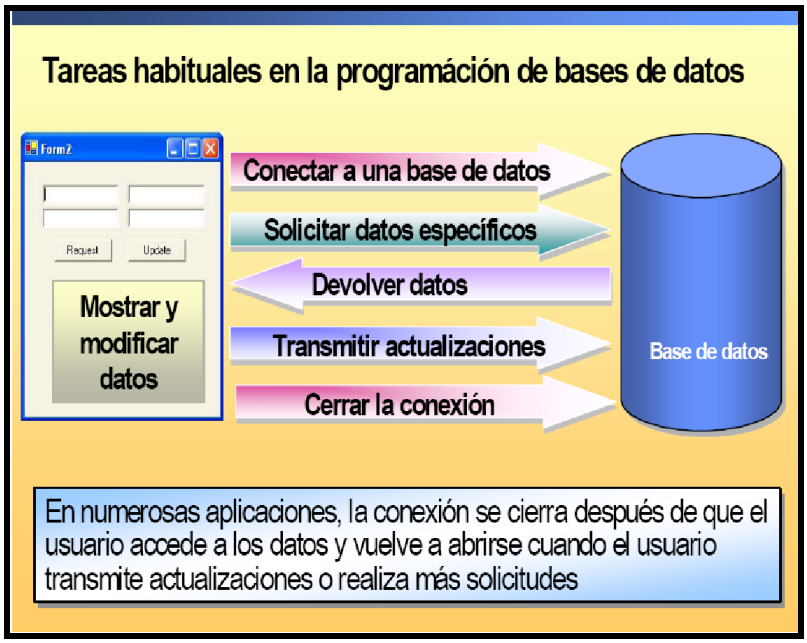
Cómo funciona la programación de bases de datos



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

¿Qué es un entorno conectado?

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.

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.

Un escenario conectado tiene los siguientes inconvenientes:

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

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?

Con el auge 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, Tablet PCs y PDAs 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.

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.

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.

Un entorno desconectado tiene los siguientes inconvenientes:

- Los datos no siempre están actualizados.
- Pueden producirse conflictos de cambios que deben solucionarse.
- 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 Windows del granjero que ejecuta Microsoft SQL Server.

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.

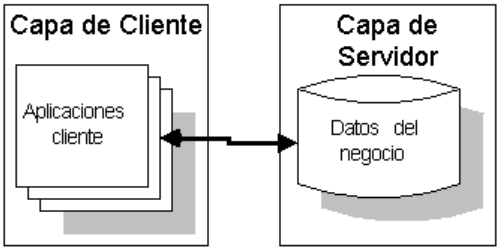
Las soluciones específicas pueden variar dependiendo de los requerimientos de negocio de una determinada aplicación.

ADO.NET

- ADO.NET es heredero de la tecnología ADO (ActiveX Data Objects)
- Supone la adaptación a .NET Framework de una arquitectura de base de datos que permitía acceder a bases de datos de cualquier proveedor en modo local y/o remoto.
- Permite su utilización bajo cualquier entorno que soporte .NET Framework (no sólo Windows).
- Basada en las clases base de la arquitectura .NET. puede ser utilizada bajo cualquier lenguaje .NET.
- Minimiza la carga de los servidores (en modo desconectado).

Un componente importante de ADO.NET es la capacidad que tiene de proporcionar a la aplicación acceso desconectado a los datos. Para entenderlo, aclaramos antes unos conceptos de arquitectura de aplicaciones..

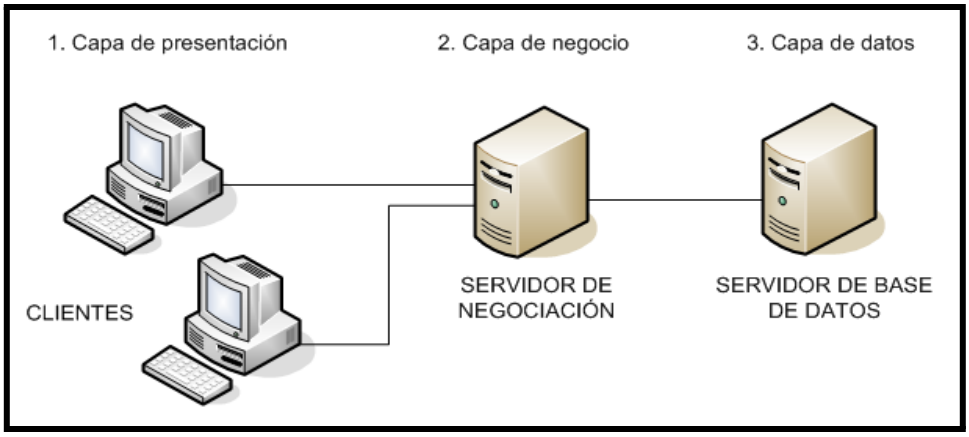
Arquitectura cliente/servidor:
Esta arquitectura consiste básicamente en que un programa -el Cliente informático- realiza peticiones a otro programa -el servidor- que le da respuesta. Esta idea se puede aplicar a programas que se ejecutan sobre una sola computadora. *La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa.*

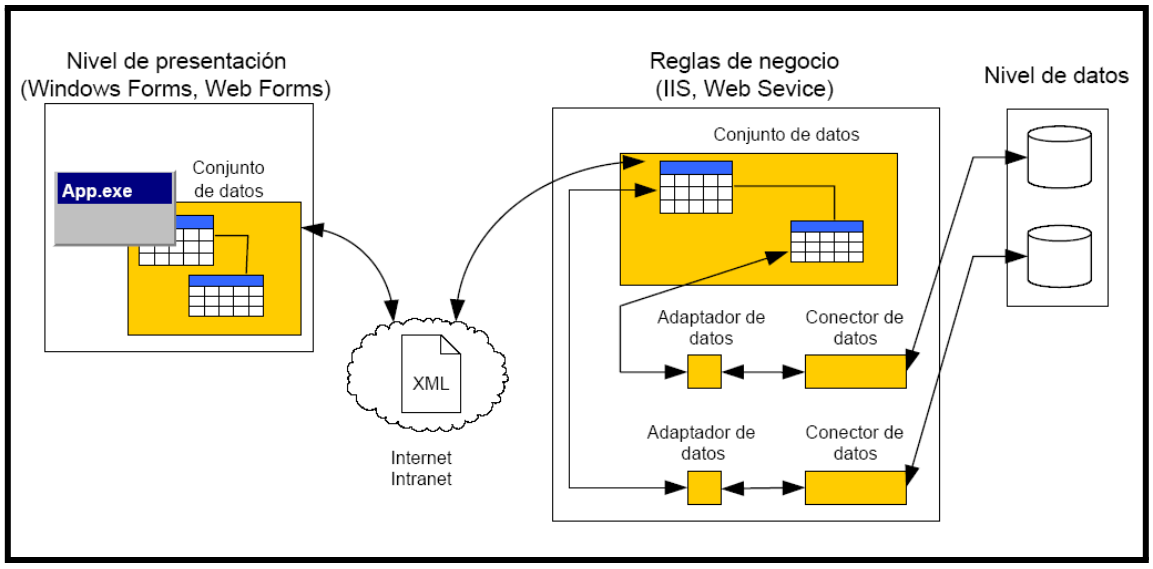


En este modelo de arquitectura, cada cliente se conecta con un servidor de base de datos y mantiene la conexión el tiempo necesario. La aplicación cliente, que suele ser un ejecutable Visual Basic o Visual C#, contiene toda la lógica de la aplicación. Las operaciones de base de datos se ejecutan directamente desde la aplicación contra el servidor de base de datos al que la aplicación se conecta. Este modelo cliente/servidor funciona bien, pero con los progresos de la www surge otro modelo conocido como **arquitectura multinivel** o de **n capas**

Una aplicación multinivel está dividida en capas que separan claramente **la interfaz, el negocio y la lógica de la base de datos.** Las aplicaciones se aíslan de las modificaciones de las bases de datos y de la lógica empresarial mediante el uso de componentes, capaces de minimizar e incluso, eliminar el impacto que producen en ellas. En la práctica, se crean componentes de lógica empresarial y de acceso a datos autónomos, aislando de sus complejidades a la aplicación, que muestra los datos y comprueba cómo interactúa con ellos el usuario final. Los componentes suelen ejecutarse en distintos servidores, aunque también pueden hacerlo en uno solo, con la base de datos en otro diferente.

En una arquitectura segmentada de esta forma, no es práctico ni deseable mantener una conexión de base de datos para todos y cada uno de los usuarios. En vez de ello, los datos pasan de capa en capa mientras están desconectados de la base de datos. La conexión se establece desde la capa de datos hasta el servidor de base de datos durante los breves instantes en que se produce la recuperación o la actualización de los datos, y luego se abandona inmediatamente la conexión.





Las ventajas que aporta esta arquitectura son:

- **Flexibilidad:** dado que la aplicación está dividida en partes lógicas, es muy fácil intercambiar capas. Por ejemplo, una capa de base de datos que se comunique con Access puede ser sustituida por otra capa de base de datos que se comunique con SQL Server sin que cambie por ello la capa de negocio.
- **Reutilización:** Dado que las aplicaciones suelen estar implementadas en capas, es muy fácil utilizar las diferentes capas en nuevas aplicaciones o compartirlas como servicio Web
- **Escalabilidad:** Esta es la razón por la que el modelo de n capas se adapta a un sitio Web mucho mejor que el modelo cliente/servidor. Si algo "escala bien" es que admite un aumento de usuarios simultáneos sin que suponga una merma

importante en el rendimiento. Dado que el usuario no está continuamente conectado con la base de datos, el servidor de base de datos puede admitir más usuarios. Las capas de una aplicación de este tipo pueden ser divididas con mucha facilidad entre múltiples computadoras físicas, creando una "batería de servidores", para que aumente la escalabilidad. *Compárese este planteamiento con el modelo cliente/servidor antiguo, donde lo único que se podía hacer para aumentar la escalabilidad era adquirir un servidor de base de datos más potente y agregar licencias de conexión adicionales.*

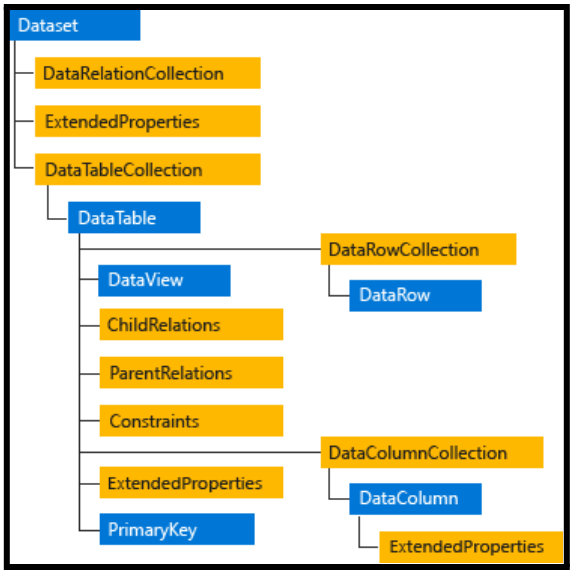
Microsoft diseñó ADO.NET con el fin de que fuera la opción ideal para una aplicación de varios niveles o capas, de ahí que por defecto su modo de trabajo sea **desconectado.**

Componentes ADO.NET

Los dos componentes principales de ADO.NET para acceder a los datos y manipular los datos son el ***DataSet*** y ***los proveedores de datos de .NET Framework***.

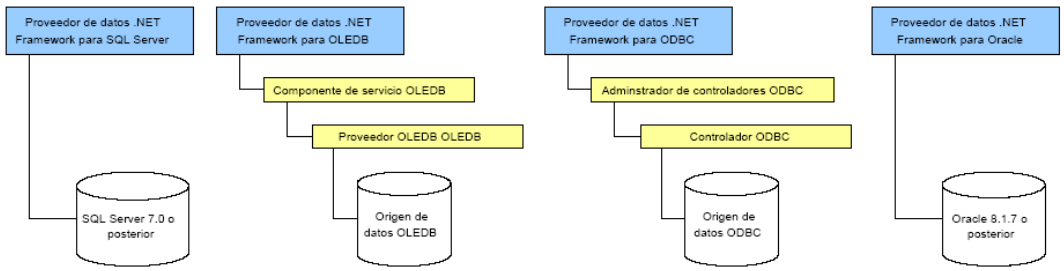
DataSet

Diseñado para el acceso a datos independientemente del origen de datos.



Proveedores de datos

- Se utilizan para conectarse a la base de datos, recuperar información y ejecutar órdenes contra la misma.
- Son dependientes del gestor de datos utilizado.
  - Proveedor de datos para SQL Server (nativo).
  - Proveedor de datos para Oracle (nativo).
  - Proveedor de datos para OleDB.
    - Utiliza los controladores OleDB para Windows.
    - Precisan de una capa adicional entre .NET y la base de datos.
  - Proveedor de datos para ODBC.

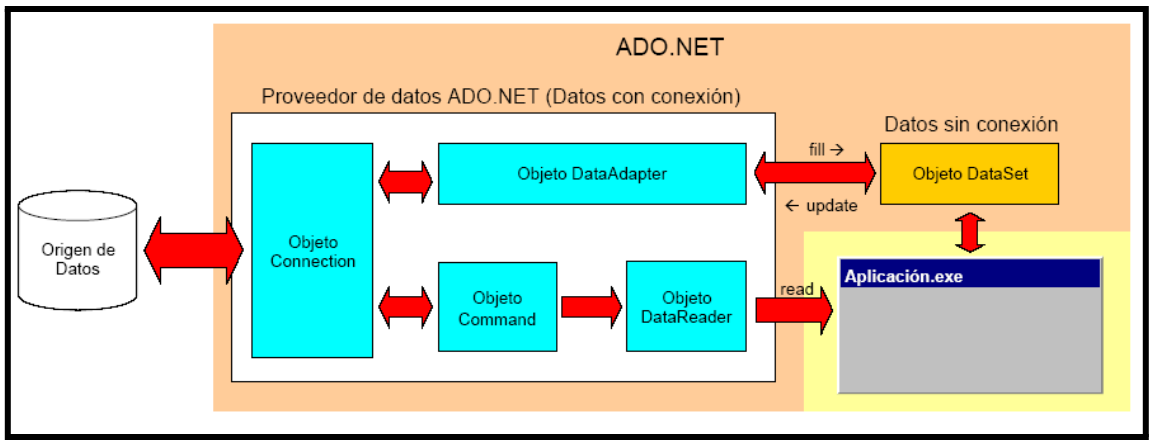


El modelo de objetos ADO.NET

Cinco objetos principales:

- 4 en el proveedor de datos *(dependen del gestor de bases de datos)*
  - Connection.
  - Command.
  - DataReader.
  - DataAdapter.

- 1 independiente del proveedor de datos.
  - DataSet.



- Objeto **Connection**: Establece la conexión con la base de datos mediante una "cadena de conexión".

- Objeto **Command**: Ejecuta una acción contra el almacén de datos, ya sea de consulta o de acción.

- Objeto **DataReader**: Conjunto de registros recuperado a partir del objeto Command.

- Objeto **DataAdapter**: Puente entre la conexión y los datos almacenados en un DataSet. Permite cargar los datos en el DataSet a partir de un origen de datos y actualizarlos.

- Objeto **DataSet**:
  - Objeto "abstracto" desligado de cualquier gestor de bases de datos.
  - Conjunto de tablas obtenidas mediante el método *Fill* del objeto DataAdapter.
  - Se puede considerar como una base de datos almacenada en la memoria caché del cliente.
    - Las tablas se cargan en la memoria caché del cliente, dejando disponible la conexión con el origen de datos para otros usuarios.

Espacios de nombres para ADO.NET

Las clases de acceso a datos de .NET Framework están en el espacio de nombres System.Data. En él se incluyen las clases que no pertenecen a ningún proveedor específico.

Existen clases distintas para cada sistema de gestión de bases de datos ubicadas en su propio espacio de nombres:

- **System.Data.OleDb.** Contiene objetos asociados al proveedor de datos OleDb como OleDbConnection, OleDbCommand, OleDbDataReader y OleDbDataAdapter.

- **System.Data.SqlClient.** Contiene objetos asociados al proveedor de datos SQL Server como SqlDbConnection, SqlDbCommand, SqlDbDataReader y SqlDbDataAdapter.

- **System.Data.Odbc**. Contiene objetos asociados al proveedor de datos de ODBC como OdbcConnection, OdbcCommand, OdbcDataReader y OdbcDataAdapter.

- System.Data.OracleClient**.** Contiene objetos asociados al proveedor de datos Oracle como OracleConnection, OracleCommand, OracleDataReader y OracleDataAdapter.

Modo conectado y modo desconectado

En cuanto a la conectividad se refiere, en el modelo de datos de ADO.NET, se observa dos formas de trabajar con los datos de una base de datos: conectado a la base de datos o desconectado de la misma. Por defecto el modo de trabajo es desconectado.

Modo desconectado:

- La conexión sólo es necesario establecerla cuando se descarga información del origen de datos.
- Cada tabla del *DataSet* precisa de un objeto DataAdapter.
  - ✓ El método *Fill* se encargará de cargar una tabla en el *DataSet*.
  - ✓ Los datos se almacenan en la memoria caché del cliente en un objeto *DataSet*.
  - ✓ Los datos almacenados en el *DataSet* se pueden modificar.
- Las modificaciones efectuadas en el *DataSet* se pueden sincronizar con el origen de datos.

- ✓ El método *Update* del objeto DataAdapter permite actualizar el origen de datos.

- La información entre el cliente y el servidor se transmite en forma de datos XML (pueden ser utilizados por otra aplicación).
- Se utiliza cuando:
  - ✓ Se necesita modificar los datos frecuentemente.
  - ✓ Es necesario que los datos estén mucho tiempo en memoria (por ejemplo en aplicaciones Windows Form).
  - ✓ Cuando no siempre es posible estar conectado al origen de datos (aplicaciones móviles).

Modo conectado:

- Utiliza los objetos Connection, Command y DataReader.
- Se establece una conexión permanente con el origen de datos.
- A partir de una conexión, el objeto Command generará un objeto DataReader con la información necesaria.
  - ✓ Los datos del objeto DataReader son de sólo lectura.
- El objeto Command, también se encargará de realizar las operaciones de actualización con la base de datos.
- La información entre el cliente y el servidor se establece en un formato binario propietario del gestor de base de datos.
- Se utiliza cuando se deben procesar los registros en un corto espacio de tiempo:
  - ✓ Realización de informes.

Uso de DataSet frente a DataReader

DataSet	DataReader
Acceso lectura/escritura a datos	Sólo lectura
Incluye múltiples tablas de distintas bases de datos	Basado en una instrucción SQL de una base de datos
Desconectado	Conectado
Vinculado a múltiples controles	Vinculado a un único control
Búsqueda de datos hacia delante y hacia atrás	Sólo hacia delante
Acceso más lento	Acceso más rápido
Soportado por las herramientas de Visual Studio .NET	Codificación manual

Los objetos **DataSet** son objetos complejos que nos permiten almacenar múltiples tablas de datos DataTables desde una fuente de datos. Los objetos DataSet son como una base de datos virtual ubicada dentro de la aplicación. Los objetos DataSet también pueden contener relaciones entre los datos de las DataTables, y pueden utilizar esas relaciones para recuperar datos.

Los objetos **DataReader** son objetos ligeros que se utilizan para leer datos desde una fuente de datos; los objetos DataReader proporcionan acceso sólo hacia delante (forward-only) y de sólo lectura (read-only) a los datos de una base de datos.

La elección entre utilizar objetos DataSet u objetos DataReader debería basarse en el uso previsto para los datos. Normalmente, los objetos DataReader se utilizan para leer datos en situaciones en las que es necesario el acceso una única vez, y de solo lectura, como cuando accedemos a una contraseña almacenada, o se cumplimenta un control enlazado a una lista. Los objetos DataSet se utilizan para un acceso a datos más complejo, como el acceso a todo el historial de pedidos de un cliente.

Algunos de los aspectos relativos al acceso a datos que se deben tener en cuenta a la hora de decidir entre objetos DataSet y DataReader incluyen:

- Acceso a datos:

Si nuestra intención es leer y escribir a nuestra fuente de datos, debemos utilizar un objeto DataSet. Los objetos DataReader son conexiones de sólo lectura y deberían utilizarse únicamente cuando los datos vayan a utilizarse en una situación de sólo lectura.

- Acceso a múltiples bases de datos:

Si nuestra intención es combinar tablas de una o más bases de datos, debemos utilizar un objeto DataSet. Los objetos DataReader se basan en una única instrucción SQL de una sola base de datos.

- Enlace a controles:

Si nuestra intención es enlazar los datos a más de un control, debemos utilizar un objeto DataSet. Los objetos DataReader sólo pueden vincularse a un único control.

- Modo conexión:

Si nuestra intención es trabajar en un modo desconectado, debemos utilizar un objeto DataSet. Los objetos DataReader deben ejecutarse en modo conectado.

- Búsqueda (scanning) de datos:

Si nuestra intención es buscar los datos hacia atrás y hacia delante, debemos utilizar un objeto DataSet. Los objetos DataReader buscan hacia adelante a medida que los datos fluyen desde la base de datos.

- Velocidad de acceso:

Si necesitamos acceso de alta velocidad a nuestra fuente de datos, utilizaremos un objeto DataReader. Los objetos DataSet son más lentos que los objetos DataReader en el acceso a una base de datos. También la sobrecarga es mayor en la creación del objeto DataSet debido a la capacidad de leer y escribir datos y búsqueda hacia delante y hacia atrás. Los objetos DataReader son más rápidos debido a la naturaleza del objeto más ligera. Hay muy poca sobrecarga para el objeto DataReader, ya que éste trabaja sólo hacia delante y sólo de lectura.

- Soporte de herramientas

Si nuestra intención es utilizar Microsoft Visual Studio® .NET para crear la conexión a datos, utilizaremos objeto DataSet. Con los objetos DataSet, podemos elegir entre

escribir nuestro propio código o utilizar el código máquina de Visual Studio .NET. Con los objetos DataReader, debemos escribir todo el código de soporte.

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. Más info en **Elegir un DataReader o un DataSet**

Conexión con la base de datos

Imprescindible tanto en modo conectado como desconectado. Pasos para crear una conexión:

1. Crear una instancia de algunas de las clases Connection.
2. Establecer la cadena de conexión mediante la propiedad ConnectionString.
3. Abrir la conexión:
   - ✓ En modo conectado la conexión permanecerá abierta hasta que la aplicación termine de trabajar con los datos.
   - ✓ En modo desconectado se cargarán los datos en un objeto DataSet y se cerrará la conexión. Una vez almacenados los datos en el DataSet, la aplicación trabajará con ellos.

Veamos detalladamente cada uno de estos **pasos**:

1. Crear una instancia de las clases:

- ✓ Para el proveedor de datos SQL:
  Dim cn As New SqlConnection

- ✓ Para el proveedor de datos OleDb:
  Dim cn as New OleDbConnection

Notas:

- ✓ Es necesario tener establecido el espacio de nombres o bien utilizar el nombre cualificado (*System.Data.SqlClient.SqlConnection*).
- ✓ Dependiendo del alcance que queramos dar a las variables utilizaremos los modificadores *Dim, Private, Public, Friend,* etc.

2. Cadena de conexión:

Todas las clases Connection de todos los proveedores tienen la propiedad *ConnectionString.* El valor de la propiedad será una expresión de cadena formada por parejas de nombres de argumentos y valores, separados por un punto y coma.
nombreArgumento = valor;...

Argumentos para una conexión para el proveedor de SQL Server:

Argumento	Valor
`Data Source` o `Server`	Nombre del servidor de base de datos.
`Inicial Catalog` o `Database`	Nombre de la base de datos a la que se va a conectar
`Integrated Security`	Si se pone a `false` se debe especificar el nombre de usuario y la contraseña; si se pone a `true` se utilizarán las credenciales de la cuenta de Windows. Los valores permitidos son `true`, `false`, `yes`, `no` y `sspi` (recomendada, equivalente a `true`)
`Persist Security Info`	Si se pone a `false` (recomendado) la información de seguridad no se devuelve como parte de la conexión.
`User ID`	Nombre de usuario de una cuenta registrada en SQL Server
`Pwd`	Contraseña de inicio de sesión para una cuenta de SQL Server

Argumentos para una conexión para el proveedor de OleDb para bases de datos Access:

Argumento	Valor
`PROVIDER`	Nombre del proveedor OleDb (para Access se utiliza `Microsoft.Jet.OleDb.4.0`)
`Data Source`	Especificación de archivo donde se encuentra el archivo .mdb

En la siguiente tabla se muestra la sintaxis de **autenticación de Windows** utilizada con los proveedores de datos .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` produce una excepción cuando se usa con el proveedor **OleDb**.

Ejemplos de cadenas de conexión:

```
'Cadena de conexión para el proveedor SQLServer
cnSQL.ConnectionString = "Server=LUIS\SQLEXPRESS;Database=Ejemplo;" & _
                  "Integrated Security=sspi;Persist Security Info=yes;" & _
                  "User ID=LuisR;pwd="
'Cadena de conexión para el proveedor OLEDB para bases de datos Access
cnOleDB.ConnectionString = "PROVIDER=Microsoft.Jet.OleDb.4.0; " & _
                           "Data Source=c:\BBDD\ejemplo.mdb"
```

3. Abrir y cerrar la conexión:

- cn.Open()
- cn.Close()

Consideraciones:

La acción más pesada cuando realizamos un acceso a una base de datos se encuentra en la conexión a la misma. En cualquiera de los dos modos de trabajo (conectado/desconectado), es imprescindible el establecimiento de esta conexión. Y esta tarea tan simple es la que más recursos consume. Por ello, es bueno tener presente las siguientes consideraciones:

- La conexión debe realizarse, siempre que se pueda con los **proveedores de acceso a datos nativos**, simplemente porque son más rápidos que los proveedores del tipo OLE DB Y ODBC.
- La conexión debe abrirse lo más tarde posible. Es recomendable definir todas las variables que podamos antes de realizar la conexión.
- La conexión debe cerrarse lo antes posible, siempre y cuando no tengamos necesidad de utilizarla posteriormente.