ADO.NET

Introducción

ADO .NET responde a las siglas de *Microsoft ActiveX Data Objects* de la plataforma .NET, y es una mejora evolutiva de la tecnología ADO. Realmente es una evolución más en las tecnologías de acceso a la información.

En este capítulo se comparan las tecnologías ADO y ADO.NET, se describen y detallan los objetos más importantes de la tecnología, el objeto `DataSet` y los proveedores de acceso a datos. Se profundizará sobre el objeto `DataSet` describiendo los objetos `DataTable`, `DataColumn`, etc. En cuanto a los proveedores de acceso a datos se describirá el proveedor para SQL Server y el proveedor para OLE DB.

Requisitos de la plataforma ADO.NET

Para utilizar los proveedores de datos de SQL Server .NET y OLE DB .NET se necesita la instalación de la versión 2.6 o superior de *Microsoft Data Access Components*. Para utilizar ADO.NET en las aplicaciones se ha de incluir el `namespace System.Data`.

Desde DAO hasta ADO

Hace varios años nació un API (*Application Programming Interface*) llamado ODBC (*Open Database Connectivity*) diseñado para proporcionar acceso a un amplio rango de orígenes de datos. Un origen de datos consiste básicamente en los datos asociados a un sistema de gestión de bases de datos, la plataforma en la que este sistema existe y la red usada para acceder a dicha plataforma.

Poco tiempo después apareció la primera interfaz orientado a objetos llamado DAO (*Data Access Objects*) que utilizaba el motor de bases de datos JET de Microsoft -utilizado por MS-Access- y que permitía a los programadores de Visual Basic conectar directamente a tablas de Access y a otras bases de datos a través de ODBC. Esta interfaz proporcionaba a los programadores una forma muy sencilla de trabajar con la información almacenada en las diferentes bases de datos. Es entonces cuando aparecen los objetos `Workspace`, `Database` y el famoso `Recordset` que los desarrolladores utilizaban para crear aplicaciones cliente-servidor. DAO era muy eficiente para bases de datos en local, o pequeños desarrollos, pero para aplicaciones distribuidas las prestaciones disminuían conforme se aumentaba el número de usuarios o la complejidad de las aplicaciones.

Por este motivo, apareció posteriormente otra tecnología llamada RDO (*Remote Data Objects*) que es una interfaz de acceso a datos orientada a objetos para ODBC. También es fácil de usar como DAO, pero es limitado ya que no funciona muy bien con el Jet o ISAM, y sólo puede acceder a bases de datos relacionales a través de drivers ODBC existentes. Pero por otro lado, permite a los desarrolladores acceder a los procedimientos almacenados y a consultas más complejas. Además es mucho más eficaz para aplicaciones complejas y con múltiples accesos simultáneos. Los objetos más utilizados en esta tecnología son `Enviroment`, `Connection` y `Resultset`.

Durante un tiempo convivían ambas tecnologías: DAO para aplicaciones sencillas o locales y RDO para aplicaciones distribuidas. Uno de los problemas que tenía la tecnología RDO es que el lenguaje SQL con el que se accede a la información debe de ser el que entienda el sistema de gestión de bases de datos correspondiente. Por ejemplo, si se desarrolla contra SQL Server, se debe utilizar sentencias T-SQL -lenguaje SQL propio para SQL Server- y si se programa contra Oracle se debe utilizar sentencias PL/SQL -lenguaje SQL propio para Oracle-, y aunque T-SQL y PL/SQL se basan en SQL estándar, tienen pequeñas diferencias que pueden hacer que una aplicación cliente-servidor que utilice RDO no funcione en otro sistema de gestión de bases de datos.

Con DAO no existía este problema y debido a este hecho, a otras circunstancias y sobre todo a la aparición de COM, el estándar ODBC evolucionó a OLE DB. OLE DB es un conjunto de interfaces basadas en COM que exponen los datos desde una gran variedad de orígenes de datos. Posteriormente surgió la tecnología ADO (*ActiveX Data Objects*) que se apoya en OLE DB y que es una interfaz que recoge lo mejor de las dos tecnologías anteriores, incorporando además otras mejoras.

Los objetos más importantes de la tecnología ADO son el objeto `RecordSet`, el objeto `Command` y el objeto `Connection`.

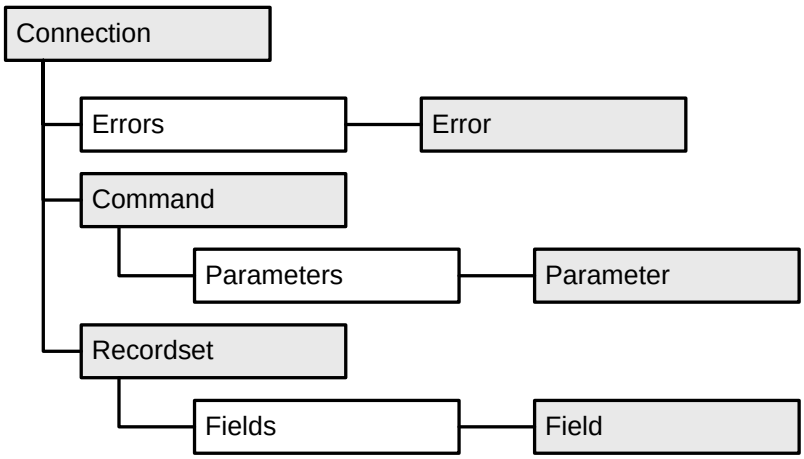

Figura 24.1. Diagrama de Objetos de ADO

ADO.NET frente a ADO

En esta sección se describen ambas tecnologías haciendo hincapié en las diferencias existentes.

ADO

La tecnología ADO es una capa COM sobre OLE DB de tal manera que las aplicaciones que utilicen esta tecnología de acceso a datos invocan a los objetos de ADO sin tener que conocer en absoluto el estándar OLE DB. Es una tecnología que se utiliza de manera sencilla desde Visual Basic o desde ASP, pero no está pensada para Visual C++ y otros lenguajes.

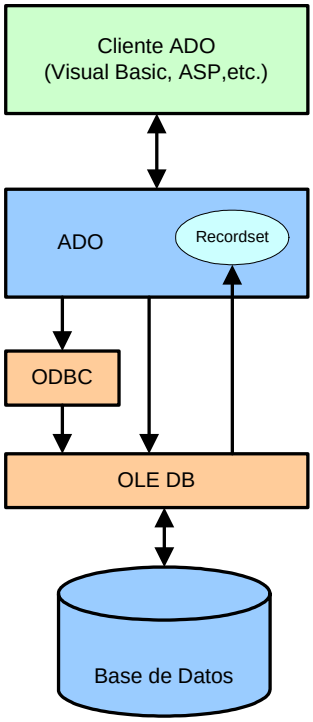

Figura 24.2. Arquitectura ADO.

El objeto clave dentro de ADO es el `Recordset` que tiene como principales ventajas su potencia y sencillez. Con este objeto se pueden utilizar cursores de cliente y cursores de servidor. Por otro lado, aunque puede trabajar desconectado de la fuente de datos, no está pensado para trabajar de ese modo, con lo cual se consiguen implementaciones menos eficaces de lo que cabía esperar. ADO está pensado para trabajar en aplicaciones tipo cliente-servidor en los que el `Recordset` está conectado a una fuente de datos y no para arquitecturas con varias capas en las que las prestaciones del `Recordset` disminuyen.

ADO.NET

La tecnología ADO.NET -en la versión beta 1 se llamaba ADO+- es un modelo de proveedor más sencillo que el par OLEDB / ADO y se integra perfectamente con XML. ADO.NET es la evolución de ADO en la nueva plataforma .NET. Tiene la misma filosofía pero se ha modificado el modelo.

Las principales características de ADO.NET son:

- Trabaja desconectado del origen de datos que se utilice.
- Tiene una fuerte integración con XML y ASP .NET.
- El uso de ADO.NET es independiente del lenguaje de programación que se utilice.

Por otra parte se ha demostrado que los niveles de trasferencia de información con ADO.NET que utiliza XML, son tan buenos como los niveles que alcanza ADO utilizando COM.

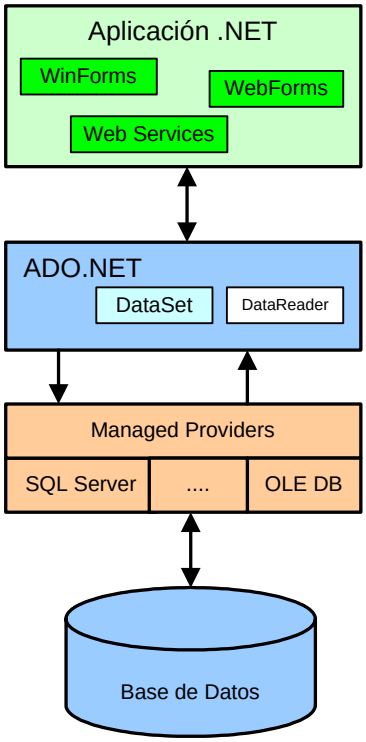

Figura 24.3. Arquitectura de ADO.NET

La tecnología ADO .NET esta basada en un nuevo modelo de componentes en la que las clases de acceso a datos y las clases contenedores forman parte del marco de trabajo de .NET. Sobre todo ADO.NET está pensado para "interoperar" con otros componentes, sistemas, etc. gracias al uso de XML y a soportar estándares como HTTP, XML o SOAP.

El modelo de ADO.NET está dividido en dos grupos:

- Los *proveedores de datos* o *Managed Data Providers*.
- Los *contenedores de datos*, que aunque están vinculados a los orígenes de datos, son independientes de ellos.

En la siguiente tabla se comparan ambas tecnologías con más detalle.

Característica	**ADO**	**ADO.NET**
Representación de datos residente en memoria.	Utiliza el objeto `RecordSet`, cuyo aspecto es como una simple tabla.	Utiliza el objeto `DataSet`, que puede contener una o más tablas representadas por los objetos `DataTable`.
Relaciones entre varias tablas.	Requiere la unión de varias tablas para mostrar finalmente una simple tabla respuesta	Soporta el objeto `DataRelation` para asociar filas en un objeto `DataTable` con filas de otro objeto `DataTable`.
Recorrido de los	La navegación por las filas	Utiliza una forma no secuencial

datos.	del `RecordSet` se realiza de forma secuencial.	de navegación para acceder a las filas de una tabla. Utiliza las relaciones para navegar desde filas de una tabla a las correspondientes filas de otra tabla.
Acceso desconectado	El objeto `RecordSet` posee esta característica pero el uso habitual es mediante accesos conectados, representados por el objeto `Connection`. La comunicación con la base de datos se realiza mediante llamadas al proveedor de datos OLE DB.	Se comunica con la base de datos mediante llamadas estándares al objeto `DataAdapter`, el cual se comunica con el proveedor de datos OLE DB, o directamente a SQL Server.
Cursores	Utiliza tanto cursores de servidor como cursores del lado cliente.	Como la arquitectura es desconectada los cursores no son aplicables.
"Programabilidad"	Utiliza el objeto `Connection` para trasmitir los comandos que tratan la estructura de datos que subyace de una fuente de datos	Usa XML. Los datos se describen a sí mismos porque los nombres de las etiquetas del código corresponden a problemas del "mundo real" solucionados por el código. Las estructuras de datos como tablas, filas y columnas no aparecen haciendo que el código sea más fácil de leer y escribir.
Compartir datos desconectados entre capas y componentes.	Utiliza COM `marshalling` para trasmitir un `Recordset` desconectado. Soporta solamente tipos de datos definidos por el estándar COM. Requiere conversiones de tipo que necesitan recursos del sistema.	Trasmite un `DataSet` mediante XML y este formato no tiene restricciones en los tipos de datos y no se requiere conversiones de tipo.
Trasmitir datos a través de Firewalls.	Problemático, porque los firewalls suelen estar configurados para prevenir peticiones a nivel de sistema, como por ejemplo COM `marshalling`.	Soportado, porque los objetos `DataSet` de ADO.NET utilizan XML que puede pasar a través de un Firewall.
Escalabilidad	Los bloqueos de la base de datos y las conexiones activas de la base de datos para las duraciones largas generan un	El acceso desconectado a los datos de la base de datos sin los bloqueos de retención de la base de datos o las conexiones

	problema en caso de recursos limitados de la base de datos	activas de la base de datos por períodos muy largos limita el problema de los recursos limitados de la base de datos

En resumen: el problema no está en elegir entre ADO.NET y ADO. El problema radica en la elección de .NET como plataforma de desarrollo. Si es así, entonces ADO.NET es la elección correcta.

Namespaces

ADO .NET se basa en los siguientes espacios de nombres de accesos a datos:

- `System.Data`, que proporciona las clases de acceso a datos generales.
- `System.Data.Common`, que contiene las clases compartidas por los proveedores de datos.
- `System.Data.OleDb`, que almacena las clases del proveedor de datos OLE DB.
- `System.Data.SqlClient`, que expone las clases del proveedor de datos para SQL Server.

¿Que es un "Managed Provider"?

El *Managed Provider* o .NET *Managed Data Provider* o simplemente .NET *Data Provider* es el proveedor de datos de la plataforma .NET que permite conectar a un origen de datos con una aplicación para recuperar y modificar información. También este proveedor sirve de puente entre el origen de datos y el objeto más importante de ADO.NET, el `DataSet` que se verá más adelante.

Los proveedores de datos en la plataforma .NET ofrecen una arquitectura de acceso a datos más simple, con características mejoradas y exponen directamente su comportamiento específico a los consumidores. Además tienen un conjunto de interfaces mucho más pequeño que los proveedores OLE DB, y no requieren de la tecnología COM.

Los *Managed Providers* acceden al objeto `DataSet` a través de la implementación del interface `IDataAdapter` (ver figura 24.4). Por otro lado acceden al origen de datos de forma conectada a través de objetos `Connection`, `Command` o `Parameter`, y proporcionan a su vez acceso a datos en forma desconectada. Los resultados obtenidos del origen de datos, pueden ser procesados directamente a través del objeto `DataReader` correspondiente, o depositados en un objeto `DataSet` para su posterior uso.

Modelo Común

ADO.NET presenta un modelo común para los objetos .NET *Data Provider* de tal forma que se pueda codificar independientemente del proveedor de datos .NET elegido.

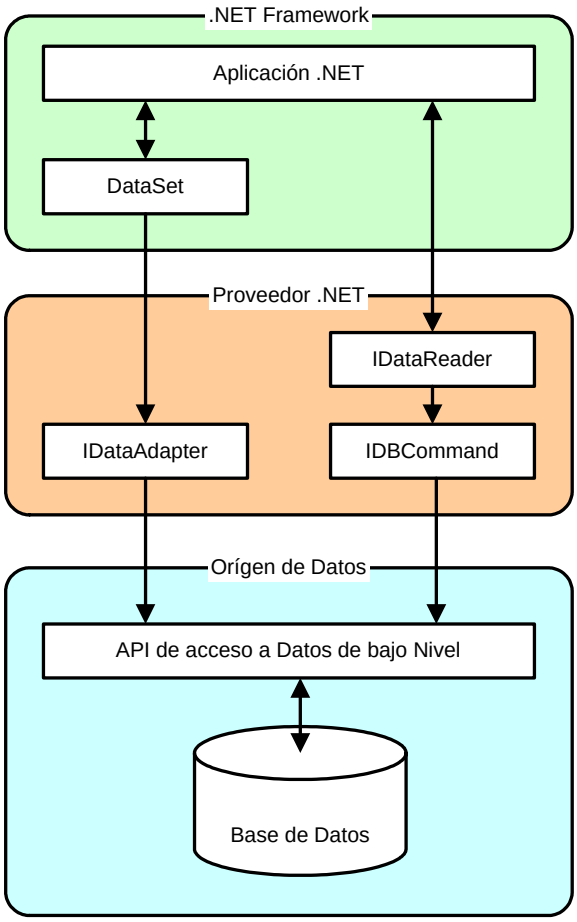

Figura 24.4. Arquitectura de los "`Managed Providers`"

Las interfaces generales, que independientemente del proveedor, se pueden utilizar con los proveedores de datos en la plataforma .NET son:

`IDBConnection`	Representa la conexión en una única sesión con un origen de datos determinado
`IDBTransaction`	Representa una transacción en local.
`IDBCommand`	Representa un comando que se ejecuta cuando está conectado a un origen de datos.
`IDataParameter`	Permite implementar un parámetro en un comando.
`IDataReader`	Lee un conjunto de datos de solo lectura y "forward-only" de un origen de datos
`IDataAdapter` *	Se encarga de rellenar y de resolver los conflictos del DataSet con el origen de datos.
`IDBDataAdapter`	Suministra los métodos de ejecución típicos para operar con bases de datos (insert, update, select y delete)

** En cualquier caso sólo el IDataAdapter es necesario.*

El siguiente ejemplo trabaja utilizando el mismo código con ambos proveedores: SQL Server .NET *Data Provider* y OLE DB .NET *Data Provider*, es decir, funciona independientemente de que el objeto `Connection` (`ObjCnn`) represente una conexión a un SQL Server .NET *Data Provider* o a un OLE DB .NET *Data Provider*.

```
IDbCommand objCmd = objCnn.CreateCommand();
objCmd.CommandText = "SELECT * FROM Employees";
IDataReader objDR = objCmd.ExecuteReader();

while objDR.Read()
  Console.WriteLine("{0}\t{1}", objDR.GetString(0), objDR.GetString(1));
```

.NET Data Providers

ADO.NET puede acceder a la información de la base de datos solamente a través de los servicios de los *managed providers*. Se recomienda utilizar los *managed providers* nativos siempre que se pueda. ADO.NET cuenta con los siguientes proveedores para los diferentes orígenes de datos:

Managed Provider para SQL Server 7.0 y SQL Server 2000

Managed Provider para proveedores OLE DB

Para utilizar los proveedores de acceso a datos de .NET debe utilizarse el `namespace` correspondiente:

`System.Data.SqlClient` para SQL Server 7.0 o SQL Server 2000

`System.Data.OleDb` para proveedores OLE DB

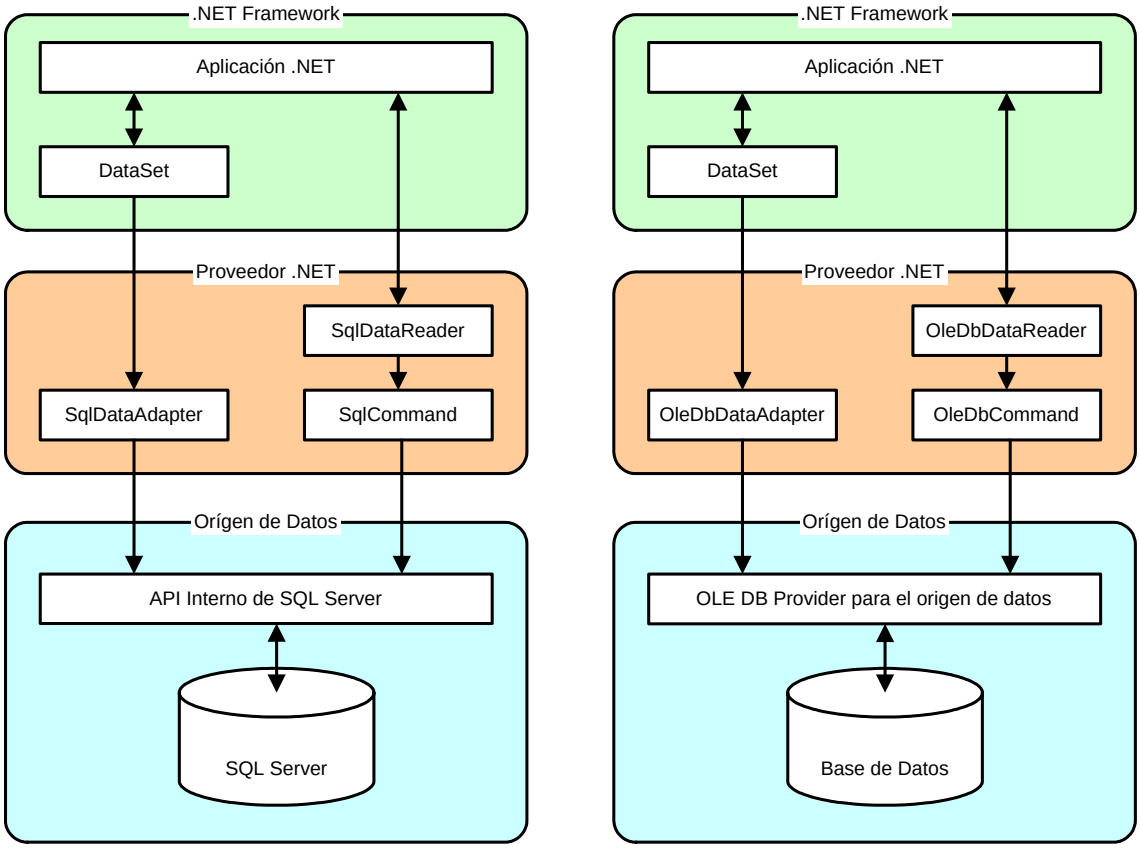

Figura 24.5. *Managed Providers* para SQL Server y para OLE DB

SQL Server .NET Data Provider

Este proveedor de acceso a datos utiliza su propio protocolo para comunicarse con SQL Server. Accede a los datos sin necesidad de añadir una capa OLE DB o una capa ODBC.

Se recomienda utilizar este proveedor tanto para aplicaciones cliente-servidor como para aplicaciones con varias capas frente a SQL Server 7.0 o superior o frente a MSDE (*Microsoft Data Engine*). Aunque se puede usar el proveedor de datos OLE DB .NET para SQL Server para acceder al origen de datos, no se recomienda su utilización ya que el proveedor SQL Server .NET utiliza el API interno de SQL Server, haciendo mucho más eficaz al proveedor. Para el caso de versiones inferiores a la 7.0 es necesario utilizar el proveedor de datos OLE DB .NET que requiere la instalación del MDAC 2.6 o posterior.

OLE DB .NET Data Provider

El OLE DB .NET Data Provider utiliza el OLE DB nativo a través de COM para acceder a los datos soportando transacciones manuales y automáticas.

Para utilizar el proveedor de datos OLE DB .NET es necesario utilizar también el proveedor OLE DB (2.6 o superior). Los siguientes proveedores son compatibles con ADO.NET.

Driver	***Proveedor***
SQLOLEDB	Microsoft OLE DB Provider para SQL Server.
MSDAORA	Microsoft OLE DB Provider para Oracle.

Microsoft.Jet.OLEDB.4.0	OLE DB Provider para Microsoft Jet.

El proveedor de datos OLE DB .NET no soporta OLE DB 2.5 con lo que no funcionarán los proveedores siguientes: Microsoft OLE DB Provider para Exchange y el Microsoft OLE DB Provider para Internet Publishing. Tampoco funciona con el proveedor de datos OLE DB para ODBC (MSDASQL).

Se recomienda este proveedor para aplicaciones de varias capas que utilicen Oracle o SQL Server 6.5 o inferior. Para aplicaciones cliente-servidor es más conveniente la utilización de Microsoft Access como origen de datos pero para aplicaciones de varias capas Microsoft no recomienda utilizar Microsoft Access.

Los componentes de ADO.NET

Los componentes de ADO.NET han sido diseñados para acceder y manipular datos. Hay dos componentes principales en ADO.NET que son `DataSet` y los proveedores de datos .NET que a su vez incluyen los objetos `Connection`, `Command`, `DataReader` y `DataAdapter`. En este apartado se describirá detalladamente cada uno de estos objetos.

La clase Connection

Es la clase encargada de establecer la conexión con el origen de datos y tiene soporte automático para *pooling* de conexiones. Dependiendo del origen de datos que se utilice se deberá utilizar `OleDbConnection` o `SqlConnection`. Se entiende por *pool* de conexiones el conjunto de conexiones "cacheadas" en el servidor.

Esta clase implementa la interface `IDbConnection`. La cadena de conexión al origen de datos se obtiene por medio de la propiedad `ConnectionString`, el estado de la conexión por medio de la propiedad `State`, para la base de datos se utiliza `Database` y para determinar el tiempo máximo de conexión la propiedad `ConnectionTimeout`.

Los métodos más importantes de esta clase son:

`Open`	Cuando se llama al método `Open`, se abre un canal físico con el origen de datos
`Close`	Cierra la Conexión, pero ésta no se destruye facilitando el *pooling* de conexiones. El consumo en memoria es bajo.
`BeginTransaction`	Comienza la transacción
`ChangeDatabase`	Cambia de base de datos
`CreateCommand`	Crea un objeto Command

La siguiente tabla describe los estados en los que se puede encontrar la conexión.

`Open`	La conexión está abierta y funcionando

`Broken`	Una conexión previa ha dejado de funcionar. Debe ser cerrada y reabierta.
`Closed`	Está cerrada.
`Connecting`	Conectándose, la conexión está siendo abierta.
`Executing`	Ejecutando un comando
`Fetching`	Recogiendo la información del origen de datos.

El proveedor sólo puede cambiar la cadena de conexión cuando está cerrada (`Closed`). Por otro lado, los desarrolladores deberían utilizar el método `Close` cuando hayan acabado de utilizar la conexión, ya que este método cierra la conexión y la devuelve al *pool* de conexiones. Sin embargo el método `Dispose` cierra la conexión y destruye la instancia del objeto eliminándola del *pool* de conexiones.

En función del proveedor de datos de la plataforma .NET que se utilice, se puede elegir entre los clases `Connection` siguientes: `OleDbConnection` y `SqlConnection`.

Clase OleDbConnection

Esta clase representa una conexión al proveedor de datos OLE DB .NET. Las propiedades más importantes que utiliza son:

- `ConnectionString` para especificar el origen de datos
- `ChangeDatabase` para cambiar la base de datos para conexiones abiertas.
- `ConnectionTimeout` para obtener el tiempo de espera. Por defecto son 15 segundos. Si se desea que sea ilimitado ha de asignársele el valor cero y sólo podrá cambiarse su valor a través de la propiedad `ConnectionString`.
- Si el estado de la conexión cambia -propiedad `State`- entonces se produce el evento `StateChange`.

Clase SqlConnection

Esta clase representa una conexión a la base de datos SQL Server 7.0 o superior. Las propiedades más importantes son las siguientes:

- Si se produce una excepción en SQL (`SqlException`) mientras se ejecuta un comando concreto, la conexión permanece abierta en caso de que el nivel de severidad sea menor que 19. Con un nivel de severidad mayor que 20, la conexión se cierra y es necesario reabrirla para continuar.
- La propiedad `ConnectionString` especifica el origen de datos.
- Un objeto `SqlConnection` lee la información de la conexión a través de la base de datos y del origen de datos. No necesita leer información del proveedor como en el caso de OLE DB .NET.

- La propiedad `ConnectionTimeout` obtiene el tiempo de espera que por defecto son 15 segundos. Si se desea que sea ilimitado ha de asignársele el valor cero y solo podrá cambiarse su valor a través de la propiedad `ConnectionString`.

- Cuando se utiliza el método `Close` si existen transacciones pendientes en ese momento, las deshace (`RollBack`).

Creación de una conexión con Visual Studio .NET

En Visual Studio .NET existe la posibilidad de realizar una conexión de forma muy sencilla utilizando el **Explorador de servidores**.

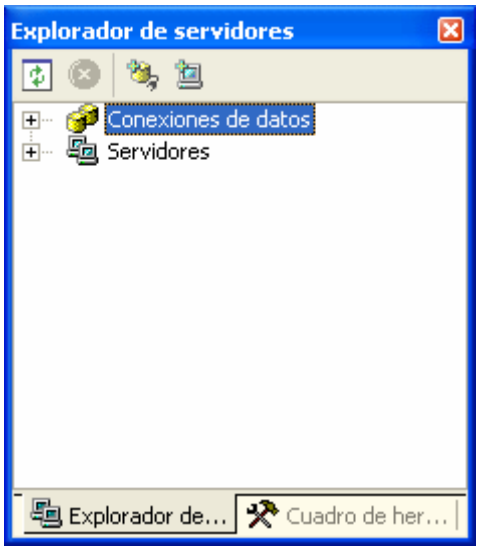

Figura 24.6. Explorador de conexiones en Visual Studio .NET

Para crear una conexión, basta con seleccionar con el botón derecho **Agregar conexión** de la figura 24.6. Se accede a configurar la conexión dependiendo del proveedor que se utilice. A continuación, se realizan dos ejemplos de conexión.

A) Para el caso de una conexión a SQL Server

Para el caso de una conexión a la base de datos Northwind en SQL Server se elige el proveedor para SQL Server, el servidor, la base de datos y el usuario junto con la contraseña (Figura 24.7).

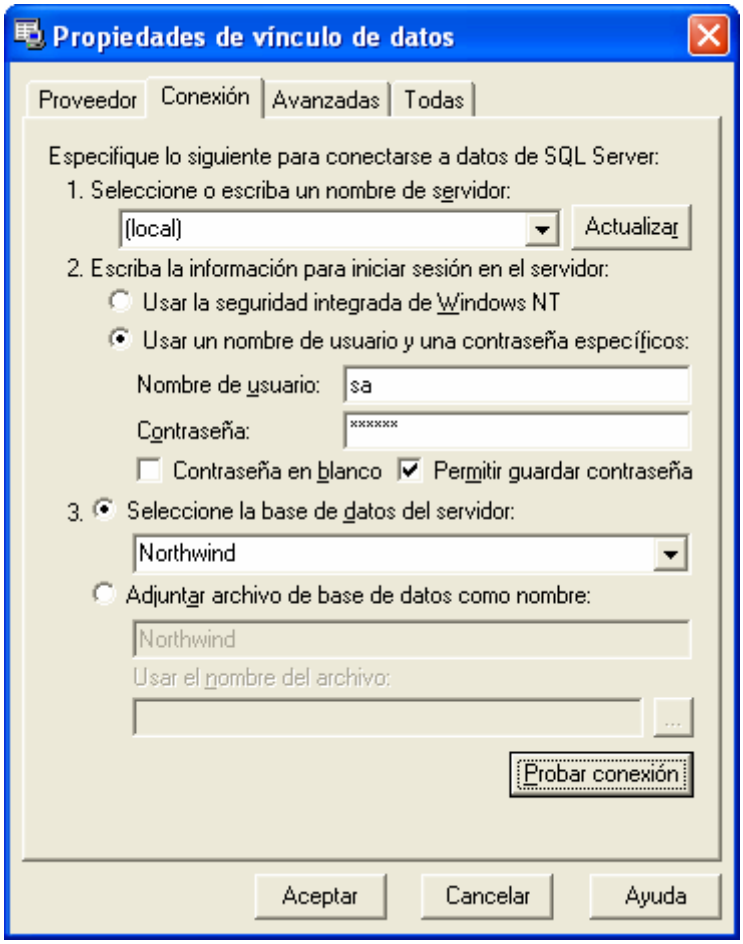

Figura 24.7. Propiedades de la conexión a SQL Server.

Una vez rellenados los datos, es posible observar todos los elementos de la base de datos.

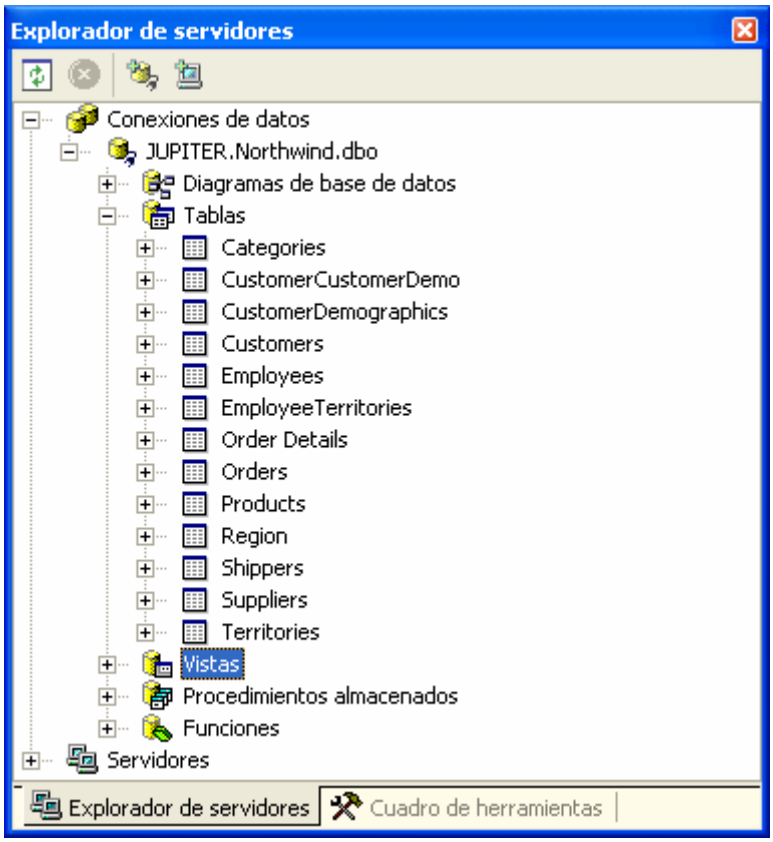

Figura 24.8. Conexión a la base de datos Northwind en SQL Server.

Una vez que la conexión esté creada basta con seleccionarla y arrastrarla hasta el formulario donde va a ser utilizada.

B) Para el caso de una conexión a una base de datos Access.

Se selecciona el proveedor de acceso a datos Jet 4.0 OLE DB Provider y se elige la base de datos Neptuno de Access 2000.

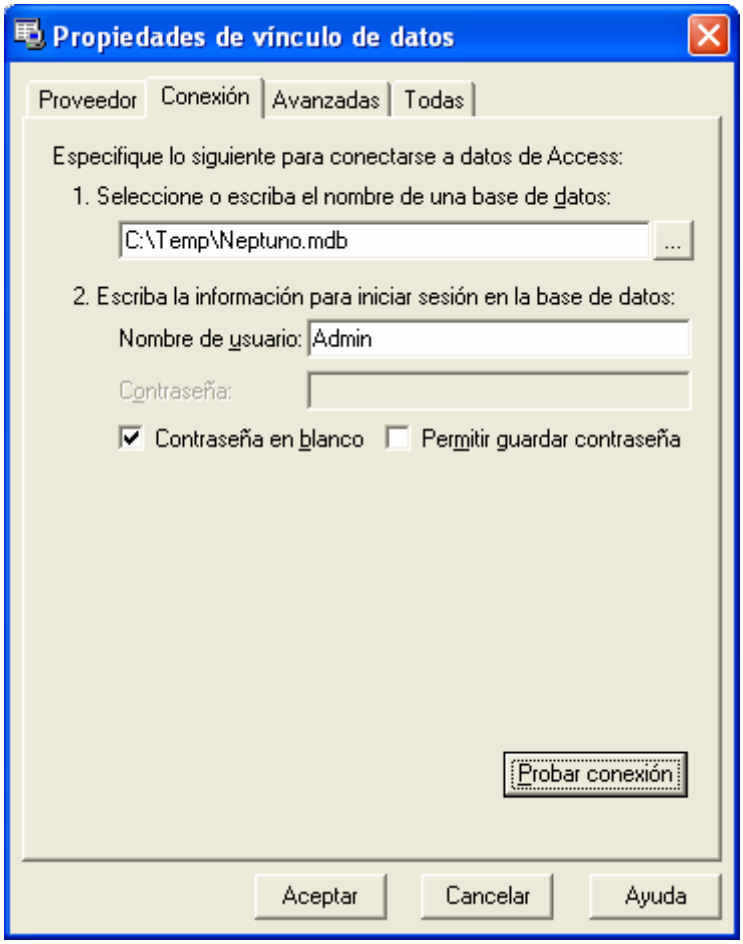

Figura 24.9. Propiedades de la conexión a una base de datos de Access

Después de crear la conexión se puede acceder a los diferentes elementos de la base de datos, que en este caso se dividen en tablas, vistas y procedimientos almacenados. Las vistas en Access son también denominadas "consultas de selección", y los procedimientos almacenados, bien pueden ser consultas con parámetros, o consultas de unión, o consultas de referencias cruzadas o consultas de acción -`Insert`, `Update`, `Create` o `Delete`- disponibles en Access.

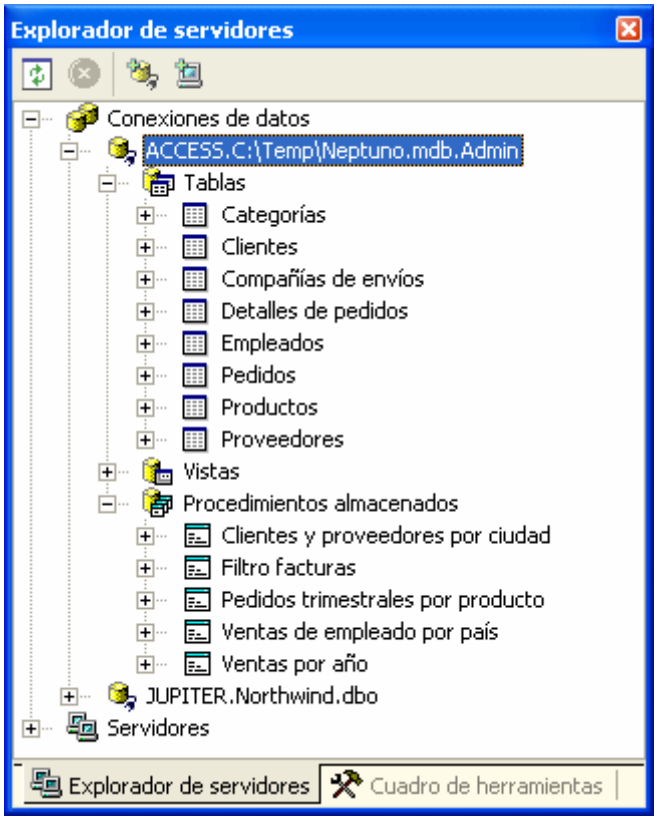

Figura 24.10. Elementos de la base de datos Neptuno

Transacciones en ADO.NET

Para empezar una transacción en ADO .NET se llama al método `BeginTransaction` que acepta el parámetro `IsolationLevel` (nivel de aislamiento) y/o el nombre, y devuelve un objeto transacción de la clase `OleDbTransaction` o `SqlTransaction` dependiendo del proveedor que se utilice.

El objeto `SqlTransaction` soporta `savepoints` o puntos de almacenamiento de la transacción que permiten deshacerla (`RollBack`) más tarde. Para almacenar un `savepoint` se utiliza el método `Save`. Esta funcionalidad es equivalente a la declaración "SAVE TRANSACTION" que se realiza en T-SQL.

Si se quiere deshacer una transacción ya almacenada, ha de utilizarse el método `Rollback` que puede llevar como parámetro el nombre del punto de almacenamiento previamente guardado. En caso de querer aceptar la transacción se utilizará el método `Commit`.

A continuación se realiza un ejemplo con SQL Server:

```
String strConexion = "SERVER=(local); uid=sa; pwd=sa; database=Northwind";
SqlConnection ObjCnn = new SqlConnection(strConexion);

ObjCnn.Open();
// Comienza la transacción
SqlTransaction ObjTran = ObjCnn.BeginTransaction();
```

```
try
{
      // Ejecución de comandos

      // Se ejecuta el Commit de la Transacción
      ObjTran.Commit();
}
catch (Exception ex)
{
      // La transacción ha fallado
      ObjTran.Rollback();
}
ObjCnn.Close();
```

A continuación se realiza un ejemplo con OLE DB.

```
String strConexion = ("Provider = Microsoft.Jet.OLEDB.4.0;Data Source = Neptuno.MDB";
OleDbConnection ObjCnnOLE= new OleDbConnection(strConexion);

ObjCnn.Open();
// Comienza la transacción
OleDbTransaction ObjTran = ObjCnnOLE.BeginTransaction();
try
{
      // Ejecución de comandos

      // Se ejecuta el Commit de la Transacción
      ObjTran.Commit();
}
catch (Exception ex)
{
      // La transacción ha fallado
      ObjTran.Rollback();
}
ObjCnnOLE.Close();
```

La clase Command

Un objeto de la clase `Command` al igual que un objeto de la clase `Connection` pertenece al proveedor de acceso a datos de la plataforma .NET, así que a la hora de codificar se puede utilizar la interfaz `ICommand` común a todos los proveedores (todos ellos la implementan), o las clases específicas de alguno de los proveedores que proporciona la plataforma .NET.

Un objeto `Command` básicamente formula una petición y se la envía al origen de datos. Si esa petición devuelve datos, el objeto `Command` se encarga de empaquetarlos y devolverlos como un objeto `DataReader`, un valor escalar o como los parámetros de salida utilizados en el propio comando.

Existen dos propiedades muy importantes de la clase `Command`:

- `CommandText`. Representa una cadena o texto cuya sintaxis es comprendida por el proveedor de acceso a datos .NET que se utilice.

- `CommandType`. Esta propiedad indica la forma en la que la propiedad `CommandText` va a ser interpretada y su valor por defecto es `CommandType.Text`. Los valores que puede tomar esta propiedad son:

    - `CommandType.Text`. Indica que la propiedad `CommandText` es un texto plano.

      Por ejemplo:

      ```
      String strSQL = ”SELECT * FROM Employees”;
      SqlCommand ObjCmd = new SqlCommand(strSQL, ObjCnn);
      // La línea siguiente no es necesaria porque Text es el
      // valor por defecto
      ObjCmd.CommandType = CommandType.Text;
      ```

    - `CommandType.StoredProcedure`. Representa que la propiedad `CommandText` es el nombre de un procedimiento almacenado.

      Por ejemplo:

      ```
      String strSP = ”CustOrdersDetail”;
      SqlCommand ObjCmd = new SqlCommand(strSP, ObjCnn);
      ObjCmd.ComamndType = CommandType.StoredProcedure;
      ```

    - `CommandType.TableDirect`. Con esta propiedad se indica que el `CommandText` es el nombre de una tabla del origen de datos. Esta opción no está soportada en el SQL Server .NET `Data Provider`.

      Por ejemplo:

      ```
      String strTabla = ”Employees”;
      OleDbCommand ObjCmd = new OleDbCommand(strTabla, ObjCnn);
      ObjCmd.CommandType = CommandType.TableDirect;
      ```

Cuando se está utilizando un objeto `Command` se necesita que un objeto `Connection` haya sido abierto. Siempre hay una transacción asociada a la conexión. Si se “resetea” la conexión entonces el objeto transacción devuelve `null`. En principio, se puede cambiar la conexión sin problema alguno para conectar con otro usuario o para conectar a otra fuente de datos, pero hay que tener en cuenta que la transacción que subyace sea compatible con la conexión.

Los parámetros del objeto `Command` pertenecen a una colección llamada `OleDbParameterCollection` o `SqlParameterCollection` dependiendo del proveedor seleccionado. Estas colecciones están compuestas por objetos `OleDbParameter` o `SqlParameter` respectivamente.

Para crear un nuevo parámetro existen dos posibilidades, o bien utilizar el operador `New` de la clase `Parameter`, o bien trabajar con el método `CreateParameter` de la clase `Command`.

Para ejecutar un comando se necesita que haya una conexión válida abierta y dependiendo del comportamiento que se desee dar al objeto `Command` se utilizará uno de los siguientes métodos:

- `ExecuteNonQuery`: No devuelve un conjunto de registros sino que devuelve el número de filas afectadas por la acción.
- `ExecuteReader`: Devuelve un conjunto de registros de sólo lectura y "forward-only". No se informa del número de filas afectadas.
- `ExecuteScalar`: Devuelve sólo el valor de la celda (0,0) del conjunto de registros.

El comportamiento que puede tener un objeto `Command` está recogido en la propiedad `CommandBehavior` que describe los resultados y la forma en la cual la consulta debería afectar al origen de datos. Se puede utilizar el método `ExecuteReader` para alterar el comportamiento del objeto `Command` mediante las siguientes opciones:

- `CloseConnection`. Cierra la conexión cuando se cierra el objeto `DataReader`
- `KeyInfo`. Muestra información de la clave primaria o *primary key* y de las columnas sin realizar bloqueo de las filas.
- `SchemaOnly`. Muestra información solo de las columnas y no bloquea las filas.
- `SequentialAccess`. El contenido de la columna puede leerse de golpe utilizando el método `GetBytes` del objeto `DataReader`.
- `SingleResult`. Devuelve un único conjunto de datos, en caso de existir varios, sólo devolverá el primero de ellos.
- `SingleRow`. Se utiliza cuando se espera que la consulta devuelva una única fila. Los proveedores de datos están optimizados con esta opción para ejecutar este tipo de consultas.

Como ya se ha descrito anteriormente la clase `Command` pertenece al proveedor de acceso a datos .NET, de tal forma que existen las clases `OleDbCommand` y `SqlCommand` para el proveedor OLE DB .NET y SQL Server .NET respectivamente.

Clase OleDbCommand

Esta clase representa una declaración OLE DB para ejecutar contra un origen de datos OLE DB. Esta declaración puede ser un comando SQL, el nombre de un procedimiento almacenado concreto, o bien cualquier texto que entienda el proveedor OLE DB.

Tiene diferentes constructores:

a) `Command Text`. Ejecuta una sentencia comprensible por el proveedor.

b) `Command Text + OleDbConnection`. Ejecuta una sentencia bajo una conexión subyacente.

c) `Command Text + OleDbConnection + OleDbTransaction`. Ejecuta una sentencia en una transacción bajo una conexión subyacente.

Antes de ejecutar el comando se asegura que la conexión está abierta.

Existen otras propiedades como `CommandType` y `CommandTimeout` que por defecto vale 30 segundos.

Al ejecutar el comando `OleDbCommand` se pueden utilizar los siguientes métodos:

- `ExecuteReader`, que devuelve un objeto `OleDbDataReader` después de ejecutar una sentencia SQL de selección SELECT.

  Por ejemplo:

```
String strSQL = ”SELECT * FROM Employees”;
OleDbCommand ObjCmd = new OleDbCommand(strSQL, ObjCnnOLE);
OleDbDataReader ObjReader = ObjCmd.ExecuteReader();
while (ObjReader.Read())
{
      Console.Write(ObjReader[0].ToString());
}
```

- `ExecuteNonQuery`, se utiliza para ejecutar consultas de acción mediante cláusulas como `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `SET` principalmente. Esta sentencia devuelve un entero indicando el número de filas afectadas por la acción. Lógicamente modifica el valor de la propiedad `RecordAffected`.

  Por ejemplo:

```
String strSQL = ”UPDATE Employees SET lastname=’Isabel’ WHERE
EmployeeID=2”;
OleDbCommand ObjCmd = new OleDbCommand(strSQL, ObjCnnOLE);
ObjCmd.ExecuteNonQuery();
```

- `ExecuteScalar`. Devuelve el único valor de la primera fila y primera columna del resultado obtenido de la ejecución. Se utiliza mucho para obtener valores calculados en procedimientos almacenados.

  Por ejemplo:

```
String strSQL = ”SELECT COUNT(*) FROM Employees”;
OleDbCommand ObjCmd = new OleDbCommand(strSQL, ObjCnnOLE);
Object ObjAux = ObjCmd.ExecuteScalar();
```

Clase SqlCommand

Representa una sentencia T-SQL o un procedimiento almacenado y tiene los siguientes constructores:

a) `Command Text`. Ejecuta una sentencia comprensible por SQL Server

b) `Command Text + SqlConnection`. Ejecuta la sentencia bajo la conexión con SQL Server subyacente.

c) `Command Text + SqlConnection + SqlTransaction`. Ejecuta una sentencia en una transacción bajo una conexión SQL Server subyacente.

Al ejecutar el comando `SqlCommand` se pueden utilizar los siguientes métodos

- `ExecuteReader`, que devuelve un objeto `SqlDataReader` después de ejecutar una sentencia SQL de selección SELECT. Se pueden ejecutar procedimientos almacenados del sistema de SQL Server.

  Por ejemplo:

```
String strSQL = ”SELECT * FROM Employees”;
SqlCommand ObjCmd = new SqlCommand(strSQL, ObjCnn);
SqlDataReader ObjReader = ObjCmd.ExecuteReader();
while (ObjReader.Read())
{
      Console.Write(ObjReader[0].ToString());
}
```

- `ExecuteNonQuery`, se utiliza para ejecutar consultas de acción mediante cláusulas como `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `SET` principalmente. Esta sentencia devuelve un entero indicando el número de filas afectadas por la acción. Lógicamente modifica el valor de la propiedad `RecordAffected`.

  Por ejemplo:

```
String strSQL = ”UPDATE Employees SET lastname=’Alvaro’ WHERE EmployeeID=2”;
SqlCommand ObjCmd = new SqlCommand(strSQL, ObjCnn);
ObjCmd.ExecuteNonQuery();
```

- `ExecuteScalar`. Devuelve el único valor de la primera fila y primera columna del resultado obtenido de la ejecución. Se usa mucho para obtener valores calculados o cantidades totales.

  Por ejemplo:

```
String strSQL=”SELECT COUNT(*) FROM Employees”;
SqlCommand ObjCmd=new SqlCommand(strSQL, ObjCnn);
Object ObjAux = ObjCmd.ExecuteScalar();
```

- `ExecuteXmlReader`. Devuelve un objeto `XmlReader` después de ejecutar el comando SELECT que explota las características XML de SQL Server 2000.

```
String strSQL="SELECT * FROM Employees FOR XML AUTO";
SqlCommand ObjCmd=new SqlCommand(strSQL, ObjCnn);
XmlReader ObjXR = cmd.ExecuteXmlReader();
while (ObjXR.Read())
{
      Console.WriteLine(ObjXR.ReadOuterXml());
}
```

Creación de un objeto Command con Visual Studio .NET

A continuación se realizarán dos ejemplos. El primero utiliza la clase `SqlCommand` en SQL Server y en el segundo se trabaja con la clase `OleDbCommand` para Access.

A) Creación de un objeto SqlCommand para SQL Server

Supónga que se dispone de un formulario al que se ha añadido un objeto `SqlConnection` llamado `sqlConnection1`. El primer paso es añadir un objeto `sqlCommand` al formulario (pestaña *Data* del **cuadro de herramientas**) .

Figura 24.11. Cuadro de herramientas de la pestaña `Data`

A continuación, en la ventana de propiedades, en la propiedad `Connection` se ha de seleccionar el objeto `sqlConnection` deseado -en este caso `sqlConnection1`-. El último paso es indicar la expresión SQL válida. Si por ejemplo se desea que el comando utilice una consulta sobre los pedidos de la base de datos Northwind, se puede utilizar la propiedad `CommandText` que dispone de un asistente para el diseño de consultas similar al que proporciona SQL Server

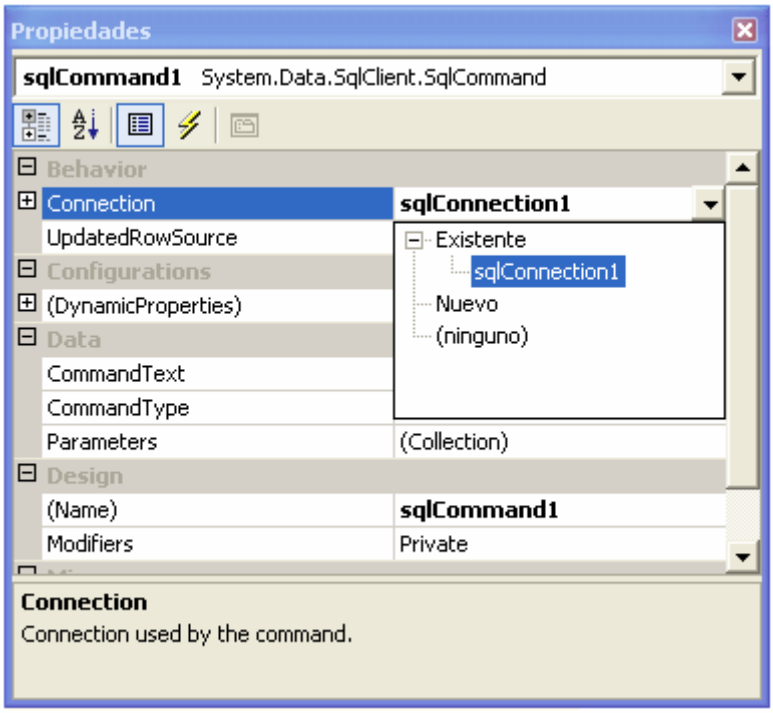

Figura 24.12. Asignación de la conexión al control `sqlCommand`

.

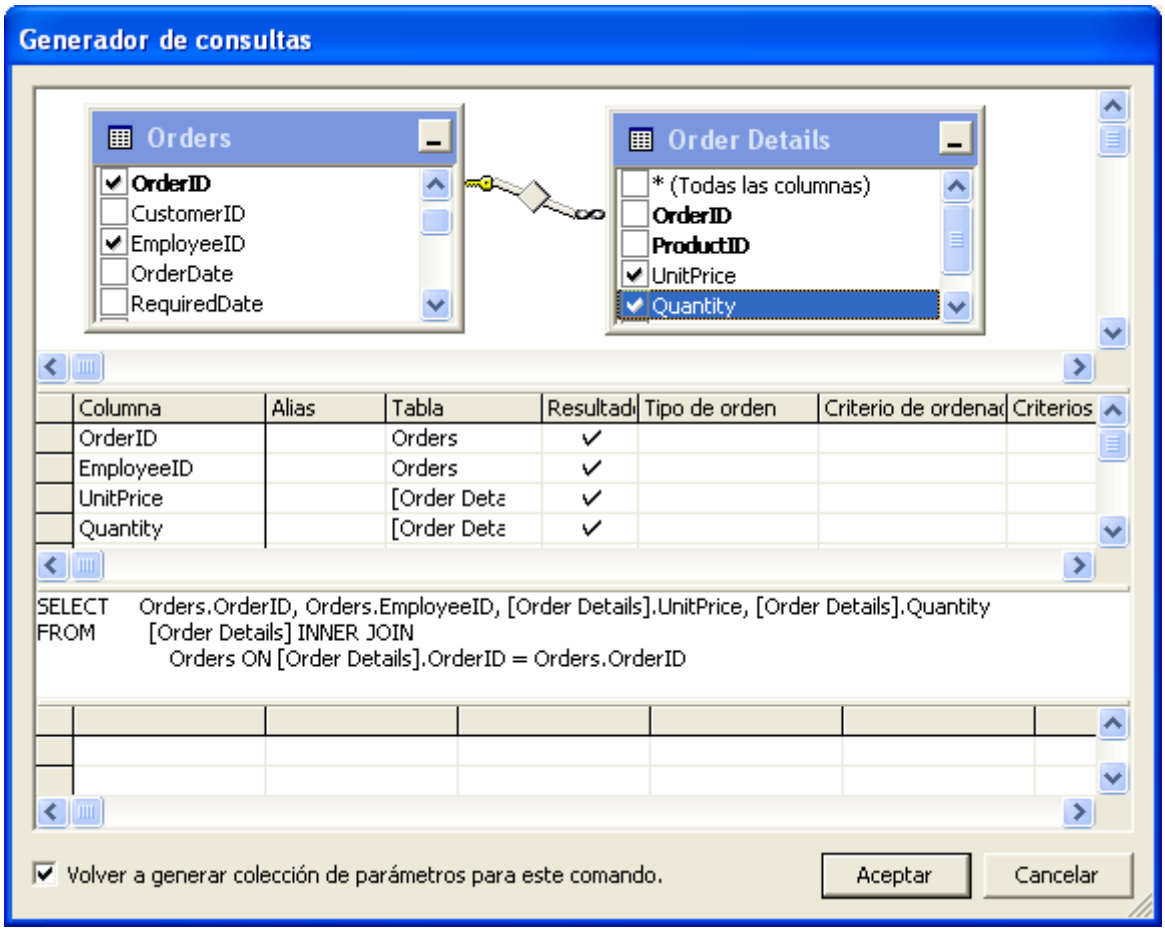

Figura 24.13. Generador de consultas de Visual Studio .NET

y una vez construida la consulta, la sentencia SQL generada es asignada a la propiedad `CommandText` del componente `SqlCommand`.

B) Creación de un objeto OleDbCommand para Access

Se parte de que existe una conexión a la base de datos Access llamada Neptuno. Posteriormente se arrastra un objeto `OleDbCommand` del cuadro de herramientas al formulario, y se asigna la conexión al comando mediante la propiedad `Connection`. Si se accede a la propiedad `CommandText` se establece la expresión del comando por medio del generador de consultas, como se indica en la figura 24.13.

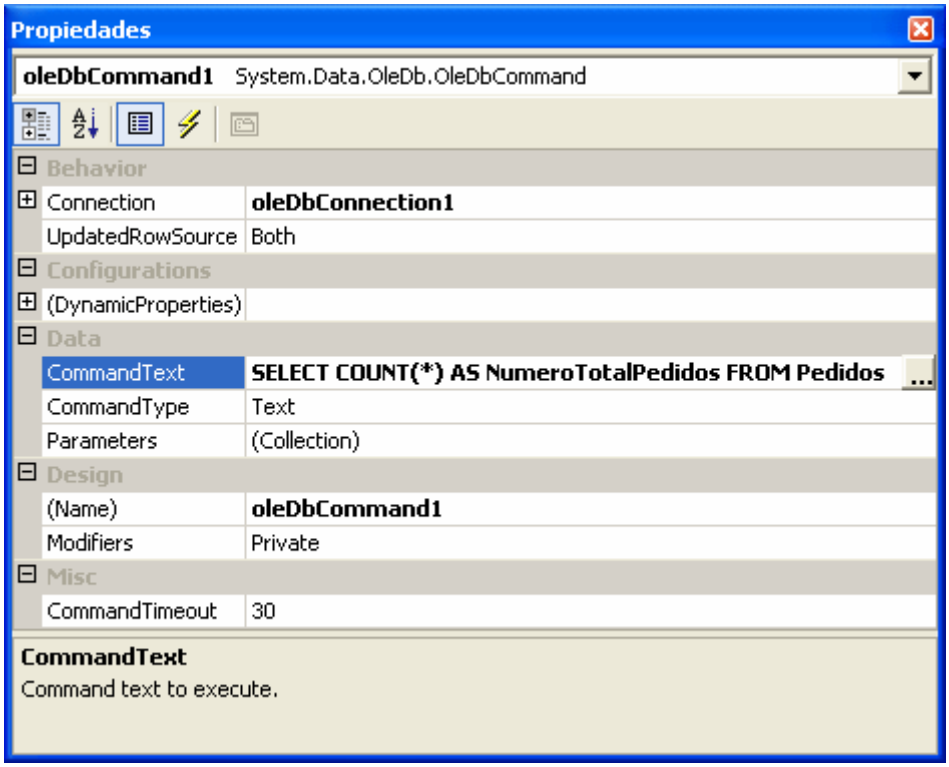

Figura 24.14. Propiedades del objeto `OleDbCommand`

Llamadas a Procedimientos Almacenados en ADO.NET

Para ejecutar un procedimiento almacenado se debe utilizar el tipo de comando para procedimientos almacenados y elegir el método de ejecución que más se adecue al resultado esperado. Si el procedimiento almacenado que se va a ejecutar posee parámetros, se deben crear antes de ejecutarlo indicando si son de entrada o de salida.

El siguiente procedimiento almacenado de la base NorthWind es utilizado en varios ejemplos. Su nombre es `CustOrdersDetail` y requiere un parámetro que corresponde con el identificador o número de pedido.

```
CREATE PROCEDURE CustOrdersDetail @OrderID int
AS
```

```
SELECT ProductName, UnitPrice=ROUND(Od.UnitPrice, 2),
  Quantity, Discount=CONVERT(int, Discount * 100),
  ExtendedPrice=ROUND(CONVERT(money,Quantity*(1-Discount)*Od.UnitPrice), 2)
FROM Products P, [Order Details] Od
WHERE Od.ProductID = P.ProductID and Od.OrderID = @OrderID
```

A continuación se presenta un ejemplo de llamada al procedimiento almacenado `CustOrderDetail` en SQL Server.

Por ejemplo*:* Supónga que existe una conexión abierta a la base de datos Northwind en SQL Server llamada `ObjCnn`.

```
// Nota: Debe estar declarado el namespace System.Data.SqlClient;

SqlCommand objCmd = new SqlCommand("CustOrdersDetail", ObjCnn);
objCmd.CommandType = CommandType.StoredProcedure;

SqlParameter objPar =new SqlParameter("@OrderID", SqlDbType.Int);
objPar.Direction = ParameterDirection.Input;
objPar.Value = 14;
objCmd.Parameters.Add(objPar);
SqlDataReader dr = objCmd.ExecuteReader(CommandBehavior.CloseConnection);
```

Nota: El proveedor de datos SQL Server .NET no soporta el carácter interrogación para obtener la información de los parámetros, pero sí lo hace el proveedor OLE DB .NET.

La clase DataReader

Un objeto de la clase `DataReader` es muy similar a un `Recordset` de ADO pero que únicamente soporta las siguientes características:

- `ReadOnly`. Solo lectura.
- `Forward Only`. Solo lee hacia delante.

Estas dos características hacen que un objeto de la clase `DataReader` sea muy rápido ya que no tiene implementadas las características de edición, eliminación, inserción, navegación de registros hacia atrás, etc. que redundan en un peor rendimiento del objeto. Es una eficiente alternativa para orígenes de datos desconectados, pero es menos escalable que la clase `Dataset`, ya que no permite ser heredado por otra clase.

La clase `DataReader` no tiene constructor y es necesario llamar al método `ExecuteReader` del objeto `Command` correspondiente para crear un objeto `DataReader`. Debe ser cerrado explícitamente.

El método `Read` se utiliza para leer los registros. Cuando se crea un objeto `DataReader` siempre se abre y se posiciona en el primer registro, con lo que no hay que hacer una llamada explicita a ningún método del tipo `MoveFirst`. El método `Read` lee siempre la siguiente fila.

```
DataReader ObjDR = ObjCMD.ExecuteReader();
ObjDR.Read();
```

Para acceder a los valores de las columnas del objeto `DataReader` se puede acceder tanto por número como por nombre.

Por ejemplo:

```
DataReader ObjDR = ObjCMD.ExecuteReader();
ObjDR.Read();
Console.Write(ObjDR.GetInt32(0).ToString());
// o Console.Write(ObjDR[“EmployeeID”].ToString());
```

Otras propiedades interesantes de la clase `DataReader` son:

- La propiedad `RecordsAffected` toma un valor determinado cuando se realizan acciones como inserciones, actualizaciones o eliminaciones de registros. En caso de realizar consultas de selección, esta propiedad no recoge ningún valor hasta que no se han leído todos los registros, y se haya cerrado el objeto `DataReader` correspondiente.

- La propiedad `NextResult` sirve para acceder al siguiente conjunto de resultados.

- `FieldCount` devuelve el número de columnas o campos del conjunto de registros pero no aporta información acerca del número de filas devueltas.

Existen multitud de métodos específicos de lectura como `GetString`, `GetBoolean`, `GetInt32`, etc. Otros métodos interesantes del objeto `DataReader` son `GetFieldType` y `GetValues` que devuelven una matriz de objetos describiendo los valores de las filas.

En la versión actual del framework .NET se describen dos clases que implementan la interface `IDataReader`, una para el estándar OLE DB y otra para SQL Server llamadas `OleDbDataReader` y `SqlDataReader` respectivamente.

OleDbDataReader

Es la clase que se utiliza en caso de un acceso de datos estándar y es válida para cualquier base de datos, incluido SQL Server. Aunque lógicamente para SQL Server se utilizará la clase `SqlDataReader`.

Cuando se leen datos y la conexión que subyace está ocupada no se puede hacer otras operaciones sobre el objeto `OleDbDataReader` excepto cerrarlo. Y una vez que el objeto `OleDbDataReader` está cerrado sólo se pueden utilizar dos de sus propiedades: `IsClosed`() y `RecordsAffected`

SqlDataReader

La clase `SqlDataReader` debe ser utilizada cuando se accede a una base de datos en SQL Server. Esta clase es muy similar a la clase `OleDbDataReader`, ya que tiene las mismas características. Existen multitud de métodos específicos de este proveedor del tipo `GetSqlXxx` que devuelven datos de distintos tipos de datos de SQL. Por ejemplo: `GetSqlBinary`, `GetSqlString`, etc.

Un ejemplo con Visual Studio .NET:

En este ejemplo se representa, en una lista de un formulario, los empleados de la base de datos de Northwind en SQL Server. Se crea un proyecto con un formulario, posteriormente se agrega una etiqueta (`label1`) y un cuadro de lista (`listBox1`) desde el cuadro de herramientas. Finalmente se escribe en el evento `Load` del formulario el siguiente código, cuya finalidad es mostrar cómo trabaja un objeto `DataReader` para rellenar una lista con datos de una base de datos.

```
String strConn = "SERVER=localhost; UID=sa; DATABASE=Northwind;";
string strCmd = "SELECT EmployeeID, LastName, FirstName FROM Employees";
string strLista;

// Rellena el DataReader
SqlDataReader ObjDR;
SqlConnection ObjCnn = new SqlConnection(strConn);
SqlCommand ObjCmd = new SqlCommand(strCmd, ObjCnn);
ObjCnn.Open();
ObjDR = ObjCmd.ExecuteReader();
label1.Text = "Lista de Empleados";

// Recogiendo por los registros.
while(ObjDR.Read())
{
      strLista = ObjDR.GetInt32(0).ToString();
      strLista += " - " + ObjDR.GetString(1);
      strLista += " " + ObjDR.GetString(2);
      listBox1.Items.Add(strLista);
}
ObjCnn.Close();
```

El resultado se representa en la siguiente figura:

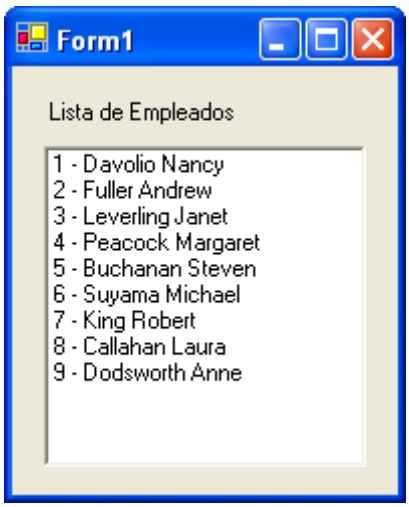

Figura 24.15. Resultado del ejemplo con `DataReader`

La clase DataAdapter

La clase `DataAdapter` se encarga de las operaciones entre la capa de datos y la capa intermedia, donde los datos son transferidos. Se puede decir que sirve como puente entre un objeto `DataSet` y un origen de datos asociado para recuperar y guardar datos.

Básicamente permite rellenar (`Fill`) el objeto `DataSet` para que sus datos coincidan con los del origen de datos y permite actualizar (`Update`) el origen de datos para que sus datos coincidan con los del `DataSet`.

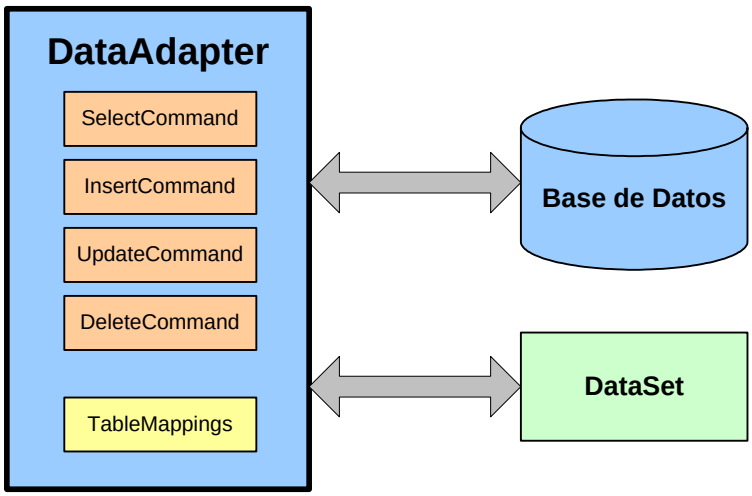

Figura 24.16. Diagrama del objeto `DataAdapter`

Los constructores de la clase son:

- `DataAdapter(Command selectCommand)`. Se utiliza un comando de selección (cláusula SELECT) como parámetro.
- `DataAdapter(String selectCommandText, String selectConnectionString)`. Se utiliza una sentencia SQL de selección con una cadena de conexión como parámetros.
- `DataAdapter(String selectCommandText, Connection selectConnection)`. Se utilizan los parámetros sentencia SQL de selección y un objeto de tipo conexión.

En la interface `IDataAdapter` se declaran los siguientes métodos (toda clase que implemente esta interface está obligada a implementarlos).

- `Fill(DataSet ds)`. Rellena un objeto de la clase `Dataset`.
- `FillSchema(DataSet ds, SchemaType st)`. Rellena un esquema con un `DataSet` indicando el tipo de esquema a rellenar.
- `Update(DataSet ds)`. Actualiza el `DataSet` correspondiente.

- `GetFillParameters()`. Recoge el conjunto de parámetros cuando se ejecuta una consulta de selección.

La colección `TableMappings` mantiene el seguimiento del enlace entre un objeto `DataTable` del `DataSet` y un origen de datos.

Un objeto de la clase `DataAdapter` soporta *batch-updates*, es decir que puede realizar varias actualizaciones en un solo proceso. Cuando una aplicación llama al método `Update`, la clase examina la propiedad `RowState` para cada fila en el `DataSet` y ejecuta la clausula `INSERT`, `UPDATE` o `DELETE` requerida.

Las principales propiedades de la clase `DataAdapter` son `SelectCommand`, `InsertCommand`, `UpdateCommand` y `DeleteCommand`, que es donde residen las sentencias SQL para consultar, insertar, actualizar y eliminar registros respectivamente.

Todos los proveedores de datos de la plataforma .NET soportan una clase que hereda de la clase `DbDataAdapter` que hereda a su vez de la clase `DataAdapter`, la cual implementa la interface `IDataAdapter`. En muchos casos es suficiente con una clase que implemente el interface `IDataAdapter`.

Las clases más representativas soportadas por los proveedores son `OleDbDataAdapter` y `SqlDataAdapter` que son muy similares.

Utilización del componente DataAdapter en Visual Studio .NET.

En el momento en que se agrega el componente `DataAdapter` desde el cuadro de herramientas hasta el formulario, aparece automáticamente un asistente para crear y configurar el componente `DataAdapter`.

A) Caso de un objeto SqlDataAdapter

Para este ejemplo se va a utilizar un componente `SqlDataAdapter` que se configurará a partir del procedimiento almacenado `CustOrdersDetail` de la base datos NorthWind en SQL Server.

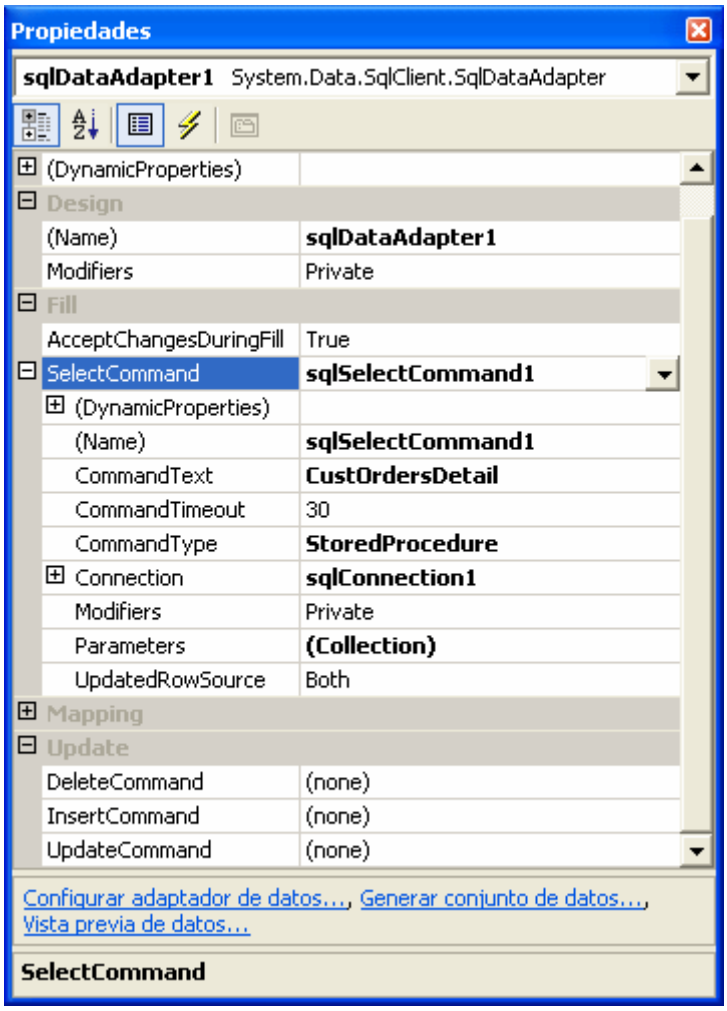

Figura 24.17. Propiedades del objeto `SqlDataAdapter`

Una vez generado el componente `sqlDataAdapter1`, se pueden modificar las propiedades así como utilizar los tres asistentes o herramientas siguientes.

1) Asistente para configurar el adaptador de datos. Este asistente es el mismo que aparece cuando se arrastra el componente `DataAdapter` en el formulario. Por ejemplo, si se selecciona el procedimiento almacenado `CustOrdersDetail` de la base datos NorthWind en SQL Server, se puede rellenar los datos para la selección, para agregar nuevos registros, para actualizar y para borrar registros.

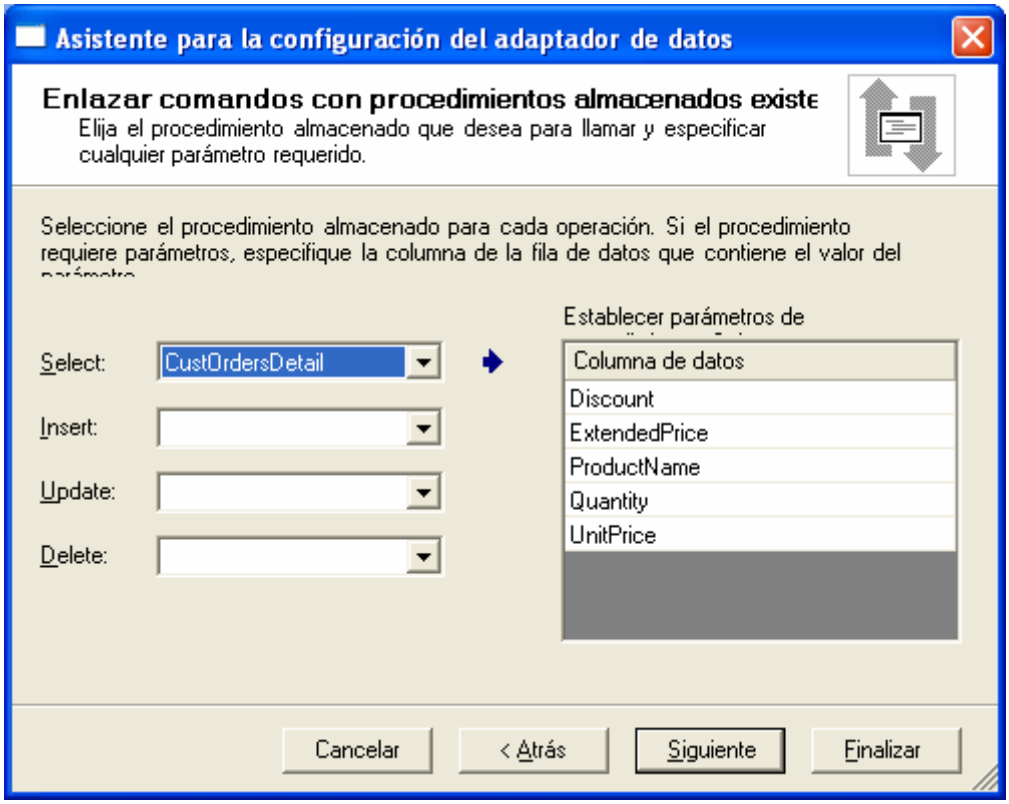

Figura 24.18. Asistente para la configuración del adaptador de datos (`DataAdapter`)

2) Asistente para generar el conjunto de datos. Es el asistente que permite generar un objeto `DataSet`.

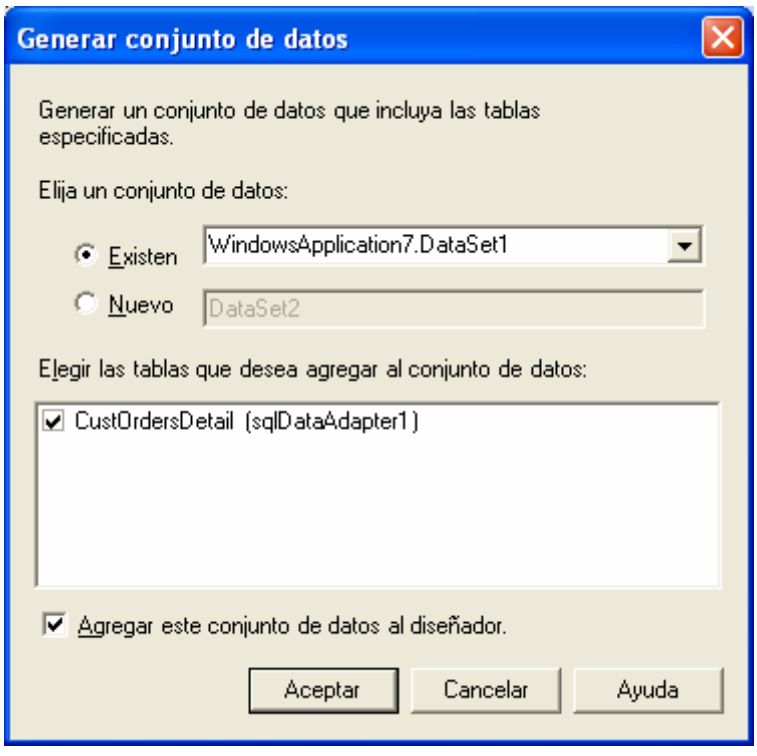

Figura 24.19. Generación del conjunto de datos (`DataSet`)

3) *Asistente para visualizar los datos*, que se utiliza para disponer de una vista previa de los datos que recupera el objeto `DataAdapter` del origen de datos.

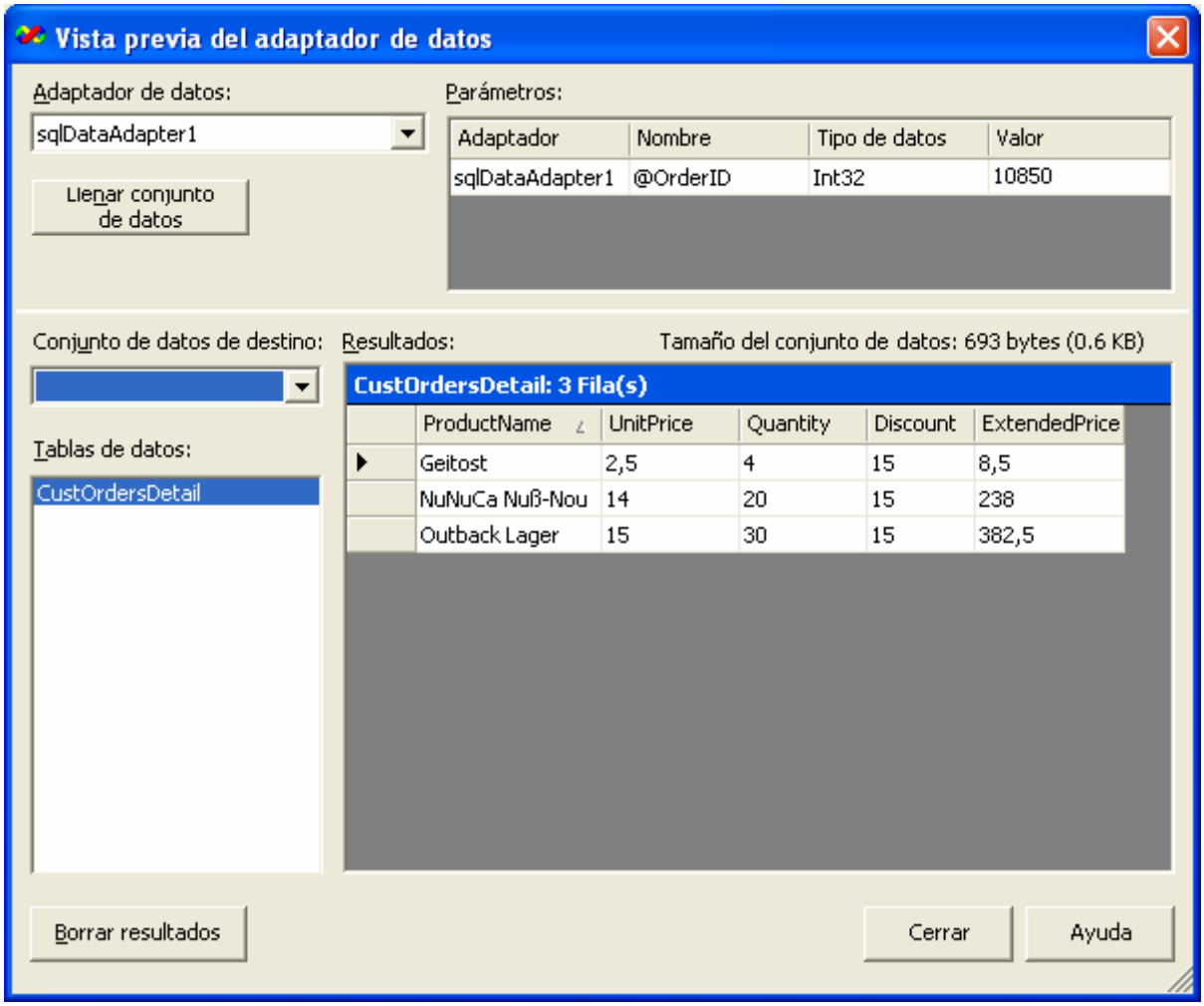

Figura 24.20. Asistente para la vista previa de datos

En este ejemplo en particular como el procedimiento almacenado recibe un parámetro, para obtener el conjunto de datos es necesario rellenarlo. En la figura 24.20 se ha elegido el código de pedido u `OrderID` con el valor 10850.

B) Caso de un objeto OleDbDataAdapter

En este caso se utiliza un componente `OleDbDataAdapter` que se configurará a partir de una consulta de selección sobre la tabla `Empleados` de la base datos Neptuno en Access.

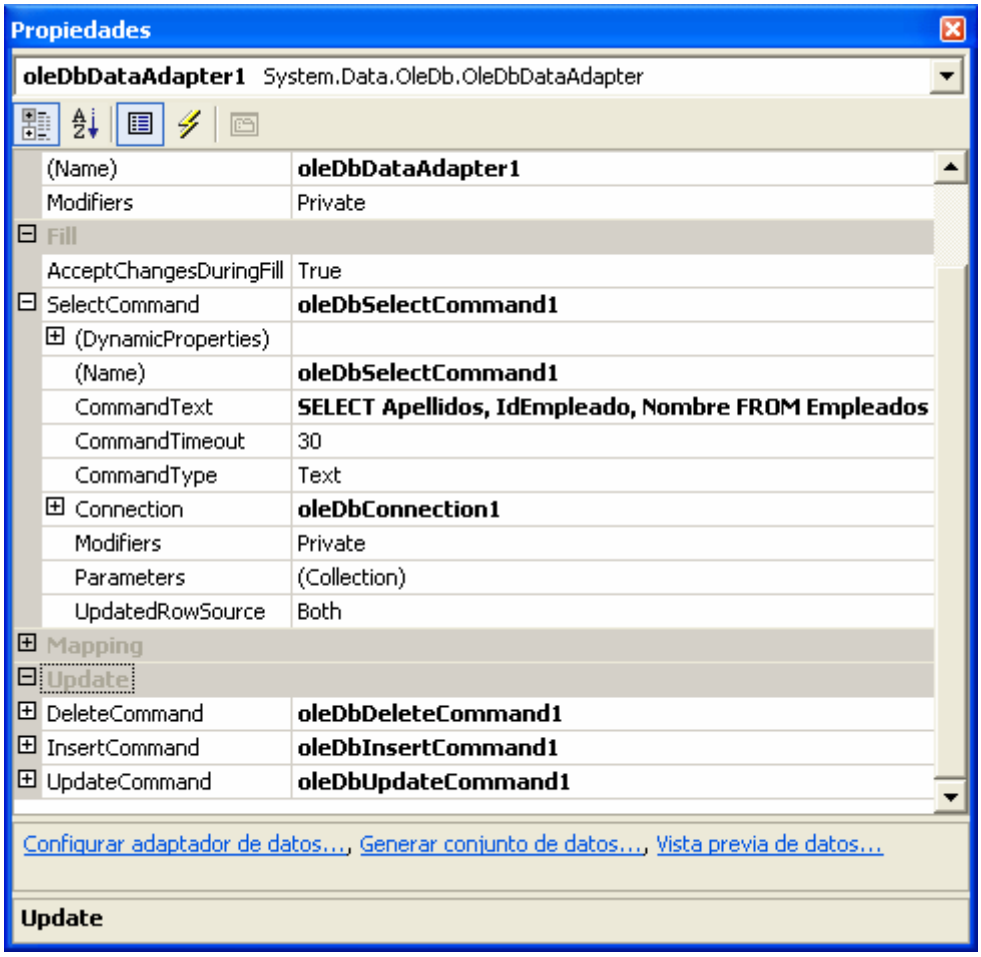

Figura 24.21. Propiedades del objeto `OleDbDataAdapter`.

Se puede observar que tiene las tres mismas herramientas o asistentes que el objeto `SqlDataAdapter`.

1) Asistente para configurar el adaptador de datos. Este asistente es el mismo que en el caso anterior salvo que no se pueden crear procedimientos almacenados.

En dicho asistente se pueden configurar unas opciones avanzadas para el caso de instrucciones SQL, que es el caso del ejemplo. Dichas opciones avanzadas permiten configurar la generación o no de las instrucciones `Insert`, `Update` y `Delete`. También permiten configurar si la concurrencia será optimista y por último si se ejecuta una sentencia SELECT después de las instrucciones `Insert` y `Update` para recuperar los diferentes valores de la base de datos.

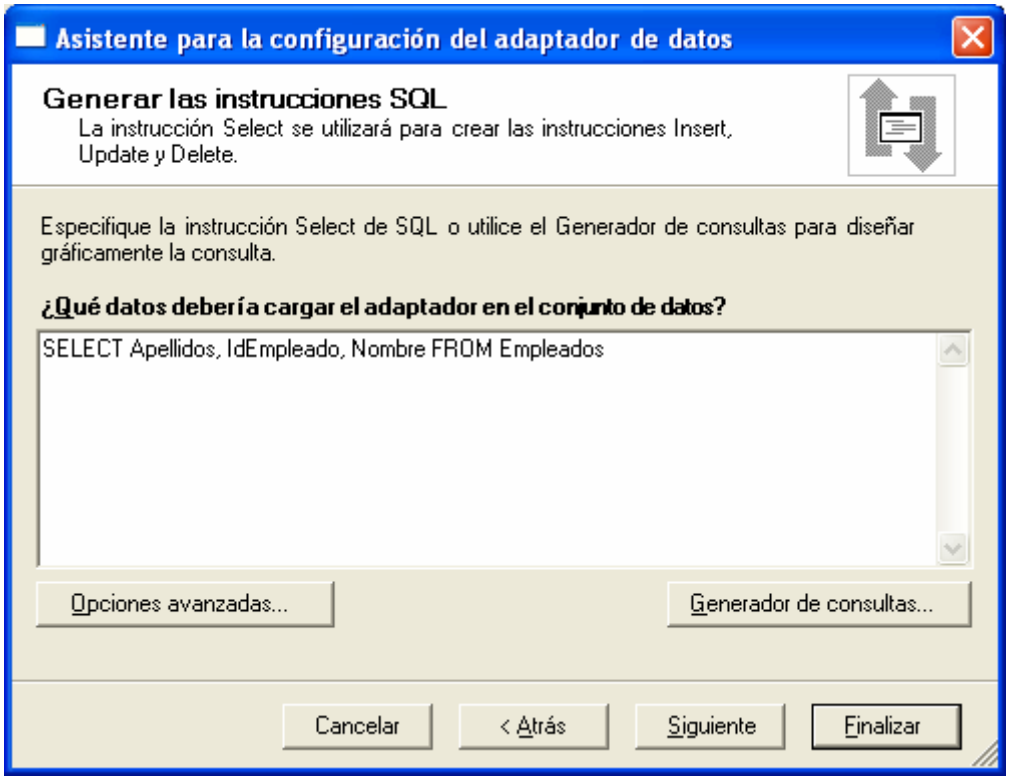

Figura 24.22. Asistente para la configuración del `DataAdapter`

2) Asistente para generar el conjunto de datos. Es el asistente que permite generar un objeto `DataSet`.

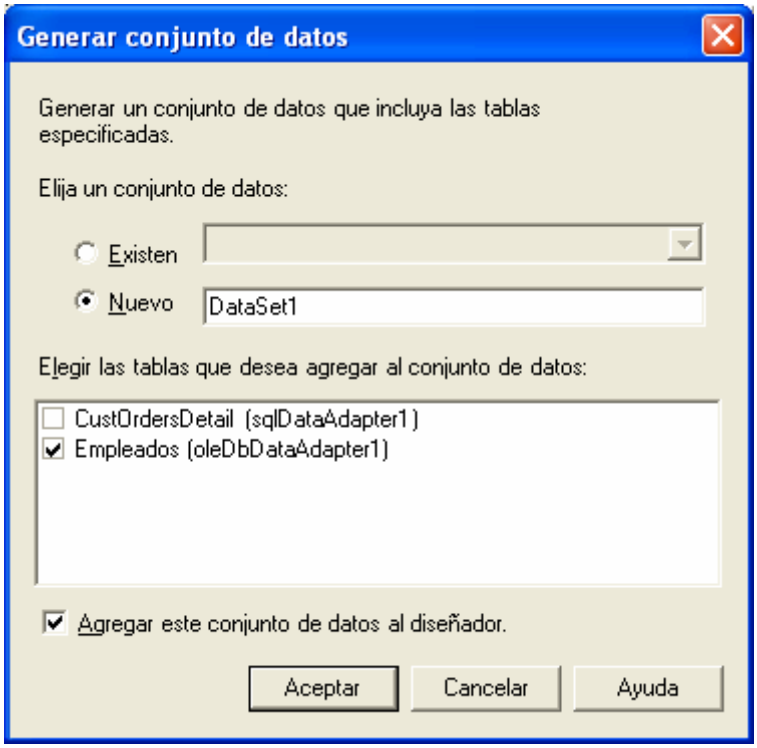

Figura 24.23. Generación del conjunto de datos

3) Asistente para visualizar los datos, que se utiliza para disponer de una vista previa de los datos que recupera el objeto `DataAdapter` del origen de datos.

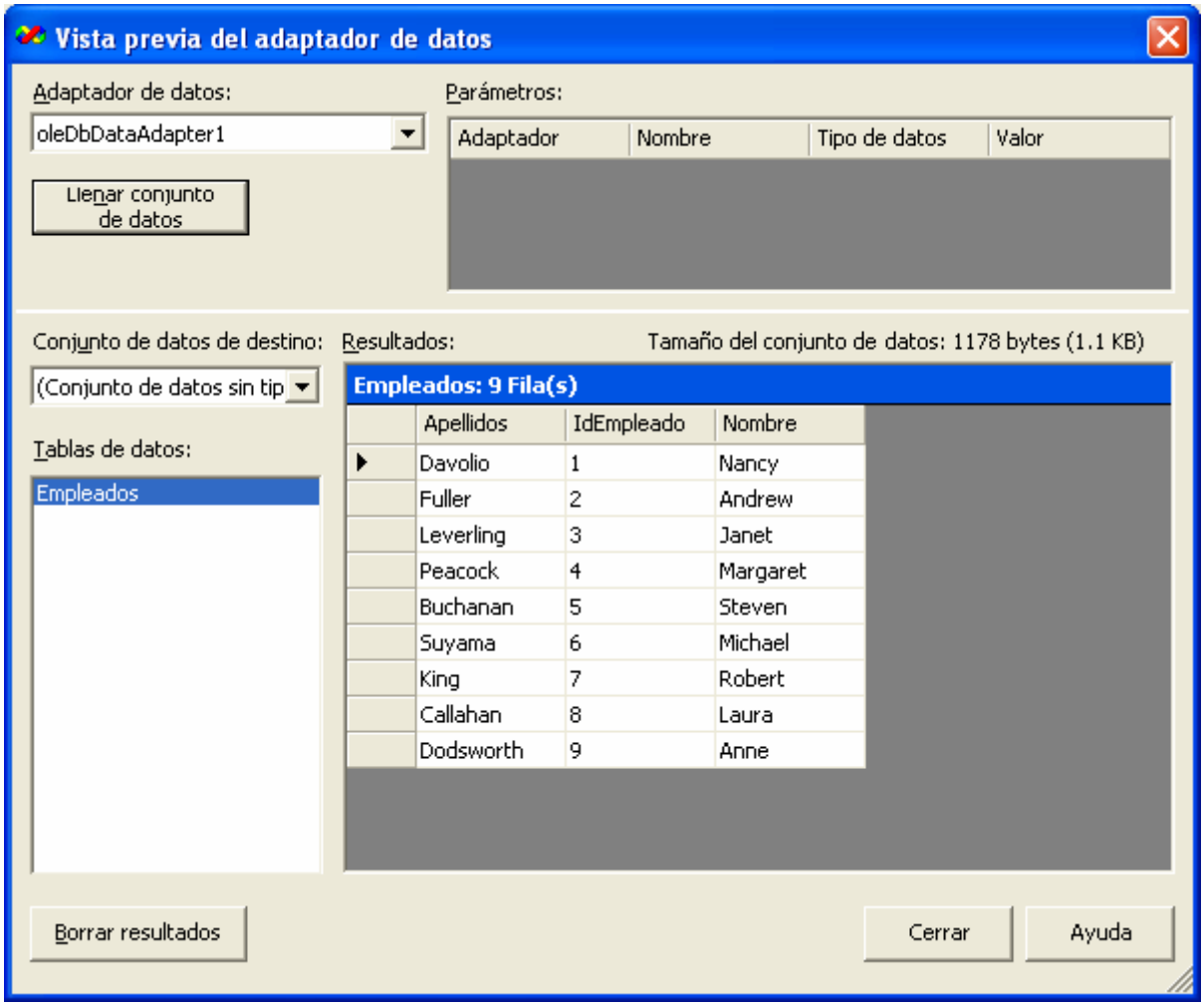

Figura 24.24. Vista previa del objeto `OleDbDataAdapter`.

La clase DataSet

La clase `DataSet` es la clase principal de la arquitectura ADO.NET. Un objeto de esta clase es una representación en memoria de los datos en forma relacional. Es un caché de datos extraídos de un contenedor de datos genérico. El objeto `DataSet` se rellena con `DataAdapters`, con datos locales, o bien con XML.

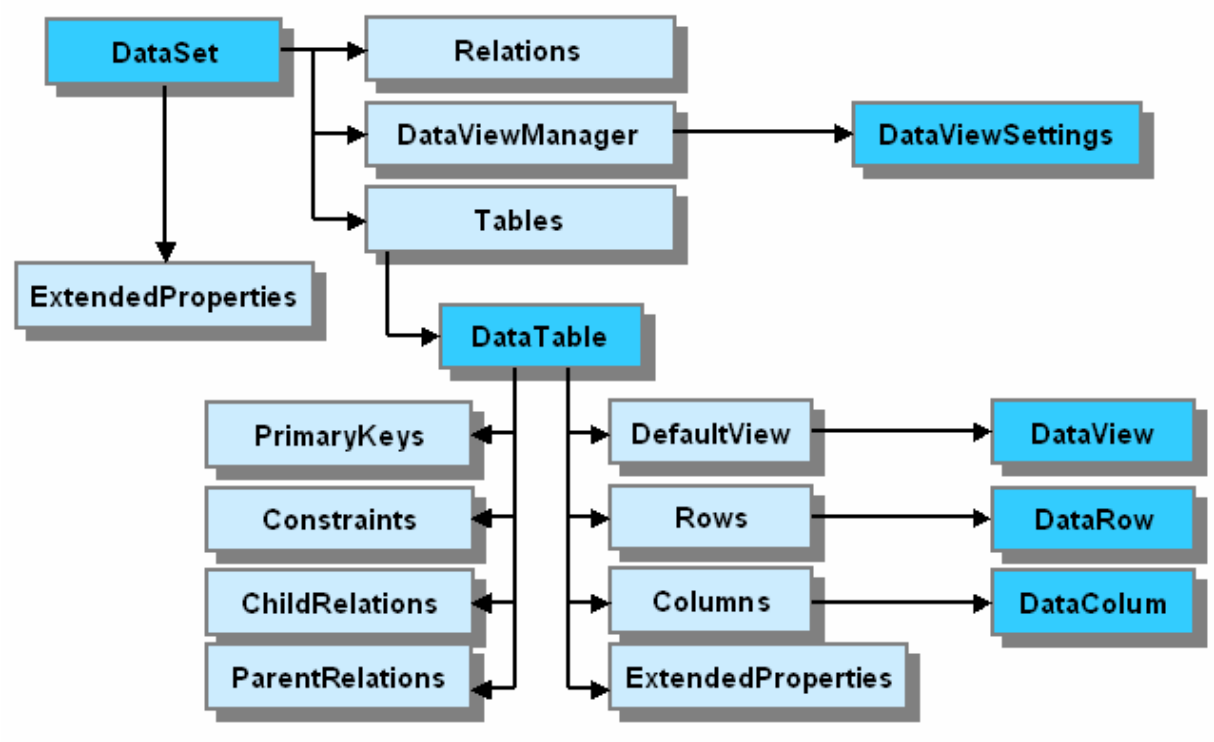

Figura 24.25. Arquitectura del objeto `DataSet`

Las colecciones más importantes son la colección `DataTables` que contiene las tablas donde reside la información y la colección `DataRelations` que contiene las distintas relaciones entre las tablas del `DataSet`. Se puede contener información específica del usuario a través de la colección `ExtendedProperties`.

Tanto los datos, como el esquema o ambos se pueden representar en formato XML, y para acceder a la información se utilizan los métodos `GetXml`, `ReadXml` y `WriteXml`.

Una característica importante de la clase `DataSet` es que permite aceptar o rechazar todos los cambios realizados en las tablas, ya sean eliminaciones, actualizaciones o creaciones de nuevos registros. Todos estos cambios se pueden aceptar en un solo paso siendo muy útil cuando se necesiten procesar distintos bloques de datos durante un tiempo en memoria. A veces puede ser interesante mantener durante toda una sesión un `DataSet` en memoria consiguiendo realizar un solo acceso a la base de datos para almacenarlo y otro al final de la sesión para almacenar los cambios.

Creación de un DataSet mediante controles en Visual Studio .NET

Para crear y rellenar un `DataSet` con Visual Studio .NET se debe realizar los siguientes pasos:

1. Se crea un formulario y se añade un control de tipo `Connection` (`OleDbConnection` o `SqlConnection`) arrastrándolo desde el cuadro de herramientas. Se configura la conexión para acceder al origen de datos deseado, tal y como se ha descrito anteriormente.

2. Se añade al formulario un control `DataAdapter` (`OleDbDataAdapter` o `SqlDataAdapter`) y se configura como también se ha descrito.

3. Se selecciona el control `DataAdapter` y se accede a la herramienta o asistente para generar el conjunto de datos o `DataSet`.

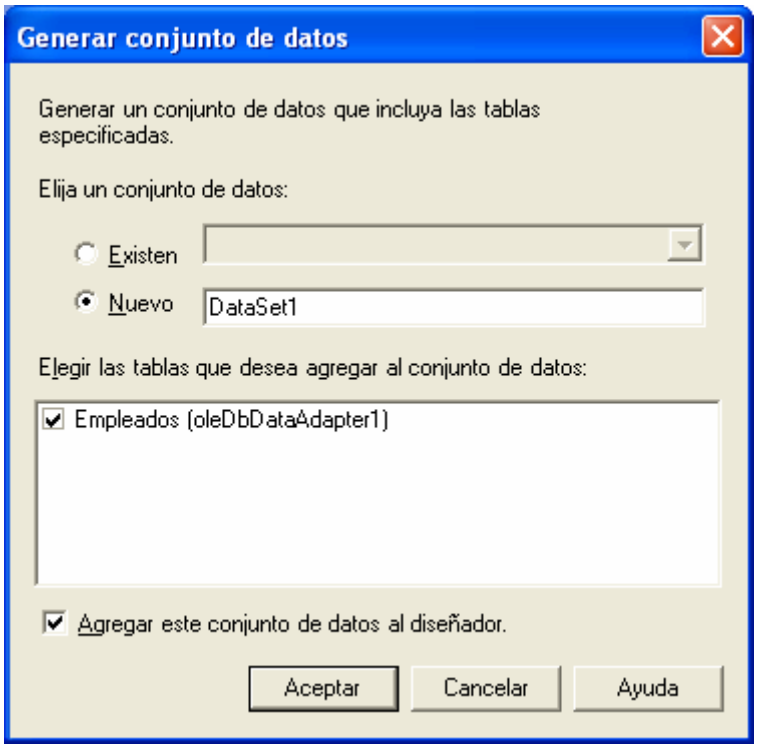

Figura 24.26. Asistente de generación del `DataSet`

Una vez que el objeto `DataSet` ha sido añadido al formulario, se puede acceder a las propiedades del conjunto de datos o a ver el esquema del `DataSet`.

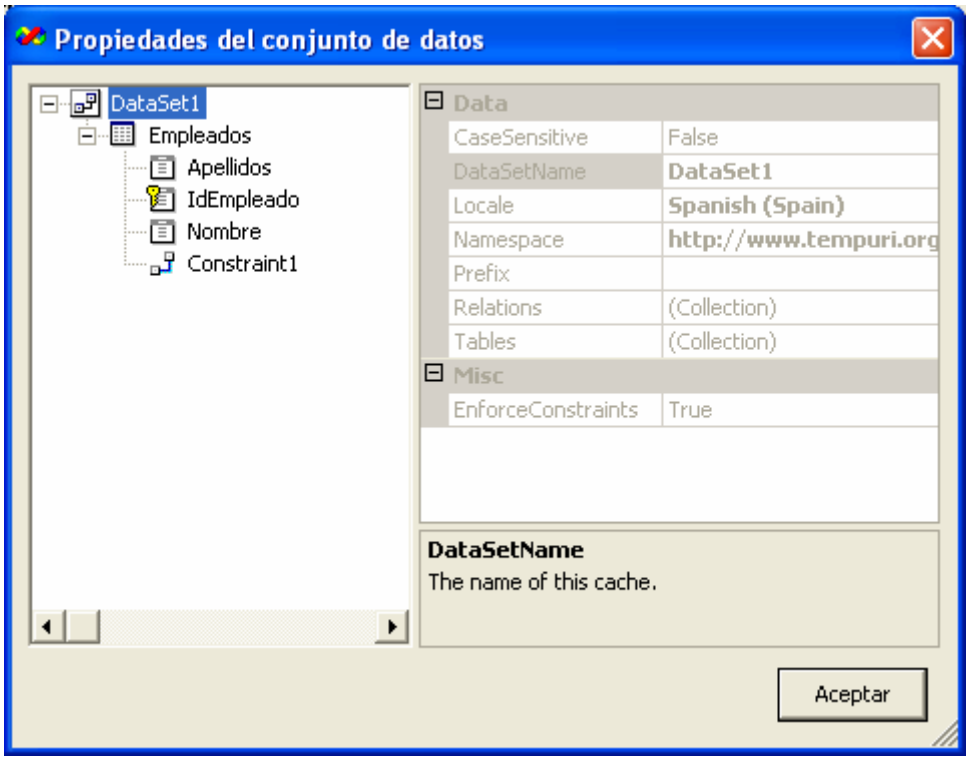

Figura 24.27. Propiedades del `DataSet` o conjunto de datos.

Si se desea agregar más tablas al `DataSet` se pueden crear más objetos `DataAdapter` y agregarlos al `DataSet` existente. La ventaja de añadir las diferentes tablas desde controles `DataAdapter` es que se agregan tanto los campos con sus tipos de datos, nombres, etc, y además se añaden las restricciones debidas a la clave primaria de las tablas en el origen de datos.

La clase DataTable

La clase `DataTable` representa una tabla en memoria y es donde reside la información de los datos. Un objeto de la clase `DataTable` se puede utilizar con o sin un objeto `DataSet`, pero si se quiere rellenar, entonces debe asociarse al `DataSet`. Con un objeto `DataTable` al igual que con un objeto `DataSet` se tiene la posibilidad de rechazar o aceptar todos los cambios en una sola vez. Por otra parte las tablas tienen relaciones, restricciones, claves primarias, filas y columnas que son objetos que se ven a continuación.

El método `Fill` del `DataAdapter` rellena un `DataSet` con sólo las columnas de la tabla y los registros. La información acerca de las restricciones de la tabla, si se desean, deben ser añadidas al `DataSet` utilizando la colección `Constraints` del objeto `DataTable`.

Se recomienda que el objeto `DataSet` posea la información existente sobre restricciones en el origen de datos, y para conseguirlo se puede recurrir o bien a llamar al método `FillSchema` del objeto `DataAdapter`, o bien a establecer la propiedad `MissingSchemaAction` del objeto `DataAdapter` a `AddWithKey` antes de llamar al método `Fill`. Añadir la información del esquema antes de rellenar el objeto `DataSet` asegura que la información de la clave primaria se incluye en el objeto `DataTable`.

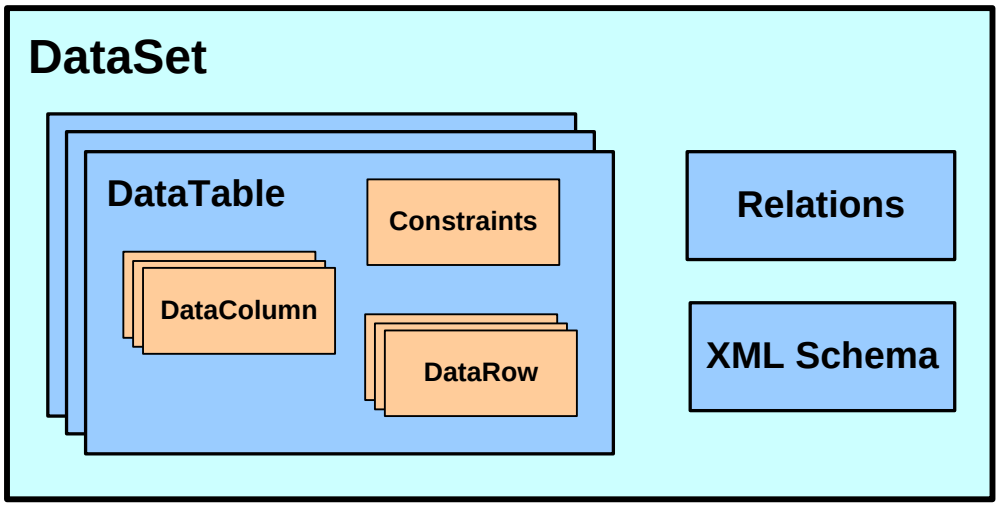

Figura 24.28. Estructura principal de los objetos `DataSet` y `DataTable`

La clase DataColumn

El conjunto de columnas de un objeto `DataTable` corresponde a una colección de objetos `DataColumn`. Un objeto `DataColumn` tiene como propiedades importantes la propiedad `ColumnName` que indica el nombre de la columna, y la propiedad `DataType` que indica el tipo de columna. Además tiene la propiedad `AllowDBNull` para saber si la base de datos permite

nulos para esa columna, la propiedad `AutoIncrement` que detecta si la columna es o no autonumérica, la propiedad `DefaultValue` que establece o lee el valor por defecto de la columna y la propiedad `Expresión` que establece o recoge la expresión utilizada para filtrar filas o la expresión de un campo calculado entre columnas o la expresión de un campo suma, promedio, etc. Las propiedades `ReadOnly` y `Unique` se utilizan para establecer o recoger si la columna es de solo lectura y cada valor el único para todas las filas.

La clase DataRow

Como en el caso de la clase `DataColumn`, el conjunto de filas de un objeto `DataTable` corresponde a una colección de objetos `DataRow`. La propiedad `Item[n]` o `Item[nombre]` lee o establece el valor de las columnas. Dos de los métodos más importantes del objeto `DataRow` son el método `IsNull(n)` o `IsNull(nombre)` que indica si una columna es nula.

Ejemplo de DataSet, DataTable, DataColumn y DataRow

El siguiente código muestra cómo utilizar los objetos `DataSet`, `DataTable`, `DataColum` y `DataRow`.

```
DataSet objDS = new DataSet();
DataTable objTabla = new DataTable("TablaLocal");

// Preparando las columnas
DataColumn objCol = new DataColumn("Ciudad");
objCol.DataType = Type.GetType("System.String");
objTabla.Columns.Add(objCol);
DataColumn objCol2 = new DataColumn("Pais");
objCol2.DataType = Type.GetType("System.String");
objTabla.Columns.Add(objCol2);

// Se añade el DataTable al Dataset (Antes de Rellenar)
objDS.Tables.Add(objTabla);

// Se añaden unas filas...
DataRow objFila = objTabla.NewRow();
objFila["Ciudad"] = "Bilbao";
objFila["Pais"] = "España";
objTabla.Rows.Add(objFila);

// Se vincula al DataGrid
dataGrid1.DataMember = " TablaLocal ";
dataGrid1.DataSource = objDS;
```

Restricciones de datos

Para establecer reglas o restricciones con los objetos en memoria, ADO.NET presenta las siguientes restricciones, que pueden ser de varios tipos:

`ForeignKeyConstraint`. Fuerza un vinculo entre dos `DataTables` del `DataSet`

`UniqueConstraint`. Se asegura que la columna que tenga esta restricción no pueda tener filas con el mismo dato en esa columna.

ADO .NET proporciona además la propiedad `PrimaryKey` del objeto `DataTable` que obtiene o establece un número de columnas como claves primarias de la tabla en memoria, pero este conjunto de columnas que forman la clave primaria en el objeto `DataTable` no tiene por que coincidir con las columnas que forman la clave primaria en la base de datos. El uso de esta clave ayuda a prevenir errores comunes cuando se actualiza la tabla en memoria y a veces el uso de esta clave primaria en memoria ahorra la reconciliación de datos posterior con la base de datos.

Ejemplo de una aplicación Maestro-Detalle con Visual Studio .NET

Se pretende realizar una aplicación .NET que conste de un formulario donde aparezcan los pedidos en un `DataGrid` y sus líneas de pedido en otro, y que cada vez que se seleccione un pedido concreto, aparezcan sus correspondientes líneas en el otro `DataGrid`. Además no se utilizarán los controles de conexión, adaptación y contenedor de datos incluidos en Visual Studio .NET.

A) Para una base de datos en SQL Server

Para realizar este ejemplo se utilizan las tablas `Orders` y `Order Details` de la base de datos Northwind en SQL Server.

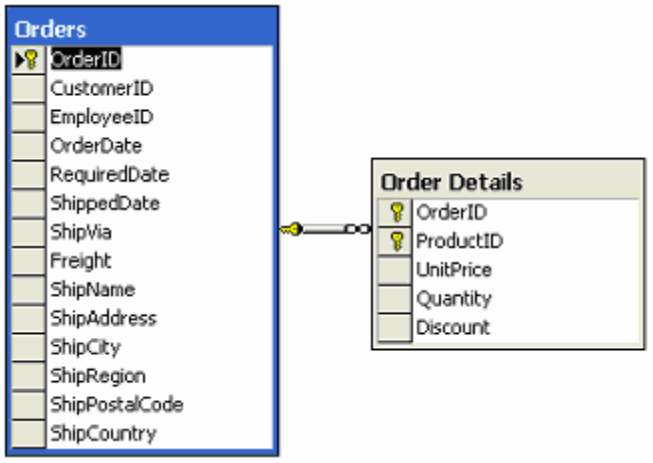

Figura 24.29. Tablas `Orders` y `Order Details` de la base de datos Northwind

Se insertan dos controles `Datagrid` en el formulario: el control `datagrid1` se situará en la parte izquierda y contendrá la información relativa a los pedidos (`Orders`) y el control `datagrid2` estará en la parte derecha y contendrá las líneas correspondientes al pedido seleccionado (`Order Details`).

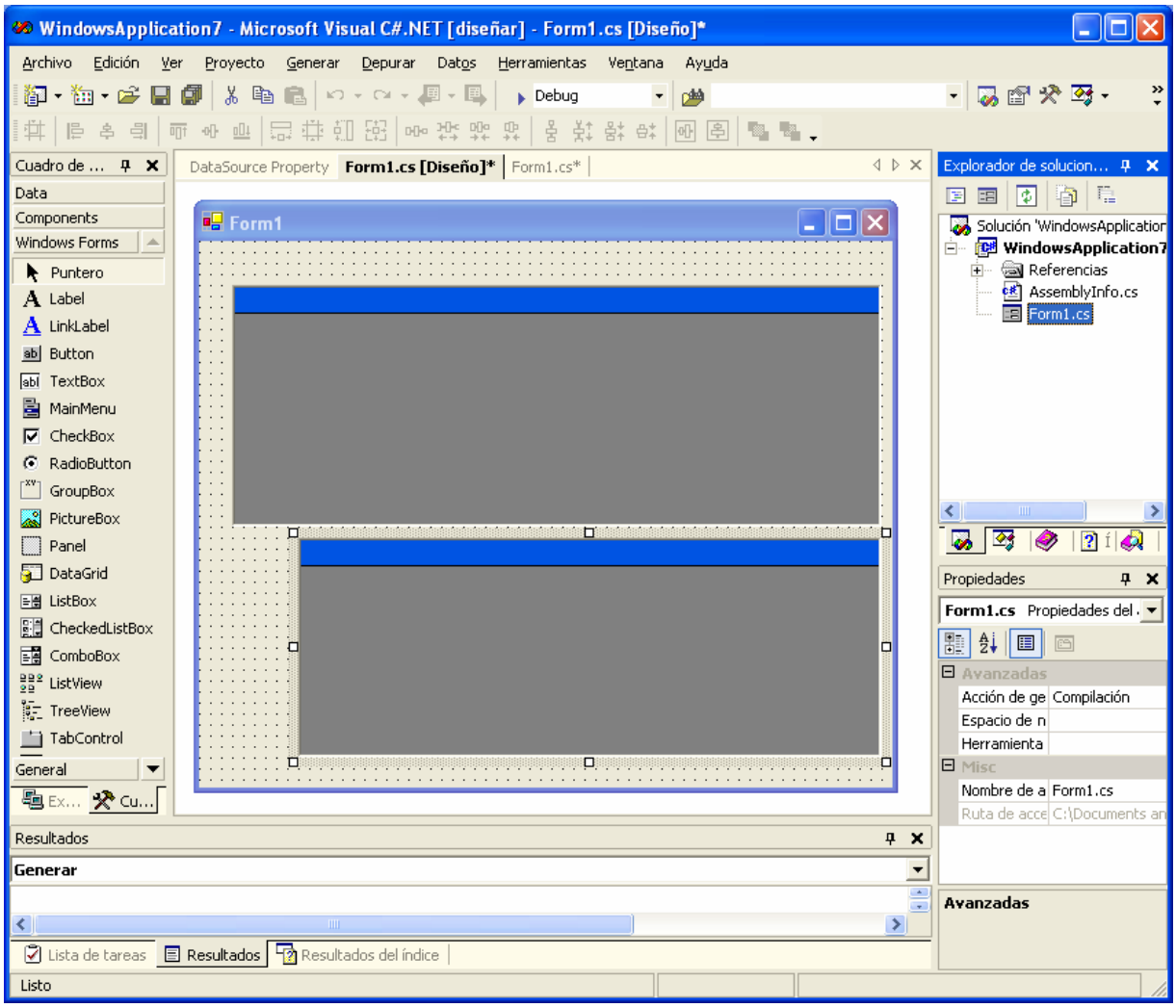

Figura 24.30. Formulario Maestro-Detalle

Como se va a trabajar con la base de datos en SQL Server se añadirá el `namespace` correspondiente:

```
using System.Data.SqlClient;
```

A continuación se define el objeto `Dataset` con el se va a trabajar y que se llama `odsPedidos`. Este objeto se define como público en la clase `Form1`.

```
public DataSet odsPedidos;
```

Una vez definido el objeto `Dataset` donde va a residir la información de los pedidos y de las líneas de pedido, se procede a cargar el objeto `Dataset` en el evento *Load* del formulario `Form1`.

```
private void Form1_Load(object sender, System.EventArgs e)
{
}
```

Lo primero que hay que hacer es definir la conexión con el servido. Después se incorpora la información de la tabla pedidos (`Orders`) al `DataSet` y posteriormente se añade al `DataSet` la información de la tabla líneas de pedido.

Una vez que se tiene el `DataSet` con la información, se deben establecer las relaciones entre las tablas que lo forman. Se puede obnservar que las tablas del `DataSet` no tienen porqué coincidir con las tablas del origen de datos, ni en número de campos, ni en las claves, ni en los nombres de las tablas, campos, etc.

Y por último se vinculan las tablas a los `DataGrids` correspondientes.

```
private void Form1_Load(object sender, System.EventArgs e)
{

      string
      sConexion="database=northwind;uid=sa;pwd=sa;server=localhost;";

      DataSet odsPedidos = new DataSet();

      // Se rellena el DataSet con la tabla Pedidos (Orders)
      string sSQLPedidos = "SELECT * FROM Orders";
      SqlDataAdapter ObjDA = new SqlDataAdapter(sSQLPedidos,sConexion);
      ObjDA.Fill(odsPedidos,"Orders");

      //Se rellena el DataSet con las lineas del Pedido (Order Details)
      string sSQLLineas = "SELECT * FROM [Order Details]";
      ObjDA.SelectCommand.CommandText = sSQLLineas;
      ObjDA.Fill(odsPedidos,"OrderDetails");

      //Se establece la relación entre las entidades en el DataSet
      DataColumn ObjColPedidos =
      odsPedidos.Tables["Orders"].Columns["OrderID"];
      DataColumn ObjColLineas =
      odsPedidos.Tables["OrderDetails"].Columns["OrderID"];
      DataRelation ObjRelacion = new
      DataRelation("relPedidos",ObjColPedidos,ObjColLineas);
      odsPedidos.Relations.Add(ObjRelacion);

      //Se vincula a los DataGrids la información
      dataGrid1.DataSource = odsPedidos;
      dataGrid1.DataMember = "Orders";
      dataGrid2.DataSource = odsPedidos;
      dataGrid2.DataMember = "Orders.relPedidos";
}
```

Y el resultado se presenta en la siguiente figura:

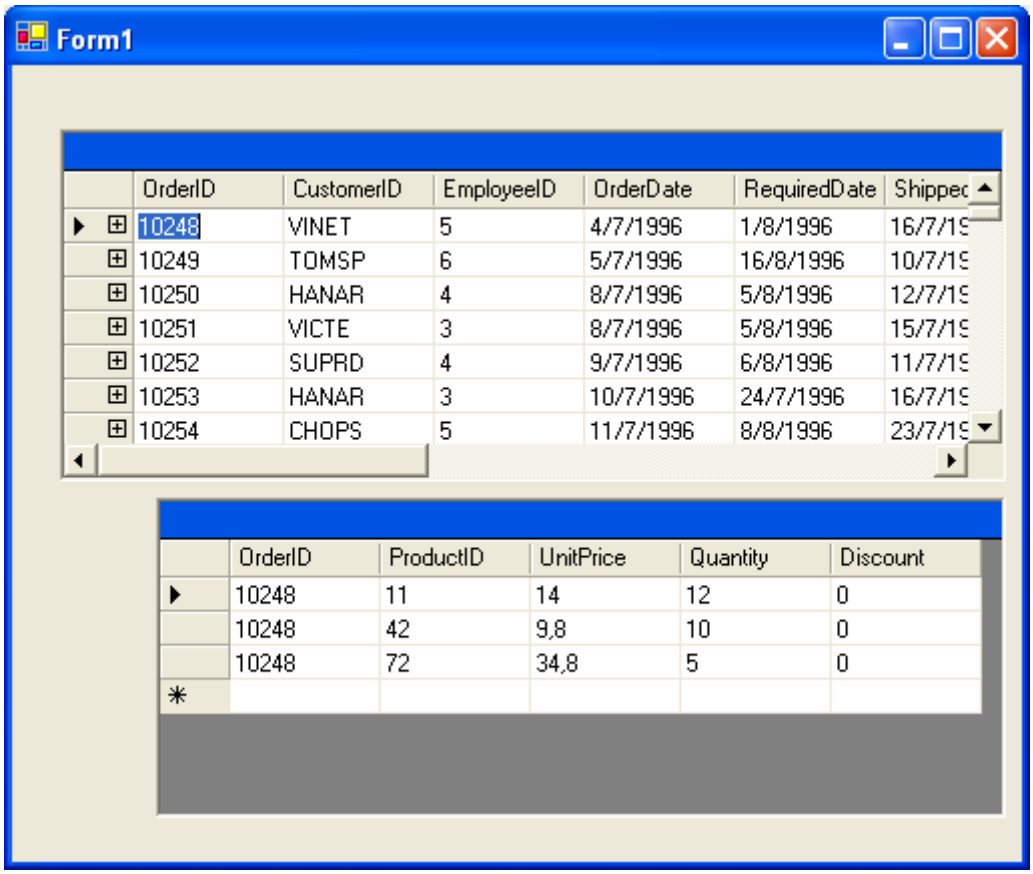

Figura 24.31. Formulario Maestro-Detalle de Pedidos y líneas de pedidos

Este ejemplo también se puede realizar utilizando los controles de conexión, adaptación y contenedor de datos incluidos en el Visual Studio .NET.

B) Para una base de datos Access

Supónga que se desea realizar un formulario Maestro-Detalle de la base de datos Neptuno en Access, pero esta vez se utilizarán los controles de Visual Studio.

En primer lugar se agrega la conexión `OleDbConnection1` a la base de datos Neptuno y después se despliega la conexión en el explorador de conexiones y se arrastran al formulario las tablas `Productos` y `Proveedores`. Entonces se observa como se crean dos objetos `DataAdapter`.

Se elige uno cualquiera de los dos `DataAdapter` y se genera el conjunto de datos o `DataSet` desde las propiedades del mismo. En la ventana del asistente de la generación del `DataSet` se elige también el otro `DataAdapter`.

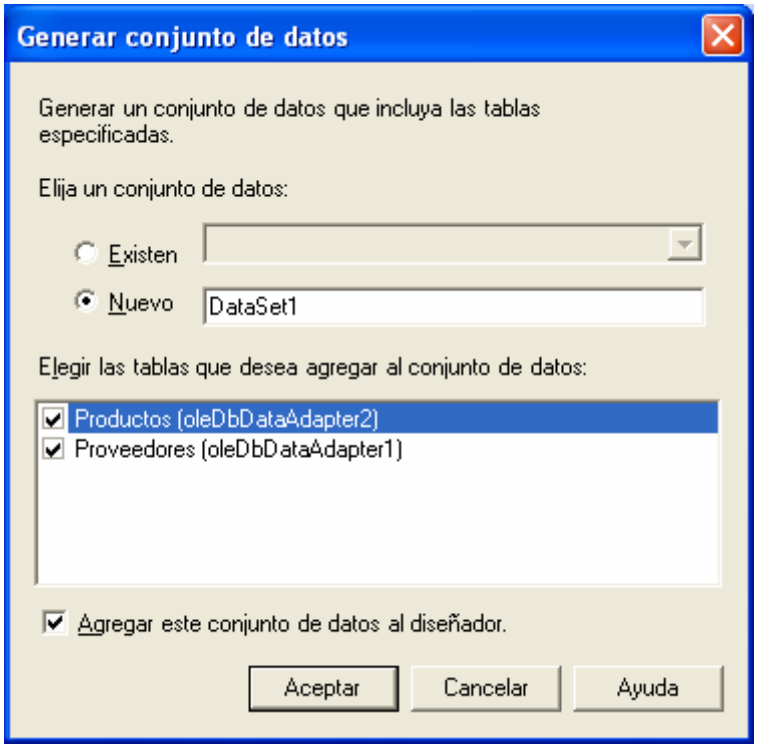

Figura 24.32 Asistente para generar un `DataSet` a partir de dos objetos `DataAdapter`.

Una vez que el `DataSet` ha sido generado, se procede a crear la relación entre las tablas generadas en el `DataSet` -`Productos` y `Proveedores`- que posteriormente rellenarán los `DataGrid` del formulario. Para crear la relación se selecciona el campo *`IDProveedor`* de la tabla `Proveedores` y se accede al menú contextual mediante el botón derecho para crear una relación que relaciona un proveedor con varios productos.

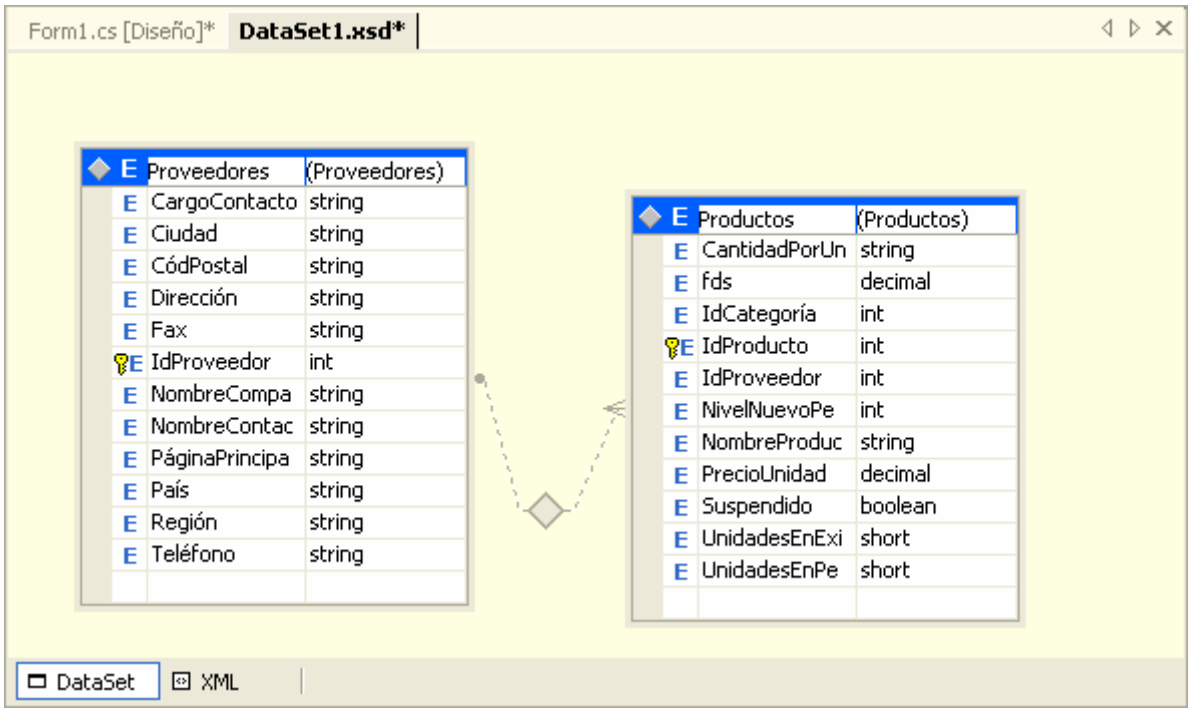

Figura 24.33. Relación de las tablas `Proveedores` y `Productos` en el `DataSet`.

Se accede a la propiedad `DataSource` del `DataGrid1` que es donde se quiere exponer los proveedores, y se selecciona el `dataSet11` como valor y en la propiedad `DataMember` se selecciona `Proveedores`.

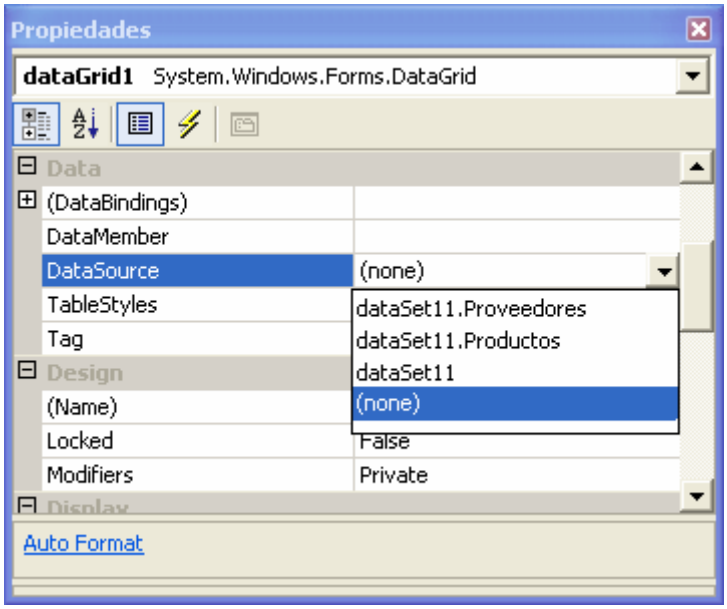

Figura 24.34. Propiedad `DataSource` del `DataGrid`.

En el caso del `DataGrid2` se rellena la propiedad `DataSource` con `dataSet11` y la propiedad `DataMember` con `Proveedores.ProveedoresProductos`, que de esta forma se marca el vinculo entre los dos controles `DataGrid`. Si se rellena la propiedad `DataMember` con el valor `Productos`, en el `DataGrid2` aparecerían todos los productos, y no los del proveedor seleccionado en el `DataGrid1`.

Ahora solo falta rellenar el `DataSet`, que se realizará al cargar el formulario, mediante los dos objetos `OleDbDataAdapter`, uno para cargar la tabla proveedores y otro para cargar la tabla productos.

```
private void Form1_Load(object sender, System.EventArgs e)
{
      oleDbDataAdapter1.Fill(dataSet11);
      oleDbDataAdapter2.Fill(dataSet11);
}
```

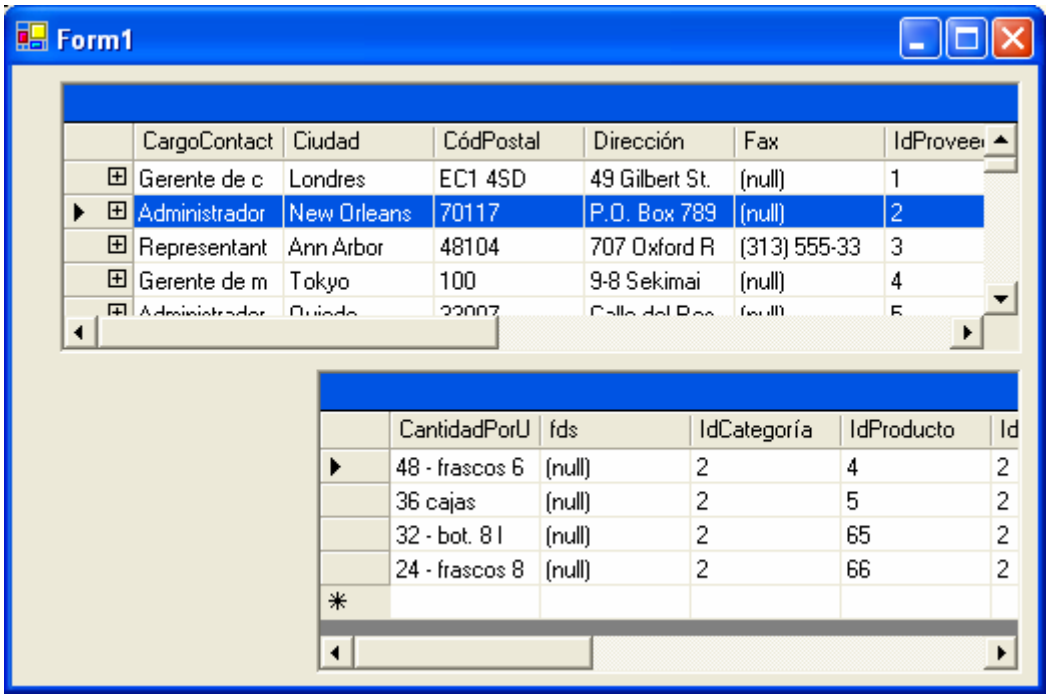

Figura 24.35. Formulario Maestro-Detalle de `Proveedores` y `Productos`.

Este ejemplo se ha realizado utilizando los controles, herramientas y asistentes que proporciona Visual Studio .NET, pero también se podría haber realizado de forma similar al ejemplo anterior -en SQL Server-, que no utiliza ningún control de datos .NET, ni tampoco asistentes.

La clase DataSet y XML

La clase `DataSet` es la única clase de la plataforma .NET que puede recoger datos de una fuente en XML y enviarlos en el mismo formato. Para realizar estas acciones un objeto `DataSet` utiliza el método `GetXml` que devuelve una representación XML del `DataSet` pero no incluye la información del esquema (para ello se deberá utilizar el método `GetXmlSchema`).

Los métodos del `DataSet` para leer son `ReadXml` que lee tanto los datos como el esquema, y el método `ReadXmlSchema` que solo se utiliza parta leer la información del esquema.

En el siguiente ejemplo se rellena un objeto `DataSet` desde un fichero XML.

```
String strFichero = "C:\XML\ejemplo.xml"
System.IO.StreamReader ObjSR = new System.IO.StreamReader(strFichero);
DataSet ObjDS = new DataSet();
ObjDS.ReadXml(ObjSR);
ObjSR.Close();
```

El método `WriteXml` se utiliza para enviar el contenido del `DataSeta` un objeto `StreamWriter` en formato XML ; es más rápido que escribir lo que devuelve el método `GetXml` y se puede utilizar independientemente de tener la información del esquema. En caso de querer escribir la información del esquema se utilizará el método `WriteXmlSchema`.

El objeto `DataSet` tiene las propiedades `Namespace` que establece u obtiene el `namespace` usado cuando se lee o escribe el documento XML, y `Prefix` que recoge o establece el prefijo XML que sirve de alias del `namespace` del `DataSet`.

Para crear un `DataSet` además de hacerlo con el método `Fill` del `DataAdapter`, se puede partir de código XML+XSD. También existen otros objetos aparte del `DataSet` leer información en XML como son `XmlReader`, `TextReader` o `StreamReader`.

```
System.Xml.XmlReader;
System.IO.StreamReader;
System.IO.TextReader
```

En Visual Studio .NET se puede ver el esquema del objeto `DataSet`, utilizando la opción correspondiente.

Figura 24.36. Representación en XML del `DataSet` de pedidos.

La clase DataView

Un objeto de la clase `DataView` representa una vista ordenada y/o filtrada de un objeto `DataTable` y se utiliza mucho cuando se desea vincular datos con los `WebForms` (Formularios Web) o los `WinForms` (Formularios Windows).

Un objeto `DataView` contiene objetos `DataRowView` que tiene como propiedades importantes `Item`(índice) para acceder a la fila deseada y `Count` para recoger el número total de filas.

Las principales acciones que se pueden realizar con un objeto `DataView` son la de filtrar y ordenar. Para filtrar es posible utilizar las propiedades `RowFilter` para obtener o establecer la expresión para filtrar las filas, o la propiedad `RowStateFilter` que recoge el estado del filtro. Para ordenar se utilizan las propiedades `Sort`, que es la expresión de cadena que se utiliza para ordenar, y `ApplyDefaultSort` que indica si se utiliza la ordenación por defecto.

A continuación se realiza un ejemplo de filtro:

```
private void FiltrarFilas()
{
      int i;

      // Se crea un DataTable con una columna.
      DataTable ObjTabla = new DataTable("Tabla");
      DataColumn ObjColumna = new DataColumn("Columna");
      ObjTabla.Columns.Add(ObjColumna);

      // Se añaden 10 filas.
      DataRow ObjFila;
      for(i = 0;i < 10 ;i++)
      {
            ObjFila = ObjTabla.NewRow();
            ObjFila["Columna"] = "item " + i;
            ObjTabla.Rows.Add(ObjFila);
      }
      ObjTabla.AcceptChanges();

      // Crear un DataView con la Tabla.
      DataView ObjVista = new DataView(ObjTabla);

      // Cambiar el valor de una fila
      ObjTabla.Rows[1]["Columna"] = "Hello";

      // Añadir una fila
      ObjFila = ObjTabla.NewRow();
      ObjFila["Columna"] = "World";
      ObjTabla.Rows.Add(ObjFila);

      // Establece el RowStateFilter para ver solo las filas
      //  añadidas y modificadas.
      ObjVista.RowStateFilter =
            DataViewRowState.Added | DataViewRowState.ModifiedCurrent;

      // Muestra las filas
      MostrarVista(ObjVista, "Recién modificada y añadida");

      // Establece el filtro para mostrar el valor original
      //  de las filas modificadas
      ObjVista.RowStateFilter=DataViewRowState.ModifiedOriginal;
      MostrarVista(ObjVista,"Valor Original");

      // Borrar tres filas.
      ObjTabla.Rows[1].Delete();
      ObjTabla.Rows[2].Delete();
      ObjTabla.Rows[3].Delete();
```

```
      // Establece el RowStateFilter para ver solo las filas
      // añadidas y modificadas
      ObjVista.RowStateFilter=DataViewRowState.Deleted;
      MostrarVista(ObjVista,"Deleted");

      //Establece el filtro para mostrar solo las en curso
      ObjVista.RowStateFilter=DataViewRowState.CurrentRows;
      MostrarVista(ObjVista,"En Curso");

      // Establece el filtro para mostrar solo las filas que no cambian
      ObjVista.RowStateFilter=DataViewRowState.Unchanged;
      MostrarVista(ObjVista,"No cambian");

      // Establece el filto para mostrar las filas originales.
      ObjVista.RowStateFilter=DataViewRowState.OriginalRows ;
      MostrarVista(ObjVista,"Filas Originales");
}

private void MostrarVista(DataView dv,string label)
{
   Console.WriteLine("\n" + label);
   for(int i = 0;i < dv.Count ;i++) {
      Console.WriteLine(dv[i]["Columna"].ToString());
}
```

Además el objeto `DataView` soporta las operaciones generales de edición por medio de las propiedades `AllowNew`, `AllowDelete` y `AllowEdit` para establecer las operaciones básicas y los métodos `AddNew` y `Delete` para agregar una fila y eliminarla respectivamente.

Las operaciones de búsqueda del objeto `DataView` vienen reflejadas por el método `Find` que devuelve el índice de la fila encontrada por los criterios de filtrado.

Por ejemplo:

```
DataView ObjDV;
Integer i;
Object vals[1];
DataTable ObjTabla;

ObjDV = New DataView(ObjTabla);
ObjDV.Sort = "Customers";

// Encontrar el cliente llamado Isabel Martin.
vals[0]= "Isabel";
vals[1] = "Martin";
i = ObjDV.Find(vals);
Console.WriteLine(ObjDV[i]);
```

Se puede personalizar diferentes vistas con un objeto `DataTable` y utilizar un objeto `DataView` para representar una de esas vistas

El control DataView en Visual Studio .NET

Para crear un control `DataView` basta con arrastrarlo al formulario. Una vez asignado al formulario se establece la propiedad `Table` donde se selecciona la tabla del `DataSet` deseada.

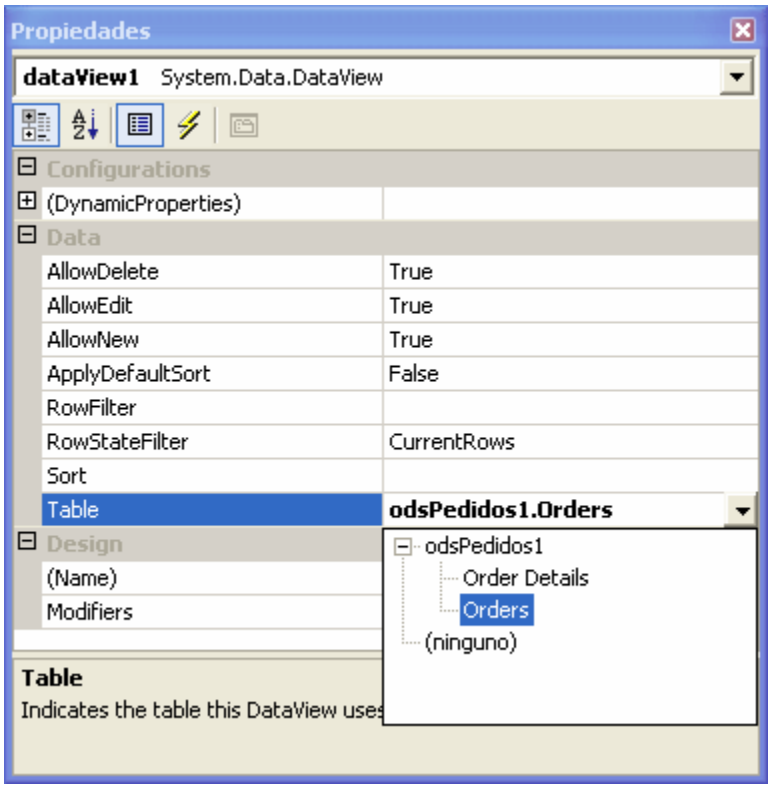

Figura 24.37. Propiedades del control `DataView`. Selección de la Tabla.

La clase DataRelation

Una relación es un vinculo dinámico establecido entre una o más columnas del mismo tipo de dos tablas y el objeto `DataRelation` representa una relación maestro-detalle entre dos tablas. Una vez que se ha establecido la relación entre dos tablas cualquier cambio que infrinja la relación provocará una excepción. Las relaciones admiten cambios en cascada desde la tabla maestro a la tabla detalle añadiendo un objeto `ForeignKeyConstraint` a la colección de restricciones del objeto `DataTable`. A continuación se muestra un ejemplo en el que se define una relación de actualización en cascada.

```
// Supónga que se tiene un DataSet llamado ObjDS
//   con las tablas Orders y OrderDetails
DataColumn ObjColMaestro = objDS.Tables(“Orders”).Columns(“OrderID”);
DataColumn ObjColDetalle = objDS.Tables(“OrderDetails”).Columns(“OrderID”);
ForeignKeyConstraint ObjFK = new ForeignKeyConstraint(“FK_OD_OrderID”,
ObjColMaestro, ObjColDetalle);
```

```
// Se crea la actualización en cascada
ObjFK.UpdateRule = Rule.Cascade;

// Se añade a la tabla OrderDetails del DataSet la restricción
ObjDS.Tables("OrderDetails").Constraints.Add(ObjFK)
```

A partir de aquí, estando en una fila de la tabla maestro es posible acceder a las filas de la tabla detalle directamente utilizando la el método `GetChildRows` del objeto `DataRow`, que devuelve un `array` de objetos `DataRow`.

Si se utilizan relaciones se debe tener en cuenta que no son transitivas, de tal forma que si la tabla A está relacionada con la tabla B, y por otro lado la tabla B está relacionada con la tabla C, no se puede decir que las tablas A y C estén relacionadas.

Creación de una relación con Visual Studio .NET.

Para crear una relación en un `DataSet` desde el diseñador de Visual Studio .NET basta con acceder a un `DataSet` donde existan varias tablas y seleccionar la opción de **agregar relación**. Una vez realizada esta acción se pueden configurar los elementos tanto primario como secundario, y las columnas que van a formar parte de la relación. Además se pueden configurar las reglas para actualizar o eliminar en cascada.

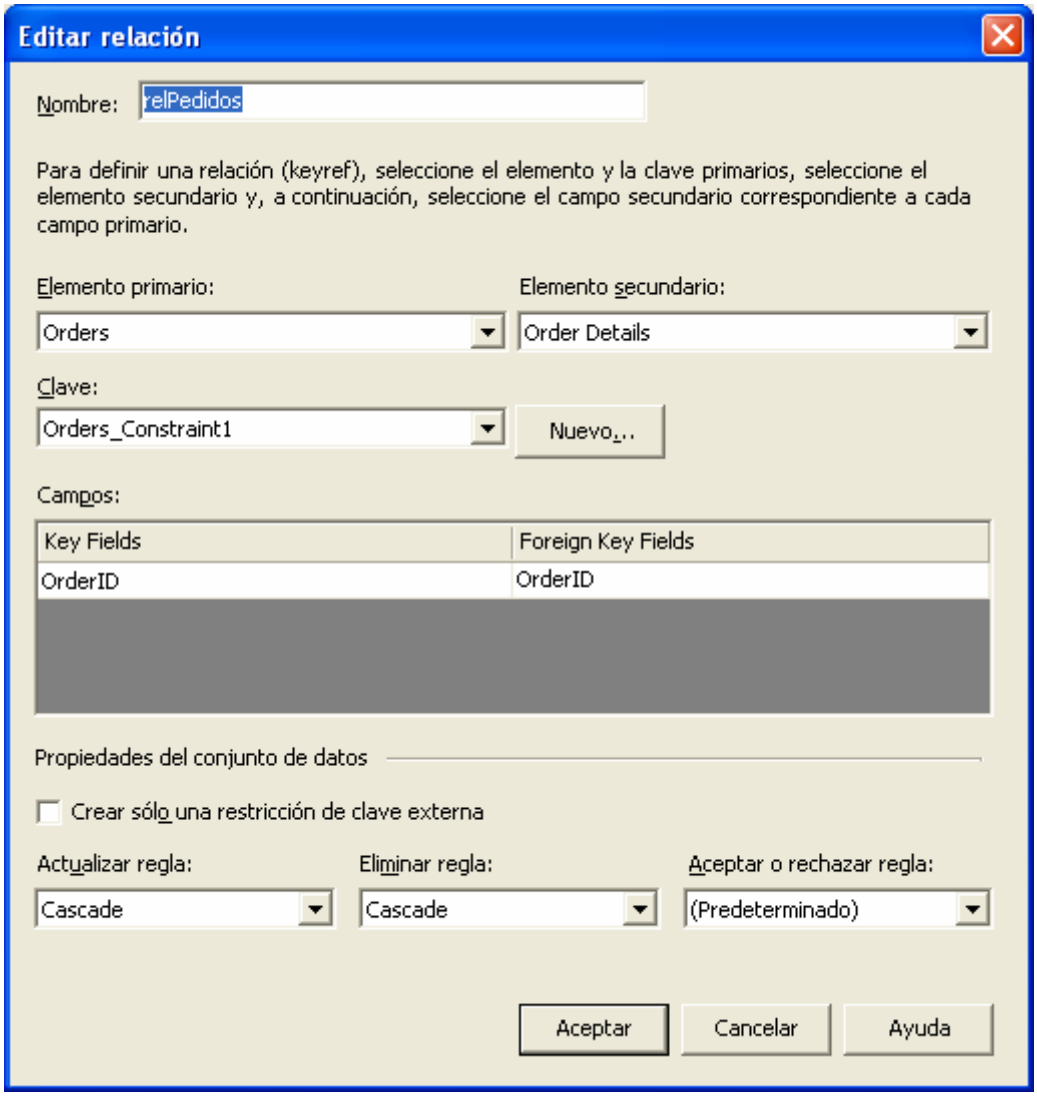

Figura 24.38. Edición de una relación

La representación de la relación se puede observar en la siguiente figura.

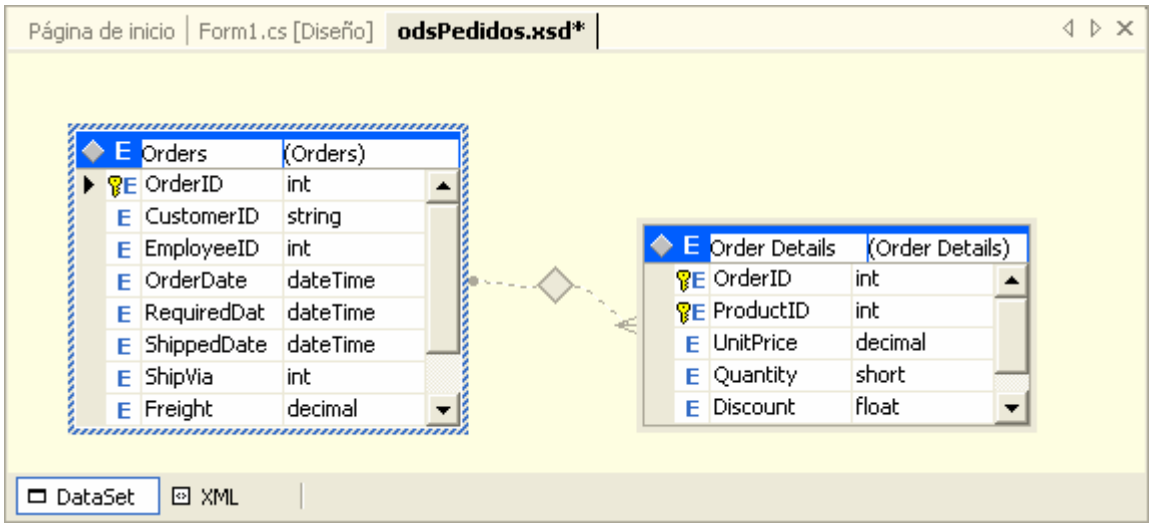

Figura 24.39. Representación gráfica del esquema del `DataSet` con Visual Studio

Una vez que la relación ha sido establecida se puede observar en el ejemplo maestro-detalle anterior, que al actualizar el valor de la columna `OrderID` 10251 por el valor 21, automáticamente las filas relacionas de la tabla `Order Details` cambian a 21, debido a la configuración en cascada, para la actualización, establecida en la relación.

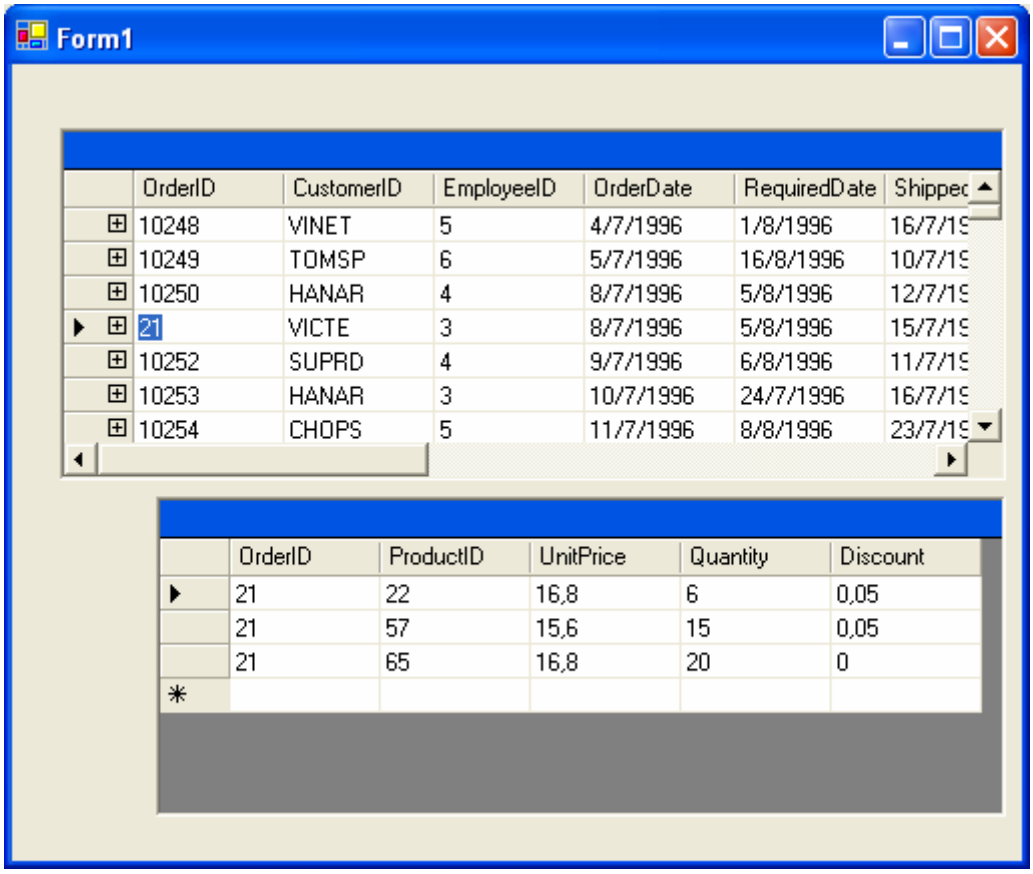

Figura 24.40. Actualización en cascada en el formulario maestro-detalle

Trabajando con un objeto DataSet

Las vistas con filtro permiten implementar esquemas del tipo maestro-detalle consiguiendo cargar en el mismo `DataSet` las tablas con la información del maestro y del detalle, posteriormente se establecen dos vistas con filtro una para la tabla maestra y otra para la tabla detalle. Con la versión anterior de ADO esto no era posible, y la única forma de traer toda la información era repetir los campos de la tabla maestro tantas veces como detalles tuviese, mediante una vista, consiguiendo peores prestaciones que con ADO .NET.

El método `Select` de la clase `DataTable` permite obtener un `array` o conjunto de filas que coinciden con un criterio especificado y bajo un orden fijado.

Cada `DataRow` tiene una propiedad `RowState` que puede ser examinada para determinar su estado. Los posibles estados de esta propiedad están representados en la siguiente tabla.

Estado	`Descripción`
`Unchanged`	No se han producido cambios desde la ultima llamada a `AcceptChanges`
`Added`	La fila ha sido añadida a la tabla pero no se ha llamado a

	`AcceptChanges`
`Modified`	Algún elemento de la fila ha sido cambiado
`Deleted`	La fila ha sido eliminada de la tabla utilizando el método `Delete`
`Detached`	La fila ha sido eliminada pero no se ha llamado a `AcceptChanges` o la fila ha sido creada pero no ha sido añadida a la tabla

Una de las formas de añadir nuevas filas a un objeto `DataTable` es crear un objeto `DataRow`, rellenarlo y luego agregarlo al objeto `DataTable`.

Por ejemplo:

```
DataRow ObjFila = ObjTabla.NewRow()
ObjFila["DNI"] = “9999999”;
ObjFila["Nombre"] = “Alvaro”;
ObjFila["Apellidos"] = “Garcia Lopez”;
ObjFila["Email"] = “agarcia@empresa.es”;
ObjTabla.Rows.Add(ObjFila);
```

Para borrar filas de un objeto `DataTable` basta seleccionar una serie de filas y luego llamar al método `Delete` o `Remove`.

`Delete` marca la fila como borrada pero mantiene el registro original y permite deshacer la eliminación.

`Remove` borra físicamente la fila de la tabla. Es como llamar al método `Delete` y posteriormente aceptar los cambios mediante `AcceptChanges`.

Las acciones de borrado de filas se realizan en el objeto `DataTable` que está en memoria. A la hora de actualizar con el origen de datos se debe tener en cuenta que una fila borrada con `Remove` no es percibida como borrada y por lo tanto no es borrada del origen de datos. Sin embargo la fila que es borrada con el método `Delete` es borrada en el origen de datos.

El hecho de marcar una fila como borrada (`Delete`) indica que la fila será eliminada de la tabla después de llamar a `AcceptChanges`. Cuando se realiza esta operación la propiedad `RowState` cambia al estado `Deleted`. Si se invoca el método `RejectChanges` entonces la propiedad `RowState` volverá a su estado inicial.

En el caso de que se borre una fila que justo se había añadido a la tabla y que había sido marcada como `New` utilizando `Delete`, `Delete` funciona como `Remove`.

Cuando se desea reconciliar las tablas en memoria con las tablas en el origen de datos se utiliza el método `GetChanges` para crear un segundo `DataSet` en local que contiene sólo los cambios que se van a hacer (filas marcadas). Posteriormente se examina la propiedad `HasErrors` del `DataSet` para ver si existen errores en alguna de las filas de las tablas del propio `DataSet`. Si se producen errores, se deberá reconciliar antes de actualizar los datos del `DataSet` (memoria) con el origen de datos. Para recoger los errores se utiliza el método `GetErrors` que devuelve los objetos `DataRow` que tienen errores. Si se posible reconciliar los

errores entonces se llama al método `Merge` para volver a incorporar los cambios del segundo `DataSet` al primero.

Para actualizar el origen de datos y refrescar la tabla se llama al método `AcceptChanges` y en caso de que se desee deshacer la actualización se llamará al método `RejectChanges`.

El método `AcceptChanges` se puede aplicar al objeto `DataSet`, al objeto `DataTable` o al objeto `DataRow`, y funciona en cascada, es decir, que si se aplica sobre el `DataSet`, se está aplicando sobre todas las tablas del `DataSet`, y si se aplica sobre el objeto `DataTable`, se está aplicando sobre todas las filas de esa tabla.

Cuando se llama al método `AcceptChanges`

- Los objetos `DataRow` que estén todavía en edición se almacenan
- La propiedad `RowState` de cada fila cambia adecuadamente.
- Las filas nuevas y modificadas pasan a ser filas normales y las borradas (`Deleted`) pasan a eliminadas definitivamente.

Cuando se llama al método `RejectChanges` se cancelan las filas que estaban en edición, se eliminan los nuevos registros y las filas modificadas o borradas (`Deleted`) pasan a su estado original.

En cuanto a la actualización del origen de datos, se recomienda utilizar como primera opción comandos SQL y como segunda opción la actualización tipo *Batch* o masiva que consiste en un proceso que extrae datos del lado cliente y los actualiza en la base de datos. Este tipo de actualización (*`Batch`*) se puede realizar con un conjunto de objetos `DataRow`, con un objeto `DataSet` o con un objeto `DataTable`. Para realizar la actualización tipo *Batch* se ha de utilizar el método `Update` del objeto `DataAdapter` que realiza una llamada a los comandos `INSERT`, `UPDATE` y `DELETE` para cada fila agregada, actualizada o borrada. Es conveniente utilizar el método `GetChanges` para enviar al origen de datos un `DataSet` más pequeño.

Ejemplo de actualización en Visual Studio .NET

Para realizar este ejemplo se ha de crear un proyecto nuevo en Visual Studio .NET, con un formulario y se ha de añadir una conexión a la base de datos Northwind en SQL Server. Se agrega, después, un control `DataAdapter` al formulario utilizando una sentencia SQL sobre la tabla de `Pedidos`:

```
SELECT
      OrderID,
      CustomerID,
      EmployeeID,
      OrderDate,
      RequiredDate,
      ShippedDate,
      ShipVia
FROM Orders
```

Una vez generado con asistente el control `DataAdapter`, entonces se generan automáticamente las propiedades `DeleteCommand`, `InsertCommand` y `UpdateCommand`.

Posteriormente con el control `DataAdapter` se genera el control `DataSet`, y se agrega un control `DataGrid` y un control `Button` al formulario. Más adelante se vincula el control `DataGrid` con la tabla `Orders` del `DataSet` y se prepara el evento `Load` del formulario para que se rellene el objeto `DataSet` que automáticamente rellenará el `Grid`.

```
private void Form1_Load(object sender, System.EventArgs e)
{
      sqlDataAdapterPedidos.Fill(dataSetPedidos,"Orders");
}
```

Posteriormente se agregan tres filas (`DataRow`) al formulario.

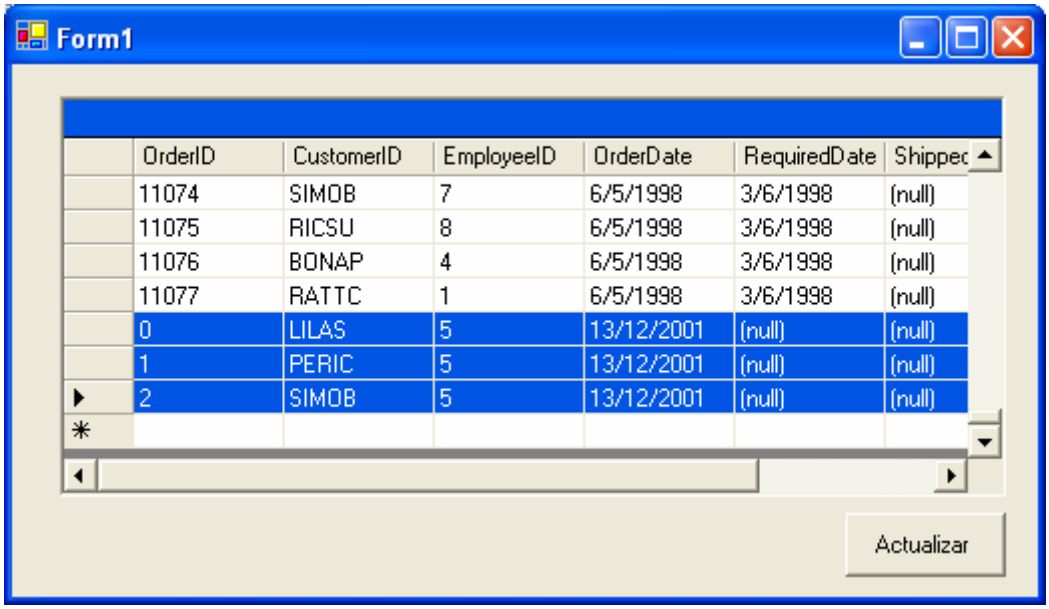

Figura 24.41. Actualización tipo *Batch* de tres filas

En este instante la información no está en la base de datos, está en memoria en el `DataSet`, de tal manera que si se cierra el formulario la información se pierde y no se actualiza en la base de datos.

Para actualizar las tres filas en un único proceso se puede utilizar un botón como el que se ha definido en este ejemplo.

```
private void buttonActualizar_Click(object sender, System.EventArgs e)
{
      if (dataSetPedidos.HasChanges())
      {
            try
            {
                  DataSet dataSetCambios = dataSetPedidos.GetChanges();
                  sqlDataAdapterPedidos.Update(dataSetCambios, "Orders");
                  dataSetPedidos.AcceptChanges();
```

```
            }
            catch (DBConcurrencyException dbError)
            {
                  // Resolver el conflicto
                  MessageBox.Show("Error:" + dbError.ToString());
            }
      }
}
```

En este ejemplo, primero se pregunta si se han realizado cambios, y si esto es así se utiliza un segundo `DataSet` para tratar solo las filas alteradas. Luego se actualizan en la base de datos dichas filas.