INDICE GENERAL

ADO.NET

ADO.NET es un conjunto de clases que exponen servicios de acceso a datos para el programador de .NET. ADO.NET ofrece abundancia de componentes para la creación de aplicaciones de uso compartido de datos distribuidas. Constituye una parte integral de .NET Framework y proporciona acceso a datos relacionales, XML y de aplicaciones. ADO.NET satisface diversas necesidades de desarrollo, como la creación de clientes de base de datos front-end y objetos empresariales de nivel medio que utilizan aplicaciones, herramientas, lenguajes o exploradores de Internet.

1. Novedades en ADO.NET

Las siguientes características son nuevas en ADO.NET.

⊟ Language-Integrated Query (LINQ)

Language-Integrated Query (LINQ) es una innovación que aporta capacidad de consulta directamente en los lenguajes de programación de .NET Framework 3.0. Las operaciones de consulta se expresan en el propio lenguaje y no como literales de cadena incrustados en el código de aplicación. LINQ se integra en varios aspectos del acceso a datos de .NET Framework.

⊟ LINQ to DataSet

LINQ to DataSet proporciona capacidad LINQ en para datos desconectados almacenados en un objeto DataSet.

⊟ LINQ to SQL

LINQ to SQL admite consultas en un modelo de objetos asignado a las estructuras de datos de una base de datos de Microsoft SQL Server sin utilizar un modelo conceptual intermedio. Cada tabla se representa mediante una clase distinta, acoplando de manera precisa el modelo de objetos al esquema de la base de datos. LINQ to SQL convierte las consultas de Language-Integrated Query del modelo de objetos a Transact-SQL y las envía a la base de datos para su ejecución. Cuando la base de datos devuelve los resultados, LINQ to SQL los vuelve a traducir a objetos.

⊟ Nuevas características en SqlClient para SQL Server 2008

La futura versión de SQL Server 2008 contiene características compatibles con el proveedor de datos de .NET Framework para SQL Server (System.Data.SqlClient).

2. Guía Básica de ADO.NET

La mayoría de las aplicaciones empresariales interactúan con datos, los cuales normalmente se almacenan en bases de datos relacionales. Los datos relacionales dependen de un esquema de la base de datos nomalizado que ha sido optimizado para el almacenamiento e integridad de los datos, y no para el desarrollo de aplicaciones. Con frecuencia, las aplicaciones utilizan modelos de datos más enriquecidos que las simples filas y columnas de un esquema relacional. Además, el conocimiento sobre la base de datos debe estar integrado en la aplicación, lo que puede dar lugar a que ésta se interrumpa cuando el esquema subyacente cambia.

Las aplicaciones empresariales frecuentemente se desacoplen del motor de la base de datos mediante su separación en capas o niveles de aplicación independientes. La lógica de acceso a los datos se limita a un nivel de abstracción de datos (DAL). No obstante, la creación y mantenimiento de un DAL sólido y

flexible puede consumir un porcentaje desproporcionado de los recursos de desarrollo de aplicaciones y costos de mantenimiento.

ADO.NET Entity Framework corrige estos problemas permitiendo a los desarrolladores de aplicaciones programar en un modelo de datos de entidad conceptual asignado de manera flexible a un esquema de almacenamiento.

⊟ Modelo de datos de entidad

Un patrón de diseño de modelado de datos común consiste en dividir el modelo de datos en tres partes:

- Un modelo conceptual que define las entidades y relaciones del sistema que se está modelando. Los modelos conceptuales se suelen utilizar para capturar los requisitos de una aplicación durante las primeras fases de un proyecto y con frecuencia se descartan a medida que el proyecto evoluciona.
- Un modelo lógico que normaliza las entidades y relaciones en tablas de base de datos relacionales con restricciones de claves externas. Muchos equipos de desarrolladores omiten la creación de un modelo conceptual y comienzan con el modelo lógico definiendo tablas, columnas y claves en la base de datos. Después, los programadores de aplicaciones escriben código que se aplica al modelo lógico escribiendo consultas SQL y llamando al procedimiento almacenado.
- Un modelo físico que especifica detalles de almacenamiento propios del motor como partición e indización. El modelo físico lo mantienen los administradores de la base de datos responsables de la seguridad, la integridad de los datos, el mantenimiento y el ajuste del rendimiento.

ADO.NET Entity Framework amplía la capacidad del modelo conceptual permitiendo a los desarrolladores escribir código que opera en objetos generados desde un modelo de datos de entidad (EDM) en lugar de hacerlo directamente en el modelo lógico (relacional). A continuación, Entity Framework asigna estas operaciones a comandos relacionales específicos de almacenamiento.

Las aplicaciones de Entity Framework ofrecen las siguientes ventajas:

- Las aplicaciones están libres de dependencias de codificación rígida de un motor de datos o de un modelo lógico concreto.
- Las asignaciones entre el modelo conceptual y el modelo lógico específico de almacenamiento pueden cambiarse sin tener que cambiar el código de la aplicación.
- Los desarrolladores pueden trabajar con un modelo de objeto de aplicación coherente que se puede asignar a varios esquemas de almacenamiento, con toda probabilidad implementados en sistemas de administración de base de datos diferentes.
- Se pueden asignar varios modelos de aplicación a un único esquema de almacenamiento.
- La compatibilidad con Language Integrated Query proporciona validación de la sintaxis en el momento de la compilación para consultas en un modelo conceptual.
- La expresividad del modelo de datos es más rica que la de una base de datos relacional tradicional, con características como el establecimiento inflexible de tipos con jerarquías de tipos, relaciones navegables y tipos complejos.

⊟ Plataforma de datos de entidad

Microsoft concibe la Plataforma de datos de entidad como una estrategia para varias versiones que reducirá la cantidad de código y mantenimiento requeridos permitiendo a los desarrolladores programar en modelos de datos de entidad conceptual. La primera versión se encuentra en la versión Visual Studio 2008 de Visual Studio y .NET Framework.

LINQ

La versión Visual Studio 2008 de Visual Studio y .NET Framework incorpora capacidades LINQ (Language Integrated Query) que integran un conjunto de operadores de consulta estándar general en la sintaxis del lenguaje de Visual C# y Visual Basic. LINQ incorpora modelos fáciles y estándar para consultar y actualizar datos, y se puede ampliar para proporcionar compatibilidad prácticamente con cualquier tipo de almacén de datos.

La versión de Visual Studio 2008 de Visual Studio y .NET Framework incluye las siguientes implementaciones de LINQ:

LINQ a DataSet

Proporciona capacidades LINQ (Language Integrated Query) para datos desconectados almacenados en DataSet.

LINQ a XML

Proporciona capacidades de modificación de documento en memoria del Modelo de objetos de documento (DOM) y es compatible con consulta LINQ.

LINQ a Objectos

Proporciona consultas LINQ para obtener acceso a estructuras de datos en memoria, como listas, matrices, diccionarios y colecciones.

LINQ a SQL

Habilita las consultas LINQ en bases de datos de Microsoft SQL Server mediante la asignación de un modelo de datos relacional a un modelo de objeto expresado en el lenguaje de programación del desarrollador.

Entity Framework

Microsoft publicará ADO.NET Entity Framework como una extensión de .NET Framework tras la salida de Visual Studio 2008.

La versión de Entity Framework incluirá los siguientes componentes:

Término	Definición
Entity Data Model (EDM)	Especifica el esquema conceptual utilizado para generar las clases programables utilizadas por el código de la aplicación. Las estructuras de almacenamiento de datos persistentes se representan en un esquema de almacenamiento, y una especificación de asignación conecta el esquema conceptual con el esquema de almacenamiento. Entity Framework utiliza EDM para definir entidades conceptuales que se pueden leer en formatos serializados mediante un lector de datos o materializadas como objetos. Los desarrolladores pueden ampliar estos objetos cuando sea necesario para la compatibilidad con diferentes necesidades de la aplicación.
Entity SQL	Define un lenguaje de consulta común basado en SQL, ampliado para expresar consultas en términos de conceptos EDM tales como tipos y relaciones navegables entre diferentes bases de datos. Entity SQL es compatible con **EntityClient** y con **Object Services**.

EntityClient	Proveedor de datos de ADO.NET que expone datos en términos de modelo de datos de entidad conceptual, consultado a través de un lenguaje **Entity SQL** común, con asignación flexible a proveedores de datos específicos de almacenamiento.
Object Services	Permite a los programadores interactuar con modelos conceptuales a través de un conjunto de clases de Common Language Runtime (CLR). Estas clases se pueden generar de manera automática desde el modelo conceptual o se pueden desarrollar de manera independiente para reflejar la estructura del modelo conceptual. Los servicios de objeto también proporcionan compatibilidad de infraestructura con Entity Framework, con servicios como administración de estados, seguimiento de cambios, resolución de identidad, relaciones de carga y navegación, propagación de cambios de objeto a modificaciones de base de datos y compatibilidad de creación de consultas para Entity SQL.
LINQ to Entities	Proporciona compatibilidad con LINQ (Language Integrated Query) para la consulta de entidades.

⊟ LINQ y datos relacionales

LINQ to SQL y LINQ to Entities son dos enfoques diferentes del trabajo con datos.

LINQ to SQL admite el desarrollo rápido de aplicaciones que consultan a las bases de datos de Microsoft SQL Server asignando directamente objetos a tablas, vistas, procedimientos almacenados y funciones con valores de tabla de SQL Server. LINQ to SQL se suministrará como parte de Visual Studio 2008 y .NET Framework.

LINQ to Entities admite consultas en un modelo de datos de entidad. LINQ to Entities se suministrará como parte de ADO.NET Entity Framework en una actualización de Visual Studio 2008 y .NET Framework.

Si está escribiendo una aplicación que requiere alguna de las características siguientes, debería considerar la posibilidad de utilizar LINQ to Entities y ADO.NET Entity Framework:

- Capacidad de definir asignación flexible en un esquema relacional existente, por ejemplo:
    - Asignar una única clase a varias tablas.
    - Asignar a diversos tipos de herencia.
    - Modelar directamente relaciones de varios a varios.
    - Exponer grupos de propiedades como un único tipo compuesto (complejo).
    - Asignar a una consulta arbitraria en la base de datos.
- Capacidad de consultar bases de datos relacionales que no sean de la familia de productos de Microsoft SQL Server.
- Capacidad de compartir un modelo entre varias aplicaciones o servicios.
- Un lenguaje de consulta basado en cadena.
- Capacidad para consultar un modelo conceptual sin materializar resultados como objetos.

Si no necesita ninguna de estas funciones, LINQ to SQL proporciona un enfoque más sencillo para aplicaciones con clases de datos asignados directamente a su base de datos de Microsoft SQL Server.

Microsoft proporcionará herramientas y directrices para ayudar a los clientes que comienzan con LINQ to SQL a migrar sus aplicaciones a LINQ to Entities cuando se requiere una asignación más flexible u obtener acceso a bases de datos que no son de Microsoft SQL Server.

Entity Framework estará disponible a través de versiones beta públicas y versiones de demostración.

3. Información General Acerca de ADO.NET

ADO.NET proporciona acceso coherente a orígenes de datos como SQL Server y XML, así como a orígenes de datos expuestos mediante OLE DB y ODBC. Las aplicaciones de consumidor que comparten datos pueden utilizar ADO.NET para conectar a estos orígenes de datos y recuperar, controlar y actualizar los datos contenidos.

ADO.NET separa el acceso a datos de la manipulación de datos y crea componentes discretos que se pueden utilizar por separado o conjuntamente. ADO.NET incluye proveedores de datos de .NET Framework para conectarse a una base de datos, ejecutar comandos y recuperar resultados. Los resultados se procesan directamente o se colocan en un objeto DataSet de ADO.NET con el fin de exponerlos al usuario para un propósito específico, combinados con datos de varios orígenes, o de pasarlos entre niveles. El objeto **DataSet** de ADO.NET también puede utilizarse independientemente de un proveedor de datos de .NET Framework para administrar datos que son locales de la aplicación o que proceden de un origen XML.

Las clases de ADO.NET se encuentran en System.Data.dll y se integran con las clases de XML incluidas en System.Xml.dll. Para obtener un ejemplo de código de ejemplo que se conecta a una base de datos, recupera datos de ésta y los muestra en la ventana de la consola.

ADO.NET proporciona funcionalidad a los programadores que escriben código administrado similar a la funcionalidad que los objetos ADO (ActiveX Data Objects) proporcionan a los programadores de modelo de objetos componentes (COM) nativo. Se recomienda utilizar ADO.NET, y no ADO, para obtener acceso a datos de aplicaciones .NET.

Declaración de privacidad: los ensamblados System.Data.dll, System.Data.Design.dll, System.Data.OracleClient.dll, System.Data.SqlXml.dll, System.Data.Linq.dll, System.Data.SqlServerCe.dll y System.Data.DataSetExtensions.dll no distinguen entre datos de usuario privados y no privados. Estos ensamblados no recopilan, almacenan o transportan datos privados del usuario. No obstante, las aplicaciones de terceros podrían recopilar, almacenar o transportar datos privados de usuario valiéndose de dichos ensamblados.

3.1. Arquitectura de ADO .NET

Tradicionalmente, el procesamiento de datos ha dependido principalmente de un modelo de dos niveles basado en una conexión. A medida que aumenta el uso que hace el procesamiento de datos de arquitecturas de varios niveles, los programadores están pasando a un enfoque sin conexión con el fin de proporcionar una mejor escalabilidad a sus aplicaciones.

⊟ Componentes de ADO.NET

Los dos componentes principales de ADO.NET 3.0 para el acceso a los datos y su manipulación son los proveedores de datos .NET Framework y DataSet.

Proveedores de datos de .NET Framework

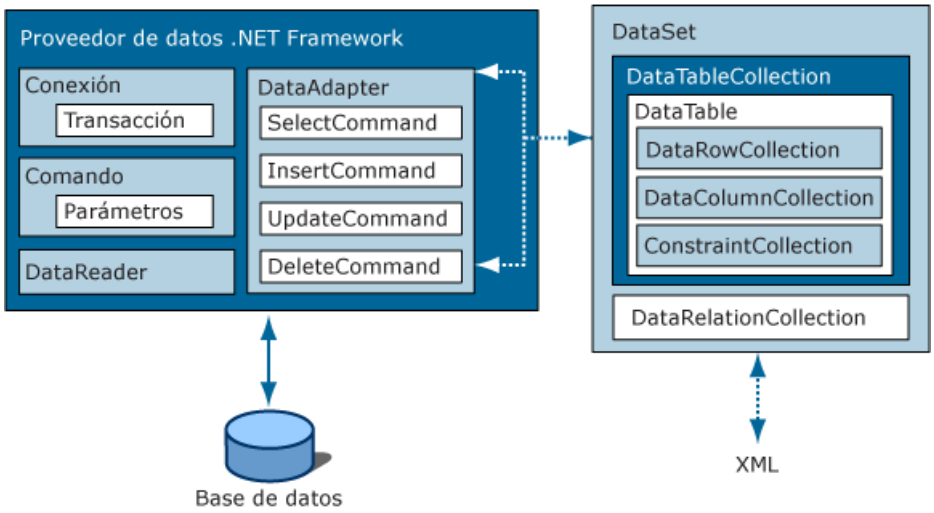
Los proveedores de datos de .NET Framework son componentes diseñados explícitamente para la manipulación de datos y el acceso rápido a datos de sólo lectura y sólo avance. El objeto **Connection** proporciona conectividad a un origen de datos. El objeto **Command** permite tener acceso a comandos de base de datos para devolver datos, modificar datos, ejecutar procedimientos almacenados y enviar o recuperar información sobre parámetros. **DataReader** proporciona una secuencia de datos de alto rendimiento desde el origen de datos. Por último, el objeto **DataAdapter** proporciona el puente entre el objeto **DataSet** y el origen de datos. **DataAdapter** utiliza objetos **Command** para ejecutar comandos SQL en el origen de datos tanto para cargar **DataSet** con datos y reconciliar en el origen de datos los cambios aplicados a los datos incluidos en el **DataSet**.

DataSet

DataSet de ADO.NETestá expresamente diseñado para el acceso a datos independientemente del origen de datos.Como resultado, se puede utilizar con múltiples y distintos orígenes de datos, con datos XML o para administrar datos locales de la aplicación. **DataSet** contiene una colección de uno o más objetos DataTable formados por filas y columnas de datos, así como información sobre claves principales, claves externas, restricciones y de relación relacionada con los datos incluidos en los objetos **DataTable**.

En el diagrama siguiente se ilustra la relación entre un proveedor de datos .NET Framework y un **DataSet**.

Arquitectura de ADO .NET



Elegir un DataReader o un DataSet

A la hora de decidir si su aplicación debe utilizar un **DataReader** o un **DataSet**, debe tener en cuenta el tipo de funcionalidad que su aplicación requiere. Use un **DataSet** para hacer lo siguiente:

- Almacene datos en la memoria caché de la aplicación para poder manipularlos. Si solamente necesita leer los resultados de una consulta, el **DataReader** es la mejor elección.
- Utilizar datos de forma remota entre un nivel y otro o desde un servicio web XML.
- Interactuar con datos dinámicamente, por ejemplo para enlazar con un control de formularios Windows Forms o para combinar y relacionar datos procedentes de varios orígenes.
- Realizar procesamientos exhaustivos de datos sin necesidad de tener una conexión abierta con el origen de datos, lo que libera la conexión para que la utilicen otros clientes.

Si no necesita la funcionalidad proporcionada por el **DataSet**, puede mejorar el rendimiento de su aplicación si utiliza el **DataReader** para devolver sus datos de sólo avance y de sólo lectura. Aunque **DataAdapter** utiliza **DataReader** para rellenar el contenido de un **DataSet**, al utilizar el **DataReader** puede mejorar el rendimiento porque no usará la memoria que utilizaría el **DataSet**, además de evitar el procesamiento necesario para crear y rellenar el contenido de **DataSet**.

⊟ XML y ADO.NET

ADO.NET aprovecha la eficacia de XML para proporcionar acceso a datos sin conexión. ADO.NET fue diseñado teniendo en cuenta las clases de XML incluidas en .NET Framework; ambos son componentes de una única arquitectura.

ADO.NET y las clases de XML incluidas en .NET Framework convergen en el objeto **DataSet**. **DataSet** se puede rellenar con datos procedentes de un origen XML, ya sea éste un archivo o una secuencia XML. **DataSet** se puede escribir como XML compatible con el consorcio World Wide Web (W3C), que incluye su esquema como esquema lenguaje de definición de esquemas XML, independientemente del origen de los datos incluidos en **DataSet**. Puesto que el formato nativo de serialización del **DataSet** es XML, es un medio excelente para mover datos de un nivel a otro, por lo que **DataSet** es idóneo para utilizar datos y contextos de esquemas de interacción remota desde y hacia un servicio web XML.

⊟ Requisitos de ADO.NET

Compatibilidad con el SDK de Microsoft .NET Framework (incluido ADO.NET) a partir de Microsoft® Windows XP, Windows 2000, Windows NT 4 con Service Pack 6a, Windows Millennium Edition, Windows 98 y Windows CE.

Nota:

El proveedor de datos de .NET Framework para OLE DB y el proveedor de datos de .NET Framework para ODBC requieren MDAC 2.6, y se recomienda MDAC 2.8 Service Pack 1 (SP1). Se pueden descargar los Service Pack más recientes en Data Access and Storage Developer Center.

Remoting or Marshaling Data Between Tiers and Clients

El diseño de **DataSet** permite transportar fácilmente datos a clientes a través de Internet mediante Servicios Web XML, así como calcular referencias de los datos entre componentes administrados mediante servicios remotos de .NET Framework. Mediante los mismos servicios, también puede utilizar de forma remota un **DataSet** con establecimiento inflexible de tipos.

Tenga en cuenta que en ADO.NET 2.0, los objetos **DataTable** también se pueden utilizar con servicios remotos y con Servicios Web XML.

⊟ LINQ to SQL

LINQ to SQL admite consultas en un modelo de objetos asignado a las estructuras de datos de una base de datos relacional sin utilizar un modelo conceptual intermedio. Cada tabla se representa mediante una clase distinta, acoplando de manera precisa el modelo de objetos al esquema de la base de datos relacional. LINQ to SQL traduce Language-integrated queries del modelo de objetos a Transact-SQL y lo envía a la base de datos para su ejecución. Cuando la base de datos devuelve los resultados, LINQ to SQL los vuelve a traducir a objetos.

3.2 Proveedores de Datos de .NET Framework

Los proveedores de datos de .NET Framework sirven para conectarse a una base de datos, ejecutar comandos y recuperar resultados. Esos resultados se procesan directamente, se colocan en un DataSet con el fin de que el usuario pueda verlos cuando los necesite, se combinan con datos de varios orígenes o se utilizan de forma remota entre niveles. Los proveedores de datos de .NET Framework son ligeros, de manera que crean un nivel mínimo entre el origen de datos y el código, con lo que aumenta el rendimiento sin sacrificar funcionalidad.

En la tabla siguiente se enumeran los proveedores de datos que se incluyen en .NET Framework.

.NET Frameworkproveedor de datos	Descripción
.NET Framework **Data Provider for** SQL Server	Proporciona acceso de datos para Microsoft SQL Server versión 7.0 o posterior.Utiliza el espacio de nombres System.Data.SqlClient.
.NET Framework **Data Provider for OLE DB**	Para orígenes de datos que se exponen mediante OLE DB.Utiliza el espacio de nombres System.Data.OleDb.
.NET Framework **Data Provider for ODBC**	Para orígenes de datos que se exponen mediante ODBC.Utiliza el espacio de nombres System.Data.Odbc.
.NET Framework **Data Provider for Oracle**	Para orígenes de datos de Oracle. El proveedor de datos de .NET Framework para Oracle es compatible con la versión 8.1.7 y posteriores del software de cliente de Oracle y utiliza el espacio de nombres System.Data.OracleClient.

Nota:

El proveedor de datos de .NET Framework para ODBC y el proveedor de datos de .NET Framework para Oracle no se incluyeron originalmente en la versión 1.0 de .NET Framework. Si necesita el proveedor de datos de .NET Framework para ODBC o el proveedor de datos de .NET Framework para Oracle y está utilizando .NET Framework 1.0, puede descargarlos del Centro para programadores de acceso a datos y almacenamiento. El espacio de nombres del proveedor de datos de .NET Framework para ODBC descargado es **Microsoft.Data.Odbc**. El espacio de nombres del proveedor de datos de .NET Framework para Oracle descargado es **System.Data.OracleClient**.

Objetos principales de los proveedores de datos de .NET Framework

En la tabla siguiente se describen los cuatro objetos centrales que constituyen un proveedor de datos de .NET Framework.

Objeto	Descripción
Connection	Establece una conexión a un origen de datos determinado. La clase base para todos los objetos **Connection** es DbConnection.
Command	Ejecuta un comando en un origen de datos. Expone **Parameters** y puede ejecutarse en el ámbito de un objeto **Transaction** desde **Connection**. La clase base para todos los objetos **Command** es DbCommand.
DataReader	Lee una secuencia de datos de sólo avance y sólo lectura desde un origen de datos. La clase base para todos los objetos **DataReader** es DbDataReader.
DataAdapter	Llena un **DataSet** y realiza las actualizaciones necesarias en el origen de datos. La clase base para todos los objetos **DataAdapter** es DbDataAdapter.

Además de las clases principales citadas en la tabla expuesta anteriormente en este documento, los proveedores de datos de .NET Framework también incluyen las que se enumeran en la tabla siguiente.

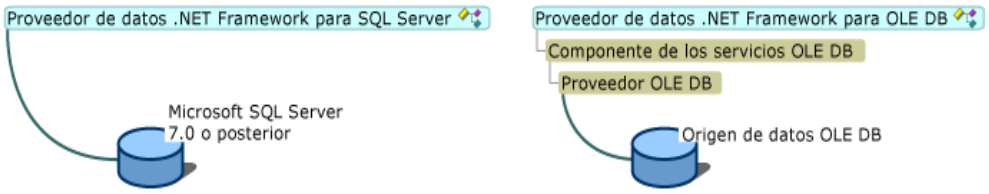
Objeto	Descripción
Transaction	Incluye comandos en las transacciones que se realizan en el origen de datos. La clase base para todos los objetos **Transaction** es DbTransaction. ADO.NET es también compatible con las transacciones que usan clases en el espacio de nombres System.Transactions.
CommandBuilder	Un objeto auxiliar que genera automáticamente las propiedades de comando de un **DataAdapter** o que obtiene de un procedimiento almacenado información acerca de parámetros con la que puede rellenar la colección **Parameters** de un objeto **Command**. La clase base para todos los objetos **CommandBuilder** es DbCommandBuilder.
ConnectionStringBuilder	Un objeto auxiliar que proporciona un modo sencillo de crear y administrar el contenido de las cadenas de conexión utilizadas por los objetos **Connection**. La clase base para todos los objetos **ConnectionStringBuilder** es DbConnectionStringBuilder.
Parameter	Define los parámetros de entrada, salida y valores devueltos para los comandos y procedimientos almacenados. La clase base para todos los objetos **Parameter** es DbParameter.
Exception	Se devuelve cuando se detecta un error en el origen de datos. En el caso de que el error se detecte en el cliente, los proveedores de datos de .NET Framework generan una excepción de .NET Framework. La clase base para todos los objetos **Exception** es DbException.
Error	Expone la información relacionada con una advertencia o error devueltos por un origen de datos.
ClientPermission	Se proporciona para los atributos de seguridad de acceso del código de los proveedores de datos de .NET Framework. La clase base para todos los objetos **ClientPermission** es DBDataPermission.

Proveedor de datos de .NET Framework para SQL Server

El proveedor de datos de .NET Framework para SQL Server usa su propio protocolo para comunicarse con SQL Server. Es ligero y presenta un buen rendimiento porque está optimizado para tener acceso a SQL Server directamente, sin agregar una capa OLE DB u ODBC. En la siguiente ilustración se compara el proveedor de datos de .NET Framework para SQL Server y el proveedor de datos de .NET Framework para OLE DB. El proveedor de datos de .NET Framework para OLE DB se comunica con un origen de datos OLE DB tanto a través del componente de servicio OLE DB, que proporciona agrupación de conexiones y servicios de transacción, como del proveedor OLE DB correspondiente al origen de datos.

Nota:
El proveedor de datos de .NET Framework para ODBC cuenta con una arquitectura similar a la del proveedor de datos de .NET Framework para OLE DB; por ejemplo, llama a un componente de servicio ODBC.

Comparación del proveedor de datos de dnprdnshort para ssNoVersionr y el proveedor de datos de dnprdnshort para OLE DB



Para utilizar el proveedor de datos de .NET Framework para SQL Server, debe tener acceso a SQL Server 7.0 o posterior. Las clases del proveedor de datos de .NET Framework para SQL Server se encuentran en el espacio de nombres System.Data.SqlClient. Para versiones anteriores de SQL Server, utilice el proveedor de datos de .NET Framework para OLE DB con el proveedor de datos OLE DB de SQL Server System.Data.OleDb.

El proveedor de datos de .NET Framework para SQL Server admite tanto transacciones locales como transacciones distribuidas. En el caso de las transacciones distribuidas, el proveedor de datos de .NET Framework para SQL Server se inscribe automáticamente y de forma predeterminada en una transacción y obtiene los detalles de la misma a través de los servicios de componentes de Windows o de System.Transactions.

En el siguiente ejemplo de código se muestra cómo puede incluir el espacio de nombres **System.Data.SqlClient** en sus aplicaciones.

```
Imports System.Data.SqlClient
```

⊟ Proveedor de datos de .NET Framework para OLE DB

El proveedor de datos de .NET Framework para OLE DB utiliza OLE DB nativo para habilitar el acceso a datos mediante la interoperabilidad COM. El proveedor de datos de .NET Framework para OLE DB admite tanto transacciones locales como transacciones distribuidas. En el caso de las transacciones distribuidas, el proveedor de datos de .NET Framework para OLE DB se inscribe automáticamente y de forma predeterminada en una transacción y obtiene los detalles de la misma a través de los servicios de componentes de Windows 2000.

En la siguiente tabla se muestran los proveedores probados con ADO.NET.

Controlador	Proveedor
SQLOLEDB	Proveedor Microsoft OLE DB para SQL Server
MSDAORA	Proveedor Microsoft OLE DB para Oracle
Microsoft.Jet.OLEDB.4.0	Proveedor OLE DB para Microsoft Jet

Nota:

No se recomienda el uso de una base de datos de Access (Jet) como origen de datos de aplicaciones multiproceso, como las aplicaciones ASP.NET. Si debe utilizar Jet como origen de datos para una aplicación ASP.NET, ha de ser consciente de que las aplicaciones ASP.NET que se conectan a una base de datos Access pueden tener problemas de conexión. Para obtener ayuda a la hora de solucionar los problemas de conexión que puedan presentarse al utilizar ASP.NET y una base de datos Access (Jet), vea el artículo Q316675, "PRB: No se puede conectar a base de datos de Access desde ASP.NET" en Microsoft Knowledge Base.

El proveedor de datos de .NET Framework para OLE DB no admite las interfaces de la versión 2.5 de OLE DB. Los proveedores OLE DB que requieren compatibilidad con interfaces de la versión 2.5 de OLE DB no

funcionarán correctamente con el proveedor de datos de .NET Framework para OLE DB. Entre ellos se incluye el proveedor Microsoft OLE DB para Exchange y el proveedor Microsoft OLE DB para la publicación en Internet.

El proveedor de datos de .NET Framework para OLE DB no funciona con el proveedor OLE DB para ODBC (MSDASQL). Para tener acceso a un origen de datos ODBC mediante ADO.NET, utilice el proveedor de datos de .NET Framework para ODBC.

Las clases del proveedor de datos de .NET Framework para OLE DB se encuentran en el espacio de nombres System.Data.OleDb. En el siguiente ejemplo de código se muestra cómo puede incluir el espacio de nombres **System.Data.OleDb** en sus aplicaciones.

```
Imports System.Data.OleDb
```

Nota:

El proveedor de datos de .NET Framework para OLE DB necesita MDAC 2.6 o una versión superior y se recomienda MDAC 2.8 Service Pack 1 (SP1). Puede descargar MDAC 2.8 SP1 del Centro para programadores de acceso a datos y almacenamiento.

Proveedor de datos de .NET Framework para ODBC

El proveedor de datos de .NET Framework para ODBC utiliza el Administrador de controladores ODBC nativos para habilitar el acceso a datos.El proveedor de datos de ODBC admite tanto transacciones locales como transacciones distribuidas. En el caso de las transacciones distribuidas, el proveedor de datos de ODBC se inscribe automáticamente y de forma predeterminada en una transacción y obtiene los detalles de la misma a través de los servicios de componentes de Windows 2000.

En la siguiente tabla se muestran los controladores ODBC que se han probado con ADO.NET.

Controlador
SQL Server
Microsoft ODBC para Oracle
Microsoft Access Driver (*.mdb)

Las clases del proveedor de datos de .NET Framework para ODBC se encuentran en el espacio de nombres System.Data.Odbc.

En el siguiente ejemplo de código se muestra cómo puede incluir el espacio de nombres **System.Data.Odbc** en sus aplicaciones.

```
Imports System.Data.Odbc
```

Nota:

El proveedor de datos de .NET Framework para ODBC necesita MDAC 2.6 o posterior, y se recomienda MDAC 2.8 SP1. Puede descargar MDAC 2.8 SP1 del Centro para programadores de acceso a datos y almacenamiento.

Proveedor de datos de .NET Framework para Oracle

El proveedor de datos de .NET Framework para Oracle habilita el acceso a datos de orígenes de datos de Oracle mediante el software de conectividad de cliente de Oracle. El proveedor de datos es compatible

con la versión 8.1.7 o posterior del software de cliente de Oracle.El proveedor de datos admite tanto transacciones locales como transacciones distribuidas.

El proveedor de datos de .NET Framework para Oracle requiere que el software de cliente de Oracle (versión 8.1.7 o posterior) esté instalado en el sistema antes de conectarse a un origen de datos de Oracle.

Las clases del proveedor de datos de .NET Framework para Oracle se encuentran en el espacio de nombres System.Data.OracleClient y están incluidas en el ensamblado **System.Data.OracleClient.dll**. Al compilar una aplicación que utiliza el proveedor de datos, debe hacer referencia tanto a **System.Data.dll** como a **System.Data.OracleClient.dll**.

En el siguiente ejemplo de código se muestra cómo puede incluir el espacio de nombres **System.Data.OracleClient** en sus aplicaciones.

```
Imports System.Data
Imports System.Data.OracleClient
```

⊟ Elegir un proveedor de datos de .NET Framework

En función del diseño y del origen de datos de su aplicación, su elección del proveedor de datos de .NET Framework puede mejorar el rendimiento, la funcionalidad y la integridad de su aplicación. En la siguiente tabla se describen las ventajas y las limitaciones de cada proveedor de datos de .NET Framework.

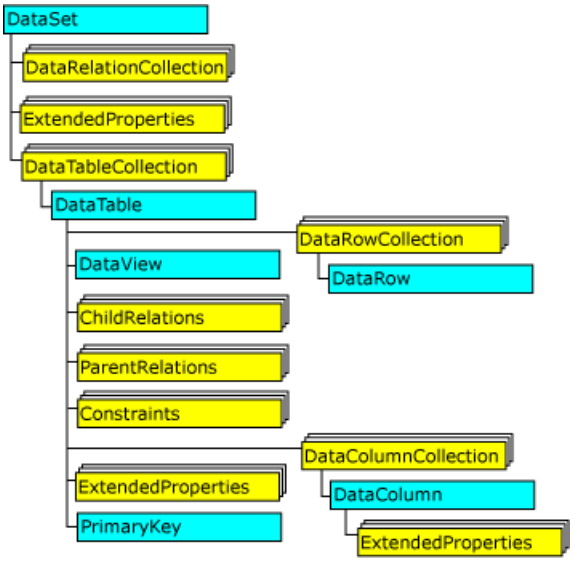
Proveedor	Notas
.NET Framework **Data Provider for SQL Server**	Recomendado para aplicaciones de nivel medio que utilizan Microsoft SQL Server 7.0 o posterior. Recomendado para aplicaciones de un único nivel que utilizan Microsoft Database Engine (MSDE) o SQL Server 7.0 o una versión posterior. Recomendado en lugar de utilizar el proveedor OLE DB para SQL Server (SQLOLEDB) con el proveedor de datos de .NET Framework para OLE DB. Para SQL Server 6.5 y versiones anteriores, debe utilizar el proveedor OLE DB para SQL Server con el proveedor de datos de .NET Framework para OLE DB.
.NET Framework **Data Provider for OLE DB**	Recomendado para aplicaciones de nivel medio que utilizan SQL Server 6.5 o anterior. Para SQL Server 7.0 o versiones posteriores, se recomienda el proveedor de datos de .NET Framework para SQL Server. También se recomienda para aplicaciones de un único nivel que utilizan bases de datos de Microsoft Access.No se recomienda el uso de bases de datos Access para una aplicación de nivel medio.
.NET Framework **Data Provider for ODBC**	Recomendado para aplicaciones de un único nivel y de nivel medio que utilizan orígenes de datos de ODBC.
.NET Framework **Data Provider for Oracle**	Recomendado para aplicaciones de un único nivel y de nivel medio que utilizan orígenes de datos de Oracle.

3.3. DataSets

El objeto DataSet es esencial para la compatibilidad con escenarios de datos distribuidos desconectados con ADO.NET. El objeto **DataSet** es una representación residente en memoria de datos que proporciona un modelo de programación relacional coherente independientemente del origen de datos.Se puede utilizar con muchos y distintos orígenes de datos, con datos XML o para administrar datos locales de la

aplicación. El **DataSet** representa un conjunto completo de datos que incluye tablas relacionadas y restricciones, así como relaciones entre las tablas. En la siguiente ilustración se muestra el modelo de objetos **DataSet**.

Modelo de objetos DataSet



Los métodos y objetos de un **DataSet** concuerdan con los del modelo de base de datos relacional.

El objeto **DataSet** también puede mantener y recargar su contenido como XML y su esquema como esquema de lenguaje de definición de esquemas XML (XSD).

⊟ DataTableCollection

Un objeto **DataSet** de ADO.NET contiene una colección de cero o más tablas representadas por objetos DataTable. El DataTableCollection contiene todos los objetos **DataTable** de un **DataSet**.

El **DataTable** se define en el espacio de nombres System.Data y representa una única tabla de datos residentes en memoria. Contiene una colección de columnas representadas por una DataColumnCollection, así como restricciones representadas por una ConstraintCollection, que juntas definen el esquema de la tabla. Una **DataTable** también contiene una colección de filas representadas por la DataRowCollection, que contiene los datos de la tabla. Una DataRow conserva, junto con su estado actual, sus versiones actual y original para identificar los cambios en los valores almacenados en la fila.

⊟ DataRelationCollection

Un **DataSet** contiene relaciones en su objeto DataRelationCollection. Una relación, representada por el objeto DataRelation, asocia las filas de una **DataTable** con las filas de otra **DataTable**. Las relaciones son análogas a las rutas de acceso de unión que podrían existir entre columnas de claves principales y externas en una base de datos relacional. Una **DataRelation** identifica las columnas coincidentes en dos tablas de un **DataSet**.

Las relaciones habilitan la navegación entre tablas de un objeto **DataSet**.Los elementos esenciales de una **DataRelation** son el nombre de la relación, el nombre de las tablas que se relacionan y las columnas relacionadas de cada tabla. Es posible crear relaciones con más de una columna por tabla si se especifica una matriz de objetos DataColumn como columnas de claves. Cuando agrega una relación al DataRelationCollection, puede agregar opcionalmente una **UniqueKeyConstraint** y una **ForeignKeyConstraint** para exigir restricciones de integridad cuando se realizan cambios en valores de columna relacionados.

⊟ ExtendedProperties

DataSet, **DataTable** y **DataColumn** tienen todos una propiedad **ExtendedProperties**. **ExtendedProperties** es una **PropertyCollection** en la que puede colocar información personalizada, como la instrucción SELECT que se ha utilizado para generar el conjunto de resultados o la hora en que se generaron los datos. La colección **ExtendedProperties** se mantiene con la información de esquema del **DataSet**.

⊟ LINQ to DataSet

LINQ to DataSet proporciona capacidades Language-Integrated Query para los datos desconectados almacenados en un objeto DataSet. LINQ to DataSet utiliza sintaxis estándar de LINQ y proporciona comprobación de sintaxis en tiempo de compilación, tipos estáticos y compatibilidad con IntelliSense cuando se utiliza la IDE de Visual Studio.

3.4. Ejecución en Paralelo

La ejecución en paralelo en .NET Framework es la posibilidad de ejecutar una aplicación en un equipo que tiene instaladas varias versiones de .NET Framework, utilizando exclusivamente la versión para la que se ha compilado la aplicación.

Una aplicación compilada mediante una versión de .NET Framework se puede ejecutar en otra versión distinta de .NET Framework. No obstante, se recomienda compilar una versión de la aplicación por cada versión instalada de .NET Framework y ejecutarlas por separado. En cualquier caso, debe tener en cuenta los cambios entre las diferentes versiones de ADO.NET que pueden afectar a la compatibilidad con versiones posteriores o anteriores de la aplicación.

⊟ Compatibilidad con versiones anteriores y posteriores

La compatibilidad con versiones posteriores implica que una aplicación se puede compilar con una versión anterior de .NET Framework, aunque se podrá ejecutar asimismo correctamente en una versión posterior de .NET Framework. El código de ADO.NET escrito para .NET Framework versión 1.1 es compatible con la versión posterior 2.0 de .NET Framework.

La compatibilidad con versiones anteriores significa que una aplicación compilada para una versión posterior de .NET Framework se ejecuta también en versiones anteriores de .NET Framework, sin que su funcionalidad se vea afectada. Por supuesto, no será así en el caso de las características incluidas en una versión nueva de .NET Framework.

Aunque los componentes de ADO.NET en .NET Framework están diseñados para ser compatibles tanto con versiones posteriores y anteriores (a excepción de las nuevas características), debe tener en cuenta varias cuestiones que afectan a la compatibilidad con las versiones posteriores o anteriores de una aplicación.

En las siguientes secciones se describen algunos problemas relacionados con la ejecución en paralelo que pueden afectar a la compatibilidad con las versiones anteriores o posteriores del código de ADO.NET:

- El proveedor de datos de .NET Framework para ODBC
- El proveedor de datos de .NET Framework para Oracle
- Seguridad de acceso del código
- El DataSet
- Ejecución de SqlCommand
- Microsoft Data Access Components (MDAC)

⊟ Proveedor de datos de .NET Framework para ODBC

A partir de la versión 1.1, el proveedor de datos de .NET Framework para ODBC (System.Data.Odbc) se incluye como parte de .NET Framework. El proveedor de datos de ODBC está disponible para los programadores de la versión 1.0 de .NET Framework mediante descarga en el sitio web del Centro para programadores de acceso a datos y almacenamiento. El espacio de nombres del proveedor de datos de .NET Framework para ODBC descargado es **Microsoft.Data.Odbc**.

Si ha programado una aplicación para la versión 1.0 de .NET Framework que usa el proveedor de datos de ODBC para conectarse al origen de datos y desea ejecutar dicha aplicación en .NET Framework versión 1.1 o posteriores, debe actualizar el espacio de nombres del proveedor de datos de ODBC a **System.Data.Odbc**. A continuación, debe compilarla de nuevo para la nueva versión de .NET Framework.

Si ha programado una aplicación para la versión 2.0 de .NET Framework que usa el proveedor de datos de ODBC para conectarse al origen de datos y desea ejecutar dicha aplicación en la versión 1.0 de .NET Framework, debe descargar el proveedor de datos de ODBC e instalarlo en el sistema de la versión 1.0 de .NET Framework. A continuación debe cambiar el espacio de nombres del proveedor de datos de ODBC a **Microsoft.Data.Odbc** y volver a compilar la aplicación para la versión 1.0 de .NET Framework.

⊟ Proveedor de datos de .NET Framework para Oracle

A partir de la versión 1.1, el proveedor de datos de .NET Framework para Oracle (System.Data.OracleClient) se incluye como parte de .NET Framework. El proveedor de datos está disponible para los programadores de la versión 1.0 de .NET Framework mediante descarga desde el sitio web del Centro para programadores de acceso a datos y almacenamiento.

Si ha programado una aplicación para la versión 2.0 de .NET Framework que usa el proveedor de datos para conectarse al origen de datos y desea ejecutar esta aplicación en la versión 1.0 de .NET Framework, debe descargar el proveedor de datos e instalarlo en el sistema de la versión 1.0 de .NET Framework.

⊟ Seguridad de acceso del código

En la versión 1.0 de .NET Framework, se requiere que los proveedores de datos de .NET Framework (System.Data.SqlClient, System.Data.OleDb) se ejecuten con un permiso FullTrust. Si se intentan utilizar los proveedores de datos de .NET Framework con la versión 1.0 de .NET Framework en una zona con un permiso inferior a FullTrust, se produce una excepción SecurityException.

Sin embargo, en la versión 2.0 de .NET Framework se pueden usar todos los proveedores de datos de .NET Framework en zonas de confianza parcial. Además, se ha agregado una nueva característica de seguridad a los proveedores de datos de .NET Framework en .NET Framework 1.1.Esta característica le permite restringir las cadenas de conexión que se pueden utilizar en una zona de seguridad determinada. Es posible también deshabilitar el uso de contraseñas en blanco para una zona de seguridad determinada.

Debido a que cada instalación de .NET Framework tiene un archivo Security.config independiente, no existen problemas de compatibilidad con la configuración de la seguridad. Sin embargo, si la aplicación depende de las capacidades de seguridad adicionales de ADO.NET incluidas en la versión 1.1 de .NET Framework y posteriores, no podrá distribuir la aplicación a un sistema de la versión 1.0.

⊟ Ejecución de SqlCommand

A partir de la versión 1.1 de .NET Framework, se ha cambiado la forma en que ExecuteReader ejecuta los comandos en el origen de datos.

En la versión 1.0 de .NET Framework, ExecuteReader ejecuta todos los comandos en el contexto del procedimiento almacenado **sp_executesql**.Como resultado, los comandos que influyen en el estado de la conexión (Activar NOCOUNT, por ejemplo) sólo se aplican a la ejecución del comando actual. Mientras la conexión permanece abierta, los comandos que se ejecutan posteriormente no modifican el estado de la conexión.

En la versión 1.1 de .NET Framework y posteriores, ExecuteReader sólo ejecuta un comando en el contexto del procedimiento almacenado **sp_executesql** si dicho comando contiene parámetros, ya que de esta forma mejora el rendimiento.Como consecuencia, si un comando que influye en el estado de la conexión se incluye en un comando sin parámetros, modifica el estado de la conexión de todos los comandos posteriores que se ejecuten mientras la conexión esté abierta.

Tomemos como ejemplo el siguiente lote de comandos, que se ejecuta en una llamada a ExecuteReader.

```
SET NOCOUNT ON;
SELECT * FROM dbo.Customers;
```

En la versión 1.1 de .NET Framework y posteriores, NOCOUNT permanecerá en ON para los comandos que se ejecuten posteriormente, mientras la conexión esté abierta. En el caso de la versión 1.0 de .NET Framework, NOCOUNT sólo permanece en ON para la ejecución actual de comandos.

Este cambio puede afectar a la compatibilidad con versiones posteriores y anteriores de la aplicación si depende del comportamiento de ExecuteReader para cualquiera de las versiones de .NET Framework.

Para las aplicaciones que se ejecutan en versiones anteriores y posteriores de .NET Framework, puede escribir el código para asegurarse de que el comportamiento sea el mismo independientemente de la versión que se esté ejecutando. Si desea asegurarse de que un comando modifica el estado de la conexión para todos los comandos que se ejecuten posteriormente, se recomienda ejecutar el comando usando ExecuteNonQuery. Si desea asegurarse de que un comando no modifica el estado de la conexión para todos los comandos que se ejecuten posteriormente, se recomienda incluir los comandos para restablecer el estado de la conexión en el comando.Por ejemplo:

```
SET NOCOUNT ON;
SELECT * FROM dbo.Customers;
SET NOCOUNT OFF;
```

Microsoft SQL Server Native Client (SQL Native Client)

Microsoft SQL Server Native Client (SQL Native Client) combina el proveedor OLE DB de SQL y el controlador ODBC de SQL en una biblioteca de vínculos dinámicos (DLL), compatible con las aplicaciones que usan distintas API de código nativo (ODBC, OLE DBE y ADO) en Microsoft SQL Server. Debe utilizar SQL Native Client en lugar de Microsoft Data Access Components (MDAC) para crear nuevas aplicaciones o mejorar las aplicaciones existentes que necesitan aprovechar las nuevas características de SQL Server 2005 como los conjuntos de resultados activos múltiples (MARS), las notificaciones de consulta, los tipos definidos por el usuario (UDT) y la compatibilidad con tipos de datos XML.

⊟ Microsoft Data Access Components (MDAC)

Los proveedores de datos de .NET Framework para OLE DB y ODBC necesitan MDAC 2.6 o posterior en todas las versiones de .NET Framework. Se recomienda la versión MDAC 2.8 SP1. Si bien esto no presenta problemas en la ejecución en paralelo, es importante tener en cuenta que, en la actualidad, MDAC no admite la ejecución en paralelo. Por lo tanto, es importante comprobar que la aplicación

seguirá funcionando correctamente con la nueva versión antes de actualizar los componentes de MDAC de la instalación.

3.5. Código de Ejemplo

El siguiente listado de códigos muestra cómo recuperar datos de la base de datos mediante el proveedor de datos de .NET Framework para SQL Server (**System.Data.SqlClient**), el proveedor de datos de .NET Framework para OLE DB (**System.Data.OleDb**), el proveedor de datos de .NET Framework para ODBC (**System.Data.Odbc**) y el proveedor de datos de .NET Framework para Oracle (**System.Data.OracleClient**). Los datos se devuelven en **DataReader**.

⊟ SqlClient

En el código de este ejemplo se asume que puede conectarse a la base de datos de ejemplo **Northwind** en Microsoft SQL Server 7.0 o en una versión posterior. El código devuelve una lista de registros de la tabla**Categories** utilizando SqlDataReader.

```
Imports System
Imports System.Data
Imports System.Data.SqlClient

Public Class Program
    Public Shared Sub Main()
        Dim connectionString As String = GetConnectionString()
        Dim queryString As String = _
         "SELECT CategoryID, CategoryName FROM dbo.Categories;"
        Using connection As New SqlConnection(connectionString)
            Dim command As SqlCommand = connection.CreateCommand()
            command.CommandText = queryString
            Try
                connection.Open()
                Dim dataReader As SqlDataReader = _
                 command.ExecuteReader()
                Do While dataReader.Read()
                    Console.WriteLine(vbTab & "{0}" & vbTab & "{1}", _
                     dataReader(0), dataReader(1))
                Loop
                dataReader.Close()
            Catch ex As Exception
                Console.WriteLine(ex.Message)
            End Try
        End Using
    End Sub
    Private Shared Function GetConnectionString() As String
        ' To avoid storing the connection string in your code,
        ' you can retrieve it from a configuration file.
        Return "Data Source=(local);Initial Catalog=Northwind;" _
           & "Integrated Security=SSPI;"
    End Function
End Class
```

OleDb

En el código de este ejemplo se asume que puede conectarse a la base de datos de ejemplo **Northwind** de Microsoft Access. El código devuelve una lista de registros de la tabla Categories utilizando OleDbDataReader.

```
Imports System
Imports System.Data
Imports System.Data.OleDb

Public Class Program
    Public Shared Sub Main()
        Dim connectionString As String = GetConnectionString()
        Dim queryString As String = _
            "SELECT CategoryID, CategoryName FROM Categories;"
        Using connection As New OleDbConnection(connectionString)
            Dim command As OleDbCommand = connection.CreateCommand()
            command.CommandText = queryString
            Try
                connection.Open()
                Dim dataReader As OleDbDataReader = _
                 command.ExecuteReader()
                Do While dataReader.Read()
                    Console.WriteLine(vbTab & "{0}" & vbTab & "{1}", _
                     dataReader(0), dataReader(1))
                Loop
                dataReader.Close()
            Catch ex As Exception
                Console.WriteLine(ex.Message)
            End Try
        End Using
    End Sub
    Private Shared Function GetConnectionString() As String
        ' To avoid storing the connection string in your code,
        ' you can retrieve it from a configuration file.
        ' Assumes Northwind.mdb is located in c:\Data folder.
        Return "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" _
           & "c:\Data\Northwind.mdb;User Id=admin;Password=;"
    End Function
End Class
```

Odbc

En el código de este ejemplo se asume que puede conectarse a la base de datos de ejemplo **Northwind** de Microsoft Access. El código devuelve una lista de registros de la tabla Categories utilizando OdbcDataReader.

```
Imports System
Imports System.Data
Imports System.Data.Odbc
```

```
Public Class Program
    Public Shared Sub Main()
        Dim connectionString As String = GetConnectionString()
        Dim queryString As String = _
            "SELECT CategoryID, CategoryName FROM Categories;"
        Using connection As New OdbcConnection(connectionString)
            Dim command As OdbcCommand = connection.CreateCommand()
            command.CommandText = queryString
            Try
                connection.Open()
                Dim dataReader As OdbcDataReader = _
                 command.ExecuteReader()
                Do While dataReader.Read()
                    Console.WriteLine(vbTab & "{0}" & vbTab & "{1}", _
                     dataReader(0), dataReader(1))
                Loop
                dataReader.Close()
            Catch ex As Exception
                Console.WriteLine(ex.Message)
            End Try
        End Using
    End Sub
    Private Shared Function GetConnectionString() As String
        ' To avoid storing the connection string in your code,
        ' you can retrieve it from a configuration file.
        ' Assumes Northwind.mdb is located in c:\Data folder.
        Return "Driver={Microsoft Access Driver (*.mdb)};" _
           & "Dbq=c:\Data\Northwind.mdb;Uid=Admin;Pwd=;"
    End Function
End Class
```

⊟ OracleClient

En el código de este ejemplo se presupone una conexión a DEMO.CUSTOMER en un servidor Oracle. También debe agregarse una referencia a System.Data.OracleClient.dll. El código devuelve los datos en OracleDataReader.

```
Imports System
Imports System.Data
Imports System.Data.OracleClient

Public Class Program
    Public Shared Sub Main()
        Dim connectionString As String = GetConnectionString()
        Dim queryString As String = _
            "SELECT CUSTOMER_ID, NAME FROM DEMO.CUSTOMER"
        Using connection As New OracleConnection(connectionString)
            Dim command As OracleCommand = connection.CreateCommand()
            command.CommandText = queryString
            Try
                connection.Open()
```

```
                Dim dataReader As OracleDataReader = _
                 command.ExecuteReader()
                Do While dataReader.Read()
                    Console.WriteLine(vbTab & "{0}" & vbTab & "{1}", _
                     dataReader(0), dataReader(1))
                Loop
                dataReader.Close()
            Catch ex As Exception
                Console.WriteLine(ex.Message)
            End Try
        End Using
    End Sub
    Private Shared Function GetConnectionString() As String
        ' To avoid storing the connection string in your code,
        ' you can retrieve it from a configuration file.
        Return "Data Source=ThisOracleServer;Integrated Security=yes;"
    End Function
End Class
```

4. Proteger Aplicaciones de ADO.NET

Para escribir una aplicación de ADO.NET segura es necesario algo más que evitar los errores de codificación más comunes, como no validar los datos proporcionados por el usuario. Una aplicación que tiene acceso a datos tiene muchos puntos débiles potenciales que un agresor puede aprovechar para obtener, manipular o destruir datos confidenciales. Por eso es importante comprender todos los aspectos de la seguridad, desde el proceso de modelo de amenazas durante la fase de diseño de su aplicación hasta la implementación y el posterior mantenimiento.

.NET Framework ofrece muchas clases, servicios y herramientas que resultan muy útiles para proteger y administrar aplicaciones de base de datos. El Common Language Runtime (CLR) proporciona un entorno de seguridad de tipos en el que ejecutar el código, junto con Code Access Security (CAS) para restringir aún más los permisos del código administrado. Si se siguen las recomendaciones de codificación del acceso seguro a datos, se reduce el daño que podría provocar un posible agresor.

El código seguro no protege de los vacíos de seguridad que provoca el propio usuario cuando trabaja con recursos no administrados como bases de datos. La mayoría de las bases de datos, como SQL Server, tienen sus propios sistemas de seguridad, que contribuyen a mejorarla cuando se implementan correctamente. Sin embargo, incluso un origen de datos con un robusto sistema de seguridad puede sufrir un ataque si no se ha configurado correctamente.

4.1. Información General de Seguridad

La protección de una aplicación es un proceso continuo. Es prácticamente imposible que se llegue a un momento en el que un programador pueda garantizar que una aplicación es invulnerable ante todo tipo de ataques, ya que no es posible predecir los tipos de ataques que surgirán en un futuro con las nuevas tecnologías. Al contrario, el hecho de que nadie haya detectado (o publicado) aún brechas de seguridad en un sistema, no quiere decir que éstas no existan o no puedan existir en un futuro. Por lo tanto, es necesario planificar la seguridad durante la fase de diseño del proyecto, así como la forma en que se mantendrá la seguridad durante el ciclo de vida de la aplicación.

⊟ Diseñar la seguridad

Uno de los mayores problemas de la programación de aplicaciones seguras es el hecho de que a menudo la seguridad no se planifica con antelación, sino que se implementa cuando el código de un proyecto se ha completado. Esta práctica genera aplicaciones poco seguras, ya que no se ha dedicado suficiente tiempo a entender qué hace que una aplicación sea segura.

La implementación de la seguridad a última hora puede producir nuevos errores, ya que el software falla debido a las nuevas restricciones o hay que volver a escribirlo para albergar funcionalidad no planificada inicialmente. Cada línea de código revisado incluye la posibilidad de insertar un error nuevo. En ese sentido, debe tener en cuenta la seguridad al principio del proceso de programación de manera que se desarrolle junto con la programación de nuevas características.

Modelo de amenazas

No se puede proteger un sistema de ataques a menos que comprenda todos los posibles ataques a los que está expuesto. El proceso de evaluación de amenazas a la seguridad, llamado modelo de amenazas, es necesario para determinar la probabilidad y las ramificaciones de las infracciones de seguridad en la aplicación ADO.NET.

El modelo de amenazas está compuesto de tres pasos principales: comprender la visión del adversario, caracterizar la seguridad del sistema y determinar las amenazas.

El modelo de amenazas es un enfoque repetitivo en la evaluación de las vulnerabilidades de la aplicación, que se usa para detectar las vulnerabilidades más peligrosas, ya que éstas exponen los datos más confidenciales. Una vez identificadas, las vulnerabilidades se clasifican en función de su gravedad y se crea un conjunto con prioridades de medidas para contrarrestar las amenazas.

⊟ Principio de los privilegios mínimos

Cuando diseñe, compile e implemente la aplicación, debe asumir que ésta será objeto de ataques. Estos ataques suelen proceder de código malintencionado que se activa con los permisos del usuario que ejecuta el código. Otros pueden provenir de código no dañino, cuyas vulnerabilidades aprovecha un atacante. Cuando se planifica la seguridad, siempre hay que asumir que se puede producir la peor situación posible.

Una medida para contrarrestar los ataques consiste en intentar establecer tantos muros en el código como sea posible mediante la ejecución con los privilegios mínimos. El principio de los privilegios mínimos indica que se deben conceder privilegios para la menor cantidad de código posible durante el tiempo mínimo necesario para conseguir que se realice el trabajo.

El procedimiento recomendado para crear aplicaciones seguras consiste en comenzar sin ningún permiso y ir agregando los mínimos permisos necesarios para la tarea concreta que se lleva a cabo. El enfoque opuesto, es decir, comenzar con todos los permisos e ir denegando permisos posteriormente de forma individual, proporciona aplicaciones poco seguras, difíciles de probar y mantener, ya que pueden existir vacíos en la seguridad causados por concesiones no intencionadas de más permisos de los que son necesarios.

⊟ Seguridad de acceso del código (CAS)

La seguridad de acceso del código (CAS) es un mecanismo que ayuda a limitar el acceso del código a recursos y operaciones protegidos. En .NET Framework, CAS realiza las funciones siguientes:

- Define permisos y conjuntos de permisos que representan el derecho de acceso a varios recursos del sistema.

- Permite a los administradores configurar la directiva de seguridad mediante la asociación de conjuntos de permisos a grupos de código.
- Permite que el código solicite los permisos que necesita para ejecutarse, así como los permisos que sería útil tener, y especifica los permisos que nunca debe tener el código.
- Concede permisos a cada ensamblado que se carga, basándose en los permisos solicitados por el código y en las operaciones permitidas por la directiva de seguridad.
- Permite que el código exija que sus llamadores tengan permisos específicos.
- Permite que el código exija que sus llamadores posean una firma digital, por lo que sólo los llamadores de una organización o un sitio concretos pueden llamar al código protegido.
- Impone restricciones en el código en tiempo de ejecución mediante la comparación de los permisos concedidos a cada llamador en la pila de llamadas con los permisos que deben poseer.

Para reducir los daños que se pueden producir cuando un ataque tiene éxito, elija un contexto de seguridad para el código de forma que se conceda acceso única y exclusivamente a los recursos necesarios para realizar el trabajo.

⊟ Seguridad de la base de datos

El principio de privilegios mínimos también se aplica al origen de los datos. A continuación se citan algunas instrucciones generales para la seguridad de base de datos:

- Crear cuentas con los privilegios mínimos posibles.
- No permitir que los usuarios obtengan acceso a cuentas administrativas tan sólo para que el código funcione.
- No devolver mensajes de error de servidor a las aplicaciones cliente.
- Validar todas las entradas, tanto en el cliente como en el servidor.
- Usar comandos con parámetros y evitar instrucciones SQL dinámicas.
- Habilitar el registro y la auditoría de seguridad en la base de datos que se utiliza, de forma que se reciba una alerta en caso de infracciones de seguridad.

⊟ Directiva de seguridad y administración

La administración inadecuada de la directiva de seguridad de acceso del código (CAS) puede crear puntos débiles en la seguridad. Cuando se utiliza una aplicación, deben seguirse las técnicas de supervisión de seguridad y deben evaluarse los riesgos a medida que surgen nuevas amenazas.

4.2. Acceso Seguro a Datos

Para escribir código de ADO.NET seguro, debe comprender los mecanismos de seguridad disponibles en el almacén de datos subyacente o en la base de datos. También debe tener en cuenta las implicaciones en la seguridad de otras características o componentes que pudieran incluirse en la aplicación.

⊟ Autenticación, autorización y permisos

Si se conecta a Microsoft SQL Server, puede usar la autenticación de Windows, también conocida como seguridad integrada, que emplea la identidad del usuario de Windows activo en lugar de pasar un identificador de usuario y una contraseña. Se recomienda encarecidamente usar la autenticación de Windows, ya que las credenciales del usuario no están expuestas en la cadena de conexión. Si no puede usar la autenticación de Windows para conectarse a SQL Server, puede crear cadenas de conexión en tiempo de ejecución mediante SqlConnectionStringBuilder.

Las credenciales utilizados para la autenticación deben tratarse de manera diferente dependiendo del tipo de aplicación. Por ejemplo, en una aplicación de formularios Windows Forms, se puede solicitar al usuario que proporcione información de autenticación o bien, se pueden utilizar las credenciales de Windows del usuario. Sin embargo, una aplicación web normalmente obtiene acceso a los datos mediante credenciales proporcionadas por la misma aplicación y no por el usuario.

Una vez que los usuarios se han autenticado, el ámbito de sus acciones depende de los permisos que se les hayan concedido. Siga siempre el principio de los privilegios mínimos y conceda sólo los permisos absolutamente necesarios.

⊟ Comandos con parámetros e inyección de código SQL

La utilización de comandos con parámetros ayuda en la protección contra ataques por inyección de código SQL, en los que un atacante "inyecta" un comando en una instrucción SQL que pone en peligro la seguridad del servidor. Los comandos con parámetros protegen de ataques por inyección de código SQL ya que garantizan que los valores recibidos desde un origen externo pasan sólo como valores y no como parte de la instrucción de Transact-SQL. Como resultado, los comandos Transact-SQL insertados en un valor no se ejecutan en el origen de datos. En cambio, se evalúan únicamente como un valor de parámetro. Además de las ventajas en el aspecto de la seguridad, los comandos con parámetros proporcionan un método práctico para organizar los valores que se pasan con una instrucción de Transact-SQL a un procedimiento almacenado.

⊟ Ataques mediante scripts

Los ataques mediante scripts constituyen otra forma de inyección que usa caracteres malintencionados insertados en una página web. El explorador no valida los caracteres insertados y los procesa como parte de la página.

⊟ Ataques mediante sondeo

Es muy frecuente que los atacantes utilicen la información de una excepción, como el nombre del servidor, de la base de datos o de una tabla, para llevar a cabo un ataque contra el sistema. Como las excepciones pueden contener información específica sobre la aplicación o el origen de datos, puede mejorar la protección de la aplicación y del origen de datos exponiendo únicamente la información necesaria al cliente.

⊟ Proteger orígenes de datos de Access y Excel de Microsoft

Microsoft Access y Microsoft Excel pueden funcionar como almacén de datos para una aplicación de ADO.NET cuando los requisitos de seguridad son mínimos o no existen. Sus características de seguridad son eficaces para fines de disuasión, aunque sólo es conveniente confiar en ellas para evitar la intromisión por parte de usuarios no informados. Los archivos de datos físicos de Access y Excel existen en el sistema de archivos y todos los usuarios deben tener acceso a ellos. Esto los hace vulnerables ante ataques que pueden dar lugar al robo o pérdida de datos, ya que los archivos se pueden copiar o modificar fácilmente. Cuando sea necesaria una seguridad potente, use SQL Server u otra base de datos basada en servidor donde los archivos de datos físicos no se puedan leer desde el sistema de archivos.

⊟ Enterprise Services

COM+ incluye su propio modelo de seguridad que se basa en las cuentas de Windows NT y en la suplantación de procesos y subprocesos. El espacio de nombres System.EnterpriseServices proporciona contenedores que permiten a las aplicaciones .NET integrar código administrado con servicios de seguridad COM+ mediante la clase ServicedComponent.

⊟ Interoperar con código no administrado

.NET Framework proporciona interacción con código no administrado, incluidos componentes COM, servicios COM+, bibliotecas de tipos externas y muchos servicios del sistema operativo. El trabajo con código no administrado supone traspasar el perímetro de seguridad del código administrado. Tanto su código como cualquier otro código que llame a su código, deben tener el permiso de código no administrado (SecurityPermission con el marcador UnmanagedCode especificado). El código no administrado puede insertar de forma involuntaria vulnerabilidades de seguridad en la aplicación. Por tanto, debe evitar la interoperabilidad con código no administrado a menos que sea absolutamente necesario.

4.3. Aplicaciones Cliente Seguras

Por lo general las aplicaciones constan de varios elementos que deben estar protegidos ante las vulnerabilidades que pueden provocar pérdidas de datos o poner en peligro el sistema de cuaquier otro modo. La creación de interfaces de usuario seguras puede impedir un gran número de problemas ya que bloquea a los atacantes antes de que puedan tener acceso a los datos o a los recursos del sistema.

⊟ Validar datos introducidos por el usuario

Al construir una aplicación en la que se obtiene acceso a datos, debe presuponer que todos los datos proporcionados por el usuario son malintencionados, a no ser que se demuestre lo contrario. De no ser así, la aplicación puede estar expuesta a ataques. .NET Framework contiene clases que ayudan a exigir un dominio de valores para los controles de información introducida por el usuario, como la limitación del número de caracteres que se pueden introducir. Los enlaces de eventos permiten escribir procedimientos para comprobar la validez de los valores. Los datos introducidos por el usuario se pueden validar y definir con establecimiento inflexible de tipos, lo que limita la exposición de una aplicación ante ataques de inyección de script y SQL.

Nota de seguridad:

También debe validar los datos introducidos por el usuario en el origen de datos, además de la aplicación cliente. Un atacante puede evitar la aplicación y atacar directamente al origen de datos.

Seguridad e introducción de datos por el usuario

Describe cómo controlar errores sutiles y potencialmente peligrosos relacionados con la introducción de datos por parte del usuario.

Validar la información especificada por el usuario en páginas Web ASP.NET

Información general sobre la validación de datos introducidos por el usuario con controles de validación de ASP.NET.

Datos proporcionados por el usuario en formularios Windows Forms

Proporciona vínculos e información para validar entrada de mouse y teclado en aplicaciones de formularios Windows Forms.

Expresiones regulares de .NET Framework

Describe cómo utilizar la clase Regex para comprobar la validez de los datos introducidos por el usuario.

⊟ Aplicaciones para Windows

En versiones anteriores, las aplicaciones Windows normalmente se ejecutaban con todos los permisos. .NET Framework proporciona la infraestructura para restringir la ejecución del código en una aplicación Windows mediante la seguridad de acceso del código (CAS). Sin embargo, CAS no es suficiente por sí solo para proteger la aplicación.

Seguridad en los formularios Windows Forms

Describe cómo proteger las aplicaciones de formularios Windows Forms y proporciona vínculos a temas relacionados.

Aplicaciones de Windows Forms y aplicaciones no administradas

Describe cómo interactuar con aplicaciones no administradas en una aplicación de formularios Windows Forms.

Implementación de ClickOnce para aplicaciones de formularios Windows Forms

Describe cómo usar la implementación de **ClickOnce** en una aplicación de formularios Windows Forms y describe las implicaciones en la seguridad.

⊟ ASP.NET y servicios web XML

Por lo general, las aplicaciones ASP.NET deben restringir el acceso a algunas porciones del sitio web y proporcionan otros mecanismos para la protección de datos y la seguridad del sitio. Estos vínculos proporcionan información útil para proteger la aplicación ASP.NET.

Los servicios web XML proporcionan datos que pueden consumir las aplicaciones ASP.NET, las aplicaciones de formularios Windows Forms u otros servicios web. Debe administrar la seguridad del propio servicio web así como la de la aplicación cliente.

⊟ Comunicación remota

La comunicación remota de .NET permite crear fácilmente aplicaciones ampliamente distribuidas, tanto si los componentes de las aplicaciones están todos en un equipo como si están repartidos por el mundo. Puede generar aplicaciones cliente que utilicen objetos en otros procesos del mismo equipo o en cualquier otro equipo que se pueda alcanzar en la red. También puede usar la comunicación remota de .NET para comunicar con otros dominios de aplicación en el mismo proceso.

4.4. Seguridad de Acceso del Código y ADO.NET

.NET Framework ofrece seguridad basada en funciones y seguridad de acceso del código (CAS); ambas se implementan utilizando una infraestructura común proporcionada por Common Language Runtime (CLR). En el mundo del código no administrado, la mayoría de las aplicaciones se ejecutan mediante los permisos del usuario o de la entidad de seguridad. Por consiguiente, los sistemas de equipos pueden resultar dañados y se pueden poner en peligro los datos privados si un usuario con un nivel elevado de privilegios ejecuta software malintencionado o que contenga errores.

El código administrado que se ejecuta en .NET Framework, sin embargo, incluye seguridad de acceso del código, que se aplica únicamente al código. El permiso para que el código se ejecute o no depende de su

origen o de otros aspectos de la identidad del código, y no sólo de la identidad de la entidad de seguridad. De esta forma, se reduce la probabilidad de que se utilice el código de manera incorrecta.

⊟ Permisos de acceso a código

Cuando se ejecuta el código, éste presenta evidencia de que lo evalúa el sistema de seguridad Common Language Runtime CLR. Este tipo de evidencia normalmente contiene el origen del código, incluidos la dirección URL, el sitio, la zona y las firmas digitales que determinan la identidad del ensamblado.

CLR permite que el código realice únicamente las operaciones para las que tiene permiso. El código puede solicitar permisos y las peticiones se aceptan en función de la directiva de seguridad que haya establecido un administrador.

Nota:

El código que se ejecuta en el CLR no se puede conceder permisos a sí mismo. Por ejemplo, el código puede solicitar y que se le asignen menos permisos de los que concede una directiva de seguridad, pero no se le concederán más permisos. A la hora de conceder permisos, comience sin ningún permiso y agregue los mínimos permisos necesarios para la tarea concreta que se lleva a cabo. Si se comienza con todos los permisos y posteriormente se deniegan permisos de forma individual, se obtienen aplicaciones poco seguras que pueden contener vacíos no intencionados en la seguridad debido a la concesión de más permisos de los que son necesarios.

Hay tres tipos de permisos de acceso a código:

- Los **permisos de acceso a código** derivan de la clase CodeAccessPermission. Se requieren permisos para tener acceso a recursos protegidos, como archivos y variables de entorno, y para realizar operaciones protegidas, como el acceso a código no administrado.
- Los **permisos de identidad** representan características que identifican un ensamblado. Los permisos se conceden al ensamblado en función de la evidencia, que puede incluir elementos como una firma digital o el origen del código. Los permisos de identidad también derivan de la clase base CodeAccessPermission.
- Los **permisos de seguridad basados en funciones** se basan en el hecho de si una entidad de seguridad tiene una identidad especificada o si es miembro de una función especificada. La clase PrincipalPermission permite comprobar los permisos declarativos e imperativos con la entidad de seguridad activa.

Para determinar si el código tiene autorización para el acceso a un recurso o para ejecutar una operación, el sistema de seguridad en tiempo de ejecución atraviesa la pila de llamadas y compara los permisos concedidos a cada llamador con el permiso que se exige. Si algún llamador de la pila de llamadas no tiene el permiso exigido, se iniciará clase SecurityException y se rechazará el acceso.

Solicitar permisos

La finalidad de solicitar permisos es informar al motor en tiempo de ejecución de los permisos que necesita la aplicación para ejecutarse y garantizar que sólo recibe los permisos que realmente necesita. Por ejemplo, si la aplicación necesita acceso de escritura al disco local, requiere FileIOPermission. Si este permiso no se ha concedido, se producirán errores en la aplicación al intentar escribir en el disco. Sin embargo, si la aplicación solicita **FileIOPermission** y el permiso no se concede, generará la excepción al comienzo y no se cargará.

En un escenario donde la aplicación sólo necesita leer datos del disco, puede solicitar que nunca se le concedan permisos de escritura. En el caso de se produzca un error o un ataque malintencionado, el código no podrá dañar los datos con los que trabaja.

⊟ Seguridad basada en funciones y CAS

La implementación de la seguridad basada en funciones y de la seguridad de acceso del código (CAS) mejora la seguridad global de la aplicación. La seguridad basada en funciones se puede basar en una cuenta de Windows o en una identidad personalizada, de forma que la información sobre la entidad de seguridad esté disponible en el subproceso actual. Además, a menudo se requiere a la aplicaciones que proporcionen acceso a datos o recursos basándose en credenciales proporcionadas por el usuario. Normalmente, dichas aplicaciones comprueban la función de un usuario y proporcionan acceso a los recursos basándose en dichas funciones.

La seguridad basada en funciones permite que un componente identifique los usuarios actuales y sus funciones asociadas en tiempo de ejecución. Esta información se asigna a continuación mediante una directiva CAS para determinar el conjunto de permisos que se conceden en tiempo de ejecución. En un dominio de aplicación especificado, el host puede cambiar la directiva de seguridad predeterminada basada en funciones y establecer una entidad de seguridad que represente a un usuario y a las funciones asociadas al usuario.

CLR usa permisos para implementar su mecanismo a fin de aplicar restricciones en el código administrado. Los permisos de seguridad basada en funciones proporcionan un mecanismo para descubrir si un usuario (o el agente que actúa en su nombre) tiene una identidad concreta o es miembro de una función especificada.

En función del tipo de aplicación que cree, deberá considerar también la posibilidad de implementar permisos basados en funciones en la base de datos.

⊟ Ensamblados

Los ensamblados componen la unidad fundamental de implementación, control de versiones, reutilización, ámbito de activación y permisos de seguridad en una aplicación de .NET Framework. Un ensamblado proporciona una colección de tipos y recursos creados para funcionar en conjunto y formar una unidad lógica de funcionalidad. En CLR, un tipo no existe si no es en el contexto de un ensamblado.

Ensamblados con nombre seguro

Un nombre seguro, o firma digital, consta de la identidad del ensamblado, que incluye su nombre de texto sencillo, el número de versión, la información de referencia cultural (si se proporciona), así como una clave pública y una firma digital. La firma digital se genera a partir de un archivo de ensamblado que usa la clave privada correspondiente. El archivo de ensamblado contiene el manifiesto del ensamblado, que contiene los nombres y códigos hash de todos los archivos que forman el ensamblado.

La asignación de un nombre seguro al ensamblado proporciona a una aplicación o a un componente una identidad única que puede usar otro software para referirse a ella de manera explícita. La asignación de nombres seguros protege a los ensamblados de la suplantación por parte de un ensamblado que contenga código malintencionado. También garantiza la coherencia entre las diferentes versiones de un componente. Debe usar nombres seguros en los ensamblados que se van a implementar en la caché de ensamblados global (GAC).

⊟ Confianza parcial en ADO.NET 2.0

En ADO.NET 2.0, se pueden ejecutar los proveedores de datos de .NET Framework para SQL Server, OLE DB, ODBC y para Oracle en entornos de confianza parcial. En versiones anteriores de .NET Framework, sólo se admitía el uso de System.Data.SqlClient en aplicaciones que no fuesen de plena confianza.

Una aplicación de confianza parcial que utilice el proveedor de SQL Server debe tener como mínimo permisos de ejecución y SqlClientPermission.

Propiedades de atributos de permiso de confianza parcial

En las situaciones de confianza parcial, se puede aplicar SqlClientPermissionAttribute a los miembros a fin de restringir aún más las capacidades disponibles para el proveedor de datos de .NET Framework para SQL Server.

Nota:

El proveedor de datos de .NET Framework para SQL Server necesita permiso de plena confianza para abrir una SqlConnection con la depuración de SQL habilitada en SQL Server 2000. La depuración de SQL para SQL Server 2005 no utiliza esta clase. Para obtener más información, vea los Libros en pantalla de SQL Server 2005.

En la siguiente tabla se muestran y se describen las propiedades SqlClientPermissionAttribute disponibles:

Propiedad de atributo de permiso	Descripción
Action	Obtiene o establece una acción de seguridad. Se hereda de SecurityAttribute.
AllowBlankPassword	Habilita o deshabilita el uso de una contraseña en blanco en una cadena de conexión. Los valores válidos son **true** (para habilitar el uso de contraseñas en blanco) y **false** (para deshabilitarlo). Se hereda de DBDataPermissionAttribute.
ConnectionString	Identifica una cadena de conexión admitida. Se pueden identificar varias cadenas de conexión. **Nota:** No incluya un id. de usuario o una contraseña en la cadena de conexión. En esta versión no se pueden modificar las restricciones de las cadenas de conexión mediante la herramienta Configuración de .NET Framework. Se hereda de DBDataPermissionAttribute.
KeyRestrictions	Identifica qué parámetros de las cadenas de conexión están permitidos o no lo están. Los parámetros de las cadenas de conexión se identifican con el formato <nombre de parámetro>=. Se pueden especificar varios parámetros separados por punto y coma (;). **Nota:** Si no especifica **KeyRestrictions**, pero establece la propiedad **KeyRestrictionBehavior** en **AllowOnly** o **PreventUsage**, no se permitirán parámetros de cadena de conexión adicionales. Se hereda de DBDataPermissionAttribute.
KeyRestrictionBehavior	Identifica los parámetros de cadenas de conexión como los únicos

	parámetros adicionales permitidos (**AllowOnly**), o bien identifica los parámetros adicionales no permitidos (**PreventUsage**). **AllowOnly** es el valor predeterminado. Se hereda de DBDataPermissionAttribute.
TypeID	Obtiene un identificador único para el atributo cuando se implementa en una clase derivada. Se hereda de Attribute.
Unrestricted	Indica si se declaran permisos protegidos para el origen. Se hereda de SecurityAttribute.

Sintaxis de ConnectionString

En el siguiente ejemplo se muestra cómo se utiliza el elemento **connectionStrings** de un archivo de configuración para permitir únicamente el uso de una determinada cadena de conexión. Para obtener más información sobre el almacenamiento de cadenas de conexión en archivos de configuración y recuperación de las mismas,.

```
<connectionStrings>
  <add name="DatabaseConnection"
    connectionString="Data Source=(local);Initial
    Catalog=Northwind;Integrated Security=true;" />
</connectionStrings>
```

Sintaxis de KeyRestrictions

En el ejemplo siguiente se habilita la misma cadena de conexión, se habilita el uso de las opciones de cadena de conexión **Encrypt** y **PacketSize**, pero se restringe el uso de otras opciones de cadena de conexión.

```
<connectionStrings>
  <add name="DatabaseConnection"
    connectionString="Data Source=(local);Initial
    Catalog=Northwind;Integrated Security=true;"
    KeyRestrictions="Encrypt=;Packet Size=;"
    KeyRestrictionBehavior="AllowOnly" />
</connectionStrings>
```

Sintaxis de KeyRestrictionBehavior con PreventUsage

En el ejemplo siguiente se habilita la misma cadena de conexión y se permiten todos los demás parámetros de conexión, excepto **User Id**, **Password** y **Persist Security Info**.

```
<connectionStrings>
  <add name="DatabaseConnection"
    connectionString="Data Source=(local);Initial
    Catalog=Northwind;Integrated Security=true;"
    KeyRestrictions="User Id=;Password=;Persist Security Info=;"
    KeyRestrictionBehavior="PreventUsage" />
</connectionStrings>
```

Sintaxis de KeyRestrictionBehavior con AllowOnly

En el ejemplo siguiente se habilitan dos cadenas de conexión que contienen los parámetros **Initial Catalog**, **Connection Timeout**, **Encrypt** y **Packet Size**. El resto de los parámetros de cadenas de conexión están restringidos.

```
<connectionStrings>
  <add name="DatabaseConnection"
    connectionString="Data Source=(local);Initial
```

```
    Catalog=Northwind;Integrated Security=true;"
    KeyRestrictions="Initial Catalog;Connection Timeout=;
       Encrypt=;Packet Size=;"
    KeyRestrictionBehavior="AllowOnly" />

  <add name="DatabaseConnection2"
    connectionString="Data Source=SqlServer2;Initial
    Catalog=Northwind2;Integrated Security=true;"
    KeyRestrictions="Initial Catalog;Connection Timeout=;
       Encrypt=;Packet Size=;"
    KeyRestrictionBehavior="AllowOnly" />
</connectionStrings>
```

Habilitar confianza parcial con un conjunto de permisos personalizados

Para habilitar el uso de permisos System.Data.SqlClient para una zona determinada, un administrador del sistema debe crear un conjunto de permisos personalizados y establecerlo como el conjunto de permisos de dicha zona. Los conjuntos de permisos predeterminados, como **LocalIntranet**, no se pueden modificar. Por ejemplo, para incluir permisos System.Data.SqlClient para un código que tiene una Zone de **LocalIntranet**, un administrador del sistema tendría que copiar el conjunto de permisos para **LocalIntranet**, cambiar el nombre a "CustomLocalIntranet", agregar los permisos System.Data.SqlClient, importar el conjunto de permisos CustomLocalIntranet mediante Herramienta de la directiva de seguridad de acceso a código (Caspol.exe) y establecer el conjunto de permisos de **LocalIntranet_Zone** en CustomLocalIntranet.

Conjunto de permisos de ejemplo

A continuación se muestra un ejemplo de un conjunto de permisos para el proveedor de datos de .NET Framework para SQL Server en un escenario que no es de plena confianza. Para obtener información sobre la creación de conjuntos de permisos.

```
<PermissionSet class="System.Security.NamedPermissionSet"
  version="1"
  Name="CustomLocalIntranet"
  Description="Custom permission set given to applications on
    the local intranet">
<IPermission class="System.Data.SqlClient.SqlClientPermission,
System.Data, Version=2.0.0000.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089"
version="1"
AllowBlankPassword="False">
<add ConnectionString="Data Source=(local);Integrated Security=true;"
 KeyRestrictions="Initial Catalog=;Connection Timeout=;
   Encrypt=;Packet Size=;"
 KeyRestrictionBehavior="AllowOnly" />
 </IPermission>
</PermissionSet>
```

⊟ Comprobar el acceso a código de ADO.NET mediante permisos de seguridad

En las situaciones de confianza parcial, puede solicitar privilegios de seguridad de acceso del código para determinados métodos en el código mediante la especificación de SqlClientPermissionAttribute. Si la directiva de seguridad restringida no permite el privilegio, se inicia una excepción antes de ejecutarse el código.

Ejemplo

En el siguiente ejemplo se muestra cómo escribir código que requiera una determinada cadena de conexión. Simula la denegación de permisos sin restricciones para System.Data.SqlClient, que podría implementar un administrador del sistema en una situación real mediante una directiva de seguridad CAS.

Nota de seguridad:

Al diseñar permisos de seguridad CAS en ADO.NET, el procedimiento correcto es comenzar con el caso más restrictivo (sin ningún permiso) y agregar a continuación los permisos específicos necesarios para la tarea determinada que el código debe realizar. El modelo opuesto, que comienza con todos los permisos y deniega a continuación un permiso concreto, no es seguro puesto que existen muchas formas de expresar la misma cadena de conexión. Por ejemplo, si comienza con todos los permisos y después intenta denegar el uso de la cadena de conexión "servidor=unServidor", la cadena "servidor=unServidor.miEmpresa.com" seguirá obteniendo permiso. Al comenzar siempre por no conceder ningún permiso, se reduce la posibilidad de que haya lagunas en el conjunto de permisos.

En el siguiente código se muestra cómo **SqlClient** realiza la petición de seguridad, que inicia SecurityException si los permisos de seguridad CAS adecuados no se encuentran en su lugar. El resultado SecurityException se muestra en la ventana de la consola.

```
Private Sub TestCAS(ByVal connectString1 As String, ByVal connectString2
As String)
    ' Simulate removing SqlClient permissions.
  Dim permission As New SqlClientPermission(PermissionState.Unrestricted)
    permission.Deny()
    ' Try to open a connection.
    Try
        Using connection As New SqlConnection(connectString1)
            connection.Open()
            Console.WriteLine("Connection opened, unexpected.")
        End Using
    Catch ex As System.Security.SecurityException
        Console.WriteLine("Failed, as expected: {0}", _
            ex.FirstPermissionThatFailed)
        ' Uncomment the following line to see Exception details.
        ' Console.WriteLine("BaseException: {0}", ex.GetBaseException())
    End Try
    SqlClientPermission.RevertAll()
    ' Add permission for a specific connection string.
    ' This would typically be achieved by the administrator
    ' deploying a CAS policy, not in your code.
    permission = New SqlClientPermission(PermissionState.None)
    permission.Add(connectString1, "", KeyRestrictionBehavior.AllowOnly)
    permission.PermitOnly()
    ' Try again, it should succeed now.
    Try
        Using connection As New SqlConnection(connectString1)
            connection.Open()
            Console.WriteLine("Connection opened, as expected.")
        End Using
    Catch ex As System.Security.SecurityException
```

```
            Console.WriteLine("Unexpected failure: {0}", ex.Message)
        End Try
        ' Try a different connection string. This should fail.
        Try
            Using connection As New SqlConnection(connectString2)
                connection.Open()
                Console.WriteLine("Connection opened, unexpected.")
            End Using
        Catch ex As System.Security.SecurityException
            Console.WriteLine("Failed, as expected: {0}", ex.Message)
        End Try
    End Sub
```

Debe poder ver este resultado en la ventana de la consola:

```
Failed, as expected: <IPermission class="System.Data.SqlClient.
SqlClientPermission, System.Data, Version=2.0.0.0,
  Culture=neutral, PublicKeyToken=b77a5c561934e089" version="1"
  AllowBlankPassword="False">
<add ConnectionString="Data Source=(local);Initial Catalog=
  Northwind;Integrated Security=SSPI" KeyRestrictions=""
KeyRestrictionBehavior="AllowOnly"/>
</IPermission>

Connection opened, as expected.
Failed, as expected: Request failed.
```

⊟ Interoperabilidad de código no administrado

El código que se ejecuta fuera de CLR se denomina código no administrado. Por lo tanto, los mecanismos de seguridad como CAS no se pueden aplicar en código no administrado. Los componentes COM, las interfaces ActiveX y las funciones de la API de Win32 son ejemplos de código no administrado. Cuando se ejecuta código no administrado se aplican consideraciones de seguridad especiales, de forma que no se ponga en peligro la seguridad global de la aplicación.

.NET Framework también es compatible con versiones anteriores de componentes COM existentes mediante el acceso a través de la interoperabilidad COM. Se pueden incluir componentes COM en una aplicación de .NET Framework usando las herramientas de la interoperabilidad COM para importar los tipos COM necesarios. Una vez que se han importado, los tipos COM están listos para su uso. La interoperabilidad COM también permite que los clientes COM tengan acceso a código administrado mediante la exportación de metadatos de ensamblado a una biblioteca de tipos y mediante el registro del componente administrado como un componente COM.

4.5. Privacidad y Seguridad de Datos

La protección y la administración de información confidencial en una aplicación de ADO.NET depende de los productos y las tecnologías utilizados para crearla. ADO.NET no proporciona de forma directa servicios para proteger ni cifrar datos.

⊟ Criptografía y códigos hash

Las clases del espacio de nombres System.Security.Cryptography de .NET Framework se pueden utilizar desde las aplicaciones de ADO.NET para impedir que terceras partes no autorizadas lean o modifiquen

los datos. Algunas clases son contenedores para Microsoft CryptoAPI no administrado, mientras que otras son implementaciones administradas.

Al contrario que la criptografía, que permite cifrar datos y descifrarlos posteriormente, el proceso hash de datos es unidireccional. La utilización de algoritmos hash en los datos resulta útil para evitar la manipulación mediante la comprobación de que los datos no han sido alterados: para cadenas de entrada idénticas, los algoritmos hash siempre generan valores de salida cortos idénticos que se pueden comparar fácilmente.

⊟ Cifrar archivos de configuración

La protección del acceso al origen de datos es uno de los objetivos más importantes a la hora de proteger una aplicación. Las cadenas de conexión presentan una posible vulnerabilidad si no se protegen. Las cadenas de conexión que se guardan en los archivos de configuración se almacenan en archivos XML estándar para los que .NET Framework ha definido un conjunto común de elementos. La configuración protegida permite cifrar información confidencial en un archivo de configuración. Si bien se ha diseñado principalmente para aplicaciones ASP.NET, la configuración protegida también se puede usar para cifrar secciones del archivo de configuración en aplicaciones de Windows.

⊟ Proteger valores de cadena en memoria

Si un objeto String contiene información confidencial, como una contraseña, un número de tarjeta de crédito o datos personales, existe el riesgo de que la información se pueda revelar una vez utilizada, porque la aplicación no puede eliminar los datos de la memoria del equipo.

Un objeto String es inmutable, porque no se puede modificar su valor una vez que se ha creado. Los cambios que parecen modificar el valor de la cadena crean de hecho una nueva instancia de un objeto String en memoria, que almacena los datos como texto sin formato. Además, no es posible predecir si las instancias de cadena se eliminarán de la memoria. La reclamación de memoria con cadenas no es determinista en la recolección de elementos no utilizados de .NET. Debe evitar el uso de las clases String y StringBuilder si los datos son realmente confidenciales.

La clase SecureString proporciona métodos para cifrar texto con la API de protección de datos (DPAPI) en memoria. La cadena se elimina posteriormente de la memoria cuando ya no es necesaria. No existe ningún método **ToString** para leer rápidamente el contenido de un objeto SecureString. Puede inicializar una nueva instancia de **SecureString** sin ningún valor o pasándole un nuevo puntero a una matriz de objetos Char. A continuación, puede usar los diferentes métodos de la clase para trabajar con la cadena.

5. Trabajar con Objetos DataSet

El DataSet de ADO.NET es una representación de datos residente en memoria que proporciona un modelo de programación relacional coherente independientemente del origen de datos que contiene. Un DataSet representa un conjunto completo de datos, incluyendo las tablas que contienen, ordenan y restringen los datos, así como las relaciones entre las tablas.

Hay varias maneras de trabajar con un DataSet, que se pueden aplicar de forma independiente o conjuntamente. Puede:

- Crear mediante programación una DataTable, DataRelation y una Constraint en un DataSet y rellenar las tablas con datos.

- Llenar el DataSet con tablas de datos de un origen de datos relacional existente mediante **DataAdapter**.
- Cargar y hacer persistente el contenido de DataSet mediante XML.

También se puede transportar un DataSet con establecimiento inflexible de tipos mediante un servicio web XML. El diseño del DataSet lo convierte en idóneo para el transporte de datos mediante servicios web XML.

5.1. Crear un DataSet

Puede crear una instancia de DataSet llamando al constructor DataSet. Si lo desea, especifique un nombre de argumento. Si no especifica ningún nombre para el DataSet, se establecerá el nombre "NewDataSet".

También es posible crear un nuevo DataSet basado en un DataSet existente. El nuevo DataSet puede ser una copia exacta del DataSet existente; un clon del DataSet que copia la estructura relacional o el esquema, pero que no contiene ningún dato del DataSet existente; o un subconjunto del DataSet, que contiene solamente las filas modificadas del DataSet existente mediante el método GetChanges.

En el siguiente ejemplo de código se muestra cómo construir una instancia de un DataSet.

```
Dim customerOrders As DataSet = New DataSet("CustomerOrders")
```

5.2. Agregar un DataTable a un DataSet

ADO.NET permite crear objetos DataTable y agregarlos a un DataSet existente. Es posible establecer información de restricciones para una DataTable mediante las propiedades PrimaryKey y Unique.

⊟ Ejemplo

En el siguiente ejemplo se construye un DataSet, se agrega un objeto DataTable nuevo al DataSet y, a continuación, se agregan tres objetos DataColumn a la tabla. Por ultimo, el código establece una columna como columna de clave principal.

```
Dim customerOrders As DataSet = New DataSet("CustomerOrders")
Dim ordersTable As DataTable = customerOrders.Tables.Add("Orders")
Dim pkOrderID As DataColumn = ordersTable.Columns.Add( _
    "OrderID", Type.GetType("System.Int32"))
ordersTable.Columns.Add("OrderQuantity", Type.GetType("System.Int32"))
ordersTable.Columns.Add("CompanyName", Type.GetType("System.String"))
ordersTable.PrimaryKey = New DataColumn() {pkOrderID}
```

⊟ Distinguir mayúsculas de minúsculas

Pueden existir dos o más tablas o relaciones con el mismo nombre, pero que difieran en mayúsculas y minúsculas, en un DataSet. En estos casos, las referencias a tablas y relaciones por nombre distinguen mayúsculas y minúsculas. Por ejemplo, si el DataSet **dataSet** contiene las tablas **Table1** y **table1**, se hace referencia por nombre a **Table1** como **dataSet.Tables["Table1"]** y se hace referencia a **table1** como **dataSet.Tables["table1"]**. Si se intentara hacer referencia a cualquiera de las tablas mediante **dataSet.Tables["TABLE1"]** se generaría una excepción.

El comportamiento de distinción entre mayúsculas y minúsculas no se aplica si sólo hay una tabla o relación con un nombre concreto. Por ejemplo, si el DataSet sólo tiene **Table1**, se puede hacer referencia a ésta mediante **dataSet.Tables["TABLE1"]**.

Nota:

La propiedad CaseSensitive del DataSet no afecta a este comportamiento. La propiedad CaseSensitive se aplica a los datos del DataSet y afecta a la ordenación, la búsqueda, el filtrado, la aplicación de restricciones, etc.

Compatibilidad con los espacios de nombres

En versiones anteriores de ADO.NET, dos tablas no podían tener el mismo nombre, aunque se encontrasen en espacios de nombres diferentes. Esta limitación ha desaparecido en ADO.NET 2.0. Un DataSet puede contener dos tablas con el mismo valor de propiedad TableName, pero con valores de propiedad Namespace diferentes.

5.3. Agregar DataRelations

En un DataSet que contiene varios objetos DataTable, es posible utilizar objetos DataRelation para relacionar una tabla con otra, navegar por las tablas y devolver filas secundarias o primarias de una tabla relacionada.

Los argumentos necesarios para crear una **DataRelation** son un nombre para la **DataRelation** que se va a crear y una matriz de una o más referencias DataColumn a las columnas que actúan como columnas primaria y secundaria en la relación. Una vez creado un objeto **DataRelation**, es posible utilizarlo para navegar por las tablas y recuperar valores.

Al agregar una **DataRelation** a una DataSet, se agrega de forma predeterminada una UniqueConstraint a la tabla primaria y una ForeignKeyConstraint a la tabla secundaria.

En el siguiente ejemplo de código se crea una **DataRelation** mediante dos objetos DataTable en un DataSet. Cada DataTable contiene una columna denominada **CustID**, que actúa como vínculo entre los dos objetos DataTable. En el ejemplo se agrega una única **DataRelation** a la colección **Relations** del DataSet. El primer argumento del ejemplo especifica el nombre de la **DataRelation** que se va a crear. El segundo argumento establece la **DataColumn** primaria y el tercer argumento establece la **DataColumn** secundaria.

```
customerOrders.Relations.Add("CustOrders", _
  customerOrders.Tables("Customers").Columns("CustID"), _
  customerOrders.Tables("Orders").Columns("CustID"))
```

Una **DataRelation** tiene también una propiedad **Nested** que, cuando tiene el valor **true**, hace que las filas de la tabla secundaria se aniden dentro de la fila asociada de la tabla primaria cuando se escriben como elementos XML mediante WriteXml.

5.4. Navegar por DataRelations

Una de las principales funciones de una DataRelation es permitir la navegación de una DataTable a otra dentro de un DataSet. Esto permite recuperar todos los objetos DataRow relacionados de una **DataTable** cuando se da una única **DataRow** de una **DataTable** relacionada. Por ejemplo, después de establecer

una **DataRelation** entre una tabla de clientes y una tabla de pedidos, es posible recuperar todas las filas de pedidos de una fila de clientes determinada mediante **GetChildRows**.

En el siguiente ejemplo de código se crea una **DataRelation** entre la tabla **Customers** y la tabla **Orders** de un **DataSet**, y se devuelven todos los pedidos de cada cliente.

```
Dim customerOrdersRelation As DataRelation = _
   customerOrders.Relations.Add("CustOrders", _
   customerOrders.Tables("Customers").Columns("CustomerID"), _
   customerOrders.Tables("Orders").Columns("CustomerID"))
Dim custRow, orderRow As DataRow
For Each custRow In customerOrders.Tables("Customers").Rows
    Console.WriteLine("Customer ID:" & custRow("CustomerID").ToString())
    For Each orderRow In custRow.GetChildRows(customerOrdersRelation)
        Console.WriteLine(orderRow("OrderID").ToString())
    Next
Next
```

El ejemplo siguiente se basa en el anterior; se relacionan cuatro tablas y se navega por dichas relaciones. Como en el ejemplo anterior, **CustomerID** relaciona la tabla **Customers** con la tabla **Orders**. Para cada cliente de la tabla **Customers** se determinan todas las filas secundarias de la tabla **Orders** con el fin de devolver el número de pedidos que tiene un cliente concreto y sus valores de **OrderID**.

El ejemplo ampliado también devuelve los valores de las tablas **OrderDetails** y **Products**. La tabla **Orders** está relacionada con la tabla **OrderDetails** mediante **OrderID** con el fin de determinar, para cada pedido de cliente, qué productos y cantidades se pidieron. Como la tabla **OrderDetails** sólo contiene el **ProductID** de un producto pedido, **OrderDetails** está relacionada con **Products** mediante **ProductID** para devolver el **ProductName**. En esta relación, **Products** es la tabla primaria y **Order Details** es la secundaria. Por lo tanto, al recorrer en iteración la tabla **OrderDetails**, se llama a **GetParentRow** para recuperar el valor de **ProductName** relacionado.

Hay que tener en cuenta que al crear la **DataRelation** para las tablas **Customers** y **Orders** no se especifica ningún valor para el marcador **createConstraints** (el valor predeterminado es **true**). Se supone que todas las filas de la tabla **Orders** tienen un valor **CustomerID** que existe en la tabla primaria **Customers**. Si un **CustomerID** existe en la tabla **Orders** pero no existe en la tabla **Customers**, una ForeignKeyConstraint hará que se inicie una excepción.

Cuando la columna secundaria pueda contener valores no incluidos en la columna primaria, hay que asignar el valor **false** al marcador **createConstraints** cuando se agregue la **DataRelation**. En el ejemplo, el marcador **createConstraints** tiene el valor **false** para la **DataRelation** entre las tablas **Orders** y **OrderDetails**. Esto permite que la aplicación devuelva todos los registros de la tabla **OrderDetails** y sólo un subconjunto de registros de la tabla **Orders** sin generar una excepción en tiempo de ejecución. El ejemplo ampliado genera el resultado con el siguiente formato.

```
      Customer ID: NORTS
        Order ID: 10517
              Order Date: 4/24/1997 12:00:00 AM
                 Product: Filo Mix
                Quantity: 6
                 Product: Raclette Courdavault
                Quantity: 4
```

```
                Product: Outback Lager
               Quantity: 6
        Order ID: 11057
              Order Date: 4/29/1998 12:00:00 AM
                Product: Outback Lager
               Quantity: 3
```

El siguiente ejemplo de código es un ejemplo ampliado en el que se devuelven los valores de las tablas **OrderDetails** y **Products**, y sólo se devuelve un subconjunto de los registros de la tabla **Orders**.

```
Dim customerOrdersRelation As DataRelation = _
   customerOrders.Relations.Add("CustOrders", _
   customerOrders.Tables("Customers").Columns("CustomerID"), _
   customerOrders.Tables("Orders").Columns("CustomerID"))
Dim orderDetailRelation As DataRelation = _
   customerOrders.Relations.Add("OrderDetail", _
   customerOrders.Tables("Orders").Columns("OrderID"), _
   customerOrders.Tables("OrderDetails").Columns("OrderID"), False)
Dim orderProductRelation As DataRelation = _
   customerOrders.Relations.Add("OrderProducts", _
   customerOrders.Tables("Products").Columns("ProductID"), _
   customerOrders.Tables("OrderDetails").Columns("ProductID"))
Dim custRow, orderRow, detailRow As DataRow
For Each custRow In customerOrders.Tables("Customers").Rows
    Console.WriteLine("Customer ID:" & custRow("CustomerID").ToString())
    For Each orderRow In custRow.GetChildRows(customerOrdersRelation)
       Console.WriteLine("  Order ID: " & orderRow("OrderID").ToString())
Console.WriteLine(vbTab & "Order Date:" & orderRow("OrderDate").ToString)
        For Each detailRow In orderRow.GetChildRows(orderDetailRelation)
            Console.WriteLine(vbTab & "   Product: " & _
              detailRow.GetParentRow(orderProductRelation) _
              ("ProductName").ToString())
            Console.WriteLine(vbTab & "  Quantity: " & _
              detailRow("Quantity").ToString())
        Next
    Next
Next
```

5.5. Combinar Contenido de un DataSet

Se puede utilizar el método Merge para combinar el contenido de DataSet, DataTable o matriz de DataRow en un **DataSet** existente. Hay varios factores y opciones que afectan a cómo se combinan los datos nuevos en un **DataSet** existente.

⊟ Claves principales

Si la tabla que recibe datos y esquema nuevos a partir de una combinación tiene una clave principal, las nuevas filas de los datos entrantes se hacen coincidir con las filas existentes que tienen los mismos valores de clave principal Original que los de los datos entrantes. Si las columnas del esquema entrante coinciden con las del esquema existente, se modificarán los datos de las filas existentes. Las columnas que no coincidan con el esquema existente se pasarán por alto o se agregarán en función del parámetro

MissingSchemaAction. Las nuevas filas con valores de clave principal que no coincidan con las filas existentes se agregarán a la tabla existente.

Si las filas entrantes o las existentes tienen un estado Added, se harán coincidir sus valores de clave principal mediante el valor de clave principal Current de la fila **Added**, ya que no existe ninguna versión de fila **Original**.

Si una tabla entrante y una tabla existente tienen una columna con el mismo nombre pero con distintos tipos de datos, se iniciará una excepción y se generará el evento MergeFailed de **DataSet**. Si una tabla entrante y una tabla existente tienen claves definidas, pero las claves principales corresponden a columnas diferentes, se iniciará una excepción y se provocará el evento **MergeFailed** de **DataSet**.

Si la tabla que recibe nuevos datos de una combinación no tiene una clave principal, las nuevas filas de los datos entrantes no se pueden hacer coincidir con las filas existentes de la tabla y se agregarán a la tabla existente.

⊟ Nombres de tabla y espacios de nombres

A los objetos DataTable se les puede asignarse también un valor de propiedad Namespace. Cuando se asignan los valores Namespace, DataSet puede contener varios objetos DataTable con el mismo valor TableName. Durante las operaciones de combinación, se utilizan tanto TableName como Namespace para identificar el destino de una combinación. Si no se ha asignado Namespace sólo se utiliza TableName para identificar el destino de una combinación.

Nota:

Este comportamiento ha cambiado en la versión 2.0 de .NET Framework. En la versión 1.1, se admitían los espacios de nombres pero eran pasados por alto durante las operaciones de combinación. Por ello, un DataSet que utiliza valores de propiedad Namespace tendrá diferentes comportamientos en función de la versión de .NET Framework que se ejecute. Por ejemplo, suponga que tiene dos **DataSets** que contienen **DataTables** con los mismos valores de propiedad TableName pero distintos valores de propiedad Namespace. En la versión 1.1 de .NET Framework, los nombres Namespace distintos serán pasados por alto cuando se combinen dos objetos DataSet. Sin embargo, en la versión 2.0 de .NET Framework, la combinación produce dos nuevos **DataTables** para crearse en el DataSet de destino. El **DataTables** original no se verá afectado por la combinación.

⊟ PreserveChanges

Cuando se pasa una matriz de **DataSet**, **DataTable** o **DataRow** al método **Merge**, es posible incluir parámetros opcionales que especifiquen si se conservarán o no los cambios en el **DataSet** existente y cómo tratar los nuevos elementos de esquema de los datos entrantes. El primero de estos parámetros después de los datos entrantes es un marcador booleano, PreserveChanges, que especifica si se conservarán o no los cambios en el **DataSet** existente. Si el marcador **PreserveChanges** está establecido en **true**, los valores entrantes no sobrescriben los existentes en la versión de fila **Current** de la fila existente. Si el marcador **PreserveChanges** está establecido en **false**, los valores entrantes sobrescriben los existentes en la versión de fila **Current** de la fila existente. Si el marcador **PreserveChanges** no está especificado, de forma predeterminada se establece en **false**. Para obtener más información sobre versiones de fila, vea Estados de fila y versiones de fila.

Cuando **PreserveChanges** es **true**, los datos de la fila existente se mantienen en la versión de fila Current de la fila existente, mientras que los datos de la versión de fila Original de la fila existente se sobrescriben con los datos de la versión de fila **Original** de la fila entrante. El RowState de la fila existente se establece en Modified. Se aplican las excepciones siguientes:

- Si la fila existente tiene un **RowState** de **Deleted**, **RowState** se mantiene en **Deleted** y no se establece en **Modified**. En este caso, los datos de la fila entrante se almacenarán en la versión de fila **Original** de la fila existente, sobrescribiendo la versión de fila **Original** de la fila existente (a menos que la fila entrante tenga un **RowState** de **Added**).

- Si la fila entrante tiene un **RowState** de **Added**, los datos de la versión de fila **Original** de la fila existente no se sobrescribirán con datos de la fila entrante, ya que ésta no tiene una versión de fila **Original**.

Cuando **PreserveChanges** es **false**, las versiones de fila **Current** y **Original** de la fila existente se sobrescriben con datos de la fila entrante y el **RowState** de la fila existente se establece como el **RowState** de la fila entrante. Se aplican las excepciones siguientes:

- Si la fila entrante tiene un **RowState** de **Unchanged** y la fila existente tiene un **RowState** de **Modified**, **Deleted** o **Added**, el **RowState** de la fila existente se establece en **Modified**.

- Si la fila entrante tiene un **RowState** de **Added** y la fila existente tiene un **RowState** de **Unchanged**, **Modified** o **Deleted**, el **RowState** de la fila existente se establece en **Modified**. Además, los datos de la versión de fila **Original** de la fila existente no se sobrescriben con datos de la fila entrante, ya que ésta no tiene una versión de fila **Original**.

⊟ MissingSchemaAction

Es posible utilizar el parámetro opcional MissingSchemaAction del método **Merge** para especificar cómo tratará **Merge** los elementos del esquema de los datos entrantes que no formen parte del **DataSet** existente.

En la siguiente tabla se describen las opciones de **MissingSchemaAction**.

Opción MissingSchemaAction	Descripción
Add	Agrega al **DataSet** la nueva información de esquema y rellena las nuevas columnas con los valores entrantes. Éste es el valor predeterminado.
AddWithKey	Agrega al **DataSet** la nueva información de esquema y de clave principal y rellena las nuevas columnas con los valores entrantes.
Error	Inicia una excepción si se encuentra información de esquema no coincidente.
Ignore	Pasa por alto la nueva información de esquema.

⊟ Restricciones

Con el método **Merge** no se comprueban las restricciones hasta que se agregan todos los datos nuevos al **DataSet** existente. Una vez agregados los datos, se aplican restricciones en los valores actuales del **DataSet**. Hay que asegurarse de que el código controla las excepciones que puedan iniciarse debido a infracciones de las restricciones.

Tomemos como ejemplo un caso en el que una fila existente de un **DataSet** es una fila **Unchanged** con un valor de clave principal de 1. Durante una operación de combinación con una fila entrante **Modified** cuyo valor de clave principal **Original** es 2 y de clave principal **Current** es 1, la fila existente y la fila entrante no se consideran coincidentes porque los valores de clave principal **Original** son diferentes. Sin

embargo, cuando se completa la combinación y se comprueban las restricciones, se iniciará una excepción porque los valores de clave principal **Current** infringen la restricción única de la columna de clave principal.

Nota:

Cuando las filas están insertadas en una base de datos con columnas de incremento automático como puede ser una columna de identidad, el valor de columna de identidad devuelto por la inserción no coincide con el valor de **DataSet**, lo que da lugar a que las filas devueltas se agreguen en lugar de combinarse.

En el siguiente ejemplo de código se combinan dos objetos **DataSet** con esquemas diferentes en un **DataSet** con los esquemas combinados de los dos objetos **DataSet** entrantes.

```
Using connection As SqlConnection = New SqlConnection( _
   connectionString)
    Dim adapter As SqlDataAdapter = New SqlDataAdapter( _
      "SELECT CustomerID, CompanyName FROM Customers", connection)
    connection.Open()
    Dim customers As DataSet = New DataSet()
    adapter.FillSchema(customers, SchemaType.Source, "Customers")
    adapter.Fill(customers, "Customers")
    Dim orders As DataSet = New DataSet()
    orders.ReadXml("Orders.xml", XmlReadMode.ReadSchema)
    orders.AcceptChanges()
    customers.Merge(orders, True, MissingSchemaAction.AddWithKey)
End Using
```

En el siguiente ejemplo de código se toma un **DataSet** existente con actualizaciones y se pasan esas actualizaciones a un **DataAdapter** para que se procesen en el origen de datos. Los resultados se combinan entonces en el **DataSet** original. Después de rechazar los cambios que produjeron un error, se confirman los cambios combinados con **AcceptChanges**.

```
Dim customers As DataTable = dataSet.Tables("Customers")
' Make modifications to the Customers table. Get changes to the DataSet.
Dim dataSetChanges As DataSet = dataSet.GetChanges()
' Add an event handler to handle the errors during Update.
AddHandler adapter.RowUpdated, New SqlRowUpdatedEventHandler( _
  AddressOf OnRowUpdated)
connection.Open()
adapter.Update(dataSetChanges, "Customers")
connection.Close()
' Merge the updates.
dataSet.Merge(dataSetChanges, True, MissingSchemaAction.Add)
' Reject changes on rows with errors and clear the error.
Dim errRows() As DataRow = dataSet.Tables("Customers").GetErrors()
Dim errRow As DataRow
For Each errRow In errRows
    errRow.RejectChanges()
    errRow.RowError = Nothing
Next
' Commit the changes.
```

```
dataSet.AcceptChanges()
Private Sub OnRowUpdated( _
    ByVal sender As Object, ByVal args As SqlRowUpdatedEventArgs)
    If args.Status = UpdateStatus.ErrorsOccurred Then
        args.Row.RowError = args.Errors.Message
        args.Status = UpdateStatus.SkipCurrentRow
    End If
End Sub
```

5.6. Copiar Contenido de un DataSet

Se puede crear una copia de DataSet de forma que se pueda trabajar con datos sin afectar a los datos originales o bien se puede trabajar con un subconjunto de los datos desde un **DataSet**. Al copiar un **DataSet** es posible:

- Crear una copia exacta del **DataSet**, incluyendo el esquema, los datos, la información de estado de fila y las versiones de fila.
- Crear un **DataSet** que contenga el esquema de un **DataSet** existente, pero sólo las filas modificadas. Se pueden devolver todas las filas modificadas o especificar un **DataRowState** determinado.
- Copiar el esquema, o estructura relacional, del **DataSet** únicamente, sin copiar ninguna fila. Las filas se pueden importar en una DataTable existente mediante ImportRow.

Para crear una copia exacta del **DataSet** que incluya tanto el esquema como los datos, utilice el método Copy del **DataSet**. En el ejemplo siguiente se muestra cómo se crea una copia exacta del **DataSet**.

```
Dim copyDataSet As DataSet = customerDataSet.Copy()
```

Para crear una copia del **DataSet** que incluya el esquema y sólo los datos que representen filas **Added**, **Modified** o **Deleted**, utilice el método GetChanges del **DataSet**. También es posible utilizar **GetChanges** para devolver únicamente las filas que tengan un estado de fila determinado si se pasa el valor **DataRowState** al llamar a **GetChanges**. En el siguiente ejemplo de código se muestra cómo pasar un **DataRowState** al llamar a **GetChanges**.

```
' Copy all changes.
Dim changeDataSet As DataSet = customerDataSet.GetChanges()
' Copy only new rows.
Dim addedDataSetAs DataSet=customerDataSet.GetChanges(DataRowState.Added)
```

Para crear una copia de un **DataSet** que sólo incluya el esquema, utilice el método Clone del **DataSet**. También es posible agregar filas existentes al **DataSet** clonado mediante el método **ImportRow** de **DataTable**.**ImportRow** agrega datos, el estado de fila e información de versión de fila a la tabla especificada. Los valores de columna sólo se agregan cuando los nombres de columna coinciden y el tipo de datos es compatible.

En el siguiente ejemplo de código se crea un clon de un **DataSet** y se agregan la filas del **DataSet** original a la tabla **Customers** del **DataSet** clonado para aquellos clientes cuya columna **CountryRegion** tenga el valor "Germany".

```
Dim germanyCustomers As DataSet = customerDataSet.Clone()
Dim copyRows() As DataRow = _
```

```
  customerDataSet.Tables("Customers").Select("CountryRegion = 'Germany'")
Dim customerTable As DataTable = germanyCustomers.Tables("Customers")
Dim copyRow As DataRow
For Each copyRow In copyRows
  customerTable.ImportRow(copyRow)
Next
```

5.7. Control de Eventos del DataSet

El objeto DataSet proporciona tres eventos: Disposed, Initialized y MergeFailed.

Evento MergeFailed

El evento de uso más común del objeto **DataSet** es **MergeFailed**, que se inicia cuando los esquemas de los objetos **DataSet** que se están combinando entran en conflicto. Esto se produce cuando los objetos DataRow de origen y de destino tienen el mismo valor de clave principal y la propiedad EnforceConstraints se establece en **true**. Por ejemplo, si las columnas de clave principal de una tabla que se está combinando son las mismas entre las tablas de los dos objetos **DataSet**, se produce una excepción y se provoca el evento **MergeFailed**. El objeto MergeFailedEventArgs pasado como parámetro al evento **MergeFailed** tiene una propiedad Conflict que identifica el conflicto en el esquema entre los dos objetos **DataSet**, y una propiedad Table que identifica el nombre de la tabla en conflicto.

En el fragmento de código siguiente se muestra cómo agregar un controlador de eventos para el evento **MergeFailed**.

```
AddHandler workDS.MergeFailed, New MergeFailedEventHandler( _
  AddressOf DataSetMergeFailed)
Private Shared Sub DataSetMergeFailed(  _
  sender As Object,args As MergeFailedEventArgs)
  Console.WriteLine("Merge failed for table " & args.Table.TableName)
  Console.WriteLine("Conflict = " & args.Conflict)
End Sub
```

Evento Initialized

El evento Initialized se produce después de que el constructor de **DataSet** inicialice una nueva instancia del objeto **DataSet**.

La propiedad IsInitialized devuelve **true** si se ha completado la inicialización de **DataSet**; de lo contrario, devuelve **false**. El método BeginInit, que comienza la inicialización de **DataSet**, establece IsInitialized en **false**. El método EndInit, que finaliza la inicialización del objeto **DataSet**, lo establece en **true**. Estos métodos los utiliza el entorno de diseño de Visual Studio para inicializar un objeto **DataSet** que está siendo utilizado por otro componente. No los utilizará habitualmente en el código.

Evento Disposed

El objeto **DataSet** se deriva de la clase MarshalByValueComponent, que expone el método Dispose y el evento Disposed. El evento Disposed agrega un controlador de eventos para escuchar el evento eliminado en el componente. Puede usar el evento Disposed de un objeto **DataSet** si desea ejecutar código al llamar al método Dispose. Dispose libera los recursos usados por MarshalByValueComponent.

Nota:

Los objetos **DataSet** y **DataTable** heredan de MarshalByValueComponent y admiten la interfaz ISerializable para la comunicación remota. Éstos son los únicos objetos ADO.NET a los que se puede tener acceso remoto.

5.8. DataSets con Establecimiento de Tipos

Además del acceso en tiempo de ejecución a valores mediante variables con establecimiento flexible de tipos, el DataSet proporciona acceso a los datos mediante una metáfora con establecimiento inflexible de tipos. Se puede tener acceso a las tablas y columnas que forman parte del **DataSet** mediante nombres descriptivos y variables con establecimiento inflexible de tipos.

Un **DataSet** con información de tipos es una clase que se deriva de un **DataSet**. Como tal, hereda todos los métodos, eventos y propiedades de un **DataSet**. Además, un **DataSet** con información de tipos proporciona métodos, eventos y propiedades con establecimiento inflexible de tipos. Esto significa que se puede tener acceso a tablas y columnas por su nombre, en lugar de utilizar métodos de una colección. Además de la mayor legibilidad del código, un **DataSet** con información de tipos también permite que el editor de código Visual Studio .NET complete automáticamente las líneas mientras escribe.

Asimismo, el **DataSet** con establecimiento inflexible de tipos proporciona acceso a valores del tipo correcto en el momento de la compilación. Con un **DataSet** con establecimiento inflexible de tipos, los errores por no coincidencia de tipos se interceptan cuando se compila el código, no en tiempo de ejecución.

5.8.1. Generar Objetos DataSet con Establecimiento Inflexible de Tipos

A partir de un esquema XML que cumple con el estándar del lenguaje de definición de esquemas XML (XSD), es posible generar un objeto DataSet con establecimiento inflexible de tipos mediante la herramienta XSD.exe incluida en Kit de desarrollo de software de Windows (SDK).

En el siguiente código se muestra la sintaxis para generar un **DataSet** con esta herramienta.

```
xsd.exe /d /l:CS XSDSchemaFileName.xsd /eld /n:XSDSchema.Namespace
```

En esta sintaxis, la directiva /d indica a la herramienta que genere un **DataSet** y /l: le indica el lenguaje que debe utilizar (por ejemplo, C# o Visual Basic .NET). La directiva /eld opcional especifica que puede usar LINQ to DataSet para realizar consultas en el objeto **DataSet.** generado. Esta opción se utiliza cuando también se especifica la opción /d. La directiva /n: opcional indica a la herramienta que genere también un espacio de nombres para el **DataSet** denominado **XSDSchema.Namespace**. El resultado del comando es XSDSchemaFileName.cs, que se puede compilar y utilizar en una aplicación de ADO.NET. El código generado puede compilarse como una biblioteca o un módulo.

En el siguiente código se muestra la sintaxis para compilar el código generado como una biblioteca mediante el compilador de C# (csc.exe).

```
csc.exe /t:library XSDSchemaFileName.cs /r:System.dll /r:System.Data.dll
```

La directiva /t: indica a la herramienta que compile en una biblioteca mientras que las directivas /r: especifican bibliotecas dependientes necesarias para la compilación. El resultado del comando es XSDSchemaFileName.dll, que se puede pasar al compilador al compilar una aplicación de ADO.NET con la directiva /r:.

En el siguiente código se muestra la sintaxis para tener acceso al espacio de nombres pasado a XSD.exe en una aplicación de ADO.NET.

```
Imports XSDSchema.Namespace
```

En el siguiente ejemplo de código se utiliza un **DataSet** con información de tipos denominado **CustomerDataSet** para cargar una lista de clientes de la base de datos **Northwind**. Una vez cargados los datos mediante el método **Fill** , el ejemplo recorre cada cliente de la tabla **Customers** utilizando el objeto **CustomersRow** (**DataRow**) con información de tipos. Esto proporciona acceso directo a la columna **CustomerID**, en lugar de tener acceso a ella mediante **DataColumnCollection**.

```
Dim customers As CustomerDataSet= New CustomerDataSet()
Dim adapter As SqlDataAdapter New SqlDataAdapter( _
  "SELECT * FROM dbo.Customers;", _
  "Data Source=(local);Integrated " & _
  "Security=SSPI;Initial Catalog=Northwind")
adapter.Fill(customers, "Customers")
Dim customerRow As CustomerDataSet.CustomersRow
For Each customerRow In customers.Customers
  Console.WriteLine(customerRow.CustomerID)
Next
```

A continuación se muestra el esquema XML utilizado para el ejemplo.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="CustomerDataSet" xmlns=""
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-
microsoft-com:xml-msdata">
  <xs:element name="CustomerDataSet" msdata:IsDataSet="true">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="Customers">
          <xs:complexType>
            <xs:sequence>
          <xs:element name="CustomerID" type="xs:string" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

5.8.2. Anotar DataSets con Establecimiento de Tipos

Las anotaciones permiten modificar los nombres de los elementos del DataSet con información de tipos sin modificar el esquema subyacente. La modificación de los nombres de los elementos del esquema subyacente haría que el **DataSet** con información de tipos hiciera referencia a objetos que no existen en el origen de datos, además de perder una referencia a los objetos existentes en el origen de datos.

Con las anotaciones es posible personalizar los nombres de objetos del **DataSet** con información de tipos para utilizar nombres más significativos; esto hace que el código sea más legible y que los clientes puedan utilizar más fácilmente el **DataSet**, al tiempo que se deja intacto el esquema subyacente. Por

ejemplo, el siguiente elemento de esquema para la tabla **Customers** de la base de datos **Northwind** da como resultado un nombre de objeto **DataRow** de **CustomersRow** y una DataRowCollection denominada **Customers**.

```
<xs:element name="Customers">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CustomerID" type="xs:string" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Un nombre **Customers** para una **DataRowCollection** es significativo en el código de cliente, pero una **DataRow** denominada **CustomersRow** puede llevar a confusión porque se trata de un único objeto. Además, en situaciones normales, se haría referencia al objeto sin el identificador **Row** y en su lugar se haría referencia al mismo simplemente como un objeto **Customer**. La solución consiste en anotar el esquema e identificar nuevos nombres para los objetos **DataRow** y **DataRowCollection**. A continuación se muestra la versión anotada del esquema anterior.

```
<xs:element name="Customers" codegen:typedName="Customer"
codegen:typedPlural="Customers">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CustomerID" type="xs:string" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Al especificar un valor **Customer** para **typedName**, el nombre de un objeto **DataRow** será **Customer**. Si se especifica el valor **Customers** para **typedPlural** se conservará el nombre **Customers** para **DataRowCollection**.

En la siguiente tabla se muestran las anotaciones disponibles.

Anotación	Descripción
typedName	Nombre del objeto.
typedPlural	Nombre de una colección de objetos.
typedParent	Nombre del objeto cuando se hace referencia al mismo en una relación primaria.
typedChildren	Nombre del método para devolver objetos de una relación secundaria.
nullValue	Valor si el valor subyacente es **DBNull**. Vea las anotaciones **nullValue** en la tabla siguiente. El valor predeterminado es **_throw**.

En la tabla siguiente se presentan los valores que se pueden especificar para la anotación **nullValue**.

Valor nullValue	Descripción
Valor de reemplazo	Especifica un valor que se va a devolver. El valor devuelto debe coincidir con el tipo del elemento. Por ejemplo, utilice nullValue="0" para devolver 0 en el caso de

	campos null integer.
_throw	Iniciar una excepción. Éste es el valor predeterminado.
_null	Devuelve una referencia nula o inicia una excepción si se encuentra un tipo primitivo.
_empty	En el caso de cadenas, devuelve **String.Empty**; de lo contrario, devuelve un objeto creado desde un constructor vacío. Si se encuentra un tipo primitivo, inicia una excepción.

En la siguiente tabla se muestran los valores predeterminados de los objetos en un **DataSet** con información de tipos y las anotaciones disponibles.

Objeto/Método/Evento	Default	Anotación
DataTable	TableNameDataTable	typedPlural
Métodos **DataTable**	NewTableNameRow AddTableNameRow DeleteTableNameRow	typedName
DataRowCollection	TableName	typedPlural
DataRow	TableNameRow	typedName
DataColumn	DataTable.ColumnNameColumn DataRow.ColumnName	typedName
Propiedad	PropertyName	typedName
Descriptor de acceso **Child**	GetChildTableNameRows	typedChildren
Descriptor de acceso **Parent**	TableNameRow	typedParent
Eventos **DataSet**	TableNameRowChangeEvent TableNameRowChangeEventHandler	typedName

Para utilizar anotaciones de **DataSet** con información de tipos debe incluir la siguiente referencia **xmlns** en el esquema del lenguaje de definición de esquemas XML (XSD).

```
xmlns:codegen="urn:schemas-microsoft-com:xml-msprop"
```

A continuación se muestra un ejemplo de esquema anotado que expone la tabla **Customers** de la base de datos **Northwind** incluyendo una relación con la tabla **Orders**.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="CustomerDataSet"
      xmlns:codegen="urn:schemas-microsoft-com:xml-msprop"
      xmlns=""
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="CustomerDataSet" msdata:IsDataSet="true">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="Customers" codegen:typedName="Customer"
codegen:typedPlural="Customers">
```

```
          <xs:complexType>
            <xs:sequence>
              <xs:element name="CustomerID"
codegen:typedName="CustomerID" type="xs:string" minOccurs="0" />
              <xs:element name="CompanyName"
codegen:typedName="CompanyName" type="xs:string" minOccurs="0" />
              <xs:element name="Phone" codegen:typedName="Phone"
codegen:nullValue="" type="xs:string" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Orders" codegen:typedName="Order"
codegen:typedPlural="Orders">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="OrderID" codegen:typedName="OrderID"
type="xs:int" minOccurs="0" />
              <xs:element name="CustomerID"
codegen:typedName="CustomerID"
                 codegen:nullValue="" type="xs:string" minOccurs="0" />
              <xs:element name="EmployeeID"
codegen:typedName="EmployeeID" codegen:nullValue="0"
type="xs:int" minOccurs="0" />
              <xs:element name="OrderAdapter"
codegen:typedName="OrderAdapter"
codegen:nullValue="1980-01-01T00:00:00"
                 type="xs:dateTime" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
    <xs:unique name="Constraint1">
      <xs:selector xpath=".//Customers" />
      <xs:field xpath="CustomerID" />
    </xs:unique>
    <xs:keyref name="CustOrders" refer="Constraint1"
codegen:typedParent="Customer" codegen:typedChildren="GetOrders">
      <xs:selector xpath=".//Orders" />
      <xs:field xpath="CustomerID" />
    </xs:keyref>
  </xs:element>
</xs:schema>
```

En el siguiente ejemplo de código se utiliza un **DataSet** con establecimiento inflexible de tipos creado a partir del esquema de ejemplo. Utiliza un SqlDataAdapter para rellenar la tabla **Customers** y otro SqlDataAdapter para rellenar la tabla **Orders**. El **DataSet** con establecimiento inflexible de tipos define las **DataRelations**.

```
' Assumes a valid SqlConnection object named connection.
Dim customerAdapter As SqlDataAdapter = New SqlDataAdapter( _
```

```
    "SELECT CustomerID, CompanyName, Phone FROM Customers", &
    connection)
Dim orderAdapter As SqlDataAdapter = New SqlDataAdapter( _
    "SELECT OrderID, CustomerID, EmployeeID, OrderAdapter FROM Orders", &
    connection)
' Populate a strongly typed DataSet.
connection.Open()
Dim customers As CustomerDataSet = New CustomerDataSet()
customerAdapter.Fill(customers, "Customers")
orderAdapter.Fill(customers, "Orders")
connection.Close()
' Add a strongly typed event.
AddHandler customers.Customers.CustomerChanged, &
    New CustomerDataSet.CustomerChangeEventHandler( _
    AddressOf OnCustomerChanged)
' Add a strongly typed DataRow.
Dim newCustomer As CustomerDataSet.Customer = _
    customers.Customers.NewCustomeromer()
newCustomer.CustomerID = "NEW01"
newCustomer.CompanyName = "My New Company"
customers.Customers.AddCustomer(newCustomer)
' Navigate the child relation.
Dim customer As CustomerDataSet.Customer
Dim order As CustomerDataSet.Order
For Each customer In customers.Customers
  Console.WriteLine(customer.CustomerID)
  For Each order In customer.GetOrders()
    Console.WriteLine(vbTab & order.OrderID)
  Next
Next
Private Shared Sub OnCustomerChanged( _
    sender As Object, e As CustomerDataSet.CustomerChangeEvent)
End Sub
```

5.9. DataTables

Un objeto DataSet está formado por una colección de tablas, relaciones y restricciones. En ADO .NET, los objetos DataTable se utilizan para representar las tablas de un **DataSet**. Un objeto **DataTable** representa una tabla de datos relacionales de la memoria; los datos son locales de la aplicación basada en .NET en la que residen, pero se pueden llenar desde un origen de datos como Microsoft SQL Server mediante un **DataAdapter**.

La clase **DataTable** es miembro del espacio de nombres **System.Data** dentro de la biblioteca de clases de .NET Framework. Se puede crear y utilizar **DataTable** de manera independiente o como miembro de un **DataSet** y los objetos **DataTable** se pueden utilizar también en combinación con otros objetos de .NET Framework, incluido DataView. Al conjunto de tablas de un **DataSet** se puede tener acceso mediante la propiedad **Tables** del objeto **DataSet**.

El esquema o la estructura de una tabla se representan mediante columnas y restricciones. El esquema de una **DataTable** se define mediante objetos DataColumn, ForeignKeyConstraint y UniqueConstraint.

Las columnas de una tabla se pueden asignar a columnas de un origen de datos, pueden contener valores calculados de expresiones, aumentar sus valores automáticamente o contener valores de clave principal.

Además del esquema, un objeto **DataTable** debe tener también filas en las que albergar y ordenar los datos. La clase DataRow representa los datos reales que contiene una tabla. La clase **DataRow**, sus propiedades y métodos se utilizan para recuperar, evaluar y manipular los datos de una tabla. Cuando se tiene acceso a los datos de una fila y se cambian, el objeto **DataRow** mantiene tanto su estado actual como el original.

Se pueden crear relaciones primarias-secundarias entre tablas utilizando una o varias columnas relacionadas de las tablas. Se pueden crear relaciones entre objetos **DataTable** mediante un objeto DataRelation. Los objetos **DataRelation** se pueden utilizar después para devolver las filas relacionadas, secundaria o primaria, de una fila concreta.

5.9.1. Crear un DataTable

Un objeto DataTable, que representa una tabla de datos relacionales en la memoria, se puede crear y usar de manera independiente o lo pueden usar otros objetos de .NET Framework, normalmente como miembro de un objeto DataSet.

Puede crear un objeto **DataTable** mediante el constructor **DataTable** adecuado. Puede agregarlo al **DataSet** mediante el método **Add** que lo agregará a la colección **Tables** del objeto **DataTable**.

También se pueden crear objetos **DataTable** dentro de un **DataSet** mediante los métodos **Fill** o **FillSchema** del objeto **DataAdapter** o desde un esquema XML predefinido o deducido, mediante los métodos **ReadXml**, **ReadXmlSchema** o **InferXmlSchema** del **DataSet**. Tenga en cuenta que una vez que se ha agregado **DataTable** como miembro de la colección **Tables** de un **DataSet**, no se puede agregar a la colección de tablas de ningún otro **DataSet**.

La primera vez que se crea un **DataTable**, no tiene esquema (estructura). Para definir el esquema de la tabla, es necesario crear objetos DataColumn y agregarlos a la colección **Columns** de la tabla. También se puede definir una columna de claves principales para la tabla y crear objetos **Constraint** y agregarlos a la colección **Constraints** de la tabla. Una vez que se ha definido el esquema de **DataTable**, se pueden agregar filas de datos a la tabla, agregando objetos **DataRow** a la colección **Rows** de la tabla.

No es necesario proporcionar un valor para la propiedad TableName cuando se crea una **DataTable**: dicha propiedad se puede especificar en otro momento o se puede dejar vacía. Sin embargo, cuando se agrega una tabla sin valor **TableName** a un **DataSet**, la tabla recibirá un nombre predeterminado incremental con el formato Table*N* y comenzando con "Table" para Table0.

Nota:

Es aconsejable evitar la convención de nomenclatura "Table*N*" al proporcionar un valor **TableName**, ya que el nombre proporcionado podría entrar en conflicto con un nombre de tabla predeterminado existente en el **DataSet**. Si el nombre proporcionado ya existe, se inicia una excepción.

En el ejemplo siguiente se crea una instancia de un objeto **DataTable**, a la que se asigna el nombre "Customers".

```
Dim workTable as DataTable = New DataTable("Customers")
```

En el siguiente ejemplo se crea una instancia de una **DataTable** agregándola a la colección **Tables** de un **DataSet**.

```
Dim customers As DataSet = New DataSet
Dim customersTable As DataTable = customers.Tables.Add("CustomersTable")
```

5.9.2. Definición de Esquema de un DataTable

El esquema, o estructura, de una tabla se representa con columnas y restricciones. El esquema de una DataTable se define mediante objetos DataColumn, ForeignKeyConstraint y UniqueConstraint. Las columnas de una tabla se pueden asignar a columnas de un origen de datos, pueden contener valores calculados de expresiones, aumentar sus valores automáticamente o contener valores de clave principal.

Las referencias a los nombres de columnas, relaciones y restricciones de una tabla hacen distinción entre mayúsculas y minúsculas. En una tabla puede haber dos o más columnas, relaciones y restricciones con el mismo nombre, pero con distinción entre mayúsculas y minúsculas. Por ejemplo, se puede tener **Col1** y **col1**. En este caso, una referencia al nombre de una de las columnas tiene que coincidir exactamente con las mayúsculas y minúsculas del nombre de la columna, de lo contrario se inicia una excepción. Por ejemplo, si la tabla **myTable** contiene las columnas **Col1** y **col1**, la referencia al nombre de **Col1** sería **myTable.Columns["Col1"]** y al de **col1** sería **myTable.Columns["col1"]**. Si se intentara hacer referencia a cualquiera de las columnas mediante **myTable.Columns["COL1"]** se generaría una excepción.

La regla de distinción entre mayúsculas y minúsculas no se aplica si sólo hay una columna, relación o restricción con un nombre concreto. Es decir, si no hay ningún otro objeto de columna, relación o restricción en la tabla que coincida con el nombre de ese objeto de columna, relación o restricción concreto, se puede hacer referencia al nombre del objeto utilizando cualquier mayúscula o minúscula y no se generará una excepción. Por ejemplo, si la tabla sólo tiene **Col1**, se puede hacer referencia al nombre usando **my.Columns["COL1"]**.

Nota:

La propiedad CaseSensitive de la **DataTable** no afecta a este comportamiento. La propiedad **CaseSensitive** se aplica a los datos de una tabla y afecta al ordenamiento, la búsqueda, el filtrado, la aplicación de restricciones, etcétera, pero no afecta a referencias a columnas, relaciones ni restricciones.

5.9.2.1. Agregar Columnas al DataTable

DataTable contiene una colección de objetos DataColumn a los que hace referencia la propiedad **Columns** de la tabla. Esta colección de columnas, junto con las restricciones que haya, define el esquema, o estructura, de la tabla.

Los objetos **DataColumn** de una tabla se crean con el constructor **DataColumn** o llamando al método **Add** de la propiedad **Columns** de la tabla, que es una DataColumnCollection. El método **Add** acepta argumentos **ColumnName**, **DataType** y **Expression** opcionales y crea una nueva **DataColumn** como miembro de la colección. También acepta un objeto **DataColumn** existente y lo agrega a la colección y devuelve una referencia a la **DataColumn** agregada si se solicita. Puesto que los objetos **DataTable** no son específicos de ningún origen de datos, al especificar el tipo de datos de una **DataColumn** se usan los tipos de .NET Framework.

En el siguiente ejemplo, se agregan cuatro columnas a **DataTable**.

```
Dim workTable As DataTable = New DataTable("Customers")
Dim workCol As DataColumn = workTable.Columns.Add( _
    "CustID", Type.GetType("System.Int32"))
workCol.AllowDBNull = false
workCol.Unique = true
workTable.Columns.Add("CustLName", Type.GetType("System.String"))
workTable.Columns.Add("CustFName", Type.GetType("System.String"))
workTable.Columns.Add("Purchases", Type.GetType("System.Double"))
```

En el ejemplo, advierta que las propiedades de la columna **CustID** se configuran de manera que no admitan valores **DBNull** y restrinjan los valores para que sean únicos. Sin embargo, si se define la columna **CustID** como columna de clave principal de la tabla, la propiedad **AllowDBNull** se establecerá automáticamente en **false** y la propiedad **Unique** se establecerá automáticamente en **true**.

Precaución:

Si no se proporciona un nombre para una determinada columna, ésta recibe el nombre predeterminado incremental Column*N,* (que empieza por "Column1"), cuando la columna se agrega a la colección **DataColumnCollection**. Se recomienda evitar la convención de nomenclatura "Column*N*" al proporcionar un nombre de columna, ya que el nombre que se proporcione podría entrar en conflicto con un nombre de columna predeterminado existente en **DataColumnCollection**. Si el nombre proporcionado ya existe, se inicia una excepción.

5.9.2.2. Crear Columnas de Expresión

Se puede definir una expresión para una columna, a fin de que pueda contener un valor calculado a partir de los valores de otra columna de la misma fila o de los valores de columna de varias filas de la tabla. Para definir la expresión que se va a evaluar, utilice la propiedad Expression de la columna de destino y la propiedad ColumnName para hacer referencia a otras columnas en la expresión. El DataType para la columna de expresión debe ser adecuado para el valor que dicha expresión devuelve.

En la tabla siguiente se enumeran varios usos posibles de las columnas de expresión de una tabla.

Tipo de expresión	Ejemplo
Comparación	"Total >= 500"
Cálculo	"UnitPrice * Quantity"
Agregación	Sum(Precio)

Se puede establecer la propiedad **Expression** en un objeto **DataColumn** existente, o se puede incluir la propiedad como el tercer argumento que se pasa al constructor DataColumn, como se muestra en el ejemplo siguiente.

```
workTable.Columns.Add("Total",Type.GetType("System.Double"))
workTable.Columns.Add("SalesTax", Type.GetType("System.Double"), _
  "Total * 0.086")
```

Las expresiones pueden hacer referencia a otras columnas de expresión; sin embargo, una referencia circular, en la que dos expresiones se hacen referencia una a otra, generará una excepción. Para obtener las reglas de escritura de expresiones, vea la propiedad Expression de la clase **DataColumn**.

5.9.2.3. Crear Columnas AutoIncrement

Para garantizar que los valores de una columna son únicos, éstos se pueden establecer de manera que se incrementen automáticamente cuando se agregan filas a la tabla. Para crear una DataColumn que se incrementa automáticamente, establezca la propiedad AutoIncrement de la columna en **true**. La DataColumn comienza entonces con el valor definido en la propiedad AutoIncrementSeed y, con cada fila agregada, el valor de la columna **AutoIncrement** aumenta en función del valor definido en la propiedad AutoIncrementStep de la columna.

En el caso de las columnas **AutoIncrement**, se recomienda que la propiedad ReadOnly de la **DataColumn** se establezca en **true**.

En el ejemplo siguiente se muestra cómo se crea una columna que comienza con un valor de 200 y va aumentando de tres en tres.

```
Dim workColumn As DataColumn = workTable.Columns.Add( _
    "CustomerID", typeof(Int32))
workColumn.AutoIncrement = true
workColumn.AutoIncrementSeed = 200
workColumn.AutoIncrementStep = 3
```

5.9.2.4. Definir Claves Principales

Generalmente, una tabla de base de datos tiene una columna o grupo de columnas que identifican de manera exclusiva cada fila de la tabla. Esta columna o grupo de columnas de identificación se denomina clave principal.

Al identificar una única DataColumn como la PrimaryKey para una DataTable, la tabla establece automáticamente la propiedad AllowDBNull de la columna como **false** y la propiedad Unique como **true**. Para las claves principales de varias columnas sólo se establece de forma automática la propiedad **AllowDBNull** en **false**.

La propiedad **PrimaryKey** de una DataTable recibe como valor una matriz de uno o varios objetos **DataColumn**, como se muestra en los ejemplos siguientes. En el primer ejemplo se define una sola columna como clave principal.

```
workTable.PrimaryKey = New DataColumn() {workTable.Columns("CustID")}
' Or
Dim columns(1) As DataColumn
columns(0) = workTable.Columns("CustID")
workTable.PrimaryKey = columns
```

En el siguiente ejemplo se definen dos columnas como clave principal.

```
workTable.PrimaryKey = New DataColumn() {workTable.Columns("CustLName"),
workTable.Columns("CustFName")}
' Or
Dim keyColumn(2) As DataColumn
keyColumn(0) = workTable.Columns("CustLName")
keyColumn(1) = workTable.Columns("CustFName")
workTable.PrimaryKey = keyColumn
```

5.9.2.5. Restricciones del DataTable

Se pueden utilizar restricciones para exigir restricciones sobre los datos de un objeto DataTable con el fin de mantener la integridad de los datos. Una restricción es una regla automática que se aplica a una columna, o a varias columnas relacionadas, que determina cómo proceder cuando se modifica de alguna manera el valor de una fila. Las restricciones se exigen cuando la propiedad EnforceConstraints del DataSet es **true**.

Existen dos tipos de restricciones en ADO.NET: ForeignKeyConstraint y UniqueConstraint. De forma predeterminada, las dos restricciones se crean automáticamente al crear una relación entre dos o más tablas agregando DataRelation al **DataSet**. Sin embargo, se puede deshabilitar este comportamiento especificando **createConstraints** = **false** al crear la relación.

⊟ ForeignKeyConstraint

ForeignKeyConstraint exige reglas sobre cómo se propagan las actualizaciones y eliminaciones a las tablas relacionadas. Por ejemplo, si se actualiza o elimina el valor de una fila de una tabla y el mismo valor también se utiliza en una o varias tablas relacionadas, **ForeignKeyConstraint** determinará qué sucede en las tablas relacionadas.

Las propiedades DeleteRule y UpdateRule de **ForeignKeyConstraint** definen la acción que se ha de realizar cuando el usuario intente eliminar o actualizar una fila en una tabla relacionada. En la tabla siguiente se describen las distintas configuraciones disponibles para las propiedades **DeleteRule** y **UpdateRule** de **ForeignKeyConstraint**

Establecimiento de reglas	Descripción
Cascade	Elimina o actualiza las filas relacionadas.
SetNull	Establece los valores de las filas relacionadas en **DBNull**.
SetDefault	Establece los valores de las filas relacionadas en el valor predeterminado.
Ninguna	No realiza ninguna acción en las filas relacionadas. Éste es el valor predeterminado.

ForeignKeyConstraint puede restringir, además de propagar, los cambios en las columnas relacionadas. Dependiendo de las propiedades establecidas para la **ForeignKeyConstraint** de una columna, y si la propiedad **EnforceConstraints** del **DataSet** es **true**, realizar ciertas operaciones en la fila primaria generará una excepción. Por ejemplo, si la propiedad **DeleteRule** de **ForeignKeyConstraint** es **None**, una fila primaria no se puede eliminar si tiene filas secundarias.

Se puede crear una restricción de clave externa entre columnas individuales o entre una matriz de columnas utilizando el constructor **ForeignKeyConstraint.** Pase el objeto **ForeignKeyConstraint** resultante al método **Add** de la propiedad **Constraints** de la tabla, que es una **ConstraintCollection**. También puede pasar argumentos de constructor a varias sobrecargas del método **Add** de **ConstraintCollection** para crear una **ForeignKeyConstraint.**

Al crear una **ForeignKeyConstraint**, se pueden pasar los valores **DeleteRule** y **UpdateRule** al constructor como argumentos, o se pueden configurar como propiedades, como en el ejemplo siguiente (en que el valor **DeleteRule** se establece como **None**).

```
Dim custOrderFK As ForeignKeyConstraint = New
ForeignKeyConstraint("CustOrderFK", _
  custDS.Tables("CustTable").Columns("CustomerID"), _
  custDS.Tables("OrdersTable").Columns("CustomerID"))
custOrderFK.DeleteRule = Rule.None
' Cannot delete a customer value that has associated existing orders.
custDS.Tables("OrdersTable").Constraints.Add(custOrderFK)
```

AcceptRejectRule

Los cambios realizados en filas se pueden aceptar con el método **AcceptChanges** o cancelar con el método **RejectChanges** de **DataSet**, **DataTable** o **DataRow**. Si un **DataSet** contiene **ForeignKeyConstraints**, al invocar los métodos **AcceptChanges** o **RejectChanges** se exige **AcceptRejectRule**. La propiedad **AcceptRejectRule** de **ForeignKeyConstraint** determina qué acción se tomará en las filas secundarias cuando se llame a **AcceptChanges** o **RejectChanges** en la fila primaria.

En la siguiente tabla se incluyen los valores disponibles para **AcceptRejectRule**.

Establecimiento de reglas	Descripción
Cascade	Acepta o rechaza los cambios en filas secundarias.
Ninguna	No realiza ninguna acción en las filas secundarias. Éste es el valor predeterminado.

⊟ UniqueConstraint

El objeto **UniqueConstraint**, que se puede asignar a una sola columna o a una matriz de columnas de una **DataTable**, garantiza que todos los datos de las columnas especificadas son únicos en cada fila. Se puede crear una restricción única para una columna o matriz de columnas con el constructor **UniqueConstraint**. Pase el objeto **UniqueConstraint** resultante al método **Add** de la propiedad **Constraints** de la tabla, que es una **ConstraintCollection**. También puede pasar argumentos de constructor a varias sobrecargas del método **Add** de una **ConstraintCollection** para crear una **UniqueConstraint.** Al crear una **UniqueConstraint** para una columna o columnas, se puede especificar opcionalmente si la columna o columnas son una clave principal.

También se puede crear una restricción única para una sola columna estableciendo su propiedad **Unique** en **true**. Por otra parte, si se establece la propiedad **Unique** de una sola columna en **false** se quita cualquier restricción única que pudiera existir. Si se define una o varias columnas como clave principal de una tabla se creará automáticamente una restricción única para la columna o columnas especificadas. Si quita una columna mediante la propiedad **PrimaryKey** de una **DataTable**, se quita la **UniqueConstraint**.

En el ejemplo siguiente se crea una **UniqueConstraint** para dos columnas de una **DataTable**.

```
Dim custTable As DataTable = custDS.Tables("Customers")
Dim custUnique As UniqueConstraint = New UniqueConstraint _
(New DataColumn() {custTable.Columns("CustomerID"), _
custTable.Columns("CompanyName")})
custDS.Tables("Customers").Constraints.Add(custUnique)
```

5.9.3. Manipular Datos en el DataTable

Después de crear una DataTable en un DataSet, se pueden realizar las mismas actividades que al utilizar una tabla de una base de datos. Se puede agregar, ver, modificar y eliminar datos en la tabla, supervisar los errores y eventos y consultar los datos de la tabla. Al modificar los datos de una **DataTable**, también se puede comprobar si los cambios son precisos y determinar si aceptarlos o rechazarlos mediante programación.

5.9.3.1. Agregar Datos al DataTable

Después de crear una DataTable y definir su estructura usando columnas y restricciones, se le pueden agregar nuevas filas de datos. Para agregar una nueva fila, declare una nueva variable como tipo DataRow. Se devuelve un nuevo objeto **DataRow** cuando se llama al método NewRow. A continuación, la **DataTable** crea el objeto **DataRow** basándose en la estructura de la tabla, definida por la DataColumnCollection.

En el ejemplo siguiente se muestra cómo se crea una nueva fila llamando al método **NewRow**.

```
Dim workRow As DataRow = workTable.NewRow()
```

A continuación, la fila recién agregada se puede manipular mediante un índice o nombre de columna, como se muestra en el siguiente ejemplo.

```
workRow("CustLName") = "Smith"
workRow(1) = "Smith"
```

Una vez que se han insertado datos en la nueva fila, se utiliza el método **Add** para agregar la fila a la DataRowCollection, como se muestra en el siguiente código.

```
workTable.Rows.Add(workRow)
```

También se puede llamar al método **Add** para agregar una nueva fila pasando una matriz de valores, con el tipo Object, tal y como se muestra en el ejemplo siguiente.

```
workTable.Rows.Add(new Object() {1, "Smith"})
```

Si se pasa una matriz de objetos con el tipo **Object** al método **Add**, se crea una nueva fila dentro de la tabla y se establece sus valores de columna en los valores de la matriz de objetos. Tenga en cuenta que los valores de la matriz se hacen coincidir en secuencia con las columnas, basándose en el orden en que aparecen en la tabla.

En el siguiente ejemplo se agregan diez filas a la tabla **Customers** recién creada.

```
Dim workRow As DataRow
Dim i As Integer
For i = 0 To 9
  workRow = workTable.NewRow()
  workRow(0) = i
  workRow(1) = "CustName" & I.ToString()
  workTable.Rows.Add(workRow)
Next
```

5.9.3.2. Ver Datos en un DataTable

Puede tener acceso al contenido de una DataTable mediante las colecciones **Rows** y **Columns** de la **DataTable**. Se puede utilizar también el método Select()()() para devolver subconjuntos de los datos de una **DataTable** según ciertos criterios como, por ejemplo, criterios de búsqueda, orden de clasificación y

estado de la fila. Además, se puede utilizar el método Find de una **DataRowCollection** cuando se busque una fila determinada mediante el valor de una clave principal.

El método **Select** del objeto **DataTable** devuelve un conjunto de objetos DataRow que cumplen con los criterios especificados. **Select** toma argumentos opcionales de una expresión de filtro, expresión de ordenación y **DataViewRowState**. La expresión de filtro identifica qué filas se van a devolver basándose en valores de **DataColumn** como, por ejemplo, LastName = 'Smith'. La expresión de ordenación sigue convenciones SQL estándar para ordenar columnas, por ejemplo LastName ASC, FirstName ASC. Para obtener las reglas de escritura de expresiones, vea la propiedad Expression de la clase **DataColumn**.

Sugerencia:

Cuando se realizan varias llamadas al método **Select** de una **DataTable**, se puede aumentar el rendimiento mediante la creación de una DataView para la **DataTable**. Al crear la **DataView**, las filas de la tabla se colocan en un índice. A continuación, el método **Select** utiliza ese índice, lo que reduce considerablemente el tiempo necesario para generar el resultado de la consulta.

El método **Select** determina qué versión de las filas se debe ver o manipular, según un DataViewRowState. En la siguiente tabla se recogen los posibles valores de la enumeración **DataViewRowState**.

Valor DataViewRowState	Descripción
CurrentRows	Filas actuales, incluidas las filas sin modificar, agregadas y modificadas.
Deleted	Una fila eliminada.
ModifiedCurrent	Una versión actual, que es una versión modificada de los datos originales. (Vea **ModifiedOriginal**.)
ModifiedOriginal	Versión original de todas las filas modificadas. La versión actual está disponible utilizando **ModifiedCurrent**.
Added	Una fila nueva.
Ninguna	Ninguna.
OriginalRows	Filas originales, incluidas las filas sin modificar y las eliminadas.
Unchanged	Una fila sin modificar.

En el ejemplo siguiente, el objeto **DataSet** se filtra de manera que se trabaje sólo con las filas cuyo **DataViewRowState** está establecido en **CurrentRows**.

```
Dim column As DataColumn
Dim row As DataRow
Dim currentRows() As DataRow = _
    workTable.Select(Nothing, Nothing, DataViewRowState.CurrentRows)
If (currentRows.Length < 1 ) Then
  Console.WriteLine("No Current Rows Found")
Else
  For Each column in workTable.Columns
    Console.Write(vbTab & column.ColumnName)
  Next
```

```
  Console.WriteLine(vbTab & "RowState")
  For Each row In currentRows
    For Each column In workTable.Columns
      Console.Write(vbTab & row(column).ToString())
    Next
    Dim rowState As String = _
        System.Enum.GetName(row.RowState.GetType(), row.RowState)
    Console.WriteLine(vbTab & rowState)
  Next
End If
```

El método **Select** se puede utilizar para devolver filas con valores de **RowState** o valores de campo distintos. En el ejemplo siguiente se devuelve una matriz **DataRow** que hace referencia a todas las filas que se han eliminado y se devuelve otra matriz **DataRow** que hace referencia a todas las filas, ordenadas por **CustLName**, donde la columna **CustID** es mayor que 5.

```
' Retrieve all deleted rows.
Dim deletedRows() As DataRow = workTable.Select(Nothing, Nothing,
DataViewRowState.Deleted)
' Retrieve rows where CustID > 5, and order by CustLName.
Dim custRows() As DataRow = workTable.Select( _
    "CustID > 5", "CustLName ASC")
```

5.9.3.3. El Método Load

Se puede utilizar el método Load para cargar una DataTable con filas desde un origen de datos. Se trata de un método sobrecargado que, en su formato más sencillo, acepta un único parámetro, un **DataReader**. En este formato, simplemente se cargan filas en la **DataTable**. Si lo desea, puede especificar el parámetro **LoadOption** para controlar el modo en que se agregan los datos a la **DataTable**.

El parámetro **LoadOption** resulta especialmente útil en aquellos casos en los que la **DataTable** ya contiene filas de datos, porque describe como se combinan los datos que provienen del origen de datos con los datos que ya existentes en la tabla. Por ejemplo, **PreserveCurrentValues** (predeterminado) especifica que en aquellos casos en los que una fila se marque como **Added** en la **DataTable**, el valor **Original** de cada columna se establece en el valor del contenido de la fila coincidente del origen de datos. El valor **Current** conserva los valores asignados al agregarse la fila y el **RowState** de la fila se establece en **Changed**.

En la siguiente tabla se ofrece una breve descripción de los valores de enumeración LoadOption.

Valor LoadOption	Descripción
OverwriteRow	Si las filas entrantes tienen el mismo valor **PrimaryKey** que una fila ya existente en la **DataTable**, los valores **Original** y **Current** de cada columna se sustituyen por los valores de la fila entrante y la propiedad **RowState** se establece en **Unchanged**. Las filas del origen de datos que aún no existen en la **DataTable** se agregan con un valor **RowState** de **Unchanged**. Esta opción activa actualiza el contenido de la **DataTable**, para que coincida con el del origen de datos.
PreserveCurrentValues	Si las filas entrantes tienen el mismo valor **PrimaryKey** que una

(predeterminado)	fila ya existente en la **DataTable**, el valor **Original** se establece en el valor del contenido de la fila entrante y no se modifica el valor **Current**. Si el **RowState** es **Added** o **Modified**, se establece en **Modified**. Si el **RowState** fue **Deleted**, sigue siendo **Deleted**. Las filas del origen de datos que aún no existen en la **DataTable** se agregan y el **RowState** se establece en **Unchanged**.
UpdateCurrentValues	Si las filas entrantes tienen el mismo valor **PrimaryKey** que la fila existente en la **DataTable**, se copia el valor **Current** al valor **Original** y, a continuación, el valor **Current** se establece en el contenido de la fila entrante. Si el **RowState** de la **DataTable** fue **Added**, el **RowState** sigue siendo **Added**. Las filas marcadas como **Modified** o **Deleted**, el **RowState** es **Modified**. Las filas del origen de datos que aún no existen en la **DataTable** se agregan y el **RowState** se establece en **Added**.

En el siguiente ejemplo se utiliza el método **Load** para mostrar una lista de cumpleaños de los empleados de la base de datos **Northwind**.

```
Private Sub LoadBirthdays(ByVal connectionString As String)
    ' Assumes that connectionString is a valid connection string
    ' to the Northwind database on SQL Server.
    Dim queryString As String = _
    "SELECT LastName, FirstName, BirthDate " & _
      " FROM dbo.Employees " & _
      "ORDER BY BirthDate, LastName, FirstName"
    ' Open and fill a DataSet.
    Dim adapter As SqlDataAdapter = New SqlDataAdapter( _
        queryString, connectionString)
    Dim employees As New DataSet
    adapter.Fill(employees, "Employees")
    ' Create a SqlDataReader for use with the Load Method.
    Dim reader As DataTableReader = employees.GetDataReader()
    ' Create an instance of DataTable and assign the first
    ' DataTable in the DataSet.Tables collection to it.
    Dim dataTableEmp As DataTable = employees.Tables(0)
    ' Fill the DataTable with data by calling Load and
    ' passing the SqlDataReader.
    dataTableEmp.Load(reader, LoadOption.OverwriteRow)
    ' Loop through the rows collection and display the values
    ' in the console window.
    Dim employeeRow As DataRow
    For Each employeeRow In dataTableEmp.Rows
        Console.WriteLine("{0:MM\\dd\\yyyy}" & ControlChars.Tab & _
          "{1}, {2}", _
          employeeRow("BirthDate"), _
          employeeRow("LastName"), _
          employeeRow("FirstName"))
    Next employeeRow
    ' Keep the window opened to view the contents.
    Console.ReadLine()
End Sub
```

5.9.3.4. Editar el DataTable

Cuando se cambian los valores de las columnas en una DataRow, los cambios se sitúan inmediatamente en el estado actual de la fila. A continuación, el DataRowState se establece en **Modified** y los cambios se aceptan o se rechazan mediante los métodos AcceptChanges o RejectChanges de la **DataRow**. **DataRow** proporciona también tres métodos que se pueden usar para suspender el estado de la fila mientras se está editando. Estos métodos son BeginEdit, EndEdit y CancelEdit.

Cuando se modifican los valores de columna de una **DataRow** directamente, **DataRow** administra los valores de columna mediante las versiones de fila **Current**, **Default** y **Original**. Además de estas versiones de fila, los métodos **BeginEdit**, **EndEdit** y **CancelEdit** utilizan una cuarta versión de fila: **Proposed**.

La versión de fila **Proposed** propuesta existe mientras dura una operacion de edición que se inicia llamando a **BeginEdit** y finaliza mediante **EndEdit** o **CancelEdit,** o bien llamando a **AcceptChanges** o **RejectChanges**.

Durante la operación de edición se puede aplicar la lógica de validación a las columnas individualmente evaluando el **ProposedValue** del evento **ColumnChanged** de la **DataTable**. El evento **ColumnChanged** contiene **DataColumnChangeEventArgs** que guardan una referencia a la columna que se está cambiando y al **ProposedValue**. Después de evaluar el valor propuesto, se puede modificar o cancelar la edición. Cuando termina la operación de edición, la fila abandona el estado de **Proposed**.

Las modificaciones se pueden confirmar llamando a **EndEdit** o cancelar llamando a **CancelEdit**. Tenga en cuenta que aunque **EndEdit** confirma las modificaciones, **DataSet** no las acepta realmente hasta que se llama a **AcceptChanges**. También hay que tener en cuenta que si se llama a **AcceptChanges** antes de finalizar la operación de edición con **EndEdit** o **CancelEdit**, dicha edición finaliza y los valores de fila **Proposed** se aceptan en las dos versiones de fila: **Current** y **Original**. De la misma manera, si se llama a **RejectChanges**, se finaliza la edición y se descartan las versiones de fila **Current** y **Proposed**. Una llamada a **EndEdit** o **CancelEdit** después de llamar a **AcceptChanges** o **RejectChanges** no tiene ningún efecto porque la edición ya ha finalizado.

En el ejemplo siguiente se muestra cómo se utilizan **BeginEdit** con **EndEdit** y **CancelEdit**. En el ejemplo también se comprueba el **ProposedValue** del evento **ColumnChanged** y se decide si cancelar la edición.

```
Dim workTable As DataTable = New DataTable
workTable.Columns.Add("LastName", Type.GetType("System.String"))
AddHandler workTable.ColumnChanged, _
  New DataColumnChangeEventHandler(AddressOf OnColumnChanged)
Dim workRow As DataRow = workTable.NewRow()
workRow(0) = "Smith"
workTable.Rows.Add(workRow)
workRow.BeginEdit()
' Causes the ColumnChanged event to write a message and cancel the edit.
workRow(0) = ""
workRow.EndEdit()
' Displays "Smith, New".
Console.WriteLine("{0}, {1}", workRow(0), workRow.RowState)
Private Shared Sub OnColumnChanged( _
```

```
  sender As Object, args As DataColumnChangeEventArgs)
  If args.Column.ColumnName = "LastName" Then
    If args.ProposedValue.ToString() = "" Then
      Console.WriteLine("Last Name cannot be blank.  Edit canceled.")
      args.Row.CancelEdit()
    End If
  End If
End Sub
```

5.9.3.5. Estados de Fila y Versiones de Fila

ADO .NET administra las filas de las tablas mediante estados de fila y versiones de fila. Un estado de fila indica el estado de una fila; las versiones de fila mantienen los valores almacenados en una fila en cuanto se modifica, incluyendo los valores actuales, originales y predeterminados. Por ejemplo, después de realizar una modificación en una columna de una fila, ésta adquiere el estado de fila **Modified** y dos versiones de fila:**Current** , que contiene los valores actuales de fila, y **Original**, que contiene los valores de fila antes de la modificación de la columna.

Cada objeto DataRow cuenta con la propiedad RowState que puede examinar para determinar el estado actual de la fila. En la siguiente tabla se ofrece una breve descripción de los valores de enumeración de **RowState**.

Valor de RowState	Descripción
Unchanged	No se han hecho cambios desde la última llamada a **AcceptChanges** o desde que **DataAdapter.Fill** creó la fila.
Added	Se ha agregado la fila a la tabla, pero no se ha llamado a **AcceptChanges**.
Modified	Se ha cambiado algún elemento de la fila.
Deleted	Se ha eliminado la fila de una tabla y no se ha llamado a **AcceptChanges**.
Detached	La fila no forma parte de ninguna **DataRowCollection**. El valor de **RowState** de una fila recién creada se establece en **Detached**. Una vez que se ha agregado la nueva **DataRow** a **DataRowCollection** llamando al método **Add**, el valor de la propiedad **RowState** se establece en **Added**. También se establece **Detached** en una fila que se ha quitado de una **DataRowCollection** con el método **Remove**, o mediante el método **Delete** seguido del método **AcceptChanges**.

Si se llama a **AcceptChanges** en un objeto DataSet, DataTable o DataRow, se quitan todas las filas con el estado de fila **Deleted**. Las filas que quedan reciben el estado de fila **Unchanged** y los valores de la versión de fila **Original** se sobrescriben con los valores de la versión de fila **Current**. Si se llama a **RejectChanges**, se quitan todas las filas con el estado de fila **Added**. Las filas que quedan reciben el estado de fila **Unchanged** y los valores de la versión de fila **Current** se sobrescriben con los valores de la versión de fila **Original**.

Las distintas versiones de una fila se pueden ver pasando un parámetro DataRowVersion con la referencia de la columna, como se muestra en el ejemplo siguiente.

```
Dim custRow As DataRow = custTable.Rows(0)
```

```
Dim custID As String = custRow("CustomerID",
DataRowVersion.Original).ToString()
```

En la siguiente tabla se ofrece una breve descripción de los valores de enumeración de **DataRowVersion**.

Valor de DataRowVersion	Descripción
Current	Valores actuales de la fila. Esta versión de fila no está disponible para filas con un valor **Deleted** en **RowState**.
Default	Ésta es la versión de fila predeterminada para una fila determinada. La versión de fila predeterminada para una fila **Added**, **Modified** o **Unchanged** es **Current**. La versión de fila predeterminada para una fila **Deleted** es **Original**. La versión de fila predeterminada para una fila **Detached** es **Proposed**.
Original	Valores originales de la fila. Esta versión de fila no está disponible para filas con un valor **Added** en **RowState**.
Proposed	Valores propuestos para la fila. Esta versión de fila existe mientras dura una operación de edición en una fila, o para una fila que no forma parte de una **DataRowCollection**.

Se puede comprobar si una **DataRow** tiene una versión de fila concreta llamando al método HasVersion y pasando **DataRowVersion** como argumento. Por ejemplo, DataRow.HasVersion(DataRowVersion.Original) devolverá **false** para las filas recién agregadas antes de llamar a **AcceptChanges**.

En el siguiente ejemplo de código, se muestran los valores en todas las filas eliminadas de una tabla. Las filas **Deleted** no tienen una versión de fila **Current** y, por lo tanto, debe pasar **DataRowVersion.Original** cuando obtenga acceso a los valores de columna.

```
Dim catTable As DataTable = catDS.Tables("Categories")
Dim delRows() As DataRow = catTable.Select(Nothing, Nothing,
DataViewRowState.Deleted)
Console.WriteLine("Deleted rows:" & vbCrLf)
Dim catCol As DataColumn
Dim delRow As DataRow
For Each catCol In catTable.Columns
  Console.Write(catCol.ColumnName & vbTab)
Next
Console.WriteLine()
For Each delRow In delRows
  For Each catCol In catTable.Columns
    Console.Write(delRow(catCol, DataRowVersion.Original) & vbTab)
  Next
  Console.WriteLine()
Next
```

5.9.3.6. Eliminar un DataRow

Existen dos métodos para eliminar un objeto DataRow de un objeto DataTable: el método **Remove** del objeto DataRowCollection y el método Delete del objeto **DataRow**. Mientras que el método **Remove**

elimina un objeto **DataRow** de la **DataRowCollection**, el método **Delete** únicamente lo marca para su eliminación. La eliminación propiamente dicha se produce cuando la aplicación llama al método **AcceptChanges**. Si se usa **Delete**, se puede comprobar mediante programación qué filas están marcadas para eliminación antes de quitarlas. Cuando una fila está marcada para eliminación, su propiedad RowState está establecida en **Deleted**.

Si utiliza un DataSet o una **DataTable** en combinación con un **DataAdapter** y un origen de datos relacional, utilice el método **Delete** de la **DataRow** para quitar la fila. El método **Delete** marca la fila como **Deleted** en **DataSet** o **DataTable** pero no la quita. En su lugar, cuando el **DataAdapter** encuentra una fila marcada como **Deleted**, ejecuta el método **DeleteCommand** para eliminar la fila en el origen de datos. A continuación se puede quitar la fila permanentemente utilizando el método **AcceptChanges**. Si utiliza **Remove** para quitar la fila, ésta desaparecerá por completo de la tabla, pero el **DataAdapter** no eliminará la fila del origen de datos.

El método **Remove** de la **DataRowCollection** toma una **DataRow** como argumento y la quita de la colección, como se muestra en el ejemplo siguiente.

```
workTable.Rows.Remove(workRow)
```

Por el contrario, en el siguiente ejemplo se muestra cómo se llama al método **Delete** en una **DataRow** para cambiar el **RowState** a **Deleted**.

```
workRow.Delete
```

Si una fila está marcada para eliminación y se llama al método **AcceptChanges** del objeto **DataTable**, la fila se quita de la **DataTable**. Por el contrario, si se llama a **RejectChanges**, el **RowState** de la fila vuelve a ser el que era antes de que se marcara como **Deleted**.

Nota:

Si el **RowState** de una **DataRow** se **Added**, lo que quiere decir que se acaba de agregar a la tabla y, a continuación, se marca como **Deleted**, se quita de la tabla.

5.9.3.7. Información de Errores de Fila

Para evitar responder a errores de fila durante la edición de valores en una DataTable, puede agregar la información de error a la fila para utilizarla más adelante. Para ello, el objeto DataRow proporciona una propiedad RowError en cada fila. Si se agregan datos a la propiedad **RowError** de una **DataRow**, la propiedad HasErrors de la **DataRow** se establece en **true**. Si la **DataRow** forma parte de una **DataTable** y **DataRow.HasErrors** es **true**, la propiedad **DataTable.HasErrors** es también **true**. Esto afecta también al **DataSet** al que pertenece la **DataTable**. Cuando se hagan pruebas para detectar errores, se puede comprobar la propiedad **HasErrors** para determinar si se ha agregado información de error a alguna fila. Si **HasErrors** es **true**, se puede utilizar el método GetErrors de la **DataTable** para obtener y examinar sólo las filas con errores, como se muestra en el ejemplo siguiente.

```
Dim workTable As DataTable = New DataTable("Customers")
workTable.Columns.Add("CustID", Type.GetType("System.Int32"))
workTable.Columns.Add("Total", Type.GetType("System.Double"))
AddHandler workTable.RowChanged, New DataRowChangeEventHandler(AddressOf
OnRowChanged)
Dim i  As Int32
For i  = 0 To 10
```

```
  workTable.Rows.Add(New Object() {i , i *100})
Next
If workTable.HasErrors Then
  Console.WriteLine("Errors in Table " & workTable.TableName)
  Dim myRow As DataRow
  For Each myRow In workTable.GetErrors()
    Console.WriteLine("CustID = " & myRow("CustID").ToString())
    Console.WriteLine(" Error = " & myRow.RowError & vbCrLf)
  Next
End If
Private Shared Sub OnRowChanged( _
    sender As Object, args As DataRowChangeEventArgs)
  ' Check for zero values.
  If CDbl(args.Row("Total")) = 0 Then args.Row.RowError = _
      "Total cannot be 0."
End Sub
```

5.9.3.8. AcceptChanges y RejectChanges

Después de comprobar la exactitud de los cambios realizados en una DataTable, se pueden aceptar con el método AcceptChanges de DataRow, DataTable o DataSet, que configurará los valores de fila **Current** de modo que sean los valores **Original** y establecerá la propiedad **RowState** en **Unchanged**. Si se aceptan o se rechazan los cambios, se elimina la información de **RowError** y se establece la propiedad **HasErrors** en **false**. Aceptar o rechazar cambios también puede afectar a la actualización de datos en el origen de datos.

Si hay restricciones de clave externa en la **DataTable**, los cambios que se acepten o se rechacen con **AcceptChanges** y **RejectChanges** se propagan a las filas secundarias de la **DataRow** de acuerdo con la **ForeignKeyConstraint.AcceptRejectRule**.

En el ejemplo siguiente se comprueba si hay filas con errores, se resuelven los errores que haya y se rechazan las filas en las que no se puede resolver el error. Tenga en cuenta que, en los errores que se resuelven, el valor **RowError** se restablece en una cadena vacía, con lo que la propiedad **HasErrors** se establece en **false**. Una vez que se han resuelto o rechazado todas las filas con errores, se llama a **AcceptChanges** para aceptar todos los cambios de toda la **DataTable**.

```
If workTable.HasErrors Then
  Dim errRow As DataRow
  For Each errRow in workTable.GetErrors()
    If errRow.RowError = "Total cannot exceed 1000." Then
      errRow("Total") = 1000
      errRow.RowError = ""    ' Clear the error.
    Else
      errRow.RejectChanges()
    End If
  Next
End If
workTable.AcceptChanges()
```

5.9.3.9. Control de Eventos del DataTable

El objeto DataTable proporciona una serie de eventos que una aplicación puede procesar. En la siguiente tabla se describen los eventos de **DataTable**.

Evento	Descripción
Initialized	Se produce después de haber llamado al método EndInit de un objeto **DataTable**. Este evento está concebido principalmente para admitir escenarios en tiempo de diseño.
ColumnChanged	Se produce después de cambiar correctamente un valor en un objeto DataColumn.
ColumnChanging	Se produce cuando se ha enviado un valor para un objeto **DataColumn**.
RowChanged	Se produce cuando se ha cambiado correctamente un valor de **DataColumn** o la propiedad RowState de un objeto DataRow en el objeto **DataTable**.
RowChanging	Se produce cuando se ha enviado un cambio para un valor **DataColumn** o la propiedad **RowState** de un objeto **DataRow** en el objeto **DataTable**.
RowDeleted	Se produce después de marcar un objeto **DataRow** de un objeto **DataTable** como **Deleted**.
RowDeleting	Se produce antes de marcar un objeto **DataRow** de un objeto **DataTable** como **Deleted**.
TableCleared	Se produce después de que una llamada al método Clear del objeto **DataTable** haya borrado correctamente todos los objetos **DataRow**.
TableClearing	Se produce después de haber llamado al método **Clear** pero antes de que se inicie la operación **Clear**.
TableNewRow	Se produce después de crear un nuevo objeto **DataRow** mediante una llamada al método **NewRow** del objeto **DataTable**.
Disposed	Se produce cuando el objeto **DataTable** se establece en **Disposed**. Se hereda de MarshalByValueComponent.

Nota:

La mayoría de las operaciones que agregan o eliminan filas no provocan eventos **ColumnChanged** ni **ColumnChanging**. Sin embargo, el método **ReadXml** sí provoca eventos **ColumnChanged** y **ColumnChanging**, a menos que **XmlReadMode** se establezca en **DiffGram** o se establezca en **Auto** cuando el documento XML que se lee es **DiffGram**.

Eventos relacionados adicionales

La propiedad Constraints contiene una instancia de ConstraintCollection. La clase ConstraintCollection expone un evento CollectionChanged. Este evento se activa cuando se agrega, modifica o quita una restricción de **ConstraintCollection**.

La propiedad Columns contiene una instancia de DataColumnCollection. La clase **DataColumnCollection** expone un evento CollectionChanged. Este evento se activa cuando se agrega, modifica o quita un objeto **DataColumn** de **DataColumnCollection**. Entre las modificaciones que provocan el inicio del evento se encuentran los cambios en el nombre, el tipo, la expresión o la posición ordinal de una columna.

La propiedad Tables de un objeto DataSet incluye una instancia de DataTableCollection. La clase **DataTableCollection** expone los eventos **CollectionChanged** y **CollectionChanging**. Estos eventos se activan cuando se agrega o se quita un objeto **DataTable** del **DataSet**.

Los cambios en **DataRows** también pueden activar eventos de un objeto DataView asociado. La clase **DataView** expone un evento ListChanged que se activa cuando cambia un valor de **DataColumn** o cuando cambian la composición o el orden de la vista. La clase DataRowView expone un evento PropertyChanged que se activa cuando cambia un valor de **DataColumn** asociado.

Secuencia de operaciones

A continuación se indica la secuencia de las operaciones que tienen lugar cuando se agrega, modifica o elimina un objeto **DataRow**.

1. Se crea el registro propuesto y se aplican los cambios.
2. Se comprueban las restricciones en columnas que no son de expresión.
3. Se provocan los eventos **RowChanging** o **RowDeleting**, según corresponda.
4. El registro propuesto se establece en el registro actual.
5. Se actualizan los índices asociados.
6. Se provocan los eventos **ListChanged** de los objetos **DataView** asociados y los eventos **PropertyChanged** de los objetos **DataRowView** asociados.
7. Se evalúan todas las columnas de expresión, pero se retrasa la comprobación de las restricciones de estas columnas.
8. Se provocan los eventos **ListChanged** de los objetos **DataView** asociados y los eventos **PropertyChanged** de los objetos **DataRowView** asociados a los que afecten las evaluaciones de la columna de expresión.
9. Se provocan los eventos **RowChanged** o **RowDeleted**, según corresponda.
10. Se comprueban las restricciones en las columnas de expresión.

Nota:

Los cambios en las columnas de expresión nunca provocan eventos **DataTable**. Los cambios en las columnas de expresión sólo provocan eventos **DataView** y **DataRowView**. Las columnas de expresión pueden tener dependencias en otras columnas y se pueden evaluar varias veces durante una única operación de **DataRow**. Cada evaluación de expresión provoca eventos, y una sola operación de **ListChanged** puede provocar varios eventos **PropertyChanged** y **DataRow** cuando se ven afectadas columnas de expresión, que posiblemente incluyen varios eventos para la misma columna de expresión.

Ejemplo

En el ejemplo siguiente se muestra cómo crear controladores de eventos para los eventos **RowChanged**, **RowChanging**, **RowDeleted**, **RowDeleting**, **ColumnChanged**, **ColumnChanging**, **TableNewRow**, **TableCleared** y **TableClearing**. Cada controlador de eventos muestra resultado en la ventana de la consola cuando se activa.

```
Private Sub DataTableEvents()
    Dim table As DataTable = New DataTable("Customers")
    ' Add two columns, id and name.
    table.Columns.Add("id", Type.GetType("System.Int32"))
    table.Columns.Add("name", Type.GetType("System.String"))
```

```
    ' Set the primary key.
    table.Columns("id").Unique = True
    table.PrimaryKey = New DataColumn() {table.Columns("id")}
    ' Add a RowChanged event handler.
    AddHandler table.RowChanged, _
           New DataRowChangeEventHandler(AddressOf Row_Changed)
    ' Add a RowChanging event handler.
    AddHandler table.RowChanging, _
           New DataRowChangeEventHandler(AddressOf Row_Changing)
    ' Add a RowDeleted event handler.
    AddHandler table.RowDeleted, New _
           DataRowChangeEventHandler(AddressOf Row_Deleted)
    ' Add a RowDeleting event handler.
    AddHandler table.RowDeleting, New _
           DataRowChangeEventHandler(AddressOf Row_Deleting)
    ' Add a ColumnChanged event handler.
    AddHandler table.ColumnChanged, _
           New DataColumnChangeEventHandler(AddressOf Column_Changed)
    ' Add a ColumnChanging event handler for the table.
    AddHandler table.ColumnChanging, New _
           DataColumnChangeEventHandler(AddressOf Column_Changing)
    ' Add a TableNewRow event handler.
    AddHandler table.TableNewRow, New _
           DataTableNewRowEventHandler(AddressOf Table_NewRow)
    ' Add a TableCleared event handler.
    AddHandler table.TableCleared, New _
           DataTableClearEventHandler(AddressOf Table_Cleared)
    ' Add a TableClearing event handler.
    AddHandler table.TableClearing, New _
           DataTableClearEventHandler(AddressOf Table_Clearing)
    ' Add a customer.
    Dim row As DataRow = table.NewRow()
    row("id") = 1
    row("name") = "Customer1"
    table.Rows.Add(row)
    table.AcceptChanges()
    ' Change the customer name.
    table.Rows(0).Item("name") = "ChangedCustomer1"
    ' Delete the row.
    table.Rows(0).Delete()
    ' Clear the table.
    table.Clear()
End Sub
Private Sub Row_Changed(ByVal sender As Object, _
    ByVal e As DataRowChangeEventArgs)
    Console.WriteLine("Row_Changed Event: name={0}; action={1}", _
     e.Row("name"), e.Action)
End Sub
Private Sub Row_Changing(ByVal sender As Object, _
    ByVal e As DataRowChangeEventArgs)
```

```
    Console.WriteLine("Row_Changing Event: name={0}; action={1}", _
     e.Row("name"), e.Action)
End Sub
Private Sub Row_Deleted(ByVal sender As Object, _
    ByVal e As DataRowChangeEventArgs)
    Console.WriteLine("Row_Deleted Event: name={0}; action={1}", _
     e.Row("name", DataRowVersion.Original), e.Action)
End Sub
Private Sub Row_Deleting(ByVal sender As Object, _
    ByVal e As DataRowChangeEventArgs)
    Console.WriteLine("Row_Deleting Event: name={0}; action={1}", _
       e.Row("name"), e.Action)
End Sub
Private Sub Column_Changed(ByVal sender As Object, _
    ByVal e As DataColumnChangeEventArgs)
Console.WriteLine("Column_Changed Event: ColumnName={0}; RowState={1}", _
       e.Column.ColumnName, e.Row.RowState)
End Sub
Private Sub Column_Changing(ByVal sender As Object, _
    ByVal e As DataColumnChangeEventArgs)
Console.WriteLine("Column_Changing Event: ColumnName={0}; RowState={1}",
       e.Column.ColumnName, e.Row.RowState)
End Sub
Private Sub Table_NewRow(ByVal sender As Object, _
ByVal e As DataTableNewRowEventArgs)
    Console.WriteLine("Table_NewRow Event: RowState={0}", _
       e.Row.RowState.ToString())
End Sub
Private Sub Table_Cleared(ByVal sender As Object, _
    ByVal e As DataTableClearEventArgs)
    Console.WriteLine("Table_Cleared Event: TableName={0}; Rows={1}", _
       e.TableName, e.Table.Rows.Count.ToString())
End Sub
Private Sub Table_Clearing(ByVal sender As Object, _
    ByVal e As DataTableClearEventArgs)
    Console.WriteLine("Table_Clearing Event: TableName={0}; Rows={1}", _
       e.TableName, e.Table.Rows.Count.ToString())
End Sub
```

5.10. DataTableReaders

El DataTableReader presenta el contenido de una DataTable o un DataSet con formato de uno o más conjuntos de resultados de sólo lectura y de sólo avance.

Al crear un **DataTableReader** desde una **DataTable**, el objeto **DataTableReader** resultante contiene un conjunto de resultados con los mismos datos que la **DataTable** desde la que se creó, con excepción de las filas que se hayan marcado como eliminadas. Las columnas aparecen en el mismo orden que en la **DataTable** original.

Un **DataTableReader** puede contener varios conjuntos de resultados si se ha creado llamando a CreateDataReader. Los resultados se encuentran en el mismo orden que las **DataTables** de la colección Tables del objeto **DataSet**.

5.10.1. Crear un DataReader

Las clases DataTable y DataSet tienen un método CreateDataReader que devuelve el contenido de la DataTable o el contenido de la colección Tables del objeto DataSet como uno o más conjuntos de resultados de sólo lectura y sólo avance.

⊟ Ejemplo

La siguiente aplicación de consola crea una instancia de DataTable. A continuación, en el ejemplo se pasa la DataTable rellenada a un procedimiento que llama al método CreateDataReader, que repite los resultados que se encuentran en el DataTableReader.

```
Private Sub TestCreateDataReader(ByVal dt As DataTable)
  ' Given a DataTable, retrieve a DataTableReader
  ' allowing access to all the tables's data:
  Using reader As DataTableReader = dt.CreateDataReader()
    Do
      If Not reader.HasRows Then
        Console.WriteLine("Empty DataTableReader")
      Else
        PrintColumns(reader)
      End If
      Console.WriteLine("========================")
    Loop While reader.NextResult()
  End Using
End Sub
Private Function GetCustomers() As DataTable
  ' Create sample Customers table, in order
  ' to demonstrate the behavior of the DataTableReader.
  Dim table As New DataTable
   ' Create two columns, ID and Name.
  Dim idColumn As DataColumn = table.Columns.Add("ID", GetType(Integer))
  table.Columns.Add("Name", GetType(String))
  ' Set the ID column as the primary key column.
  table.PrimaryKey = New DataColumn() {idColumn}
  table.Rows.Add(New Object() {1, "Mary"})
  table.Rows.Add(New Object() {2, "Andy"})
  table.Rows.Add(New Object() {3, "Peter"})
  table.Rows.Add(New Object() {4, "Russ"})
  Return table
End Function
Private Sub PrintColumns( _
   ByVal reader As DataTableReader)
  ' Loop through all the rows in the DataTableReader.
  Do While reader.Read()
    For i As Integer = 0 To reader.FieldCount - 1
```

```
      Console.Write(reader(i).ToString() & " ")
    Next
    Console.WriteLine()
  Loop
End Sub
```

En el ejemplo se muestra el siguiente resultado en la ventana de consola:

```
1 Mary
2 Andy
3 Peter
4 Russ
```

5.10.2. Navegar por DataTables

El DataTableReader obtiene el contenido de uno o más objetos DataTable con formato de uno o más conjuntos de sólo lectura y de sólo avance.

Un DataTableReader puede contener varios conjuntos de resultados si se crea mediante el método CreateDataReader. Si hay más de un conjunto de resultados, el método NextResult desplaza el cursor hasta el siguiente conjunto de resultados. Este proceso es de sólo avance. No es posible volver a un conjunto de resultados anterior.

⊟ Ejemplo

En el siguiente ejemplo, el método TestConstructor crea dos instancias de DataTable . Para mostrar este constructor para la clase DataTableReader, se crea en el ejemplo un nuevo **DataTableReader** basado en una matriz que contiene las dos **DataTables** y realiza una operación sencilla, que imprime el contenido de las primeras columnas en la ventana de la consola.

```
Private Sub TestConstructor()
   ' Create two data adapters, one for each of the two
   ' DataTables to be filled.
   Dim customerDataTable As DataTable = GetCustomers()
   Dim productDataTable As DataTable = GetProducts()
   ' Create the new DataTableReader.
   Using reader As New DataTableReader( _
      New DataTable() {customerDataTable, productDataTable})
      ' Print the contents of each of the result sets.
      Do
         PrintColumns(reader)
      Loop While reader.NextResult()
   End Using
   Console.WriteLine("Press Enter to finish.")
   Console.ReadLine()
End Sub
Private Function GetCustomers() As DataTable
   ' Create sample Customers table, in order
   ' to demonstrate the behavior of the DataTableReader.
   Dim table As New DataTable
   ' Create two columns, ID and Name.
   Dim idColumn As DataColumn = table.Columns.Add("ID", GetType(Integer))
   table.Columns.Add("Name", GetType(String))
```

```
    ' Set the ID column as the primary key column.
    table.PrimaryKey = New DataColumn() {idColumn}
    table.Rows.Add(New Object() {1, "Mary"})
    table.Rows.Add(New Object() {2, "Andy"})
    table.Rows.Add(New Object() {3, "Peter"})
    table.Rows.Add(New Object() {4, "Russ"})
    Return table
End Function
Private Function GetProducts() As DataTable
    ' Create sample Products table, in order
    ' to demonstrate the behavior of the DataTableReader.
    Dim table As New DataTable
    ' Create two columns, ID and Name.
    Dim idColumn As DataColumn = table.Columns.Add("ID", GetType(Integer))
    table.Columns.Add("Name", GetType(String))
    ' Set the ID column as the primary key column.
    table.PrimaryKey = New DataColumn() {idColumn}
    table.Rows.Add(New Object() {1, "Wireless Network Card"})
    table.Rows.Add(New Object() {2, "Hard Drive"})
    table.Rows.Add(New Object() {3, "Monitor"})
    table.Rows.Add(New Object() {4, "CPU"})
    Return table
End Function
Private Sub PrintColumns(ByVal reader As DataTableReader)
    ' Loop through all the rows in the DataTableReader.
    Do While reader.Read()
        For i As Integer = 0 To reader.FieldCount - 1
            Console.Write(reader(i).ToString() & " ")
        Next
        Console.WriteLine()
    Loop
End Sub
```

5.11. DataViews

Una DataView le permite crear diferentes vistas de los datos almacenados en una DataTable, una capacidad que suele utilizarse en aplicaciones de enlace a datos. Mediante una **DataView** puede exponer los datos de una tabla con distintos criterios de ordenación y filtrar los datos por el estado de fila o basándose en una expresión de filtro.

Una **DataView** proporciona una vista de datos dinámica en la **DataTable** subyacente: el contenido, el orden y la pertenencia reflejan los cambios en cuanto se producen. Este comportamiento difiere del método **Select** de la **DataTable**, que devuelve una matriz de DataRow de una tabla basada en un filtro o un orden determinados: estecontenido refleja cambios en la tabla subyacente, pero la pertenencia y la ordenación siguen siendo estáticas. Las capacidades dinámicas de la **DataView** hacen que resulte ideal para las aplicaciones de enlace a datos.

Una **DataView** proporciona una vista dinámica de un único conjunto de datos, similar a la vista de una base de datos, a la que puede aplicar distintos criterios de ordenación y filtrado. Sin embargo, al contrario que una vista de base de datos, una **DataView** no puede tratarse como una tabla y no puede

proporcionar una vista de tablas combinadas. Tampoco puede excluir columnas que existen en la tabla de origen ni puede anexar columnas, como columnas de cálculo, que no existen en la tabla de origen.

Puede utilizar un DataViewManager para administrar la configuración de vista para todas las tablas de un **DataSet**. El **DataViewManager** proporciona una forma cómoda de administrar la configuración de vista predeterminada para cada tabla. Al enlazar un control a más de una tabla de un **DataSet**, el enlace a un **DataViewManager** es la elección ideal.

5.11.1. Crear DataView

Hay dos formas de crear una DataView. Puede utilizar el constructor **DataView** o puede crear una referencia a la propiedad DefaultView de la DataTable. El constructor **DataView** puede estar vacío o puede aceptar también **DataTable** como único argumento o **DataTable** junto con el criterio de filtro o de ordenación, y un filtro de estado de fila.

Como el índice de una **DataView** se crea al mismo tiempo que **DataView** y cuando se modifica alguna de las propiedades **Sort**, **RowFilter** o **RowStateFilter**, conseguirá un rendimiento óptimo si suministra cualquier criterio inicial de ordenación o filtro como argumentos del constructor al crear la **DataView**. Al crear una **DataView** sin especificar criterios de ordenación o de filtro y establecer posteriormente las propiedades **Sort**, **RowFilter** o **RowStateFilter** hace que el índice se construya dos veces como mínimo: una vez al crear la **DataView** y otra vez más al modificar cualquiera de las propiedades de ordenación o filtro.

Tenga en cuenta que si crea la **DataView** con el constructor que no toma ningún argumento, no podrá utilizar la **DataView** hasta que no establezca la propiedad **Table**.

En el ejemplo de código siguiente se muestra cómo crear una **DataView** con el constructor **DataView**. Con la **DataTable** se suministran un **RowFilter**, una columna **Sort** y un **DataViewRowState**.

```
Dim custDV As DataView = New DataView(custDS.Tables("Customers"), _
    "Country = 'USA'", "ContactName", DataViewRowState.CurrentRows)
```

En el siguiente ejemplo de código se muestra cómo obtener una referencia a la **DataView** predeterminada de una **DataTable** mediante la propiedad **DefaultView** de la tabla.

```
Dim custDV As DataView = custDS.Tables("Customers").DefaultView
```

5.11.2. Ordenar y Filtrar Datos

El DataView proporciona varias formas de ordenación y filtrado de datos en un DataTable:

- Mediante la propiedad Sort puede especificar criterios simples o múltiples de ordenación de columnas e incluir parámetros ASC (ascendente) y DESC (descendente).
- Mediante la propiedad ApplyDefaultSort puede crear automáticamente un criterio de ordenación, en orden ascendente, basado en la columna o columnas de clave principal de la tabla. ApplyDefaultSort sólo se aplica cuando la propiedad **Sort** es una referencia nula o una cadena vacía y cuando la tabla tiene definida una clave principal.
- Mediante la propiedad RowFilter puede especificar subconjuntos de filas basándose en sus valores de columna.

Si desea devolver los resultados de una consulta determinada en los datos, en lugar de proporcionar una vista dinámica de un subconjunto de los datos, para conseguir el máximo rendimiento utilice los métodos Find o FindRows de la **DataView** en lugar de establecer la propiedad **RowFilter**. El establecimiento de la propiedad **RowFilter** hace que se vuelva a generar el índice de los datos, lo que agrega sobrecarga a la aplicación y reduce el rendimiento. La propiedad **RowFilter** es más idónea en una aplicación enlazada a datos donde un control enlazado muestra resultados filtrados. Los métodos **Find** y **FindRows** aprovechan el índice actual, sin necesidad de volver a generarlo.

- Mediante la propiedad RowStateFilter puede especificar las versiones de fila que desea ver. La **DataView** administra implícitamente qué versión de fila exponer, dependiendo del **RowState** de la fila subyacente. Por ejemplo, si el **RowStateFilter** está establecido como **DataViewRowState.Deleted**, la **DataView** expone la versión de fila **Original** de todas las filas **Deleted** porque no hay ninguna versión de fila **Current**. Mediante la propiedad **RowVersion** de la **DataRowView** puede determinar qué versión de una fila se está exponiendo.

  En la siguiente tabla se muestran las opciones de **DataViewRowState**.

Opciones de DataViewRowState	Descripción
CurrentRows	La versión de fila **Current** de todas las filas **Unchanged**, **Added** y **Modified**. Éste es el valor predeterminado.
Added	La versión de fila **Current** de todas las filas **Added**.
Deleted	La versión de fila **Original** de todas las filas **Deleted**.
ModifiedCurrent	La versión de fila **Current** de todas las filas **Modified**.
ModifiedOriginal	La versión de fila **Original** de todas las filas **Modified**.
Ninguna	Ninguna fila.
OriginalRows	La versión de fila **Original** de todas las filas **Unchanged**, **Modified** y **Deleted**.
Unchanged	La versión de fila **Current** de todas las filas **Unchanged**.

En el siguiente ejemplo de código se crea una vista que muestra todos los productos cuyo número de unidades en existencia es menor o igual que el nivel de nuevo pedido, ordenados en primer lugar por id. de proveedor y después por nombre de producto.

```
Dim prodView As DataView = New DataView(prodDS.Tables("Products"), _
   "UnitsInStock <= ReorderLevel", "SupplierID, ProductName", _
   DataViewRowState.CurrentRows)
```

5.11.3. DataRows y DataRowViews

Un DataView expone una colección enumerable de objetos DataRowView. Los objetos **DataRowView** exponen valores como matrices de objetos indizadas por el nombre o la referencia ordinal de la columna de la tabla subyacente. Mediante la propiedad Row puede tener acceso a la DataRow expuesta por la propiedad **DataRowView** de la **DataRowView**.

Cuando se ven valores mediante **DataRowView**, la propiedad RowStateFilter de la **DataView** determina qué versión de fila de la **DataRow** subyacente se expone.

El siguiente ejemplo de código muestra todos los valores actuales y originales de una tabla.

```
Dim catView As DataView = New DataView(catDS.Tables("Categories"))
Console.WriteLine("Current Values:")
WriteView(catView)
Console.WriteLine("Original Values:")
catView.RowStateFilter = DataViewRowState.ModifiedOriginal
WriteView(catView)
Public Shared Sub WriteView(thisDataView As DataView)
  Dim rowView As DataRowView
  Dim i As Integer
  For Each rowView In thisDataView
    For i = 0 To thisDataView.Table.Columns.Count - 1
      Console.Write(rowView(i) & vbTab)
    Next
    Console.WriteLine()
  Next
End Sub
```

5.11.4. Buscar Filas

Es posible buscar filas en función de los valores clave de ordenación mediante los métodos Find y FindRows de la DataView. La diferenciación entre mayúsculas y minúsculas de los valores de búsqueda en los métodos **Find** y **FindRows** está determinada por la propiedad **CaseSensitive** de la DataTable subyacente. Los valores de búsqueda deben coincidir en su totalidad con los valores de clave de ordenación existentes para que se devuelva un resultado.

El método **Find** devuelve un entero con el índice de la DataRowView que coincide con los criterios de búsqueda. Si más de una fila coincide con los criterios de búsqueda, sólo se devolverá el índice de la primera **DataRowView** coincidente. Si no se encuentra ninguna coincidencia, **Find** devuelve -1.

Para devolver resultados de la búsqueda que coincidan con varias filas, utilice el método **FindRows**. **FindRows** funciona igual que el método **Find**, excepto en que devuelve una matriz de **DataRowView** que hace referencia a todas las filas coincidentes de la **DataView**. Si no se encuentra ninguna coincidencia, la matriz de **DataRowView** estará vacía.

Para utilizar los métodos **Find** o **FindRows** debe especificar un criterio de ordenación; para ello, establezca **ApplyDefaultSort** como **true** o utilice la propiedad **Sort**. Si no se especifica ningún criterio de ordenación, se inicia una excepción.

Los métodos **Find** y **FindRows** toman como entrada una matriz de valores cuya longitud coincide con el número de columnas del criterio de ordenación. En el caso de una ordenación por una única columna, puede pasar un único valor. Para los criterios de ordenación que contienen varias columnas, debe pasar una matriz de objetos. Tenga en cuenta que para una ordenación según varias columnas, los valores de la matriz de objetos deben coincidir con el orden de las columnas especificado en la propiedad **Sort** de **DataView**.

En el siguiente ejemplo de código se muestra una llamada al método **Find** en una **DataView** con un criterio de ordenación de una única columna.

```
Dim custView As DataView = _
  New DataView(custDS.Tables("Customers"), "", _
  "CompanyName", DataViewRowState.CurrentRows)
Dim rowIndex As Integer = custView.Find("The Cracker Box")
If rowIndex = -1 Then
  Console.WriteLine("No match found.")
Else
  Console.WriteLine("{0}, {1}", _
    custView(rowIndex)("CustomerID").ToString(), _
    custView(rowIndex)("CompanyName").ToString())
End If
```

Si la propiedad **Sort** especifica varias columnas, debe pasar una matriz de objetos con los valores de búsqueda de cada columna en el orden especificado por la propiedad **Sort**, como en el siguiente ejemplo de código.

```
Dim custView As DataView = _
  New DataView(custDS.Tables("Customers"), "", _
  "CompanyName, ContactName", DataViewRowState.CurrentRows)
Dim foundRows() As DataRowView = _
  custView.FindRows(New object() {"The Cracker Box", "Liu Wong"})
If foundRows.Length = 0 Then
  Console.WriteLine("No match found.")
Else
  Dim myDRV As DataRowView
  For Each myDRV In foundRows
    Console.WriteLine("{0}, {1}", _
      myDRV("CompanyName").ToString(), myDRV("ContactName").ToString())
  Next
End If
```

5.11.5. ChildViews y Relaciones

Si existe una relación entre tablas de un DataSet, puede crear una DataView que contenga filas de la tabla secundaria relacionada mediante el método CreateChildView del DataRowView para las filas de la tabla primaria. Por ejemplo, en el código siguiente se muestran **Categories** y los **Products** relacionados en orden alfabético por **CategoryName** y **ProductName**.

```
Dim catTable As DataTable = catDS.Tables("Categories")
Dim prodTable As DataTable = catDS.Tables("Products")
' Create a relation between the Categories and Products tables.
Dim relation As DataRelation = catDS.Relations.Add("CatProdRel", _
  catTable.Columns("CategoryID"), _
  prodTable.Columns("CategoryID"))
' Create DataViews for the Categories and Products tables.
Dim catView As DataView = New DataView(catTable, "", _
  "CategoryName", DataViewRowState.CurrentRows)
Dim prodView As DataView
' Iterate through the Categories table.
```

```
Dim catDRV, prodDRV As DataRowView
For Each catDRV In catView
  Console.WriteLine(catDRV("CategoryName"))
  ' Create a DataView of the child product records.
  prodView = catDRV.CreateChildView(relation)
  prodView.Sort = "ProductName"
  For Each prodDRV In prodView
    Console.WriteLine(vbTab & prodDRV("ProductName"))
  Next
Next
```

5.11.6. Modificar Objetos DataView

Puede utilizar DataView para agregar, eliminar o modificar filas de datos de la tabla subyacente. La posibilidad de utilizar **DataView** para modificar los datos de la tabla subyacente se controla estableciendo una de las tres propiedades booleanas de **DataView**. Dichas propiedades son AllowNew, AllowEdit y AllowDelete. Están establecidas como **true** de forma predeterminada.

Si **AllowNew** tiene el valor **true**, puede utilizar el método AddNew de la **DataView** para crear una nueva DataRowView. Tenga en cuenta que no se agrega realmente una nueva fila al DataTable subyacente hasta que se llama al método EndEdit de la **DataRowView**. Si se llama al método CancelEdit de la **DataRowView**, se descartará la nueva fila. Tenga en cuenta también que sólo puede modificar una **DataRowView** cada vez. Si se llama al método **AddNew** o **BeginEdit** de la **DataRowView** mientras hay una fila pendiente, se llamará implícitamente a **EndEdit** en la fila pendiente. Cuando se llama a **EndEdit** se aplican los cambios a la **DataTable** subyacente; más tarde se pueden confirmar o rechazar mediante los métodos **AcceptChanges** o **RejectChanges** del objeto **DataTable**, **DataSet** o **DataRow**. Cuando **AllowNew** tiene el valor **false**, se iniciará una excepción si llama al método **AddNew** de la **DataRowView**.

Si **AllowEdit** tiene el valor **true**, puede modificar el contenido de la **DataRow** mediante **DataRowView**. Puede confirmar los cambios realizados en la fila subyacente mediante **DataRowView.EndEdit** o rechazarlos con **DataRowView.CancelEdit**. Tenga en cuenta que sólo puede modificar una fila cada vez. Si se llama al método **AddNew** o **BeginEdit** de la **DataRowView** mientras hay una fila pendiente, se llamará implícitamente a **EndEdit** en la fila pendiente. Cuando se llama a **EndEdit**, los cambios propuestos se ponen en la versión de fila **Current** de la **DataRow** subyacente; más tarde se pueden confirmar o rechazar mediante los métodos **AcceptChanges** o **RejectChanges** del objeto **DataTable**, **DataSet** o **DataRow**. Cuando **AllowEdit** tiene el valor **false**, se iniciará una excepción si intenta modificar un valor de la **DataRowView**.

Cuando se está modificando una **DataRowView** existente, seguirán provocándose eventos con los cambios propuestos de la **DataTable** subyacente. Tenga en cuenta que si llama a **EndEdit** o a **CancelEdit** en la **DataRow** subyacente, los cambios pendientes se aplicarán o se cancelarán independientemente de que se llame o no a **EndEdit** o a **CancelEdit** en la **DataRowView**.

Si **AllowDelete** tiene el valor **true**, puede eliminar filas en la **DataView** mediante el método **Delete** del objeto **DataView** o **DataRowView** y las filas se eliminarán de la **DataTable** subyacente. Más tarde puede confirmar o rechazar las eliminaciones mediante **AcceptChanges** o **RejectChanges**, respectivamente. Cuando **AllowDelete** tiene el valor **false**, se iniciará una excepción si llama al método **Delete** del objeto **DataView** o **DataRowView**.

En el siguiente ejemplo de código se deshabilita el uso de **DataView** para eliminar filas y se agrega una nueva fila a la tabla subyacente mediante la **DataView**.

```
Dim custTable As DataTable = custDS.Tables("Customers")
Dim custView As DataView = custTable.DefaultView
custView.Sort = "CompanyName"
custView.AllowDelete = False
Dim newDRV As DataRowView = custView.AddNew()
newDRV("CustomerID") = "ABCDE"
newDRV("CompanyName") = "ABC Products"
newDRV.EndEdit()
```

5.11.7. Tratamiento de Eventos del DataView

Puede utilizar el evento ListChanged del DataView para determinar si se ha actualizado una vista. Entre las actualizaciones que provocan el evento se incluyen la agregación de una fila a la tabla subyacente, así como su eliminación o modificación; la agregación de una columna al esquema de la tabla subyacente o su eliminación y la modificación de una relación primaria o secundaria. El evento **ListChanged** también le notifica si la lista de filas que está viendo ha cambiado considerablemente por la aplicación de un nuevo criterio de ordenación o de un filtro.

El evento **ListChanged** implementa el delegado **ListChangedEventHandler** del espacio de nombres System.ComponentModel y toma como entrada un objeto ListChangedEventArgs. Puede determinar el tipo de cambio producido mediante el valor de enumeración ListChangedType en la propiedad **ListChangedType** del objeto **ListChangedEventArgs**.Para los cambios que implican agregar, eliminar o mover filas, se puede tener acceso al nuevo índice de la fila agregada o movida y al índice anterior de la fila eliminada mediante la propiedad **NewIndex** del objeto **ListChangedEventArgs**. En el caso de una fila movida, se puede tener acceso al índice anterior de la fila movida mediante la propiedad **OldIndex** del objeto **ListChangedEventArgs**.

DataViewManager también expone un evento **ListChanged** para notificarle si se ha agregado o quitado una tabla, o si se ha realizado un cambio en la colección **Relations** del **DataSet** subyacente.

En el siguiente ejemplo de código se muestra cómo agregar un controlador de eventos **ListChanged**.

```
AddHandler custView.ListChanged, _
  New System.ComponentModel.ListChangedEventHandler( _
  AddressOf OnListChanged)
Private Shared Sub OnListChanged( _
  sender As Object, args As System.ComponentModel.ListChangedEventArgs)
  Console.WriteLine("ListChanged:")
  Console.WriteLine(vbTab & "    Type = " & _
    System.Enum.GetName(args.ListChangedType.GetType(), _
    args.ListChangedType))
  Console.WriteLine(vbTab & "OldIndex = " & args.OldIndex)
  Console.WriteLine(vbTab & "NewIndex = " & args.NewIndex)
End Sub
```

5.11.8. Administrar DataViews

Puede utilizar un DataViewManager para administrar la configuración de vista para todas las tablas de un DataView. Si tiene un control que desea enlazar a varias tablas, como una cuadrícula que navega por relaciones, es ideal utilizar un **DataViewManager**.

El **DataViewManager** contiene una colección de objetos DataViewSetting que se utilizan para establecer la configuración de vista de las tablas del DataSet. La DataViewSettingCollection contiene un objeto DataViewSetting para cada tabla de un **DataSet**. Puede establecer las propiedades **ApplyDefaultSort**, **Sort**, **RowFilter** y **RowStateFilter** predeterminadas de la tabla a la que se hace referencia mediante su **DataViewSetting**. Puede hacer referencia al **DataViewSetting** de una tabla determinada por su nombre o por su referencia ordinal, o si pasa una referencia a dicho objeto de tabla. Puede tener acceso a la colección de objetos **DataViewSetting** de un **DataViewManager** mediante la propiedad **DataViewSettings**.

En el siguiente código de ejemplo se rellena un **DataSet** con las tablas **Customers**, **Orders** y **Order Details** de la base de datos **Northwind** de SQL Server, se crean las relaciones entre las tablas, se utiliza un **DataViewManager** para establecer la configuración predeterminada de **DataView** y se enlaza una **DataGrid** al **DataViewManager**. En el ejemplo se establece la configuración predeterminada de **DataView** para todas las tablas del **DataSet** de manera que se ordene según la clave principal de la tabla (**ApplyDefaultSort** = **true**) y se modifica el criterio de ordenación de la tabla **Customers** para que se ordene por **CompanyName**.

```
' Assumes connection is a valid SqlConnection to Northwind.
' Create a Connection, DataAdapters, and a DataSet.
Dim custDA As SqlDataAdapter = New SqlDataAdapter( _
  "SELECT CustomerID, CompanyName FROM Customers", connection)
Dim orderDA As SqlDataAdapter = New SqlDataAdapter( _
  "SELECT OrderID, CustomerID FROM Orders", connection)
Dim ordDetDA As SqlDataAdapter = New SqlDataAdapter( _
  "SELECT OrderID, ProductID, Quantity FROM [Order Details]", connection)
Dim custDS As DataSet = New DataSet()
' Open the Connection.
connection.Open()
    ' Fill the DataSet with schema information and data.
    custDA.MissingSchemaAction = MissingSchemaAction.AddWithKey
    orderDA.MissingSchemaAction = MissingSchemaAction.AddWithKey
    ordDetDA.MissingSchemaAction = MissingSchemaAction.AddWithKey
    custDA.Fill(custDS, "Customers")
    orderDA.Fill(custDS, "Orders")
    ordDetDA.Fill(custDS, "OrderDetails")
    ' Close the Connection.
    connection.Close()
    ' Create relationships.
    custDS.Relations.Add("CustomerOrders", _
          custDS.Tables("Customers").Columns("CustomerID"), _
          custDS.Tables("Orders").Columns("CustomerID"))
    custDS.Relations.Add("OrderDetails", _
          custDS.Tables("Orders").Columns("OrderID"), _
          custDS.Tables("OrderDetails").Columns("OrderID"))
' Create default DataView settings.
Dim viewManager As DataViewManager = New DataViewManager(custDS)
```

```
Dim viewSetting As DataViewSetting
For Each viewSetting In viewManager.DataViewSettings
  viewSetting.ApplyDefaultSort = True
Next
viewManager.DataViewSettings("Customers").Sort = "CompanyName"
' Bind to a DataGrid.
Dim grid As System.Windows.Forms.DataGrid = New
System.Windows.Forms.DataGrid()
grid.SetDataBinding(viewManager, "Customers")
```

5.11.9. Crear DataTable Desde un DataView

Después de recuperar datos de un origen de datos y rellenar una DataTable con los mismos, puede ordenarlos, filtrarlos o limitar los datos devueltos sin volver a recuperarlos. Esto es posible gracias a la clase DataView. Si además necesita crear una DataTable nueva desde la DataView, puede utilizar el método ToTable para copiar todas las filas y columnas, o bien un subconjunto de los datos en una DataTable nueva. El método ToTable proporciona sobrecargas para:

- Crear una DataTable que contenga columnas que son un subconjunto de las columnas de la DataView.
- Crear una DataTable que incluya solamente filas distintas de la DataView, de forma análoga a la palabra clave DISTINCT en Transact-SQL.

⊟ Ejemplo

En el siguiente ejemplo de aplicación de consola se crea una DataTable que contiene datos de la tabla **Person.Contact** en la base de datos de ejemplo **AdventureWorks**. A continuación, se crea una DataView ordenada y filtrada basada en la DataTable. Después de mostrar el contenido de la DataTable y la DataView, se crea una nueva DataTable desde el DataView mediante el método ToTable y se selecciona solamente un subconjunto de las columnas disponibles. Por último, se muestra el contenido del DataTable nuevo.

```
Private Sub DemonstrateDataView()
    ' Retrieve a DataTable from the AdventureWorks sample database.
    ' connectionString is assumed to be a valid connection string.
    Dim adapter As New SqlDataAdapter( _
       "SELECT FirstName, LastName, EmailAddress FROM Person.Contact
WHERE FirstName LIKE 'Mich%'", connectionString)
    Dim table As New DataTable
    adapter.Fill(table)
    Console.WriteLine("Original table name: " & table.TableName)
    ' Print current table values.
    PrintTableOrView(table, "Current Values in Table")
    ' Now create a DataView based on the DataTable.
    ' Sort and filter the data.
    Dim view As DataView = table.DefaultView
    view.Sort = "LastName, FirstName"
    view.RowFilter = "LastName > 'M'"
    PrintTableOrView(view, "Current Values in View")
    ' Create a new DataTable based on the DataView,
```

```
    ' requesting only two columns with distinct values
    ' in the columns.
    Dim newTable As DataTable = view.ToTable("UniqueLastNames", True,
"FirstName", "LastName")
    PrintTableOrView(newTable, "Table created from DataView")
    Console.WriteLine("New table name: " & newTable.TableName)
    Console.WriteLine("Press any key to continue.")
    Console.ReadKey()
    End Sub
Private Sub PrintTableOrView(ByVal dv As DataView, ByVal label As String)
    Dim sw As System.IO.StringWriter
    Dim output As String
    Dim table As DataTable = dv.Table
    Console.WriteLine(label)
    ' Loop through each row in the view.
    For Each rowView As DataRowView In dv
        sw = New System.IO.StringWriter
        ' Loop through each column.
        For Each col As DataColumn In table.Columns
            ' Output the value of each column's data.
            sw.Write(rowView(col.ColumnName).ToString() & ", ")
        Next
        output = sw.ToString
        ' Trim off the trailing ", ", so the output looks correct.
        If output.Length > 2 Then
            output = output.Substring(0, output.Length - 2)
        End If
        ' Display the row in the console window.
        Console.WriteLine(output)
    Next
    Console.WriteLine()
End Sub
Private Sub PrintTableOrView(ByVal table As DataTable, ByVal label As
String)
    Dim sw As System.IO.StringWriter
    Dim output As String
    Console.WriteLine(label)
    ' Loop through each row in the table.
    For Each row As DataRow In table.Rows
        sw = New System.IO.StringWriter
        ' Loop through each column.
        For Each col As DataColumn In table.Columns
            ' Output the value of each column's data.
            sw.Write(row(col).ToString() & ", ")
        Next
        output = sw.ToString
        ' Trim off the trailing ", ", so the output looks correct.
        If output.Length > 2 Then
            output = output.Substring(0, output.Length - 2)
        End If
```

```
        ' Display the row in the console window.
        Console.WriteLine(output)
    Next
    Console.WriteLine()
    End Sub
End Module
```

5.12. Utilizar XML en un DataSet

Con ADO.NET es posible llenar un DataSet a partir de una secuencia o un documento XML. Se puede utilizar la secuencia o el documento XML para suministrar datos al DataSet, información de esquema o ambas cosas. La información suministrada desde la secuencia o el documento XML puede combinarse con datos o información de esquema existente ya presente en el DataSet.

ADO.NET también permite crear una representación XML de un DataSet, con o sin su esquema, para transportar el DataSet a través de HTTP con el fin de que lo utilice otra aplicación u otra plataforma compatible con XML. En una representación XML de un DataSet, los datos se escriben en XML y el esquema, si está incluido en línea en la representación, se escribe utilizando el lenguaje de definición de esquemas XML (XSD). XML y el esquema XML proporcionan un formato cómodo para transferir el contenido de un DataSet a y desde clientes remotos.

5.12.1. DiffGrams

Un DiffGram es un formato XML que identifica las versiones actual y original de los elementos de datos. El DataSet utiliza el formato DiffGram para cargar y hacer persistente su contenido, así como para serializar su contenido con el fin de transportarlo a través de una conexión de red. Cuando un DataSet se escribe como un DiffGram, llena el DiffGram con toda la información necesaria para volver a crear de forma precisa el contenido, aunque no el esquema del DataSet, incluyendo los valores de columna de las versiones de fila **Original** y **Current**, la información de error de fila y el orden de las filas.

Cuando se envía y recupera un DataSet desde un servicio web XML, se utiliza implícitamente el formato DiffGram. Además, al cargar el contenido de un DataSet desde XML mediante el método **ReadXml**, o al escribir el contenido de un DataSet en XML mediante el método **WriteXml**, se puede especificar que el contenido se lea o se escriba como un DiffGram.

Si bien .NET Framework utiliza principalmente el formato DiffGram como formato de serialización para el contenido de un DataSet, también es posible usar DiffGrams para modificar datos en tablas de una base de datos de Microsoft SQL Server.

Los Diffgram se generan escribiendo el contenido de todas las tablas en un elemento **<diffgram>**.

Para generar un Diffgram

1. Genere una lista de tablas raíz (es decir, tablas sin elemento primario).

2. Para cada tabla y sus descendientes en la lista, escriba la versión actual de todas las filas en la primera sección del Diffram.

3. Para cada tabla de DataSet, escriba la versión original de todas las filas, si hay alguna, en la sección **<before>** del Diffgram.

4. Para todas las filas que tengan errores, escriba el contenido del error en la sección **<errors>** del Diffgram.

El Diffram se procesa en orden, del principio del archivo XML al final.

Para procesar un Diffgram

1. Procese la primera sección del Diffgram, que contiene la versión actual de las filas.

2. Procese la segunda sección o la sección **<before>** que contiene la versión original de las filas modificadas y eliminadas.

> **Nota:**
>
> Si una fila está marcada como eliminada, la operación de eliminación puede suprimir también sus descendientes, dependiendo de la propiedad **Cascade** del actual DataSet.

3. Procese la sección **<errors>**. Establezca la información de error de la fila y la columna especificadas para cada elemento de esta sección.

> **Nota:**
>
> Si establece XmlWriteMode en Diffgram, el contenido del DataSet de destino y del DataSet original podría ser distinto.

⊟ Formato DiffGram

El formato DiffGram se divide en tres secciones: los datos actuales, los datos originales o "before" y una sección de errores, tal y como se muestra en el siguiente ejemplo.

```
<?xml version="1.0"?>
<diffgr:diffgram
        xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
        xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <DataInstance>
   </DataInstance>
  <diffgr:before>
  </diffgr:before>
  <diffgr:errors>
  </diffgr:errors>
</diffgr:diffgram>
```

El formato DiffGram consta de los bloques de datos siguientes:

< DataInstance >

El nombre de este elemento, ***DataInstance***, se utiliza con fines explicativos en esta documentación. Un elemento ***DataInstance*** representa un DataSet o una fila de una DataTable. En lugar de *DataInstance*, el elemento contendría el nombre del DataSet o del DataTable. Este bloque del formato DiffGram contiene los datos actuales, se hayan modificado o no. Un elemento, o fila, que se haya modificado se identifica con la anotación **diffgr:hasChanges**.

<diffgr:before>

Este bloque del formato DiffGram contiene la versión original de una fila. Los elementos de este bloque se hacen coincidir con los del bloque ***DataInstance*** mediante la anotación **diffgr:id**.

<diffgr:errors>

Este bloque del formato DiffGram contiene información de error para una fila determinada del bloque ***DataInstance***. Los elementos de este bloque se hacen coincidir con los del bloque ***DataInstance*** mediante la anotación **diffgr:id**.

⊟ Anotaciones de DiffGram

Los DiffGrams utilizan varias anotaciones para relacionar elementos de los distintos bloques de DiffGram que representan versiones de fila o información de error diferentes en el DataSet.

En la siguiente tabla se describen las anotaciones de DiffGram definidas en el espacio de nombres **urn:schemas-microsoft-com:xml-diffgram-v1** de DiffGram.

Anotación	Descripción
id	Se utiliza para emparejar los elementos de los bloques **<diffgr:before>** y **<diffgr:errors>** con elementos del bloque ***<DataInstance>***. Los valores que tienen la anotación **diffgr:id** tienen el formato [NombreTabla][IdentificadorFila]. Por ejemplo: <Customers diffgr:id="Customers1">.
parentId	Identifica qué elemento del bloque ***<DataInstance>*** es el principal del elemento actual. Los valores que tienen la anotación **diffgr:parentId** tienen el formato [NombreTabla][IdentificadorFila]. Por ejemplo: <Orders diffgr:parentId="Customers1">.
hasChanges	Identifica una fila del bloque ***<DataInstance>*** como modificada. La anotación **hasChanges** puede tener uno de los tres valores siguientes: **inserted** Identifica una fila agregada (**Added**). **modified** Identifica una fila modificada **Modified** que contiene una versión de fila **Original** en el bloque **<diffgr:before>**. Hay que tener en cuenta que las filas **Deleted** tendrán una versión de fila **Original** en el bloque **<diffgr:before>**, pero no habrá ningún elemento anotado en el bloque ***<DataInstance>***. **descent** Identifica un elemento en el que se han modificado uno o más elementos secundarios de una relación primaria-secundaria.
hasErrors	Identifica una fila del bloque ***<DataInstance>*** como que tiene un error **RowError**. El elemento erróneo se sitúa en el bloque **<diffgr:errors>**.
Error	Contiene el texto de **RowError** para un elemento determinado en el bloque **<diffgr:errors>**.

El DataSet incluye otras anotaciones al leer o escribir su contenido como un DiffGram. En la siguiente tabla se describen estas anotaciones adicionales, que se definen en el espacio de nombres **urn:schemas-microsoft-com:xml-msdata**.

Anotación	Descripción
RowOrder	Conserva el orden de fila de los datos originales e identifica el índice de una fila de una DataTable determinada.
Hidden	Identifica una columna que tiene el valor **MappingType.Hidden** en la propiedad

	ColumnMapping. El atributo se escribe con el formato **msdata:hidden**[NombreColumna]="valor". Por ejemplo: <Customers diffgr:id="Customers1" msdata:hiddenContactTitle="Owner">. Hay que tener en cuenta que las columnas ocultas sólo se escriben como un atributo de DiffGram si contienen datos. De lo contrario, se pasan por alto.

DiffGram de ejemplo

A continuación se muestra un ejemplo del formato DiffGram. En este ejemplo se muestra el resultado de una actualización de una fila de una tabla antes de que se hayan confirmado los cambios. La fila cuyo CustomerID es "ALFKI" se ha modificado, pero no se ha actualizado. Como resultado, hay una fila **Current** con un valor **diffgr:id** de "Customers1" en el bloque ***<DataInstance>*** y una fila **Original** con un valor **diffgr:id** de "Customers1" en el bloque **<diffgr:before>**. La fila cuyo valor de CustomerID es "ANATR" incluye un **RowError**, por lo que se anota con diffgr:hasErrors="true" y hay un elemento relacionado en el bloque **<diffgr:errors>**.

```
<diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
  <CustomerDataSet>
    <Customers diffgr:id="Customers1" msdata:rowOrder="0"
diffgr:hasChanges="modified">
      <CustomerID>ALFKI</CustomerID>
      <CompanyName>New Company</CompanyName>
    </Customers>
    <Customers diffgr:id="Customers2" msdata:rowOrder="1"
diffgram:hasErrors="true">
      <CustomerID>ANATR</CustomerID>
      <CompanyName>Ana Trujillo Emparedados y Helados</CompanyName>
    </Customers>
    <Customers diffgr:id="Customers3" msdata:rowOrder="2">
      <CustomerID>ANTON</CustomerID>
      <CompanyName>Antonio Moreno Taquera</CompanyName>
    </Customers>
    <Customers diffgr:id="Customers4" msdata:rowOrder="3">
      <CustomerID>AROUT</CustomerID>
      <CompanyName>Around the Horn</CompanyName>
    </Customers>
  </CustomerDataSet>
  <diffgr:before>
    <Customers diffgr:id="Customers1" msdata:rowOrder="0">
      <CustomerID>ALFKI</CustomerID>
      <CompanyName>Alfreds Futterkiste</CompanyName>
    </Customers>
  </diffgr:before>
  <diffgr:errors>
    <Customers diffgr:id="Customers2" diffgr:Error="An optimistic
concurrency violation has occurred for this row."/>
  </diffgr:errors>
</diffgr:diffgram>
```

5.12.2. Cargar DataSet desde un XML

Es posible crear el contenido de un DataSet de ADO.NET a partir de una secuencia o de un documento XML. Además, con .NET Framework se dispone de una gran flexibilidad sobre qué información se carga desde XML y cómo se crea el esquema o la estructura relacional del DataSet.

Para rellenar un DataSet con datos XML, utilice el método **ReadXml** del objeto DataSet. El método **ReadXml** lee desde un archivo, una secuencia o un **XmlReader** y toma como argumentos el origen de XML y un argumento **XmlReadMode** opcional. Para obtener más información sobre el **XmlReader**, vea Leer datos XML con XmlTextReader. El método **ReadXml** lee el contenido de la secuencia o el documento XML y carga datos en el DataSet. También crea el esquema relacional del DataSet dependiendo del **XmlReadMode** especificado y de si ya existe o no un esquema relacional.

En la tabla siguiente se describen las opciones para el argumento **XmlReadMode**.

Opción	Descripción
Automático	Éste es el valor predeterminado. Examina el código XML y elige la opción más apropiada, en el orden siguiente: • Si el código XML es un DiffGram, se utiliza **DiffGram**. • Si el DataSet contiene un esquema o el código XML contiene un esquema en línea, se utiliza **ReadSchema**. • Si el DataSet no contiene un esquema y el código XML no contiene un esquema en línea, se utiliza **InferSchema**. Si conoce el formato del código XML que se está leyendo, para mejorar el rendimiento se recomienda establecer un **XmlReadMode** explícito en lugar de permitir el uso del valor predeterminado **Auto**.
ReadSchema	Lee cualquier esquema en línea y carga los datos y el esquema. Si el DataSet ya contiene un esquema, se agregan nuevas tablas del esquema en línea al esquema existente en el DataSet. Si ya existe alguna tabla del esquema en línea en el DataSet, se iniciará una excepción. No podrá modificar el esquema de una tabla existente mediante **XmlReadMode.ReadSchema**. Si el DataSet no contiene un esquema y no hay ningún esquema en línea, no se leerá ningún dato. Es posible definir el esquema en línea mediante el esquema del lenguaje de definición de esquemas XML (XSD).
IgnoreSchema	Pasa por alto cualquier esquema en línea y carga los datos en el esquema del DataSet existente. Se descartan todos los datos que no coincidan con el esquema existente. Si no existe ningún esquema en el DataSet, no se cargará ningún dato. Si los datos son un DiffGram, **IgnoreSchema** tiene la misma funcionalidad que **DiffGram**.
InferSchema	Pasa por alto cualquier esquema en línea, deduce el esquema por la estructura de los datos XML y, a continuación, carga los datos. Si el DataSet ya contiene un esquema, se extiende el esquema actual mediante la adición de columnas a las tablas existentes. Si no existen tablas, no se agregarán tablas adicionales.Se iniciará una excepción si ya existe una tabla inferida con un espacio de nombres diferente o si alguna columna inferida entra en conflicto con columnas existentes.
DiffGram	Lee un DiffGram y agrega los datos al esquema actual. **DiffGram** combina las filas nuevas con las filas existentes en las que coinciden los valores de identificador.
Fragmento	Sigue leyendo varios fragmentos de XML hasta llegar al final de la secuencia. Los fragmentos que coinciden con el esquema del DataSet se anexan a las tablas apropiadas. Los fragmentos que no coinciden con el esquema del DataSet se descartan.

Nota:

Si pasa un **XmlReader** a un **ReadXml** que se encuentra en medio de un documento XML, **ReadXml** leerá hasta el siguiente nodo de elemento y lo considerará un elemento raíz, leyendo hasta el final del nodo de elemento únicamente. Esto no se aplica si especifica **XmlReadMode.Fragment**.

Entidades DTD

Si el código XML contiene entidades definidas en un esquema de definición de tipo de documento (DTD), se iniciará una excepción si intenta cargar un DataSet pasando a **ReadXml** un nombre de archivo, una secuencia o un **XmlReader** sin validación. En su lugar, debe crear un **XmlValidatingReader**, donde **EntityHandling** tenga el valor **EntityHandling.ExpandEntities**, y pasar el **XmlValidatingReader** a **ReadXml**. El **XmlValidatingReader** expandirá las entidades antes de que las lea el DataSet.

En los siguientes ejemplos de código se muestra cómo cargar un DataSet desde una secuencia XML. En el primer ejemplo se muestra un nombre de archivo que se pasa al método **ReadXml**. En el segundo ejemplo se muestra una cadena que contiene el código XML que se carga mediante un StringReader.

```
Dim dataSet As DataSet = New DataSet
dataSet.ReadXml("input.xml", XmlReadMode.ReadSchema)
Dim dataSet As DataSet = New DataSet
Dim dataTable As DataTable = New DataTable("table1")
dataTable.Columns.Add("col1", Type.GetType("System.String"))
dataSet.Tables.Add(dataTable)
Dim xmlData As String =
"<XmlDS><table1><col1>Value1</col1></table1><table1><col1>Value2</col1></
table1></XmlDS>"
Dim xmlSR As System.IO.StringReader = New System.IO.StringReader(xmlData)
dataSet.ReadXml(xmlSR, XmlReadMode.IgnoreSchema)
```

Nota:

Si llama a **ReadXml** para cargar un archivo muy grande, el rendimiento puede resultar muy lento. Para garantizar un rendimiento óptimo de **ReadXml** (en un archivo de gran tamaño) llame al método BeginLoadData para cada una de las tablas del DataSety, a continuación, llame a **ReadXml**. Por último, llame a EndLoadData para cada una de las tablas del DataSet, tal y como se muestra en el siguiente ejemplo.

```
Dim dataTable As DataTable
For Each dataTable In dataSet.Tables
   dataTable.BeginLoadData()
Next
dataSet.ReadXml("file.xml")
For Each dataTable in dataSet.Tables
   dataTable.EndLoadData()
Next
```

Nota:

Si el esquema XSD de un DataSet incluye un **targetNamespace**, no se podrán leer los datos y pueden aparecer excepciones al llamar a **ReadXml** para cargar el DataSet con XML que contenga elementos con un espacio de nombres sin certificar. En este caso, para leer los elementos no certificados, establezca **elementFormDefault** en "completo" en el esquema XSD. Por ejemplo:

```
<xsd:schema id="customDataSet"
  elementFormDefault="qualified"
```

```
  targetNamespace="http://www.tempuri.org/customDataSet.xsd"
  xmlns="http://www.tempuri.org/customDataSet.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
</xsd:schema>
```

Combinar datos a partir de XML

Si el DataSet ya contiene datos, los nuevos datos procedentes del código XML se agregarán a los ya presentes en el DataSet. **ReadXml** no combina a partir de XML en el DataSet ninguna información de fila que tenga claves principales coincidentes. Si desea sobrescribir información de fila existente con nueva información procedente de XML, utilice **ReadXml** para crear un nuevo DataSet y, a continuación, realice una operación Merge el nuevo DataSet en el DataSet existente. Hay que tener en cuenta que al cargar un DiffGram mediante un **ReadXML** cuyo valor de **XmlReadMode** es **DiffGram** se combinarán las filas que tengan el mismo identificador único.

5.12.3. Escribir Contenido del DataSet como Datos XML

En ADO.NET es posible escribir una representación XML de un DataSet, con o sin su esquema. Si la información de esquema está incluida en línea con el código XML, se escribirá con el lenguaje de definición de esquemas XML (XSD). El esquema contiene las definiciones de tabla del DataSet, así como las definiciones de relaciones y restricciones.

Cuando un DataSet se escribe como datos XML, las filas del DataSet se escriben en sus versiones actuales. Sin embargo, el DataSet también se puede escribir como un DiffGram, de forma que se incluyan los valores actuales y originales de las filas.

La representación XML del DataSet se puede escribir en un archivo, una secuencia, un **XmlWriter** o una cadena. Estas opciones ofrecen una gran flexibilidad en cuanto a la forma de transportar la representación XML del DataSet. Para obtener la representación XML del DataSet como una cadena, utilice el método **GetXml**, como se muestra en el ejemplo siguiente.

```
Dim xmlDS As String = custDS.GetXml()
```

GetXml devuelve la representación XML del DataSet sin información del esquema. Para escribir la información de esquema del DataSet (como esquema XML) en una cadena, utilice **GetXmlSchema**.

Para escribir un DataSet en un archivo, una secuencia o **XmlWriter**, utilice el método **WriteXml**. El primer parámetro que se pasa a **WriteXml** es el destino del resultado XML. Por ejemplo, pasar una cadena que contiene un nombre de archivo, un objeto **System.IO.TextWriter**, etc. Puede pasar un segundo parámetro opcional de un **XmlWriteMode** para especificar cómo se va a escribir el resultado XML.

En la siguiente tabla se muestran las opciones de **XmlWriteMode**.

Opción XmlWriteMode	Descripción
IgnoreSchema	Escribe el contenido actual del DataSet como datos XML, sin un esquema XML. Éste es el valor predeterminado.
WriteSchema	Escribe el contenido actual del DataSet como datos XML con la estructura relacional como un esquema XML en línea.

DiffGram	Escribe el DataSet completo como un DiffGram, incluidos los valores originales y actuales.

Al escribir una representación XML de un DataSet que contiene objetos **DataRelation**, lo más probable es que desee que el código XML resultante tenga las filas secundarias de cada relación anidadas dentro de sus elementos primarios relacionados. Para ellos establezca la propiedad **Nested** de la **DataRelation** en **true** cuando agregue la **DataRelation** al DataSet.

A continuación se muestran dos ejemplos de cómo escribir la representación XML de un DataSet en un archivo. En el primer ejemplo, el nombre de archivo del código XML resultante se pasa como una cadena a **WriteXml**. En el segundo ejemplo se pasa un objeto **System.IO.StreamWriter**.

```
custDS.WriteXml("Customers.xml", XmlWriteMode.WriteSchema)
Dim xmlSW As System.IO.StreamWriter = New
System.IO.StreamWriter("Customers.xml")
custDS.WriteXml(xmlSW, XmlWriteMode.WriteSchema)
xmlSW.Close()
```

⊟ Asignar columnas a elementos, atributos y texto XML

Se puede especificar cómo se representará en XML una columna de una tabla si se utiliza la propiedad **ColumnMapping** del objeto **DataColumn**. En la siguiente tabla se muestran los distintos valores de **MappingType** para la propiedad **ColumnMapping** de una columna de tabla y el código XML resultante.

Valor MappingType	Descripción
Elemento	Éste es el valor predeterminado. La columna se escribe como un elemento XML, donde ColumnName es el nombre del elemento y el contenido de la columna se escribe como el texto del elemento. Por ejemplo: `<ColumnName>Column Contents</ColumnName>`
Attribute	La columna se escribe como un atributo XML del elemento XML para la fila actual, donde ColumnName es el nombre del atributo y el contenido de la columna se escribe como el valor del atributo. Por ejemplo: `<RowElement ColumnName="Column Contents" />`
SimpleContent	El contenido de la columna se escribe como texto en el elemento XML de la fila actual. Por ejemplo: `<RowElement>Column Contents</RowElement>` Hay que tener en cuenta que **SimpleContent** no se puede establecer para una columna de una tabla que tenga columnas **Element** o relaciones anidadas.
Hidden	La columna no se escribe en el resultado XML.

5.12.4. Cargar la Información de Esquema de un DataSet desde XML

El esquema de un DataSet (sus tablas, columnas, relaciones y restricciones) se puede definir mediante programación, crear mediante los métodos **Fill** o **FillSchema** de un DataAdapter o cargar desde un documento XML. Para cargar información de esquema de un **DataSet** desde un documento XML puede utilizar el método **ReadXmlSchema** o **InferXmlSchema** del **DataSet**. **ReadXmlSchema** permite cargar o deducir la información de esquema de un **DataSet** desde el documento que contiene el esquema de lenguaje de definición de esquema XML (XSD) o desde un documento XML con un esquema

XML en línea. **InferXmlSchema** le permite deducir el esquema a partir del documento XML y pasar por alto ciertos espacios de nombres XML que especifique.

Nota:

Es posible que el orden de las tablas de un objeto **DataSet** no se conserve si usa servicios web o serialización XML para transferir un objeto **DataSet** creado en memoria mediante construcciones XSD (como las relaciones anidadas). Por tanto, en este caso el destinatario del objeto **DataSet** no debe depender del orden de las tablas. Sin embargo, el orden de las tablas siempre se mantiene si el esquema del objeto **DataSet** que se transfiere se recupera de archivos XSD, en lugar de crearse en memoria.

ReadXmlSchema

Para cargar el esquema de un **DataSet** desde un documento XML sin cargar ningún dato, puede utilizar el método **ReadXmlSchema** del **DataSet**. **ReadXmlSchema** crea el esquema del **DataSet** definido mediante el esquema del lenguaje de definición de esquemas XML (XSD).

El método **ReadXmlSchema** toma un único argumento, que puede ser un nombre de archivo, una secuencia o un **XmlReader** que contiene el documento XML que se va a cargar. El documento XML puede contener únicamente el esquema o puede contener el esquema en línea con elementos XML que contienen datos.

Si el documento XML pasado a **ReadXmlSchema** no contiene información de esquema en línea, **ReadXmlSchema** deducirá el esquema a partir de los elementos del documento XML. Si el objeto **DataSet** ya contiene un esquema, se agregarán nuevas tablas si no existen ya para extender el esquema actual. En las tablas existentes no se agregarán nuevas columnas.Si una columna que se va a agregar ya existe en el **DataSet** pero tiene un tipo incompatible con la columna encontrada en el código XML, se iniciará una excepción.

Mientras que **ReadXmlSchema** carga o deduce únicamente el esquema de un **DataSet**, el método **ReadXml** del **DataSet** carga o deduce el esquema y los datos contenidos en el documento XML.

En los siguientes ejemplos de código se muestra cómo cargar el esquema de un **DataSet** desde un documento o una secuencia XML. En el primer ejemplo se muestra cómo se pasa un nombre de archivo de esquema XML al método **ReadXmlSchema**. En el segundo ejemplo se muestra un objeto **System.IO.StreamReader**.

```
Dim dataSet As DataSet = New DataSet
dataSet.ReadXmlSchema("schema.xsd")
Dim xmlStream As System.IO.StreamReader = New System.IO.StreamReader
("schema.xsd")
Dim dataSet As DataSet = New DataSet
dataSet.ReadXmlSchema(xmlStream)
xmlStream.Close()
```

InferXmlSchema

También puede indicar al **DataSet** que deduzca su esquema a partir de un documento XML mediante el método **InferXmlSchema** del **DataSet**. **InferXmlSchema** funciona del mismo modo que **ReadXml** con un **XmlReadMode** de **InferSchema** (carga datos y deduce el esquema) y **ReadXmlSchema** si el documento que se lee no contiene ningún esquema en línea. Sin embargo, **InferXmlSchema** ofrece la posibilidad adicional de especificar que se pasen por alto determinados espacios de nombres XML cuando se deduzca el esquema. **InferXmlSchema** acepta dos argumentos necesarios: la ubicación del

documento XML (especificada por un nombre de archivo, una secuencia o un **XmlReader**) y una matriz de cadena de espacios de nombres XML que la operación debe pasar por alto.

Por ejemplo, tomemos el siguiente código XML:

```
<NewDataSet xmlns:od="urn:schemas-microsoft-com:officedata">
<Categories>
  <CategoryID od:adotype="3">1</CategoryID>
  <CategoryName od:maxLength="15"
od:adotype="130">Beverages</CategoryName>
  <Description od:adotype="203">Soft drinks and teas</Description>
</Categories>
<Products>
  <ProductID od:adotype="20">1</ProductID>
  <ReorderLevel od:adotype="3">10</ReorderLevel>
  <Discontinued od:adotype="11">0</Discontinued>
</Products>
</NewDataSet>
```

Debido a los atributos especificados para los elementos en el documento XML anterior, tanto el método **ReadXmlSchema** como el método **ReadXml** con un **XmlReadMode** de **InferSchema** crean tablas para todos los elementos del documento: **Categories**, **CategoryID**, **CategoryName**, **Description**, **Products**, **ProductID**, **ReorderLevel** y **Discontinued**. Sin embargo, una estructura más adecuada consistiría en crear únicamente las tablas **Categories** y **Products** y, a continuación, crea las columnas **CategoryID**, **CategoryName** y **Description** en la tabla **Categories** y las columnas **ProductID**, **ReorderLevel** y **Discontinued** en la tabla **Products**. Para asegurarse de que el esquema inferido pasa por alto los atributos especificados en los elementos XML, utilice el método **InferXmlSchema** y especifique que se debe pasar por alto el espacio de nombres XML para **officedata**, como se muestra en el siguiente ejemplo.

```
Dim dataSet As DataSet = New DataSet
dataSet.InferXmlSchema("input_od.xml", New String() {"urn:schemas-
microsoft-com:officedata"})
```

5.12.5. Escribir la Información de Esquema de un DataSet como XSD

Puede escribir el esquema de un DataSet como un esquema de lenguaje de definición de esquemas XML (XSD), de forma que pueda transportarlo, con o sin datos relacionados, a un documento XML. El esquema XML, que se puede escribir en un archivo, una secuencia, un XmlWriter o una cadena, es útil para generar un **DataSet** con establecimiento inflexible de tipos.

Puede especificar cómo se representa una columna de una tabla en el esquema XML mediante la propiedad **ColumnMapping** del objeto DataColumn.

Para escribir el esquema de un **DataSet** como un esquema XML en un archivo, una secuencia o **XmlWriter**, utilice el método **WriteXmlSchema** del **DataSet**. **WriteXmlSchema** toma un parámetro que especifica el destino del esquema XML resultante. En los siguientes ejemplos de código se muestra cómo escribir el esquema XML de un **DataSet** en un archivo si se pasa una cadena que contiene un nombre de archivo y un objeto StreamWriter.

```
dataSet.WriteXmlSchema("Customers.xsd")
Dim writer As System.IO.StreamWriter = New
System.IO.StreamWriter("Customers.xsd")
dataSet.WriteXmlSchema(writer)
writer.Close()
```

Para obtener el esquema de un **DataSet** y escribirlo como una cadena de esquema XML, utilice el método **GetXmlSchema** como se muestra en el ejemplo siguiente.

```
Dim schemaString As String = dataSet.GetXmlSchema()
```

5.12.6. Sincronización del DataSet y XmlDataDocument

El DataSet de ADO.NET proporciona una representación relacional de datos. Para el acceso a datos jerárquicos puede utilizar las clases XML disponibles en .NET Framework. Históricamente, estas dos representaciones de datos se han utilizado independientemente. Sin embargo, .NET Framework habilita el acceso sincrónico en tiempo real tanto a la representación relacional como a la representación jerárquica de los datos mediante los objetos **DataSet** y XmlDataDocument, respectivamente.

Cuando un **DataSet** se sincroniza con un **XmlDataDocument**, ambos objetos trabajan con un único conjunto de datos. Esto significa que si se realiza un cambio en el **DataSet**, dicho cambio quedará reflejado en el **XmlDataDocument**, y viceversa. La relación entre el **DataSet** y el **XmlDataDocument** ofrece una gran flexibilidad, ya que permite que una única aplicación, utilizando un único conjunto de datos, tenga acceso a todo el conjunto de servicios existentes en el **DataSet** (como controles de formularios Web Forms y formularios Windows Forms, y diseñadores de Visual Studio .NET), así como al conjunto de servicios XML, incluyendo XSL (Extensible Stylesheet Language), XSLT (XSL Transformations) y el lenguage de rutas XML (XPath). No tiene que elegir el conjunto de servicios a los que desea dirigir la aplicación, ya que ambos están disponibles.

Hay varias formas de sincronizar un **DataSet** con un **XmlDataDocument**. Puede:

- Llenar un **DataSet** con el esquema (estructura relacional) y los datos y, después, sincronizarlo con un nuevo **XmlDataDocument**. Esto ofrece una vista jerárquica de los datos relacionales existentes. Por ejemplo:

  ```
  Dim dataSet As DataSet = New DataSet
  ' Add code here to populate the DataSet with schema and data.
  Dim xmlDoc As XmlDataDocument = New XmlDataDocument(dataSet)
  ```

- Llenar un **DataSet** con el esquema únicamente (como un **DataSet** con establecimiento inflexible de tipos), sincronizarlo con un **XmlDataDocument** y cargar el **XmlDataDocument** desde un documento XML. Esto ofrece una vista relacional de los datos jerárquicos existentes. Los nombres de tabla y de columna del esquema del **DataSet** deben coincidir con los nombres de los elementos XML con los que desea sincronizarlos. La coincidencia distingue mayúsculas de minúsculas.

  Hay que tener en cuenta que en el esquema del **DataSet** sólo es necesario que coincidan los elementos XML que desee exponer en la vista relacional. De esta forma puede tener un documento XML muy grande y una "ventana" relacional muy pequeña de ese documento. El **XmlDataDocument** conserva todo el documento XML, incluso aunque el **DataSet** sólo exponga una pequeña parte del mismo.

En el siguiente ejemplo de código se muestran los pasos para crear un **DataSet**, llenar su esquema y, a continuación, sincronizarlo con un **XmlDataDocument**. Hay que tener en cuenta que en el esquema del **DataSet** sólo es necesario que coincidan los elementos del **XmlDataDocument** que desee exponer mediante el **DataSet**.

```
Dim dataSet As DataSet = New DataSet
' Add code here to populate the DataSet with schema, but not data.
Dim xmlDoc As XmlDataDocument = New XmlDataDocument(dataSet)
xmlDoc.Load("XMLDocument.xml")
```

No puede cargar un **XmlDataDocument** si está sincronizado con un **DataSet** que contiene datos. Se iniciará una excepción.

- Crear un nuevo **XmlDataDocument**, cargarlo desde un documento XML y, a continuación, tener acceso a la vista relacional de los datos mediante la propiedad **DataSet** del **XmlDataDocument**. Debe establecer el esquema del **DataSet** antes de poder ver cualquiera de los datos del **XmlDataDocument** mediante el **DataSet**. Una vez más, los nombres de tabla y de columna del esquema del **DataSet** deben coincidir con los nombres de los elementos XML con los que desea sincronizarlos. La coincidencia distingue mayúsculas de minúsculas.

  En el siguiente ejemplo de código se muestra cómo tener acceso a la vista relacional de los datos de un **XmlDataDocument**.

```
Dim xmlDoc As XmlDataDocument = New XmlDataDocument
Dim dataSet As DataSet = xmlDoc.DataSet
'Add code here to create the schema of the DataSet to view the data
xmlDoc.Load("XMLDocument.xml")
```

Otra ventaja de sincronizar un **XmlDataDocument** con un **DataSet** es que se conserva la fidelidad de un documento XML. Si el **DataSet** se llena desde un documento XML mediante **ReadXml**, cuando se vuelvan a escribir los datos como un documento XML mediante **WriteXml** pueden diferir considerablemente con respecto al documento XML original. Esto se debe a que el **DataSet** no conserva el formato, como el espacio en blanco, ni la información jerárquica, como el orden de los elementos, del documento XML. El **DataSet** tampoco contiene elementos del documento XML que se omitieron porque no coincidían con el esquema del **Dataset**. La sincronización de un **XmlDataDocument** con un **DataSet** permite conservar el formato y la estructura jerárquica de los elementos del documento XML original en el **XmlDataDocument**, mientras que el **DataSet** sólo contiene los datos y la información de esquema apropiados para el **DataSet**.

Al sincronizar un **DataSet** con un **XmlDataDocument**, los resultados obtenidos pueden diferir dependiendo de si los objetos DataRelation están o no anidados.

5.12.6.1. Sincronizar el DataSet con el XmlDataDocument

En esta sección se muestra un paso del procesamiento de una orden de compra, donde se utiliza un DataSet con establecimiento inflexible de tipos sincronizado con un XmlDataDocument. En el ejemplo siguiente se crea un **DataSet** con un esquema minimizado que sólo coincide con una parte del documento XML de origen. En los ejemplos se utiliza un **XmlDataDocument** para conservar la fidelidad del documento XML de origen, lo que permite utilizar el **DataSet** para exponer un subconjunto del documento XML.

El siguiente documento XML contiene toda la información relativa a una orden de compra: información del cliente, artículos pedidos, información de envío, etc.

```
<?xml version="1.0" standalone="yes"?>
<PurchaseOrder>
  <Customers>
    <CustomerID>CHOPS</CustomerID>
    <Orders>
      <OrderID>10966</OrderID>
      <OrderDetails>
        <OrderID>10966</OrderID>
        <ProductID>37</ProductID>
        <UnitPrice>26</UnitPrice>
        <Quantity>8</Quantity>
        <Discount>0</Discount>
      </OrderDetails>
      <OrderDetails>
        <OrderID>10966</OrderID>
        <ProductID>56</ProductID>
        <UnitPrice>38</UnitPrice>
        <Quantity>12</Quantity>
        <Discount>0.15</Discount>
      </OrderDetails>
      <OrderDetails>
        <OrderID>10966</OrderID>
        <ProductID>62</ProductID>
        <UnitPrice>49.3</UnitPrice>
        <Quantity>12</Quantity>
        <Discount>0.15</Discount>
      </OrderDetails>
      <CustomerID>CHOPS</CustomerID>
      <EmployeeID>4</EmployeeID>
      <OrderDate>1998-03-20T00:00:00.0000000</OrderDate>
      <RequiredDate>1998-04-17T00:00:00.0000000</RequiredDate>
      <ShippedDate>1998-04-08T00:00:00.0000000</ShippedDate>
      <ShipVia>1</ShipVia>
      <Freight>27.19</Freight>
      <ShipName>Chop-suey Chinese</ShipName>
      <ShipAddress>Hauptstr. 31</ShipAddress>
      <ShipCity>Bern</ShipCity>
      <ShipPostalCode>3012</ShipPostalCode>
      <ShipCountry>Switzerland</ShipCountry>
    </Orders>
    <CompanyName>Chop-suey Chinese</CompanyName>
    <ContactName>Yang Wang</ContactName>
    <ContactTitle>Owner</ContactTitle>
    <Address>Hauptstr. 29</Address>
    <City>Bern</City>
    <PostalCode>3012</PostalCode>
    <Country>Switzerland</Country>
```

```
    <Phone>0452-076545</Phone>
  </Customers>
  <Shippers>
    <ShipperID>1</ShipperID>
    <CompanyName>Speedy Express</CompanyName>
    <Phone>(503) 555-0100</Phone>
  </Shippers>
  <Shippers>
    <ShipperID>2</ShipperID>
    <CompanyName>United Package</CompanyName>
    <Phone>(503) 555-0101</Phone>
  </Shippers>
  <Shippers>
    <ShipperID>3</ShipperID>
    <CompanyName>Federal Shipping</CompanyName>
    <Phone>(503) 555-0102</Phone>
  </Shippers>
  <Products>
    <ProductID>37</ProductID>
    <ProductName>Gravad lax</ProductName>
    <QuantityPerUnit>12 - 500 g pkgs.</QuantityPerUnit>
    <UnitsInStock>11</UnitsInStock>
    <UnitsOnOrder>50</UnitsOnOrder>
    <ReorderLevel>25</ReorderLevel>
  </Products>
  <Products>
    <ProductID>56</ProductID>
    <ProductName>Gnocchi di nonna Alice</ProductName>
    <QuantityPerUnit>24 - 250 g pkgs.</QuantityPerUnit>
    <UnitsInStock>21</UnitsInStock>
    <UnitsOnOrder>10</UnitsOnOrder>
    <ReorderLevel>30</ReorderLevel>
  </Products>
  <Products>
    <ProductID>62</ProductID>
    <ProductName>Tarte au sucre</ProductName>
    <QuantityPerUnit>48 pies</QuantityPerUnit>
    <UnitsInStock>17</UnitsInStock>
    <UnitsOnOrder>0</UnitsOnOrder>
    <ReorderLevel>0</ReorderLevel>
  </Products>
</PurchaseOrder>
```

Un paso del procesamiento de la información de la orden de compra contenida en el documento XML anterior es para rellenar la orden desde el inventario actual de la compañía. El empleado responsable de rellenar la orden de compra desde el almacén de la compañía no necesita ver todo el contenido de la orden de compra; sólo necesita ver la información de producto. Para exponer únicamente la información de producto desde el documento XML, cree un **DataSet** con establecimiento inflexible de tipos con un esquema, escrito como un esquema del lenguaje de definición de esquema XML (XSD), que se asigne a los productos y cantidades pedidos.

En el código siguiente se muestra el esquema a partir del cual se genera el **DataSet** con establecimiento inflexible de tipos para este ejemplo.

```
<?xml version="1.0" standalone="yes"?>
<xs:schema id="OrderDetail" xmlns=""
                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
                  xmlns:codegen="urn:schemas-microsoft-com:xml-msprop"
                  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="OrderDetail" msdata:IsDataSet="true">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="OrderDetails" codegen:typedName="LineItem"
codegen:typedPlural="LineItems">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="OrderID" type="xs:int" minOccurs="0"
codegen:typedName="OrderID"/>
              <xs:element name="Quantity" type="xs:short" minOccurs="0"
codegen:typedName="Quantity"/>
              <xs:element name="ProductID" type="xs:int" minOccurs="0"
codegen:typedName="ProductID"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Products" codegen:typedName="Product"
codegen:typedPlural="Products">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ProductID" type="xs:int" minOccurs="0"
codegen:typedName="ProductID"/>
              <xs:element name="ProductName" type="xs:string"
minOccurs="0" codegen:typedName="ProductName"/>
              <xs:element name="QuantityPerUnit" type="xs:string"
minOccurs="0" codegen:typedName="QuantityPerUnit"/>
              <xs:element name="UnitsInStock" type="xs:short"
minOccurs="0" codegen:typedName="UnitsInStock"/>
              <xs:element name="UnitsOnOrder" type="xs:short"
minOccurs="0" codegen:typedName="UnitsOnOrder"/>
              <xs:element name="ReorderLevel" type="xs:short"
minOccurs="0" codegen:typedName="ReorderLevel"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
    <xs:unique name="Constraint1">
      <xs:selector xpath=".//Products" />
      <xs:field xpath="ProductID" />
    </xs:unique>
```

```
    <xs:keyref name="Relation1" refer="Constraint1"
codegen:typedChildren="GetLineItems" codegen:typedParent="Product">
      <xs:selector xpath=".//OrderDetails" />
      <xs:field xpath="ProductID" />
    </xs:keyref>
  </xs:element>
</xs:schema>
```

Observe que en el esquema del **DataSet** sólo se incluye información de los elementos **OrderDetails** y **Products** del documento XML original. La sincronización del **DataSet** con un **XmlDataDocument** garantiza que los elementos no incluidos en el **DataSet** persistirán con el documento XML.

Con el **DataSet** con establecimiento inflexible de tipos generado a partir del esquema XML (con un espacio de nombres **Northwind.FillOrder**), se puede exponer una parte del documento XML original si sincroniza el **DataSet** con el **XmlDataDocument** cargado desde el documento XML de origen. Hay que tener en cuenta que el **DataSet** generado desde el esquema contiene estructura, pero no datos. Los datos se rellenan al cargar el XML en el **XmlDataDocument**. Si intenta cargar un **XmlDataDocument** sincronizado con un **DataSet** que ya contiene datos, se iniciará una excepción.

Una vez actualizado el **DataSet** (y el **XmlDataDocument**), el **XmlDataDocument** puede escribir el documento XML modificado con los elementos omitidos por el **DataSet** todavía intactos, como se muestra a continuación. En el escenario de la orden de compra, una vez rellenados los artículos pedidos, se puede pasar el documento XML modificado al siguiente paso del proceso de pedido, quizás al departamento de envíos de la compañía.

```
Imports System
Imports System.Data
Imports System.Xml
Imports Northwind.FillOrder
Public class Sample
  Public Shared Sub Main()
    Dim orderDS As OrderDetail = New OrderDetail
    Dim xmlDocument As XmlDataDocument = New XmlDataDocument(orderDS)
    xmlDocument.Load("Order.xml")
    Dim orderItem As OrderDetail.LineItem
    Dim product As OrderDetail.Product
    For Each orderItem In orderDS.LineItems
      product = orderItem.Product
      ' Remove quantity from the current stock.
      product.UnitsInStock = CType(product.UnitsInStock -
orderItem.Quantity, Short)
    ' If the remaining stock is less than the reorder level, order more.
      If ((product.UnitsInStock + product.UnitsOnOrder) <
product.ReorderLevel) Then
        product.UnitsOnOrder = CType(product.UnitsOnOrder +
product.ReorderLevel, Short)
      End If
    Next
    xmlDocument.Save("Order_out.xml")
  End Sub
End Class
```

5.12.6.2. Realizar una Consulta de XPath en un DataSet

La relación entre un DataSet y un XmlDataDocument sincronizados le permite utilizar servicios XML, como la consulta XPath (XML Path Language), que tiene acceso al **XmlDataDocument** y puede realizar ciertas funciones más cómodamente que si tuviera acceso directamente al **DataSet**. Por ejemplo, en lugar de utilizar el método **Select** de una DataTable para navegar por relaciones con otras tablas de un **DataSet**, puede realizar una consulta de XPath en un **XmlDataDocument** sincronizado con el **DataSet** para obtener una lista de elementos XML en forma de una XmlNodeList. Los nodos de la **XmlNodeList**, convertidos en nodos XmlElement, se pueden pasar entonces al método **GetRowFromElement** del **XmlDataDocument** para devolver referencias de DataRow coincidentes con las filas de la tabla del **DataSet** sincronizado.

Por ejemplo, en el siguiente ejemplo de código se realiza una consulta "secundaria" de XPath. El **DataSet** se rellena con tres tablas: **Customers**, **Orders** y **OrderDetails**. En el ejemplo se crea primero una relación primaria-secundaria entre las tablas **Customers** y **Orders**, y entre las tablas **Orders** y **OrderDetails**. Después se realiza una consulta XPath para devolver una **XmlNodeList** de nodos de **Customers**, donde un nodo secundario de **OrderDetails** tiene un nodo **ProductID** con el valor 43. El ejemplo utiliza la consulta XPath para determinar qué clientes han pedido el producto cuyo **ProductID** es 43.

```
' Assumes that connection is a valid SqlConnection.
connection.Open()
Dim dataSet As DataSet = New DataSet("CustomerOrders")
Dim customerAdapter As SqlDataAdapter = New SqlDataAdapter( _
  "SELECT * FROM Customers", connection)
customerAdapter.Fill(dataSet, "Customers")
Dim orderAdapter As SqlDataAdapter = New SqlDataAdapter( _
  "SELECT * FROM Orders", connection)
orderAdapter.Fill(dataSet, "Orders")
Dim detailAdapter As SqlDataAdapter = New SqlDataAdapter( _
  "SELECT * FROM [Order Details]", connection)
detailAdapter.Fill(dataSet, "OrderDetails")
connection.Close()
dataSet.Relations.Add("CustOrders", _
dataSet.Tables("Customers").Columns("CustomerID"), _
dataSet.Tables("Orders").Columns("CustomerID")).Nested = true
dataSet.Relations.Add("OrderDetail", _
  dataSet.Tables("Orders").Columns("OrderID"), _
dataSet.Tables("OrderDetails").Columns("OrderID"), false).Nested = true
Dim xmlDoc As XmlDataDocument = New XmlDataDocument(dataSet)
Dim nodeList As XmlNodeList = xmlDoc.DocumentElement.SelectNodes( _
  "descendant::Customers[*/OrderDetails/ProductID=43]")
Dim dataRow As DataRow
Dim xmlNode As XmlNode
For Each xmlNode In nodeList
  dataRow = xmlDoc.GetRowFromElement(CType(xmlNode, XmlElement))
  If Not dataRow Is Nothing then Console.WriteLine(xmlRow(0).ToString())
Next
```

5.12.6.3. Aplicar una Transformación XSL a un DataSet

El método **WriteXml** del DataSet le permite escribir el contenido de un **DataSet** como datos XML. Normalmente se transforma ese XML en otro formato mediante transformaciones XSL (XSLT). Sin embargo, la sincronización de un **DataSet** con un XmlDataDocument le permite aplicar una hoja de estilos de XSLT al contenido de un **DataSet** sin tener que escribir primero el contenido del **DataSet** como datos XML mediante **WriteXml**.

En el siguiente ejemplo se llena un **DataSet** con tablas y relaciones, se sincroniza el **DataSet** con un **XmlDataDocument** y se escribe una parte del **DataSet** como un archivo HTML utilizando una hoja de estilos de XSLT. A continuación se muestra el contenido de la hoja de estilos XSLT.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match="CustomerOrders">
  <HTML>
  <STYLE>
  BODY {font-family:verdana;font-size:9pt}
  TD   {font-size:8pt}
  </STYLE>
    <BODY>
    <TABLE BORDER="1">
      <xsl:apply-templates select="Customers"/>
    </TABLE>
    </BODY>
  </HTML>
</xsl:template>
<xsl:template match="Customers">
    <TR><TD>
      <xsl:value-of select="ContactName"/>, <xsl:value-of
select="Phone"/><BR/>
    </TD></TR>
      <xsl:apply-templates select="Orders"/>
</xsl:template>
<xsl:template match="Orders">
  <TABLE BORDER="1">
    <TR><TD valign="top"><B>Order:</B></TD><TD valign="top"><xsl:value-of
select="OrderID"/></TD></TR>
    <TR><TD valign="top"><B>Date:</B></TD><TD valign="top"><xsl:value-of
select="OrderDate"/></TD></TR>
    <TR><TD valign="top"><B>Ship To:</B></TD>
        <TD valign="top"><xsl:value-of select="ShipName"/><BR/>
        <xsl:value-of select="ShipAddress"/><BR/>
        <xsl:value-of select="ShipCity"/>, <xsl:value-of
select="ShipRegion"/>  <xsl:value-of select="ShipPostalCode"/><BR/>
        <xsl:value-of select="ShipCountry"/></TD></TR>
  </TABLE>
</xsl:template>
</xsl:stylesheet>
```

En el código siguiente se rellena el **DataSet** y se aplica la hoja de estilos XSLT.

Nota:

Si se aplica una hoja de estilos XSLT a un **DataSet** que contenga relaciones, se obtiene un mayor rendimiento al establecer la propiedad **Nested** de la DataRelation como **true** para cada relación anidada. Esto le permite utilizar hojas de estilos XSLT que implementan un procesamiento natural, de arriba abajo, para navegar por la jerarquía y transformar los datos, en lugar de utilizar ejes de ubicación XPath de rendimiento intensivo (por ejemplo, el elemento anterior y el siguiente en expresiones de prueba de nodos de la hoja de estilos) para navegar por ésta.

```
' Assumes connection is a valid SqlConnection.
Dim dataSet As DataSet = New DataSet("CustomerOrders")
Dim customerAdapter As SqlDataAdapter = New SqlDataAdapter( _
  "SELECT * FROM Customers", connection)
customerAdapter.Fill(dataSet, "Customers")
Dim orderAdapter As SqlDataAdapter = New SqlDataAdapter( _
  "SELECT * FROM Orders", connection)
orderAdapter.Fill(dataSet, "Orders")
connection.Close()
dataSet.Relations.Add("CustOrders", _
dataSet.Tables("Customers").Columns("CustomerID"), _
dataSet.Tables("Orders").Columns("CustomerID")).Nested = true
Dim xmlDoc As XmlDataDocument = New XmlDataDocument(dataSet)
Dim xslTran As XslTransform = New XslTransform
xslTran.Load("transform.xsl")
Dim writer As XmlTextWriter = New XmlTextWriter( _
  "xslt_output.html", System.Text.Encoding.UTF8)
xslTran.Transform(xmlDoc, Nothing, writer)
writer.Close()
```

5.12.7. Anidar DataRelations

En una representación relacional de datos, las tablas individuales contienen filas que están relacionadas entre sí por una columna o un conjunto de columnas. En el DataSet de ADO.NET, la relación entre tablas se implementa mediante una DataRelation. Cuando crea una **DataRelation**, las relaciones primaria-secundaria de las columnas se administran sólo mediante la relación. Las tablas y las columnas son entidades independientes. En la representación jerárquica de los datos proporcionada por XML, las relaciones primaria-secundaria se representan mediante elementos primarios que contienen elementos secundarios anidados.

Para facilitar el anidamiento de objetos secundarios cuando un **DataSet** se sincroniza con un XmlDataDocument o se escribe como XML mediante **WriteXml**, la **DataRelation** expone una propiedad **Nested**. Al establecer la propiedad **Nested** de una **DataRelation** como **true**, las filas secundarias de la relación se anidan dentro de la columna primaria cuando se escriben como datos XML o cuando se sincronizan con un **XmlDataDocument**. La propiedad **Nested** de la **DataRelation** es **false** de manera predeterminada.

Por ejemplo, considere el siguiente **DataSet**.

```
' Assumes connection is a valid SqlConnection.
Dim customerAdapter As SqlDataAdapter = New SqlDataAdapter( _
  "SELECT CustomerID, CompanyName FROM Customers", connection)
Dim orderAdapter As SqlDataAdapter = New SqlDataAdapter( _
```

```
  "SELECT OrderID, CustomerID, OrderDate FROM Orders", connection)
connection.Open()
Dim dataSet As DataSet = New DataSet("CustomerOrders")
customerAdapter.Fill(dataSet, "Customers")
orderAdapter.Fill(dataSet, "Orders")
connection.Close()
Dim customerOrders As DataRelation = dataSet.Relations.Add( _
  "CustOrders", dataSet.Tables("Customers").Columns("CustomerID"), _
  dataSet.Tables("Orders").Columns("CustomerID"))
```

Como la propiedad **Nested** del objeto **DataRelation** no tiene el valor **true** para este **DataSet**, los objetos secundarios no se anidarán dentro de los elementos primarios cuando este **DataSet** se represente como datos XML. La transformación de la representación XML de un objeto **DataSet** que contiene objetos **DataSet** relacionados con relaciones de datos no anidadas puede provocar un rendimiento lento. Se recomienda anidar las relaciones de datos. Para ello, debe establecer la propiedad **Nested** en **True**. A continuación, debe escribir código en la hoja de estilos XSLT que utilice expresiones de consulta XPath con jerarquía de arriba a abajo para localizar y transformar los datos.

En el siguiente ejemplo de código se muestra el resultado de llamar a **WriteXml** en el **DataSet**.

```
<CustomerOrders>
  <Customers>
    <CustomerID>ALFKI</CustomerID>
    <CompanyName>Alfreds Futterkiste</CompanyName>
  </Customers>
  <Customers>
    <CustomerID>ANATR</CustomerID>
    <CompanyName>Ana Trujillo Emparedados y helados</CompanyName>
  </Customers>
  <Orders>
    <OrderID>10643</OrderID>
    <CustomerID>ALFKI</CustomerID>
    <OrderDate>1997-08-25T00:00:00</OrderDate>
  </Orders>
  <Orders>
    <OrderID>10692</OrderID>
    <CustomerID>ALFKI</CustomerID>
    <OrderDate>1997-10-03T00:00:00</OrderDate>
  </Orders>
  <Orders>
    <OrderID>10308</OrderID>
    <CustomerID>ANATR</CustomerID>
    <OrderDate>1996-09-18T00:00:00</OrderDate>
  </Orders>
</CustomerOrders>
```

Hay que tener en cuenta que el elemento **Customers** y los elementos **Orders** se muestran como elementos relacionados. Si desea que los elementos **Orders** aparezcan como secundarios de sus respectivos elementos primarios, la propiedad **Nested** de la **DataRelation** debe ser **true** y se debe agregar lo siguiente:

```
customerOrders.Nested = True
```

En el siguiente código se muestra cuál sería el resultado, con los elementos **Orders** anidados dentro de sus respectivos elementos primarios.

```
<CustomerOrders>
  <Customers>
    <CustomerID>ALFKI</CustomerID>
    <Orders>
      <OrderID>10643</OrderID>
      <CustomerID>ALFKI</CustomerID>
      <OrderDate>1997-08-25T00:00:00</OrderDate>
    </Orders>
    <Orders>
      <OrderID>10692</OrderID>
      <CustomerID>ALFKI</CustomerID>
      <OrderDate>1997-10-03T00:00:00</OrderDate>
    </Orders>
    <CompanyName>Alfreds Futterkiste</CompanyName>
  </Customers>
  <Customers>
    <CustomerID>ANATR</CustomerID>
    <Orders>
      <OrderID>10308</OrderID>
      <CustomerID>ANATR</CustomerID>
      <OrderDate>1996-09-18T00:00:00</OrderDate>
    </Orders>
    <CompanyName>Ana Trujillo Emparedados y helados</CompanyName>
  </Customers>
</CustomerOrders>
```

5.13. Consumir DataSet a partir de un Servicio Web XML

La arquitectura del DataSet tiene un diseño desconectado, en parte para facilitar el transporte de datos a través de Internet. **DataSet** es "serializable" en el sentido de que puede especificarse como entrada o como salida de servicios web XML sin necesidad de programación adicional para transmitir en secuencia el contenido del **DataSet** desde un servicio web XML a un cliente y viceversa. El **DataSet** se convierte implícitamente en una secuencia XML mediante el formato DiffGram, se envía a través de la red y se reconstruye a partir de la secuencia XML como un **DataSet** del extremo receptor. Esto proporciona un método muy sencillo y flexible para transmitir y devolver datos relacionales mediante servicios web XML.

En el siguiente ejemplo se muestra cómo crear un servicio web XML y un cliente que utilice **DataSet** para transportar datos relacionales (incluyendo datos modificados) y resolver cualquier actualización en el origen de datos inicial.

Nota:
Al crear un servicio web XML, es recomendable tener siempre en cuenta las implicaciones de seguridad.

Para crear un servicio web XML que devuelve y consume un DataSet

1. Cree el servicio web XML.

En el ejemplo se crea un servicio web XML que devuelve datos, una lista de clientes de la base de datos **Northwind**, y recibe un **DataSet** con actualizaciones relacionadas con los datos que el servicio web XML resuelve y devuelve al origen de datos inicial.

El servicio web XML expone dos métodos: **GetCustomers**, para devolver la lista de clientes, y **UpdateCustomers**, para resolver las actualizaciones en el origen de datos. El servicio web XML se almacena en un archivo del servidor web denominado DataSetSample.asmx. En el siguiente código se describe el contenido de DataSetSample.asmx.

```
<% @ WebService Language = "vb" Class = "Sample" %>
Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports System.Web.Services
<WebService(Namespace:="http://microsoft.com/webservices/")> _
Public Class Sample
Public connection As SqlConnection = New SqlConnection("Data
Source=(local);Integrated Security=SSPI;Initial Catalog=Northwind")
  <WebMethod( Description := "Returns Northwind Customers",
EnableSession := False )> _
  Public Function GetCustomers() As DataSet
    Dim adapter As SqlDataAdapter = New SqlDataAdapter( _
     "SELECT CustomerID, CompanyName FROM Customers", connection)
    Dim custDS As DataSet = New DataSet()
    adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey
    adapter.Fill(custDS, "Customers")
    Return custDS
  End Function
  <WebMethod( Description := "Updates Northwind Customers",
EnableSession := False )> _
  Public Function UpdateCustomers(custDS As DataSet) As DataSet
    Dim adapter As SqlDataAdapter = New SqlDataAdapter()
    adapter.InsertCommand = New SqlCommand( _
      "INSERT INTO Customers (CustomerID, CompanyName) " & _
      "Values(@CustomerID, @CompanyName)", connection)
    adapter.InsertCommand.Parameters.Add( _
      "@CustomerID", SqlDbType.NChar, 5, "CustomerID")
    adapter.InsertCommand.Parameters.Add( _
      "@CompanyName", SqlDbType.NChar, 15, "CompanyName")
    adapter.UpdateCommand = New SqlCommand( _
      "UPDATE Customers Set CustomerID = @CustomerID, " & _
      "CompanyName = @CompanyName WHERE CustomerID = " & _
      @OldCustomerID", connection)
    adapter.UpdateCommand.Parameters.Add( _
      "@CustomerID", SqlDbType.NChar, 5, "CustomerID")
    adapter.UpdateCommand.Parameters.Add( _
      "@CompanyName", SqlDbType.NChar, 15, "CompanyName")
    Dim parameter As SqlParameter = _
```

```
      adapter.UpdateCommand.Parameters.Add( _
      "@OldCustomerID", SqlDbType.NChar, 5, "CustomerID")
    parameter.SourceVersion = DataRowVersion.Original
    adapter.DeleteCommand = New SqlCommand( _
      "DELETE FROM Customers WHERE CustomerID = @CustomerID", _
      connection)
    parameter = adapter.DeleteCommand.Parameters.Add( _
      "@CustomerID", SqlDbType.NChar, 5, "CustomerID")
    parameter.SourceVersion = DataRowVersion.Original
    adapter.Update(custDS, "Customers")
    Return custDS
  End Function
End Class
```

En una situación típica, se escribiría el método **UpdateCustomers** para detectar infracciones de la simultaneidad optimista. Para simplificar, el ejemplo no lo incluye.

2. Cree un proxy de servicio web XML.

Los clientes del servicio web XML necesitan un proxy SOAP para utilizar los métodos expuestos. Visual Studio puede generar el proxy. Al establecer una referencia web a un servicio web existente desde Visual Studio, el comportamiento descrito en este paso se produce de forma transparente. Si desea crear la clase de proxy personalmente, continúe con la explicación. Sin embargo, el uso de Visual Studio para crear la clase de proxy para la aplicación cliente es suficiente en la mayoría de los casos.

Es posible crear un proxy con la herramienta Lenguaje de descripción de servicios web. Por ejemplo, si el servicio web XML se expone en la dirección URL http://myserver/data/DataSetSample.asmx, hay que utilizar un comando como el siguiente para crear un proxy de Visual Basic .NET con un espacio de nombres de **WebData.DSSample** y almacenarlo en el archivo sample.vb.

```
wsdl /l:VB /out:sample.vb http://myserver/data/DataSetSample.asmx
/n:WebData.DSSample
```

Para crear un proxy de C# en el archivo sample.cs, hay que utilizar el comando siguiente.

```
wsdl /l:CS /out:sample.cs http://myserver/data/DataSetSample.asmx
/n:WebData.DSSample
```

El proxy puede compilarse como una biblioteca e importarse en el cliente del servicio web XML. Para compilar el código de proxy de Visual Basic .NET almacenado en sample.vb como sample.dll, hay que utilizar el siguiente comando.

```
vbc /t:library /out:sample.dll sample.vb /r:System.dll
/r:System.Web.Services.dll /r:System.Data.dll /r:System.Xml.dll
```

Para compilar el código de proxy de C# almacenado en sample.cs como sample.dll, hay que utilizar el siguiente comando.

```
csc /t:library /out:sample.dll sample.cs /r:System.dll
/r:System.Web.Services.dll /r:System.Data.dll /r:System.Xml.dll
```

3. Cree un cliente de servicio web XML.

Si desea que Visual Studio genere la clase de proxy del servicio web, cree el proyecto cliente y, en la ventana del Explorador de soluciones, haga clic con el botón secundario del mouse en el proyecto; a continuación, haga clic en **Agregar referencia web** y seleccione el servicio web en la lista de servicios web disponibles. Puede que sea necesario proporcionar la dirección del punto final de servicios web, en caso de que el servicio web no se encuentre disponible en la solución o en el equipo actuales. Si crea el proxy del servicio web XML personalmente, tal y como se describe en el paso anterior, puede importarlo en el código de cliente y utilizar los métodos de servicio web XML. En el siguiente ejemplo de código se importa la biblioteca de proxy, se llama a **GetCustomers** para obtener una lista de clientes, se agrega un nuevo cliente y se devuelve un **DataSet** con las actualizaciones a**UpdateCustomers**.

Tenga en cuenta que en el ejemplo se pasa el **DataSet** devuelto por **DataSet.GetChanges** a **UpdateCustomers**, ya que sólo es necesario pasar a **UpdateCustomers** las filas modificadas. **UpdateCustomers** devuelve el **DataSet** resuelto, que puede **Merge** con el **DataSet** existente para incorporar los cambios resueltos y cualquier información de error de fila de la actualización. En el código siguiente se da por sentado que se ha utilizado Visual Studio para crear la referencia web y que se ha cambiado el nombre de la misma a DsSample en el cuadro de diálogo **Agregar referencia web**.

```
Imports System
Imports System.Data

Public Class Client
  Public Shared Sub Main()
    Dim proxySample As New DsSample.Sample ()  ' Proxy object.
    Dim customersDataSet As DataSet = proxySample.GetCustomers()
    Dim customersTable As DataTable = _
      customersDataSet.Tables("Customers")
    Dim rowAs DataRow = customersTable.NewRow()
    row("CustomerID") = "ABCDE"
    row("CompanyName") = "New Company Name"
    customersTable.Rows.Add(row)
    Dim updateDataSet As DataSet = _
      proxySample.UpdateCustomers(customersDataSet.GetChanges())
    customersDataSet.Merge(updateDataSet)
    customersDataSet.AcceptChanges()
  End Sub
End Class
```

Si decide crear la clase de proxy personalmente, debe seguir los pasos adicionales que se describen a continuación. Para compilar el ejemplo, indique la biblioteca de proxy creada (sample.dll) y las bibliotecas de .NET relacionadas. Para compilar la versión de Visual Basic .NET del ejemplo, almacenada en el archivo client.vb, hay que utilizar el comando siguiente.

```
vbc client.vb /r:sample.dll /r:System.dll /r:System.Data.dll
/r:System.Xml.dll /r:System.Web.Services.dll
```

6. Manipular Datos

La principal función de cualquier aplicación de base de datos es conectarse a un origen de datos y recuperar los datos contenidos . Los proveedores de datos de .NET Framework para ADO.NET sirven

como puente entre una aplicación y un origen de datos, permitiéndole ejecutar comandos y recuperar datos mediante un **DataReader** o un **DataAdapter**. Una función clave de cualquier aplicación de base de datos es la capacidad de actualizar los datos almacenados en la misma. En ADO.NET, los datos se actualizan mediante **DataAdapter**, DataSet y objetos **Command** y también mediante transacciones.

6.1. Conectar con un Origen de Datos

En ADO.NET se utiliza un objeto **Connection** para conectar con un determinado origen de datos mediante una cadena de conexión en la que se proporciona la información de autenticación necesaria. El objeto **Connection** utilizado depende del tipo de origen de datos.

Cada proveedor de datos de .NET Framework incluido en .NET Framework cuenta con un objeto **Connection**: el proveedor de datos de .NET Framework para OLE DB incluye un objeto OleDbConnection, el proveedor de datos de .NET Framework para SQL Server incluye un objeto SqlConnection, el proveedor de datos de .NET Framework para ODBC incluye un objeto OdbcConnection y el proveedor de datos de .NET Framework para Oracle incluye un objeto OracleConnection.

6.1.1. Establecer la Conexión

Para conectarse a Microsoft SQL Server 7.0 o posterior, utilice el objeto SqlConnection del proveedor de datos de .NET Framework para SQL Server. Para conectarse a un origen de datos OLE DB o a Microsoft SQL Server 6.*x* o una versión anterior, utilice el objeto OleDbConnection del proveedor de datos de .NET Framework para OLE DB. Para conectarse a un origen de datos ODBC, utilice el objeto OdbcConnection del proveedor de datos de .NET Framework para ODBC. Para conectarse a un origen de datos Oracle, utilice el objeto OracleConnection del proveedor de datos de .NET Framework para ODBC.

⊟ Cierre de conexiones

Recomendamos cerrar siempre la conexión cuando termine de utilizarla, para que la conexión pueda regresar al grupo. El bloque **Using** de Visual Basic o C# elimina automáticamente la conexión cuando el código sale del bloque, incluso en el caso de una excepción no controlada.

También puede utilizar los métodos **Close** o **Dispose** del objeto de conexión correspondiente al proveedor que esté utilizando. Es posible que las conexiones que no se cierran explícitamente no se puedan agregar ni puedan regresar al grupo. Por ejemplo, una conexión que se ha salido del ámbito pero que no se ha cerrado explícitamente sólo se devolverá al grupo de conexión si se ha alcanzado el tamaño máximo del grupo y la conexión aún es válida.

Nota:

No llame a **Close** o a **Dispose** en un objeto **Connection**, un objeto **DataReader** o cualquier otro objeto administrado en el método **Finalize** de la clase. En un finalizador, libere sólo los recursos no administrados que pertenezcan directamente a su clase. Si la clase no dispone de recursos no administrados, no incluya un método **Finalize** en la definición de clase.

⊟ Conexión a SQL Server

El proveedor de datos de .NET Framework para SQL Server admite un formato de cadena de conexión que es similar al de OLE DB (ADO). Para consultar los nombres y valores válidos de formato de cadena, vea la propiedad ConnectionString del objeto SqlConnection. También puede usar la clase SqlConnectionStringBuilder para crear cadenas de conexión sintácticamente válidas en tiempo de ejecución.

El siguiente código de ejemplo demuestra cómo crear y abrir una conexión a una base de datos SQL Server 7.0 o posterior.

```
' Assumes connectionString is a valid connection string.
Using connection As New SqlConnection(connectionString)
    connection.Open()
    ' Do work here.
End Using
```

Seguridad integrada y ASP.NET

La seguridad integrada de SQL Server (también conocida como conexiones de confianza) ayuda a proteger las conexiones a SQL Server dado que no expone el id. y la contraseña de un usuario en la cadena de conexión y es el método recomendado para autenticar una conexión. La seguridad integrada utiliza la identidad de seguridad actual, o símbolo (token), del proceso en ejecución, que en aplicaciones de escritorio, es normalmente la identidad del usuario que actualmente ha iniciado la sesión.

La identidad de seguridad para aplicaciones ASP.NET se puede establecer en una de varias opciones diferentes.

⊟ Conexión a un origen de datos OLE DB

El proveedor de datos de .NET Framework para OLE DB proporciona conectividad a orígenes de datos expuestos mediante OLE DB y a Microsoft SQL Server 6.*x* o anterior (a través de SQLOLEDB, el proveedor OLE DB para SQL Server), utilizando el objeto **OleDbConnection**.

En el proveedor de datos de .NET Framework para OLE DB, el formato de cadena de conexión es idéntico al utilizado en ADO, con las siguientes excepciones:

- Se necesita la palabra clave **Proveedor**.
- No se admiten las palabras clave **URL**, **Proveedor remoto** y **Servidor remoto**.

También puede usar OleDbConnectionStringBuilder para crear cadenas de conexión en tiempo de ejecución.

Nota:

El objeto **OleDbConnection** no admite la configuración ni la recuperación de propiedades dinámicas específicas de un proveedor OLE DB. Sólo se admiten las propiedades que se pueden proporcionar en la cadena de conexión para el proveedor OLE DB.

El siguiente código de ejemplo demuestra cómo crear y abrir una conexión a un origen de datos OLE DB.

```
' Assumes connectionString is a valid connection string.
Using connection As New OleDbConnection(connectionString)
    connection.Open()
    ' Do work here.
End Using
```

⊟ No utilice archivos de vínculo de datos universal

Se puede proporcionar información de conexión para un objeto **OleDbConnection** en un archivo de vínculos de datos universal (UDL), pero conviene evitarlo. Los archivos UDL no están cifrados y exponen la información de cadena de conexión en texto sin cifrar. Un archivo UDL no se puede proteger mediante .NET Framework, ya que se trata de un recurso basado en un archivo externo a la aplicación.

Conexión a un origen de datos ODBC

El proveedor de .NET Framework para ODBC proporciona conectividad a orígenes de datos expuestos mediante ODBC utilizando el objeto **OdbcConnection**.

En el proveedor de datos de .NET Framework para ODBC, el formato de cadena de conexión está diseñado para que coincida lo más posible con el de ODBC. También puede proporcionar un nombre de origen de datos (DSN) ODBC.

Nota:

El proveedor de datos de .NET Framework para ODBC no se incluye en .NET Framework 1.0. Si necesita utilizar el proveedor de datos para ODBC de .NET Framework y está utilizando .NET Framework 1.0, puede descargarlo en este sitio web de Microsoft.El espacio de nombres del proveedor de datos de .NET Framework para ODBC descargado es **Microsoft.Data.Odbc**.

El siguiente código de ejemplo demuestra cómo crear y abrir una conexión a un origen de datos ODBC.

```
' Assumes connectionString is a valid connection string.
Using connection As New OdbcConnection(connectionString)
    connection.Open()
    ' Do work here.
End Using
```

Conexión a un origen de datos Oracle

El proveedor de .NET Framework para Oracle proporciona conectividad a orígenes de datos Oracle utilizando el objeto **OracleConnection**.

En el proveedor de datos de .NET Framework para Oracle, el formato de cadena de conexión está diseñado para que coincida lo más posible con el del proveedor OLE DB para Oracle (MSDAORA).

El siguiente código de ejemplo demuestra cómo crear y abrir una conexión a un origen de datos Oracle.

```
' Assumes connectionString is a valid connection string.
Using connection As New OracleConnection(connectionString)
    connection.Open()
    ' Do work here.
End Using
```

6.1.2. Eventos de Conexión

Todos los proveedores de datos de .NET Framework tienen objetos **Connection** con dos eventos que se pueden utilizar para recuperar mensajes informativos de un origen de datos o para determinar si ha cambiado el estado de un objeto **Connection**. En la tabla siguiente se enumeran los eventos del objeto **Connection**.

Evento	Descripción
InfoMessage	Se produce cuando se devuelve un mensaje informativo desde un origen de datos. Los mensajes informativos son aquellos procedentes de orígenes de datos que no inician una excepción.
StateChange	Se produce cuando cambia el estado del objeto **Connection**.

Trabajar con el evento InfoMessage

Con el evento InfoMessage del objeto SqlConnection puede recuperar advertencias o mensajes informativos de un origen de datos de SQL Server. Si se devuelven errores desde el origen de datos con un nivel de seguridad entre 11 y 16, se inicia una excepción. Sin embargo, el evento InfoMessage se puede utilizar para obtener mensajes del origen de datos que no estén asociados a un error. En el caso de Microsoft SQL Server, cualquier error que tenga la gravedad 10, como máximo, se considera de tipo informativo y se captura mediante el evento InfoMessage. Para obtener más información, vea el tema "Niveles de gravedad de mensajes de error" en los Libros en pantalla de SQL Server.

El evento InfoMessage recibe un objeto InfoMessageEventArgs que contiene en su propiedad **Errors** una colección de los mensajes del origen de datos. Puede consultar los objetos **Error** de esa colección para conocer el número de error y el texto del mensaje, así como el origen del error. El proveedor de datos de .NET Framework para SQL Server incluye asimismo datos acerca de la base de datos, el procedimiento almacenado y el número de línea donde se originó el mensaje.

Ejemplo
En el ejemplo de código siguiente se muestra cómo se puede agregar un controlador de eventos para el evento InfoMessage.

```
' Assumes that connection represents a SqlConnection object.
  AddHandler connection.InfoMessage, _
    New SqlInfoMessageEventHandler(AddressOf OnInfoMessage)
Private Shared Sub OnInfoMessage(sender As Object, _
  args As SqlInfoMessageEventArgs)
  Dim err As SqlError
  For Each err In args.Errors
    Console.WriteLine("The {0} has received a severity {1}, _
       state {2} error number {3}\n" & _
      "on line {4} of procedure {5} on server {6}:\n{7}", _
      err.Source, err.Class, err.State, err.Number, err.LineNumber, _
    err.Procedure, err.Server, err.Message)
  Next
End Sub
```

Controlar errores como InfoMessages

Normalmente, el evento InfoMessage sólo se activa para mensajes informativos y de advertencia enviados desde el servidor. Sin embargo, cuando se produce un error real, se detiene la ejecución de los métodos **ExecuteNonQuery** o **ExecuteReader** que iniciaron la operación del servidor y se inicia una excepción.

Si desea seguir procesando el resto de las instrucciones de un comando, independientemente de los errores producidos en el servidor, establezca la propiedad FireInfoMessageEventOnUserErrors de SqlConnection como **true**. De esta forma, la conexión activa el evento InfoMessage para errores, en lugar de iniciar una excepción e interrumpir el procesamiento. La aplicación cliente puede controlar el evento y reaccionar ante las situaciones de error.

Nota:

Los errores con un nivel de gravedad de 17, como mínimo, que hacen que el servidor interrumpa el procesamiento de comandos, deben controlarse como excepciones. En este caso, se inicia una excepción, independientemente del modo en que se controle el error en el evento InfoMessage.

Trabajar con el evento StateChange

El evento **StateChange** se produce cuando cambia el estado de un objeto **Connection**. El evento **StateChange** recibe StateChangeEventArgs que permiten determinar el cambio de estado de **Connection** por medio de las propiedades **OriginalState** y **CurrentState**. La propiedad **OriginalState** es una enumeración ConnectionState que indica el estado del objeto **Connection** antes del cambio. **CurrentState** es una enumeración ConnectionState que indica el estado del objeto **Connection** después del cambio.

En el ejemplo de código siguiente se utiliza el evento **StateChange** para escribir un mensaje en la consola cuando cambia el estado del objeto **Connection**.

```
' Assumes connection represents a SqlConnection object.
  AddHandler connection.StateChange, _
    New StateChangeEventHandler(AddressOf OnStateChange)
Protected Shared Sub OnStateChange( _
  sender As Object, args As StateChangeEventArgs)
  Console.WriteLine( _
  "The current Connection state has changed from {0} to {1}.", _
  args.OriginalState, args.CurrentState)
End Sub
```

6.1.3. Contadores de Rendimiento

ADO.NET 2.0 permite aprovechar mucho más los contadores de rendimiento y ofrece la oportunidad de trabajar tanto con System.Data.SqlClient como con System.Data.OracleClient. Los contadores de rendimiento System.Data.SqlClient que estaban disponibles en las versiones anteriores de ADO.NET se han descartado y se han sustituido por los nuevos contadores de rendimiento que se describen aquí. Puede utilizar los contadores de rendimiento de ADO.NET para supervisar el estado de su aplicación y los recursos de conexión que emplea. Los contadores de rendimiento se pueden controlar con el Monitor de rendimiento de Windows pero también se puede tener acceso a ellos mediante programación usando la clase PerformanceCounter del espacio de nombres System.Diagnostics.

⊟ Contadores de rendimiento disponibles

Actualmente hay disponibles 14 contadores de rendimiento para System.Data.SqlClient y System.Data.OracleClient, tal y como se describe en la siguiente tabla. Tenga en cuenta que los nombres de los contadores individuales no se localizan en las versiones regionales de Microsoft .NET Framework.

Contador de rendimiento	Descripción
HardConnectsPerSecond	El número de conexiones por segundo que se establecen con un servidor de base de datos.
HardDisconnectsPerSecond	El número de desconexiones por segundo que se producen con un servidor de base de datos.
NumberOfActiveConnectionPoolGroups	El número de conjuntos de grupos de conexiones únicas que están activos. Este contador depende del número de cadenas de conexión única que haya en el AppDomain.
NumberOfActiveConnectionPools	El número total de grupos de conexiones.
NumberOfActiveConnections	El número de conexiones activas que se están

	utilizando actualmente. **Nota:** Este contador de rendimiento no está habilitado de forma predeterminada.
NumberOfFreeConnections	El número de conexiones que se pueden utilizar en los grupos de conexiones. **Nota:** Este contador de rendimiento no está habilitado de forma predeterminada.
NumberOfInactiveConnectionPoolGroups	El número de conjuntos de grupos de conexiones únicas que están marcados para ser eliminados. Este contador depende del número de cadenas de conexión única que haya en el AppDomain.
NumberOfInactiveConnectionPools	El número de grupos de conexiones inactivas que no han tenido ninguna actividad recientemente y que están a la espera de ser eliminadas.
NumberOfNonPooledConnections	El número de conexiones activas que no están agrupadas.
NumberOfPooledConnections	El número de conexiones activas que administra la infraestructura de agrupación de conexiones.
NumberOfReclaimedConnections	El número de conexiones que se han reclamado a través de la recolección de elementos no utilizados si la aplicación no llamó a **Close** o **Dispose**. Si no se cierran o eliminan explícitamente las conexiones, el rendimiento se verá perjudicado.
NumberOfStasisConnections	El número de conexiones que están actualmente en espera de que se finalice una acción y que por lo tanto no pueden ser utilizadas por la aplicación.
SoftConnectsPerSecond	El número de conexiones activas que se están extrayendo del grupo de conexiones. **Nota:** Este contador de rendimiento no está habilitado de forma predeterminada.
SoftDisconnectsPerSecond	El número de conexiones activas que se devuelven al grupo de conexiones. **Nota:** Este contador de rendimiento no está habilitado de forma predeterminada.

Conjuntos de grupos de conexiones y grupos de conexiones

Si utiliza la autenticación de Windows (seguridad integrada) debe supervisar los contadores de rendimiento **NumberOfActiveConnectionPoolGroups** y **NumberOfActiveConnectionPools**. El

motivo es que los conjuntos de grupos de conexiones se corresponden con cadenas de conexión única. Si se usa seguridad integrada, los grupos de conexiones se asignan a cadenas de conexión y además crean grupos diferentes para cada identidad de Windows. Por ejemplo, si Alfredo y Julia, los dos dentro del mismo AppDomain, utilizan la cadena de conexión "Data Source=MySqlServer;Integrated Security=true", se crea un conjunto de grupos de conexiones para la cadena de conexión y dos grupos adicionales, uno para Alfredo y otro para Julia. Si Francisco y Marta utilizan una cadena de conexión con un inicio de sesión de SQL idéntico, "Data Source=MySqlServer;User Id=lowPrivUser;Password=Strong?Password", sólo se creará un grupo para la identidad **lowPrivUser**.

Activar contadores desactivados de forma predeterminada

Los contadores de rendimiento **NumberOfFreeConnections**, **NumberOfActiveConnections**, **SoftDisconnectsPerSecond** y **SoftConnectsPerSecond** están desactivados de forma predeterminada. Agregue la siguiente información al archivo de configuración de la aplicación para activarlos:

```
<system.diagnostics>
  <switches>
    <add name="ConnectionPoolPerformanceCounterDetail"
         value="4"/>
  </switches>
</system.diagnostics>
```

Recuperar los valores de los contadores de rendimiento

La siguiente aplicación de consola muestra cómo recuperar valores de los contadores de rendimiento en su aplicación. Las conexiones deben estar abiertas y activas para que se devuelva información de todos los contadores de rendimiento de ADO.NET.

Nota:

En este ejemplo se usa la base de datos de ejemplo **AdventureWorks** que se incluye con SQL Server 2005. Las cadenas de conexión que se incluyen en el código de ejemplo presuponen que la base de datos está instalada y disponible en el equipo local con el nombre de instancia SqlExpress y que se han creado inicios de sesión de SQL Server que coinciden con los proporcionados en las cadenas de conexión. Quizá deba habilitar inicios de sesión de SQL Server si su servidor se ha configurado usando la configuración de seguridad predeterminada, que sólo admite la autenticación de Windows. Modifique las cadenas de conexión según sea necesario para su entorno.

Ejemplo

```
Imports System.Data.SqlClient
Imports System.Diagnostics
Imports System.Runtime.InteropServices

Class Program
    Private PerfCounters(9) As PerformanceCounter
    Private connection As SqlConnection = New SqlConnection
    Public Shared Sub Main()
        Dim prog As Program = New Program
        ' Open a connection and create the performance counters.
        prog.connection.ConnectionString = _
           GetIntegratedSecurityConnectionString()
        prog.SetUpPerformanceCounters()
        Console.WriteLine("Available Performance Counters:")
        ' Create the connections and display the results.
        prog.CreateConnections()
```

```
        Console.WriteLine("Press Enter to finish.")
        Console.ReadLine()
    End Sub
    Private Sub CreateConnections()
        ' List the Performance counters.
        WritePerformanceCounters()
        ' Create 4 connections and display counter information.
        Dim connection1 As SqlConnection = New SqlConnection( _
           GetIntegratedSecurityConnectionString)
        connection1.Open()
        Console.WriteLine("Opened the 1st Connection:")
        WritePerformanceCounters()
        Dim connection2 As SqlConnection = New SqlConnection( _
           GetSqlConnectionStringDifferent)
        connection2.Open()
        Console.WriteLine("Opened the 2nd Connection:")
        WritePerformanceCounters()
        Console.WriteLine("Opened the 3rd Connection:")
        Dim connection3 As SqlConnection = New SqlConnection( _
           GetSqlConnectionString)
        connection3.Open()
        WritePerformanceCounters()
        Dim connection4 As SqlConnection = New SqlConnection( _
           GetSqlConnectionString)
        connection4.Open()
        Console.WriteLine("Opened the 4th Connection:")
        WritePerformanceCounters()
        connection1.Close()
        Console.WriteLine("Closed the 1st Connection:")
        WritePerformanceCounters()
        connection2.Close()
        Console.WriteLine("Closed the 2nd Connection:")
        WritePerformanceCounters()
        connection3.Close()
        Console.WriteLine("Closed the 3rd Connection:")
        WritePerformanceCounters()
        connection4.Close()
        Console.WriteLine("Closed the 4th Connection:")
        WritePerformanceCounters()
    End Sub
    Private Enum ADO_Net_Performance_Counters
        NumberOfActiveConnectionPools
        NumberOfReclaimedConnections
        HardConnectsPerSecond
        HardDisconnectsPerSecond
        NumberOfActiveConnectionPoolGroups
        NumberOfInactiveConnectionPoolGroups
        NumberOfInactiveConnectionPools
        NumberOfNonPooledConnections
        NumberOfPooledConnections
```

```
        NumberOfStasisConnections
        ' The following performance counters are more expensive to track.
     ' Enable ConnectionPoolPerformanceCounterDetail in your config file.
        '     SoftConnectsPerSecond
        '     SoftDisconnectsPerSecond
        '     NumberOfActiveConnections
        '     NumberOfFreeConnections
    End Enum
    Private Sub SetUpPerformanceCounters()
        connection.Close()
        Me.PerfCounters(9) = New PerformanceCounter()
        Dim instanceName As String = GetInstanceName()
        Dim apc As Type = GetType(ADO_Net_Performance_Counters)
        Dim i As Integer = 0
        Dim s As String = ""
        For Each s In [Enum].GetNames(apc)
            Me.PerfCounters(i) = New PerformanceCounter()
    Me.PerfCounters(i).CategoryName = ".NET Data Provider for SqlServer"
            Me.PerfCounters(i).CounterName = s
            Me.PerfCounters(i).InstanceName = instanceName
            i = (i + 1)
        Next
    End Sub
    Private Declare Function GetCurrentProcessId Lib "kernel32.dll" () As
Integer
    Private Function GetInstanceName() As String
        'This works for Winforms apps.
        Dim instanceName As String = _
           System.Reflection.Assembly.GetEntryAssembly.GetName.Name
        ' Must replace special characters like (, ), #, /, \\
        Dim instanceName2 As String = _
 AppDomain.CurrentDomain.FriendlyName.ToString.Replace("(", "[") _
.Replace(")", "]").Replace("#", "_").Replace("/", "_").Replace("\\", "_")
        'For ASP.NET applications your instanceName will be your
        ' CurrentDomain's FriendlyName. Replace the line above that sets
        ' the instanceName with this: instanceName =
        ' AppDomain.CurrentDomain.FriendlyName.ToString.Replace("(", "[")
        '    .Replace(")", "]").Replace("#", "_").Replace("/",
        ' "_").Replace("\\", "_")
        Dim pid As String = GetCurrentProcessId.ToString
        instanceName = (instanceName + ("[" & (pid & "]")))
        Console.WriteLine("Instance Name: {0}", instanceName)
        Console.WriteLine("---------------------------")
        Return instanceName
    End Function
    Private Sub WritePerformanceCounters()
        Console.WriteLine("---------------------------")
        For Each p As PerformanceCounter In Me.PerfCounters
            Console.WriteLine("{0} = {1}", p.CounterName, p.NextValue)
        Next
```

```
        Console.WriteLine("---------------------------")
    End Sub
    Private Shared Function GetIntegratedSecurityConnectionString() As
String
        ' To avoid storing the connection string in your code,
        ' you can retrive it from a configuration file.
        Return ("Data Source=.\SqlExpress;Integrated Security=True;" &
          "Initial Catalog=AdventureWorks")
    End Function
    Private Shared Function GetSqlConnectionString() As String
        ' To avoid storing the connection string in your code,
        ' you can retrive it from a configuration file.
   Return ("Data Source=.\SqlExpress;User Id=LowPriv;Password=Data!05;" &
          "Initial Catalog=AdventureWorks")
    End Function
    Private Shared Function GetSqlConnectionStringDifferent() As String
        ' To avoid storing the connection string in your code,
        ' you can retrive it from a configuration file.
   Return ("Initial Catalog=AdventureWorks;Data Source=.\SqlExpress;" & _
          "User Id=LowPriv;Password=Data!05;")
    End Function
End Class
```

6.2. Cadenas de Conexión

.NET Framework 2.0 proporciona nuevas capacidades para trabajar con cadenas de conexión, incluida la introducción de nuevas palabras clave en las clases generadoras de cadenas de conexión, que facilitan la creación de cadenas de conexión válidas en tiempo de ejecución.

Una cadena de conexión contiene información de inicialización que se transfiere como un parámetro desde un proveedor de datos a un origen de datos. La sintaxis depende del proveedor de datos y la cadena de conexión se analiza durante el intento para abrir una conexión. Los errores de sintaxis generan una excepción en tiempo de ejecución, pero otros errores sólo se producen después de que el origen de datos recibe la información de conexión. Una vez validada la cadena de conexión, el origen de datos aplica las opciones especificadas en ella y abre la conexión.

El formato de una cadena de conexión es una lista de pares de parámetros de clave y valor delimitados por punto y coma.

keyword1=value; keyword2=value;

Las palabras clave no distinguen entre mayúsculas y minúsculas y los espacios entre los pares de clave-valor se omiten. Sin embargo, los valores pueden distinguir entre mayúsculas y minúsculas, en función del origen de datos. Los valores que contengan un punto y coma, caracteres de comilla sencilla o caracteres de comilla doble deben colocarse entre comillas dobles.

La sintaxis válida de cadena de conexión depende del proveedor y ha evolucionado a lo largo de los años desde las primeras API como ODBC. El proveedor de datos de .NET Framework para SQL Server incorpora muchos elementos de la sintaxis antigua y, por lo general, es más flexible con la sintaxis común de cadena de conexión. Con frecuencia existen sinónimos igualmente válidos para los elementos de sintaxis de la cadena de conexión, pero algunos errores de sintaxis y ortografía pueden generar

problemas. Por ejemplo, "Integrated Security=true" es válido, en tanto que "IntegratedSecurity=true" genera un error. Además, las cadenas de conexión construidas en tiempo de ejecución a partir de entrada de usuario no validada pueden dar lugar a ataques de inyección de cadenas, lo que pone en peligro la seguridad en el origen de datos.

Para solucionar estos problemas, ADO.NET 2.0 incorpora nuevos generadores de cadenas de conexión para cada proveedor de datos de .NET Framework. Las palabras clave se exponen como propiedades, lo que permite validar la sintaxis de la cadena de conexión antes de su envío al origen de datos. También existen nuevas clases que simplifican el almacenamiento y la recuperación de cadenas de conexión en archivos de configuración y su cifrado mediante configuración protegida.

6.2.1. Generadores de Cadenas de Conexión

En versiones anteriores de ADO.NET, no se producía la comprobación de las cadenas de conexión con valores de cadena concatenados en tiempo de compilación, por lo que, en tiempo de ejecución, una palabra clave incorrecta generaba una excepción ArgumentException. Cada uno de los proveedores de datos de .NET Framework admitía una sintaxis diferente para las palabras claves de cadenas de conexión, lo que dificultaba la construcción de cadenas de conexión válidas de forma manual. Para solucionar este problema, ADO.NET 2.0 incorpora nuevos generadores de cadenas de conexión para cada proveedor de datos de .NET Framework. Cada uno de los proveedores de datos incluye una clase generadora de cadenas de conexión con establecimiento inflexible de tipos que hereda de DbConnectionStringBuilder. En la tabla siguiente se indican los proveedores de datos de .NET Framework y sus clases generadoras de cadenas de conexión asociadas.

Proveedor	Clase ConnectionStringBuilder
System.Data.SqlClient	SqlConnectionStringBuilder
System.Data.OleDb	OleDbConnectionStringBuilder
System.Data.Odbc	OdbcConnectionStringBuilder
System.Data.OracleClient	OracleConnectionStringBuilder

⊟ Evitar ataques de inyección de cadenas de conexión

Cuando se utiliza la concatenación dinámica de cadenas para generar cadenas de conexión basadas en datos introducidos por el usuario, se pueden producir ataques de inyección de cadenas de conexión. Si no se valida la cadena y no se crean secuencias de escape para los caracteres o el texto malintencionado, los atacantes pueden tener acceso a datos confidenciales y a otros recursos del servidor. Por ejemplo, un atacante puede realizar un ataque si proporciona un punto y coma y anexa un valor adicional. La cadena de conexión se analiza utilizando un algoritmo del tipo "el último gana", y la entrada hostil se sustituye por un valor legítimo.

Las clases generadoras de cadenas de conexión se han diseñado para eliminar la adivinación y desarrollar protección ante errores de sintaxis y vulnerabilidades de seguridad. Proporcionan métodos y propiedades que corresponden a los pares clave/valor conocidos permitidos por cada proveedor de datos. Cada clase mantiene una colección fija de sinónimos y puede convertir un sinónimo al correspondiente nombre de clave conocido. En los pares clave/valor se realizan comprobaciones y los pares no válidos inician una excepción. Además, los valores inyectados se controlan de forma segura.

En el ejemplo siguiente se muestra cómo SqlConnectionStringBuilder controla un valor adicional insertado en la configuración **Initial Catalog**.

```
Dim builder As New System.Data.SqlClient.SqlConnectionStringBuilder
builder("Data Source") = "(local)"
builder("Integrated Security") = True
builder("Initial Catalog") = "AdventureWorks;NewValue=Bad"
Console.WriteLine(builder.ConnectionString)
```

El resultado muestra que SqlConnectionStringBuilder controla esta situación correctamente, ya que establece el escape del valor adicional entre comillas dobles en lugar de anexarlo a la cadena de conexión como un nuevo par clave/valor.

```
data source=(local);Integrated Security=True;
initial catalog="AdventureWorks;NewValue=Bad"
```

⊟ Generar cadenas de conexión a partir de archivos de configuración

Si determinados elementos de una cadena de conexión se conocen de antemano, se pueden almacenar en un archivo de configuración y recuperar en tiempo de ejecución para construir una cadena de conexión completa. Por ejemplo, se puede conocer por adelantado el nombre de la base de datos, pero no el del servidor. También es posible que desee que un usuario indique un nombre y una contraseña en tiempo de ejecución sin que pueda inyectar otros valores en ella.

Uno de los constructores sobrecargados de un generador de cadenas de conexión toma String como argumento, lo que permite proporcionar una cadena de conexión parcial que se puede completar después con los datos introducidos por el usuario. La cadena de conexión parcial se puede almacenar en un archivo de configuración y recuperarse en tiempo de ejecución.

Nota:

El espacio de nombres System.Configuration permite el acceso mediante programación a archivos de configuración que usan WebConfigurationManager en aplicaciones web y ConfigurationManager en aplicaciones para Windows.

Ejemplo

En este ejemplo se muestra la recuperación de una cadena de conexión incluida en un archivo de configuración y cómo se completa mediante el establecimiento de las propiedades DataSource, UserID y Password de SqlConnectionStringBuilder. El archivo de configuración se define de la siguiente forma.

```
<connectionStrings>
  <clear/>
  <add name="partialConnectString"
    connectionString="Initial Catalog=Northwind;"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

Nota:

Para ejecutar el código, debe establecer una referencia al archivo **System.Configuration.dll** del proyecto.

```
Private Sub BuildConnectionString(ByVal dataSource As String, _
    ByVal userName As String, ByVal userPassword As String)
    ' Retrieve the partial connection string named databaseConnection
    ' from the application's app.config or web.config file.
```

```
    Dim settings As ConnectionStringSettings = _
       ConfigurationManager.ConnectionStrings("partialConnectString")
    If Not settings Is Nothing Then
        ' Retrieve the partial connection string.
        Dim connectString As String = settings.ConnectionString
        Console.WriteLine("Original: {0}", connectString)
        ' Create a new SqlConnectionStringBuilder based on the
        ' partial connection string retrieved from the config file.
        Dim builder As New SqlConnectionStringBuilder(connectString)
        ' Supply the additional values.
        builder.DataSource = dataSource
        builder.UserID = userName
        builder.Password = userPassword
        Console.WriteLine("Modified: {0}", builder.ConnectionString)
    End If
End Sub
```

6.2.2. Cadenas de Conexión y Archivos de Configuración

La incrustación de cadenas de conexión en el código de la aplicación puede producir vulnerabilidades en la seguridad y problemas de mantenimiento. Las cadenas de conexión sin cifrar compiladas en el código fuente de una aplicación se pueden ver con la herramienta Desensamblador de MSIL (Ildasm.exe). Además, si la cadena de conexión cambia en algún momento, será necesario compilar de nuevo la aplicación. Por estas razones, se recomienda almacenar las cadenas de conexión en un archivo de configuración de la aplicación.

⊟ Trabajar con archivos de configuración de la aplicación

Los archivos de configuración de la aplicación contienen valores específicos de una aplicación determinada. Por ejemplo, una aplicación de ASP.NET puede tener uno o varios archivos **web.config** y una aplicación de Windows puede tener un archivo **app.config** opcional. Los archivos de configuración comparten elementos comunes, aunque su nombre y ubicación varían en función del host de la aplicación.

Sección connectionStrings

Las cadenas de conexión se pueden almacenar como pares de clave y valor en la sección **connectionStrings** del elemento **configuration** en el archivo de configuración de una aplicación. Los elementos secundarios incluyen **add**, **clear** y **remove**.

El siguiente fragmento del archivo de configuración muestra el esquema y la sintaxis para almacenar una cadena de conexión. El atributo **name** es un nombre que se proporciona para identificar de forma única una cadena de conexión, de forma que se pueda recuperar en tiempo de ejecución. **providerName** es el nombre invariable del proveedor de datos de .NET Framework registrado en el archivo machine.config.

```
<?xml version='1.0' encoding='utf-8'?>
  <configuration>
    <connectionStrings>
      <clear />
      <add name="Name"
       providerName="System.Data.ProviderName"
       connectionString="Valid Connection String;" />
```

```
    </connectionStrings>
  </configuration>
```

Nota:

Puede guardar parte de la cadena de conexión en un archivo de configuración y usar la clase DbConnectionStringBuilder para completarla en tiempo de ejecución. Esto resulta útil en escenarios en los que no se conocen los elementos de la cadena de conexión por anticipado o cuando no desea guardar información confidencial en un archivo de configuración.

Uso de archivos de configuración externos

Los archivos de configuración externos son archivos independientes que contienen un fragmento de un archivo de configuración compuesto de una sola sección. El archivo de configuración principal hace referencia al archivo de configuración externo. El almacenamiento de la sección **connectionStrings** en un archivo físicamente independiente resulta útil en situaciones en las que es posible que se edite la cadena de conexión después de implementar la aplicación. Por ejemplo, si se modifican los archivos de configuración, ASP.NET reinicia de forma predeterminada el dominio de la aplicación, lo que provoca la pérdida de la información de estado. Sin embargo, la modificación de un archivo de configuración externo no provoca el reinicio de la aplicación. Los archivos de configuración externos no se limitan a ASP.NET; también se pueden utilizar en aplicaciones de Windows. Además, la seguridad y permisos de acceso a los archivos se pueden usar para restringir el acceso a los archivos de configuración externos. El trabajo con archivos de configuración externos en tiempo de ejecución es transparente y no requiere código especial.

Para almacenar cadenas de conexión en un archivo de configuración externo, cree un archivo independiente que contenga únicamente la sección **connectionStrings**. No incluya elementos, secciones ni atributos adicionales. En este ejemplo se muestra la sintaxis de un archivo de configuración externo.

```
  <connectionStrings>
    <add name="Name"
     providerName="System.Data.ProviderName"
     connectionString="Valid Connection String;" />
  </connectionStrings>
```

En el archivo de configuración principal de la aplicación, use el atributo **configSource** para especificar el nombre completo y la ubicación del archivo externo. En este ejemplo se hace referencia a un archivo de configuración externo denominado connections.config.

```
<?xml version='1.0' encoding='utf-8'?>
<configuration>
    <connectionStrings configSource="connections.config"/>
</configuration>
```

Recuperar cadenas de conexión en tiempo de ejecución

Las nuevas clases del espacio de nombres System.Configuration de .NET Framework 2.0 simplifican la recuperación de cadenas de conexión de archivos de configuración en tiempo de ejecución. La cadena de conexión se puede recuperar mediante programación con el nombre de la cadena o el nombre de proveedor.

Nota:

El archivo **machine.config** también contiene una sección **connectionStrings**, donde se encuentran las cadenas de conexión que usa Visual Studio. Al recuperar cadenas de conexión mediante el nombre del proveedor del archivo **app.config** en una aplicación de Windows, primero se cargan las

cadenas de conexión del archivo **machine.config** y, a continuación, las entradas de **app.config**. Si se agrega **clear** inmediatamente después del elemento **connectionStrings**, se quitarán todas las referencias heredadas de la estructura de datos en memoria, de forma que sólo se tendrán en cuenta las cadenas de conexión definidas en el archivo **app.config** local.

Trabajar con clases de configuración

<codeEntityReference autoUpgrade="true">T:System.Configuration.ConfigurationManager</codeEntityReference> se usa en .NET Framework 2.0 cuando trabaja con archivos de configuración externos en el equipo local, en lugar del objeto obsoleto <codeEntityReference autoUpgrade="true">T:System.Configuration.ConfigurationSettings</codeEntityReference>. <codeEntityReference autoUpgrade="true">T:System.Web.Configuration.WebConfigurationManager</codeEntityReference> se usa para trabajar con archivos de configuración de ASP.NET. Esta característica se ha diseñado para trabajar con archivos de configuración en un servidor web y permite el acceso mediante programación a secciones del archivo de configuración como **system.web**.

Nota:

El acceso a los archivos de configuración en tiempo de ejecución requiere la concesión de permisos al llamador; los permisos necesarios dependen del tipo de aplicación, del archivo de configuración y de la ubicación.

Puede usar ConnectionStringSettingsCollection para recuperar cadenas de conexión de archivos de configuración de aplicación. Esta clase contiene una colección de objetos ConnectionStringSettings, cada uno de los cuales representa una única entrada en la sección **connectionStrings**. Sus propiedades se asignan a los atributos de cadenas de conexión, lo que permite recuperar una cadena de conexión mediante la especificación de su nombre o del nombre del proveedor.

Propiedad	Descripción
Name	Nombre de la cadena de conexión. Se asigna al atributo **name**.
ProviderName	Nombre completo del proveedor. Se asigna al atributo **providerName**.
ConnectionString	Cadena de conexión. Se asigna al atributo **connectionString**.

Ejemplo: mostrar todas las cadenas de conexión

Este ejemplo recorre en iteración la colección **ConnectionStringSettings** y muestra las propiedades Name, ProviderName y ConnectionString en la ventana de la consola.

Nota:

System.Configuration.dll no se incluye en todos los tipos de proyectos y es posible que deba establecer una referencia a este elemento para usar las clases de configuración. El nombre y la ubicación de un archivo de configuración de aplicación determinado varían en función del tipo de aplicación y del proceso de hospedaje.

```
Imports System.Configuration

Class Program
    Shared Sub Main()
        GetConnectionStrings()
        Console.ReadLine()
```

```
    End Sub
    Private Shared Sub GetConnectionStrings()
        Dim settings As ConnectionStringSettingsCollection = _
            ConfigurationManager.ConnectionStrings
        If Not settings Is Nothing Then
            For Each cs As ConnectionStringSettings In settings
                Console.WriteLine(cs.Name)
                Console.WriteLine(cs.ProviderName)
                Console.WriteLine(cs.ConnectionString)
            Next
        End If
    End Sub
End Class
```

Ejemplo: recuperar una cadena de conexión por su nombre

El siguiente ejemplo muestra cómo recuperar una cadena de conexión de un archivo de configuración mediante la especificación del nombre. El código crea un objeto ConnectionStringSettings, de forma que el parámetro de entrada proporcionado coincida con el nombre de ConnectionStrings. Si no se encuentra una coincidencia de nombre, la función devuelve **null** (**Nothing** en Visual Basic).

```
' Retrieves a connection string by name.
' Returns Nothing if the name is not found.
Private Shared Function GetConnectionStringByName( _
    ByVal name As String) As String
    ' Assume failure
    Dim returnValue As String = Nothing
    ' Look for the name in the connectionStrings section.
    Dim settings As ConnectionStringSettings = _
       ConfigurationManager.ConnectionStrings(name)
    ' If found, return the connection string.
    If Not settings Is Nothing Then
        returnValue = settings.ConnectionString
    End If
    Return returnValue
End Function
```

Ejemplo: recuperar una cadena de conexión por el nombre de proveedor

En este ejemplo se muestra cómo recuperar una cadena de conexión mediante la especificación del nombre invariable de proveedor con el formato System.Data.ProviderName. El código recorre en iteración ConnectionStringSettingsCollection y devuelve la cadena de conexión del primer valor de ProviderName encontrado. Si no se encuentra el nombre del proveedor, la función devuelve **null** (**Nothing** en Visual Basic).

```
' Retrieve a connection string by specifying the providerName.
' Assumes one connection string per provider in the config file.
Private Shared Function GetConnectionStringByProvider( _
    ByVal providerName As String) As String
    'Return Nothing on failure.
    Dim returnValue As String = Nothing
    ' Get the collection of connection strings.
    Dim settings As ConnectionStringSettingsCollection = _
        ConfigurationManager.ConnectionStrings
```

```
    ' Walk through the collection and return the first
    ' connection string matching the providerName.
    If Not settings Is Nothing Then
        For Each cs As ConnectionStringSettings In settings
            If cs.ProviderName = providerName Then
                returnValue = cs.ConnectionString
                Exit For
            End If
        Next
    End If
    Return returnValue
End Function
```

6.2.3. Sintaxis de la Cadena de Conexión

Cada proveedor de datos de .NET Framework tiene un objeto **Connection** que hereda de DbConnection, además de una propiedad ConnectionString específica del proveedor. La sintaxis de la cadena de conexión específica de cada proveedor se indica en su propiedad **ConnectionString**. En la tabla siguiente se muestran los cuatro proveedores de datos que se incluyen en .NET Framework.

Proveedor de datos de .NET Framework	Descripción
System.Data.SqlClient	Proporciona acceso de datos para Microsoft SQL Server versión 7.0 o posterior.
System.Data.OleDb	Proporciona acceso de datos para orígenes de datos que se exponen mediante OLE DB.
System.Data.Odbc	Proporciona acceso de datos para orígenes de datos que se exponen mediante ODBC.
System.Data.OracleClient	Proporciona acceso de datos para Oracle versión 8.1.7 o superior.

ADO.NET 2.0 incorpora nuevos generadores de cadenas de conexión para cada proveedor de datos de .NET Framework, lo que evita tener que estar adivinando a la hora de crear cadenas de conexión sintácticamente válidas.

⊟ Especificar la autenticación de Windows

Recomendamos usar la autenticación de Windows (en ocasiones denominada seguridad integrada) para conectarse a orígenes de datos que sean compatibles. La sintaxis utilizada en la cadena de conexión varía dependiendo del proveedor. En la siguiente tabla se muestra la sintaxis de autenticación de Windows utilizada con los proveedores de datos de .NET Framework.

Proveedor	Sintaxis
SqlClient	Integrated Security=true; -- or -- Integrated Security=SSPI;
OleDb	Integrated Security=SSPI;

Odbc	Trusted_Connection=yes;
OracleClient	Integrated Security=yes;

Nota:

Integrated Security=true inicia una excepción cuando se utiliza con el proveedor **OleDb**.

Trabajar con cadenas de conexión SqlClient

La propiedad ConnectionString de una SqlConnection permite obtener o establecer una cadena de conexión para una base de datos SQL Server 7.0 o posterior. Si necesita conectarse a una versión anterior de SQL Server, deberá utilizar el proveedor de datos de .NET para OleDb. La mayoría de las palabras clave de cadenas de conexión se corresponden también con las propiedades de SqlConnectionStringBuilder.

En cada una de las siguientes formas de sintaxis se utilizará autenticación de Windows para conectarse a la base de datos **AdventureWorks** en un servidor local.

```
"Persist Security Info=False;Integrated Security=true;
    Initial Catalog=AdventureWorks;Server=MSSQL1"
"Persist Security Info=False;Integrated Security=SSPI;
    database=AdventureWorks;server=(local)"
"Persist Security Info=False;Trusted_Connection=True;
    database=AdventureWorks;server=(local)"
```

Usar inicios de sesión de SQL Server

Es preferible utilizar la autenticación de Windows para conectarse a SQL Server. No obstante, si se requiere autenticación de SQL Server, deberá utilizar la siguiente sintaxis para especificar un nombre de usuario y una contraseña. En este ejemplo, los asteriscos representan un nombre de usuario y una contraseña válidos.

```
"Persist Security Info=False;User ID=*****;Password=*****;Initial
Catalog=AdventureWorks;Server=MySqlServer"
```

Nota de seguridad:

La configuración predeterminada de la palabra clave **PersistSecurity Info** es **false**. Si se establece en **true** o **yes**, permite obtener información de seguridad confidencial de la conexión, incluidos el identificador de usuario y la contraseña, una vez abierta la conexión. Mantenga el valor **PersistSecurity Info** establecido en **false** para garantizar que los orígenes que no sean de confianza no puedan tener acceso a información confidencial de la cadena de conexión.

Para conectarse a una instancia con nombre de SQL Server 2000 o posterior, utilice la sintaxis nombre de servidor\nombre de instancia.

```
Data Source=MySqlServer\MSSQL1;"
```

A la hora de crear una cadena de conexión, también puede establecer la propiedad DataSource del **SqlConnectionStringBuilder** en el nombre de instancia. La propiedad DataSource de un objeto SqlConnection es de sólo lectura.

Configurar la biblioteca de red

La biblioteca de red se puede utilizar para establecer una conexión con una instancia de SQL Server. En este ejemplo, la biblioteca de red es Win32 Winsock TCP/IP (dbmssocn), y 1433 es el puerto que se está utilizando.

```
Network Library=dbmssocn;Data Source=000.000.000.000,1433;
```

SQL Server permite utilizar las siguientes bibliotecas de red al establecer una conexión.

Nombre	Biblioteca
dbnmpntw	Canalizaciones con nombre Win32
dbmssocn	Win32 Winsock TCP/IP
dbmsspxn	Win32 SPX/IPX
dbmsvinn	Win32 Banyan Vines
dbmsrpcn	Multiprotocolo Win32 (Windows RPC)

Especificar una versión de SQL mediante palabras clave Type System Version

Las palabras **Type System Version** permiten especificar una versión anterior de SQL para las aplicaciones que se ejecutan en una instancia de SQL Server 2005. Esto evita posibles problemas con tipos incompatibles que podrían provocar que fallase la aplicación. Por ejemplo, debe utilizar el siguiente fragmento de cadena de conexión para conectarse a una instancia de SQL Server 2005 si quiere que la aplicación sólo funcione con el sistema de tipo SQL Server 2000.

```
"Type System Version=SQL Server 2000;"
```

La versión del sistema de tipos se puede especificar usando la propiedad TypeSystemVersion de un SqlConnectionStringBuilder.

En la siguiente tabla se indican los valores posibles.

Valor	Descripción
Latest	Es la opción predeterminada. Usa la versión más reciente que pueda controlar este par cliente/servidor. La versión utilizada avanzará automáticamente a medida que se actualicen los componentes del cliente y del servidor.
SQL Server 2008	Utiliza el sistema de tipo SQL Server 2008.
SQL Server 2005	Utiliza el sistema de tipo SQL Server 2005. No se realiza ninguna conversión para la versión actual de ADO.NET.
SQL Server 2000	Utiliza el sistema de tipo SQL Server 2000. Al conectar a una instancia de SQL Server 2005, se realizarán las comparaciones siguientes: XML to NTEXT UDT to VARBINARY VARCHAR (MAX), NVARCHAR (MAX) y VARBINARY (MAX) a TEXT, NEXT e IMAGE, respectivamente.

⊟ Adjuntar a instancias de usuario de SQL Server Express

Las instancias de usuario son una nueva característica que sólo está disponible en SQL Server 2005 Express Edition. Permiten que un usuario que ejecuta una cuenta de Windows local y con privilegios mínimos asocie y ejecute una base de datos de SQL Server sin necesidad de tener privilegios administrativos. Una instancia de usuario se ejecuta con las credenciales de Windows del usuario, no como un servicio.

Las instancias de usuario también funcionan con las propiedades **SqlConnectionStringBuilder** UserInstance y AttachDBFilename.

La palabra clave de cadena de conexión **AttachDbFileName** se utiliza para adjuntar el archivo de base de datos primario (.mdf), que debe incluir el nombre de ruta de acceso completo. **AttachDbFileName** se corresponde también con las claves "extended properties" e "initial file name" dentro de una cadena de conexión SqlConnection. En este ejemplo se utiliza una ruta de acceso absoluta a la ubicación del archivo de datos primario.

```
"AttachDbFileName=c:\data\Northwind.mdf;Integrated Security=true;Initial
Catalog=Northwind;"
```

Utilizar la cadena de sustitución DataDirectory

AttachDbFileName se ha ampliado en ADO.NET 2.0 con la incorporación de la cadena de sustitución **|DataDirectory|** (entre barras verticales). **DataDirectory** se utiliza junto con **AttachDbFileName** para indicar una ruta de acceso relativa a un archivo de datos, lo que permite a los desarrolladores crear cadenas de conexión basadas en una ruta relativa al origen de datos en lugar de tener que especificar la ruta completa.

La ubicación física que indica **DataDirectory** depende del tipo de aplicación. En este ejemplo, el archivo Northwind.mdf que se va a adjuntar se encuentra en la carpeta \app_data de la aplicación.

```
Data Source=.\\SQLExpress;Integrated Security=true;
User Instance=true;
AttachDBFilename=|DataDirectory|\app_data\Northwind.mdf;
Initial Catalog=Northwind;
```

Si se usa **DataDirectory**, la ruta del archivo resultante no puede estar más alta en la estructura de directorios que el directorio que señala la cadena de sustitución. Por ejemplo, si el **DataDirectory** totalmente expandido es C:\AppDirectory\app_data, la cadena de conexión de ejemplo que se muestra arriba funcionará porque está por debajo de c:\AppDirectory. Sin embargo, si se intenta especificar **DataDirectory** como |DataDirectory|\..\data, se producirá un error porque \data no es un subdirectorio de \AppDirectory.

Si la cadena de conexión tiene una cadena de sustitución con formato incorrecto, se producirá una ArgumentException.

Nota:

System.Data.SqlClient resuelve las cadenas de sustitución en rutas de acceso completas del sistema de archivos del equipo local. Por eso, no se admiten rutas de acceso a servidores remotos, HTTP ni UNC. Si el servidor no se encuentra en el equipo local, se produce una excepción.

⊟ Usar TrustServerCertificate

La palabra clave **TrustServerCertificate** es nueva en ADO.NET 2.0 y sólo es válida cuando se conecta a una instancia de SQL Server 2005 con un certificado válido. Cuando **TrustServerCertificate** se establece en **true**, la capa de transporte utilizará SSL para cifrar el canal y evitar recorrer la cadena de certificados para validar la confianza.

```
"TrustServerCertificate=true;"
```

Nota:

Si **TrustServerCertificate** se establece en **true** y se activa el cifrado, se utilizará el nivel de cifrado

especificado en el servidor aunque **Encrypt** esté establecido en **false** en la cadena de conexión. De lo contrario, se producirá un error en la conexión.

Habilitar el cifrado

Si desea habilitar el cifrado porque no se ha incluido un certificado en el servidor, deben estar activadas las opciones **Forzar cifrado de protocolo** y **Confiar en certificado de servidor** en el Administrador de configuración de SQL Server. En este caso, el cifrado utilizará un certificado del servidor autofirmado sin validación si no se ha incluido en el servidor ningún certificado que se pueda comprobar.

La configuración de las aplicaciones no puede reducir el nivel de seguridad configurado en SQL Server, sino que en todo caso puede reforzarlo. Una aplicación puede solicitar cifrado estableciendo las palabras clave **TrustServerCertificate** y **Encrypt** en **true**, con lo que se garantiza el cifrado aunque no se haya proporcionado un certificado de servidor y no se haya configurado **Forzar cifrado de protocolo** para el cliente. Sin embargo, si en la configuración del cliente no está habilitado **TrustServerCertificate**, seguirá siendo necesario un certificado de servidor.

En la siguiente tabla se describen todos los casos.

Configuración de cliente Forzar cifrado de protocolo	Configuración de cliente Confiar en certificado de servidor	Cadena de conexión/atributo Cifrar/Utilizar cifrado para los datos	Cadena de conexión/atributo Confiar en certificado de servidor	Resultado
No	N/A	No (opción predeterminada)	Se omite.	No se produce cifrado.
No	N/A	Sí	No (opción predeterminada)	Sólo se produce cifrado si hay un certificado de servidor que se puede comprobar; en caso contrario, el intento de conexión falla.
No	N/A	Sí	Sí	Sólo se produce cifrado si hay un certificado de servidor que se puede comprobar; en caso contrario, el intento de conexión falla.
Sí	No	Se omite.	Se omite.	Sólo se produce cifrado si hay un certificado de servidor que se puede comprobar; en caso contrario, el intento de conexión falla.

Sí	Sí	No (opción predeterminada)	Se omite.	Sólo se produce cifrado si hay un certificado de servidor que se puede comprobar; en caso contrario, el intento de conexión falla.
Sí	Sí	Sí	No (opción predeterminada)	Sólo se produce cifrado si hay un certificado de servidor que se puede comprobar; en caso contrario, el intento de conexión falla.
Sí	Sí	Sí	Sí	Sólo se produce cifrado si hay un certificado de servidor que se puede comprobar; en caso contrario, el intento de conexión falla.

⊟ Cadenas de conexión OleDb

La propiedad ConnectionString de una OleDbConnection permite obtener o establecer una cadena de conexión para un origen de datos OLE DB, como Microsoft Access o SQL Server 6.5 o anterior. Utilice una SqlConnection para SQL Server 7.0 o posterior. Las cadenas de conexión de **OleDb** son también compatibles con la clase OleDbConnectionStringBuilder.

Sintaxis de cadena de conexión OleDb

Para una cadena de conexión OleDbConnection, debe proporcionar un nombre de proveedor. La siguiente cadena de conexión conecta a una base de datos Microsoft Access mediante el proveedor Jet. Tenga en cuenta que las palabras clave **UserID** y **Password** son opcionales si la base de datos no está protegida (opción predeterminada).

```
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=d:\Northwind.mdb;User
ID=Admin;Password=;
```

Si la base de datos Jet está protegida, deberá proporcionar la ubicación del archivo de información del grupo de trabajo.

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=d:\Northwind.mdb;Jet
OLEDB:System Database=d:\NorthwindSystem.mdw;User
ID=*****;Password=*****;
```

En SQL Server 6.5 o anterior, utilice la palabra clave **sqloledb**.

```
Provider=sqloledb;Data Source=MySqlServer;Initial Catalog=pubs;User
Id=*****;Password=*****;
```

Nota de seguridad:

Se puede proporcionar información de conexión para un objeto **OleDbConnection** en un archivo UDL (Universal Data Link), pero conviene evitarlo. Los archivos UDL no están cifrados y exponen la información de cadena de conexión en texto sin cifrar. Un archivo UDL no se puede proteger mediante .NET Framework, ya que se trata de un recurso basado en un archivo externo a la aplicación.

Utilizar DataDirectory para conectarse a Access/Jet

DataDirectory no es exclusivo de **SqlClient**. También se puede utilizar con los proveedores de datos de .NET System.Data.OleDb y System.Data.Odbc. La siguiente cadena OleDbConnection de ejemplo muestra la sintaxis necesaria para conectarse a la base de datos Northwind.mdb situada en la carpeta app_data de la aplicación. La base de datos del sistema (System.mdw) también está almacenada en esa ubicación.

```
"Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=|DataDirectory|\Northwind.mdb;
Jet OLEDB:System Database=|DataDirectory|\System.mdw;"
```

Nota de seguridad:

No es necesario especificar la ubicación de la base de datos del sistema en la cadena de conexión si la base de datos Access/Jet no está protegida. La seguridad está desactivada de forma predeterminada y todos los usuarios se conectan como el usuario Admin integrado, con una contraseña en blanco. Aun cuando la seguridad a nivel de usuario esté correctamente implementada, una base de datos Jet sigue siendo vulnerable a los ataques. Por eso no se recomienda almacenar información delicada en una base de datos Access/Jet a causa de la fragilidad inherente a su esquema de seguridad basado en archivos.

Conectarse a Excel

Para conectarse a un libro de Excel se utiliza el proveedor Microsoft Jet. En la siguiente cadena de conexión, la palabra clave **Extended Properties** establece propiedades que son específicas de Excel. "HDR=Yes;" indica que la primera fila contiene nombres de columna, no datos, e "IMEX=1;" indica al controlador que siempre lea las columnas de datos "entremezclados" como texto.

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\MyExcel.xls;Extended
Properties=""Excel 8.0;HDR=Yes;IMEX=1""
```

Tenga en cuenta que el carácter de comilla doble necesario en **Extended Properties** también debe ir incluido entre comillas dobles.

Sintaxis de cadena de conexión del proveedor de formas de datos

Utilice las palabras clave **Provider** y **Data Provider** cuando use el proveedor de formas de datos de Microsoft. En el siguiente ejemplo se utiliza el proveedor de formas para conectarse a una instancia local de SQL Server.

```
"Provider=MSDataShape;Data Provider=SQLOLEDB;Data Source=(local);Initial
Catalog=pubs;Integrated Security=SSPI;"
```

⊟ Cadenas de conexión Odbc

La propiedad ConnectionString de una OdbcConnection permite obtener o establecer una cadena de conexión para un origen de datos OLE DB. Las cadenas de conexión Odbc también son compatibles con OdbcConnectionStringBuilder.

En la siguiente cadena de conexión se utiliza el Controlador de texto de Microsoft.

```
Driver={Microsoft Text Driver (*.txt; *.csv)};DBQ=d:\bin
```

Utilizar DataDirectory para conectarse a Visual FoxPro

El siguiente ejemplo de cadena de conexión OdbcConnection muestra cómo se utiliza **DataDirectory** para conectarse a un archivo de Microsoft Visual FoxPro.

```
"Driver={Microsoft Visual FoxPro Driver};
SourceDB=|DataDirectory|\MyData.DBC;SourceType=DBC;"
```

⊟ Cadenas de conexión Oracle

La propiedad ConnectionString de una OracleConnection permite obtener o establecer una cadena de conexión para un origen de datos OLE DB. Las cadenas de conexión Oracle también son compatibles con OracleConnectionStringBuilder.

```
Data Source=Oracle9i;User ID=*****;Password=*****;
```

6.2.4. Proteger la Información de Conexión

La protección del acceso al origen de datos es uno de los objetivos más importantes a la hora de proteger una aplicación. Las cadenas de conexión presentan una posible vulnerabilidad si no se protegen. El almacenamiento de la información de conexión en texto sin formato o en la memoria ponen en riesgo el sistema completo. Las cadenas de conexión incrustadas en el código fuente se pueden leer con Desensamblador de MSIL (Ildasm.exe) para ver Lenguaje intermedio de Microsoft (MSIL) en un ensamblado compilado.

⊟ Evitar vulnerabilidades de seguridad

Pueden surgir vulnerabilidades de seguridad que afecten a las cadenas de conexión en función del tipo de autenticación usado, de la forma en que las cadenas de conexión se almacenan en memoria y en disco, y de las técnicas usadas para construirlas en tiempo de ejecución.

Uso de autenticación de Windows

Para contribuir a limitar el acceso al origen de datos, debe proteger la información de la conexión como, por ejemplo, el id. de usuario, la contraseña y el nombre de origen de datos. Para evitar la exposición de información de usuario, se recomienda usar la autenticación de Windows (suele aparecer también como seguridad integrada) siempre que sea posible. La autenticación de Windows se especifica en una cadena de conexión mediante las palabras clave Integrated Security o Trusted_Connection, lo que elimina la necesidad de usar un identificador de usuario y una contraseña. Cuando se usa autenticación de Windows, este sistema operativo autentica a los usuarios y el acceso a los recursos del servidor y de la base de datos se determina mediante la concesión de permisos a usuarios y grupos de Windows.

Cuando no sea posible usar la autenticación de Windows, es necesario extremar las precauciones, ya que las credenciales de usuario están expuestas en la cadena de conexión. En una aplicación ASP.NET, puede configurar una cuenta de Windows como una identidad fija que se usa para conectarse a las bases de datos y a otros recursos de red. Para ello, habilite la suplantación en el elemento de identidad del archivo **web.config** y especifique un nombre de usuario y una contraseña.

```
<identity impersonate="true"
        userName="MyDomain\UserAccount"
        password="*****" />
```

La cuenta de identidad fija debe ser una cuenta con pocos privilegios a la que sólo se le concedan los permisos necesarios en la base de datos. Además, debe cifrar el archivo de configuración para que el nombre de usuario y la contraseña no se expongan en texto no cifrado.

No usar archivos UDL

Evite almacenar cadenas de conexión de OleDbConnection en un archivo de vínculo de datos universal (UDL). Los archivos UDL se almacenan en texto no cifrado y no se pueden cifrar. Un archivo UDL no se puede proteger ni cifrar mediante .NET Framework, ya que se trata de un recurso basado en un archivo externo a la aplicación.

Prevención de ataques de inyección de cadenas de conexión

Se pueden producir ataques de inyección de cadenas de conexión cuando se usa la concatenación dinámica de cadenas para generar cadenas de conexión basadas en la entrada del usuario. Si no se valida la entrada del usuario y no se crean secuencias de escape para los caracteres o el texto malintencionado, los atacantes pueden tener acceso a datos confidenciales y a otros recursos del servidor. Para solucionar este problema, ADO.NET 2.0 incluye nuevas clases de generación de cadenas de conexión para validar la sintaxis de las cadenas y garantizar que no se insertan parámetros adicionales.

Uso del valor predeterminado en Persist Security Info

El valor predeterminado de **Persist Security Info** es false, y se recomienda mantener este valor predeterminado en todas las cadenas de conexión. Si **Persist Security Info** se establece en **true** o **yes**, permitirá obtener información confidencial de seguridad de la conexión, incluidos el identificador del usuario y la contraseña, una vez que esté abierta. Si **Persist Security Info** se establece en **false** o **no**, la información de seguridad se elimina tras usarla para abrir la conexión. Esto permite garantizar que los orígenes que no sean de confianza no tengan acceso a la información confidencial de seguridad.

⊟ Almacenar cadenas de conexión en archivos de configuración

Las cadenas de conexión también se pueden almacenar en archivos de configuración, lo que elimina la necesidad de incrustarlas en el código de la aplicación. Los archivos de configuración son archivos XML estándar para los que .NET Framework ha definido un conjunto común de elementos. Generalmente, las cadenas de conexión de los archivos de configuración se almacenan en el elemento **<connectionStrings>** del archivo **app.config** si se trata de una aplicación de Windows o del archivo **web.config** si se trata de una aplicación ASP.NET.

⊟ Cifrar secciones del archivo de configuración mediante una configuración protegida

ASP.NET 2.0 incluye una nueva característica denominada configuración protegida, que permite cifrar información confidencial en un archivo de configuración. Si bien se ha diseñado principalmente para ASP.NET, la configuración protegida también se puede usar para cifrar secciones del archivo de configuración en aplicaciones de Windows.

El siguiente fragmento de un archivo de configuración muestra la sección **connectionStrings** después de su cifrado. La sección **configProtectionProvider** especifica el proveedor de configuración protegida que se usa para cifrar y descifrar las cadenas de conexión. La sección **EncryptedData** contiene el texto cifrado.

```
<connectionStrings
configProtectionProvider="DataProtectionConfigurationProvider">
  <EncryptedData>
    <CipherData>
      <CipherValue>AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAAH2... </CipherValue>
    </CipherData>
  </EncryptedData>
</connectionStrings>
```

Cuando se recupera la cadena de conexión cifrada en tiempo de ejecución, .NET Framework usa el proveedor especificado para descifrar el nodo **CipherValue** y que así esté disponible para la aplicación. No es necesario escribir ningún código adicional para administrar el proceso de descifrado.

Proveedores de configuración protegida

Los proveedores de configuración protegida se registran en la sección **configProtectedData** del archivo **machine.config** en el equipo local, tal como se muestra en el siguiente fragmento, donde se pueden ver los dos proveedores de configuración protegida que proporciona .NET Framework. Los valores que se muestran se han truncado para facilitar la lectura.

```
<configProtectedData defaultProvider="RsaProtectedConfigurationProvider">
  <providers>
    <add name="RsaProtectedConfigurationProvider"
   type="System.Configuration.RsaProtectedConfigurationProvider, ... />
    <add name="DataProtectionConfigurationProvider"
   type="System.Configuration.DpapiProtectedConfigurationProvider, ... />
  </providers>
</configProtectedData>
```

Puede configurar otros proveedores de configuración protegida si los agrega al archivo **machine.config**. Asimismo, puede crear su propio proveedor de configuración protegida heredando de la clase base abstracta ProtectedConfigurationProvider. En la tabla siguiente se describen los dos archivos de configuración incluidos en .NET Framework.

Proveedor	Descripción
RSAProtectedConfigurationProvider	Usa el algoritmo de cifrado RSA para cifrar y descifrar datos. Los algoritmos RSA se pueden usar para el cifrado de clave pública y para firmas digitales. También se conoce como cifrado de "clave pública" o asimétrico, ya que usa dos claves diferentes. Puede usar la herramienta Herramienta Registro de IIS en ASP.NET (Aspnet_regiis.exe) para cifrar secciones de un archivo Web.config y administrar las claves de cifrado. ASP.NET descifra el archivo de configuración cuando lo procesa. La identidad de la aplicación ASP.NET debe tener acceso de lectura a la clave de cifrado utilizada para cifrar y descifrar las secciones cifradas.
DPAPIProtectedConfigurationProvider	Usa la API de protección de datos (DPAPI) de Windows para cifrar y descifrar las secciones de configuración. Usa los servicios criptográficos integrados de Windows y se puede configurar para la protección específica de equipo o para la protección específica de cuenta de usuario. La protección específica de equipo resulta útil cuando varias aplicaciones del mismo servidor deben compartir información. La protección específica de cuenta de usuario se puede utilizar para los servicios que se ejecutan con una identidad de usuario concreta, como un entorno de hospedaje compartido. Cada aplicación se ejecuta con una identidad independiente que limita el acceso a recursos como los archivos y las bases de datos.

Ambos proveedores proporcionan cifrado de datos de alta seguridad. No obstante, si prevé usar el mismo archivo de configuración de cifrado en varios servidores como, por ejemplo, una granja de servidores web, sólo **RsaProtectedConfigurationProvider** permite exportar las claves de cifrado usadas para cifrar los datos e importarlas a otro servidor.

Uso de clases de configuración

El espacio de nombres System.Configuration proporciona clases para trabajar con valores de configuración mediante programación. La clase ConfigurationManager proporciona acceso a los archivos de configuración de equipo, aplicación y usuario. Si crea una aplicación ASP.NET, puede usar la clase WebConfigurationManager, que proporciona las mismas funciones a la vez que permite tener acceso a configuración única de las aplicaciones ASP.NET, como la que se encuentra en **<system.web>**.

Nota:

El espacio de nombres System.Security.Cryptography contiene clases que proporcionan opciones adicionales para cifrar y descifrar datos. Use estas clases si requiere servicios criptográficos que no están disponibles cuando se usa la configuración protegida. Algunas de estas clases son contenedores de Microsoft CryptoAPI no administrado, mientras que otras son simplemente implementaciones administradas.

Ejemplo de App.config

En este ejemplo se muestra cómo alternar el cifrado de la sección **connectionStrings** de un archivo **app.config** para una aplicación de Windows. En este ejemplo, el procedimiento recibe el nombre de la aplicación como argumento, por ejemplo, "MyApplication.exe". A continuación, el archivo **app.config** se cifra y se copia en la carpeta que contiene el ejecutable con el nombre "MyApplication.exe.config".

Nota:

La cadena de conexión sólo se puede descifrar en el equipo donde se ha cifrado.

El código usa el método OpenExeConfiguration para abrir el archivo **app.config** y editarlo; el método GetSection devuelve la sección **connectionStrings**. A continuación, el código comprueba la propiedad IsProtected y llama a ProtectSection para cifrar la sección si no está cifrada. Para descifrar la sección se llama al método UnProtectSection()()(). El método Save completa la operación y guarda los cambios.

Nota:

Para ejecutar el código, debe establecer una referencia al archivo **System.Configuration.dll** del proyecto.

```
Shared Sub ToggleConfigEncryption(ByVal exeConfigName As String)
    ' Takes the executable file name without the
    ' .config extension.
    Try
        ' Open the configuration file and retrieve
        ' the connectionStrings section.
        Dim config As Configuration = ConfigurationManager. _
            OpenExeConfiguration(exeConfigName)
        Dim section As ConnectionStringsSection = DirectCast( _
            config.GetSection("connectionStrings"), _
            ConnectionStringsSection)
        If section.SectionInformation.IsProtected Then
            ' Remove encryption.
            section.SectionInformation.UnprotectSection()
        Else
            ' Encrypt the section.
            section.SectionInformation.ProtectSection( _
              "DataProtectionConfigurationProvider")
        End If
```

```
        ' Save the current configuration.
        config.Save()
        Console.WriteLine("Protected={0}", _
        section.SectionInformation.IsProtected)
    Catch ex As Exception
        Console.WriteLine(ex.Message)
    End Try
End Sub
```

Ejemplo de Web.config

Este ejemplo usa el método OpenWebConfiguration de **WebConfigurationManager**. Observe que en este caso puede indicar la ruta de acceso relativo al archivo **Web.config** mediante una tilde. El código requiere una referencia a la clase **System.Web.Configuration**.

```
Shared Sub ToggleWebEncrypt()
    ' Open the Web.config file.
    Dim config As Configuration = WebConfigurationManager. _
      OpenWebConfiguration("~")
    ' Get the connectionStrings section.
    Dim section As ConnectionStringsSection = DirectCast( _
        config.GetSection("connectionStrings"), _
        ConnectionStringsSection)
    ' Toggle encryption.
    If section.SectionInformation.IsProtected Then
        section.SectionInformation.UnprotectSection()
    Else
        section.SectionInformation.ProtectSection( _
          "DataProtectionConfigurationProvider")
    End If
    ' Save changes to the Web.config file.
    config.Save()
End Sub
```

6.3. Agrupación de Conexiones

La conexión a un origen de datos puede ser un proceso largo. Para reducir el costo de abrir conexiones, ADO.NET emplea una técnica de optimización llamada agrupación de conexiones, que reduce el costo de abrir y cerrar conexiones repetidas veces. Los proveedores de datos de .NET Framework tratan de forma diferente la agrupación de conexiones.

6.3.1. Agrupación de Conexiones en SQL Server

La conexión a un servidor de base de datos consta normalmente de varios pasos que requieren mucho tiempo. Se debe establecer un canal físico, como un socket o una canalización con nombre, debe tener lugar el protocolo de enlace con el servidor, se debe analizar la información de la cadena de conexión, el servidor debe autenticar la conexión, se deben ejecutar comprobaciones para la inscripción en la transacción actual, etc.

En la práctica, la mayoría de las aplicaciones solamente utilizan unas cuantas configuraciones diferentes para las conexiones. Esto significa que durante la ejecución de la aplicación, muchas conexiones

idénticas se abrirán y cerrarán de forma repetida. Para reducir el costo de la apertura de conexiones, ADO.NET emplea una técnica de optimización denominada agrupación de conexiones.

La agrupación de conexiones reduce el número de veces que es necesario abrir nuevas conexiones. El concentrador mantiene la propiedad de la conexión física. Para administrar las conexiones, mantiene un conjunto de conexiones activas para cada configuración de conexión dada. Cada vez que un usuario llama a **Open** en una conexión, el agrupador comprueba si hay una conexión disponible en el grupo. Si hay disponible una conexión agrupada, la devuelve a la persona que llama en lugar de abrir una nueva. Cuando la aplicación llama a **Close** en la conexión, el agrupador la devuelve al conjunto agrupado de conexiones activas en lugar de cerrarla. Una vez que la conexión vuelve al grupo, ya está preparada para volverse a utilizar en la siguiente llamada a **Open**.

Sólo se pueden agrupar conexiones con la misma configuración. ADO.NET mantiene varios grupos de forma simultánea, uno para cada configuración. Las conexiones se dividen en grupos por cadena de conexión, y por identidad de Windows si se utiliza seguridad integrada. Las conexiones también se agrupan en función de si están incluidas en una transacción.

La agrupación de conexiones puede mejorar de forma significativa el rendimiento y la escalabilidad de la aplicación. De forma predeterminada, la agrupación de conexiones está habilitada en ADO.NET. A menos que la deshabilite explícitamente, el agrupador optimiza las conexiones a medida que se abren y cierran en la aplicación. También puede proporcionar varios modificadores de cadena de conexión para controlar el comportamiento de agrupación de conexiones. Para obtener más información, vea "Control de la agrupación de conexiones con palabras clave de cadena de conexión" más adelante en este tema.

⊟ Creación y asignación del grupo

Cuando se abre una conexión por primera vez, se crea un grupo de conexión basado en un algoritmo de coincidencia exacta que asocia el grupo con la cadena de conexión de la conexión. Cada grupo de conexión se asocia con una cadena de conexión distinta. Si se abre una nueva conexión y la cadena de conexión no coincide exactamente con un grupo existente, se crea un nuevo grupo. Las conexiones se agrupan por proceso, por dominio de aplicación, por cadena de conexión y, cuando se utiliza seguridad integrada, por identidad de Windows. Las cadenas de conexión también deben ser una coincidencia exacta; las palabras clave indicadas en un orden diferente para la misma conexión se agruparán por separado.

En el siguiente ejemplo con C#, se crean tres nuevos objetos SqlConnection, pero sólo se necesitan dos grupos de conexión para administrarlos. Observe que las cadenas de conexión primera y segunda difieren en el valor asignado a Initial Catalog.

```
using (SqlConnection connection = new SqlConnection(
  "Integrated Security=SSPI;Initial Catalog=Northwind"))
    {
        connection.Open();
        // Pool A is created.
    }
using (SqlConnection connection = new SqlConnection(
  "Integrated Security=SSPI;Initial Catalog=pubs"))
    {
        connection.Open();
        // Pool B is created because the connection strings differ.
    }
```

```
using (SqlConnection connection = new SqlConnection(
  "Integrated Security=SSPI;Initial Catalog=Northwind"))
    {
        connection.Open();
        // The connection string matches pool A.
    }
```

Si no se especifica **MinPoolSize** en la cadena de conexión o se especifica como cero, las conexiones del grupo se cerrarán tras un período de inactividad. No obstante, si el **MinPoolSize** especificado es mayor que cero, el grupo de conexión no se destruye hasta que se descarga el **AppDomain** y finaliza el proceso. El mantenimiento de grupos inactivos o vacíos supone una sobrecarga mínima para el sistema.

Nota:

El grupo se borra automáticamente cuando se produce un error irrecuperable, como una conmutación por error.

Adición de conexiones

Por cada cadena de conexión única se crea un grupo de conexión. Cuando se crea un grupo, se crean y agregan al grupo varios objetos de conexión y se satisface así el requisito de tamaño mínimo del grupo. Las conexiones se agregan al grupo cuando es necesario, hasta el tamaño máximo del grupo especificado (100 es el valor predeterminado), y se liberan de nuevo en el grupo cuando se cierran o eliminan.

Cuando se solicita un objeto SqlConnection, se obtiene del grupo si se encuentra disponible una conexión que se pueda utilizar. Una conexión de este tipo debe estar sin utilizar, tener un contexto de transacción coincidente o no estar asociada con ningún contexto de transacción y tener un vínculo válido al servidor.

El concentrador de conexión satisface las solicitudes de conexión al reasignar las conexiones conforme se liberan de nuevo en el grupo. Si se ha alcanzado el tamaño máximo del grupo y no hay disponible ninguna conexión que se pueda utilizar, la solicitud se pone en la cola. A continuación, el concentrador intenta reclamar las conexiones hasta que se agota el tiempo de espera (el valor predeterminado es 15 segundos). Si no puede satisfacer la solicitud antes de que se agote el tiempo de espera de la conexión, se inicia una excepción.

Precaución:

Se recomienda encarecidamente cerrar siempre la conexión cuando se termine de utilizar para que regrese al grupo. Para ello, puede utilizar los métodos **Close** o **Dispose** del objeto **Connection**, o abrir todas las conexiones dentro de una instrucción **using** en C#, o de una instrucción **Using** en Visual Basic. Es posible que las conexiones que no se cierran explícitamente no se puedan agregar ni puedan regresar al grupo.

Nota:

No llame a **Close** o a **Dispose** en un objeto **Connection**, un objeto **DataReader** o cualquier otro objeto administrado en el método **Finalize** de la clase. En un finalizador, libere sólo los recursos no administrados que pertenezcan directamente a su clase. Si la clase no dispone de recursos no administrados, no incluya un método **Finalize** en la definición de clase.

Cómo quitar conexiones

El agrupador de conexiones quita una conexión del grupo después de haber estado inactiva durante un período de tiempo prolongado, o si detecta que se ha roto la conexión con el servidor. Tenga en cuenta que una conexión rota sólo puede detectarse después de intentar comunicarse con el servidor. Si se

encuentra que una conexión ya no está conectada al servidor, se marca como no válida. Las conexiones no válidas se quitan del grupo de conexión sólo cuando se cierran o reclaman.

Si existe una conexión en un servidor que ha desaparecido, se puede extraer del grupo aunque el agrupador de conexiones no haya detectado la conexión rota y la haya marcado como no válida. El motivo es que la sobrecarga de comprobar que la conexión es aún válida eliminaría los beneficios de tener un concentrador y ocasionaría que se produjera otro viaje de ida y vuelta (round trip) al servidor. Cuando esto ocurre, el primer intento para usar la conexión detectará que ésta se ha roto y se iniciará una excepción.

⊟ Borrado del grupo

ADO.NET 2.0 introduce dos nuevos métodos para borrar el grupo: ClearAllPools y ClearPool. **ClearAllPools** borra los grupos de conexión de un proveedor dado, y **ClearPool** borra el grupo de conexión que está asociado a una conexión concreta. Si en el momento de la llamada se están usando conexiones, se marcan de forma adecuada. Cuando se cierran, se eliminan en lugar de devolverse al grupo.

⊟ Compatibilidad con transacciones

Las conexiones se extraen del grupo y se asignan en función del contexto de transacción. A menos que se especifique Enlist=false en la cadena de conexión, el grupo de conexión garantiza que la conexión está dada de alta en el contexto de Current. Cuando se cierra una conexión y se devuelve al grupo con una transacción **System.Transactions** dada de alta, se reserva de forma que la siguiente solicitud de ese grupo de conexiones con la misma transacción **System.Transactions** devolverá la misma conexión, si está disponible. Si se emite dicha solicitud y no hay conexiones agrupadas disponibles, se extrae una conexión de la parte sin transacción del grupo y se le da de alta. Si no hay conexiones disponibles en cualquier área del grupo, se crea y da de alta una nueva conexión.

Cuando se cierra una conexión, se libera de nuevo en el grupo y en la subdivisión adecuada en función de su contexto de transacción. Por lo tanto, puede cerrar la conexión sin generar un error, incluso aunque aún haya pendiente una transacción distribuida. Esto permite confirmar o anular la transacción distribuida más adelante.

⊟ Control de la agrupación de conexiones con palabras clave de cadena de conexión

La propiedad **ConnectionString** del objeto SqlConnection admite pares de clave y valor de cadena de conexión que se pueden utilizar para ajustar el comportamiento de la lógica de agrupación de conexiones.

⊟ Fragmentación de grupos

La fragmentación de grupos es un problema común en muchas aplicaciones web en las que la aplicación puede crear gran cantidad de grupos que no están libres hasta que finaliza el proceso. El resultado es un gran número de conexiones abiertas que consumen memoria, lo que da lugar a un bajo rendimiento.

Fragmentación de grupos debido a la seguridad integrada

Las conexiones se agrupan de acuerdo con la cadena de conexión y la identidad del usuario. Por lo tanto, si utiliza autenticación básica o autenticación de Windows en el sitio web y un inicio de sesión de seguridad integrada, obtendrá un grupo por usuario. Aunque de esta manera se mejora el rendimiento de las posteriores solicitudes de base de datos de un solo usuario, ese usuario no podrá aprovechar las conexiones realizadas por otros usuarios. Además, como resultado habrá una conexión como mínimo por usuario al servidor de base de datos. Se trata de un efecto secundario de una determinada arquitectura

de aplicaciones web que los programadores deben sopesar frente a los requisitos de seguridad y auditoría.

Fragmentación de grupos debido a muchas bases de datos
Muchos proveedores de servicios Internet hospedan varios sitios web en un único servidor. Puede que utilicen una sola base de datos para confirmar un inicio de sesión de autenticación de formularios y luego abran una conexión a una base de datos específica para ese usuario o grupo de usuarios. La conexión a la base de datos de autenticación es agrupada y utilizada por todo el mundo. Sin embargo, hay un grupo independiente de conexiones con cada base de datos, lo que implica un aumento del número de conexiones con el servidor.

Este es también un efecto secundario del diseño de la aplicación. Existe, sin embargo, una forma relativamente sencilla de evitarlo sin comprometer la seguridad cuando se establece conexión con SQL Server. En lugar de realizar una conexión a una base de datos diferente por cada usuario o grupo, realice una conexión a la misma base de datos en el servidor y, luego, ejecute la instrucción USE de Transact-SQL para cambiar a la base de datos deseada. En el siguiente fragmento de código se muestra la creación de una conexión inicial con la base de datos **master** y el cambio a la base de datos deseada especificada en la variable de cadena databaseName.

```
' Assumes that command is a valid SqlCommand object and that
' connectionString connects to master.
    command.Text = "USE DatabaseName"
Using connection As New SqlConnection(connectionString)
    connection.Open()
    command.ExecuteNonQuery()
End Using
```

⊟ Funciones de aplicación y agrupación de conexiones

Una vez activada una función de aplicación de SQL Server al llamar al procedimiento almacenado de sistema **sp_setapprole**, no se puede restablecer el contexto de seguridad de la conexión. Sin embargo, cuando se habilita la agrupación, la conexión se devuelve al grupo y se produce un error al utilizar de nuevo la conexión agrupada.

Alternativas a las funciones de aplicación
Si usa SQL Server 2005, se recomienda aprovechar las ventajas de los nuevos mecanismos de seguridad que se pueden usar en lugar de las funciones de la aplicación.

6.3.2. Agrupación de Conexiones OLE DB, ODBC y Oracle

La agrupación de conexiones puede mejorar de forma significativa el rendimiento y la escalabilidad de una aplicación. En esta sección se describe la agrupación de conexiones en los proveedores de datos para OLE DB, ODBC y Oracle de .NET Framework.

⊟ Agrupación de conexiones para OLE DB

El proveedor de datos de .NET Framework para OLE DB agrupa automáticamente las conexiones mediante la agrupación de sesiones OLE DB. Se pueden utilizar argumentos de cadena de conexión para habilitar o deshabilitar servicios OLE DB, incluida la agrupación. Por ejemplo, la siguiente cadena de conexión deshabilita la agrupación de sesiones OLE DB y la inscripción automática de transacciones.

```
Provider=SQLOLEDB;OLE DB Services=-4;Data Source=localhost;Integrated
Security=SSPI;
```

Se recomienda cerrar siempre o eliminar una conexión cuando termine de utilizarla, para que la conexión pueda regresar al grupo. Es posible que las conexiones que no se cierran explícitamente no puedan regresar al grupo. Por ejemplo, una conexión que se ha salido del ámbito pero que no se ha cerrado explícitamente sólo se devolverá al grupo de conexión si se ha alcanzado el tamaño máximo del grupo y la conexión aún es válida.

⊟ Agrupación de conexiones para ODBC

La agrupación de conexiones para el proveedor de datos de .NET Framework para ODBC se administra a través del Administrador de controladores ODBC que se utiliza en la conexión, y que no está influido por dicho proveedor.

Para habilitar o deshabilitar la agrupación de conexiones, abra **Administrador de orígenes de datos ODBC** en la carpeta Herramientas administrativas del Panel de control. La ficha **Agrupación de conexiones** permite especificar los parámetros de agrupación de conexiones de cada controlador ODBC instalado. Tenga en cuenta que los cambios en la agrupación de conexiones de un controlador ODBC específico afectarán a todas las aplicaciones que utilicen dicho controlador.

⊟ Agrupación de conexiones para OracleClient

El proveedor de datos de .NET Framework para Oracle ofrece agrupación automática de conexiones para la aplicación cliente de ADO.NET. También puede suministrar varios modificadores de cadena de conexión para controlar el comportamiento de agrupación de conexiones (vea "Control de la agrupación de conexiones con palabras clave de cadena de conexión", más adelante en este tema).

Creación y asignación del grupo

Cuando se abre una conexión, se crea un grupo de conexión basado en un algoritmo de coincidencia exacta que asocia el grupo con la cadena de conexión de la conexión. Cada grupo de conexión se asocia con una cadena de conexión distinta. Si se abre una nueva conexión y la cadena de conexión no coincide exactamente con un grupo existente, se crea un nuevo grupo.

Una vez creados, los grupos de conexión no se destruyen hasta que finaliza el proceso activo. Mantener grupos inactivos o vacíos consume muy pocos recursos del sistema.

Adición de conexiones

Para cada cadena de conexión única se crea un grupo de conexión. Cuando se crea un grupo, se crean y agregan al grupo varios objetos de conexión; así se satisface el requisito de tamaño mínimo del grupo. Las conexiones se agregan al grupo cuando es necesario, hasta el tamaño máximo del grupo.

Cuando se solicita un objeto OracleConnection, se obtiene del grupo si se encuentra disponible una conexión que se pueda utilizar. Una conexión de este tipo debe estar sin utilizar en ese momento, tener un contexto de transacción coincidente o no estar asociada con ningún contexto de transacción, y tener un vínculo válido al servidor.

Si se ha alcanzado el tamaño máximo del grupo y no hay disponible ninguna conexión que se pueda utilizar, la solicitud se pone en la cola. El concentrador de conexión satisface estas solicitudes al reasignar las conexiones conforme se liberan de nuevo en el grupo, lo cual ocurre cuando éstas se cierran o eliminan.

Eliminación de conexiones

El concentrador de conexión quita una conexión del grupo después de que ha estado inactiva durante un período de tiempo prolongado o si detecta que se ha roto la conexión al servidor. Tenga en cuenta que esto sólo puede detectarse después de intentar comunicarse con el servidor. Si se encuentra que una

conexión ya no está conectada al servidor, se marca como no válida. El concentrador de conexión analiza periódicamente los grupos en busca de objetos que se han liberado en el grupo y marcado como no válidos. Luego, estas conexiones se quitan de forma permanente.

Si existe una conexión a un servidor que ha desaparecido, se puede extraer del grupo si el concentrador de conexión no ha detectado la conexión rota y la ha marcado como no válida. Cuando esto se produce, se genera una excepción. No obstante, aun así deberá cerrar la conexión para liberarla de nuevo en el grupo.

No llame a **Close** o a **Dispose** en un objeto **Connection**, un objeto **DataReader** o cualquier otro objeto administrado en el método **Finalize** de la clase. En un finalizador, libere sólo los recursos no administrados que pertenezcan directamente a su clase. Si la clase no dispone de recursos no administrados, no incluya un método **Finalize** en la definición de clase.

Compatibilidad con transacciones
Las conexiones se extraen del grupo y se asignan en función del contexto de transacción. Es necesario que el subproceso solicitante y la conexión asignada coincidan. Por lo tanto, cada grupo de conexión se divide realmente en conexiones que no tienen asociado ningún contexto de transacción y en N subdivisiones, cada una de las cuales contiene conexiones con un contexto de transacción particular.

Cuando se cierra una conexión, se libera de nuevo en el grupo y en la subdivisión adecuada en función de su contexto de transacción. Por lo tanto, puede cerrar la conexión sin generar un error, incluso aunque aún haya pendiente una transacción distribuida. Esto le permite confirmar o anular la transacción distribuida más adelante.

Control de la agrupación de conexiones con palabras clave de cadena de conexión
La propiedad ConnectionString del objeto OracleConnection admite pares de clave y valor de cadena de conexión que se pueden utilizar para ajustar el comportamiento de la lógica de agrupación de conexiones.

En la siguiente tabla se describen los valores ConnectionString que puede utilizar para ajustar el comportamiento de agrupación de conexiones.

Nombre	Default	Descripción
Connection Lifetime	0	Cuando una conexión se devuelve al grupo, su hora de creación se compara con la hora actual y, si ese marco temporal (en segundos) supera el valor especificado por **Connection Lifetime**, la conexión se destruye. Esto resulta de utilidad en configuraciones agrupadas para forzar el equilibrio de carga entre un servidor en ejecución y uno que acaba de conectarse. Un valor de cero (0) hará que las conexiones agrupadas tengan el tiempo de espera máximo.
Enlist	'true'	Cuando es **true**, el concentrador inscribe automáticamente la conexión en el contexto de transacción actual del subproceso de creación, si existe un contexto de transacción.
Max Pool Size	100	El número máximo de conexiones permitido en el grupo.
Min Pool Size	0	El número mínimo de conexiones mantenido en el grupo.

Pooling	'true'	Cuando es **true**, la conexión se extrae del grupo adecuado o, si es necesario, se crea y agrega al grupo correcto.

6.4. Comandos

Una vez establecida una conexión a un origen de datos, puede ejecutar comandos y devolver resultados desde el mismo mediante un objeto DbCommand. Para crear un comando, puede utilizar uno de los constructores de comando del proveedor de datos de .NET Framework con el que esté trabajando. Los constructores pueden aceptar argumentos opcionales, como una instrucción SQL para ejecutar en el origen de datos, un objeto DbConnection o un objeto DbTransaction. También puede configurar dichos objetos como propiedades del comando. También puede crear un comando para una determinada conexión mediante el método CreateCommand de un objeto **DbConnection**. La instrucción SQL que ejecuta el comando se puede configurar mediante la propiedad CommandText.

Cada proveedor de datos de .NET Framework incluido en .NET Framework cuenta con un objeto **Command**: El proveedor de datos de .NET Framework para OLE DB incluye un objeto OleDbCommand, el proveedor de datos de .NET Framework para SQL Server incluye un objeto SqlCommand, el proveedor de datos de .NET Framework para ODBC incluye un objeto OdbcCommand y el proveedor de datos de .NET Framework para Oracle incluye un objeto OracleCommand.

6.4.1. Ejecutar un Comando

Cada proveedor de datos de .NET Framework incluido en .NET Framework dispone de su propio objeto command que hereda de DbCommand. El proveedor de datos de .NET Framework para OLE DB incluye un objeto OleDbCommand, el proveedor de datos de .NET Framework para SQL Server incluye un objeto SqlCommand, el proveedor de datos de .NET Framework para ODBC incluye un objeto OdbcCommand y el proveedor de datos de .NET Framework para Oracle incluye un objeto OracleCommand. Cada uno de estos objetos expone métodos para ejecutar comandos que se basan en el tipo de comando y el valor devuelto deseado, tal como se describe en la tabla siguiente.

Comando	Valor devuelto
ExecuteReader	Devuelve un objeto **DataReader**.
ExecuteScalar	Devuelve un solo valor escalar.
ExecuteNonQuery	Ejecuta un comando que no devuelve ninguna fila.
ExecuteXMLReader	Devuelve un valor XmlReader. Sólo está disponible para un objeto **SqlCommand**.

Cada objeto command con establecimiento inflexible de tipos admite también una enumeración CommandType que especifica cómo se interpreta una cadena de comando, tal como se describe en la tabla siguiente.

CommandType	Descripción
Text	Comando de SQL que define las instrucciones que se van a ejecutar en el origen de datos.

StoredProcedure	Nombre del procedimiento almacenado. Puede usar la propiedad **Parameters** de un comando para tener acceso a los parámetros de entrada y de salida y a los valores devueltos, independientemente del método **Execute** al que se llame. Al usar **ExecuteReader**, no es posible el acceso a los valores devueltos y los parámetros de salida hasta que se cierra **DataReader**.
TableDirect	Nombre de una tabla.

Ejemplo

En el ejemplo de código siguiente se muestra cómo se crea un objeto SqlCommand para ejecutar un procedimiento almacenado mediante el establecimiento de sus propiedades. Para especificar el parámetro de entrada del procedimiento almacenado se usa un objeto SqlParameter. El comando se ejecuta con el método ExecuteReader y el resultado de SqlDataReader se muestra en la ventana de consola.

```
Shared Sub GetSalesByCategory(ByVal connectionString As String, _
    ByVal categoryName As String)
    Using connection As New SqlConnection(connectionString)
        ' Create the command and set its properties.
        Dim command As SqlCommand = New SqlCommand()
        command.Connection = connection
        command.CommandText = "SalesByCategory"
        command.CommandType = CommandType.StoredProcedure
        ' Add the input parameter and set its properties.
        Dim parameter As New SqlParameter()
        parameter.ParameterName = "@CategoryName"
        parameter.SqlDbType = SqlDbType.NVarChar
        parameter.Direction = ParameterDirection.Input
        parameter.Value = categoryName
        ' Add the parameter to the Parameters collection.
        command.Parameters.Add(parameter)
        ' Open the connection and execute the reader.
        connection.Open()
        Dim reader As SqlDataReader = command.ExecuteReader()
        If reader.HasRows Then
            Do While reader.Read()
                Console.WriteLine("{0}: {1:C}", _
                  reader(0), reader(1))
            Loop
        Else
            Console.WriteLine("No rows returned.")
        End If
    End Using
End Sub
```

Solución de problemas de comandos

El proveedor de datos de .NET Framework para SQL Server agrega contadores de rendimiento que permiten detectar problemas intermitentes relacionados con errores en la ejecución de comandos.

6.4.2. Configurar Parámetros

Los objetos de comando usan parámetros para pasar valores a instrucciones SQL o procedimientos almacenados que permiten realizar operaciones de comprobación de tipos y validación. A diferencia del texto de comando, la entrada de parámetros se trata como un valor literal, y no como código ejecutable. Esta característica ayuda en la protección contra ataques por "inyección de código SQL", en los que un atacante inserta un comando en una instrucción SQL que pone en peligro la seguridad del servidor. Además de las ventajas en la seguridad, los comandos con parámetros proporcionan un método práctico para organizar los valores que se pasan a un origen de datos.

Para crear un objeto DbParameter, se puede usar su constructor o bien se puede agregar a DbParameterCollection mediante una llamada al método **Add** de la colección DbParameterCollection. El método **Add** acepta como entrada argumentos del constructor o cualquier objeto de parámetro ya existente, en función del proveedor de datos.

⊟ Proporcionar la propiedad ParameterDirection

Cuando se agregan parámetros distintos de los parámetros de entrada, se debe proporcionar una propiedad ParameterDirection. En la tabla siguiente se muestran los valores de **ParameterDirection** que se pueden usar con la enumeración ParameterDirection.

Nombre del miembro	Descripción
Input	Se trata de un parámetro de entrada. Éste es el valor predeterminado.
InputOutput	El parámetro se puede comportar tanto de entrada como de salida.
Output	Se trata de un parámetro de salida.
ReturnValue	El parámetro representa un valor devuelto de una operación como, por ejemplo, un procedimiento almacenado, una función integrada o una función definida por el usuario.

⊟ Trabajar con marcadores de posición de parámetros

La sintaxis de los marcadores de posición de parámetros varía en función del origen de datos. Los proveedores de datos de .NET Framework administran la asignación de nombres y la especificación de parámetros y de marcadores de posición de parámetros de forma diferente. Esta sintaxis se personaliza para un origen de datos específico, como se describe en la tabla siguiente.

Proveedor de datos	Sintaxis de denominación de parámetros
System.Data.SqlClient	Usa parámetros con nombre con el formato **@**nombreDeParámetro.
System.Data.OleDb	Usa marcadores de parámetro de posición, indicados con un signo de interrogación (**?**).
System.Data.Odbc	Usa marcadores de parámetro de posición, indicados con un signo de interrogación (**?**).
System.Data.OracleClient	Usa parámetros con nombre, con el formato **:**nombreDeParámetro (o nombreDeParámetro).

⊟ Especificar tipos de datos de parámetro

El tipo de datos de un parámetro es específico del proveedor de datos de .NET Framework. Al especificar el tipo, el valor de **Parameter** se convierte en el tipo del proveedor de datos de .NET Framework antes

de pasar el valor al origen de datos. Si lo desea, puede especificar el tipo de un objeto **Parameter** de forma genérica estableciendo la propiedad **DbType** del objeto **Parameter** en un DbType determinado.

El tipo del proveedor de datos de .NET Framework de un objeto **Parameter** se deduce del tipo de .NET Framework del **Value** del objeto **Parameter**, o del **DbType** del objeto **Parameter**. En la siguiente tabla se muestra el tipo deducido de **Parameter** en función del objeto que se ha pasado como valor **Parameter** o del **DbType** especificado.

Tipo de .NET Framework	System.Data .DbType	SqlDbType	OleDbType	OdbcType	OracleType
bool	Bolean	Bit	Boolean	Bit	Byte
byte	Byte	TinyInt	UnsignedTinyInt	TinyInt	Byte
byte[]	Binary	VarBinary**.** Esta conversión implícita generará un error en el caso de que la matriz de bytes tenga un tamaño superior al tamaño máximo de un tipo VarBinary, que es de 8.000 bytes. En matrices de bytes con más de 8.000 bytes, establezca de forma explícita el tipo SqlDbType.	VarBinary	Binary	Raw
char		No se admite la deducción de un tipo SqlDbType a partir de char.	Char	Char	Byte
DateTime	DateTime	DateTime	DBTimeStamp	DateTime	DateTime
DateTimeOffset	DateTime Offset	DateTimeOffset en SQL Server 2008. La deducción de un tipo SqlDbType a partir de DateTimeOffset no se admite en versiones de SQL Server anteriores a SQL Server 2008.			DateTime

Decimal	Decimal	Decimal	Decimal	Numeric	Number
double	Double	Float	Double	Double	Double
float	Single	Real	Single	Real	Float
Guid	Guid	UniqueIdentifier	Guid	UniqueIdentifier	Raw
Int16	Int16	SmallInt	SmallInt	SmallInt	Int16
Int32	Int32	Int	Int	Int	Int32
Int64	Int64	BigInt	BigInt	BigInt	Number
object	Object	Variant	Variant	No se admite la deducción de un tipo OdbcType a partir de Object.	Blob
string	String	NVarChar. Esta conversión implícita generará un error en el caso de que la cadena tenga un tamaño superior al tamaño máximo de un tipo NVarChar, que es de 4.000 caracteres. En cadenas con más de 4.000 caracteres, establezca de forma explícita el tipo SqlDbType.	VarWChar	NVarChar	NVarChar
Timespan	Time	Time en SQL Server 2008. La deducción de un tipo SqlDbType a partir de TimeSpan no se admite en versiones de SQL Server anteriores a SQL Server 2008.	DBTime	Time	DateTime
UInt16	UInt16	No se admite la deducción de un tipo SqlDbType a partir de	UnsignedSmallInt	Int	UInt16

		UInt16.			
UInt32	UInt32	No se admite la deducción de un tipo SqlDbType a partir de UInt32.	UnsignedInt	BigInt	UInt32
UInt64	UInt64	No se admite la deducción de un tipo SqlDbType a partir de UInt64.	UnsignedBigInt	Numeric	Number
	AnsiString	VarChar	VarChar	VarChar	VarChar
	AnsiString FixedLength	Char	Char	Char	Char
	Currency	Money	Currency	No es posible deducir el valor de **OdbcType** a partir de **Currency**.	Number
	Date	Date en SQL Server 2008. La deducción de un tipo SqlDbType a partir de Date no se admite en versiones de SQL Server anteriores a SQL Server 2008.	DBDate	Date	DateTime
	SByte	No se admite la deducción de un tipo SqlDbType a partir de SByte.	TinyInt	No se admite la deducción de un tipo **OdbcType** a partir de SByte.	SByte
	String FixedLength	NChar	WChar	NChar	NChar
	Time	Time en SQL Server 2008. La deducción de un tipo SqlDbType a partir de Time no se admite en versiones de SQL Server anteriores a SQL Server 2008.	DBTime	Time	DateTime
	VarNumeric	No se admite la	VarNumeric	No se admite la	Number

		deducción de un tipo SqlDbType a partir de VarNumeric.		deducción de un tipo **OdbcType** a partir de VarNumeric.	

Nota:

Las conversiones de valores de tipo decimal en otros tipos de valor son conversiones de restricción que redondean el valor decimal al valor entero más próximo a cero. Si el resultado de la conversión no puede representarse en el tipo de destino, se produce OverflowException.

Nota:

Cuando se envía un valor de parámetro null al servidor, se debe especificar DBNull en lugar de **null** (**Nothing** en Visual Basic). El valor nulo en el sistema es un objeto vacío que no tiene ningún valor. Para representar los valores nulos, se usa DBNull. Para obtener más información acerca de valores nulos de base de datos, vea Tratamiento de valores NULL (ADO.NET).

⊟ Derivar la información de parámetros

Los parámetros también se pueden derivar de un procedimiento almacenado mediante la clase **DbCommandBuilder**. Las clases **SqlCommandBuilder** y **OleDbCommandBuilder** proporcionan un método estático, **DeriveParameters**, que rellena automáticamente la colección de parámetros de un objeto de comando que usa información de parámetros procedente de un procedimiento almacenado. Tenga en cuenta que **DeriveParameters** sobrescribirá toda la información de parámetros existente en el comando.

Nota:

La derivación de información de parámetros afecta al rendimiento, ya que precisa un viaje adicional de ida y vuelta (round trip) al origen de datos para recuperar la información. Si la información de los parámetros se conoce en tiempo de diseño, se puede mejorar el rendimiento de la aplicación si se establecen los parámetros con los valores correspondientes de forma explícita.

⊟ Usar parámetros con SqlCommand y con un procedimiento almacenado

Los procedimientos almacenados ofrecen numerosas ventajas en el caso de aplicaciones que procesan datos. Mediante el uso de procedimientos almacenados, las operaciones de bases de datos se pueden encapsular en un solo comando, optimizar para lograr el mejor rendimiento, y mejorar con seguridad adicional. Aunque es cierto que para llamar a un procedimiento almacenado basta con pasar en forma de instrucción SQL su nombre seguido de los argumentos de parámetros, el uso de la colección Parameters del objeto DbCommand de ADO.NET permite definir más explícitamente los parámetros del procedimiento almacenado, y tener acceso a los parámetros de salida y a los valores devueltos.

Nota:

Las instrucciones con parámetros se ejecutan en el servidor utilizando **sp_executesql,**; esto permite volver a utilizar el plan de consultas. Los cursores o las variables locales del lote de **sp_executesql** no son visibles para el lote que llama a **sp_executesql**. Los cambios en el contexto de base de datos sólo se mantienen hasta el final de la instrucción **sp_executesql**.

Cuando se usan parámetros con SqlCommand para ejecutar un procedimiento almacenado de SQL Server, los nombres de los parámetros agregados a la colección Parameters deben coincidir con los nombres de los marcadores de parámetro del procedimiento almacenado. El proveedor de datos de .NET Framework para SQL Server no admite el uso del marcador de posición de signo de interrogación de cierre (?) para pasar parámetros a una instrucción SQL o a un procedimiento almacenado. Este

proveedor trata los parámetros del procedimiento almacenado como parámetros con nombre y busca marcadores de parámetros coincidentes. Por ejemplo, el procedimiento almacenado **CustOrderHist** se define usando un parámetro denominado @CustomerID. Cuando el código ejecuta el procedimiento almacenado, también debe usar un parámetro denominado @CustomerID.

```
CREATE PROCEDURE dbo.CustOrderHist @CustomerID varchar(5)
```

Ejemplo

En este ejemplo se muestra cómo llamar a un procedimiento almacenado de SQL Server en la base de datos de ejemplo **Northwind**. El nombre del procedimiento almacenado es dbo.SalesByCategory e incluye un parámetro de entrada denominado @CategoryName con el tipo de datos nvarchar(15). El código crea una nueva clase SqlConnection dentro de un bloque en uso, de forma que la conexión se cierre cuando finalice el procedimiento. Se crean los objetos SqlCommand y SqlParameter, y se establecen sus propiedades. SqlDataReader ejecuta **SqlCommand** y devuelve el conjunto de resultados del procedimiento almacenado, mostrándolos en la ventana de consola.

Nota:

En lugar de crear objetos **SqlCommand** y **SqlParameter** y, a continuación, establecer propiedades en instrucciones independientes, puede usar uno de los constructores sobrecargados para establecer varias propiedades en una única instrucción.

```
Shared Sub GetSalesByCategory(ByVal connectionString As String, _
    ByVal categoryName As String)
    Using connection As New SqlConnection(connectionString)
        ' Create the command and set its properties.
        Dim command As SqlCommand = New SqlCommand()
        command.Connection = connection
        command.CommandText = "SalesByCategory"
        command.CommandType = CommandType.StoredProcedure
        ' Add the input parameter and set its properties.
        Dim parameter As New SqlParameter()
        parameter.ParameterName = "@CategoryName"
        parameter.SqlDbType = SqlDbType.NVarChar
        parameter.Direction = ParameterDirection.Input
        parameter.Value = categoryName
        ' Add the parameter to the Parameters collection.
        command.Parameters.Add(parameter)
        ' Open the connection and execute the reader.
        connection.Open()
        Dim reader As SqlDataReader = command.ExecuteReader()
        If reader.HasRows Then
            Do While reader.Read()
                Console.WriteLine("{0}: {1:C}", _
                  reader(0), reader(1))
            Loop
        Else
            Console.WriteLine("No rows returned.")
        End If
    End Using
End Sub
```

Utilizar parámetros con OleDbCommand o con OdbcCommand

Cuando se usan parámetros con OleDbCommand o con OdbcCommand, el orden de los parámetros agregados a la colección **Parameters** debe coincidir con el de los parámetros definidos en el procedimiento almacenado. El proveedor de datos de .NET Framework para OLE DB y el proveedor de datos de .NET Framework para ODBC consideran los parámetros de un procedimiento almacenado como marcadores de posición y aplican los valores de los parámetros en orden. Además, los parámetros de valores devueltos deben ser los primeros que se agreguen a la colección **Parameters**.

El proveedor de datos de .NET Framework para OLE DB y el proveedor de datos de .NET Framework para ODBC no admiten el uso de parámetros con nombre para pasar parámetros a una instrucción SQL o a un procedimiento almacenado. En este caso, se debe utilizar el marcador de posición de signo interrogación de cierre (?), como se muestra en el ejemplo siguiente.

```
SELECT * FROM Customers WHERE CustomerID = ?
```

Por eso, el orden en que se agregan los objetos **Parameter** a la colección **Parameters** debe coincidir exactamente con la posición del marcador de posición de interrogación de cierre correspondiente al parámetro.

Ejemplo de OleDb

```
Dim command As OleDbCommand = New OleDbCommand( _
  "SampleProc", connection)
command.CommandType = CommandType.StoredProcedure
Dim parameter As OleDbParameter = command.Parameters.Add( _
  "RETURN_VALUE", OleDbType.Integer)
parameter.Direction = ParameterDirection.ReturnValue
parameter = command.Parameters.Add( _
  "@InputParm", OleDbType.VarChar, 12)
parameter.Value = "Sample Value"
parameter = command.Parameters.Add( _
  "@OutputParm", OleDbType.VarChar, 28)
parameter.Direction = ParameterDirection.Output
```

Ejemplo de Odbc

```
Dim command As OdbcCommand = New OdbcCommand( _
  "{ ? = CALL SampleProc(?, ?) }", connection)
command.CommandType = CommandType.StoredProcedure
Dim parameter As OdbcParameter = command.Parameters.Add("RETURN_VALUE",
OdbcType.Int)
parameter.Direction = ParameterDirection.ReturnValue
parameter = command.Parameters.Add( _
  "@InputParm", OdbcType.VarChar, 12)
parameter.Value = "Sample Value"
parameter = command.Parameters.Add( _
  "@OutputParm", OdbcType.VarChar, 28)
parameter.Direction = ParameterDirection.Output
```

6.4.3. Generar Comandos con Objetos CommandBuilder

Cuando la propiedad **SelectCommand** se especifica de forma dinámica en tiempo de ejecución, por ejemplo a través de una herramienta de consulta que acepta un comando de texto del usuario, existe la posibilidad de que no se pueda especificar adecuadamente en tiempo de diseño el comando

InsertCommand, **UpdateCommand** o **DeleteCommand** correspondiente. Si el objeto DataTable se asigna a una única tabla de base de datos o se genera a partir de ella, puede utilizar el objeto DbCommandBuilder para generar automáticamente las propiedades **DeleteCommand**, **InsertCommand** y **UpdateCommand** de DbDataAdapter.

El requisito mínimo para que la generación automática de comandos funcione correctamente consiste en establecer la propiedad **SelectCommand**. El esquema de tabla que recupera la propiedad **SelectCommand** determina la sintaxis de las instrucciones INSERT, UPDATE y DELETE generadas automáticamente.

DbCommandBuilder debe ejecutar **SelectCommand** con el objeto de devolver los metadatos necesarios para construir los comandos SQL INSERT, UPDATE y DELETE. Por eso es necesario realizar un viaje adicional al origen de datos, con el consiguiente efecto adverso en el rendimiento.Para mejorar el rendimiento, debe especificar los comandos de forma explícita, en lugar de utilizar DbCommandBuilder.

SelectCommand también debe devolver como mínimo una clave principal o una columna única. Si no hay ninguna, se genera una excepción **InvalidOperation** y no se genera ningún comando.

Cuando se asocia con un objeto **DataAdapter**, DbCommandBuilder genera automáticamente las propiedades **InsertCommand**, **UpdateCommand** y **DeleteCommand** del objeto **DataAdapter** si son referencias nulas. Si ya existe algún objeto **Command** para una propiedad, se utilizará el objeto **Command** existente.

Las vistas de bases de datos creadas al unir una o varias tablas no se consideran una tabla única de base de datos. En este caso no puede utilizar DbCommandBuilder para generar comandos automáticamente y deberá especificarlos de manera explícita.

Es posible que desee asignar parámetros de salida a la fila actualizada de un **DataSet**. Una tarea habitual consiste en recuperar, a partir del origen de datos, el valor de un campo de identidad de generación automática o una marca de tiempo. DbCommandBuilder no asigna de forma predeterminada los parámetros de salida a las columnas de una fila actualizada. En este caso, debe especificar el comando de forma explícita.

⊟ Reglas para comandos generados automáticamente

En la tabla siguiente se muestran las reglas de la generación automática de comandos.

Comando	Regla
InsertCommand	Inserta una fila en el origen de datos para todas las filas de la tabla con una RowState con el valor Added. Inserta valores para todas las columnas actualizables, pero no para determinadas columnas como identidades, expresiones o marcas de tiempo.
UpdateCommand	Actualiza filas en el origen de datos para todas las filas de la tabla con una propiedad **RowState** con el valor Modified. Actualiza los valores de todas las columnas, con excepción de las que no son actualizables, como identidades o expresiones. Actualiza todas las filas en las que los valores de columna en el origen de datos coinciden con los valores de la columna de clave principal de la fila, siempre que las restantes columnas del origen de datos coincidan con los valores originales de la fila.
DeleteCommand	Elimina filas en el origen de datos para todas las filas de la tabla con una propiedad **RowState** con el valor Deleted. Elimina todas las filas en las que los valores de columna coinciden con los valores de la columna de clave principal

	de la fila, siempre que las restantes columnas del origen de datos coincidan con los valores originales de la fila.

Modelo de simultaneidad optimista para actualizaciones y eliminaciones

La lógica empleada en la generación automática de comandos para las instrucciones UPDATE y DELETE se basa en la simultaneidad optimista, es decir, los registros no se bloquean para ser editados y pueden ser modificados en cualquier momento por otros usuarios o procesos. Dado que existe la posibilidad de que un registro haya sido modificado después de que haya sido devuelto por la instrucción SELECT y antes de que se emita la instrucción UPDATE o DELETE, la instrucción UPDATE o DELETE generada automáticamente incluye una cláusula WHERE que especifica que la fila sólo se actualiza cuando contiene todos los valores originales y no ha sido eliminada del origen de datos. Esto evita que se sobrescriban los datos nuevos. Si una actualización generada automáticamente intenta actualizar una fila que ha sido eliminada o que no contiene los valores originales del DataSet, el comando no tienen ningún efecto en los registros y se inicia una excepción DBConcurrencyException.

Si desea que la instrucción UPDATE o DELETE se ejecute sin tener en cuenta los valores originales, debe establecer de forma explícita la propiedad **UpdateCommand** del **DataAdapter** sin utilizar la generación automática de comandos.

Limitaciones de la lógica de generación automática de comandos

La generación automática de comandos tiene las siguientes limitaciones.

Sólo tablas no relacionadas

La lógica de generación automática de comandos crea instrucciones INSERT, UPDATE o DELETE para tablas independientes sin tener en cuenta las relaciones que éstas puedan tener con otras tablas en el origen de datos. Por eso, se puede producir un error al llamar a **Update** para realizar cambios en una columna que participa de una restricción de clave externa en la base de datos.Para evitar esa excepción, no utilice DbCommandBuilder al actualizar las columnas que participan en una restricción de clave externa. En este caso debe especificar de forma explícita las instrucciones que se van a utilizar para llevar a cabo la operación.

Nombres de tabla y columna

La lógica de generación automática de comandos ocasiona un error cuando los nombres de las tablas o columnas incluyen algún carácter especial, como espacios, puntos, signos de exclamación y otros caracteres no alfanuméricos, aun en el caso de que se incluyan entre corchetes. Se pueden utilizar nombres completos de tabla, como catalog.schema.table.

Utilizar CommandBuilder para generar automáticamente una instrucción SQL

Para generar instrucciones SQL automáticamente para un **DataAdapter**, defina en primer lugar la propiedad **SelectCommand** del **DataAdapter** y, a continuación, cree un objeto **CommandBuilder** y especifique como argumento el **DataAdapter** para el que **CommandBuilder** generará automáticamente las instrucciones SQL.

```
' Assumes that connection is a valid SqlConnection object
' inside of a Using block.
Dim adapter As SqlDataAdapter = New SqlDataAdapter( _
  "SELECT * FROM dbo.Customers", connection)
Dim builder As SqlCommandBuilder = New SqlCommandBuilder(adapter)
builder.QuotePrefix = "["
builder.QuoteSuffix = "]"
```

Modificar SelectCommand

Es posible que se inicie una excepción si modifica valor **CommandText** de **SelectCommand** después de generar automáticamente los comandos INSERT, UPDATE o DELETE. Si el valor **SelectCommand.CommandText** modificado contiene información del esquema que sea incoherente con el valor **SelectCommand.CommandText** utilizado en el momento de la generación automática de los comandos de inserción, actualización o eliminación, las futuras llamadas al método **DataAdapter.Update** pueden tratar de obtener acceso a columnas que ya no existen en la tabla actual a la que hace referencia **SelectCommand**, con lo que se iniciará una excepción.

Puede actualizar la información del esquema que utiliza **CommandBuilder** para generar automáticamente los comandos; para ello, basta con llamar al método **RefreshSchema** de **CommandBuilder**.

Si desea conocer el comando generado automáticamente, puede obtener una referencia a los comandos generados automáticamente mediante los métodos **GetInsertCommand**, **GetUpdateCommand** y **GetDeleteCommand** del objeto **CommandBuilder** y la comprobación de la propiedad **CommandText** del comando asociado.

En el ejemplo de código siguiente se escribe en la consola el comando de actualización generado automáticamente.

```
Console.WriteLine(builder.GetUpdateCommand().CommandText)
```

En el siguiente ejemplo se vuelve a crear la tabla Customers en el conjunto de datos custDS.Se llama al método **RefreshSchema** para actualizar los comandos generados automáticamente con la información de esta nueva columna.

```
' Assumes an open SqlConnection and SqlDataAdapter inside of a Using
block.
adapter.SelectCommand.CommandText = _
  "SELECT CustomerID, ContactName FROM dbo.Customers"
builder.RefreshSchema()
custDS.Tables.Remove(custDS.Tables("Customers"))
adapter.Fill(custDS, "Customers")
```

6.4.4. Obtener un Unico Valor de una Base de Datos

En ocasiones se debe devolver información de bases de datos consistente en un único valor, en lugar de una tabla o una secuencia de datos. Por ejemplo, puede que desee devolver el resultado de una función de agregada como COUNT(*), SUM(Price) o AVG(Quantity). El objeto **Command** permite devolver valores únicos mediante el método **ExecuteScalar**. El método **ExecuteScalar** devuelve como valor escalar el valor correspondiente a la primera columna de la primera fila del conjunto de resultados.

El ejemplo de código siguiente inserta un valor nuevo en la base de datos utilizando SqlCommand. El método ExecuteScalar se utiliza para devolver el valor de columna de identidad para el registro insertado.

```
Public Function AddProductCategory( _
  ByVal newName As String, ByVal connString As String) As Integer
    Dim newProdID As Int32 = 0
    Dim sql As String = _
     "INSERT INTO Production.ProductCategory (Name) VALUES (@Name); " _
```

```
        & "SELECT CAST(scope_identity() AS int);"
    Using conn As New SqlConnection(connString)
        Dim cmd As New SqlCommand(sql, conn)
        cmd.Parameters.Add("@Name", SqlDbType.VarChar)
        cmd.Parameters("@Name").Value = newName
        Try
            conn.Open()
            newProdID = Convert.ToInt32(cmd.ExecuteScalar())
        Catch ex As Exception
            Console.WriteLine(ex.Message)
        End Try
    End Using
    Return newProdID
End Function
```

6.4.5. Utilizar Comandos para Modificar Datos

Mediante un proveedor de datos de .NET Framework puede ejecutar procedimientos almacenados o instrucciones de lenguaje de definición de datos, como CREATE TABLE y ALTER COLUMN, para manipular los esquemas de una base de datos o catálogo. Estos comandos no devuelven filas de la forma en que lo haría una consulta y, en consecuencia, el objeto **Command** proporciona un método **ExecuteNonQuery** para procesarlos.

Además de utilizar **ExecuteNonQuery** para modificar esquemas, este método se puede usar también para procesar instrucciones SQL que modifican datos sin devolver ninguna fila, como INSERT, UPDATE y DELETE.

Aunque el método **ExecuteNonQuery** no devuelve ninguna fila, mediante la colección **Parameters** del objeto **Command** se pueden pasar y devolver parámetros de entrada y de salida, así como valores devueltos.

6.4.5.1. Actualizar Datos en un Origen de Datos

Las instrucciones SQL que modifican datos (por ejemplo INSERT, UPDATE o DELETE) no devuelven ninguna fila. De la misma forma, muchos procedimientos almacenados realizan alguna acción pero no devuelven filas. Para ejecutar comandos que no devuelvan filas, cree un objeto **Command** con el comando SQL adecuado y una **Connection**, incluidos los **Parameters** necesarios. El comando se debe ejecutar con el método **ExecuteNonQuery** del objeto **Command**.

El método **ExecuteNonQuery** devuelve un entero que representa el número de filas que se ven afectadas por la instrucción o por el procedimiento almacenado que se haya ejecutado. Si se ejecutan varias instrucciones, el valor devuelto es la suma de los registros afectados por todas las instrucciones ejecutadas.

⊟ Ejemplo

En el ejemplo de código siguiente se ejecuta una instrucción INSERT para insertar un registro en una base de datos mediante el método **ExecuteNonQuery**.

```
' Assumes connection is a valid SqlConnection.
connection.Open()
Dim queryString As String = "INSERT INTO Customers " & _
```

```
  "(CustomerID, CompanyName) Values('NWIND', 'Northwind Traders')"
Dim command As SqlCommand = New SqlCommand(queryString, connection)
Dim recordsAffected As Int32 = command.ExecuteNonQuery()
```

En el ejemplo de código siguiente se ejecuta el procedimiento almacenado que se creó con el ejemplo de código de la sección Realizar operaciones de catálogo. El procedimiento almacenado no devuelve ninguna fila, por lo que se utiliza el método **ExecuteNonQuery**, aunque el procedimiento almacenado reciba un parámetro de entrada y devuelva un parámetro de salida y un valor devuelto.

En el caso del objeto OleDbCommand, se debe agregar en primer lugar el parámetro **ReturnValue** a la colección **Parameters**.

```
' Assumes connection is a valid SqlConnection.
Dim command As SqlCommand = _
   New SqlCommand("InsertCategory" , connection)
command.CommandType = CommandType.StoredProcedure
Dim parameter As SqlParameter = _
 command.Parameters.Add("@RowCount", SqlDbType.Int)
parameter.Direction = ParameterDirection.ReturnValue
parameter = command.Parameters.Add( _
  "@CategoryName", SqlDbType.NChar, 15)
parameter = command.Parameters.Add("@Identity", SqlDbType.Int)
parameter.Direction = ParameterDirection.Output
command.Parameters("@CategoryName").Value = "New Category"
command.ExecuteNonQuery()
Dim categoryID As Int32 = CInt(command.Parameters("@Identity").Value)
Dim rowCount As Int32 = CInt(command.Parameters("@RowCount").Value)
```

6.4.5.2. Realizar Operaciones de Catálogo

Para ejecutar un comando que modifique una base de datos o un catálogo, por ejemplo una instrucción CREATE TABLE o CREATE PROCEDURE, debe crear un objeto **Command** mediante las instrucciones SQL adecuadas y un objeto **Connection**. El comando se debe ejecutar con el método **ExecuteNonQuery** del objeto **Command**.

En el ejemplo de código siguiente se crea un procedimiento almacenado en una base de datos de Microsoft SQL Server.

```
' Assumes connection is a valid SqlConnection.
Dim queryString As String = "CREATE PROCEDURE InsertCategory " & _
    "@CategoryName nchar(15), " & _
    "@Identity int OUT " & _
    "AS " & _
    "INSERT INTO Categories (CategoryName) VALUES(@CategoryName) " & _
    "SET @Identity = @@Identity " & _
    "RETURN @@ROWCOUNT"
Dim command As SqlCommand = New SqlCommand(queryString, connection)
command.ExecuteNonQuery()
```

6.5. DataReaders

Puede utilizar el **DataReader** de ADO.NET para recuperar secuencias de datos de sólo lectura y sólo avance de una base de datos. Los resultados se devuelven a medida que se ejecuta la consulta y se almacenan en el búfer de red del cliente hasta que se solicitan con el método **Read** del **DataReader**. Con el **DataReader** puede aumentar el rendimiento de la aplicación tanto al recuperar datos en cuanto están disponibles como al almacenar (de forma predeterminada) una fila cada vez en memoria, lo que reduce la sobrecarga del sistema.

Cada proveedor de datos de .NET Framework incluido en .NET Framework cuenta con un objeto **DataReader**: el proveedor de datos de .NET Framework para OLE DB incluye un objeto **OleDbDataReader**, el proveedor de datos de .NET Framework para SQL Server incluye un objeto **SqlDataReader**, el proveedor de datos de .NET Framework para ODBC incluye un objeto **OdbcDataReader** y el proveedor de datos de .NET Framework para Oracle incluye un objeto **OracleDataReader**.

6.5.1. Recuperar Datos Mediante DataReader

La recuperación de datos mediante **DataReader** implica crear una instancia del objeto **Command** y de un **DataReader** a continuación, para lo cual se llama a **Command.ExecuteReader** a fin de recuperar filas de un origen de datos. En el ejemplo siguiente se muestra cómo se utiliza un **DataReader**, donde reader representa un DataReader válido y command representa un objeto Command válido.

```
reader = command.ExecuteReader();
```

Puede utilizar el método **Read** del objeto **DataReader** para obtener una fila a partir de los resultados de una consulta. Para tener acceso a cada columna de la fila devuelta, puede pasar a **DataReader** el nombre o referencia numérica de la columna en cuestión. Sin embargo, el mejor rendimiento se logra con los métodos que ofrece **DataReader** y que permiten tener acceso a los valores de las columnas en sus tipos de datos nativos (**GetDateTime**, **GetDouble**, **GetGuid**, **GetInt32**, etc). Para obtener una lista de métodos de descriptor de acceso con tipo para **DataReaders** de proveedores de datos específicos, vea las secciones OleDbDataReader y SqlDataReader. Si se utilizan los métodos de descriptor de acceso con tipo, dando por supuesto que se conoce el tipo de datos subyacentes, se reduce el número de conversiones de tipo necesarias para recuperar el valor de una columna.

Nota:

En la versión de .NET Framework de Windows Server 2003 se incluye una propiedad adicional para el **DataReader**, **HasRows**, que permite determinar si el **DataReader** ha devuelto algún resultado antes de realizar una lectura del mismo.

En el ejemplo de código siguiente se repite por un objeto **DataReader** y se devuelven dos columnas de cada fila.

```
Private Sub HasRows(ByVal connection As SqlConnection)
    Using connection
        Dim command As SqlCommand = New SqlCommand( _
          "SELECT CategoryID, CategoryName FROM Categories;", _
          connection)
        connection.Open()
        Dim reader As SqlDataReader = command.ExecuteReader()
        If reader.HasRows Then
            Do While reader.Read()
                Console.WriteLine(reader.GetInt32(0) _
```

```
                    & vbTab & reader.GetString(1))
            Loop
        Else
            Console.WriteLine("No rows found.")
        End If
        reader.Close()
    End Using
End Sub
```

DataReader proporciona una secuencia de datos sin búfer que permite a la lógica de los procedimientos procesar eficazmente y de forma secuencial los resultados procedentes de un origen de datos. **DataReader** es la mejor opción cuando se trata de recuperar grandes cantidades de datos, ya que éstos no se almacenan en la memoria caché.

Cerrar el DataReader

Siempre debe llamar al método **Close** cuando haya terminado de utilizar el objeto **DataReader**.

Si **Command** contiene parámetros de salida o valores devueltos, éstos no estarán disponibles hasta que se cierre el **DataReader**.

Tenga en cuenta que mientras está abierto un **DataReader**, éste utiliza de forma exclusiva el objeto **Connection**. No se podrá ejecutar ningún comando para el objeto **Connection** hasta que se cierre el **DataReader** original, incluida la creación de otro **DataReader**.

Nota:

No llame a **Close** o **Dispose** para objetos **Connection** o **DataReader** ni para ningún otro objeto administrado en el método **Finalize** de su clase. En un finalizador, libere sólo los recursos no administrados que pertenezcan directamente a su clase. Si la clase no dispone de recursos no administrados, no incluya un método **Finalize** en la definición de clase.

Recuperar varios conjuntos de resultados con NextResult

En el caso en que se devuelvan varios resultados, el **DataReader** proporciona el método **NextResult** para recorrer los conjuntos de resultados en orden. En el siguiente ejemplo se muestra el SqlDataReader mientras procesa los resultados de las dos instrucciones SELECT mediante el método ExecuteReader.

```
Private Sub RetrieveMultipleResults(ByVal connection As SqlConnection)
    Using connection
        Dim command As SqlCommand = New SqlCommand( _
          "SELECT CategoryID, CategoryName FROM Categories;" & _
          "SELECT EmployeeID, LastName FROM Employees", connection)
        connection.Open()
        Dim reader As SqlDataReader = command.ExecuteReader()
        Do While reader.HasRows
            Console.WriteLine(vbTab & reader.GetName(0) _
              & vbTab & reader.GetName(1))
            Do While reader.Read()
                Console.WriteLine(vbTab & reader.GetInt32(0) _
                  & vbTab & reader.GetString(1))
            Loop
            reader.NextResult()
        Loop
    End Using
```

```
End Sub
```

Obtener información del esquema a partir del DataReader

Mientras hay abierto un **DataReader**, puede utilizar el método **GetSchemaTable** para recuperar información del esquema acerca del conjunto actual de resultados. **GetSchemaTable** devuelve un objeto DataTable rellenado con filas y columnas que contienen la información del esquema del conjunto actual de resultados. **DataTable** contiene una fila por cada una de las columnas del conjunto de resultados. Cada columna de una fila de la tabla de esquema está asociada a una propiedad de la columna que se devuelve en el conjunto de resultados. **ColumnName** es el nombre de la propiedad y el valor de la columna es el de la propiedad. En el ejemplo de código siguiente se muestra la información del esquema de **DataReader**.

```
Private Sub GetSchemaInfo(ByVal connection As SqlConnection)
    Using connection
        Dim command As SqlCommand = New SqlCommand( _
          "SELECT CategoryID, CategoryName FROM Categories;", _
          connection)
        connection.Open()
        Dim reader As SqlDataReader = command.ExecuteReader()
        Dim schemaTable As DataTable = reader.GetSchemaTable()
        Dim row As DataRow
        Dim column As DataColumn
        For Each row In schemaTable.Rows
            For Each column In schemaTable.Columns
                Console.WriteLine(String.Format("{0} = {1}", _
                  column.ColumnName, row(column)))
            Next
            Console.WriteLine()
        Next
        reader.Close()
    End Using
End Sub
```

Trabajar con capítulos de OLE DB

Mediante OleDbDataReader puede recuperar conjuntos jerárquicos de filas o capítulos (tipo **DBTYPE_HCHAPTER** de OLE DB y tipo **adChapter** de ADO) . Cuando se devuelve en forma de **DataReader** una consulta que incluye un capítulo, éste se devuelve como una columna del **DataReader** y se expone como un objeto **DataReader**.

También se puede utilizar **DataSet** de ADO.NET para representar conjuntos jerárquicos de filas utilizando relaciones entre tablas primarias y secundarias.

En el ejemplo de código siguiente se utiliza el proveedor MSDataShape para generar un capítulo con la columna de pedidos realizados por cada uno de los clientes de una lista.

```
Using connection As OleDbConnection = New OleDbConnection( _
  "Provider=MSDataShape;Data Provider=SQLOLEDB;" & _
  "Data Source=localhost;Integrated Security=SSPI;Initial
Catalog=northwind")
Dim custCMD As OleDbCommand = New OleDbCommand( _
  "SHAPE {SELECT CustomerID, CompanyName FROM Customers} " & _
"APPEND ({SELECT CustomerID, OrderID FROM Orders} AS CustomerOrders " & _
```

```
  "RELATE CustomerID TO CustomerID)", connection)
connection.Open()
Dim custReader As OleDbDataReader = custCMD.ExecuteReader()
Dim orderReader As OleDbDataReader
Do While custReader.Read()
  Console.WriteLine("Orders for " & custReader.GetString(1))
  ' custReader.GetString(1) = CompanyName
  orderReader = custReader.GetValue(2)
  ' custReader.GetValue(2) = Orders chapter as DataReader
  Do While orderReader.Read()
    Console.WriteLine(vbTab & orderReader.GetInt32(1))
    ' orderReader.GetInt32(1) = OrderID
  Loop
  orderReader.Close()
Loop
' Make sure to always close readers and connections.
custReader.Close()
End Using
```

Devolver resultados con cursores REF CURSOR de Oracle

El proveedor de datos de .NET Framework para Oracle admite el uso de cursores REF CURSOR de Oracle para devolver los resultados de una consulta. Un REF CURSOR de Oracle se devuelve en forma de objeto OracleDataReader.

Puede recuperar un objeto **OracleDataReader**, que representa un REF CURSOR de Oracle, mediante el método ExecuteReader. También puede especificar un OracleCommand que devuelva uno o varios cursores REF CURSOR de Oracle como **SelectCommand** de un OracleDataAdapter utilizado para rellenar un DataSet.

Para obtener acceso a un REF CURSOR devuelto desde un origen de datos de Oracle, cree un **OracleCommand** para la consulta y agregue un parámetro de salida que establezca una referencia entre el REF CURSOR y la colección **Parameters** de **OracleCommand**. El nombre del parámetro debe coincidir con el nombre del parámetro REF CURSOR de la consulta. Establezca el tipo del parámetro en **OracleType.Cursor**. El método **ExecuteReader** del **OracleCommand** devolverá un **OracleDataReader** para el REF CURSOR.

Si **OracleCommand** devuelve varios cursores REF CURSOR, agregue varios parámetros de salida. Puede tener acceso a los distintos cursores REF CURSOR llamando al método **OracleCommand.ExecuteReader**. La llamada a **ExecuteReader** devuelve un objeto **OracleDataReader** que haga referencia al primer REF CURSOR. A continuación, puede llamar al método **OracleDataReader.NextResult** para obtener acceso a los cursores REF CURSOR posteriores. Aunque los parámetros de la colección **OracleCommand.Parameters** tengan el mismo nombre que los parámetros de salida de REF CURSOR, **OracleDataReader** obtendrá acceso a éstos en el mismo orden en el que se agregaron a la colección **Parameters**.

Por ejemplo, considere el siguiente paquete de Oracle y, concretamente, el cuerpo del paquete.

```
CREATE OR REPLACE PACKAGE CURSPKG AS
  TYPE T_CURSOR IS REF CURSOR;
  PROCEDURE OPEN_TWO_CURSORS (EMPCURSOR OUT T_CURSOR,
    DEPTCURSOR OUT T_CURSOR);
```

```
END CURSPKG;
CREATE OR REPLACE PACKAGE BODY CURSPKG AS
  PROCEDURE OPEN_TWO_CURSORS (EMPCURSOR OUT T_CURSOR,
    DEPTCURSOR OUT T_CURSOR)
  IS
  BEGIN
    OPEN EMPCURSOR FOR SELECT * FROM DEMO.EMPLOYEE;
    OPEN DEPTCURSOR FOR SELECT * FROM DEMO.DEPARTMENT;
  END OPEN_TWO_CURSORS;
END CURSPKG;
```

En el código siguiente se crea un **OracleCommand** que devuelve los cursores REF CURSOR del paquete anterior de Oracle mediante la adición de dos parámetros de tipo **OracleType.Cursor** a la colección **Parameters**.

```
Dim cursCmd As OracleCommand = New
OracleCommand("CURSPKG.OPEN_TWO_CURSORS", oraConn)
cursCmd.Parameters.Add("EMPCURSOR", OracleType.Cursor).Direction =
ParameterDirection.Output
cursCmd.Parameters.Add("DEPTCURSOR", OracleType.Cursor).Direction =
ParameterDirection.Output
```

El código siguiente devuelve los resultados del comando anterior utilizando los métodos **Read** y **NextResult** del **OracleDataReader**. Los parámetros REF CURSOR se devuelven en orden.

```
oraConn.Open()
Dim cursCmd As OracleCommand = New
OracleCommand("CURSPKG.OPEN_TWO_CURSORS", oraConn)
cursCmd.CommandType = CommandType.StoredProcedure
cursCmd.Parameters.Add("EMPCURSOR", OracleType.Cursor).Direction =
ParameterDirection.Output
cursCmd.Parameters.Add("DEPTCURSOR", OracleType.Cursor).Direction =
ParameterDirection.Output
Dim reader As OracleDataReader = cursCmd.ExecuteReader()
Console.WriteLine(vbCrLf & "Emp ID" & vbTab & "Name")
Do While reader.Read()
  Console.WriteLine("{0}" & vbTab & "{1}, {2}",
reader.GetOracleNumber(0), reader.GetString(1), reader.GetString(2))
Loop
reader.NextResult()
Console.WriteLine(vbCrLf & "Dept ID" & vbTab & "Name")
Do While reader.Read()
  Console.WriteLine("{0}" & vbTab & "{1}", reader.GetOracleNumber(0),
reader.GetString(1))
Loop
' Make sure to always close readers and connections.
reader.Close()
oraConn.Close()
```

En el siguiente ejemplo se utiliza el comando anterior para rellenar un **DataSet** con los resultados del paquete de Oracle.

Nota:

Se recomienda que el usuario controle también cualquier conversión del tipo NUMBER de Oracle a un tipo válido de .NET Framework antes de almacenar el valor en **DataRow** para evitar que se produzca una excepción **OverflowException**. Puede utilizar el evento **FillError** para determinar si se ha producido una excepción **OverflowException**.

```
Dim ds As DataSet = New DataSet()
Dim adapter As OracleDataAdapter = New OracleDataAdapter(cursCmd)
adapter.TableMappings.Add("Table", "Employees")
adapter.TableMappings.Add("Table1", "Departments")
adapter.Fill(ds)
```

6.5.2. Recuperar Datos Binarios

El objeto **DataReader** carga de forma predeterminada los datos que recibe en una fila, siempre que hay disponibles suficientes datos para llenarla. Sin embargo, los objetos binarios grandes (BLOB) se deben tratar de otra forma, ya que pueden llegar a contener grandes cantidades de datos (del orden de gigabytes) que no pueden almacenarse en una sola fila. El método **Command.ExecuteReader** se puede sobrecargar para aceptar un argumento CommandBehavior y modificar el comportamiento predeterminado de **DataReader**. Puede pasar SequentialAccess al método **ExecuteReader** para modificar el comportamiento predeterminado de **DataReader** de forma que en lugar de cargar los datos por filas, los vaya cargando secuencialmente a medida que los vaya recibiendo. Este sistema es idóneo para cargar BLOB y otras estructuras de datos grandes. Tenga en cuenta que este comportamiento puede depender del origen de datos. Por ejemplo, si se devuelve un BLOB desde Microsoft Access, el BLOB completo se cargará en memoria, en lugar de hacerlo secuencialmente a medida que se recibe.

Al configurar **DataReader** para que utilice **SequentialAccess** debe tener en cuenta la secuencia en que va a tener acceso a los datos devueltos. El comportamiento predeterminado de **DataReader**, consistente en cargar una fila completa de datos en cuanto hay datos suficientes, permite tener acceso a ellos en cualquier orden hasta que se lee la fila siguiente. Sin embargo, al utilizar **SequentialAccess** es necesario tener acceso a los campos que devuelve **DataReader** en el orden adecuado. Por ejemplo, si la consulta devuelve tres columnas y la tercera es un BLOB, debe devolver los valores de los campos primero y segundo antes de tener acceso a los datos BLOB del tercer campo. Si trata de tener acceso al tercer campo antes que al primero o el segundo, puede que éstos dejen de estar disponibles. Esto se debe a que **SequentialAccess** cambia la forma en que el **DataReader** devuelve los datos, haciendo que lo haga de forma secuencial con lo que los datos dejan de estar disponibles en el momento en que el **DataReader** lee datos posteriores.

Cuando intente obtener acceso a los datos del campo BLOB, utilice los descriptores de acceso con información de tipos **GetBytes** o **GetChars** del **DataReader**, que llenan una matriz con los datos. En el caso de los datos de caracteres, puede utilizar también **GetString**; no obstante, si desea conservar los recursos del sistema, es mejor que no cargue un valor BLOB completo en una sola variable de cadena. En lugar de ello, puede especificar un tamaño determinado de búfer para los datos que se van a devolver, así como la ubicación de comienzo para leer el primer byte o carácter de los datos devueltos. **GetBytes** y **GetChars** devuelven un valor de tipo **long** que representa el número de bytes o caracteres devueltos. Si pasa una matriz con valores null a **GetBytes** o **GetChars**, el valor de tipo long devuelto contiene el número total de bytes o caracteres del BLOB. También puede especificar un índice de la matriz como posición de comienzo para la lectura de datos.

⊟ Ejemplo

En el siguiente ejemplo se devuelve el identificador y el logotipo del editor desde la base de datos de ejemplo **pubs** de Microsoft SQL Server 2000. El identificador de editor (pub_id) es un campo de caracteres, mientras que el logotipo es una imagen de BLOB. Como el campo **logo** es un mapa de bits, el ejemplo devuelve datos binarios mediante **GetBytes**. Tenga en cuenta que la necesidad de tener acceso a los datos de forma secuencial hace que en la fila actual de datos se tenga acceso al identificador de editor antes que al logotipo.

```
' Assumes that connection is a valid SqlConnection object.
Dim command As SqlCommand = New SqlCommand( _
  "SELECT pub_id, logo FROM pub_info", connection)
' Writes the BLOB to a file (*.bmp).
Dim stream As FileStream
' Streams the binary data to the FileStream object.
Dim writer As BinaryWriter
' The size of the BLOB buffer.
Dim bufferSize As Integer = 100
' The BLOB byte() buffer to be filled by GetBytes.
Dim outByte(bufferSize - 1) As Byte
' The bytes returned from GetBytes.
Dim retval As Long
' The starting position in the BLOB output.
Dim startIndex As Long = 0
' The publisher id to use in the file name.
Dim pubID As String = ""
' Open the connection and read data into the DataReader.
connection.Open()
Dim reader As SqlDataReader =
command.ExecuteReader(CommandBehavior.SequentialAccess)
Do While reader.Read()
  ' Get the publisher id, which must occur before getting the logo.
  pubID = reader.GetString(0)
  ' Create a file to hold the output.
  stream = New FileStream( _
    "logo" & pubID & ".bmp", FileMode.OpenOrCreate, FileAccess.Write)
  writer = New BinaryWriter(stream)
  ' Reset the starting byte for a new BLOB.
  startIndex = 0
  ' Read bytes into outByte() and retain the number of bytes returned.
  retval = reader.GetBytes(1, startIndex, outByte, 0, bufferSize)
  ' Continue while there are bytes beyond the size of the buffer.
  Do While retval = bufferSize
    writer.Write(outByte)
    writer.Flush()
    ' Reposition start index to end of the last buffer and fill buffer.
    startIndex += bufferSize
    retval = reader.GetBytes(1, startIndex, outByte, 0, bufferSize)
  Loop
  ' Write the remaining buffer.
  writer.Write(outByte, 0 , retval - 1)
  writer.Flush()
```

```
  ' Close the output file.
  writer.Close()
  stream.Close()
Loop
' Close the reader and the connection.
reader.Close()
connection.Close()
```

6.6. DataAdapters

Un **DataAdapter** se utiliza para recuperar datos de un origen de datos y llenar tablas con un **DataSet**. **DataAdapter** también resuelve los cambios realizados en **DataSet** de vuelta al origen de datos. Mediante el objeto **Connection** del proveedor de datos de .NET Framework, **Connection** se conecta a un origen de datos y utiliza objetos **Command** para recuperar datos del origen de datos y resolver los cambios a dicho origen.

Cada proveedor de datos de .NET Framework incluye un objeto **DataAdapter**: el proveedor de datos de .NET Framework para OLE DB incluye un objeto **OleDbDataAdapter**, el proveedor de datos de .NET Framework para SQL Server incluye un objeto **SqlDataAdapter**, el proveedor de datos de .NET Framework para ODBC incluye un objeto **OdbcDataAdapter** y el proveedor de datos de .NET Framework para Oracle incluye un objeto **OracleDataAdapter**.

6.6.1. Rellenar un Objeto DataSet desde un Objeto DataAdapter

DataSet de ADO.NET es una representación residente en memoria de datos que proporciona un modelo de programación relacional coherente e independiente del origen de datos. **DataSet** representa un conjunto completo de datos que incluye tablas, restricciones y relaciones entre las tablas. Dado que **DataSet** es independiente del origen de datos, **DataSet** puede incluir datos locales de la aplicación y datos de otros muchos orígenes. La interacción con los orígenes de datos existentes se controla mediante el **DataAdapter**.

La propiedad **SelectCommand** de **DataAdapter** es un objeto **Command** que recupera datos del origen de datos. Las propiedades **InsertCommand**, **UpdateCommand** y **DeleteCommand** de **DataAdapter** son objetos **Command** que permiten administrar las actualizaciones de los datos en el origen de datos para reflejar las modificaciones efectuadas en los datos de **DataSet**.

El método **Fill** de **DataAdapter** se usa para rellenar un objeto **DataSet** con los resultados del **SelectCommand** de **DataAdapter**. **Fill** toma como argumentos un **DataSet** que se debe rellenar y un objeto **DataTable**, o su nombre, que se debe rellenar con las filas que devuelve **SelectCommand**.

Nota:

El uso de **DataAdapter** para recuperar la totalidad de una tabla lleva tiempo, en especial si la tabla incluye un gran número de filas. Esto se debe a que el acceso a la base de datos, la localización y el procesamiento de los datos, y la posterior transferencia de los mismos al cliente son procesos largos. La extracción de la tabla completa al cliente también bloquea todas las filas en el servidor. Para mejorar el rendimiento, puede usar la cláusula **WHERE** para reducir en gran medida el número de filas que se devuelven al cliente. También puede reducir la cantidad de datos que se devuelven al

cliente si enumera de forma explícita las columnas necesarias en la instrucción **SELECT**. Otra solución consiste en recuperar las filas por lotes (por ejemplo varios cientos de filas de una vez) y recuperar solo el siguiente lote cuando el cliente haya finalizado con el lote actual.

El método **Fill** utiliza el objeto **DataReader** de forma implícita para devolver los nombres y tipos de columna que se usan para crear las tablas de **DataSet**, y los datos para rellenar las filas de las tablas en **DataSet**. Las tablas y columnas sólo se crean cuando no existen; en caso contrario, **Fill** utiliza el esquema existente de **DataSet**. Los tipos de columna se crean como tipos de .NET Framework de acuerdo con las tablas que figuran en Asignar tipos de datos (ADO.NET). No se crean claves principales a menos que existan en el origen de datos y **DataAdapter.MissingSchemaAction** se establezca en **MissingSchemaAction.AddWithKey**. Si el método **Fill** encuentra que una tabla tiene una clave principal, sobrescribe los datos de **DataSet** con los del origen de datos en las filas donde los valores de columna de clave principal coinciden con los de la fila que devuelve el origen de datos. Si no se detecta ninguna clave principal, los datos se agregan a las tablas de **DataSet**. **Fill** utiliza las asignaciones que puedan existir cuando se rellene **DataSet**.

Nota:

Si **SelectCommand** devuelve los resultados de OUTER JOIN, **DataAdapter** no establece un valor **PrimaryKey** para el objeto **DataTable** resultante. Debe definir **PrimaryKey** para asegurarse de que las filas duplicadas se resuelven correctamente. Para obtener más información, vea Definir claves principales (ADO.NET).

En el ejemplo de código siguiente se crea una instancia de SqlDataAdapter que utiliza un objeto SqlConnection a la base de datos **Northwind** de Microsoft SQL Server y se rellena un objeto DataTable en un **DataSet** con la lista de clientes. La instrucción SQL y los argumentos SqlConnection pasados al constructor SqlDataAdapter se utilizan para crear la propiedad SelectCommand del SqlDataAdapter.

Ejemplo

```
' Assumes that connection is a valid SqlConnection object.
Dim queryString As String = _
  "SELECT CustomerID, CompanyName FROM dbo.Customers"
Dim adapter As SqlDataAdapter = New SqlDataAdapter( _
  queryString, connection)
Dim customers As DataSet = New DataSet
adapter.Fill(customers, "Customers")
```

Nota:

El código que se muestra en este ejemplo no abre ni cierra explícitamente el objeto **Connection**. El método **Fill** abre de forma implícita el objeto **Connection** que **DataAdapter** utiliza cuando encuentra que la conexión no está abierta todavía. Si el método **Fill** ha abierto la conexión, también la cierra cuando termina de utilizarla. Este hecho simplifica el código cuando se trabaja con una operación única, como **Fill** o **Update**. Sin embargo, en el caso de que se estén realizando varias operaciones que necesiten tener abierta una conexión, se puede mejorar el rendimiento de la aplicación llamando explícitamente al método **Open** de **Connection**, realizando las operaciones en el origen de datos y, finalmente, llamando al método **Close** de **Connection**. Es conveniente mantener abiertas las conexiones con el origen de datos el menor tiempo posible para liberar recursos, de manera que estén disponibles para otras aplicaciones cliente.

Varios conjuntos de resultados

Si **DataAdapter** encuentra varios conjuntos de resultados, crea varias tablas en **DataSet**. Las tablas reciben de forma predeterminada el nombre secuencial Table*N*, comenzando por "Table" que representa Table0. Si se pasa un nombre de tabla como argumento al método **Fill**, las tablas reciben de forma

predeterminada el nombre secuencial TableName*N*, comenzando por "TableName" que representa TableName0.

⊟ Llenar un DataSet desde múltiples DataAdapter

Se puede usar cualquier número de objetos **DataAdapter** con **DataSet**. Cada **DataAdapter** se puede usar para rellenar uno o varios objetos **DataTable** y resolver de nuevo las actualizaciones en el origen de datos correspondiente. Se pueden agregar objetos **DataRelation** y **Constraint** a **DataSet** localmente, lo que permite relacionar datos procedentes de varios orígenes distintos. Por ejemplo, un **DataSet** puede contener datos de una base de datos de Microsoft SQL Server, una base de datos de IBM DB2 expuesta mediante OLE DB y un origen de datos que genera secuencias XML. La comunicación con cada origen de datos se puede controlar usando uno o varios objetos **DataAdapter**.

Ejemplo

En el ejemplo de código siguiente se rellena una lista de clientes a partir de la base de datos **Northwind** almacenada en Microsoft SQL Server 2000, y una lista de pedidos a partir de la base de datos **Northwind** almacenada en Microsoft Access 2000. Las tablas rellenas se relacionan entre sí mediante **DataRelation**, con lo que se puede mostrar una lista de clientes con los pedidos que ha realizado cada uno.

```
' Assumes that customerConnection is a valid SqlConnection object.
' Assumes that orderConnection is a valid OleDbConnection object.
Dim custAdapter As SqlDataAdapter = New SqlDataAdapter( _
  "SELECT * FROM dbo.Customers", customerConnection)
Dim ordAdapter As OleDbDataAdapter = New OleDbDataAdapter( _
  "SELECT * FROM Orders", orderConnection)
Dim customerOrders As DataSet = New DataSet()
custAdapter.Fill(customerOrders, "Customers")
ordAdapter.Fill(customerOrders, "Orders")
Dim relation As DataRelation = _
  customerOrders.Relations.Add("CustOrders", _
  customerOrders.Tables("Customers").Columns("CustomerID"), _
  customerOrders.Tables("Orders").Columns("CustomerID"))
Dim pRow, cRow As DataRow
For Each pRow In customerOrders.Tables("Customers").Rows
  Console.WriteLine(pRow("CustomerID").ToString())
  For Each cRow In pRow.GetChildRows(relation)
    Console.WriteLine(vbTab & cRow("OrderID").ToString())
  Next
Next
```

⊟ Tipo decimal de SQL Server

De forma predeterminada, **DataSet** almacena datos utilizando tipos de datos de .NET Framework. En la mayor parte de las aplicaciones, estos tipos proporcionan una representación adecuada de la información del origen de datos. Sin embargo, esa representación puede ocasionar problemas cuando el tipo de datos del origen de datos es decimal o numérico de SQL Server. El tipo de datos **decimal** de .NET Framework permite un máximo de 28 dígitos significativos, mientras que el tipo de datos **decimal** de SQL Server permite 38 dígitos. Si **SqlDataAdapter** determina durante una operación **Fill** que la precisión de un campo **decimal** de SQL Server es superior a 28 caracteres, la fila actual no se agrega a **DataTable**. En su lugar, se produce el evento **FillError** que permite determinar si se va a producir o no una pérdida de precisión y tomar las medidas adecuadas. Para obtener el valor **decimal** de SQL Server, también se puede utilizar un objeto SqlDataReader y llamar al método GetSqlDecimal.

ADO.NET 2.0 incorpora compatibilidad mejorada para System.Data.SqlTypes en **DataSet**.

⊟ Capítulos de OLE DB

Se pueden usar conjuntos jerárquicos de filas, o capítulos (tipo **DBTYPE_HCHAPTER** de OLE DB y tipo **adChapter** de ADO), para rellenar el contenido de **DataSet**. Cuando OleDbDataAdapter encuentra una columna que tiene un capítulo durante una operación **Fill**, se crea un objeto **DataTable** para dicha columna y la tabla se rellena con las columnas y filas del capítulo. Para asignar un nombre a la tabla creada para la columna con capítulo se usa tanto el nombre de la tabla primaria como el de la columna con capítulo. El formato del nombre es "nombreDeTablaPrimariaNombreDeColumnaConCapítulo". Si ya existe una tabla en **DataSet** que tenga el nombre de la columna con capítulo, la tabla actual se rellena con los datos del capítulo. Si ninguna de las columnas de la tabla existente coincide con una de las columnas del capítulo, se agrega una nueva columna a la tabla.

Antes de que las tablas de **DataSet** se rellenen con los datos de las columnas con capítulos, se crea una relación entre las tablas primaria y secundaria del conjunto jerárquico de filas; para ello, se agrega una columna de tipo entero a las tablas primaria y secundaria, se establece el valor de incremento automático para la columna de la tabla primaria y se crea un objeto **DataRelation** usando las columnas agregadas de ambas tablas. Para asignar un nombre a la relación se utilizan los nombres de la tabla primaria y de la columna con capítulo. El formato es "nombreDeTablaPrimariaNombreDeColumnaConCapítulo".

Tenga en cuenta que la columna relacionada sólo existe en **DataSet**. Las operaciones de relleno que se realicen posteriormente desde el origen de datos pueden provocar que se agreguen nuevas filas a las tablas en lugar de que se introduzcan los cambios en las filas existentes.

Tenga en cuenta además que, si se utiliza una sobrecarga de **DataAdapter.Fill** que acepte un objeto **DataTable**, solo se rellanará esa tabla. En este caso también se agrega a la tabla una columna de tipo entero y con incremento automático, aunque no se crea ni rellena ninguna tabla secundaria, ni se crea ninguna relación.

En el ejemplo siguiente se utiliza el proveedor MSDataShape para generar un capítulo con la columna de pedidos realizados por cada uno de los clientes de una lista. A continuación, se rellena un **DataSet** con los datos.

```
Using connection As OleDbConnection = New OleDbConnection( _
  "Provider=MSDataShape;Data Provider=SQLOLEDB;" & _
  "Data Source=(local);Integrated " & _
  "Security=SSPI;Initial Catalog=northwind")
Dim adapter As OleDbDataAdapter = New OleDbDataAdapter( _
  "SHAPE {SELECT CustomerID, CompanyName FROM Customers} " & _
  "APPEND ({SELECT CustomerID, OrderID FROM Orders} AS Orders " & _
  "RELATE CustomerID TO CustomerID)", connection)
Dim customers As DataSet = New DataSet()
adapter.Fill(customers, "Customers")
End Using
```

Una vez completada la operación **Fill**, **DataSet** contiene dos tablas: **Customers** y **CustomersOrders**, donde **CustomersOrders** representa la columna con capítulo. Se agrega una columna adicional denominada **Orders** a la tabla **Customers**, y una columna adicional denominada CustomersOrders a la tabla **CustomersOrders**. Se establece el valor de incremento automático para la columna **Orders** de la tabla **Customers**. Se crea también una relación **DataRelation**, **CustomersOrders**, utilizando las

columnas que se han agregado a las tablas, siendo **Customers** la tabla primaria. Las siguientes tablas muestran algunos ejemplos de los resultados.

Nombre de tabla: Customers

CustomerID	CompanyName	Orders
ALFKI	Alfreds Futterkiste	0
ANATR	Ana Trujillo Emparedados y helados	1

Nombre de tabla: CustomersOrders

CustomerID	OrderID	CustomersOrders
ALFKI	10643	0
ALFKI	10692	0
ANATR	10308	1
ANATR	10625	1

6.6.2. Parámetros DataAdapter

DbDataAdapter tiene cuatro propiedades que se utilizan para recuperar y actualizar datos en el origen de datos: la propiedad SelectCommand devuelve datos del origen de datos y las propiedades InsertCommand , UpdateCommand y DeleteCommand se utilizan para administrar los cambios en el origen de datos. La propiedad **SelectCommand** debe establecerse antes de llamar al método **Fill** de **DataAdapter**. Las propiedades **InsertCommand**, **UpdateCommand** o **DeleteCommand** se deben establecer antes llamar al método **Update** de **DataAdapter**, en función de las modificaciones realizadas en los datos en DataTable. Por ejemplo, si se han agregado filas, se debe establecer **InsertCommand** antes de llamar a **Update**. Cuando **Update** procesa una fila insertada, actualizada o eliminada, **DataAdapter** utiliza la propiedad **Command** que corresponde a la acción en cuestión. La información actual relacionada con la fila modificada se pasa al objeto **Command** a través de la colección **Parameters**.

Cuando actualice una fila en el origen de datos, llame a la instrucción UPDATE que utiliza un identificador único para identificar la fila de la tabla que debe actualizarse. El identificador único suele ser el valor del campo de clave principal. La instrucción UPDATE utiliza parámetros que contienen el identificador único y las columnas y valores que se van a actualizar, como muestra la siguiente instrucción Transact-SQL.

```
UPDATE Customers SET CompanyName = @CompanyName
  WHERE CustomerID = @CustomerID
```

Nota:

La sintaxis de los marcadores de posición de parámetros depende del origen de datos. En este ejemplo se muestran marcadores de posición para un origen de datos de SQL Server. Utilice signos de interrogación de cierre (?) como marcadores de posición de para los parámetros System.Data.OleDb y System.Data.Odbc.

En este ejemplo de Visual Basic, el campo **CompanyName** se actualiza con el valor del parámetro @CompanyName para la fila cuyo **CustomerID** coincida con el valor del parámetro @CustomerID. Los parámetros recuperan información de la fila modificada mediante la propiedad SourceColumn del objeto

SqlParameter. A continuación se muestran los parámetros del ejemplo anterior de la instrucción UPDATE. En el código se parte de que el *adapter* de la variable representa a un objeto SqlDataAdapter válido.

```
adapter.Parameters.Add( _
  "@CompanyName", SqlDbType.NChar, 15, "CompanyName")
Dim parameter As SqlParameter = _
  adapter.UpdateCommand.Parameters.Add("@CustomerID", _
  SqlDbType.NChar, 5, "CustomerID")
parameter.SourceVersion = DataRowVersion.Original
```

El método **Add** de la colección **Parameters** toma el nombre del parámetro, el tipo de datos, el tamaño (si corresponde al tipo) y el nombre de la propiedad SourceColumn de **DataTable**. Tenga en cuenta que SourceVersion del parámetro @CustomerID se establece en **Original**. De esta forma se garantiza que la fila existente en el origen de datos se actualice cuando el valor de la columna o columnas identificadas haya cambiado en la fila DataRow modificada. En ese caso, el valor de la fila **Original** coincidiría con el valor actual en el origen de datos y el valor de la fila **Current** contendría el valor actualizado. No se asigna ningún valor a **SourceVersion** para el parámetro @CompanyName, por lo que se utiliza el valor predeterminado, el de la fila **Current**.

Nota:

En las operaciones **Fill** de **DataAdapter** y los métodos **Get** de **DataReader**, el tipo .NET Framework se deduce del tipo devuelto desde el proveedor de datos de .NET Framework. Los métodos de descriptor de acceso y los tipos de .NET Framework deducidos para tipos de datos de Microsoft SQL Server, OLE DB y ODBC se describen en Asignar tipos de datos (ADO.NET).

Parameter.SourceColumn, Parameter.SourceVersion

SourceColumn y **SourceVersion** se pueden pasar como argumentos al constructor **Parameter**, o también se pueden establecer como propiedades de un **Parameter** existente. **SourceColumn** es el nombre de DataColumn de DataRow en la que se recupera el valor de **Parameter** . **SourceVersion** especifica la versión de **DataRow** que utiliza **DataAdapter** para recuperar el valor.

En la tabla siguiente se muestran los valores de la enumeración DataRowVersion disponibles para su uso con **SourceVersion**.

Enumeración DataRowVersion	Descripción
Current	El parámetro utiliza el valor actual de la columna. Éste es el valor predeterminado.
Default	El parámetro utiliza el **DefaultValue** de la columna.
Original	El parámetro utiliza el valor original de la columna.
Proposed	El parámetro utiliza un valor propuesto.

En el ejemplo de código de **SqlClient** de la siguiente sección se define un parámetro para UpdateCommand donde la columna **CustomerID** se utiliza como **SourceColumn** para dos parámetros: *@CustomerID* (SET CustomerID = @CustomerID) y *@OldCustomerID* (WHERE CustomerID = @OldCustomerID). El parámetro *@CustomerID* se utiliza para actualizar la columna **CustomerID** de forma que tenga el valor actual de **DataRow**. Como resultado, se usa **CustomerID SourceColumn** con **SourceVersion** de **Current**. El parámetro *@OldCustomerID* se utiliza para identificar la fila actual en el origen de datos. Dado que el valor de la columna coincidente se encuentra en la versión **Original** de la fila, también se usa el mismo objeto **SourceColumn** (**CustomerID**) con **SourceVersion** de **Original**.

Trabajar con parámetros SqlClient

En el ejemplo siguiente se muestra cómo crear SqlDataAdapter y establecer MissingSchemaAction en AddWithKey para recuperar información de esquema adicional de la base de datos. Las propiedades SelectCommand, InsertCommand, UpdateCommand y DeleteCommand establecen sus correspondientes objetos SqlParameter agregados a la colección Parameters . El método devuelve un objeto **SqlDataAdapter**.

```
Public Function CreateSqlDataAdapter( _
    ByVal connection As SqlConnection) As SqlDataAdapter
    Dim adapter As SqlDataAdapter = New SqlDataAdapter
    adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey
    ' Create the commands.
    adapter.SelectCommand = New SqlCommand( _
        "SELECT CustomerID, CompanyName FROM CUSTOMERS", connection)
    adapter.InsertCommand = New SqlCommand( _
        "INSERT INTO Customers (CustomerID, CompanyName) " & _
         "VALUES (@CustomerID, @CompanyName)", connection)
    adapter.UpdateCommand = New SqlCommand( _
      "UPDATE Customers SET CustomerID = @CustomerID, CompanyName = " & _
        "@CompanyName WHERE CustomerID = @oldCustomerID", connection)
    adapter.DeleteCommand = New SqlCommand( _
     "DELETE FROM Customers WHERE CustomerID = @CustomerID", connection)
    ' Create the parameters.
    adapter.InsertCommand.Parameters.Add("@CustomerID", _
        SqlDbType.Char, 5, "CustomerID")
    adapter.InsertCommand.Parameters.Add("@CompanyName", _
        SqlDbType.VarChar, 40, "CompanyName")
    adapter.UpdateCommand.Parameters.Add("@CustomerID", _
        SqlDbType.Char, 5, "CustomerID")
    adapter.UpdateCommand.Parameters.Add("@CompanyName", _
        SqlDbType.VarChar, 40, "CompanyName")
    adapter.UpdateCommand.Parameters.Add("@oldCustomerID", _
        SqlDbType.Char, 5, "CustomerID").SourceVersion = _
        DataRowVersion.Original
    adapter.DeleteCommand.Parameters.Add("@CustomerID", _
        SqlDbType.Char, 5, "CustomerID").SourceVersion = _
        DataRowVersion.Original
    Return adapter
End Function
```

Marcadores de posición de parámetros OleDb

En el caso de los objetos OleDbDataAdapter y OdbcDataAdapter, debe utilizar signos de interrogación de cierre (?) como marcadores de posición para identificar los parámetros.

```
Dim selectSQL As String = _
  "SELECT CustomerID, CompanyName FROM Customers " & _
  "WHERE CountryRegion = ? AND City = ?"
Dim insertSQL AS String = _
  "INSERT INTO Customers (CustomerID, CompanyName) VALUES (?, ?)"
Dim updateSQL AS String = _
  "UPDATE Customers SET CustomerID = ?, CompanyName = ? " & _
```

```
  WHERE CustomerID = ?"
Dim deleteSQL As String = "DELETE FROM Customers WHERE CustomerID = ?"
```

Las instrucciones de consulta con parámetros definen qué parámetros de entrada y de salida se deben crear. Para crear un parámetro, se utiliza el método **Parameters.Add** o el constructor **Parameter** con el fin de especificar el nombre de columna, tipo de datos y tamaño. En el caso de tipos de datos intrínsecos, como **Integer**, no es necesario incluir el tamaño, aunque se puede especificar el tamaño predeterminado.

En el ejemplo de código siguiente se crean los parámetros para una instrucción SQL y, a continuación, se llena un **DataSet**.

Ejemplo de OleDb

```
' Assumes that connection is a valid OleDbConnection object.
Dim adapter As OleDbDataAdapter = New OleDbDataAdapter
Dim selectCMD AS OleDbCommand = New OleDbCommand(selectSQL, connection)
adapter.SelectCommand = selectCMD
' Add parameters and set values.
selectCMD.Parameters.Add( _
  "@CountryRegion", OleDbType.VarChar, 15).Value = "UK"
selectCMD.Parameters.Add( _
  "@City", OleDbType.VarChar, 15).Value = "London"
Dim customers As DataSet = New DataSet
adapter.Fill(customers, "Customers")
```

Parámetros Odbc

```
' Assumes that connection is a valid OdbcConnection object.
Dim adapter As OdbcDataAdapter = New OdbcDataAdapter
Dim selectCMD AS OdbcCommand = New OdbcCommand(selectSQL, connection)
adapter.SelectCommand = selectCMD
' Add Parameters and set values.
selectCMD.Parameters.Add("@CountryRegion", OdbcType.VarChar, 15).Value =
"UK"
selectCMD.Parameters.Add("@City", OdbcType.VarChar, 15).Value = "London"
Dim customers As DataSet = New DataSet
adapter.Fill(customers, "Customers")
```

Nota:

Si no se proporciona un nombre para un parámetro, éste toma un nombre predeterminado incremental del tipo ParameterN, que comienza por "Parameter1". Se recomienda evitar la convención de nomenclatura del tipo "ParameterN" al asignar un nombre de parámetro, ya que dicho nombre podría entrar en conflicto con un nombre de parámetro predeterminado existente en **ParameterCollection**. Si el nombre proporcionado ya existe, se inicia una excepción.

6.6.3. Agregar Restricciones Existentes al DataSet

El método **Fill** de **DataAdapter** llena un DataSet sólo con las columnas y filas de un origen de datos. Aunque las restricciones se suelen establecer en el origen de datos, el método **Fill** no agrega de forma predeterminada esta información del esquema al **DataSet**. Para llenar un **DataSet** con la información de restricciones de clave principal existentes en el origen de datos, puede llamar al método **FillSchema** de **DataAdapter** o establecer la propiedad **MissingSchemaAction** de **DataAdapter** con el valor **AddWithKey** antes de llamar a **Fill**. De esta forma se garantiza que las restricciones de clave principal

del **DataSet** reflejan las del origen de datos. La información de restricciones de clave externa no se incluye y se debe crear explícitamente.

Al agregar la información del esquema a un **DataSet** antes de llenarlo con datos, se garantiza que se incluyen las restricciones de clave principal con los objetos DataTable en el **DataSet**. Como resultado, al realizar llamadas adicionales para llenar el **DataSet**, la información de la columna de clave principal se puede utilizar para hacer coincidir las nuevas filas del origen de datos con las filas actuales de cada una de las tablas **DataTable**, con lo que los datos actuales de las tablas se sobrescriben con los del origen de datos. Sin la información del esquema, las filas nuevas del origen de datos se agregan al **DataSet**, con lo que se obtienen filas duplicadas.

Nota:

Si una columna del origen de datos es de incremento automático, el método **FillSchema** o el método **Fill** con el valor **AddWithKey** en la propiedad **MissingSchemaAction** crean una **DataColumn** con el valor de la propiedad **AutoIncrement** establecido como **true**. Sin embargo, en este caso debe definir manualmente los valores de **AutoIncrementStep** y **AutoIncrementSeed**.

Al utilizar **FillSchema** o establecer el valor **AddWithKey** en **MissingSchemaAction**, se precisa un proceso adicional en el origen de datos para determinar la información de la columna de clave principal. Este proceso adicional puede reducir el rendimiento. Si conoce en la fase de diseño la información de la clave principal, es aconsejable especificar de modo explícito la columna o columnas que la forman para mejorar el rendimiento.

En el ejemplo de código siguiente se muestra cómo agregar la información del esquema a un **DataSet** mediante **FillSchema**.

```
Dim custDataSet As DataSet = New DataSet()
custAdapter.FillSchema(custDataSet, SchemaType.Source, "Customers")
custAdapter.Fill(custDataSet, "Customers")
```

En el ejemplo de código siguiente se muestra la forma de agregar la información del esquema a un **DataSet** mediante la propiedad **MissingSchemaAction.AddWithKey** del método **Fill**.

```
Dim custDataSet As DataSet = New DataSet()
custAdapter.MissingSchemaAction = MissingSchemaAction.AddWithKey
custAdapter.Fill(custDataSet, "Customers")
```

Controlar varios conjuntos de resultados

Cuando **DataAdapter** encuentra varios conjuntos de resultados devueltos por **SelectCommand**, crea varias tablas en el **DataSet**. Las tablas reciben de forma predeterminada el nombre secuencial que comienza en cero **Table***N*, comenzando por **Table** en lugar de "Table0". Si se pasa un nombre de tabla como argumento al método **FillSchema**, las tablas reciben el nombre secuencial que comienza por cero **TableName***N*, comenzando por **TableName** en lugar de "TableName0".

Nota:

Si se llama al método **FillSchema** del objeto **OleDbDataAdapter** para un comando que devuelve múltiples conjuntos de resultados, sólo se devuelve la información del esquema del primer conjunto de resultados. Al devolver la información del esquema de múltiples conjuntos de resultados mediante **OleDbDataAdapter**, es aconsejable asignar a la propiedad **MissingSchemaAction** el valor **AddWithKey** y obtener la información del esquema al llamar al método **Fill**.

6.6.4. Asignar DataAdapter

DataAdapter contiene una colección con cero o más objetos DataTableMapping en su propiedad **TableMappings**. Un objeto **DataTableMapping** proporciona una asignación principal entre los datos devueltos por una consulta realizada en un origen de datos y una DataTable. El nombre del objeto **DataTableMapping** se puede pasar en lugar del nombre de **DataTable** al método **Fill** de **DataAdapter**. En el ejemplo siguiente se crea un objeto **DataTableMapping** denominado **AuthorsMapping** en la tabla **Authors**.

```
workAdapter.TableMappings.Add("AuthorsMapping", "Authors")
```

Un objeto **DataTableMapping** permite usar en un **DataTable** nombres de columna distintos a los de la base de datos. **DataAdapter** usa la asignación para hacer coincidir las columnas al actualizar la tabla.

Si no se especifica un nombre de **TableName** o de **DataTableMapping** al llamar a los métodos **Fill** o **Update** de **DataAdapter**, **DataAdapter** busca un objeto **DataTableMapping** denominado "Table". Si no existe ese objeto **DataTableMapping**, el nombre de **TableName** de **DataTable** es "Table". Puede especificar un objeto **DataTableMapping** predeterminado si crea un **DataTableMapping** denominado "Table".

En el ejemplo de código siguiente se crea un **DataTableMapping** (a partir del espacio de nombres System.Data.Common) y, al darle el nombre "Table", lo convierte en la asignación predeterminada del **DataAdapter** especificado. A continuación, en el ejemplo se asignan las columnas de la primera tabla de resultados de la consulta (la tabla **Customers** de la base de datos **Northwind**) a un conjunto de nombres más descriptivos existentes en la tabla **Northwind Customers** del DataSet. En las columnas que no se asignan se usa el nombre de la columna en el origen de datos.

```
Dim mapping As DataTableMapping = _
  adapter.TableMappings.Add("Table", "NorthwindCustomers")
mapping.ColumnMappings.Add("CompanyName", "Company")
mapping.ColumnMappings.Add("ContactName", "Contact")
mapping.ColumnMappings.Add("PostalCode", "ZIPCode")
adapter.Fill(custDS)
```

En situaciones más avanzadas, puede que desee que el mismo **DataAdapter** permita la carga de distintas tablas, cada una con sus propias asignaciones. Para ello, basta con agregar más objetos **DataTableMapping**.

Cuando al método **Fill** se le pasa una instancia de un **DataSet** y un nombre de **DataTableMapping**, se utiliza la asignación de ese nombre, si existe, o un **DataTable** con ese nombre, en caso contrario.

En los ejemplos siguientes se crea un **DataTableMapping** denominado **Customers** y una **DataTable** denominada **BizTalkSchema**. En el ejemplo se asignan a continuación las filas devueltas por la instrucción SELECT a la **DataTable BizTalkSchema**.

```
Dim mapping As ITableMapping = _
  adapter.TableMappings.Add("Customers", "BizTalkSchema")
mapping.ColumnMappings.Add("CustomerID", "ClientID")
mapping.ColumnMappings.Add("CompanyName", "ClientName")
mapping.ColumnMappings.Add("ContactName", "Contact")
mapping.ColumnMappings.Add("PostalCode", "ZIP")
adapter.Fill(custDS, "Customers")
```

Nota:

Si no se proporciona un nombre de columna de origen en una asignación de columnas o no se identifica un nombre de tabla de origen en una asignación de tablas, se generan nombres predeterminados de forma automática. Si no se proporciona una columna de origen en una asignación de columnas, la asignación de columna recibe el nombre predeterminado secuencial **SourceColumn***N,* comenzando por **SourceColumn1**. Si no se proporciona un nombre de tabla de origen en una asignación de tablas, la asignación de tabla recibe el nombre predeterminado secuencial **SourceTable***N*, comenzando por **SourceTable1**.

Nota:

Es recomendable evitar la convención de nombres de **SourceColumn***N* para una asignación de columnas o **SourceTable***N* para una asignación de tablas, ya que es posible que el nombre proporcionado entre en conflicto con el nombre predeterminado de asignación de columnas de la **ColumnMappingCollection** o con el nombre de asignación de tablas de la **DataTableMappingCollection**. Si el nombre proporcionado ya existe, se iniciará una excepción.

Controlar varios conjuntos de resultados

Cuando **SelectCommand** devuelve varias tablas, **Fill** genera automáticamente nombres de tabla con valores secuenciales para todas las tablas del **DataSet**, comenzando por el nombre de tabla especificado y continuando con el nombre **TableName***N*, el cual comienza por **TableName1**. Puede usar asignaciones de tabla para asignar el nombre generado automáticamente a un nombre que desee especificar para la tabla en el **DataSet**. Por ejemplo, en el caso de un **SelectCommand** que devuelve dos tablas, **Customers** y **Orders**, puede emitir la llamada siguiente a **Fill**.

```
adapter.Fill(customersDataSet, "Customers")
```

Se crean dos tablas en el **DataSet**: **Customers** y **Customers1**. Puede usar asignaciones de tabla para asegurarse de que la segunda se denomina **Orders** en lugar de **Customers1**. Para ello, asigne la tabla de origen de **Customers1** a la tabla **Orders** del **DataSet**, como se muestra en el ejemplo siguiente.

```
adapter.TableMappings.Add("Customers1", "Orders")
adapter.Fill(customersDataSet, "Customers")
```

6.6.5. Paginar a Través de un Resultado de Consulta

La paginación a través del resultado de una consulta es un proceso que consiste en devolver los resultados de una consulta en subconjuntos menores de datos, o páginas. Se trata de una práctica frecuente para presentar los resultados a un usuario en fragmentos pequeños y fáciles de administrar.

DataAdapter permite devolver únicamente una página de datos mediante sobrecargas del método **Fill**. Sin embargo, quizás no sea la mejor opción para paginar a través de resultados de consultas grandes ya que, aunque el **DataAdapter** rellena la DataTable o el DataSet de destino sólo con los registros solicitados, se siguen utilizando los recursos para devolver toda la consulta. Para devolver una página de datos a partir de un origen de datos sin utilizar los recursos necesarios para devolver toda la consulta, hay que especificar otros criterios adicionales para la consulta que reduzcan las filas devueltas a las filas únicamente necesarias.

Para utilizar el método **Fill** para devolver una página de datos, especifique un parámetro **startRecord** que represente el primer registro de la página de datos y un parámetro **maxRecords** que represente el número de registros de la página de datos.

En el siguiente ejemplo de código se muestra cómo utilizar el método **Fill** para devolver la primera página del resultado de una consulta cuando el tamaño de la página es de cinco registros.

```
Dim currentIndex As Integer = 0
```

```
Dim pageSize As Integer = 5
Dim orderSQL As String = "SELECT * FROM dbo.Orders ORDER BY OrderID"
' Assumes that connection is a valid SqlConnection object.
Dim adapter As SqlDataAdapter = _
  New SqlDataAdapter(orderSQL, connection)
Dim dataSet As DataSet = New DataSet()
adapter.Fill(dataSet, currentIndex, pageSize, "Orders")
```

En el ejemplo anterior, el **DataSet** sólo se rellena con cinco registros pero se devuelve toda la tabla **Pedidos**. Para rellenar el **DataSet** con esos mismos cinco registros, pero devolver únicamente cinco registros, hay que utilizar las cláusulas TOP y WHERE en la instrucción SQL, como en el ejemplo de código siguiente.

```
Dim pageSize As Integer = 5
Dim orderSQL As String = "SELECT TOP " & pageSize & _
  " * FROM Orders ORDER BY OrderID"
Dim adapter As SqlDataAdapter = _
  New SqlDataAdapter(orderSQL, connection)
Dim dataSet As DataSet = New DataSet()
adapter.Fill(dataSet, "Orders")
```

Hay que tener en cuenta que, al paginar a través del resultado de una consulta de esta forma, hay que conservar el identificador único por el que están ordenadas las filas con el fin de pasar el id. único al comando con el fin de devolver la siguiente página de registros, tal y como se muestra en el ejemplo de código siguiente.

```
Dim lastRecord As String = _
  dataSet.Tables("Orders").Rows(pageSize - 1)("OrderID").ToString()
```

Para devolver la siguiente página de registros utilizando la sobrecarga del método **Fill** que toma los parámetros **startRecord** y **maxRecords**, hay que incrementar el índice del registro actual en el tamaño de página y rellenar la tabla. Recuerde que el servidor de base de datos devuelve todos los resultados de la consulta aunque sólo se agregue una página de registros al **DataSet**. En el siguiente ejemplo de código se vacía el contenido de las filas de la tabla antes de rellenarse con la siguiente página de datos. Quizás se desee conservar un cierto número de filas devueltas en una caché local para reducir los viajes al servidor de base de datos.

```
currentIndex = currentIndex + pageSize
dataSet.Tables("Orders").Rows.Clear()
adapter.Fill(dataSet, currentIndex, pageSize, "Orders")
```

Para devolver la siguiente página de registros sin que el servidor de base de datos tenga que devolver toda la consulta, hay que especificar criterios restrictivos en la instrucción SELECT. Como el ejemplo anterior conservaba el último registro devuelto, es posible utilizarlo en la cláusula WHERE con el fin de especificar un punto de partida para la consulta, como se muestra en el ejemplo de código siguiente.

```
orderSQL = "SELECT TOP " & pageSize & _
  " * FROM Orders WHERE OrderID > " & lastRecord & " ORDER BY OrderID"
adapter.SelectCommand.CommandText = orderSQL
dataSet.Tables("Orders").Rows.Clear()
adapter.Fill(dataSet, "Orders")
```

6.6.6. Actualizar Orígenes de Datos con DataAdapters

El método **Update** de DataAdapter se llama para reflejar en el origen de datos todos los cambios efectuados en DataSet. El método **Update**, al igual que el método **Fill**, acepta como argumentos una instancia de **DataSet** y, de forma opcional, un objeto DataTable o un nombre de **DataTable**. La instancia de **DataSet** es el **DataSet** que contiene los cambios efectuados, y **DataTable** identifica la tabla desde la que se pueden recuperar esos cambios. Si no se especifica **DataTable**, se utiliza el primer **DataTable** de **DataSet**.

Al llamar al método **Update**, **DataAdapter** analiza los cambios efectuados y ejecuta el comando apropiado (INSERT, UPDATE o DELETE). Cuando **DataAdapter** encuentra un cambio en DataRow, utiliza los comandos InsertCommand, UpdateCommand o DeleteCommand para reflejarlo. De esta forma, se obtiene el máximo rendimiento de la aplicación de ADO.NET al especificar la sintaxis del comando en la fase de diseño y utilizar, siempre que es posible, procedimientos almacenados. Antes de llamar a **Update** deben establecerse de forma explícita los comandos. Si se llama a **Update** y el comando correspondiente a una actualización determinada no existe (por ejemplo, no hay un comando **DeleteCommand** para las filas eliminadas), se inicia una excepción.

Nota:

Si está utilizando procedimientos almacenados de SQL Server para editar o eliminar datos con **DataAdapter**, asegúrese de que no utiliza SET NOCOUNT ON en la definición del procedimiento almacenado. Esto hace que el recuento de filas afectadas vuelva a cero, lo que **DataAdapter** interpreta como un conflicto de concurrencia. En este evento, se iniciará DBConcurrencyException.

Se pueden usar los parámetros de comando para especificar los valores de entrada y salida de una instrucción SQL o un procedimiento almacenado para cada fila modificada en **DataSet**.

Nota:

Es importante comprender la diferencia entre eliminar una fila de una DataTable y quitar la fila. Al llamar al método **Remove** o **RemoveAt**, la fila se quita inmediatamente. Cualquier fila correspondiente en el origen de datos back end no se verá afectada si a continuación se pasa **DataTable** o **DataSet** a **DataAdapter** y se llama a **Update**. Al utilizar el método **Delete**, la fila permanece en **DataTable** y se marca para eliminación. Si a continuación se pasa **DataTable** o **DataSet** a **DataAdapter** y se llama a **Update**, la fila correspondiente en el origen de datos back end se elimina.

Si **DataTable** está asignada a una única base de datos o se ha generado a partir de ella, puede utilizar el objeto DbCommandBuilder para generar automáticamente los objetos **DeleteCommand**, **InsertCommand** y **UpdateCommand** de **DataAdapter**.

Utilizar UpdatedRowSource para asignar valores a DataSet

Puede controlar la forma en que los valores devueltos desde el origen de datos se asignan a **DataTable** después de una llamada al método Update de **DataAdapter**, utilizando la propiedad UpdatedRowSource de un objeto DbCommand. Al asignar la propiedad **UpdatedRowSource** a uno de los valores de enumeración UpdateRowSource, puede determinar si los parámetros que devuelven los comandos **DataAdapter** se deben omitir o se deben aplicar a la fila cambiada en **DataSet**. También puede especificar si la primera fila devuelta (si existe) se aplica a la fila modificada en **DataTable**.

En la tabla siguiente se describen los distintos valores de la enumeración **UpdateRowSource** y la forma en que afectan al comportamiento del comando utilizado con **DataAdapter**.

Enumeración UpdatedRowSource	Descripción

Both	Tanto los parámetros de salida como la primera fila del conjunto de resultados devuelto se pueden asignar a la fila modificada en **DataSet**.
FirstReturnedRecord	Sólo los datos de la primera fila del conjunto de resultados devuelto se pueden asignar a la fila modificada en el **DataSet**.
None	Se pasan por alto todos los parámetros de salida y las filas del conjunto de resultados devuelto.
OutputParameters	Sólo los parámetros de salida se pueden asignar a la fila modificada en **DataSet**.

El método **Update** vuelve a resolver los cambios en el origen de datos; sin embargo, puede que otros clientes hayan modificado datos en el origen de datos desde la última vez que se llenó **DataSet**. Para actualizar **DataSet** con datos actuales, utilice el **DataAdapter** y el método **Fill**. De esta forma se agregan las filas nuevas a la tabla y se actualiza la información en las filas ya existentes. El método **Fill** determina si se va a agregar una nueva fila o si se va a actualizar una fila existente mediante el examen de los valores de clave principal de las filas de **DataSet** y las filas devueltas por **SelectCommand**. Si el método **Fill** encuentra un valor de clave principal de una fila de **DataSet** que coincide con un valor de clave principal de una fila de los resultados devueltos por **SelectCommand**, éste actualiza la fila existente con la información de la fila devuelta por **SelectCommand** y establece el RowState de la fila existente en **Unchanged**. Si una fila devuelta por **SelectCommand** tiene un valor de clave principal que no coincide con ninguno de los valores de clave principal de las filas de **DataSet**, el método **Fill** agrega una nueva fila con un **RowState** de **Unchanged**.

Nota:

Si **SelectCommand** devuelve los resultados de una combinación externa (OUTER JOIN), **DataAdapter** no establecerá un valor **PrimaryKey** para la tabla **DataTable** resultante. Debe definir **PrimaryKey** para asegurarse de que las filas duplicadas se resuelven correctamente.

Para controlar las excepciones que se puedan producir al llamar al método **Update**, se puede utilizar el evento **RowUpdated** para responder a los errores de actualización de filas en cuanto se producen, o bien se puede establecer **DataAdapter.ContinueUpdateOnError** en **true** antes de llamar a **Update** y responder a la información de error almacenada en la propiedad **RowError** de una determinada fila una vez completada la actualización.

Nota Llamar a **AcceptChanges** en **DataSet**, **DataTable** o **DataRow** hará que todos los valores **Original** de **DataRow** se sobrescriban con los valores **Current** de **DataRow**. Si se han modificado los valores de campo que identifican de forma única a una fila, los valores **Original** dejarán de coincidir con los valores del origen de datos después de llamar a **AcceptChanges**. Se llama automáticamente a **AcceptChanges** para cada fila durante una llamada al método Update de **DataAdapter**. Puede conservar los valores originales durante una llamada al método Update estableciendo primero la propiedad **AcceptChangesDuringUpdate** de **DataAdapter** en false o creando un controlador de eventos para el evento **RowUpdated** y estableciendo Status en SkipCurrentRow.

⊟ Ejemplo

Los ejemplos siguientes muestran cómo realizar las actualizaciones en las filas modificadas estableciendo de forma explícita **UpdateCommand** de **DataAdapter** y llamando a su método **Update** . Observe cómo el parámetro especificado en la cláusula WHERE de la instrucción UPDATE tiene el valor adecuado para usar el valor **Original** de **SourceColumn**. Este hecho es muy importante ya que el valor **Current** puede

haber sido modificado de forma que ya no coincida con el valor del origen de datos. El valor **Original** es el que se usó para rellenar la tabla **DataTable** a partir del origen de datos.

```
Private Sub AdapterUpdate(ByVal connectionString As String)
    Using connection As SqlConnection = New SqlConnection( _
       connectionString)
        Dim adapter As SqlDataAdapter = New SqlDataAdapter( _
          "SELECT CategoryID, CategoryName FROM dbo.Categories", _
          connection)
        adapter.UpdateCommand = New SqlCommand( _
          "UPDATE Categories SET CategoryName = @CategoryName " & _
           "WHERE CategoryID = @CategoryID", connection)
        adapter.UpdateCommand.Parameters.Add( _
           "@CategoryName", SqlDbType.NVarChar, 15, "CategoryName")
        Dim parameter As SqlParameter = _
           adapter.UpdateCommand.Parameters.Add( _
           "@CategoryID", SqlDbType.Int)
        parameter.SourceColumn = "CategoryID"
        parameter.SourceVersion = DataRowVersion.Original
        Dim categoryTable As New DataTable
        adapter.Fill(categoryTable)
        Dim categoryRow As DataRow = categoryTable.Rows(0)
        categoryRow("CategoryName") = "New Beverages"
        adapter.Update(categoryTable)
        Console.WriteLine("Rows after update.")
        Dim row As DataRow
        For Each row In categoryTable.Rows
            Console.WriteLine("{0}: {1}", row(0), row(1))
        Next
    End Using
End Sub
```

⊟ Columnas AutoIncrement

Si las tablas del origen de datos tienen columnas con incremento automático, puede rellenar las columnas de **DataSet** devolviendo el valor de incremento automático como un parámetro de salida de un procedimiento almacenado y asignándolo a una columna de la tabla, o devolviendo el valor de incremento automático de la primera fila de un conjunto de resultados devuelto por un procedimiento almacenado o una instrucción SQL, o mediante el evento **RowUpdated** de **DataAdapter** para ejecutar una instrucción SELECT adicional.

⊟ Orden de las inserciones, actualizaciones y eliminaciones

En algunas circunstancias, es importante el orden en que se envían al origen de datos los cambios realizados en el **DataSet**. Por ejemplo, si se actualiza el valor de una clave principal de una fila existente y se ha agregado una nueva fila con el nuevo valor de la clave principal como una clave externa, es importante que la actualización de la fila se procese antes que la inserción.

Puede usar el método **Select** de **DataTable** para devolver una matriz **DataRow** que sólo haga referencia a filas con un estado **RowState** determinado. A continuación, puede pasar la matriz **DataRow** al método **Update** de **DataAdapter** para procesar las filas modificadas. Al especificar un subconjunto de filas que modificar, puede controlar el orden en que se procesan las inserciones, actualizaciones y eliminaciones.

Ejemplo

Por ejemplo, en el código siguiente se garantiza que en primer lugar se realizan en la tabla las eliminaciones de filas, después las actualizaciones y finalmente las inserciones.

```
Dim table As DataTable = dataSet.Tables("Customers")
' First process deletes.
dataSet.Update(table.Select(Nothing, Nothing, DataViewRowState.Deleted))
' Next process updates.
adapter.Update(table.Select(Nothing, Nothing,
  DataViewRowState.ModifiedCurrent))
' Finally, process inserts.
dataAdpater.Update(table.Select(Nothing, Nothing,DataViewRowState.Added))
```

6.6.7. Control de Eventos del DataAdapter

DataAdapter de ADO.NET expone tres eventos que se pueden utilizar para responder a los cambios efectuados en los datos en el origen. En la siguiente tabla se muestran los eventos de **DataAdapter**.

Evento	Descripción
RowUpdating	Está a punto de comenzar una operación UPDATE, INSERT o DELETE en una fila (mediante una llamada a uno de los métodos **Update**).
RowUpdated	Se ha completado una operación UPDATE, INSERT o DELETE en una fila (mediante una llamada a uno de los métodos **Update**).
FillError	Se ha producido un error durante una operación **Fill**.

RowUpdating y RowUpdated

El evento **RowUpdating** se activa antes de que se produzca la actualización de una fila del DataSet en el origen de datos. El evento **RowUpdated** se activa después de que se produzca la actualización de una fila del **DataSet** en el origen de datos. Por lo tanto, puede utilizar **RowUpdating** para modificar el comportamiento de la actualización antes de que tenga lugar, proporcionar un control adicional del proceso durante la actualización, conservar una referencia a la fila actualizada, cancelar la actualización actual y programarla como parte de un proceso por lotes que se ejecutará después, entre otras acciones. **RowUpdated** es útil para reaccionar cuando se producen errores y excepciones durante la actualización. Puede agregar información de errores al **DataSet**, así como procedimientos de reintento, etcétera.

Los argumentos RowUpdatingEventArgs y RowUpdatedEventArgs que se pasan a los eventos **RowUpdating** y **RowUpdated** incluyen lo siguiente: una propiedad **Command** que hace referencia al objeto **Command** que se está utilizando para realizar la actualización; una propiedad **Row** que hace referencia al objeto **DataRow** que contiene la información actualizada; una propiedad **StatementType** para el tipo de actualización que se está llevando a cabo; el valor de **TableMapping**, si es pertinente y el valor de **Status** de la operación.

Puede utilizar la propiedad **Status** para determinar si se ha producido o no un error durante la operación y, si lo desea, controlar las acciones que se emprenden en las filas actuales y las resultantes de la operación. Cuando se produce el evento, la propiedad **Status** toma el valor **Continue** o **ErrorsOccurred**. En la tabla siguiente se muestran los valores que se pueden asignar a la propiedad **Status** para controlar las acciones posteriores en el proceso de actualización.

Estado	Descripción
Continue	Continuar la operación de actualización.
ErrorsOccurred	Anular la operación de actualización e iniciar una excepción.
SkipCurrentRow	Omitir la fila actual y continuar la operación de actualización.
SkipAllRemainingRows	Anular la operación de actualización sin iniciar una excepción.

Al establecer a la propiedad **Status** en el valor **ErrorsOccurred** se inicia una excepción. Puede controlar qué excepciones se inician si establece la propiedad **Errors** en la excepción que desee. El uso de un valor distinto para la propiedad **Status** evita que se inicie una excepción.

También puede utilizar la propiedad **ContinueUpdateOnError** para controlar los errores producidos en las filas actualizadas. Cuando **DataAdapter.ContinueUpdateOnError** tiene el valor **true** y la actualización de una fila inicia una excepción, el texto de la excepción se coloca en la información **RowError** de la fila en cuestión y el proceso continúa sin que se inicie una excepción. De esta forma, puede responder a los errores cuando se complete el proceso **Update**, a diferencia del evento **RowUpdated**, que permite responder a los errores cuando se detectan.

En el ejemplo de código siguiente se muestra cómo se pueden agregar y quitar controladores de eventos. El controlador de eventos **RowUpdating** mantiene un registro de todos los registros eliminados y una marca de tiempo asociada a cada uno de ellos. El controlador de eventos **RowUpdated** agrega información de error a la propiedad **RowError** de la fila correspondiente en el **DataSet**, evita que se inicie la excepción y deja continuar el proceso (al igual que **ContinueUpdateOnError** = **true**).

```
' Assumes that connection is a valid SqlConnection object.
Dim custAdapter As SqlDataAdapter = New SqlDataAdapter( _
  "SELECT CustomerID, CompanyName FROM Customers", connection)
' Add handlers.
AddHandler custAdapter.RowUpdating, New SqlRowUpdatingEventHandler( _
  AddressOf OnRowUpdating)
AddHandler custAdapter.RowUpdated, New SqlRowUpdatedEventHandler(
  AddressOf OnRowUpdated)
' Set DataAdapter command properties, fill DataSet, and modify DataSet.
custAdapter.Update(custDS, "Customers")
' Remove handlers.
RemoveHandler custAdapter.RowUpdating, _
  New SqlRowUpdatingEventHandler(AddressOf OnRowUpdating)
RemoveHandler custAdapter.RowUpdated, _
  New SqlRowUpdatedEventHandler(AddressOf OnRowUpdated)
Private Shared Sub OnRowUpdating(sender As Object, _
  args As SqlRowUpdatingEventArgs)
  If args.StatementType = StatementType.Delete Then
    Dim tw As System.IO.TextWriter = _
  System.IO.File.AppendText("Deletes.log")
    tw.WriteLine( _
      "{0}: Customer {1} Deleted.", DateTime.Now, args.Row(_
      "CustomerID", DataRowVersion.Original))
    tw.Close()
```

```
  End If
End Sub
Private Shared Sub OnRowUpdated( _
  sender As Object, args As SqlRowUpdatedEventArgs)
  If args.Status = UpdateStatus.ErrorsOccurred
    args.Status = UpdateStatus.SkipCurrentRow
    args.Row.RowError = args.Errors.Message
  End If
End Sub
```

FillError

Cuando se produce un error durante una operación **Fill**, el objeto **DataAdapter** provoca el evento **FillError**.Este tipo de error suele producirse si al convertir los datos de la fila que se agrega a un tipo de .NET Framework se produce una pérdida de precisión.

Si el error se produce durante una operación **Fill**, la fila actual no se agrega al objeto **DataTable**. El evento **FillError** permite resolver el error y agregar la fila o bien pasar por alto la fila excluida y continuar con la operación **Fill**.

El objeto **FillErrorEventArgs** que se pasa al evento **FillError** puede contener varias propiedades que permiten reaccionar en caso de errores y resolverlos. En la tabla siguiente se muestran las propiedades del objeto **FillErrorEventArgs**.

Propiedad	Descripción
Errors	**Exception** que se ha producido.
DataTable	Objeto **DataTable** que se estaba llenando cuando ocurrió el error.
Values	Matriz de objetos que contiene los valores de la fila que se está agregando cuando se produce el error. Las referencias de orden de la matriz **Values** coinciden con las de las columnas de la fila que se está agregando. Por ejemplo, **Values[0]** es el valor que se agrega en la primera columna de la fila.
Continue	Permite elegir si desea iniciar una excepción o no. Si establece la propiedad **Continue** en **false**, la operación **Fill** en curso se detiene y se inicia una excepción. Si establece la propiedad **Continue** en **true**, la operación **Fill** continúa pese al error.

En el siguiente ejemplo de código se agrega un controlador para el evento **FillError** del objeto **DataAdapter**. En el código del evento **FillError**, el ejemplo determina si hay una posible pérdida de precisión y ofrece la posibilidad de reaccionar ante la excepción.

```
AddHandler adapter.FillError, New FillErrorEventHandler( _
  AddressOf FillError)
Dim dataSet As DataSet = New DataSet
adapter.Fill(dataSet, "ThisTable")
Private Shared Sub FillError(sender As Object, _
  args As FillErrorEventArgs)
  If args.Errors.GetType() Is Type.GetType("System.OverflowException")
Then
    ' Code to handle precision loss.
    ' Add a row to table using the values from the first two columns.
    DataRow myRow = args.DataTable.Rows.Add(New Object() _
      {args.Values(0), args.Values(1), DBNull.Value})
```

```
    ' Set the RowError containing the value for the third column.
    args.RowError = _
      "OverflowException encountered. Value from data source: " & _
      args.Values(2)
    args.Continue = True
  End If
End Sub
```

6.6.8. Realizar Operaciones por Lotes con DataAdapters

La compatibilidad con las operaciones por lotes en ADO.NET permite que un DataAdapter agrupe operaciones INSERT, UPDATE y DELETE desde un DataSet o una DataTable al servidor, en lugar de enviar las operaciones de una en una. La reducción del número de viajes de ida y vuelta (round trip) al servidor tiene como resultado una mejora considerable del rendimiento. Las actualizaciones por lotes son compatibles con los proveedores de datos de .NET para SQL Server (System.Data.SqlClient) y Oracle (System.Data.OracleClient).

Al actualizar una base de datos con modificaciones de un DataSet en versiones anteriores de ADO.NET, el método **Update** de un **DataAdapter** realizaba actualizaciones de las filas de la base de datos de una en una. A medida que recorría las filas de la DataTable especificada, examinaba cada DataRow para ver si se había modificado. Si se había modificado la fila, llamaba al **UpdateCommand**, **InsertCommand** o **DeleteCommand** apropiado, en función del valor de la propiedad RowState de la fila. Cada actualización de una fila implicaba un viaje de ida y vuelta (round trip) a la base de datos.

En ADO.NET 2.0, el DbDataAdapter expone una propiedad UpdateBatchSize. Si se establece el **UpdateBatchSize** en un valor entero positivo, se producen actualizaciones en la base de datos que se envían como lotes del tamaño especificado. Por ejemplo, si se establece el **UpdateBatchSize** en 10, se agrupan 10 instrucciones separadas y se envían en un único lote. Si se establece el **UpdateBatchSize** en 0, el DataAdapter utilizará el mayor tamaño de lote admitido por el servidor. Si se establece el valor en 1, se deshabilitan las actualizaciones por lotes y las filas se envían de una en una.

Si se ejecuta un lote demasiado grande, el rendimiento podría verse afectado. Por tanto, es conveniente realizar pruebas a fin de determinar el valor óptimo del tamaño del lote antes de implementar la aplicación.

⊟ Utilizar la propiedad UpdateBatchSize

Al habilitar las actualizaciones por lotes, el valor de la propiedad UpdatedRowSource de **UpdateCommand**, **InsertCommand** y **DeleteCommand** del DataAdapter debe establecerse en None o OutputParameters. Al realizar una actualización por lotes, el valor FirstReturnedRecord o Both de la propiedad UpdatedRowSource del comando no es válido.

En el siguiente procedimiento se muestra cómo se utiliza la propiedad **UpdateBatchSize**. En el procedimiento se toman dos argumentos, un objeto DataSet con columnas que representan los campos **ProductCategoryID** y **Name** en la tabla **Production.ProductCategory**, y un entero que representa el tamaño del lote, es decir, el número de filas del mismo. El código crea un objeto SqlDataAdapter nuevo y se establecen las propiedades UpdateCommand, InsertCommand y DeleteCommand. En el código se supone que el objeto DataSet tiene filas modificadas. Se establece la propiedad **UpdateBatchSize** y se ejecuta la actualización.

```
Public Sub BatchUpdate( _
```

```
  ByVal dataTable As DataTable, ByVal batchSize As Int32)
    ' Assumes GetConnectionString() returns a valid connection string.
    Dim connectionString As String = GetConnectionString()
    ' Connect to the AdventureWorks database.
    Using connection As New SqlConnection(connectionString)
        ' Create a SqlDataAdapter.
        Dim adapter As New SqlDataAdapter()
        'Set the UPDATE command and parameters.
        adapter.UpdateCommand = New SqlCommand( _
          "UPDATE Production.ProductCategory SET " _
          & "Name=@Name WHERE ProductCategoryID=@ProdCatID;", _
          connection)
        adapter.UpdateCommand.Parameters.Add("@Name", _
          SqlDbType.NVarChar, 50, "Name")
        adapter.UpdateCommand.Parameters.Add("@ProdCatID",  _
          SqlDbType.Int, 4, " ProductCategoryID ")
        adapter.UpdateCommand.UpdatedRowSource = _
          UpdateRowSource.None
        'Set the INSERT command and parameter.
        adapter.InsertCommand = New SqlCommand( _
       "INSERT INTO Production.ProductCategory (Name) VALUES (@Name);", _
       connection)
        adapter.InsertCommand.Parameters.Add("@Name", _
          SqlDbType.NVarChar, 50, "Name")
        adapter.InsertCommand.UpdatedRowSource = _
          UpdateRowSource.None
        'Set the DELETE command and parameter.
        adapter.DeleteCommand = New SqlCommand( _
          "DELETE FROM Production.ProductCategory " _
          & "WHERE ProductCategoryID=@ProdCatID;", connection)
        adapter.DeleteCommand.Parameters.Add("@ProdCatID", _
           SqlDbType.Int, 4, " ProductCategoryID ")
        adapter.DeleteCommand.UpdatedRowSource = UpdateRowSource.None
        ' Set the batch size.
        adapter.UpdateBatchSize = batchSize
        ' Execute the update.
        adapter.Update(dataTable)
    End Using
End Sub
```

Controlar errores y eventos relacionados con la actualización por lotes

DataAdapter tiene dos eventos relacionados con la actualización: **RowUpdating** y **RowUpdated**. En las versiones anteriores de ADO.NET, cuando se deshabilita el procesamiento por lotes, cada uno de estos eventos se genera una vez para cada fila procesada. **RowUpdating** se genera antes de que tenga lugar la actualización y **RowUpdated** se genera una vez completada la actualización de la base de datos.

Cambios en el comportamiento de eventos con actualizaciones por lotes

Si se habilita el procesamiento por lotes, se actualizan varias filas en una única operación de base de datos. Por tanto, sólo se produce un evento **RowUpdated** para cada lote, mientras que el evento **RowUpdating** se produce para cada fila procesada. Si se deshabilita el procesamiento por lotes, los dos eventos se activan con entrelazado individualizado, donde los eventos **RowUpdating** y **RowUpdated** se

activan para una fila y, a continuación, se activan los eventos **RowUpdating** y **RowUpdated** para la siguiente fila, hasta que se hayan procesado todas las filas.

Obtener acceso a filas actualizadas
Si se deshabilita el procesamiento por lotes, se puede obtener acceso a la fila que se está actualizando mediante la propiedad Row de la clase RowUpdatedEventArgs.

Cuando se habilita el procesamiento por lotes, se genera un único evento **RowUpdated** para varias filas. Por tanto, el valor de la propiedad **Row** para cada fila es nulo. Aún así, los eventos **RowUpdating** se generarán para cada fila. El método CopyToRows de la clase RowUpdatedEventArgs permite obtener acceso a las filas procesadas al copiar referencias a las mismas en una matriz. Si no se está procesando ninguna fila, **CopyToRows** inicia una ArgumentNullException. Utilice la propiedad RowCount para devolver el número de filas procesadas antes de llamar al método CopyToRows.

Controlar errores de datos
La ejecución por lotes tiene el mismo efecto que la ejecución de cada instrucción por separado. Las instrucciones se ejecutan en el mismo orden en el que se agregaron al lote. Los errores se controlan de la misma forma en el modo de procesamiento por lotes que cuando éste se encuentra deshabilitado. Cada fila se procesa por separado. Sólo aquellas filas procesadas correctamente en la base de datos se actualizarán en la DataRow correspondiente dentro de la DataTable.

El proveedor de datos y el servidor de base de datos back-end determinan qué construcciones SQL son compatibles para la ejecución por lotes. Es posible que se inicie una excepción si se envía una instrucción no compatible para su ejecución.

6.7. Recuperar Valores de Identidad o Valores Autonuméricos

Una clave principal de una base de datos relacional es una columna o combinación de columnas que siempre contienen valores únicos. Conocer el valor de la clave principal permite localizar la fila que la contiene. Los motores de bases de datos relacionales, como SQL Server, Oracle y Microsoft Access/Jet admiten la creación de columnas de incremento automático que pueden designarse como claves principales. Estos valores los genera el servidor cuando se agregan filas a una tabla. En SQL Server se establece la propiedad de identidad de una columna, en Oracle se crea una secuencia y en Microsoft Access se crea una columna Autonumérica.

DataColumn también se puede utilizar para generar de manera automática valores incrementales estableciendo la propiedad AutoIncrement en true. No obstante, podría haber valores duplicados en instancias distintas de DataTable si varias aplicaciones cliente están generando por separado valores incrementales de manera automática. Si se tiene un servidor que genera de manera automática valores incrementales se eliminan posibles conflictos, pues se permite a cada usuario recuperar el valor generado para cada fila insertada.

Durante una llamada al método **Update** de **DataAdapter**, la base de datos puede volver a enviar datos a la aplicación ADO.NET como parámetros de salida o como el primer registro devuelto del conjunto de resultados de una instrucción SELECT ejecutada en el mismo lote que la instrucción INSERT. ADO.NET puede recuperar estos valores y actualizar las columnas correspondientes en el DataRow que se está actualizando.

Algunos motores de base de datos, como los de Microsoft Access Jet, no admiten parámetros de salida y no pueden procesar varias instrucciones en un único lote. Cuando trabaje con el motor de base de datos de Jet, puede recuperar el nuevo valor Autonumérico generado para una fila insertada ejecutando un comando SELECT distinto en un controlador de eventos para el evento **RowUpdated** de **DataAdapter**.

Nota:

Una opción alternativa al uso de un valor de incremento automático es utilizar el método NewGuid de un objeto Guid para generar un GUID (identificador único global) en el equipo cliente que se pueda copiar al servidor cuando se inserte una nueva fila. El método **NewGuid** genera un valor binario de 16 bits que se crea mediante un algoritmo que permite que haya una alta probabilidad de que no se duplique ningún valor. En una base de datos de SQL Server, el GUID se almacena en una columna **uniqueidentifier** que SQL Server puede generar automáticamente mediante la función Transact-SQL **NEWID()**. Utilizar un GUID como clave principal puede afectar de manera negativa al rendimiento. SQL Server 2005 incorpora compatibilidad para la función **NEWSEQUENTIALID()**, que genera un GUID secuencial que no está garantizado que sea único global, pero que se puede indizar de forma más eficaz.

Recuperar valores de columnas de identidad de SQL Server

Cuando trabaje con Microsoft SQL Server, puede crear procedimientos almacenados con un parámetro de salida para devolver el valor de identidad de una fila insertada. La siguiente tabla describe las tres funciones de Transact-SQL en SQL Server que se pueden utilizar para recuperar valores de columna de identidad.

Función	Descripción
SCOPE_IDENTITY	Devuelve el último valor de identidad en el ámbito de ejecución actual. SCOPE_IDENTITY se recomienda en la mayoría de los casos.
@@IDENTITY	Contiene el último valor de identidad generado en cualquier tabla de la sesión actual. @@IDENTITY puede verse afectado por los desencadenadores y no devolver el valor de identidad esperado.
IDENT_CURRENT	Devuelve el último valor de identidad generado para una tabla concreta de cualquier sesión y en cualquier ámbito.

El siguiente procedimiento almacenado muestra cómo insertar una fila en la tabla **Categories** y utilizar un parámetro de salida para devolver el nuevo valor de identidad generado por la función Transact-SQL SCOPE_IDENTITY().

```
CREATE PROCEDURE dbo.InsertCategory
  @CategoryName nvarchar(15),
  @Identity int OUT
AS
INSERT INTO Categories (CategoryName) VALUES(@CategoryName)
SET @Identity = SCOPE_IDENTITY()
```

El procedimiento almacenado se puede especificar como el origen de InsertCommand de un objeto SqlDataAdapter. La propiedad CommandType de InsertCommand debe establecerse en StoredProcedure. La salida de identidad se recupera creando un SqlParameter que tiene un ParameterDirection de Output. Cuando se procesa **InsertCommand**, el valor de identidad de incremento automático se devuelve y se coloca en la columna **CategoryID** de la fila actual si se establece la propiedad UpdatedRowSource del comando de inserción en **UpdateRowSource.OutputParameters** o **UpdateRowSource.Both**.

Si el comando de inserción ejecuta un lote que incluye tanto una instrucción INSERT como una instrucción SELECT que devuelven el nuevo valor de identidad, entonces puede recuperar el nuevo valor

estableciendo la propiedad **UpdatedRowSource** del comando de inserción en **UpdateRowSource.FirstReturnedRecord**.

```
Private Sub RetrieveIdentity(ByVal connectionString As String)
    Using connection As SqlConnection = New SqlConnection( _
       connectionString)
        ' Create a SqlDataAdapter based on a SELECT query.
        Dim adapter As SqlDataAdapter = New SqlDataAdapter( _
           "SELECT CategoryID, CategoryName FROM dbo.Categories", _
           connection)
        ' Create the SqlCommand to execute the stored procedure.
        adapter.InsertCommand = New SqlCommand("dbo.InsertCategory", _
           connection)
        adapter.InsertCommand.CommandType = CommandType.StoredProcedure
        ' Add the parameter for the CategoryName. Specifying the
        ' ParameterDirection for an input parameter is not required.
        adapter.InsertCommand.Parameters.Add( _
          "@CategoryName", SqlDbType.NVarChar, 15, "CategoryName")
        ' Add the SqlParameter to retrieve the new identity value.
        ' Specify the ParameterDirection as Output.
        Dim parameter As SqlParameter = _
           adapter.InsertCommand.Parameters.Add( _
          "@Identity", SqlDbType.Int, 0, "CategoryID")
        parameter.Direction = ParameterDirection.Output
        ' Create a DataTable and fill it.
        Dim categories As DataTable = New DataTable
        adapter.Fill(categories)
        ' Add a new row.
        Dim newRow As DataRow = categories.NewRow()
        newRow("CategoryName") = "New Category"
        categories.Rows.Add(newRow)
        ' Update the database.
        adapter.Update(categories)
        Console.WriteLine("List All Rows:")
        Dim row As DataRow
        For Each row In categories.Rows
            Console.WriteLine("{0}: {1}", row(0), row(1))
        Next
    End Using
End Sub
```

Combinar nuevos valores de identidad

Un caso frecuente es llamar al método **GetChanges** de **DataTable** para crear una copia que contiene únicamente filas modificadas y utilizar la nueva copia al llamar al método **Update** de **DataAdapter**. Esto es especialmente útil cuando hay que calcular las referencias de las filas modificadas en un componente independiente que realiza la actualización. Después de la actualización, la copia puede contener nuevos valores de identidad que se deben volver a combinar en el **DataTable** original. Probablemente los nuevos valores de identidad son diferentes a los valores originales de **DataTable**. Para conseguir la combinación, se deben mantener los valores originales de las columnas **AutoIncrement** en la copia para poder localizar y actualizar filas existentes en el original **DataTable**, en lugar de anexar filas nuevas con los nuevos valores de identidad. No obstante, de manera predeterminada estos valores se pierden

después de una llamada al método **Update** de **DataAdapter**, debido a que se llama implícitamente a **AcceptChanges** en cada **DataRow** actualizada.

Hay dos maneras de mantener los valores originales de **DataColumn** en **DataRow** durante una actualización de **DataAdapter**:

- El primer método para mantener los valores originales consiste en establecer la propiedad **AcceptChangesDuringUpdate** de **DataAdapter** en **false**. Esto afecta a cada **DataRow** de **DataTable** que se está actualizando.
- El segundo método consiste en escribir código en el controlador de eventos **RowUpdated** de **DataAdapter** para establecer Status en SkipCurrentRow. **DataRow** se actualiza pero se mantiene el valor original de cada **DataColumn**. Este método permite mantener los valores originales en algunas filas y no en otras. Por ejemplo, el código puede mantener los valores originales de filas agregadas y no los de filas editadas o eliminadas comprobando primero StatementType y, a continuación, estableciendo Status en SkipCurrentRow únicamente para filas con un **StatementType** de **Insert**.

Cuando se utiliza alguno de estos métodos para mantener los valores originales de **DataRow** durante una actualización de **DataAdapter**, ADO.NET realiza una serie de acciones para establecer los valores actuales de **DataRow** a los nuevos valores devueltos por parámetros de salida o por la primera fila devuelta de un conjunto de resultados, al tiempo que se mantiene el valor original de cada **DataColumn**. Primero, se llama al método **AcceptChanges** de **DataRow** para mantener los valores actuales como valores originales y, a continuación, se asignan los nuevos valores. Después de estas acciones, las **DataRows** que tienen la propiedad RowState establecida en Added tendrán su propiedad **RowState** establecida en Modified, lo que puede ser inesperado.

El modo en que se aplican los resultados del comando a cada DataRow que se actualiza lo determina la propiedad UpdatedRowSource de cada DbCommand. Esta propiedad se establece en un valor desde la enumeración **UpdateRowSource**.

La siguiente tabla describe cómo afectan los valores de enumeración **UpdateRowSource** a la propiedad RowState de las filas actualizadas.

Nombre del miembro	Descripción
Both	Se llama a **AcceptChanges** y tanto los parámetros de salida como los valores de la primera fila de cualquier conjunto de resultados devuelto se colocan en la **DataRow** que se está actualizando. Si no hay valores que aplicar, **RowState** será Unchanged.
FirstReturnedRecord	Si se devuelve una fila, se llama a **AcceptChanges** y la fila se asigna a la fila modificada en **DataTable**, estableciendo **RowState** en **Modified**. Si no se devuelve ninguna fila, entonces no se llama a **AcceptChanges** y **RowState** permanece en **Added**.
None	Se pasan por alto todos los parámetros o filas devueltos. No hay llamada a **AcceptChanges** y **RowState** permanece en **Added**.
OutputParameters	Se llama a **AcceptChanges** y todos los parámetros de salida se asignan a la fila modificada en **DataTable**, estableciendo **RowState** en **Modified**. Si no hay parámetros de salida, **RowState** será **Unchanged**.

Ejemplo

Este ejemplo muestra la extracción de filas modificadas desde **DataTable** y el uso de SqlDataAdapter para actualizar el origen de datos y recuperar un nuevo valor de columna de identidad. InsertCommand ejecuta dos instrucciones Transact-SQL; la primera es la instrucción INSERT y la segunda es la instrucción SELECT.

```
INSERT INTO dbo.Shippers (CompanyName)
VALUES (@CompanyName);
SELECT ShipperID, CompanyName FROM dbo.Shippers
WHERE ShipperID = SCOPE_IDENTITY();
```

La propiedad **UpdatedRowSource** del comando de inserción se establece en **UpdateRowSource.FirstReturnedRow** y la propiedad MissingSchemaAction de **DataAdapter** se establece en **MissingSchemaAction.AddWithKey**. **DataTable** se rellena y el código agrega una nueva fila a **DataTable**. A continuación, las filas modificadas se extraen en un nuevo **DataTable**, que se pasa a **DataAdapter**, el cual actualiza el servidor.

```
Private Sub MergeIdentityColumns(ByVal connectionString As String)
    Using connection As SqlConnection = New SqlConnection( _
       connectionString)
        ' Create the DataAdapter
        Dim adapter As SqlDataAdapter = New SqlDataAdapter( _
          "SELECT ShipperID, CompanyName FROM dbo.Shippers", connection)
        ' Add the InsertCommand to retrieve new identity value.
        adapter.InsertCommand = New SqlCommand( _
            "INSERT INTO dbo.Shippers (CompanyName) " & _
            "VALUES (@CompanyName); " & _
            "SELECT ShipperID, CompanyName FROM dbo.Shippers " & _
            "WHERE ShipperID = SCOPE_IDENTITY();", _
            connection)
        ' Add the parameter for the inserted value.
        adapter.InsertCommand.Parameters.Add( _
           New SqlParameter("@CompanyName", SqlDbType.NVarChar, 40, _
           "CompanyName"))
        adapter.InsertCommand.UpdatedRowSource = UpdateRowSource.Both
        ' MissingSchemaAction adds any missing schema to
        ' the DataTable, including identity columns
        adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey
        ' Fill the DataTable.
        Dim shipper As New DataTable
        adapter.Fill(shipper)
        ' Add a new shipper.
        Dim newRow As DataRow = shipper.NewRow()
        newRow("CompanyName") = "New Shipper"
        shipper.Rows.Add(newRow)
        ' Add changed rows to a new DataTable. This
        ' DataTable will be used by the DataAdapter.
        Dim dataChanges As DataTable = shipper.GetChanges()
        ' Add the event handler.
        AddHandler adapter.RowUpdated, New _
           SqlRowUpdatedEventHandler(AddressOf OnRowUpdated)
        ' Update the datasource with the modified records.
```

```
        adapter.Update(dataChanges)
        ' Merge the two DataTables.
        shipper.Merge(dataChanges)
        ' Commit the changes.
        shipper.AcceptChanges()
        Console.WriteLine("Rows after merge.")
        Dim row As DataRow
        For Each row In shipper.Rows
            Console.WriteLine("{0}: {1}", row(0), row(1))
        Next
    End Using
End Sub
```

El controlador de eventos **OnRowUpdated** comprueba StatementType de SqlRowUpdatedEventArgs para determinar si la fila es una inserción. Si lo es, entonces la propiedad se establece Status en SkipCurrentRow. La fila está actualizada, pero los valores originales de la fila se mantienen. En el cuerpo principal del procedimiento, se llama al método Merge para combinar el nuevo valor de identidad en el **DataTable** original y, finalmente, se llama a **AcceptChanges**.

```
Private Sub OnRowUpdated( _
    ByVal sender As Object, ByVal e As SqlRowUpdatedEventArgs)
    ' If this is an insert, then skip this row.
    If e.StatementType = StatementType.Insert Then
        e.Status = UpdateStatus.SkipCurrentRow
    End If
End Sub
```

⊟ Recuperar valores de autonumeración de Microsoft Access

Esta sección incluye un ejemplo que muestra cómo recuperar valores de **Autonumber** desde una base de datos de Jet 4.0. El motor de la base de datos de Jet no admite la ejecución de varias instrucciones en un lote o el uso de parámetros de salida, por lo que no es posible utilizar ninguna de estas técnicas para devolver el nuevo valor **Autonumber** asignado a una fila insertada. No obstante, se puede agregar código al controlador de eventos **RowUpdated** que ejecuta una instrucción SELECT @@IDENTITY independiente para recuperar el valor **Autonumber** nuevo.

Ejemplo

En lugar de agregar información de esquema utilizando **MissingSchemaAction.AddWithKey**, este ejemplo configura **DataTable** con el esquema adecuado antes de llamar a OleDbDataAdapter para rellenar **DataTable**. En este caso, la columna **CategoryID** se configura para disminuir el valor asignado a cada fila insertada empezando desde cero, estableciendo AutoIncrement en **true**, AutoIncrementSeed en 0 y AutoIncrementStep en -1. Entonces, el código agrega dos filas nuevas y utiliza **GetChanges** para agregar las filas modificadas a un nuevo **DataTable** que se pasa al método **Update**.

```
Shared connection As OleDbConnection = Nothing
Private Shared Sub MergeIdentityColumns(ByVal connection As
OleDbConnection)
    Using connection
        ' Create a DataAdapter based on a SELECT query.
        Dim adapter As OleDbDataAdapter = New OleDbDataAdapter( _
          "SELECT CategoryID, CategoryName FROM Categories", _
          connection)
        ' Create the INSERT command for the new category.
```

```
        adapter.InsertCommand = New OleDbCommand( _
          "INSERT INTO Categories (CategoryName) Values(?)", connection)
        adapter.InsertCommand.CommandType = CommandType.Text
        ' Add the parameter for the CategoryName.
        adapter.InsertCommand.Parameters.Add( _
          "@CategoryName", OleDbType.VarWChar, 15, "CategoryName")
        adapter.InsertCommand.UpdatedRowSource = UpdateRowSource.Both
        ' Create a DataTable.
        Dim categories As DataTable = New DataTable
        ' Create the CategoryID column and set its auto
        ' incrementing properties to decrement from zero.
        Dim column As New DataColumn()
        column.DataType = System.Type.GetType("System.Int32")
        column.ColumnName = "CategoryID"
        column.AutoIncrement = True
        column.AutoIncrementSeed = 0
        column.AutoIncrementStep = -1
        categories.Columns.Add(column)
        ' Create the CategoryName column.
        column = New DataColumn()
        column.DataType = System.Type.GetType("System.String")
        column.ColumnName = "CategoryName"
        categories.Columns.Add(column)
        ' Set the primary key on CategoryID.
        Dim pKey(1) As DataColumn
        pKey(0) = categories.Columns("CategoryID")
        categories.PrimaryKey = pKey
        ' Fetch the data and fill the DataTable.
        adapter.Fill(categories)
        ' Add a new row.
        Dim newRow As DataRow = categories.NewRow()
        newRow("CategoryName") = "New Category"
        categories.Rows.Add(newRow)
        ' Add another new row.
        Dim newRow2 As DataRow = categories.NewRow()
        newRow2("CategoryName") = "Another New Category"
        categories.Rows.Add(newRow2)
        ' Add changed rows to a new DataTable that will be
        ' used to post the inserts to the database.
        Dim dataChanges As DataTable = categories.GetChanges()
        ' Include an event to fill in the Autonumber value.
        AddHandler adapter.RowUpdated, _
          New OleDbRowUpdatedEventHandler(AddressOf OnRowUpdated)
        ' Update the database, inserting the new rows.
        adapter.Update(dataChanges)
        Console.WriteLine("Rows before merge:")
        Dim row1 As DataRow
        For Each row1 In categories.Rows
            Console.WriteLine("  {0}: {1}", row1(0), row1(1))
        Next
```

```
        ' Merge the two DataTables.
        categories.Merge(dataChanges)
        ' Commit the changes.
        categories.AcceptChanges()
        Console.WriteLine("Rows after merge:")
        Dim row As DataRow
        For Each row In categories.Rows
            Console.WriteLine("  {0}: {1}", row(0), row(1))
        Next
    End Using
End Sub
```

El controlador de eventos **RowUpdated** utiliza el mismo OleDbConnection abierto que la instrucción **Update** de **OleDbDataAdapter**. Comprueba el **StatementType** de OleDbRowUpdatedEventArgs de las filas insertadas. Se crea un nuevo OleDbCommand en cada nueva fila insertada para ejecutar la instrucción SELECT @@IDENTITY en la conexión, devolviendo el nuevo valor **Autonumber**, que está situado en la columna **CategoryID** de **DataRow**. La propiedad **Status** se establece luego en **UpdateStatus.SkipCurrentRow** para suprimir la llamada oculta a **AcceptChanges**. En el cuerpo principal del procedimiento, se llama al método **Merge** para combinar los dos objetos **DataTable** y, finalmente, se llama a **AcceptChanges**.

```
Private Shared Sub OnRowUpdated( _
    ByVal sender As Object, ByVal e As OleDbRowUpdatedEventArgs)
    ' Conditionally execute this code block on inserts only.
    If e.StatementType = StatementType.Insert Then
        ' Retrieve the Autonumber and store it in the CategoryID column.
        Dim cmdNewID As New OleDbCommand("SELECT @@IDENTITY", _
           connection)
        e.Row("CategoryID") = CInt(cmdNewID.ExecuteScalar)
        e.Status = UpdateStatus.SkipCurrentRow
    End If
End Sub
```

6.8. Transacciones

Una transacción consiste en un comando único o en un grupo de comandos que se ejecutan como un paquete. Las transacciones permiten combinar varias operaciones en una sola unidad de trabajo. El uso de una transacción le da a la aplicación la posibilidad de anular (revertir) todos los cambios ejecutados desde dentro de la transacción si se produce algún error en algún momento del proceso de transacción.

Una transacción debe ajustarse a las propiedades ACID (atomicidad, coherencia, aislamiento y durabilidad) para poder garantizar la coherencia de datos. La mayoría de los sistemas de bases de datos relacionales, como Microsoft SQL Server, admiten transacciones, al proporcionar funciones de bloqueo, registro y administración de transacciones cada vez que una aplicación cliente realiza una operación de actualización, inserción o eliminación.

Si la base de datos admite transacciones, puede agrupar varias operaciones de base de datos en una sola transacción para evitar incoherencias en la base de datos. Si en un punto de la transacción se produjera un error, todas las actualizaciones podrían revertirse y devolverse al estado que tenían antes de la transacción. Si no se producen errores, las actualizaciones se pueden finalizar mediante la confirmación de la transacción como completada.

Por ejemplo, en una aplicación de banca en la que los fondos se transfieren de una cuenta a otra, en una de las cuentas se ingresa un importe en una tabla de base de datos y en la otra cuenta se carga el mismo importe al mismo tiempo en otra tabla de base de datos. Como los equipos pueden dar error, es posible actualizar una fila de una tabla y no poder actualizar la fila de la otra tabla.

Las transacciones que requieren varios recursos pueden reducir la simultaneidad si la duración del bloqueo es demasiado larga. Por ello, haga la transacción lo más corta posible. Las transacciones explícitas en procedimientos almacenados suelen dar mejores resultados cuando una transacción implica el uso de varias tablas en la misma base de datos o servidor. Se pueden crear transacciones en procedimientos almacenados de SQL Server utilizando Transact-SQL **BEGIN TRANSACTION**, **COMMIT TRANSACTION** o instrucciones **ROLLBACK TRANSACTION**.

Las transacciones que implican varios administradores de recursos, como un a transacción entre SQL Server y Oracle, requieren una transacción distribuida.

6.8.1. Transacciones Locales

Las transacciones de ADO.NET se utilizan cuando se desea enlazar varias tareas para que se ejecuten como una sola unidad de trabajo. Por ejemplo, imagine que una aplicación realiza dos tareas. Primero, actualiza una tabla con información de pedidos. Luego, actualiza una tabla que contiene la información de inventario, cargando en cuenta los elementos pedidos. Si alguna de las tareas da error, ambas actualizaciones se revierten.

⊟ Determinación del tipo de transacción

Una transacción se considera local cuando consta de una única fase y es controlada directamente por la base de datos. Las transacciones se consideran distribuidas cuando se coordinan mediante un monitor de transacciones y utilizan mecanismos a prueba de errores (como confirmación en dos fases) en la resolución de transacciones.

Cada proveedor de datos de .NET Framework tiene su propio objeto **Transaction** para realizar transacciones locales. Si necesita que una transacción se realice en una base de datos de SQL Server, seleccione una transacción de System.Data.SqlClient. En transacciones de Oracle, utilice el proveedor System.Data.OracleClient. Además, existe una nueva clase DbTransaction disponible para la escritura de código independiente del proveedor que requiere transacciones.

Nota:

Las transacciones son más eficientes cuando se realizan en el servidor. Si trabaja con una base de datos de SQL Server que hace uso masivo de transacciones explícitas, debería estudiar la posibilidad de escribirlas como procedimientos almacenados mediante la instrucción BEGIN TRANSACTION de Transact-SQL.

⊟ Realización de una transacción mediante una única conexión

En ADO.NET, las transacciones se controlan con el objeto **Connection**. Puede iniciar una transacción local con el método **BeginTransaction**. Una vez iniciada una transacción, puede inscribir un comando en esa transacción con la propiedad **Transaction** de un objeto **Command**. Luego, puede confirmar o revertir las modificaciones realizadas en el origen de datos según el resultado correcto o incorrecto de los componentes de la transacción.

Nota:

El método **EnlistDistributedTransaction** no se debe emplear en transacciones locales.

El ámbito de la transacción está limitado a la conexión. En el siguiente ejemplo se realiza una transacción explícita que consta de por dos comandos independientes en el bloque **try**. Los comandos ejecutan instrucciones INSERT con respecto a la tabla Production.ScrapReason de la base de datos de ejemplo AdventureWorks de SQL Server 2005, que se confirman si no se produce ninguna excepción. El código del bloque **catch** revierte la transacción si se produce una excepción. Si la transacción se anula o la conexión se cierra antes de que se haya completado la transacción, ésta se revierte automáticamente.

⊟ Procedimientos

Para realizar una transacción

1. Llame al método BeginTransaction del objeto SqlConnection para marcar el comienzo de la transacción. El método BeginTransaction devuelve una referencia a la transacción. Esta referencia se asigna a los objetos SqlCommand que están inscritos en la transacción.

2. Asigne el objeto **Transaction** a la propiedad Transaction del objeto SqlCommand que se va a ejecutar. Si el comando se ejecuta en una conexión con una transacción activa y el objeto **Transaction** no se ha asignado a la propiedad **Transaction** del objeto **Command**, se inicia una excepción.

3. Ejecute los comandos necesarios.

4. Llame al método Commit del objeto SqlTransaction para completar la transacción, o al método Rollback para finalizarla. Si la conexión se cierra o elimina antes de que se hayan ejecutado los métodos Commit o Rollback, la transacción se revierte.

6.8.2. Transacciones Distribuidas

Entre otras cosas, una transacción es un conjunto de tareas relacionadas que se ejecutan correctamente (confirman) o dan error (anulan) como una unidad. Una transacción distribuida es una transacción que afecta a varios recursos. Para que una transacción distribuida se confirme, todos los participantes deben garantizar que los cambios en los datos serán permanentes. Los cambios deben mantenerse a pesar de bloqueos del sistema u otros eventos imprevistos. Si alguno de los participantes no cumple esta garantía, toda la transacción da error y se revertirán los cambios en los datos en el ámbito de la transacción.

Nota:

Si intenta confirmar o revertir una transacción al iniciar un **DataReader** mientras la transacción está activa, se produce una excepción.

⊟ Trabajo con System.Transactions

En .NET Framework, las transacciones distribuidas se administran a través de la API del espacio de nombres System.Transactions. Cuando hay implicados varios administradores de recursos persistentes, la API System.Transactions delegará el control de las transacciones distribuidas en un monitor de transacciones como el Coordinador de transacciones distribuidas de Microsoft (MS DTC).

Una nueva característica de ADO.NET 2.0 es la compatibilidad con la inscripción en una transacción distribuida mediante el método **EnlistTransaction**, que inscribe una conexión en una de instancia

Transaction. En las versiones anteriores de ADO.NET, la inscripción explícita en transacciones distribuidas se realizaba mediante el método **EnlistDistributedTransaction** de una conexión que inscribía ésta en una instancia ITransaction, en la que se permitía la compatibilidad con versiones anteriores.

Cuando se utiliza una transacción System.Transactions con el proveedor de datos de .NET Framework para SQL Server en una base de datos SQL Server 2005, se usará automáticamente una Transaction ligera. A continuación, la transacción se puede promover a una transacción distribuida completa si es necesario.

Nota:

El número máximo de transacciones distribuidas en las que puede participar una base de datos de Oracle a la vez se establece de forma predeterminada en 10. Después de la décima transacción en una conexión a una base de datos de Oracle, se inicia una excepción. Oracle no admite **DDL** en las transacciones distribuidas.

Inscripción automática en una transacción distribuida

La inscripción automática es el método predeterminado (y preferido) de integrar conexiones ADO.NET con **System.Transactions**. Un objeto de conexión se inscribirá automáticamente en una transacción distribuida existente si se determina que hay una transacción activa, que, en términos de **System.Transaction**, significa que **Transaction.Current** no es nula.La inscripción automática en transacciones tiene lugar cuando se abre la conexión. No ocurrirá después, incluso si se ejecuta un comando dentro del ámbito de una transacción. Puede deshabilitar la inscripción automática en transacciones existentes si especifica Enlist=false como un parámetro de cadena de conexión para una ConnectionString, o OLE DB Services=-7 como un parámetro de cadena de conexión para una ConnectionString.

Inscripción manual en una transacción distribuida

Si la inscripción automática está deshabilitada o es necesario inscribir la conexión en una transacción que se ha iniciado después de abierta la conexión, puede inscribirla en una transacción distribuida existente mediante el método **EnlistTransaction** del objeto DbConnection correspondiente al proveedor con el que esté trabajando.La inscripción en una transacción distribuida existente garantiza que, si la transacción se confirma o revierte, también se confirmarán o revertirán las modificaciones realizadas por el código en el origen de datos.

La inscripción en transacciones distribuidas es especialmente conveniente al agrupar objetos empresariales. Si se agrupa un objeto empresarial con una conexión abierta, la inscripción automática en transacciones sólo se produce cuando se abre esa conexión. Si se realizan varias transacciones con el objeto empresarial agrupado, la conexión abierta para ese objeto no se inscribirá automáticamente en las transacciones recién iniciadas. En este caso, puede deshabilitar la inscripción automática de la conexión en la transacción e inscribir la conexión en las transacciones mediante **EnlistTransaction**.

EnlistTransaction acepta un único argumento del tipo Transaction que es una referencia a la transacción existente. Después de llamar al método **EnlistTransaction** de la conexión, todas las modificaciones realizadas en el origen de datos mediante la conexión se incluyen en la transacción. Si se pasa un valor nulo, se anula la inscripción de la conexión de su inscripción actual en transacciones distribuidas. Tenga en cuenta que la conexión se debe abrir antes de llamar a **EnlistTransaction**.

Nota:

Una vez que una conexión se inscribe explícitamente en una transacción, no se puede anular su inscripción ni inscribirse en otra transacción hasta que finaliza la primera transacción.

Precaución:

Si la conexión ya ha comenzado una transacción mediante el método BeginTransaction de la conexión, **EnlistTransaction** inicia una excepción. No obstante, si la transacción es una transacción local iniciada en el origen de datos (por ejemplo, al ejecutar la instrucción BEGIN TRANSACTION de forma explícita mediante un SqlCommand), **EnlistTransaction** la revertirá e inscribirá en la transacción distribuida existente como se ha solicitado. No recibirá aviso de que la transacción local se ha revertido y deberá administrar todas las transacciones locales no iniciadas mediante BeginTransaction. Si utiliza el proveedor de datos de .NET Framework para SQL Server (**SqlClient**) con SQL Server 2005, al intentar una inscripción se producirá una excepción.Todos los demás casos no se detectarán.

Transacciones promocionadas en SQL Server 2005

SQL Server 2005 admite transacciones promocionadas en las que una transacción ligera se puede promover automáticamente a distribuida solamente cuando es necesario. Las transacciones promocionadas no invocan la sobrecarga adicional de las transacciones distribuidas a menos que sea necesario.

Configuración de transacciones distribuidas

Es posible que deba habilitar el MS DTC a través de la red para usar transacciones distribuidas en un sistema operativo más nuevo con los últimos Service Packs aplicados, como Windows XP o Windows Server 2003. Si tiene habilitado Firewall de Windows (valor predeterminado en Windows XP Service Pack 2), debe permitir que el servicio MS DTC use la red o debe abrir el puerto de MS DTC.

6.8.3. Integración de System.Transactions con SQL Server

.NET Framework versión 2.0 incluye un nuevo marco de trabajo de transacciones al que se puede obtener acceso a través del espacio de nombres System.Transactions. Este marco de trabajo expone las transacciones de tal forma que se integra completamente en .NET Framework, incluyendo ADO.NET.

Además de las mejoras de programación, System.Transactions y ADO.NET pueden funcionar juntos para coordinar las optimizaciones al trabajar con transacciones. Una transacción promocionada es una transacción ligera (local) que, en caso necesario, se puede promover automáticamente a una transacción completamente distribuida.

En ADO.NET 2.0, System.Data.SqlClient agrega compatibilidad con transacciones promocionadas al trabajar con SQL Server 2005. Las transacciones promocionadas no invocan la sobrecarga adicional de las transacciones distribuidas a menos que sea necesario. Son automáticas, es decir, no necesitan que intervenga el programador.

Las transacciones promocionadas solo están disponibles cuando se utiliza el proveedor de datos de .NET Framework para SQL Server (**SqlClient**) con SQL Server 2005.

Creación de transacciones promocionadas

El proveedor de .NET Framework para SQL Server ofrece compatibilidad con transacciones promocionadas, que se administran a través de las clases del espacio de nombres System.Transactions de .NET Framework. Las transacciones promocionadas optimizan las transacciones distribuidas ya que aplazan la creación de las mismas hasta que es necesario. Si sólo se necesita un administrador de recursos, no tiene lugar ninguna transacción distribuida.

Nota:

En un caso que no es de plena confianza, se requiere DistributedTransactionPermission cuando la transacción aumenta al nivel de transacción distribuida.

Situaciones de uso de transacciones promocionadas

Normalmente, las transacciones distribuidas consumen muchos recursos del sistema, siendo el encargado de administrarlas Microsoft DTC (Coordinador de transacciones distribuidas), que integra todos los administradores de recursos a los que se tiene acceso en la transacción. Una transacción promocionada es una forma especial de transacción de System.Transactions que delega con efectividad el trabajo en una transacción de SQL Server 2005 simple. System.Transactions, System.Data.SqlClient y SQL Server 2005 coordinan el trabajo que supone administrar la transacción y promoverla a una transacción completamente distribuida cuando es necesario.

La ventaja de utilizar transacciones promocionadas es que cuando se abre una conexión utilizando una transacción TransactionScope activa, y no hay ninguna otra conexión abierta, la transacción se confirma como una transacción ligera, en lugar de incurrir en la sobrecarga adicional de una transacción completamente distribuida.

Palabras clave de cadena de conexión

La propiedad ConnectionString admite una palabra clave, **Enlist**, que indica si System.Data.SqlClient detectará contextos transaccionales e inscribirá automáticamente la conexión en una transacción distribuida. Si Enlist=true, la conexión se inscribe automáticamente en el contexto de transacción actual del subproceso de apertura. Si Enlist=false, la conexión **SqlClient** no interactúa con una transacción distribuida. El valor predeterminado de **Enlist** es true. Si no se especifica **Enlist** en la cadena de conexión, la conexión se da de alta automáticamente en una transacción distribuida si se detecta una al abrirse la conexión.

Las palabras clave **Transaction Binding** en una cadena de conexión SqlConnection controlan la asociación de la conexión con una transacción **System.Transactions** dada de alta. También está disponible mediante la propiedad TransactionBinding de SqlConnectionStringBuilder.

La siguiente tabla describe los posibles valores.

Palabra clave	Descripción
Desenlace implícito	Es la opción predeterminada. La conexión se separa de la transacción cuando termina, y vuelve a cambiar al modo de confirmación automática.
Desenlace explícito	La conexión sigue adjuntada a la transacción hasta que ésta se cierra. La conexión producirá errores si no está activa o no coincide con Current.

Uso de TransactionScope

La clase TransactionScope crea un bloque de código transaccional al inscribir implícitamente las conexiones en una transacción distribuida. Debe llamar al método Complete al final del bloque TransactionScope antes de abandonarlo. Al salir del bloque se invoca el método Dispose. Si se ha producido una excepción que ocasiona que el código salga del ámbito, la transacción se considera anulada.

Se recomienda el uso de un bloque **using** para asegurarse de que se llama a Dispose en el objeto TransactionScope cuando se sale de dicho bloque. Si no se confirman ni revierten las transacciones pendientes, el rendimiento puede verse seriamente afectado ya que el tiempo de espera predeterminado

de TransactionScope es un minuto. Si no utiliza una instrucción **using**, todo el trabajo deberá realizarlo en un bloque **Try** y llamar explícitamente al método Dispose en el bloque **Finally**.

Si se produce una excepción en TransactionScope, la transacción se marca como incoherente y se abandona. Se revertirá cuando se elimine el TransactionScope. Si no se produce ninguna excepción, las transacciones participantes se confirman.

Nota:

La clase **TransactionScope** crea una transacción con un IsolationLevel predeterminado de **Serializable**. Dependiendo de la aplicación, podría estudiar la posibilidad de reducir el nivel de aislamiento para evitar una elevada contención en la aplicación.

Nota:

Se recomienda que sólo realice actualizaciones, inserciones y eliminaciones en transacciones distribuidas, ya que consumen una cantidad considerable de recursos de base de datos. Las instrucciones SELECT pueden bloquear los recursos de base de datos de forma innecesaria y, en algunas situaciones, es posible que tengan que utilizarse transacciones para las selecciones. Todo el trabajo que no sea de base de datos debe realizarse fuera del ámbito de la transacción, a menos que estén implicados otros administradores de recursos de transacción. Aunque una excepción en el ámbito de la transacción impide que se confirme la misma, la clase TransactionScope no deja revertir los cambios que haya realizado el código fuera del ámbito de la propia transacción. Si es necesario realizar alguna acción cuando se revierta la transacción, deberá escribir su propia implementación de la interfaz IEnlistmentNotification y darla de alta explícitamente en la transacción.

Ejemplo

Trabajar con System.Transactions requiere disponer de una referencia a System.Transactions.dll.

La siguiente función muestra cómo crear una transacción promocionada en dos instancias de SQL Server diferentes, representadas por dos objetos SqlConnection diferentes, que se incluyen en un bloque TransactionScope. El código crea el bloque TransactionScope con una instrucción **using** y abre la primera conexión, que automáticamente se da de alta en TransactionScope. La transacción se inscribe inicialmente como una transacción ligera y no como una completamente distribuida. La segunda conexión se inscribe en TransactionScope únicamente si el comando de la primera conexión no produce una excepción. Cuando se abre la segunda conexión, la transacción se promociona automáticamente a una transacción completamente distribuida. Se invoca el método Complete, que confirma la transacción únicamente si no se han producido excepciones. Si en algún punto del bloque TransactionScope se ha producido una excepción, no se llamará a **Complete** y, cuando se elimine TransactionScope al final de su bloque **using**, se revertirá la transacción distribuida.

```
' This function takes arguments for the 2 connection strings and commands
' in order to create a transaction involving two SQL Servers. It returns
' a value > 0 if the transaction committed, 0 if the transaction rolled
' back. To test this code, you can connect to two different databases on
' the same server by altering the connection string, or to another RDBMS
' such as Oracle by altering the code in the connection2 code block.
Public Function CreateTransactionScope( _
  ByVal connectString1 As String, ByVal connectString2 As String, _
  ByVal commandText1 As String, ByVal commandText2 As String) As Integer
    ' Initialize the return value to zero and create a StringWriter to
display results.
    Dim returnValue As Integer = 0
    Dim writer As System.IO.StringWriter = New System.IO.StringWriter
```

```
    ' Create the TransactionScope in which to execute the commands,
    ' guaranteeing that both commands will commit or roll back as a
    ' single unit of work.
    Using scope As New TransactionScope()
        Using connection1 As New SqlConnection(connectString1)
            Try
                ' Opening the connection automatically enlists it in the
                ' TransactionScope as a lightweight transaction.
                connection1.Open()
  ' Create the SqlCommand object and execute the first command.
  Dim command1 As SqlCommand = New SqlCommand(commandText1, connection1)
  returnValue = command1.ExecuteNonQuery()
  writer.WriteLine("Rows to be affected by command1: {0}", returnValue)
        ' If you get here, this means that command1 succeeded. By nesting
        ' the Using block for connection2 inside that of connection1, you
        ' conserve server and network resources by opening connection2
        ' only when there is a chance that the transaction can commit.
                Using connection2 As New SqlConnection(connectString2)
                    Try
                     ' The transaction is promoted to a full distributed
                     ' transaction when connection2 is opened.
                        connection2.Open()
                     ' Execute the second command in the second database.
                        returnValue = 0
Dim command2 As SqlCommand = New SqlCommand(commandText2, connection2)
                        returnValue = command2.ExecuteNonQuery()
writer.WriteLine("Rows to be affected by command2: {0}", returnValue)
                    Catch ex As Exception
                        ' Display information that command2 failed.
          writer.WriteLine("returnValue for command2: {0}", returnValue)
          writer.WriteLine("Exception Message2: {0}", ex.Message)
                    End Try
                End Using
            Catch ex As Exception
                ' Display information that command1 failed.
         writer.WriteLine("returnValue for command1: {0}", returnValue)
                writer.WriteLine("Exception Message1: {0}", ex.Message)
            End Try
        End Using
        ' If an exception has been thrown, Complete will
        ' not be called and the transaction is rolled back.
        scope.Complete()
    End Using
    ' The returnValue is greater than 0 if the transaction committed.
    If returnValue > 0 Then
        writer.WriteLine("Transaction was committed.")
    Else
  ' You could write additional business logic here, notify the caller by
  ' throwing a TransactionAbortedException, or log the failure.
      writer.WriteLine("Transaction rolled back.")
```

```
     End If
    ' Display messages.
    Console.WriteLine(writer.ToString())
    Return returnValue
End Function
```

6.9. Modificar Datos con Procedimientos Almacenados

Los procedimientos almacenados pueden aceptar datos como parámetros de entrada y pueden devolver datos como parámetros de salida, conjuntos de resultados o valores de retorno. En el ejemplo siguiente se muestra cómo ADO.NET envía y recibe parámetros de entrada, parámetros de salida y valores de retorno. El ejemplo inserta un nuevo registro en una tabla cuya columna de clave principal es una columna de identidad en una base de datos de SQL Server.

Nota:

Si está utilizando procedimientos almacenados de SQL Server para editar o eliminar datos con SqlDataAdapter, asegúrese de que no utiliza SET NOCOUNT ON en la definición del procedimiento almacenado. Esto hace que el recuento de filas afectadas vuelva a cero, lo que **DataAdapter** interpreta como un conflicto de concurrencia. En este caso, se iniciará una DBConcurrencyException.

Ejemplo

En el ejemplo se utiliza el siguiente procedimiento almacenado para insertar una nueva categoría en la tabla **Categories** de **Northwind**. El procedimiento almacenado recibe el valor de la columna **CategoryName** como parámetro de entrada y usa la función SCOPE_IDENTITY() para recuperar el nuevo valor del campo de identidad, **CategoryID**, y devolverlo en un parámetro de salida. La instrucción RETURN utiliza la función @@ROWCOUNT para devolver el número de filas insertadas.

```
CREATE PROCEDURE dbo.InsertCategory
  @CategoryName nvarchar(15),
  @Identity int OUT
AS
INSERT INTO Categories (CategoryName) VALUES(@CategoryName)
SET @Identity = SCOPE_IDENTITY()
RETURN @@ROWCOUNT
```

En el siguiente ejemplo de código se utiliza el anterior procedimiento almacenado InsertCategory como origen de la propiedad InsertCommand de SqlDataAdapter. El parámetro de salida @Identity se reflejará en DataSet una vez que se haya insertado el registro en la base de dados al llamar al método **Update** del SqlDataAdapter. El código también recupera el valor devuelto.

Nota:

En el caso de OleDbDataAdapter, los parámetros con un ParameterDirection con el valor **ReturnValue** se deben especificar antes que los restantes parámetros.

```
Private Sub ReturnIdentity(ByVal connectionString As String)
    Using connection As SqlConnection = New SqlConnection( _
       connectionString)
        ' Create a SqlDataAdapter based on a SELECT query.
        Dim adapter As SqlDataAdapter = New SqlDataAdapter( _
           "SELECT CategoryID, CategoryName FROM dbo.Categories", _
           connection)
```

```
        ' Create a SqlCommand to execute the stored procedure.
        adapter.InsertCommand = New SqlCommand("dbo.InsertCategory", _
           connection)
        adapter.InsertCommand.CommandType = CommandType.StoredProcedure
        ' Create a parameter for the ReturnValue.
        Dim parameter As SqlParameter = _
           adapter.InsertCommand.Parameters.Add( _
          "@RowCount", SqlDbType.Int)
        parameter.Direction = ParameterDirection.ReturnValue
        ' Create an input parameter for the CategoryName.
        ' You do not need to specify direction for input parameters.
        adapter.InsertCommand.Parameters.Add( _
          "@CategoryName", SqlDbType.NChar, 15, "CategoryName")
        ' Create an output parameter for the new identity value.
        parameter = adapter.InsertCommand.Parameters.Add( _
          "@Identity", SqlDbType.Int, 0, "CategoryID")
        parameter.Direction = ParameterDirection.Output
        ' Create a DataTable and fill it.
        Dim categories As DataTable = New DataTable
        adapter.Fill(categories)
        ' Add a new row.
        Dim newRow As DataRow = categories.NewRow()
        newRow("CategoryName") = "New Category"
        categories.Rows.Add(newRow)
        ' Update the database.
        adapter.Update(categories)
        ' Retrieve the ReturnValue.
        Dim rowCount As Int32 = _
           CInt(adapter.InsertCommand.Parameters("@RowCount").Value)
        Console.WriteLine("ReturnValue: {0}", rowCount.ToString())
        Console.WriteLine("All Rows:")
        Dim row As DataRow
        For Each row In categories.Rows
            Console.WriteLine("  {0}: {1}", row(0), row(1))
        Next
    End Using
End Sub
```