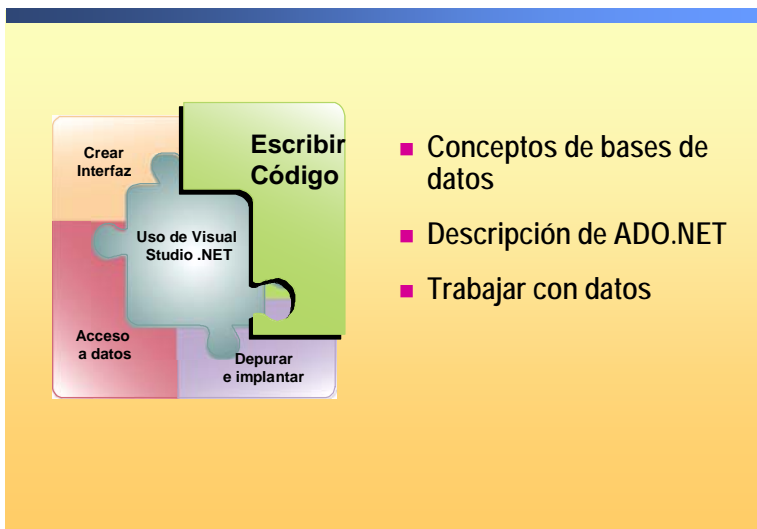


Acceso a datos con ADO.NET

Índice

Descripción	1
Lección: conceptos de bases de datos	2
Lección: descripción de ADO.NET	14
Lección: trabajar con datos	29
Acceder a datos con DataReaders	43

Descripción



Introducción

Este módulo explica cómo utilizar Microsoft® ADO.NET con una aplicación Microsoft Windows® Forms para crear, leer, actualizar y eliminar registros de bases de datos Microsoft SQL Server™ y Microsoft Access (Jet 4.0).

Objetivos

En este módulo, aprenderemos a:

- Definir la terminología básica sobre bases de datos, incluyendo los conceptos de base de datos, tabla, registro, campo y clave.
- Describir algunos objetos ADO.NET utilizados habitualmente.
- Crear y abrir una conexión a una base de datos.
- Crear, leer, actualizar y eliminar registros de una base de datos.
- Utilizar el Asistente para formularios de datos para crear una sencilla aplicación de acceso a datos.
- Mostrar y modificar datos extraídos de una base de datos.

Lección: conceptos de bases de datos

- Terminología de las bases de datos
- Cómo funciona la programación de bases de datos
- ¿Qué es SQL?
- ¿Qué es un entorno conectado?
- ¿Qué es un entorno desconectado?

Introducción

Esta lección ofrece una descripción general de los conceptos básicos de bases de datos. Hace una introducción al lenguaje de consultas estructurado (*Structured Query Language*, SQL) y explica la diferencia entre entornos de datos conectados y desconectados.

Objetivos de la lección

En esta lección, aprenderemos a:

- Describir los elementos básicos de las bases de datos, como base de datos, tabla, registro, campo y clave.
- Explicar el proceso general de acceso a datos.
- Utilizar sentencias básicas de SQL.
- Explicar la diferencia entre un entorno conectado y uno desconectado.
- Utilizar el Explorador de servidores para visualizar elementos de las bases de datos.

Estructura de la lección

Esta lección incluye los siguientes temas y actividades:

- Terminología de las bases de datos
- ¿Cómo funciona la programación de bases de datos?
- ¿Qué es SQL?
- ¿Qué es un entorno conectado?
- ¿Qué es un entorno desconectado?
- Práctica: identificar escenarios de datos conectados o desconectados

Terminología de las bases de datos

El diagrama muestra una tabla titulada "Tabla de empleados" con tres columnas: "ID de empleado", "Apellido" y "Nombre". Las primeras dos filas están completadas con los datos de Tony Small (ID 3) y James Smith (ID 5). Las siguientes tres filas contienen solo puntos, representando registros adicionales. A la izquierda de la tabla, el texto "Filas (registros)" apunta a las filas de datos. Debajo de la tabla, el texto "Columnas (campos)" apunta a las columnas de encabezado. En la parte inferior izquierda, un recuadro titulado "Relaciones" muestra dos tablas, "Empleados" y "Pedidos", con flechas que indican relaciones entre ellas.

ID de empleado	Apellido	Nombre
3	Small	Tony
5	Smith	James
.	.	.
.	.	.
.	.	.

Introducción

Para trabajar con bases de datos, es importante conocer cierta terminología básica.

Definiciones

Los siguientes términos se definen en el contexto de las bases de datos relacionales.

■ Base de datos relacional

Una *base de datos relacional* es un tipo de base de datos que almacena información en tablas. Las bases de datos relacionales utilizan valores coincidentes de dos tablas para relacionar datos en una tabla con datos de otra. En una base de datos relacional, normalmente almacenamos un tipo específico de datos sólo una vez.

■ Tabla

Una *tabla* es un objeto base de datos que almacena datos en registros (filas) y campos (columnas). Normalmente, los datos tienen relación con una categoría concreta de cosas, como empleados o pedidos.

■ Registro

Un *registro* es una colección de datos sobre una persona, un lugar, un evento o algún otro elemento. Los registros son los equivalentes lógicos de filas en una tabla. Por ejemplo, un registro en la tabla Empleados debería tener información sobre un empleado particular.

■ Campo

Un registro está compuesto de varios campos. Cada *campo* de un registro contiene una pieza de información sobre el registro. Por ejemplo, el registro de un Empleado tiene campos para el ID del empleado, Apellido, Nombre, etc.

- Clave principal

Una *clave principal* identifica de modo único cada fila de una tabla. La clave principal es un campo o una combinación de campos cuyo valor es único para cada fila (o registro) de la tabla. Por ejemplo, el campo *Employee ID* es la clave primaria para la tabla Empleados. No puede haber dos empleados con el mismo ID.

- Clave foránea

Una *clave foránea* es uno o más campos (columnas) de una tabla que hacen referencia al campo o campos de la clave principal de otra tabla. Una clave foránea indica cómo están relacionadas las tablas.

- Relación

Una *relación* es una asociación establecida entre campos comunes (columnas) de dos tablas. Una relación puede ser de uno a uno, de uno a muchos, o de muchos a muchos. Las relaciones permiten que los resultados de las consultas incluyan datos de varias tablas. Una relación uno a uno entre una tabla Clientes y una tabla Pedidos permitiría que una consulta devolviera todos los pedidos de un determinado cliente.

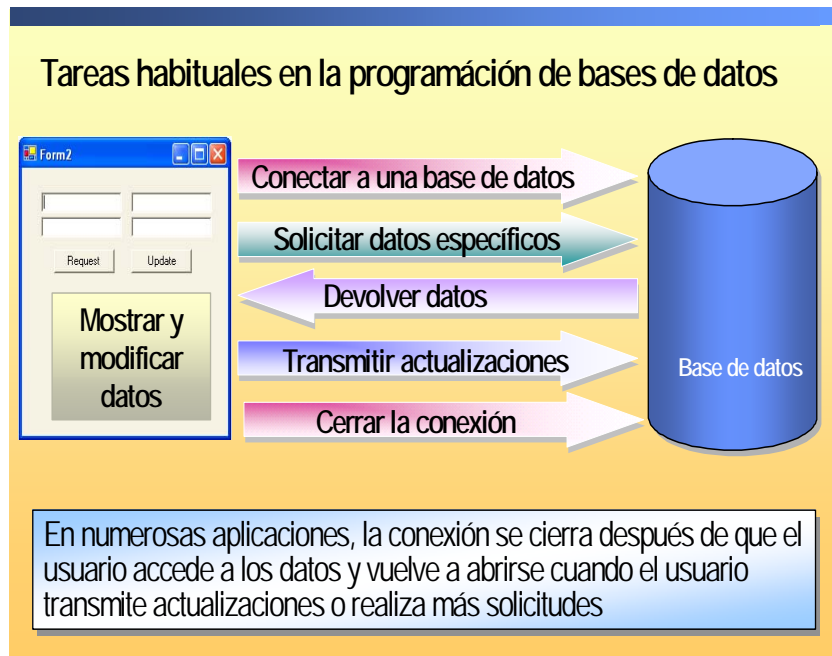
- Acceso de sólo lectura

El *acceso de sólo lectura* a una base de datos significa que podemos recuperar (leer) los datos pero no podemos modificarlos (escribir). Un archivo de sólo lectura o un documento puede visualizarse o imprimirse pero no puede modificarse de ningún modo.

- Acceso de lectura/escritura

El *acceso de lectura/escritura* a una base de datos significa que podemos recuperar (leer) los datos y modificarlos (escribir).

Cómo funciona la programación de bases de datos



Introducción

Cuando desarrollamos aplicaciones, tenemos diferentes requerimientos para trabajar con datos. En algunos casos, simplemente desearemos mostrar datos en un formulario. En otros casos, es posible que necesitemos crear una forma de compartir información con otra organización.

Principales tareas de la programación de bases de datos

Cuando trabajamos con datos, utilizamos varios objetos para recuperar y modificar información de una base de datos. En general, cuando trabajamos con bases de datos en ADO.NET, realizamos las siguientes tareas:

1. Conectar a una base de datos.
2. Solicitar datos específicos.
Especificar los datos que se desean recuperar y si se necesita acceso de solo lectura o de lectura/escritura a los datos.
3. Recuperar y mostrar los datos.
4. Cerrar la conexión (en algunas aplicaciones).
5. Modificar los datos recuperados (si se dispone de acceso de lectura/escritura).
6. Volver a abrir la conexión (en algunas aplicaciones).
7. Transmitir a la base de datos los cambios realizados en los datos.
8. Cerrar la conexión.

En lecciones posteriores de este módulo, aprenderemos más sobre los objetos y herramientas específicos que podemos utilizar en ADO.NET para trabajar con datos.

¿Qué es SQL?

Definición: SQL es un lenguaje estándar del mercado que ha evolucionado hasta convertirse en el medio de mayor aceptación para realizar consultas y modificar datos de una base de datos

■ **Sintaxis de instrucciones SQL habituales**

- Para especificar exactamente qué registros se desea recuperar, utilizar

`SELECT Campo FROM Tabla`

- Para limitar la selección de registros, utilizar

`SELECT * FROM Tabla WHERE Campo = "String"`

- Para devolver registros en orden ascendente, utilizar

`SELECT * FROM Tabla ORDER BY Campo ASC`

■ **Ejemplo**

`SELECT Nombre FROM Empleados`

Introducción

Antes de que podamos escribir una aplicación que utilice datos, necesitamos saber algo sobre el Lenguaje de consulta estructurado (*Structured Query Language*, SQL). Utilizamos SQL para especificar exactamente qué registros recuperar de una base de datos.

Definición

SQL es un lenguaje estándar de mercado que ha evolucionado hasta convertirse en el medio de mayor aceptación para realizar consultas y modificar datos de una base de datos.

Nota SQL está definido por el estándar American National Standards Institute (ANSI). Existen diferentes implementaciones de SQL para diferentes motores de bases de datos. Por ejemplo, existe una versión de SQL para bases de datos Microsoft Access (Jet 4.0) y una para SQL Server.

La instrucción SELECT

Utilizando SQL, podemos especificar exactamente qué registros deseamos recuperar y el orden en que los deseamos. Podemos crear una instrucción SQL que recupere información de múltiples tablas a la vez, o podemos crear una instrucción que recupere un registro específico.

La instrucción SELECT devuelve campos específicos de una o más tablas de una base de datos. La siguiente instrucción recupera el nombre de todos los registros de la tabla Empleados:

```
SELECT Nombre FROM Empleados
```

Para seleccionar todos los campos de la tabla, se utiliza un asterisco (*):

```
SELECT * FROM Empleados
```

Ejemplo con Visual Basic .NET

Cuando utilizamos instrucciones SQL en Microsoft Visual Basic® .NET, debemos poner comillas en las instrucciones, como muestra el siguiente código:

```
Dim sqlString As String  
sqlString = "SELECT * FROM Empleados"
```

Cláusula WHERE

Podemos utilizar la cláusula WHERE para limitar la selección de registros utilizando múltiples campos para filtrar consultas. El siguiente código muestra cómo utilizar la cláusula básica WHERE para recuperar todos los registros de la tabla Empleados cuyo apellido sea igual a "Danseglio":

```
SELECT * FROM Empleados WHERE Apellido = 'Danseglio'
```

Existen otras formas de la cláusula WHERE que podemos utilizar para depurar más nuestras consultas, como sigue:

- **Cláusula WHERE IN**

Podemos utilizar la cláusula WHERE IN para obtener todos los registros que satisfacen criterios específicos. Por ejemplo, podemos devolver apellidos de empleados en determinados estados, como muestra el siguiente código:

```
SELECT Apellido  
FROM Empleados  
WHERE Empleados.Estado IN ('NY', 'WA')
```

- **Cláusula WHERE BETWEEN**

También podemos devolver una selección de registros que se encuentren entre dos criterios específicos. Observe el uso de signo de almohadilla (#) enmarcando las fechas en el siguiente código:

```
SELECT OrderID  
FROM Orders  
WHERE OrderDate BETWEEN #01/01/93# AND #01/31/93#
```

- **Cláusula WHERE LIKE**

Podemos utilizar la cláusula WHERE LIKE para obtener todos los registros en los que exista una condición concreta en un campo específico. Por ejemplo, el siguiente código muestra cómo obtener todos los registros en los que el apellido empiece con la letra "D."

```
SELECT Apellido FROM Empleados WHERE Apellido LIKE 'D%'
```

Cláusula ORDER BY

Podemos utilizar la cláusula ORDER BY para obtener registros en un orden determinado. La opción ASC indica orden ascendente. La opción DESC indica orden descendente. El siguiente código devuelve todos los campos de la tabla Empleados y los ordena por el apellido:

```
SELECT * FROM Empleados ORDER BY Apellido ASC
```


¿Qué es un entorno conectado?

- Un entorno conectado es aquel en que los usuarios están conectados de forma continuada a una fuente de datos
- Ventajas:
 - El entorno es más fácil de mantener
 - La concurrencia se controla más fácilmente
 - Es más probable que los datos sean actuales que en otros escenarios
- Inconvenientes:
 - Debe tener una conexión de red constante
 - Escalabilidad limitada

Introducción

Un entorno conectado es aquel en el que un usuario o una aplicación están conectados continuamente a una fuente de datos. Durante muchos años de la historia de la informática, el único entorno disponible era el entorno conectado.

Ventajas

Un escenario conectado proporciona las siguientes ventajas:

- Un entorno conectado es más fácil de mantener.
- La concurrencia se controla más fácilmente.
- Es más probable que los datos estén más actualizados que en un escenario desconectado.

Inconvenientes

Un escenario conectado tiene los siguientes inconvenientes:

- Debe mantenerse una conexión de red constante.
- Un escenario conectado proporciona una escalabilidad limitada.

Ejemplos

He aquí algunos ejemplos en los que debe utilizarse una conexión continua:

- Una fábrica que requiere una conexión en tiempo real para controlar la salida de producción y el almacén.
- Un agente de bolsa que requiere una conexión constante a los valores del mercado.

¿Qué es un entorno desconectado?

- Un entorno desconectado es aquel en el que los datos pueden modificarse independientemente y los cambios se escriben en la base de datos posteriormente
- Ventajas:
 - Las conexiones se utilizan durante el menor tiempo posible, permitiendo que menos conexiones den servicio a más usuarios
 - Un entorno desconectado mejora la escalabilidad y el rendimiento de las aplicaciones
- Inconvenientes:
 - Los datos no siempre están actualizados
 - Pueden producirse conflictos de cambios y deben solucionarse

Introducción

Con la llegada de Internet, los entornos desconectados se han convertido en algo habitual, y con el creciente uso de dispositivos de mano, los escenarios desconectados se están convirtiendo en algo casi universal. Equipos portátiles, Pocket PCs y Tablet PCs permiten utilizar aplicaciones sin conexión a los servidores o a las bases de datos.

En numerosas situaciones, la gente no trabaja en entornos totalmente conectados o desconectados, sino en un entorno que combina ambas opciones.

Definición

Un entorno desconectado es aquel en el que un usuario o una aplicación no están conectados constantemente a una fuente de datos. Las aplicaciones de Internet utilizan frecuentemente arquitecturas desconectadas. Se abre la conexión, se recuperan los datos y la conexión se cierra. El usuario trabaja con los datos en el navegador y la conexión vuelve a abrirse para actualizar u otras peticiones.

Los usuarios móviles que trabajan con equipos portátiles son también los usuarios principales de los entornos desconectados. Los usuarios pueden llevarse un subconjunto de datos en un equipo desconectado y posteriormente fusionar los cambios con el almacén de datos central.

Ventajas

Un entorno desconectado proporciona las siguientes ventajas:

- Las conexiones se utilizan durante el menor tiempo posible, permitiendo que menos conexiones den servicio a más usuarios.
- Un entorno desconectado mejora la escalabilidad y el rendimiento de las aplicaciones, maximizando la disponibilidad de conexiones.

Inconvenientes

Un entorno desconectado tiene los siguientes inconvenientes:

- Los datos no siempre están actualizados.
- Pueden producirse conflictos de cambios que deben solucionarse.

Ejemplos

- La transmisión de datos puede percibirse más lenta de lo que sería en entornos conectados.

He aquí algunos ejemplos en los que podría ser apropiado un entorno desconectado:

- Una aplicación que mantiene datos de clientes en un equipo portátil de un representante.
- Una aplicación que hace un seguimiento de lluvias y precipitaciones.
- Una aplicación que un granjero utiliza para contar el ganado. La aplicación está en el dispositivo basado en Microsoft Windows CE del granjero que ejecuta Microsoft SQL Server 2000 Windows CE Edition.


Gestión de actualizaciones en una aplicación desconectada

En un entorno desconectado, varios usuarios pueden modificar los datos de los mismos registros al mismo tiempo; por ello, nuestra aplicación debe gestionar conflictos en las actualizaciones de datos. Existen varias formas para gestionarlos:

- Permitir que prevalezcan las actualizaciones más recientes.
- Permitir que prevalezcan las primeras actualizaciones realizadas.
- Escribir código en la aplicación que permita a los usuarios determinar qué cambios deberían conservarse. Las soluciones específicas pueden variar dependiendo de los requerimientos de negocio de una determinada aplicación.

Nota Si desea más información sobre la gestión de conflictos, consulte “Introducción a la concurrencia de datos en ADO.NET” en la documentación de Microsoft Visual Studio® .NET.

Práctica: identificar escenarios de datos conectados o desconectados



■ En esta práctica,

- Trabajaremos en parejas para analizar cinco escenarios de negocio en los cuales se requiere acceso a datos
- Para cada escenario, escogeremos un entorno conectado o desconectado, dependiendo de los requerimientos de la aplicación
- Determinaremos si se requiere acceso de solo lectura o de lectura/escritura

Introducción

En esta práctica, trabajaremos en parejas cinco escenarios y determinaremos si la conveniencia de implementarlos en un escenario conectado o desconectado. También determinaremos si el escenario requiere acceso de sólo lectura o de lectura/escritura.

Instrucciones

1. Analizar cada escenario.
2. Escoger un entorno conectado o desconectado, dependiendo de los requerimientos de la aplicación.
3. Determinar si se requiere acceso de sólo lectura o de lectura/escritura.

Escenario 1

Una empresa de inversión financiera necesita un sistema que permita a los clientes utilizar Internet para hacer un seguimiento de sus carteras de valores. La empresa necesita ofrecer servicios como gráficos de valores históricos, el historial de la cuenta, detalles de transacciones particulares, y análisis limitados de las inversiones.

Desconectado, sólo lectura. Los clientes únicamente accederán a datos históricos, y no realizarán transacciones a través de Internet.

Escenario 2

Un grupo de científicos está realizando un experimento que requiere un sistema que almacene y muestre datos de 50 participantes. La aplicación debe medir de forma continuada respuestas fisiológicas a estímulos que son coordinados por los responsables del experimento. Una limitación del experimento es que los científicos deben detener manualmente los estímulos durante una hora cuando el sistema les avisa de que el 60 por ciento de los participantes han llegado al nivel 8 en 4 de 10 medidas.

Conectado, lectura/escritura. Los datos deben actualizarse constantemente y totalizarse, lo cual requiere un entorno conectado. Los datos en intervalos de tiempo recientes deben calcularse sobre los niveles actuales, lo que requiere escribir los datos además de leerlos.

Escenario 3

Debe crearse un sistema para una empresa de revelado que permita a sus clientes enviar fotografías escaneadas o enviadas por correo electrónico, pagar por los servicios a través de Internet, y comprobar el estado de sus pedidos.

Desconectado, lectura/escritura. El examen de los pedidos es de sólo lectura. La creación de nuevos pedidos, incluyendo el envío de fotografías, es de lectura/escritura.

Escenario 4

Debe crearse un sistema para un equipo de ventas móvil. Cada vendedor debe poder conectarse a la red de área extensa de la oficina, a la intranet de la compañía y a Internet utilizando dispositivos de comunicaciones móviles. Los vendedores son responsables de crear y enviar los pedidos de los clientes fuera de la oficina, tras verificar la disponibilidad de inventario y la disponibilidad de crédito. También necesitarán comprobar los envíos a los clientes y el estado de los pedidos, y enviar informes diarios con independencia de que puedan o no establecer una conexión a Internet. Los requerimientos del sistema especifican que las deducciones y créditos del inventario y los envíos deben producirse dos veces al día. Los vendedores deben poder verificar, pero no cambiar, el saldo de las cuentas de los clientes.

Desconectado, acceso de lectura/escritura para procesar los datos de pedidos, envíos e inventario incluso en caso de conexiones telefónicas a través de módems en teléfonos móviles. Acceso de sólo lectura para la información sobre el crédito de los clientes.

Escenario 5

Una empresa de inversión financiera necesita acceder a los datos de tres mercados bursátiles distintos y a una base de datos interna que realiza el seguimiento de las transacciones de los inversores para sus clientes. La base de datos interna recibe descargas continuas del flujo de datos del mercado bursátil. Los datos de los clientes de cada inversor son replicados localmente en su disco duro del PC. El inversor utiliza estos datos para mantener cuentas para dos o tres clientes, planificar inversiones proyectadas, y completarlas cada día. ¿Qué entorno de datos y permisos de acceso se requieren entre los PCs particulares de los inversores y la base de datos interna?

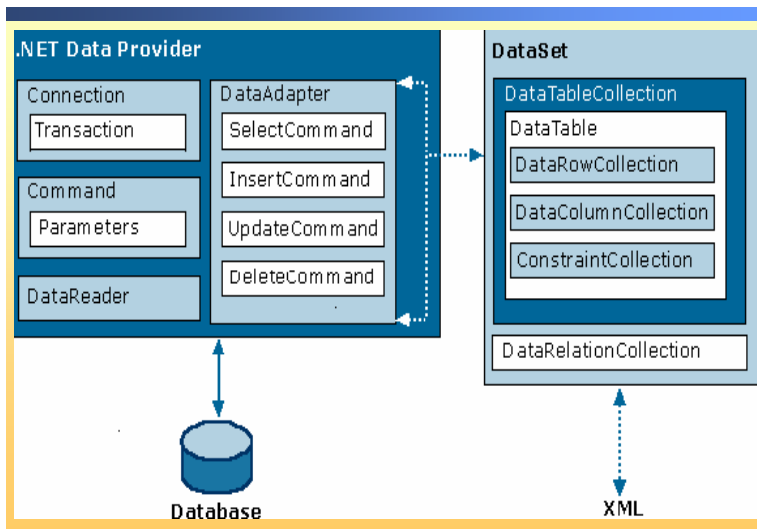
Desconectado, lectura/escritura. Los inversores particulares pueden obtener actualizaciones en microsegundos de los datos del mercado bursátil. Este escenario requeriría un entorno desconectado con acceso de sólo lectura. La base de datos interna y los inversores particulares también funcionarían en un entorno desconectado pero que se actualizará con rapidez, requiriendo la capacidad de lectura/escritura.

Lección: descripción de ADO.NET

- ¿Qué es ADO .NET?
- Objetos comunes de ADO.NET
- Cómo trabajar con bases de datos en Server Explorer
- ADO.NET y XML

Introducción	Esta lección ofrece una descripción de ADO.NET y de su papel principal en aplicaciones que trabajan con bases de datos.
Objetivos de la lección	<p>En esta lección, aprenderemos a:</p> <ul style="list-style-type: none">■ Describir los principales componentes de la arquitectura de ADO.NET.■ Explicar cómo trabajar con datos utilizando objetos en el modelo de objetos de ADO.NET.■ Describir el soporte de XML en ADO.NET.
Estructura de la lección	<p>Esta lección incluye los siguientes temas y actividades:</p> <ul style="list-style-type: none">■ ¿Qué es ADO.NET?■ Objetos comunes de ADO.NET■ Demostración del instructor: Uso del Explorador de servidores■ Cómo trabajar con bases de datos en el Explorador de servidores■ ADO.NET y XML■ Práctica: acceso a datos de sólo lectura

¿Qué es ADO.NET?



Introducción

Con independencia de lo que hagamos con los datos, hay ciertos conceptos fundamentales que debemos conocer sobre los datos en ADO.NET. Es posible que nunca necesitemos conocer algunos de los detalles de la gestión de datos, pero es útil conocer la arquitectura de los datos en ADO.NET, cuáles son los principales componentes de datos, y cómo se acoplan las piezas. Este tema no proporciona detalles muy elaborados, sino que es una introducción a los conceptos relacionados con la integración de datos en ADO.NET.

Definición

ADO.NET es un conjunto de clases para trabajar con datos. Proporciona:

- Un sistema diseñado para entornos desconectados.
- Un modelo de programación con soporte avanzado de XML.
- Un conjunto de clases, interfaces, estructuras y enumeraciones que gestionan el acceso a datos desde el .NET Framework.

Componentes de ADO.NET

Los componentes de ADO.NET han sido diseñados para separar el acceso a datos de la manipulación de los datos. Existen dos componentes principales de ADO.NET que lo cumplen: el componente **DataSet** y los proveedores de datos .NET. Los proveedores de datos .NET constan de un conjunto de componentes que incluyen los objetos **Connection**, **Command**, **DataReader** y **DataAdapter**. Los componentes del proveedor de datos .NET están diseñados explícitamente para la manipulación de datos desconectados.

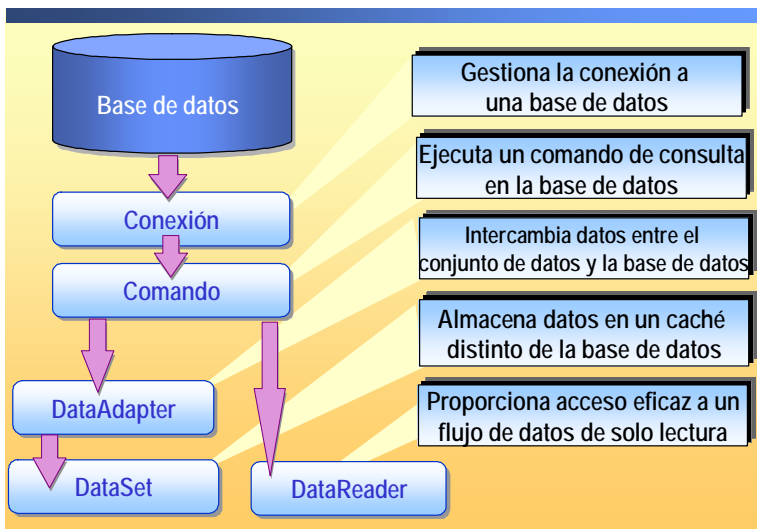
ADO.NET y Windows Forms proporcionan componentes para el consumidor de datos que podemos utilizar para mostrar nuestros datos. Incluyen controles como **DataGrid**, que pueden ser enlazados a datos, y propiedades de enlace a datos en la mayoría de controles estándares de Windows, como los controles **TextBox**, **Label**, **ComboBox** y **ListBox**.

El .NET Framework incluye numerosos proveedores de datos .NET, incluyendo el proveedor de datos de .NET para SQL Server, el proveedor de datos de .NET OLE DB para SQL, y el proveedor OLE DB para Microsoft Jet. Si necesitamos un proveedor personalizado, podemos escribir proveedores de datos .NET para cualquier fuente de datos. En este módulo, nos centraremos en el proveedor de datos de .NET OLE DB para SQL.

Nota Si deseamos obtener más información sobre los proveedores de datos incluidos en .NET, consultar “Utilizar proveedores de datos de .NET para obtener acceso a datos” en la documentación de Visual Studio .NET.

Si deseamos obtener más información sobre la creación de proveedores personalizados, consultar “Ejemplo de proveedor de datos de .NET, Implementación de Visual Basic” en la documentación de Visual Studio .NET.

Objetos comunes de ADO.NET



Introducción

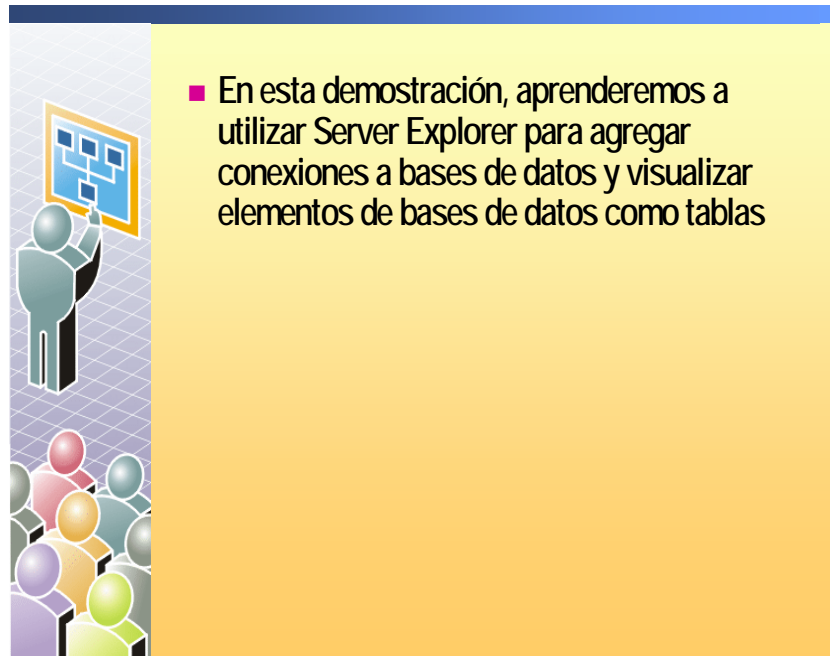
Los proveedores de datos de .NET y el espacio de nombres **System.Data** proporcionan los objetos ADO.NET que utilizaremos en un escenario desconectado.

Objetos ADO.NET

ADO.NET proporciona un modelo de objetos común para proveedores de datos de .NET. La siguiente tabla describe los principales objetos ADO.NET que utilizaremos en un escenario desconectado.

Objeto	Descripción
Connection	Establece y gestiona una conexión a una fuente de datos específica. Por ejemplo, la clase OleDbConnection se conecta a fuentes de datos OLE DB.
Command	Ejecuta un comando en una fuente de datos. Por ejemplo, la clase OleDbCommand puede ejecutar instrucciones SQL en una fuente de datos OLE DB.
DataSet	Diseñado para acceder a datos con independencia de la fuente de datos. En consecuencia, podemos utilizarlo con varias y diferentes fuentes de datos, con datos XML, o para gestionar datos locales a la aplicación. El objeto DataSet contiene una colección de uno o más objetos DataTable formados por filas y columnas de datos, además de clave principal, clave foránea, restricciones e información de la relación sobre los datos en los objetos DataTable .
DataReader	Proporciona un flujo de datos eficaz, <i>sólo-reenvío</i> y de <i>sólo-lectura</i> desde una fuente de datos.
DataAdapter	Utiliza los objetos Connection , Command y DataReader implícitamente para poblar un objeto DataSet y para actualizar la fuente de datos central con los cambios efectuados en el DataSet . Por ejemplo, OleDbDataAdapter puede gestionar la interacción entre un DataSet y una a base de datos Access.

Demostración: Uso del Explorador de servidores



En esta demostración, aprenderemos cómo utilizar el Explorador de servidores para agregar conexiones a bases de datos y visualizar elementos de bases de datos como tablas.

⚡ Crear un nuevo proyecto

1. Iniciar un nuevo proyecto de Microsoft Visual Basic® .NET basado en la plantilla *Windows Application*.
2. Poner el nombre **TitlesDataGrid** al proyecto, establecer la ubicación, y hacer clic en **Aceptar**.

⚡ Mostrar las capacidades de Server Explorer utilizando la tabla *Titles*

1. En el Explorador de servidores, en Servidores, expandir el nodo *nombre_del_pc*, expandir el nodo **Servidores SQL Server**, y expandir el nodo *nombre_del_pc*/NETSDK para que se muestren todas las bases de datos SQL disponibles en el equipo.
2. Expandir el nodo de la base de datos **Pubs**, expandir el nodo **Tablas** bajo *Pubs*, y expandir los campos bajo la tabla *Titles*.
3. Hacer clic con el botón derecho en la tabla **Titles** y explicar las opciones que aparecen.
4. Hacer clic en **Recuperar datos de la tabla**, y mostrar los datos de los campos de la tabla en la ventana de resultados. Explicar que los datos pueden actualizarse aquí, y que se aplicarán las limitaciones impuestas por el diseño de la tabla SQL. Cerrar la ventana de resultados.
5. Hacer clic en varios campos de la tabla *Titles* en el Explorador de servidores y explicar de qué forma la información sobre el tipo y longitud de los datos

que se muestra en la ventana *Propiedades* puede resultar útil para crear una aplicación.

✚ Agregar Connection y el DataAdapter al formulario

1. En el Explorador de servidores, arrastrar la tabla *Titles* a *Form1*.
Cuando se arrastre la tabla al formulario, se crearán **SqlConnection1** y **SqlDataAdapter1** y se añadirán a la bandeja de componentes.
2. En la ventana *Propiedades*, cambiar la propiedad **Name** de **SqlConnection1** por **PubsConn** y cambiar la propiedad **Name** de **SqlDataAdapter1** por **PubsSqlDataAdapter**.
3. Expandir la propiedad **InsertCommand** de **PubsSqlDataAdapter**.
Verificar que la propiedad **Connection** está establecida en **PubsConn**.
Explicar que el objeto de conexión es la fuente de datos de **DataAdapter**.
También puede explicarse que **DataAdapter** gestiona los comandos SQL **Select**, **Insert**, **Delete** y **Update**.

✚ Generar el DataSet para la tabla Products

4. En la ventana *Propiedades*, en el área de descripción de la parte inferior de la lista de propiedades, hacer clic en **Generar Dataset**.
5. En el cuadro de diálogo **Generar Dataset**, escribir **PubsDataSet** en el cuadro de texto **Nuevo**.
6. En el resto de cuadros de texto, dejar los valores predeterminados. Hacer clic en **Aceptar**.

PubsDataSet1 debería aparecer en la bandeja de componentes.

✚ Agregar un DataGrid al formulario y establecer sus propiedades

1. En la ficha **Windows Forms** del Cuadro de herramientas, agregar un control **DataGrid** al formulario.
2. En la ventana *Propiedades*, poner al control **DataGrid** el nombre **TitlesDataGrid**.
3. En la ventana *Propiedades*, establecer la propiedad **DataSource** del control **DataGrid** en **PubsDataSet1**, y establecer la propiedad **DataMember** para **Titles**.
4. Ajuste el tamaño del formulario y el control **DataGrid** para que todas las filas y columnas estén visibles.

La rejilla de datos debería mostrar los nombres de los campos de la tabla *Titles*.

✚ Poblar el DataSet con los datos de la tabla Titles

- Abrir un controlador de eventos para el evento **Form1_Load**. En el controlador de eventos **Form1_Load**, invocar el método **Fill** del adaptador de datos para poblar la tabla *Titles* con datos. El código debería ser como el siguiente:

```
PubsSqlDataAdapter.Fill(PubsDataSet1)
```

✚ Probar la aplicación

1. Ejecutar la aplicación.

Los datos de *Titles* deberían poblar la rejilla de datos.

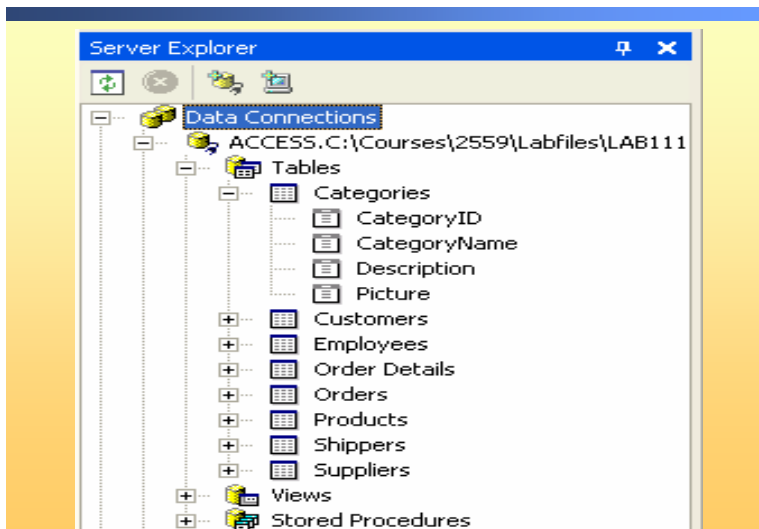
2. Mostrar la funcionalidad de la rejilla de datos utilizando las barras de desplazamiento y modificando los anchos de las columnas.

Nota Explicar que esta aplicación requeriría ahora un mayor desarrollo: agregar más capacidades de navegación, personalizar el formulario para ajustarse a los requerimientos de diseño de negocio, agregar acceso a datos específico del negocio y actualizar capacidades.

Código de la solución

El código de la solución para esta demostración se encuentra en la carpeta Solution dentro del fichero demos12.zip.

Cómo trabajar con bases de datos en el Explorador de servidores



Introducción

Podemos utilizar el Explorador de servidores para visualizar y manipular enlaces a datos, conexiones de bases de datos, y recursos del sistema en cualquier servidor para el que tengamos acceso en red. Utilizando el Explorador de servidores, podemos:

- Crear y abrir conexiones de datos a Microsoft Access, a servidores ejecutando Microsoft SQL Server y a otras bases de datos.
- Iniciar sesión en servidores y mostrar sus bases de datos, tablas, campos y sus datos sin abandonar el entorno de desarrollo ni utilizar el software de la base de datos.
- Visualizar los servicios del sistema, incluyendo el registro de sucesos, colas de mensajes, contadores de rendimiento y otros servicios del sistema.
- Visualizar información sobre los Servicios Web XML disponibles y los métodos y esquemas que proporcionan.
- Almacenar proyectos y referencias a bases de datos.
- Crear componentes de datos que hagan referencia al recurso de datos o monitorizar su actividad arrastrando nodos desde el Explorador de servidores a nuestros proyectos de Visual Studio .NET.
- Interactuar con recursos de datos programando los componentes de datos creados en nuestros proyectos de Visual Studio .NET.

Abrir el Explorador de servidores

Podemos abrir el Explorador de servidores en cualquier momento durante el proceso de desarrollo, mientras trabajamos con cualquier tipo de proyecto o elemento.

✍ Abrir el Explorador de servidores

- En el menú **Ver**, hacer clic en **Explorador de servidores**.
– O –
- Si la ficha **Explorador de servidores** se muestra en el extremo izquierdo de la pantalla, hacer clic en esa ficha.

Agregar y eliminar conexiones a datos

El Explorador de servidores muestra las conexiones a bases de datos bajo el nodo *Conexiones de datos*. Después de establecer una conexión, podemos diseñar programas para abrir conexiones y recuperar y manipular los datos. De modo predeterminado, el Explorador de servidores muestra conexiones a datos y enlaces a servidores utilizados con anterioridad.

✚ Agregar una conexión a datos en el Explorador de servidores

1. En el menú **Herramientas**, hacer clic en **Conectar con base de datos**.
Se abre el cuadro de diálogo **Propiedades del vínculo de datos**.
2. En la ficha **Proveedor** del cuadro de diálogo **Propiedades del vínculo de datos**, seleccionar un proveedor.
3. En la ficha **Conexión** del cuadro de diálogo **Propiedades del vínculo de datos**, proporcionar la información que se solicita. Los campos de entrada que se muestran pueden variar en función del proveedor seleccionado en la ficha **Proveedor**.

Por ejemplo, si seleccionamos el proveedor OLE DB para SQL Server, la ficha **Conexión** muestra campos para el nombre de servidor, el tipo de autenticación y la base de datos.

4. Hacer clic en **Aceptar** para establecer la conexión de datos.

El cuadro de diálogo **Propiedades del vínculo de datos** se cierra, y la nueva conexión de datos aparece debajo del nodo *Conexiones de datos*, con el nombre del servidor y de la base de datos a la que se accede. Por ejemplo, si creamos una conexión de datos a una base de datos denominada *NWind* en un servidor llamado *Server1*, aparece una nueva conexión con el nombre *Server1.NWind.dbo* bajo el nodo *Conexiones de datos*.

✚ Eliminar una conexión de datos desde el Explorador de servidores

1. En el Explorador de servidores, expandir el nodo **Conexiones de datos**.
2. Seleccionar la conexión a la base de datos deseada.
3. Pulsar **DELETE**.

No se produce ningún efecto en la base de datos. Hemos eliminado la referencia desde nuestra vista.

Arrastrar y soltar recursos de datos

Podemos arrastrar elementos desde el Explorador de servidores y soltarlos en el Diseñador de Windows Forms. Ubicar elementos en el Diseñador de Windows Forms crea nuevos recursos de datos que están preconfigurados para recuperar información de fuentes de datos seleccionadas.

✚ Crear un nuevo componente de datos utilizando el Explorador de servidores

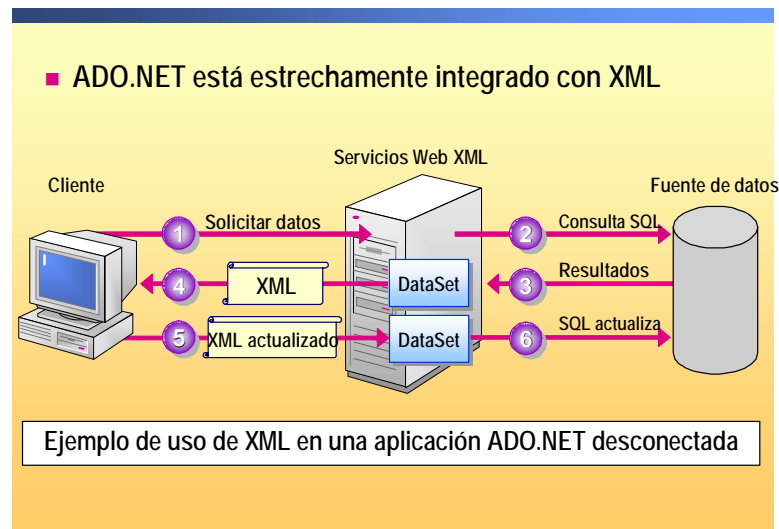
Podemos crear un componente de datos preconfigurado para hacer referencia a un determinado recurso.

1. En la vista de Diseño, abrir el formulario al que deseamos agregar un componente de datos.
2. En el **Explorador de servidores**, seleccionar el elemento de datos que deseamos utilizar. Un ejemplo de elemento de datos es un campo o una tabla.
3. Arrastrar el elemento desde el **Explorador de servidores** a la superficie del diseñador.

Visualizar elementos de la base de datos

Podemos utilizar el Explorador de servidores para visualizar y recuperar información de todas las bases de datos instaladas en un servidor. Podemos hacer una lista de tablas, vistas, procedimientos almacenados y funciones de la base de datos en el Explorador de servidores, expandir tablas individuales para hacer una lista de sus columnas y disparadores, y hacer clic con el botón derecho sobre la tabla para seleccionar el Diseñador de tablas del menú contextual.

ADO.NET y XML



Introducción

ADO.NET está estrechamente integrado con XML. Por ello, ADO.NET facilita la conversión de datos relacionales a formato XML. También podemos convertir datos de XML en tablas y relaciones.

Importancia del uso de XML

XML es una forma útil y portátil de representar datos de un modo abierto e independiente de la plataforma. Una característica importante de los datos XML es que están basados en texto. El uso de datos basados en texto, en contraposición a pasar datos binarios como el conjunto de resultados de ADO, facilita el intercambio de datos XML entre aplicaciones y servicios, aunque se estén ejecutando en diferentes plataformas. XML también permite que las organizaciones intercambien datos sin necesidad de una personalización adicional del software propietario de cada organización.

Soporte de XML

El modelo de objetos ADO.NET incluye soporte extensivo para XML. Cuando utilice el soporte de XML en ADO.NET, debe tener en cuenta los siguientes hechos y directrices:

- Se pueden leer datos de un **DataSet** en formato XML. El formato XML es útil si desea intercambiar datos entre aplicaciones o servicios en un entorno distribuido.
- Se puede rellenar un **DataSet** con datos XML. Esto resulta útil si se reciben datos XML de otra aplicación o servicio y se desea actualizar una base de datos utilizando estos datos.
- Puede crearse un esquema XML para la representación XML de los datos en un **DataSet**. El esquema XML puede utilizarse para realizar tareas como serializar los datos XML a un flujo o archivo.

- Pueden cargarse datos XML desde un flujo o un archivo a un árbol Document Object Model (DOM). A continuación, pueden manipularse los datos como XML o como un **DataSet**. Para ello, debe disponer de un esquema XML para describir la estructura de los datos para el **DataSet**.
- Pueden crearse **DataSets** tipados. Un **DataSet** tipado es una subclase de **DataSet**, con propiedades y métodos adicionales para tener disponible la funcionalidad de **DataSet**. Para describir la representación XML del **DataSet** tipado, Visual Studio .NET genera una definición equivalente del esquema XML para este **DataSet**.

Ejemplo de uso de XML en una aplicación desconectada


Este ejemplo describe cómo utilizar XML en una aplicación ADO.NET desconectada. Podemos utilizar XML para intercambiar datos entre las distintas partes del sistema del modo siguiente:

1. La aplicación cliente invoca un servicio Web XML para solicitar datos de una base de datos.
2. El servicio Web XML consulta una fuente de datos para obtener los datos solicitados.
3. El servicio Web XML carga los resultados en un **DataSet**.
4. El servicio Web XML traduce los datos a formato XML y devuelve los datos XML a la aplicación cliente.
5. La aplicación cliente procesa los datos XML de algún modo. Por ejemplo, el cliente puede cargar los datos XML en un **DataSet** y enlazarlos a un control del interfaz de usuario como un **DataGrid**. Cuando la aplicación cliente esté lista, invoca un servicio Web XML para actualizar la fuente de datos con los cambios de los datos.
6. El servicio Web XML carga los nuevos datos XML en un **DataSet** y utiliza los nuevos datos para actualizar la fuente de datos.

Nota Si deseamos obtener más información sobre cómo poblar un **DataSet** con un flujo XML, consultar “Clase **DataSet**, XML” en la documentación de Visual Studio .NET.

Si desea obtener más información sobre la obtención de datos como XML desde SQL Server, consultar “Proveedor de datos de .NET SQL Server, XML” en la documentación de Visual Studio .NET.

Práctica: acceso a datos en modo lectura



- En esta práctica,
- 1. Iniciaremos una nueva aplicación de Windows Forms
- 2. Escogeremos el tipo de conexión y base de datos
- 3. Agregaremos la Conexión y el **DataAdapter** al formulario
- 4. Generaremos el **DataSet**
- 5. Agregaremos un control **DataGrid** al formulario y estableceremos sus propiedades
- 6. Utilizaremos el método **Fill** para poblar el **DataSet**
- 7. Ejecutaremos la aplicación para visualizar datos de sólo lectura

Introducción

En esta práctica, abriremos un nuevo proyecto y utilizaremos la funcionalidad de arrastrar y soltar que proporciona el Explorador de servidores para crear un conjunto de datos y poblar un a rejilla de datos con datos de sólo lectura.

Nota La práctica se centra en los conceptos de esta lección y, por ello, es posible que no cumpla las recomendaciones de seguridad de Microsoft. Por ejemplo, esta práctica no cumple la recomendación de utilizar cuentas de usuario particulares en lugar de cuentas de administrador, o la recomendación de que todas las cuentas utilicen contraseñas seguras.

Instrucciones

⚡ Crear un nuevo proyecto

1. Iniciar un nuevo proyecto de Visual Basic .NET basado en la plantilla *Windows Application*.
2. Asignar al proyecto el nombre **PubsDataGrid**, establecer la ubicación de la carpeta, y hacer clic en **Aceptar**.

✚ Agregar Connection y DataAdapter al formulario

1. En el Explorador de servidores, bajo Servidores, expandir el nodo de *nombre_del_pc*, expandir el nodo **Servidores SQL Server**, y expandir el nodo *nombre_del_pc/NetSDK* para que se muestren todas las bases de datos SQL Server disponibles en el equipo.
2. Expandir el nodo de la base de datos **Pubs**, y expandir el nodo **Tablas** bajo Pubs.
3. Arrastrar la tabla *Authors* a **Form1**.
Cuando se arrastra la tabla al formulario, se crea **SqlConnection1** y **SqlDataAdapter1** y se agregan a la bandeja de componentes.
4. En la ventana *Propiedades*, cambiar la propiedad **Name** de **SqlConnection1** por **PubsConn** y cambiar la propiedad **Name** de **SqlDataAdapter1** por **PubsSqlDataAdapter**.
5. Expandir la propiedad **InsertCommand** de **PubsSqlDataAdapter**. Verificar que la propiedad **Connection** está establecida como **PubsConn**. Observar que el **DataAdapter** y **Connection** que se han agregado ya están conectados entre sí.

✚ Generar el DataSet para la tabla Products

1. En la ventana *Propiedades*, en el área de descripción en la parte inferior de la lista de propiedades, hacer clic en **Generar Dataset**.
2. En el cuadro de diálogo **Generar Dataset**, escribir **PubsDataSet** en el cuadro de texto **Nuevo**.
3. En el resto de cuadros de texto, dejar los valores predeterminados. Hacer clic en **Aceptar**.
PubsDataSet1 debería figurar en la bandeja de componentes.

✚ Agregar DataGrid al formulario y establecer sus propiedades

1. En la ficha **Windows Forms** del Cuadro de herramientas, agregar un control **DataGrid** al formulario.
2. En la ventana *Propiedades*, asignar al control **DataGrid** el nombre **PubsDataGrid**.
3. En la ventana *Propiedades*, establecer la propiedad **DataSource** del control **DataGrid** en **PubsDataSet1**, y establecer la propiedad **DataMember** en **Authors**.
4. Modificar el tamaño del formulario y del control **DataGrid** para que todas las filas y columnas estén visibles.
La rejilla de datos debería mostrar los nombres de campo de la tabla *Authors*.

⚡ Poblar el DataSet con los datos de la tabla *Authors*

- Abrir un controlador de eventos para el evento **Form1_Load**. En el controlador de eventos **Form1_Load**, invocar el método **Fill** del adaptador de datos para poblar la tabla *Authors* con datos. El código debería ser parecido al siguiente:

```
PubsSqlDataAdapter.Fill(PubsDataSet1)
```

⚡ Probar la aplicación

1. Ejecutar la aplicación.
Los datos de *Authors* deberían poblar la rejilla de datos.
2. Navegar por los datos utilizando las barras de desplazamiento de la rejilla de datos.

Nota Aunque hemos creado una aplicación que funciona y que muestra datos en modo lectura desde una fuente de datos, sus funcionalidades son limitadas. Por ejemplo, únicamente se puede navegar con las barras de desplazamiento de la rejilla de datos. En una aplicación real, es posible que quisiésemos modificar más adelante la selección de campos que deben mostrarse. Para agregar más funcionalidades al formulario, podemos permitir a los usuarios que visualicen detalles de los productos. También es posible permitir que los usuarios seleccionen un elemento, ir a otro formulario que herede datos de éste o crear un pedido o ver los pedidos existentes.

Archivos de solución

Los archivos de solución de esta práctica se encuentran en la carpeta Solution dentro del fichero practs12.zip.

Lección: trabajar con datos

- Cómo utilizar un objeto **Connection**
- Cómo utilizar un objeto **DataAdapter**
- Cómo utilizar un objeto **DataSet**
- Cómo utilizar un control **DataGrid**
- Cómo utilizar el asistente **Data Form Wizard**

Introducción

ADO.NET proporciona todos los componentes necesarios para conectarnos a fuentes de datos y trabajar con datos en nuestras aplicaciones. En esta lección, aprenderemos a utilizar objetos ADO.NET para conectarnos a una base de datos, recuperar modificar y transmitir datos actualizados a la base de datos. También aprenderemos cómo mostrar datos en un formulario Windows Forms utilizando el control **DataGrid**. Finalmente, aprenderemos cómo utilizar el Asistente para formularios de datos para automatizar el proceso de trabajar con datos. El asistente simplifica la tarea de agregar controles y establecer propiedades a unos pocos pasos sencillos.

Objetivos de la lección

En esta lección, aprenderemos a:

- Establecer una conexión a una fuente de datos utilizando el objeto **Connection**.
- Recuperar datos de una fuente de datos utilizando los objetos **DataAdapter** y **DataSet**.
- Enviar actualizaciones a una fuente de datos utilizando los objetos **DataAdapter** y **DataSet**.
- Utilizar un control **DataGrid** para mostrar datos.
- Utilizar el Asistente para formularios de datos para agregar a un proyecto un nuevo formulario enlazado a datos.

Estructura de la lección

Esta lección incluye los siguientes temas y actividades:

- Cómo utilizar un objeto **Connection**
- Cómo utilizar un objeto **DataAdapter**
- Cómo utilizar un objeto **DataSet**
- Cómo utilizar un control **DataGrid**
- Cómo utilizar el Asistente para formularios de datos

Cómo utilizar un objeto Connection

- Utilizamos Connection para:
 - Elegir el tipo de conexión
 - Especificar la fuente de datos
 - Abrir la conexión a la fuente de datos
 - Cerrar la conexión a la fuente de datos
- Ejemplo de conexión a una base de datos SQL Server

```
Dim PubsSQLConn As SqlClient.SqlConnection
PubsSQLConn = New SqlClient.SqlConnection( )
PubsSQLConn.ConnectionString = "Integrated Security=True;" & _
    "Data Source=local;Initial Catalog=Pubs;"
PubsSQLConn.Open( )
```

Introducción	Antes de trabajar con datos, es necesario establecer primero una conexión con una fuente de datos. Para conectar con una fuente de datos, escogemos el tipo de conexión, especificamos la fuente de datos, y abrimos la conexión a la fuente de datos. Cuando acabamos de trabajar con los datos, cerraremos la conexión.
Escoger el tipo de conexión	Podemos utilizar el objeto Connection para conectar a una fuente de datos específica. Podemos utilizar SqlConnection para conectar a bases de datos SQL Server o OleDbConnection para conectar a otros tipos de fuentes de datos.
Especificar la fuente de datos	Después de escoger el tipo de conexión, utilizamos una propiedad ConnectionString para especificar el proveedor de datos, la fuente de datos, y demás información utilizada para establecer la conexión. El formato de estas cadenas difiere ligeramente entre el espacio de nombres SqlClient y el espacio de nombres OleDb .
Abrir y cerrar la conexión de datos	El objeto Connection soporta un método Open que abre la conexión después de que se hayan establecido las propiedades de la conexión, y un método Close que cierra la conexión a la base de datos después de que todas las transacciones se hayan liberado.

Ejemplo de SQL

El siguiente ejemplo muestra cómo utilizar un objeto **Connection**, la propiedad **ConnectionString** y el método **Open** para conectarnos a una base de datos SQL Server utilizando el Data Provider .NET de SQL Server:

```
Dim PubsSQLConn As SqlConnection
PubsSQLConn = New SqlConnection( )
PubsSQLConn.ConnectionString = "Integrated Security=True;" & _
    "Data Source=local;Initial Catalog=Pubs;"
PubsSQLConn.Open( )
```

Ejemplo de Jet 4.0

El siguiente ejemplo muestra cómo conectarnos a una base de datos Access utilizando el Data Provider OLE DB Jet 4.0:

```
Dim NWindOLEDBConn As OleDb.OleDbConnection
NWindOLEDBConn = New OleDb.OleDbConnection( )
NWindOLEDBConn.ConnectionString = _
    "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\NWind.MDB"
NWindOLEDBConn.Open( )
```


Cómo utilizar un objeto DataAdapter

■ To create a DataAdapter

- Declare with the **Dim** keyword
- Pass a query string and a **Connection** object as parameters

```
Dim PubsAdapter As SqlDataAdapter = New SqlDataAdapter _
    ("Select * from Titles", PubsSQLConn)
```

■ Key methods of DataAdapter:

- **Fill** method populates a data set
- **Update** method transmits changes to the data store

Introducción

Tras establecer una conexión con una fuente de datos, podemos utilizar un adaptador de datos para intercambiar datos entre la fuente de datos y un *dataset*. En muchas aplicaciones, esto significa leer datos de una base de datos en un *dataset* mediante un adaptador de datos y a continuación escribir los datos cambiados desde el *dataset* al adaptador de datos y nuevamente a la base de datos. Sin embargo, un adaptador de datos puede mover datos entre cualquier fuente y un *dataset*. Por ejemplo, podría existir un adaptador que mueva datos entre un servidor que ejecute Microsoft Exchange y un *dataset*.

Tipos de adaptadores de datos

Visual Studio incorpora dos adaptadores de datos principales para utilizarse con bases de datos:

- **OleDbDataAdapter** es apropiado para utilizarlo con cualquier fuente de datos que proporcione un proveedor OLE DB.
- **SqlDataAdapter** es específico a SQL Server. Como no funciona a través de una capa OLE DB, es más rápido que **OleDbDataAdapter**. Sin embargo, sólo puede ser utilizado con SQL Server versión 7.0 o superior.

Creación del DataAdapter

Podemos instanciar el **DataAdapter** utilizando la palabra clave **Dim** y pasando una cadena de consulta y un objeto **Connection** como parámetros. El **DataAdapter** verificará si la **Connection** está abierta, y si no lo está, la abrirá para nosotros y la cerrará cuando nuestra llamada al método se complete. Esta aproximación es útil si ya hemos establecido las propiedades de un objeto **Connection** en nuestra aplicación y sólo necesitamos abrir la conexión para poblar las tablas de datos.

Ejemplos

Los siguientes ejemplos muestran cómo crear una instancia de una clase **Sql Client DataAdapter**. El código para crear una instancia de la clase **DataAdapter** se muestra en negrita.

Ejemplo de SQL

El siguiente ejemplo muestra cómo acceder a la tabla *Titles* en la base de datos *Pubs*:

```
Dim PubsSQLConn As SqlConnection
PubsSQLConn = New SqlConnection( )
PubsSQLConn.ConnectionString = "Integrated Security=True;" & _
    "Data Source=local;Initial Catalog=Pubs;"

Dim PubsAdapter As SqlDataAdapter = New SqlDataAdapter _
    ("Select * from Titles", PubsSQLConn)

' Code to populate the dataset
Dim titlesDataset As DataSet = New DataSet
PubsAdapter.Fill(titlesDataset)

' Code to bind the dataset to form controls might go here
```

Ejemplo de Jet 4.0

El siguiente ejemplo muestra cómo acceder a la tabla *Customers* en la base de datos *Northwind* utilizando el OLE DB Provider para Microsoft Jet:

```
Private NWindOleDbConn as OleDb.OleDbConnection

Private Sub Form1_Load(...) Handles MyBase.Load
    NWindOleDbConn = New OleDb.OleDbConnection( )
    NWindOleDbConn.ConnectionString = "Provider= " & _
        "Microsoft.Jet.OLEDB.4.0;Data Source=c:\NWind.mdb"

Dim NWindAdapter As New OleDb.OleDbDataAdapter _
    ("Select * from Customers", NWindOleDbConn)

' Code to populate the DataSet might go here
' Code to bind the DataSet to form controls might go here
```

Métodos clave

DataAdapter soporta métodos específicos para mover datos entre el *dataset* y la fuente de datos. Podemos utilizar un adaptador de datos para realizar las siguientes operaciones:

- Recuperar filas de una fuente de datos y poblarlas en tablas de datos correspondientes en un *dataset*.

Para recuperar filas de una fuente de datos y poblar un *dataset*, utilizar el método **Fill** de **SqlDataAdapter** u **OleDbDataAdapter**. Cuando invocamos el método, se invoca una sentencia SQL SELECT en la fuente de datos. Ejecutar este método abrirá y cerrará el objeto de conexión.

- Transmitir los cambios realizados a una tabla *dataset* a la fuente de datos correspondiente.

Para transmitir los cambios en una tabla *dataset* a la fuente de datos, utilizar el método **Update** del adaptador. Cuando invocamos el método, ejecuta cualquier sentencia SQL INSERT, UPDATE o DELETE que sea necesaria, dependiendo de si el registro afectado es nuevo, modificado o borrado. Ejecutar este método abrirá y cerrará el objeto conexión.

Ejemplo

```
' Code to manipulate the data locally by using the DataSet
' goes here
```

```
' Update the database by means of the data adapter
```

```
PubsAdapter.Update (PubsDataSet)
```

Ejemplo de Jet 4.0

```
' Create the Connection, the DataAdapter, and the DataSet,
' and populate the dataset
```

```
Dim NWindConn As OleDb.OleDbConnection
NWindConn = New OleDb.OleDbConnection( )
NWindConn.ConnectionString = "Provider= " & _
    "Microsoft.Jet.OLEDB.4.0;Data Source=c:\NWind.mdb"
```

```
Dim NWindAdapt As New OleDb.OleDbDataAdapter("SELECT * " & _
    "from Customers", NWindConn)
```

```
Dim NWindDataSet As DataSet = New DataSet( )
NWindAdapt.Fill(NWindDataSet, "customersTable")
```

```
' Code to manipulate the data locally by using the DataSet
' goes here
```

```
' Update the database by means of the data adapter
```

```
NWindAdapt.Update (NWindDataSet.Tables("customersTable"))
```

Cómo utilizar un objeto DataSet

- How DataSets work
 - Store data in a disconnected cache
 - Use a hierarchical object model of tables, rows, and columns
- You can populate a DataSet by:
 - Using the **Fill** method
 - Manually populating the tables
 - Reading an XML document or stream
 - Merging or copying the contents of another **DataSet**

Introducción

Después de especificar los datos que queremos recuperar, el próximo paso es poblar un *dataset* con datos desde la base de datos. Los *datasets* almacenan datos en un caché desconectado. La estructura de un *dataset* es similar a la de una base de datos relacional y ofrece un modelo de objetos jerárquico de tablas, filas y columnas. Además, contiene restricciones y relaciones que están definidas para el *dataset*.

Poblar datasets

Un *dataset* es un contenedor, de modo que necesitamos poblarlo con datos. Podemos poblar un *dataset* de varios modos:

- Invocar el método **Fill** de un adaptador de datos. Llamando este método hace que el adaptador ejecute una sentencia SQL o un procedimiento almacenado y pueble una tabla en el *dataset* con los resultados. Si el *dataset* contiene múltiples tablas, probablemente tendremos adaptadores de datos separados para cada tabla, y por tanto deberemos llamar a cada método **Fill** de cada adaptador por separado.
- Poblar tablas manualmente en el *dataset* creando **DataRows** y añadirlas a la colección **Rows** de la tabla. Podemos hacer esto sólo en tiempo de ejecución. No podemos establecer la colección **Rows** en tiempo de diseño.
- Leer un documento XML o flujo en el *dataset*.
- Copiar los contenidos de otro *dataset*. Esta técnica puede ser útil si nuestra aplicación obtiene *datasets* de diferentes fuentes (diferentes Servicios Web XML, por ejemplo), pero necesita consolidarlos en un único *dataset*.

Ejemplo

Utilizamos un **DataAdapter** para acceder a datos almacenados en una base de datos y almacenar los datos en **DataTables** dentro de un **DataSet** e nuestra aplicación.

Ejemplo de SQL

Las sentencias en negrita del siguiente ejemplo muestran cómo poblar un **DataSet** denominado PubsDataSet con datos de una base de datos SQL utilizando un **DataAdapter**. Sólo hay una tabla y el *dataset* hará referencia a ella utilizando un número de índice. El comando **Update** utiliza la misma tabla para actualizar los cambios a la base de datos.

```
' Create the Connection, the DataAdapter, and the DataSet,
' and populate the dataset

Dim PubsSQLConn As SqlConnection
PubsSQLConn = New SqlConnection( )
PubsSQLConn.ConnectionString = "Integrated Security=True;" & _
    "Data Source=local;Initial Catalog=Pubs;"

Dim PubsAdapter As SqlDataAdapter = New SqlDataAdapter _
    ("Select * from Authors", PubsSQLConn)
Dim PubsDataSet As DataSet = New DataSet( )
PubsAdapter.Fill(PubsDataSet)

' Code to manipulate the data locally by using the DataSet
' goes here

' Update the database by means of the data adapter

PubsAdapter.Update (PubsDataSet.Tables("authorsTable"))
```

Ejemplo de Jet 4.0

Las sentencias en negrita del siguiente ejemplo muestran cómo poblar un **DataSet** llamado NWindDataSet con datos desde una base de datos OLE DB utilizando un **DataAdapter**.

```
' Create the Connection, the DataAdapter, and the DataSet,
' and populate the dataset:

Dim NWindConn As OleDb.OleDbConnection
NWindConn = New OleDb.OleDbConnection( )
NWindConn.ConnectionString = "Provider= " & _
    "Microsoft.Jet.OLEDB.4.0;Data Source=c:\NWind.mdb"

Dim NWindAdapt As New OleDb.OleDbDataAdapter("SELECT * " & _
    "from Customers", NWindConn)
Dim NWindDataSet As DataSet = New DataSet( )
NWindAdapt.Fill(NWindDataSet)

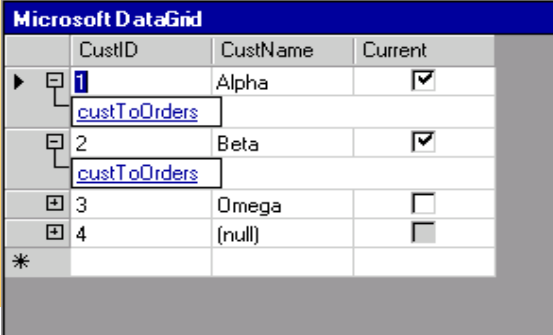
' Code to manipulate the data locally by using the DataSet
' goes here

' Update the database by means of the data adapter

NWindConn.Open( )
NWindAdapt.Update (NWindDataSet)
NWindConn.Close( )
```

Cómo utilizar un control DataGrid

- Use a DataGrid control to display data from a single data source or multiple data sources
 - From the Toolbox, add a **DataGrid** control to a form
 - Set the properties of the **DataGrid** control



CustID	CustName	Current
1	Alpha	<input checked="" type="checkbox"/>
custToOrders		
2	Beta	<input checked="" type="checkbox"/>
custToOrders		
3	Omega	<input type="checkbox"/>
4	(null)	<input type="checkbox"/>
*		

Introducción

Después de poblar un *dataset*, podemos visualizar y modificar datos utilizando el control **DataGrid** de Windows Forms. **DataGrid** muestra datos en una serie de filas y columnas. El caso más simple es cuando la rejilla se enlaza con una fuente de datos que contiene una única tabla sin relaciones. En ese caso, los datos aparecen simplemente en filas y columnas, como en una hoja de cálculo.

Cómo funciona el control DataGrid

Si el **DataGrid** se enlaza a datos con múltiples tablas relacionadas y si se activa la navegación en la rejilla, éste visualizará expansores en cada fila. Un expansor permite la navegación desde una tabla padre a una hija. Haciendo clic en un nodo se muestra la tabla hija y haciendo clic en el botón **Atrás** muestra la tabla padre original. En este modo, la rejilla muestra las relaciones jerárquicas entre tablas.

DataGrid puede proporcionar un interfaz de usuario para un *dataset*, navegación entre tablas relacionadas y ricas capacidades de formateo y edición.

La visualización y manipulación de datos son funciones separadas: el control gestiona el interfaz de usuario, mientras que las actualizaciones de datos las gestiona la arquitectura de enlace de datos de Windows Forms y por los proveedores de datos ADO.NET. Por tanto, múltiples controles enlazados a la misma fuente de datos permanecerán sincronizados.

Enlace de datos al control

Para que el control **DataGrid** funcione, debemos enlazarlo con una fuente de datos utilizando las propiedades **DataSource** y **DataMember** en tiempo de diseño o el método **SetDataBinding** en tiempo de ejecución. Este enlace vincula el **DataGrid** a un objeto de fuente de datos instanciado (como un **DataSet** o **DataTable**) y el control **DataGrid** se actualiza con los resultados de acciones realizadas sobre los datos.

La mayoría de acciones específicas de datos no son realizadas por el **DataGrid**. Se realiza por medio de la fuente de datos.

✍ Para enlazar el control **DataGrid** a una única tabla en el diseñador

1. Establecer la propiedad del control **DataSource** al objeto que contiene los elementos de datos a los que queremos enlazar.
2. Si la fuente de datos es un *dataset*, establecer la propiedad **DataMember** al nombre de la tabla con la que queremos enlazar.
3. Si la fuente de datos es un *dataset* o una vista de datos basada en una tabla de un *dataset*, añadir código al formulario para poblar el *dataset*.

Enlace simple

Utilizamos enlace simple para vincular un control a un simple campo en un **DataSet**. Por ejemplo, podríamos usar enlace simple para un control **TextBox** o **Label**. Utilizando la propiedad **DataBindings** de un control con capacidad de vinculación a datos, podemos especificar qué **DataSet** y qué campo enlazar con qué propiedad.

Ejemplo de SQL

El siguiente ejemplo muestra cómo enlazar datos a un control **TextBox**:

```
Dim PubsSQLConn As SqlConnection
PubsSQLConn = New SqlConnection( )
PubsSQLConn.ConnectionString = "Integrated Security=True;" & _
    "Data Source=local;Initial Catalog=Pubs;"
PubsSQLConn.Open( )

Dim PubsAdapter As SqlDataAdapter = New SqlDataAdapter _
    ("Select * from Titles", PubsSQLConn)

Dim PubsDataSet As DataSet = New DataSet( )
PubsAdapter.Fill(PubsDataSet)

TextBox1.DataBindings.Add(New Binding("Text", _
    PubsDataSet, "Authors.FirstName"))
```

Ejemplo de Jet 4.0

El código **TextBox** no difiere de cuando utilizamos un Jet 4.0. Sólo el código de **Connection** y **DataAdapter**, como muestran los ejemplos anteriores.

Enlace complejo

Utilizamos enlace complejo para enlazar un control a múltiples campos en un **DataSet**. Por ejemplo, utilizamos enlace complejo para un control **DataGrid**. Utilizando la propiedad **DataSource** de estos controles, podemos especificar la tabla a utilizar.

Ejemplo de Jet 4.0

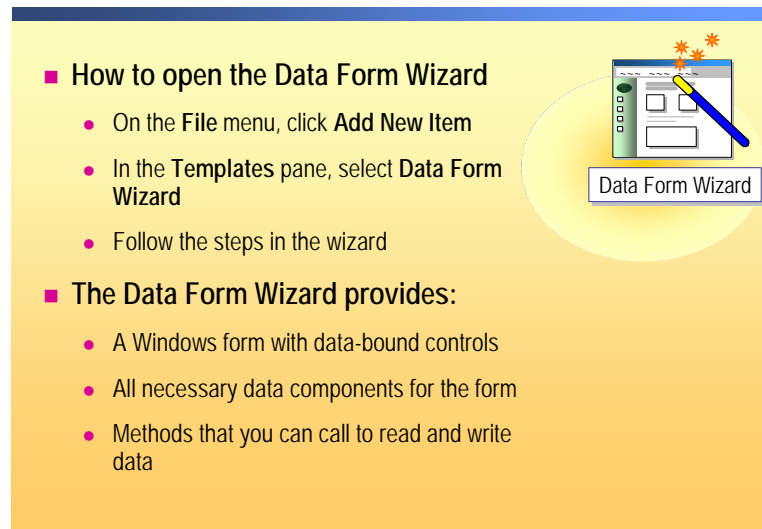
El siguiente ejemplo muestra cómo utilizar la propiedad **DataSource** de un control **DataGrid** para una base de datos Jet 4.0. El código que establece la propiedad está formateado en negrita.

```
Dim NwindOleDbConn As OleDb.OleDbConnection = _  
    New OleDb.OleDbConnection( )  
Dim NwindAdapter As New OleDb.OleDbDataAdapter _  
    ("Select * from Customers", NwindOleDbConn)  
Dim NwindDataSet As DataSet = New DataSet( )  
  
NwindOleDbConn.ConnectionString = "Provider= " & _  
    "Microsoft.Jet.OLEDB.4.0;datasource=c:\Nwind.mdb"  
NwindOleDbConn.Open( )  
NwindAdapter.Fill(NwindDataSet)  
  
DataGrid1.DataSource = NwindDataSet  
DataGrid1.DataMember = "Customers"
```

Ejemplo de SQL

El código **TextBox** no difiere de cuando utilizamos un proveedor SQL. Sólo el código de **Connection** y **DataAdapter**, como muestran los ejemplos anteriores.

Cómo utilizar el Asistente para formularios de datos



Introducción

El Asistente para formularios de datos nos ayuda a crear una aplicación Windows Forms con controles enlazados a datos. Los controles muestran datos desde cualquier fuente de datos que especifiquemos.

Ejecución del Asistente para formularios de datos

Ejecutar el Asistente para formularios de datos es como añadir un elemento a un proyecto existente.

⚡ Para ejecutar el Asistente para formularios de datos

1. Crear o abrir una aplicación Windows Forms.

Nota El asistente puede crear un formulario únicamente en un tipo de proyecto que soporte formularios.

2. Desde el menú **Archivo**, hacer clic en **Agregar nuevo elemento**. Aparece el cuadro de diálogo **Agregar nuevo elemento**.
3. En el panel **Plantillas**, hacer clic en **Asistente para formularios de datos**.
4. Poner un nombre en el formulario de datos y hacer clic en **Abrir**. Se inicia el asistente.
5. Seguir los pasos del asistente.

Qué hace el asistente

El Asistente para formularios de datos nos guía a través del proceso completo de crear un formulario enlazado a datos. Cuando hayamos finalizado, tendremos:

- Un formulario Windows Form.
- Controles enlazados a datos en el formulario.
- Todos los componentes de datos necesarios para el formulario.
- Métodos que podemos invocar para leer y, para Windows Forms, escribir datos.

El formulario y los controles

Para visualizar datos, el formulario generado por el asistente incluye controles que están enlazados a una fuente de datos. Por ejemplo, el formulario puede contener un control de rejilla.

En un formulario Windows Forms, podemos seleccionar entre una rejilla o controles específicos que muestren un registro cada vez. Podemos también generar botones que permitan a los usuarios navegar entre registros e incluir botones para guardar los cambios a los registros y añadir nuevos.

Elementos de datos

El Asistente para formularios de datos puede crear nuevos componentes de datos o utilizar componentes que estén disponibles en nuestro proyecto. Un componente de datos para el formulario está formado por varios elementos discretos:

- Una conexión a una fuente de datos, como una base de datos.
- Un adaptador de datos que referencia una sentencia SQL o procedimiento almacenado para obtener datos.
- Un *dataset* tipado contiene los registros recuperados de la base de datos. El *dataset* incluye un esquema, un archivo de clase *dataset* y una instancia del *dataset* en el formulario.

Si utilizamos el asistente para crear una fuente de datos, conectará nuestra aplicación a la base de datos especificada. Dentro de la base de datos, el asistente puede acceder a tablas o vistas.

El asistente puede opcionalmente crear y utilizar una referencia a un Servicio Web XML como la fuente de datos. En ese escenario, se asume que el Servicio Web XML es capaz de devolver un *dataset*.

Podemos utilizar el asistente para acceder a múltiples tablas y establecer relaciones entre ellas. En este caso, los controles en el formulario nos permiten visualizar tanto registros padre como hijos.

Métodos Fill y Update

Una de las tareas que necesitamos realizar es rellenar el *dataset* con datos de la base de datos. Por defecto, para rellenar el *dataset*, el asistente genera un botón **Load** que invoca el código que realiza la operación de llenado. Si estamos creando un nuevo *dataset* con el asistente, éste genera el método **Fill** automáticamente. Si estamos trabajando con un *dataset* existente, el asistente nos ofrece estas opciones para crear un método **Fill**:

- Seleccionar un método existente

Si nuestro proyecto proporciona un método que devuelve un *dataset* (tanto como valor de retorno de una función o como un parámetro), podemos seleccionar ese método y el asistente genera código para invocar ese método.

- Obviar el método

Seleccione esta opción si queremos rellenar el *dataset* por nosotros mismos en lugar de que el código de llenado se invoque automáticamente.

Si estamos generando un formulario Windows Form, el asistente también puede generar un método **Update** que podemos invocar para enviar cambios desde el *dataset* de retorno a la fuente de datos.

Pasos adicionales

Cuando el asistente finaliza y tenemos un nuevo formulario, éste está listo para ejecutarse. Hay algunos pocos pasos adicionales que podemos desear realizar y una variedad de modos en los que puede ampliar lo que el asistente ha realizado.

Por ejemplo, para trabajar con el formulario que acabamos de crear, necesitamos establecer el formulario como el formulario de inicio para que podamos probarlo.

⚡ Para establecer el formulario de inicio en Windows Forms

1. En el **Explorador de soluciones**, hacer clic con el botón derecho en el proyecto, y a continuación hacer clic en **Propiedades**.

La página de propiedades del **Proyecto** se abre con las propiedades **General** visualizadas.

2. Seleccionar el formulario que queremos que sea el formulario de inicio desde la lista de **Objeto inicial**.

El asistente crea controles, elementos de datos y todo el código necesario para ejecutar el formulario. Podemos leer y modificar este código si es necesario. Podemos querer realizar las siguientes tareas para mejorar el formulario que crea el asistente:

- Cambiar el aspecto y disposición de los controles en el formulario.
- Si no queremos utilizar el botón **Load** para rellenar el *dataset*, podemos mover el código de llenado a otra ubicación. Un ejemplo típico es moverlo desde el procesador de eventos del botón al método de inicialización del formulario de modo que el *dataset* se rellena automáticamente cuando el formulario se ejecuta.
- Refinar las consultas que el asistente genera para que el adaptador de datos recupere un subconjunto de los datos y que añada o modifique parámetros a los comandos del adaptador.
- Añadir validación o lógica de negocio al formulario.
- Añadir lógica de navegación para prevenir que ocurran excepciones cuando nos movemos más allá del primer y último registro.

Acceder a datos con DataReaders

- Creating a DataReader
- Reading Data from a DataReader
- Using DataSets vs. DataReaders

El beneficio de utilizar un **DataSet** es que nos ofrece una vista desconectada de la base de datos. Para aplicaciones de larga ejecución, es a menudo el mejor método. Sin embargo, los desarrolladores de aplicaciones Web generalmente realizan operaciones cortas y sencillas con cada petición, como visualizar datos. Para estas breves operaciones, generalmente no es eficiente mantener un objeto **DataSet**. En tales casos, podemos utilizar un **DataReader**.

Seleccionar entre un DataReader y un DataSet

Al decidir si nuestra aplicación debería utilizar un **DataReader** o un **DataSet**, consideremos el tipo de funcionalidad que nuestra aplicación requiere. Usar un **DataSet** cuando necesitemos realizar alguna de las siguientes acciones:

- Transmitir datos entre niveles o desde un Servicio Web XML.
- Interactuar dinámicamente con datos, como cuando enlazamos a un control Windows Forms o combinamos y relacionamos datos desde múltiples fuentes.
- Almacenar localmente datos en caché para nuestra aplicación.
- Proporcionar una vista jerárquica XML de datos relacionales y utilizar herramientas como Transformaciones XSL o una consulta XML Path Language (XPath) Query sobre nuestros datos.

Si deseamos obtener más información, consultar “XML and the DataSet” en la documentación del .NET Framework SDK.

- Realizar procesamiento intensivo sobre datos que no requiera una conexión abierta a la fuente de datos, dejando la conexión disponible a otros clientes.

Si no necesitamos la funcionalidad que proporciona la clase **DataSet**, podemos mejorar el rendimiento de nuestra aplicación utilizando la clase **DataReader** para devolver nuestros datos de modo *sólo-lectura/sólo-reenvío*. Aunque la clase **DataAdapter** utiliza un **DataReader** para rellenar los contenidos de un **DataSet**, utilizando la clase **DataReader** en su lugar podemos asegurar los siguientes beneficios de rendimiento:

- La memoria que sería consumida por el **DataSet** estará de este modo disponible.
- El procesamiento que sería requerido para crear y rellenar los contenidos del **DataSet** en este caso no será necesario.

En esta sección, aprenderemos cómo leer datos desde una fuente de datos utilizando **DataReaders**.

Crear un DataReader

Objetivo

Describir cómo recuperar datos desde una base de datos utilizando un **DataReader**.

Presentación

Podemos también utilizar un objeto **DataReader** para leer datos desde una base de datos.

- Create and open the database connection

- Create the DataReader from a command

```
Dim mySqlCommand As New SqlCommand( ↵  
    "select * from customers", mySqlConnection)  
Dim myReader As SqlDataReader = ↵  
    mySqlCommand.ExecuteReader()
```

- Close the DataReader and the Connection

```
If Not (myReader Is Nothing) Then  
    myReader.Close()  
End If  
If mySqlConnection.State = ↵  
    ConnectionState.Open Then  
    mySqlConnection.Close()  
End If
```

Para minimizar la dificultad de tales situaciones, el **DataReader** ha sido diseñado para producir un flujo sólo-lectura, sólo-reenvío que se devuelve desde la base de datos. Únicamente un registro a la vez está siempre en memoria. Hay dos clases **DataReader**: **SqlDataReader** y **OleDbDataReader**.

Un **DataReader** mantiene la conexión abierta hasta que se cierra el **DataReader**.

Para utilizar un **SqlDataReader**, declarar un **SqlCommand** en vez de un **SqlDataAdapter**. El **SqlCommand** expone un método **ExecuteReader** que toma un **SqlDataReader** como parámetro. Observar que cuando utilizamos un **SqlCommand**, debemos explícitamente abrir y cerrar **SqlConnection**.

```

Dim mySqlConnection As New ↵
    SqlConnection("server=(local)\NetSDK; ↵
        Trusted_Connection=yes;database=northwind")
mySqlConnection.Open()

Dim mySqlCommand As New SqlCommand( ↵
    "select * from customers", mySqlConnection)

Dim myReader As SqlDataReader = mySqlCommand.ExecuteReader()
'...
If Not (myReader Is Nothing) Then
    myReader.Close()
End If

If mySqlConnection.State = ConnectionState.Open Then
    mySqlConnection.Close()
End If

```

Gestión de errores con un DataReader

Cuando utilizamos conexiones con el objeto **DataReader**, siempre deberíamos utilizar un bloque **Try/Finally** para garantizar que, si se lanza alguna excepción, las conexiones se cerrarán.

```

Dim myReader As SqlDataReader = Nothing

Dim mySqlConnection As New ↵
    SqlConnection("server=(local)\NetSDK; ↵
        Trusted_Connection=yes;database=northwind")

Dim mySqlCommand As New SqlCommand( ↵
    "select * from customers", mySqlConnection)

Try
    mySqlConnection.Open()
    myReader = mySqlCommand.ExecuteReader()
Catch e As Exception
    Console.WriteLine(e.ToString())
Finally
    If Not (myReader Is Nothing) Then
        myReader.Close()
    End If

    If mySqlConnection.State = ConnectionState.Open Then
        mySqlConnection.Close()
    End If
End Try

```

Leer datos de un DataReader

- Call read for each record
 - Returns **False** when there are no more records
- Get fields
 - Parameter is the ordinal position or name of the field

```
While myReader.Read()  
    Console.Write(myReader("CustomerID").ToString() + _  
                  " " )  
    Console.WriteLine(myReader("CompanyName").ToString())  
End While
```

- Call close to free up the reader and the connection

Una vez que hemos invocado el método **ExecuteReader** del objeto **Command**, accedemos a un registro en el **DataReader** invocando el método **Read**. La posición inicial en el **DataReader** se sitúa antes del primer registro; por tanto, debemos invocar a **Read** antes de acceder a cualquier dato. Cuando no hay más registros disponibles, el método **Read** devuelve **False**.

Leer campos desde el registro actual

Para recuperar datos desde campos en el registro actual, podemos llamar un método **Get** (por ejemplo: **GetDateTime**, **GetDouble**, **GetInt32** o **GetString**). El parámetro del método **Get** es el valor ordinal del campo que queremos leer. Por ejemplo, el siguiente código lee los campos *CustomerID* y *CompanyName* desde el registro actual del **DataReader** utilizando el método **GetString**:

```
Dim customerID As String = myReader.GetString(0)  
Dim companyName As String = myReader.GetString(1)
```

Podemos también referenciar los campos de datos en el registro actual del **DataReader** por nombre y a continuación llamar a una función de conversión apropiada, como se muestra en el siguiente ejemplo de código:

```
Dim customerID As String = myReader("CustomerID").ToString()  
Dim companyName As String = myReader("CompanyName").ToString()
```


Recorrer todos los registros

Para recorrer todos los registros y mostrarlos por la consola en un **DataReader**, podemos utilizar un bucle **While**, como se muestra en el siguiente código de ejemplo:

```
While myReader.Read()  
    Console.Write(myReader("CustomerID").ToString() + _  
        "      ")  
    Console.WriteLine(myReader("CompanyName").ToString())  
End While
```

Cerrar el DataReader

Mientras que el **DataReader** está en uso, la conexión asociada está ocupada sirviendo el **DataReader**. Por tanto, debemos llamar a **Close** para cerrar el **DataReader** cuando hayamos acabado de utilizarlo.

```
If Not (myReader Is Nothing) Then  
    myReader.Close()  
End If  
  
If mySqlConnection.State = ConnectionState.Open Then  
    mySqlConnection.Close()  
End If
```

Uso de DataSets frente a DataReaders

DataSet	DataReader
1. Create a database connection	1. Create a database connection
2. Store query in a DataAdapter	2. Open the database connection
3. Populate the DataSet with the Fill method	3. Store query in a SqlCommand
4. Create a DataView	4. Populate the DataReader with the ExecuteReader method
5. Bind the DataView to list-bound control	5. Call the Read method for each record, and the method Get for each field
	6. Manually display data
	7. Close the DataReader and the connection

El procedimiento general para acceder a bases de datos desde ASP.NET varía, dependiendo de si utilizamos un **DataSet** o un **DataReader**:

Usando DataSets	Usando DataReaders
1. Conectar a la base de datos utilizando SqlConnection u OleDbConnection .	1. Conectar a la base de datos utilizando SqlConnection u OleDbConnection .
2. Almacenar la consulta de la base de datos en un SqlDataAdapter o en un OleDbDataAdapter .	2. Abrir la conexión con el método Open .
3. Poblar un DataSet desde el DataAdapter utilizando Fill .	3. Almacenar la consulta de la base de datos en objetos SqlCommand u OleDbCommand .
4. Crear una nueva DataView para la tabla deseada.	4. Poblar un DataReader desde el Command utilizando el método ExecuteReader .
5. Enlazar un control servidor, como el DataGrid al DataView .	5. Invocar los métodos Read y Get del DataReader para leer datos.
	6. Visualizar manualmente los datos.
	7. Cerrar el DataReader y la conexión.