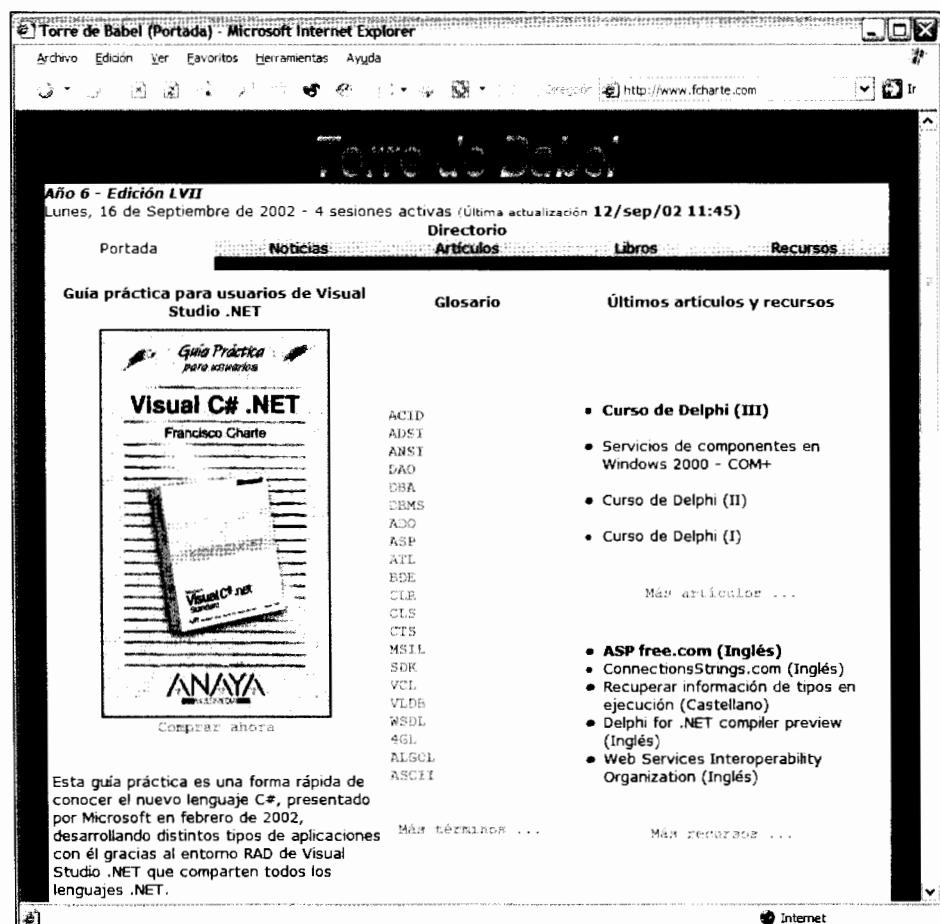


Figura B.1. Sede web de Torre de Babel

Por lo demás no es necesario ningún paso adicional para poder usar los ejemplos. No olvide que para comprender las técnicas teóricas es fundamental la práctica y que, aunque se facilite el código para que no tenga que introducirlo manualmente, es importante que lo examine cuidadosamente y que lo utilice para experimentar, introduciendo cambios y usándolo como base para sus propios ejemplos.

Atención al lector

Tanto la editorial como el autor de este libro se ponen a disposición del lector para cualquier problema que pudiera encontrar en el uso del CD-ROM, su contenido o bien el del libro que tiene en sus manos. En la web de Anaya Multimedia, <http://www.AnayaMultimedia.com>, podrá encontrar información adicional, en caso de que la hubiera, así como el enlace necesario para solicitar ayuda. De igual forma, el autor mantiene una sede, llamada *Torre de Babel*, en la dirección <http://www.fcharte.com>, con datos sobre éste y otros libros, artículos y desde donde puede ponerse en contacto con él en caso de que encuentre algún tipo de problema.

@@IDENTITY - Transact-SQL, 250

A

AcceptChanges()

 DataSet, 165, 261

 DataTable, 166

ACID - Glosario, 501

Add() - DataRowCollection, 166

AddNew() - DataView, 169, 276

ADO - Glosario, 501

ADSI - Glosario, 501

AllowDBNull - DataColumn, 168

AllowDelete - DataView, 276

AllowEdit - DataView, 276

AllowNew - DataView, 276

ALTER - SQL, 50

ALTER TABLE - SQL, 54

ALTER VIEW - SQL, 56

Ámbitos

 Microsoft.Data.Odbc, 155

System.Data, 154

System.Data.Common, 154

System.Data.OleDb, 154

System.Data.OracleClient, 155

System.Data.SqlClient, 154

 System.Data.SqlTypes, 155

AND - SQL, 59

ANSI - Glosario, 501

API - Glosario, 501

AppendChild() - XmlNode, 303

ASC

 CREATE INDEX, 55

 ORDER BY, 61

ASCII - Glosario, 501

ASP.NET

 AutoPostBack, 375

 DataBind(), 372

 DataBinder, 372

 DataList, 380

 Repeater, 380

Attributes - XmlNode, 302

AutoIncrement - DataColumn, 168
AutoPostBack - ASP.NET, 375
AVG() - SELECT, 62

B

BDE - Glosario, 501
BEGIN - Transact-SQL, 68
BEGIN TRANSACTION -
 Transact-SQL, 436
BeginEdit()
 DataRow, 167
 DataRowView, 276
BeginTransaction() - IDbConnection,
 161, 178, 438
BETWEEN - WHERE, 59
BFILE - PL/SQL, 70
bigint - Transact-SQL, 67
binary - Transact-SQL, 67
BindingContext - Form, 377
BindingManagerBase
 CurrentChanged, 379
 PositionChanged, 379
 Refresh(), 379
 ResumeBinding(), 379
 SuspendBindings(), 379
 System.Windows.Forms, 378
BREAK - Transact-SQL, 68
Broken - ConnectionState, 176

C

CancelCommand - DataGridView, 406
CancelEdit()
 DataRow, 167
 DataRowView, 276
CaseSensitive - DataSet, 237
Cells - DataGridViewItem, 406
ChangeDatabase() -
 IDbConnection, 178
CHAR

PL/SQL, 70
SQL, 52
char - Transact-SQL, 67
ChildColumns - DataRelation, 169
ChildNodes - XmlNode, 302
ChildRelations - DataTable, 166
ChildTable - DataRelation, 169
Close()
 IDataReader, 158
 IDbConnection, 161, 176
CloseConnection -
 CommandBehavior, 218
Closed - ConnectionState, 176
CLS - Glosario, 502
COBOL - Glosario, 502
ColumnMappings -
 ITableMapping, 156
ColumnName - DataColumn, 167, 454
Columns
 DataTable, 165, 236
 OleDbSchemaGuid, 203
COM - Glosario, 502
ComboBox - Controles, 370
CommandBehavior
 CloseConnection, 218
 Default, 218
 KeyInfo, 218
 SchemaOnly, 209
 SchemaOnly, 218
 SingleResult, 218
 SingleRow, 218
CommandText -
 IDbCommand, 161, 217
CommandType
 IDbCommand, 161, 217
 StoredProcedure, 232
 TableDirect, 220
COMMIT TRANSACTION -
 Transact-SQL, 436
Commit() - IDbTransaction, 161, 438
complexType - XSD, 125

- Conceptos
 DataReader, 144
 DataSet, 141
- Connecting
 ConnectionState, 176
 IDbCommand, 161
 IDbTransaction, 438
- ConnectionString
 Broken, 176
 Closed, 176
 Connecting, 176
 Executing, 176
 Fetching, 176
 IDbConnection, 176
 Open, 176
- ConnectionString -
 IDbConnection, 161, 176
- ConnectionTimeout -
 IDbConnection, 178
- Constraint
 ConstraintName, 168
 System.Data, 168
- ConstraintCollection -
 System.Data, 165
- ConstraintName - Constraint, 168
- Constraints - DataTable, 165, 236
- CONTINUE - Transact-SQL, 68
- Control - DataBindings, 370
- Controles
 ComboBox, 370
 DataGridView, 370
 ListBox, 370
 TextBox, 370
- Count - CurrencyManager, 379
- COUNT() - SELECT, 61
- CREATE - SQL, 50
- CREATE FUNCTION - PL/SQL, 72
- CREATE INDEX
 ASC, 55
 DESC, 55
 SQL, 54
- CREATE PROCEDURE
 PL/SQL, 72
 Transact-SQL, 69
- CREATE ROLE - SQL, 65
- CREATE USER - SQL, 65
- CREATE VIEW - SQL, 55
- CreateCommand() - IDbConnection,
 161, 178, 216
- CreateDataView() -
 DataViewManager, 277
- CreateNavigator() - XmlNode, 303
- CreateParameter - IDbCommand, 226
- CurrencyManager
 Count, 379
 Position, 379
 System.Windows.Forms, 378
- Current - XPathNodeIterator, 306
- CurrentChanged -
 BindingManagerBase, 379
- cursor - Transact-SQL, 67
- D**
- DAO - Glosario, 502
- DataAdapter -
 System.Data.Common, 159
- Database - IDbConnection, 177
- DATABASE - SQL, 50
- DataBind() - ASP.NET, 372
- DataBinder - ASP.NET, 372
- DataBindings - Control, 370
- DataColumn
 AllowDBNull, 168
 AutoIncrement, 168
 ColumnName, 167, 454
 DataType, 167, 454
 DefaultValue, 168
 Expression, 454
 MaxLength, 167
 System.Data, 165
 Unique, 168

- DataColumnMapping -
 - System.Data.Common, 156
- DataColumnMappingCollection -
 - System.Data.Common, 156
- DataGrid
 - CancelCommand, 406
 - Controles, 370
 - DataKeyField, 376
 - EditCommand, 406
 - EditItemIndex, 406
 - UpdateCommand, 406
- DataGridItem
 - Cells, 406
 - ItemIndex, 406
 - System.Windows.Forms, 406
- DataKeyField - DataGrid, 376
- DataList
 - ASP.NET, 380
 - ItemTemplate, 381
- DataMember - ListBox, 375
- DataReader - Conceptos, 144
- DataRelation
 - ChildColumns, 169
 - ChildTable, 169
 - ParentColumns, 169
 - ParentTable, 169
 - System.Data, 169
- DataRelationCollection -
 - System.Data, 164
- DataRow
 - BeginEdit(), 167
 - CancelEdit(), 167
 - Delete(), 166
 - EndEdit(), 167
 - GetColumnsInError(), 167
 - HasErrors, 167
 - HasVersion(), 167
 - Item, 166
 - RowState, 166, 241
 - System.Data, 165
- DataRowCollection - Add(), 166
- DataRowVersion -
 - System.Data, 167
- DataRowView
 - BeginEdit(), 276
 - CancelEdit(), 276
 - EndEdit(), 276
 - System.Data, 276
- DataSet
 - AcceptChanges(), 165, 261
 - CaseSensitive, 237
 - Conceptos, 141
 - DataSetName, 164
 - DataViewManager, 277
 - DefaultViewManager, 277
 - EnforceConstraints, 168
 - HasChanges(), 164
 - ReadXml(), 165
 - ReadXmlSchema(), 165, 260
 - RejectChanges(), 165
 - Relations, 164, 236
 - System.Data, 163
 - Tables, 164, 236
 - WriteXml(), 165, 260
 - WriteXmlSchema(), 165, 260
 - XmlDataDocument, 310
- DataSetColumn -
 - IColumnMapping, 156
- DataSetName - DataSet, 164
- DataSetTable - ITableMapping, 156
- DataSource - ListBox, 372
- DataTable
 - AcceptChanges(), 166
 - ChildRelations, 166
 - Columns, 165, 236
 - Constraints, 165, 236
 - DefaultView, 279
 - GetChildRows(), 286
 - NewRow(), 166
 - ParentRelations, 166
 - PrimaryKey, 165
 - RejectChanges(), 166

RowDeleted, 166
RowDeleting, 166
Rows, 165, 236
Select(), 166, 237, 272
System.Data, 165
DataTableCollection -
 System.Data, 164
DataTableMapping -
 System.Data.Common, 156
DataTableMappingCollection -
 System.Data.Common, 156
DataTextField - ListBox, 375
DataType - DataColumn, 167, 454
DataValueField - ListBox, 375
 DataView
 AddNew(), 169, 276
 AllowDelete, 276
 AllowEdit, 276
 AllowNew, 276
 Delete(), 169, 276
 Find(), 169, 281
 FindRows(), 169, 281
 Item, 169, 276
 RowFilter, 169
 RowStateFilter, 276
 Sort, 169, 277
 StateFilter, 169
 System.Data, 169, 271
 DataViewManager
 CreateDataGridView(), 277
 DataSet, 277
 DataGridViewSettings, 277
 System.Data, 271
 DataGridViewSettings -
 DataGridViewManager, 277
 DATE
 PL/SQL, 70
 SQL, 52
 DBA - Glosario, 502
 DbDataAdapter -
 System.Data.Common, 159
 DbDataPermission -
 System.Data.Common, 163
 DbDataPermissionAttribute -
 System.Data.Common, 163
 DBMS - Glosario, 502
 DCL - Glosario, 502
 DDL - Glosario, 502
 DECIMAL - PL/SQL, 70
 decimal - Transact-SQL, 67
 DECLARE
 PL/SQL, 70
 Transact-SQL, 67
 Default - CommandBehavior, 218
 DEFAULT - PL/SQL, 72
 DefaultValue - DataColumn, 168
 DefaultView - DataTable, 279
 DefaultViewManager - DataSet, 277
 DELETE
 SQL, 56
 UPDATE, 65
 Delete()
 DataRow, 166
 DataGridView, 169, 276
 DeleteCommand -
 IDbDataAdapter, 159
 DeleteRule -
 ForeignConstraint, 168
 DESC
 CREATE INDEX, 55
 ORDER BY, 61
 DirectoryEntry -
 System.DirectoryServices, 134
 DirectorySearcher -
 System.DirectoryServices, 134
 DisplayMember - ListBox, 372
 DLL - Glosario, 502
 DML - Glosario, 502
 DocumentElement -
 XmlDocument, 302
 DOM - Glosario, 502
 DOUBLE PRECISION - PL/SQL, 70

DROP - SQL, 50
DROP INDEX - SQL, 55
DROP TABLE - SQL, 54
DROP VIEW - SQL, 56
DSN - Glosario, 502
DTD - Glosario, 502

E

EditCommand - DataGrid, 406
EditItemIndex - DataGrid, 406
element - XSD, 126
ELSE
 PL/SQL, 71
 Transact-SQL, 68
ELSIF - PL/SQL, 71
END - Transact-SQL, 68
EndEdit()
 DataRow, 167
 DataRowView, 276
EnforceConstraints - DataSet, 168
EXECUTE - Transact-SQL, 95
ExecuteDataSet() - SqlHelper, 496
ExecuteNonQuery()
 IDbCommand, 218
 SqlHelper, 496
ExecuteOracleNonQuery() -
 OracleCommand, 219
ExecuteOracleScalar() -
 OracleCommand, 219
ExecuteReader()
 IDbCommand, 218
 SqlHelper, 496
ExecuteScalar()
 IDbCommand, 218
 SqlHelper, 496
ExecuteXmlReader()
 SqlCommand, 219
 SqlHelper, 496
Executing - ConnectionState, 176
Expression - DataColumn, 454

F

Fetching - ConnectionState, 176
FieldCount - IDataRecord, 157
Fill() - SqlDataAdapter, 159, 239
FillSchema() - SqlDataAdapter, 247
Find() - DataView, 169, 281
FindRows() - DataView, 169, 281
FirstChild - XmlNode, 302
FK - Glosario, 502
FLOAT
 PL/SQL, 70
 SQL, 52
float - Transact-SQL, 67
FOR - PL/SQL, 71
for-each - XSLT, 307
FOREIGN KEY - SQL, 53
ForeignConstraint
 DeleteRule, 168
 System.Data, 168
 UpdateRule, 168
Form - BindingContext, 377
FROM - SQL, 58
FromStream() - PictureBox, 464

G

GAC - Glosario, 503
GetBoolean() - IDataRecord, 157
GetByte() - IDataRecord, 157
GetChildRows() - DataTable, 286
GetColumnsInError() - DataRow, 167
GetDeleteCommand() -
 SqlCommandBuilder, 163
GetElementFromRow() -
 XmlDataDocument, 310
GetFloat() - IDataRecord, 157
GetInsertCommand() -
 SqlCommandBuilder, 163
GetOleDbSchemaTable() -
 OleDbConnection, 202

GetOracleBinary() -	JDBC, 503
OracleDataReader, 158	LDAP, 503
GetOracleNumber() -	MDAC, 503
OracleDataReader, 158	MSDE, 503
GetOracleString() -	MSIL, 503
OracleDataReader, 158	ODBC, 503
GetRowFromElement() -	OLE DB, 503
XmlDataDocument, 310	PK, 503
GetSchemaTable() -	PL/SQL, 503
IDataReader, 209	RDBMS, 503
GetSqlByte() - SqlDataReader, 158	SAX, 503
GetSqlDouble() - SqlDataReader, 158	SGBD, 503
GetSqlString() - SqlDataReader, 158	SGML, 503
GetString() - IDataRecord, 157	SQL, 503
GetUpdateCommand() -	T-SQL, 504
SqlCommandBuilder, 163	UDL, 504
Glosario	URI, 504
ACID, 501	URL, 504
ADO, 501	VBA, 504
ADSI, 501	W3C, 504
ANSI, 501	WWW, 504
API, 501	XML, 504
ASCII, 501	XSD, 504
BDE, 501	XSL, 504
CLS, 502	XSLT, 504
COBOL, 502	GRANT - SQL, 65
COM, 502	GROUP BY - SELECT, 62
DAO, 502	
DBA, 502	
DBMS, 502	
DCL, 502	
DDL, 502	
DLL, 502	
DML, 502	
DOM, 502	
DSN, 502	
DTD, 502	
FK, 502	
GAC, 503	
HTML, 503	
HTTP, 503	
	H
	HasChanges() - DataSet, 164
	HasChildNodes - XmlNode, 302
	HasErrors - DataRow, 167
	HasVersion() - DataRow, 167
	HTML - Glosario, 503
	HTTP - Glosario, 503
	I
	IBindingList - System.Data, 275
	IColumnMapping

DataSetColumn, 156
SourceColumn, 156
System.Data, 156
IColumnMappingCollection -
 System.Data, 156
IDataAdapter
 Fill(), 159, 239
 FillSchema(), 247
 MissingMappingAction, 239
 MissingSchemaAction, 239
 System.Data, 159, 414
 Update(), 159, 241
IDataReader
 Close(), 158
 GetSchemaTable(), 209
 NextResult(), 158, 222
 Read(), 157
 RecordsAffected, 157
 System.Data, 157, 414
IDataRecord
 FieldCount, 157
 GetBoolean(), 157
 GetByte(), 157
 GetFloat(), 157
 GetString(), 157
 Item, 157
 System.Data, 157
IDbCommand
 CommandText, 161, 217
 CommandType, 161, 217
 Connection, 161
 CreateParameter, 226
 ExecuteNonQuery(), 218
 ExecuteReader(), 218
 ExecuteScalar(), 218
 Parameters, 217
 Properties, 232
 System.Data, 161, 414
IDbConnection
 BeginTransaction(), 161, 178, 438
 ChangeDatabase(), 178
 Close(), 161, 176
 ConnectionState, 176
 ConnectionString, 161, 176
 ConnectionTimeout, 178
 CreateCommand(), 161, 178, 216
 Database, 177
 Open(), 161, 176
 System.Data, 161, 176, 414
IDbDataAdapter
 DeleteCommand, 159
 InsertCommand, 159
 RowUpdated, 445
 SelectCommand, 159
 System.Data, 159, 414
 UpdateCommand, 159
IDbTransaction
 Commit(), 161, 438
 Connection, 438
 IsolationLevel, 161, 438
 Rollback(), 161, 438
 System.Data, 161, 437
IF
 PL / SQL, 71
 Transact-SQL, 68
image - Transact-SQL, 67
IN - WHERE, 59
INDEX - SQL, 51
INFORMATION_SCHEMA -
 Transact-SQL, 206
INSERT - SQL, 56
INSERT INTO - SQL, 57
InsertAfter() - XmlNode, 303
InsertBefore() - XmlNode, 303
InsertCommand -
 IDbDataAdapter, 159
int - Transact-SQL, 67
INTEGER - SQL, 52
InterBase
 RETNAL, 118
 SET TERM, 119
 SUSPEND, 119

IsolationLevel -
 IDbTransaction, 161, 438

ITableMapping
 ColumnMappings, 156
 DataSetTable, 156
 SourceTable, 156
 System.Data, 156

ITableMappingCollection -
 System.Data, 156

Item
 DataRow, 166
 DataView, 169, 276
 IDataRecord, 157

ItemIndex - DataGridViewItem, 406

ItemTemplate - DataList, 381

IXPathNavigable -
 System.Xml, 302

J

JDBC - Glosario, 503

K

KeyInfo - CommandBehavior, 218

L

LastChild - XmlNode, 303

LDAP - Glosario, 503

LIKE - WHERE, 59

ListBox

Controles, 370

DataMember, 375

DataSource, 372

DataTextField, 375

DataValueField, 375

DisplayMember, 372

ValueMember, 372

Load() - XmlDocument, 302

LOOP - PL/SQL, 71

M

MAX() - SELECT, 62

MaxLength - DataColumn, 167

MDAC - Glosario, 503

Microsoft.Data.Odbc
 Ámbitos, 155
 OdbcDataAdapter, 159
 OdbcDataReader, 158

MIN() - SELECT, 62

MissingMappingAction -
 IDataAdapter, 239

MissingSchemaAction -
 IDataAdapter, 239

money - Transact-SQL, 67

MoveNext() -
 XPathNodeIterator, 306

MSDE - Glosario, 503

MSIL - Glosario, 503

N

Name - XmlNode, 302

NATURAL - PL/SQL, 70

NewRow() - DataTable, 166

NextResult() - IDataReader, 158, 222

NextSibling - XmlNode, 303

NodeType - XmlNode, 302

NOT - SQL, 59

NOT NULL - SQL, 52

NULL - SQL, 57

O

ODBC - Glosario, 503

OdbcCommand -
 System.Data.Odbc, 162

OdbcCommandBuilder -
 System.Data.Odbc, 163

OdbcConnection -
 System.Data.Odbc, 162

OdbcDataAdapter -
 Microsoft.Data.Odbc, 159

OdbcDataReader -
 Microsoft.Data.Odbc, 158

OdbcTransaction -
 System.Data.Odbc, 162

OLE DB - Glosario, 503

OleDbCommand -
 System.Data.OleDb, 162

OleDbCommandBuilder -
 System.Data.OleDb, 163

OleDbConnection
 GetOleDbSchemaTable(), 202
 System.Data.OleDb, 162

OleDbDataAdapter -
 System.Data.OleDb, 159

OleDbDataReader -
 System.Data.OleDb, 158

OleDbSchemaGuid
 Columns, 203
 Procedure_Parameters, 203
 Procedures, 203
 Table_Constraints, 203
 Tables, 203
 Views, 203

OleDbTransaction -
 System.Data.OleDb, 162

Open - ConnectionState, 176

OPEN - PL/SQL, 109

Open() - IDbConnection, 161, 176

OR - SQL, 59

OracleCommand
 ExecuteNonQuery(), 219
 ExecuteScalar(), 219
 System.Data.OracleClient, 162

OracleCommandBuilder -
 System.Data.OracleClient, 163

OracleConnection -
 System.Data.OracleClient, 162

OracleDataAdapter -
 System.Data.OracleClient, 159

OracleDataReader
 GetOracleBinary(), 158
 GetOracleNumber(), 158
 GetOracleString(), 158
 System.Data.OracleClient, 158

OracleTransaction -
 System.Data.OracleClient, 162

ORDER BY
 ASC, 61
 DESC, 61
 SQL, 61

P

Parameters - IDbCommand, 217

ParentColumns - DataRelation, 169

ParentNode - XmlNode, 303

ParentRelations - DataTable, 166

ParentTable - DataRelation, 169

PictureBox - FromStream(), 464

PK - Glosario, 503

PL/SQL
 BFILE, 70
 CHAR, 70
 CREATE FUNCTION, 72
 CREATE PROCEDURE, 72
 DATE, 70
 DECIMAL, 70
 DECLARE, 70
 DEFAULT, 72
 DOUBLE PRECISION, 70
 ELSE, 71
 ELSIF, 71
 FLOAT, 70
 FOR, 71
 Glosario, 503
 IF, 71
 LOOP, 71
 NATURAL, 70
 OPEN, 109
 POSITIVE, 70

- REF CURSOR, 109
 RETURN, 72
 ROWID, 70
 SIGNTYPE, 70
 STRING, 70
 THEN, 71
 VARCHAR, 70
 WHILE, 71
Position - CurrencyManager, 379
PositionChanged -
 BindingManagerBase, 379
POSITIVE - PL/SQL, 70
PreviousSibling - XmlNode, 303
PRIMARY KEY - SQL, 52
PrimaryKey - DataTable, 165
PROCEDURE - SQL, 51
Procedure_Parameters -
 OleDbSchemaGuid, 203
Procedures -
 OleDbSchemaGuid, 203
Properties - IDbCommand, 232
PropertyManager -
 System.Windows.Forms, 378
- R**
- RDBMS - Glosario, 503
Read() - IDataReader, 157
ReadXml() - DataSet, 165
ReadXmlSchema() - DataSet, 165, 260
RecordsAffected - IDataReader, 157
REF CURSOR - PL/SQL, 109
REFERENCES - SQL, 53
Refresh() - BindingManagerBase, 379
RejectChanges()
 DataSet, 165
 DataTable, 166
Relations - DataSet, 164, 236
RemoveChild() - XmlNode, 303
Repeater - ASP.NET, 380
ReplaceChild() - XmlNode, 303
- ResumeBinding() -
 BindingManagerBase, 379
RETURN
 PL/SQL, 72
 Transact-SQL, 69
RETVAL - InterBase, 118
REVOKE - SQL, 65
ROLLBACK TRANSACTION -
 Transact-SQL, 436
Rollback() - IDbTransaction, 161, 438
RowDeleted - DataTable, 166
RowDeleting - DataTable, 166
RowFilter - DataView, 169
ROWID - PL/SQL, 70
Rows - DataTable, 165, 236
RowState - DataRow, 166, 241
RowStateFilter - DataView, 276
RowUpdated - IDbDataAdapter, 445
RowUpdatedEventArgs -
 System.Data.Common, 163
RowUpdatingEventArgs -
 System.Data.Common, 163

S

- SAX - Glosario**, 503
SchemaOnly -
 CommandBehavior, 209, 218
SELECT
 AVG(), 62
 COUNT(), 61
 GROUP BY, 62
 MAX(), 62
 MIN(), 62
 SQL, 56
 SUM(), 61
Select()
 DataTable, 166, 237, 272
 XPathNavigator, 306
SelectCommand -
 IDbDataAdapter, 159

SelectNodes() - XmlNode, 303
SET - UPDATE, 64
SET TERM - InterBase, 119
SGBD - Glosario, 503
SGML - Glosario, 503
SIGNTYPE - PL/SQL, 70
SingleResult - CommandBehavior, 218
SingleRow - CommandBehavior, 218
SMALLINT - SQL, 52
smallint - Transact-SQL, 67
Sort - DataView, 169, 277
SourceColumn - IColumnMapping, 156
SourceTable - ITableMapping, 156
SQL
 ALTER, 50
 ALTER TABLE, 54
 ALTER VIEW, 56
 AND, 59
 CHAR, 52
 CREATE, 50
 CREATE INDEX, 54
 CREATE ROLE, 65
 CREATE USER, 65
 CREATE VIEW, 55
 DATABASE, 50
 DATE, 52
 DELETE, 56
 DROP, 50
 DROP INDEX, 55
 DROP TABLE, 54
 DROP VIEW, 56
 FLOAT, 52
 FOREIGN KEY, 53
 FROM, 58
 Glosario, 503
 GRANT, 65
 INDEX, 51
 INSERT, 56
 INSERT INTO, 57
 INTEGER, 52
 NOT, 59
 NOT NULL, 52
 NULL, 57
 OR, 59
 ORDER BY, 61
 PRIMARY KEY, 52
 PROCEDURE, 51
 REFERENCES, 53
 REVOKE, 65
 SELECT, 56
 SMALLINT, 52
 TABLE, 50
 TIME, 52
 TRIGGER, 51
 UNIQUE, 52
 UPDATE, 56, 64
 VALUES, 57
 VARCHAR, 52
 VIEW, 51
 WHERE, 59
SqlCommand
 ExecuteXmlReader(), 219
 System.Data.SqlClient, 162
SqlCommandBuilder
 GetDeleteCommand(), 163
 GetInsertCommand(), 163
 GetUpdateCommand(), 163
 System.Data.SqlClient, 163
SqlConnection -
 System.Data.SqlClient, 162
SqlDataAdapter -
 System.Data.SqlClient, 159
SqlDataReader
 GetSqlByte(), 158
 GetSqlDouble(), 158
 GetSqlString(), 158
 System.Data.SqlClient, 158
SqlHelper
 ExecuteDataSet(), 496
 ExecuteNonQuery(), 496
 ExecuteReader(), 496
 ExecuteScalar(), 496

- ExecuteXmlReader(), 496
SqlTransaction -
 System.Data.SqlClient, 162
StateFilter - DataView, 169
StoredProcedured - CommandType, 232
STRING - PL/SQL, 70
SUM() - SELECT, 61
SUSPEND - InterBase, 119
SuspendBindings() -
 BindingManagerBase, 379
System.Data
 Ámbitos, 154
 Constraint, 168
 ConstraintCollection, 165
 DataColumn, 165
 DataRelation, 169
 DataRelationCollection, 164
 DataRow, 165
 DataRowVersion, 167
 DataRowView, 276
 DataSet, 163
 DataTable, 165
 DataTableCollection, 164
 DataView, 169, 271
 DataViewManager, 271
 ForeignConstraint, 168
 IBindingList, 275
 IColumnMapping, 156
 IColumnMappingCollection, 156
 IDataAdapter, 159, 414
 IDataReader, 157, 414
 IDataRecord, 157
 IDbCommand, 161, 414
 IDbConnection, 161, 176, 414
 IDbDataAdapter, 159, 414
 IDbTransaction, 161, 437
 ITableMapping, 156
 ITableMappingCollection, 156
 UniqueConstraint, 168
System.Data.Common
 Ámbitos, 154
DataAdapter, 159
DataColumnMapping, 156
DataColumnMappingCollection, 156
DataTableMapping, 156
DataTableMappingCollection, 156
DbDataAdapter, 159
DbDataPermission, 163
DbDataPermissionAttribute, 163
RowUpdatedEventArgs, 163
RowUpdatingEventArgs, 163
System.Data.Odbc
 OdbcCommand, 162
 OdbcCommandBuilder, 163
 OdbcConnection, 162
 OdbcTransaction, 162
System.Data.OleDb
 Ámbitos, 154
 OleDbCommand, 162
 OleDbCommandBuilder, 163
 OleDbConnection, 162
 OleDbDataAdapter, 159
 OleDbDataReader, 158
 OleDbTransaction, 162
System.Data.OracleClient
 Ámbitos, 155
 OracleCommand, 162
 OracleCommandBuilder, 163
 OracleConnection, 162
 OracleDataAdapter, 159
 OracleDataReader, 158
 OracleTransaction, 162
System.Data.SqlClient
 Ámbitos, 154
 SqlCommand, 162
 SqlCommandBuilder, 163
 SqlConnection, 162
 SqlDataAdapter, 159
 SqlDataReader, 158
 SqlTransaction, 162
System.Data.SqlTypes - Ámbitos, 155
System.DirectoryServices

- DirectoryEntry, 134
DirectorySearcher, 134
System.Windows.Forms
 BindingManagerBase, 378
 CurrencyManager, 378
 DataGridViewItem, 406
 PropertyManager, 378
System.Xml
 IXPathNavigable, 302
 XmlDataDocument, 302
 XmlDocument, 297
 XmlNode, 302
 XmlNodeReader, 299
 XmlReader, 296
 XmlTextReader, 299
 XmlValidatingReader, 299
 XmlWriter, 302
System.Xml.XPath
 XPathDocument, 297
 XPathNavigator, 306
 XPathNodeIterator, 306
System.Xml.Xsl
 Transform(), 308
 XslTransform, 298, 308
- T**
- TABLE - SQL, 50
table - Transact-SQL, 67
Table_Constraints -
 OleDbSchemaGuid, 203
TableDirect - CommandType, 220
Tables
 DataSet, 164, 236
 OleDbSchemaGuid, 203
targetSchema - XML, 129
template - XSLT, 307
text - Transact-SQL, 67
TextBox - Controles, 370
THEN - PL/SQL, 71
TIME - SQL, 52
- Transact-SQL
 @@IDENTITY, 250
 BEGIN, 68
 BEGIN TRANSACTION, 436
 bigint, 67
 binary, 67
 BREAK, 68
 char, 67
 COMMIT TRANSACTION, 436
 CONTINUE, 68
 CREATE PROCEDURE, 69
 cursor, 67
 decimal, 67
 DECLARE, 67
 ELSE, 68
 END, 68
 EXECUTE, 95
 float, 67
 IF, 68
 image, 67
 INFORMATION_SCHEMA, 206
 int, 67
 money, 67
 RETURN, 69
 ROLLBACK TRANSACTION, 436
 smallint, 67
 table, 67
 text, 67
 varbinary, 67
 varchar, 67
 WHILE, 68
 Transform() - System.Xml.Xsl, 308
 TRIGGER - SQL, 51
 T-SQL - Glosario, 504
- U**
- UDL - Glosario, 504
Unique - DataColumn, 168
UNIQUE - SQL, 52
UniqueConstraint - System.Data, 168

- UPDATE
 DELETE, 65
 SET, 64
 SQL, 56
 SQL, 64
- Update() - *IDataAdapter*, 159, 241
- UpdateCommand
 DataGrid, 406
 `IDbDataAdapter`, 159
- UpdateRule - *ForeignConstraint*, 168
- URI - Glosario, 504
- URL - Glosario, 504
- V**
- ValidationType -
 `XmlValidatingReader`, 300
- Value - *XmlNode*, 302
- ValueMember - *ListBox*, 372
- value-of - XSLT, 307
- VALUES - SQL, 57
- varbinary - Transact-SQL, 67
- VARCHAR
 PL/SQL, 70
 SQL, 52
- varchar - Transact-SQL, 67
- VBA - Glosario, 504
- VIEW - SQL, 51
- Views - *OleDbSchemaGuid*, 203
- W**
- W3C - Glosario, 504
- WHERE
 BETWEEN, 59
 IN, 59
 LIKE, 59
 SQL, 59
- WHILE
 PL/SQL, 71
 Transact-SQL, 68
- WriteXml() - *DataSet*, 165, 260
- WriteXmlSchema() - *DataSet*, 165, 260
- WWW - Glosario, 504
- X**
- XML
 Glosario, 504
 targetSchema, 129
- XmlDataDocument
 `DataSet`, 310
 GetElementFromRow(), 310
 GetRowFromElement(), 310
 System.Xml, 302
- XmlElement
 DocumentElement, 302
 Load(), 302
 System.Xml, 297
- XmlNode
 AppendChild(), 303
 Attributes, 302
 ChildNodes, 302
 CreateNavigator(), 303
 FirstChild, 302
 HasChildNodes, 302
 InsertAfter(), 303
 InsertBefore(), 303
 LastChild, 303
 Name, 302
 NextSibling, 303
 NodeType, 302
 ParentNode, 303
 PreviousSibling, 303
 RemoveChild(), 303
 ReplaceChild(), 303
 SelectNodes(), 303
 System.Xml, 302
 Value, 302
- XmlNodeReader - System.Xml, 299
- XmlReader - System.Xml, 296
- XmlTextReader - System.Xml, 299

XmlValidatingReader
 System.Xml, 299
 ValidationType, 300
XmlWriter - System.Xml, 302
XPathDocument -
 System.Xml.XPath, 297
XPathNavigator
 Select(), 306
 System.Xml.XPath, 306
XPathNodeIterator
 Current, 306
 MoveNext(), 306
 System.Xml.XPath, 306

XSD
 complexType, 125
 element, 126
 Glosario, 504
XSL - Glosario, 504
XSLT
 for-each, 307
 Glosario, 504
 template, 307
 value-of, 307
XslTransform -
 System.Xml.Xsl, 298, 308