

# Interacción persona-computadora



## Tema 9. ADO.NET

Luís Rodríguez Baena ([luis.rodriguez@upsam.net](mailto:luis.rodriguez@upsam.net))

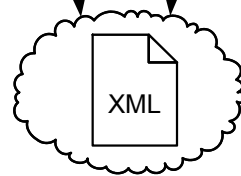
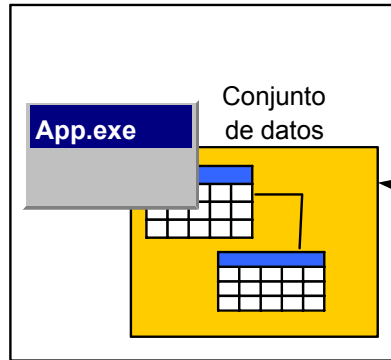
Universidad Pontificia de Salamanca (campus Madrid)  
Facultad de Informática

# Introducción

- ❑ ADO.NET es heredero de la tecnología ADO (ActiveX Data Objects) implantada por Microsoft hacia el año 2002.
- ❑ Supone la adaptación a .NET Framework de una arquitectura de base de datos que permitía acceder a bases de datos de cualquier proveedor ya se en modo local o remoto.
- ❑ ADO.NET.
  - Permite su utilización bajo cualquier entorno que soporte .NET Framework (no sólo bajo Windows).
  - Basada en las clases base de la arquitectura .NET.
    - ✓ Puede ser utilizada bajo cualquier lenguaje .NET.
  - Permite la ampliación a aplicaciones basadas en el modelo cliente/servidor.
  - Minimiza la carga de los servidores (modo desconectado).

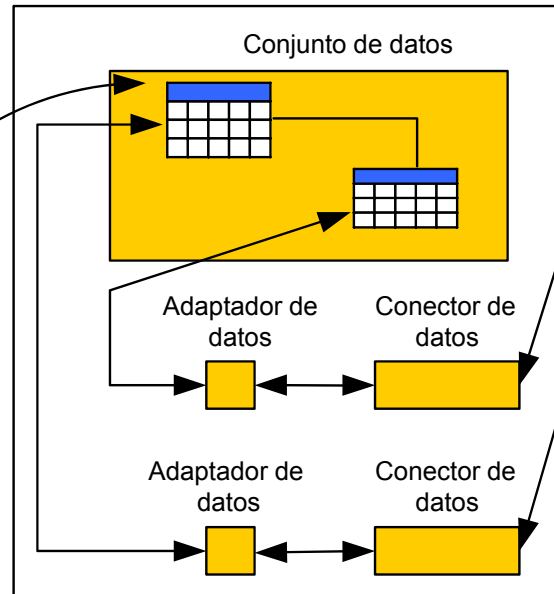
# Componentes ADO.NET

Nivel de presentación  
(Windows Forms, Web Forms)

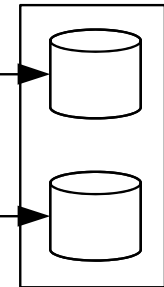


Internet  
Intranet

Reglas de negocio  
(IIS, Web Service)

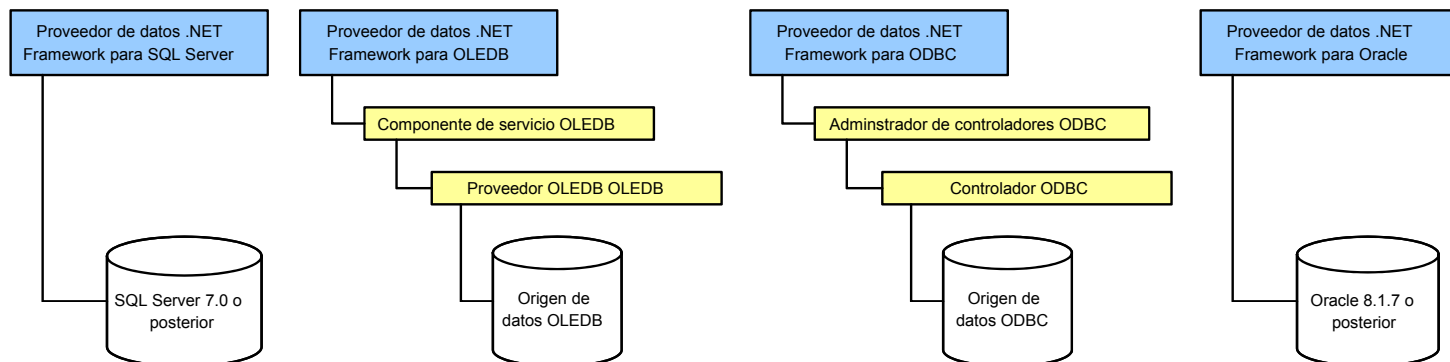


Nivel de datos



# Proveedores de datos

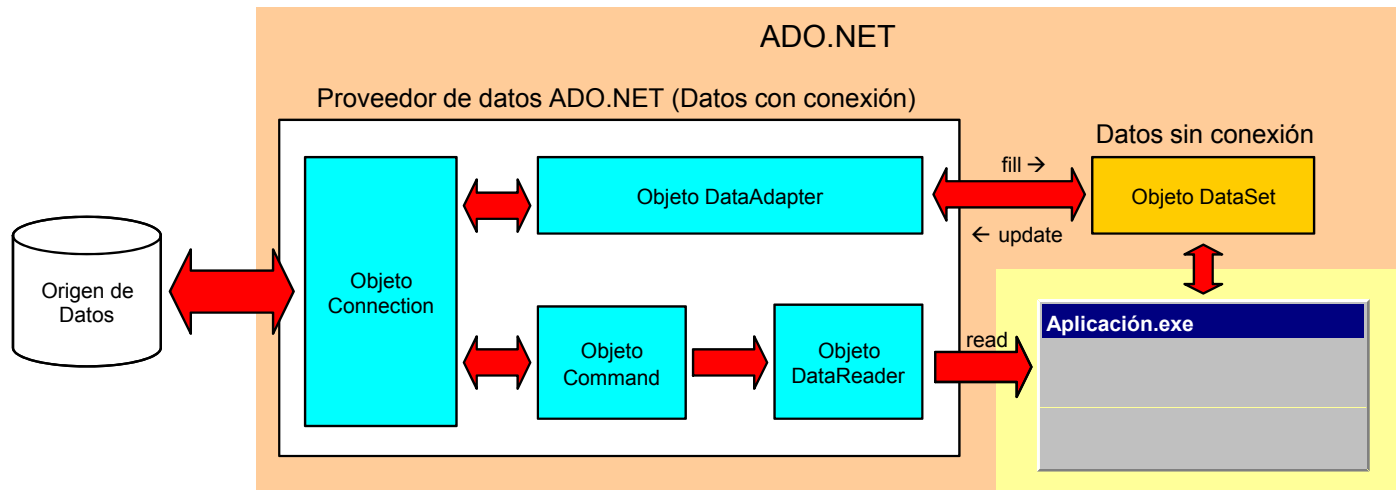
- ❑ Se utilizan para conectarse a la base de datos, recuperar información y ejecutar órdenes contra la misma.
- ❑ Dependientes del gestor de datos utilizado.
  - Proveedor de datos para SQL Server.
  - Proveedor de datos para OLEDB.
    - ✓ Utiliza los controladores nativos OLEDB para Windows.
    - ✓ Precisan de una capa adicional entre .NET y la base de datos.
  - Proveedor de datos para ODBC.
  - Proveedor de datos para Oracle (a partir de .NET Framework 1.1).



# Modelo de objetos ADO.NET

## ❑ Cinco objetos principales:

- En el proveedor de datos (dependen del gestor de bases de datos).
  - ✓ Connection.
  - ✓ Command.
  - ✓ DataReader.
  - ✓ DataAdapter.
- Independiente del proveedor de datos.
  - ✓ DataSet.



# Modelo de objetos ADO.NET (II)

## 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`.

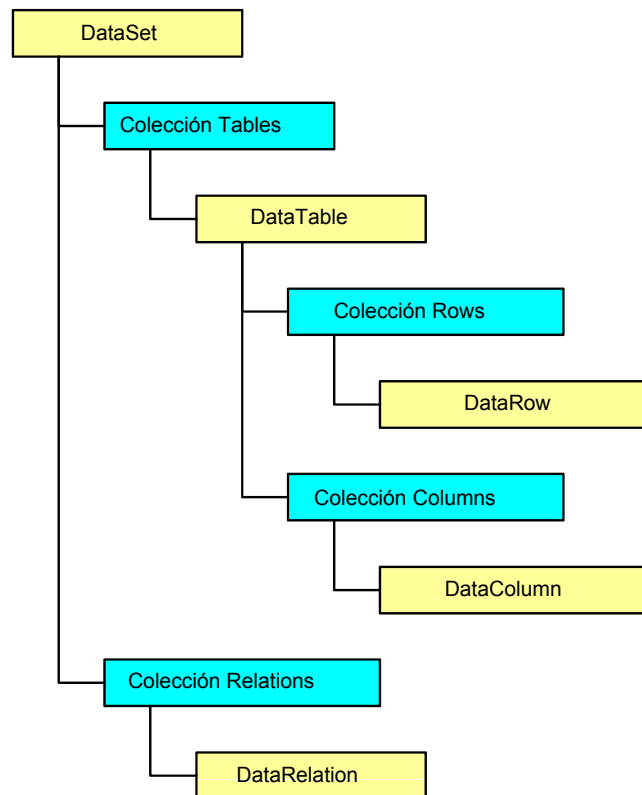
## Objeto `DataAdapter`.

- 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.

# Modelo de objetos ADO.NET (III)

## ❑ 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`.
  - Incluye las clases que no pertenecen a ningún proveedor específico.
- ❑ Los nombres de las clases pertenecientes al proveedor de datos .NET son nombres genéricos.
  - 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 `SqlConnection`, `SqlCommand`, `SqlDataReader` y `SqlDataAdapter`.
    - ✓ `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

## ❑ 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 pocos registros durante poco tiempo o cuando no sea necesario compartir la información con otras aplicaciones.
  - ✓ Realización de informes.
  - ✓ Páginas dinámicas ASP de Internet.

# Modo conectado y modo desconectado (II)

## ❑ 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).

# Conexión con la base de datos

- ❑ Imprescindible tanto en modo conectado como desconectado.
- ❑ Pasos para crear una conexión:
  - Crear una instancia de algunas de las clases `Connection`.
  - Establecer la cadena de conexión mediante la propiedad `ConnectionString`.
  - Abrir la conexión.
    - ✓ En modo conectado la conexión permanecerá abierta hasta que termine la aplicación.
    - ✓ En modo desconectado se cargarán los datos en un objeto `DataSet` y se cerrará la conexión.

# Conexión con la base de datos (II)

## ❑ Crear una instancia de las clases.

- Para el proveedor de datos SQL.

```
Dim cnSQL As New SqlConnection
```

- Para el proveedor de datos OleDb.

```
Dim cnOleDb as New OleDbConnection
```

- Notas:

- ✓ Los nombres de las instancias serán `cnSQL` y `cnOleDb` respectivamente.
- ✓ 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.

# Conexión con la base de datos (III)

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

# Conexión con la base de datos (IV)

## ❑ Cadena de conexión (*continuación*).

- 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 <code>Microsoft.Jet.OleDb.4.0</code> )
Data Source	Especificación de archivo donde se encuentra el archivo <code>.mdb</code>

- 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=false;" & _
                        "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"
```

## ❑ Abrir y cerrar la conexión.

- `cnOleDb.Open()`
- `cnOleDb.Close()`

# Trabajar en modo conectado

## ❑ Recuperar filas.

- Se realiza con el método `ExecuteReader` de la clase `Command`.
  - ✓ `Command` es una clase del proveedor de datos, por lo que es preciso elegir la subclase adecuada (`OleDbCommand`, `SqlCommand`, `OdbcCommand` u `OracleCommand`).
  - ✓ En el constructor se pasaría la sentencia SQL de recuperación (sentencia `SELECT`) necesaria para recuperar filas.
  - ✓ `ExecuteReader` devuelve un objeto de la clase `DataReader`.
    - También es una clase del proveedor de datos.
    - Representa el conjunto de filas recuperados por la `SELECT`.
    - Es sólo de lectura y sólo se puede ir a la siguiente fila.

```
'Cadena de conexión para el proveedor SQLServer
Dim cnSQL As New SqlConnection
cnSQL.ConnectionString = "Server=LUIS\SQLEXPRESS;Database=Ejemplo;" & _
                        "Integrated Security=sspi;Persist Security Info=yes;" & _
                        "User ID=LuisR;pwd="

cnSQL.Open()
'Crear una orden de recuperación
Dim miOrden As SqlCommand = New SqlCommand("SELECT * FROM Clientes", cnSQL)
'Crear el DataReader
Dim drClientes As SqlDataReader
'Ejecutar la consulta
drClientes = miOrden.ExecuteReader
```

# Trabajar en modo conectado (II)

## ❑ Acceder a las filas recuperadas.

- El método `Read` carga en el objeto `DataReader` la siguiente fila.
  - ✓ Devuelve `false` si no hay más filas.
- Los métodos `Getxxx`, permiten acceder al contenidos de las columnas.
  - ✓ Hay un método `Getxxx` por cada tipo de datos.
  - ✓ Utiliza como argumento el número de columna al que se accede.

```
Console.WriteLine("Listado de clientes")
Console.WriteLine("-----")
Console.WriteLine("{0,-6} {1,-25} {2,-10} {3,-20} {4,-10} ", _
    "Codigo", "Apellidos", "Nombre", "Ciudad", "Provincia")
Console.WriteLine(New String("-", 75))
Do While drClientes.Read()
    Console.WriteLine("{0,6} {1,-25} {2,-10} {3,-20} {4,-10} ", _
        drClientes.GetInt32(0), _
        drClientes.GetString(2), _
        drClientes.GetString(1), _
        drClientes.GetString(3), _
        drClientes.GetString(4))
Loop
drClientes.Close()
```



# Trabajar en modo conectado (III)

## ❑ Ejecutar funciones agregadas.

- El método `ExecuteScalar` de la clase `Command` devuelve sólo la primera columna de una sentencia `SELECT`.

```
'Obtener el número total de registros de la tabla dbo.Clientes
miOrden = New SqlCommand("SELECT COUNT(*) FROM Clientes", cnSQL)
Console.WriteLine("Número de filas: " & miOrden.ExecuteScalar)
```

## ❑ Ejecutar ordenes SQL de actualización.

- Se utiliza el método `ExecuteNonQuery` de la clase `Command`.
- Se ejecuta la sentencia SQL con la que se cree el objeto `Command`.

```
'Modificar la Ciudad del cliente con código 10101
miOrden = New SqlCommand("UPDATE Clientes SET Ciudad='Altea' WHERE IdCliente=10101", cnSQL)
miOrden.ExecuteNonQuery()
'Añadir el cliente con IdCliente 10104
miOrden = New SqlCommand("INSERT INTO Clientes " & _
                        "(IdCliente,Apellido,Nombre,Ciudad,Provincia)" & _
                        "VALUES (10104,'Ordóñez','Manuel','Tarrasa','Barcelona','08227')", cnSQL)
miOrden.ExecuteNonQuery()
'Eliminar el cliente con código 10298
miOrden = New SqlCommand("DELETE FROM Clientes WHERE IdCliente=10298", cnSQL)
miOrden.ExecuteNonQuery()
```

# Modo desconectado.

## Adaptadores de datos

- ❑ En modo desconectado los adaptadores de datos se utilizan para relacionar una conexión con un conjunto de datos.
  - Adapta los datos del formato nativo del gestor de bases de datos para que puedan ser utilizados en el DataSet (XML).
    - ✓ Carga las diferentes tablas en el DataSet.
    - ✓ Actualiza las modificaciones del DataSet en el origen de datos.
  - Normalmente se utilizará un adaptador de datos por cada tabla que queramos recuperar del origen de datos.

- ❑ Crear la instancia del adaptador de datos.

- Constructor.

```
Dim nombreAdaptador As xxxDataAdapter
```

```
nombreAdaptador = New xxxDataAdapter(selectSQL, conexión)
```

- ✓ xxxDataAdapter es alguna de las clases SqlDataAdapter, OleDbDataAdapter, OdbcDataAdapter y OracleDataAdapter.
- ✓ selectSQL es una cadena con la instrucción SQL que recuperará los datos de la tabla que se añadirá al DataSet.
- ✓ conexión es un objeto Connection ya abierto.

# Adaptadores de datos (II)

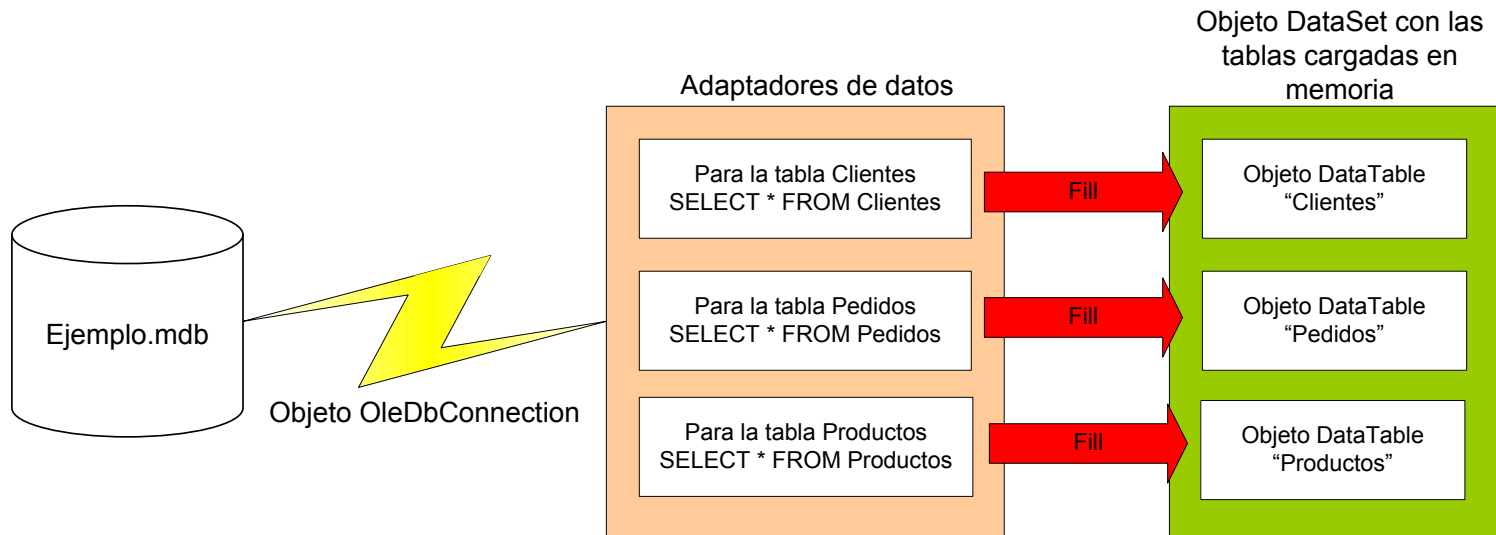
## ❑ Rellenar el conjunto de datos

- El adaptador de datos permite cargar las tablas recuperadas a partir de la sentencia SQL en objetos de tipo `DataTable` del conjunto de datos.
- El método `Fill` permite realizar la carga de datos.

*objetoDataAdapter.Fill(objetoDataSet, nombreObjetoDataTable)*

- ✓ Precisa la existencia de un objeto de la clase `DataSet`.
- ✓ *nombreObjetoDataTable* es una expresión de cadena que identificará la tabla dentro del conjunto de datos.
- ✓ Los datos que se cargarán en la tabla serán los que se recuperen mediante la orden SQL del constructor del adaptador de datos.

# Conexiones, adaptadores y conjuntos de datos



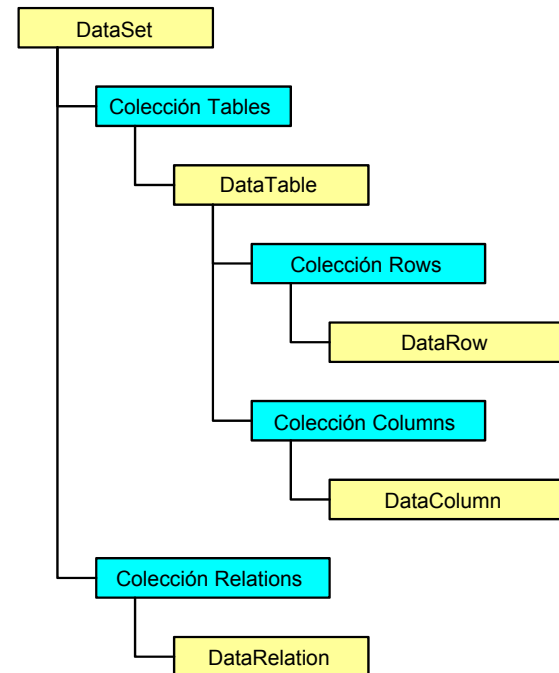
# Conexiones, adaptadores y conjuntos de datos (II)

- ❑ Se conecta a una base de datos Access y carga en el conjunto de datos las tablas Clientes y Pedidos.

```
'Establece los espacios de nombre
Imports System.Data
Imports System.Data.OleDb
...
'Declaración de variables. Dependiendo de su alcance pueden llevar otros modificadores
Dim cn As New OleDbConnection
Dim daClientes As OleDbDataAdapter
Dim daProductos As OleDbDataAdapter
Dim daPedidos As OleDbDataAdapter
Dim ds As New DataSet
' Cadena de conexión para el proveedor OleDb para bases de datos Access
cn.ConnectionString = "PROVIDER=Microsoft.Jet.Oledb.4.0;" & _
    "Data Source=C:\BBDD\Ejemplo.mdb"
cn.Open()
' Crear los adaptadores de datos
daClientes = New OleDbDataAdapter("SELECT * FROM Clientes", cn)
daPedidos = New OleDbDataAdapter("SELECT * FROM Pedidos", cn)
daProductos = New OleDbDataAdapter("SELECT * FROM Productos", cn)
' Rellenar el Dataset
daClientes.Fill(ds, "Clientes")
daPedidos.Fill(ds, "Pedidos")
daProductos.Fill(ds, "Productos")
' Una vez cargado el dataset se puede cerrar la conexión
cn.Close()
```

# La clase DataSet

- ❑ Almacena en la memoria caché del cliente los resultados de la consulta SQL establecida en un adaptador de datos.
  - Los datos están disponibles en modo desconectado.
  - Forman una pequeña base de datos con la información necesaria para la aplicación.



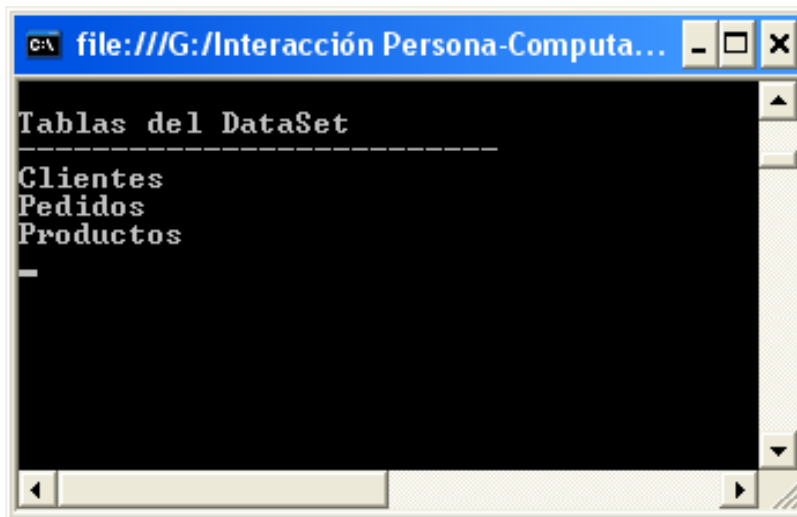
# La clase DataSet (II)

## □ La colección Tables.

- Cada tabla añadida por el método `Fill` del adaptador de datos formará un objeto de tipo `DataTable`.
- La propiedad `Tables` de la clase `DataSet` permite acceder al conjunto de objetos `DataTable` cargados.
- Se puede hacer referencia a cada una de las tablas indicando el índice de la misma o mediante el nombre.
  - ✓ Para hacer referencia a la primera tabla cargada (Clientes)  
`ds.Tables(0)`
  - ✓ Para hacer referencia a la tabla Pedidos  
`ds.Tables("Pedidos")`

# La clase DataSet (III)

```
'Ejemplo: obtener las tablas contenidas en el DataSet
Console.WriteLine("Tablas del DataSet")
Console.WriteLine("-----")
For Each dt As DataTable In ds.Tables
    Console.WriteLine(dt.TableName)
Next
```



```
file:///G:/Interacción Persona-Computa...
Tablas del DataSet
-----
Clientes
Pedidos
Productos
-
```

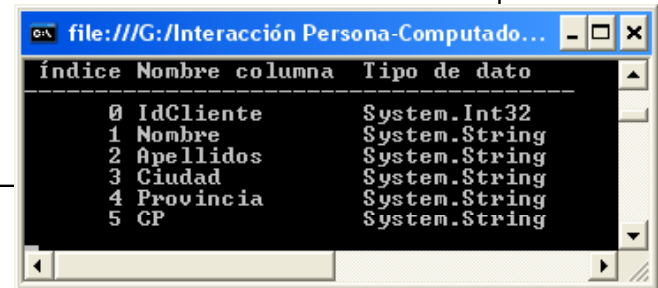


# La clase DataTable

❑ Cada tabla (objeto `DataTable`) está formada por:

- Una colección de columnas (colección de objetos `DataColumn`).
  - ✓ La propiedad `Columns` permite acceder a cada una de las columnas.
  - ✓ Cada columna guarda información de las características de cada uno de los campos (tipo de dato, longitud, etc.).
  - ✓ Es posible acceder a ellas mediante el índice o mediante el nombre de la columna.

```
'Obtener las columnas de la tabla Clientes
Console.WriteLine("Columnas de la tabla Clientes")
Console.WriteLine("-----")
Console.WriteLine()
Console.WriteLine("{0,7} {1,-15} {2,-15}", _
    "Índice", "Nombre columna", "Tipo de dato")
Console.WriteLine(New String("-", 39))
For i As Integer = 0 To ds.Tables(0).Columns.Count - 1
    Console.WriteLine("{0,7} {1,-15} {2,-15}", i, _
        ds.Tables(0).Columns(i).ColumnName, _
        ds.Tables(0).Columns(i).DataType)
Next
```



Índice	Nombre columna	Tipo de dato
0	IdCliente	System.Int32
1	Nombre	System.String
2	Apellidos	System.String
3	Ciudad	System.String
4	Provincia	System.String
5	CP	System.String

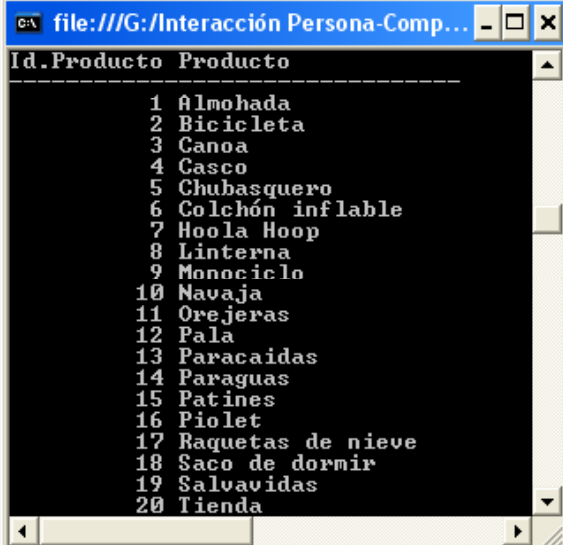
# La clase DataTable (II)

- ❑ Una colección de filas (colección de objetos DataRow).
  - La propiedad Rows permite acceder a cada una de las filas.
  - Cada fila de la colección Rows está identificada por su índice (posición de la fila dentro de la colección).
    - ✓ Es posible acceder a ellas de forma similar a un array.
  - El valor de cada campo está disponible mediante la colección Item del objeto DataRow.
    - ✓ Accedemos a cada campo por su índice o por el nombre del campo.
- ❑ Obtener la información de todas las filas de una tabla.

```
'Listado de la tabla Clientes
Console.WriteLine("Listado de la tabla Clientes")
Console.WriteLine("-----")
Console.WriteLine("{0,-6} {1,-20} {2,-10} {3,-20} {4,-11} {5,-5} ", _
                  "Codigo", "Apellidos", "Nombre", "Ciudad", "Provincia", "CP")
Console.WriteLine(New String("-", 79))
For i As Integer = 0 To ds.Tables(0).Rows.Count - 1
    Console.WriteLine("{0,-6} {1,-20} {2,-10} {3,-20} {4,-11} {5,-5} ", _
                      ds.Tables(0).Rows(i).Item("IdCliente"), _
                      ds.Tables(0).Rows(i).Item("Apellidos"), _
                      ds.Tables(0).Rows(i).Item("Nombre"), _
                      ds.Tables(0).Rows(i).Item("Ciudad"), _
                      ds.Tables(0).Rows(i).Item("Provincia"), _
                      ds.Tables(0).Rows(i).Item("CP"))
Next
```

# La clase DataTable (III)

```
'Listado de la tabla Productos
Console.WriteLine("Listado de la tabla Productos")
Console.WriteLine("-----")
Console.WriteLine()
Console.WriteLine("{0,-11} {1,-20}", "Id.Producto", "Producto")
Console.WriteLine(New String("-", 32))
For Each dr As DataRow In ds.Tables(2).Rows
    Console.WriteLine("{0,11} {1,-20}", _
        dr.Item("IdProducto"), dr.Item("Producto"))
Next
```



```
file:///G:/Interacción Persona-Comp...
Id.Producto  Producto
-----
1 Almohada
2 Bicicleta
3 Canoa
4 Casco
5 Chubasquero
6 Colchón inflable
7 Hoola Hoop
8 Linterna
9 Monociclo
10 Navaja
11 Orejeras
12 Pala
13 Paracaidas
14 Paraguas
15 Patines
16 Piolet
17 Raquetas de nieve
18 Saco de dormir
19 Salvavidas
20 Tienda
```

# Filtrar y ordenar registros en un DataTable

- ❑ El método `Select` del objeto `DataTable` devuelve una colección de objetos `DataRow` con los registros seleccionados.
  - Para devolver un conjunto de filas que cumplan el filtro  
`objetoDataTable.Select(expresiónFiltro)`
  - Para obtener un conjunto de filas que cumplan un filtro ordenados a partir de uno o varios campos.  
`objetoDataTable.Select(expresiónFiltro,expresiónSort)`
  - `expresiónSort` es una cadena que contiene los nombres de las columnas por los que queremos ordenar separadas por comas.
    - ✓ El criterio de ordenación es ascendente. Cada nombre de columna puede ir seguido de las claves `ASC` o `DESC` para indicar el tipo de ordenación.
      - "Apellido", ordena por la columna apellidos.
      - "Apellido,Nombre", ordena por las columnas Apellido y Nombre
      - "Provincia DESC, IdCliente", ordena descendentemente por Provincia y a continuación, de forma ascendente por IdCliente.

# Filtrar y ordenar registros en un DataTable (II)

- *expresiónFiltro* es una cadena con una expresión lógica que contiene el criterio de búsqueda en el siguiente formato:

*nombreColumna opRelación valor*

- La columna y el valor deben tener el mismo tipo de dato.
- Los operadores de relación serán =, <, >, <=, >=, <> o LIKE.
- Es posible unir varias expresiones con los operadores AND u OR.

- Para valores numéricos:

"Precio > 100"

- Para valores de cadena, el valor se debe encerrar entre comillas simples.

"Ciudad = "'Madrid'"

- Para valores de fecha, es necesario encerrar las fechas entre almohadillas.

"Fecha > #23/04/2004#"

- El operador LIKE compara una columna con un patrón. En el patrón se pueden utilizar comodines como el % o el \*.

"Ciudad LIKE 'M%'"

# Filtrar y ordenar registros en un DataTable (III)

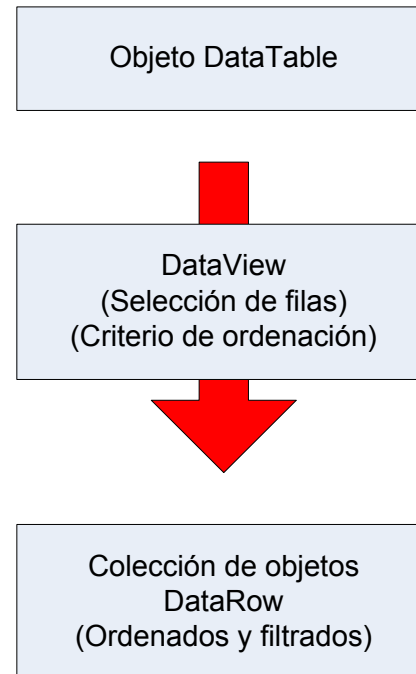
```
'Filtrar los Clientes cuya provincia empiece por M y ordenarlos por apellidos
Console.WriteLine("{0,-6} {1,-20} {2,-10} {3,-20} {4,-11} {5,-5} ", _
                  "Codigo", "Apellidos", "Nombre", "Ciudad", "Provincia", "CP")
Console.WriteLine(New String("-", 79))
For Each dr As DataRow In ds.Tables(0).Select("Ciudad LIKE 'M%'", "Apellidos")
    Console.WriteLine("{0,-6} {1,-20} {2,-10} {3,-20} {4,-11} {5,-5}", _
                    dr.Item("IdCliente"), dr.Item("Apellidos"), _
                    dr.Item("Nombre"), dr.Item("Ciudad"), _
                    dr.Item("Provincia"), dr.Item("CP"))
Next
'Buscar con Select un Id.Cliente introducido por teclado
Console.Write("Código cliente: ")
Dim id As String = Console.ReadLine()
Dim filas As DataRow() = ds.Tables("Clientes").Select("IdCliente=" & id)
If filas.Length = 0 Then
    Console.WriteLine("No existe")
Else
    Console.WriteLine("{0,-6} {1,-20} {2,-10} {3,-20} {4,-11} {5,-5}", _
                    filas(0).Item("IdCliente"), filas(0).Item("Apellidos"), _
                    filas(0).Item("Nombre"), filas(0).Item("Ciudad"), _
                    filas(0).Item("Provincia"), filas(0).Item("CP"))
End If
```

# Filtrar y ordenar registros en un DataTable (IV)

```
'Buscar con Select los clientes de una provincia introducida por teclado
Console.Write("Provincia: ")
Dim prov As String = Console.ReadLine()
For Each dr As DataRow In ds.Tables(0).Select("Provincia =" & prov & "'")
    Console.WriteLine("{0,-6} {1,-20} {2,-10} {3,-20} {4,-11} {5,-5}", _
        dr.Item("IdCliente"), dr.Item("Apellidos"), _
        dr.Item("Nombre"), dr.Item("Ciudad"), _
        dr.Item("Provincia"), dr.Item("CP"))
Next
'Buscar con Select los pedidos mayores que un precio determinado
Console.Write("Precio: ")
Dim precio As Double = Console.ReadLine()
For Each dr As DataRow In ds.Tables("Pedidos").Select("Precio >" & precio)
    Console.WriteLine("{0,-6} {1,-6} {2,-20} {3,20}", _
        dr.Item("IdPedido"), dr.Item("IdCliente"), _
        dr.Item("Fecha"), dr.Item("Precio"))
Next
'Buscar con Select los pedidos de una fecha determinada
Console.Write("Fecha (mm/dd/aaaa): ")
Dim fecha As String = Console.ReadLine()
For Each dr As DataRow In ds.Tables("Pedidos").Select("Fecha =" & fecha & "#")
    Console.WriteLine("{0,-6} {1,-6} {2,-20} {3,20}", _
        dr.Item("IdPedido"), dr.Item("IdCliente"), _
        dr.Item("Fecha"), dr.Item("Precio"))
Next
```

# La clase DataView

- ❑ Capa intermedia que se sitúa entre la tabla.
  - Proporciona una vista filtrada y ordenada de las filas de la tabla.
  - Los datos de la tabla no varía, sólo cambia el filtro y la ordenación de las filas.
  - Por su versatilidad se suele utilizar en aplicaciones que necesiten un enlace a datos.
- ❑ Permite crear distintas vistas de una DataTable.
  - Por ejemplo...
    - ✓ Puede servir para mostrar los datos de una tabla con distintos criterios de ordenación.
    - ✓ Pueden enlazarse a controles distintos de una aplicación para mostrar en una rejilla todas las filas y en otra las filas eliminadas.
- ❑ Presenta una vista dinámica de la tabla a la que está asociado.
  - Las modificaciones que se efectúen en el DataView pueden tener efecto en la tabla.
- ❑ Es similar al concepto de vista de una base de datos, pero...
  - No puede excluir ni añadir columnas de la tabla a la que está asociado
  - No puede proporcionar vistas de tablas combinadas.





# La clase DataView (II)

## ❑ Construir un objeto DataView.

- A partir de la propiedad `DefaultView` de la clase `DataTable`.

```
Dim dv As DataView = ds.Tables("Clientes").DefaultView
```

- ✓ Proporciona una vista de todas las filas de la tabla `Clientes` a con su ordenación original.

- Mediante el constructor de la clase `DataView`.

```
DataView(dataTable, expresiónFiltro, expresiónSort,  
dataViewRowState)
```

- ✓ *expresiónFiltro* es una cadena con el criterio de selección en el mismo formato que en el método `Select` de la clase `DataTable`.
- ✓ *expresiónSort* es una cadena con la expresión de ordenación en el mismo formato que el método `Select` de la clase `DataTable`.
- ✓ *dataViewRowState* es un miembro de la enumeración `DataViewRowState` que indica el estado de las filas que se seleccionarán (*ver diapositiva siguiente*).

# La clase DataView (III)

## ❑ Filtrar y ordenar filas.

- La propiedad `RowFilter` de la clase `DataView` permite filtrar los registros de la vista.
  - ✓ El valor de la propiedad sería una cadena con la expresión de filtro (igual que el método `Select` de la clase `DataTable`).
- La propiedad `Sort` permite ordenar las filas de la vista.
  - ✓ El valor de la propiedad sería una cadena con la expresión de ordenación (igual que el método `Select` de la clase `DataTable`).
- La propiedad `RowStateFilter` es un miembro de la enumeración `DataViewRowState` que establece el filtro sobre el estado de la fila.

Nombre de miembro	Descripción
<b>Added</b>	Fila nueva.
<b>CurrentRows</b>	Filas actuales, incluidas las filas sin modificar, las nuevas y las modificadas.
<b>Deleted</b>	Fila eliminada.
<b>ModifiedCurrent</b>	Versión actual, que es una versión modificada de los datos originales (vea <b>ModifiedOriginal</b> ).
<b>ModifiedOriginal</b>	Versión original (aunque se haya modificado y esté disponible como <b>ModifiedCurrent</b> ).
<b>None</b>	Ninguno.
<b>OriginalRows</b>	Filas originales, incluidas las filas sin modificar y las eliminadas.
<b>Unchanged</b>	Fila sin modificar.

# La clase DataView (IV)

## ❑ Acceso a las filas de la vista.

- El número de filas resultante se obtiene con la propiedad `Count`.
- El objeto `DataView` proporciona una colección objetos `DataRow` accesible mediante la propiedad `Item`.
  - ✓ Podemos acceder a ellas a partir del índice de cada elemento de la propiedad `Item`.

`dv.Item(0).Item("IdCliente")`

- Proporciona el identificador de cliente de la primera fila (fila 0) de la vista de datos.

```
'Filtrar los registros de la provincia de Madrid ordenados por Ciudad con un DataView
Dim dv As DataView = ds.Tables("Clientes").DefaultView
dv.RowFilter = "Provincia='Madrid'"
dv.Sort = "Ciudad"
For i As Integer = 0 To dv.Count - 1
    System.Console.WriteLine("{0,-6} {1,-20} {2,-10} {3,-20} {4,-11} {5,-5}", _
        dv.Item(i).Item("IdCliente"), _
        dv.Item(i).Item("Apellidos"), _
        dv.Item(i).Item("Nombre"), _
        dv.Item(i).Item("Ciudad"), _
        dv.Item(i).Item("Provincia"), dv.Item(i).Item("CP"))
Next
```

# Enlazar datos a un formulario

- ❑ Los controles de un formulario se pueden enlazar a casi cualquier colección de datos.
- ❑ El enlace a datos permite vincular un control con un origen de datos (array, objetos de base de datos, etc).
- ❑ Existen dos tipos de enlaces:
  - Enlaces de datos sencillos.
    - ✓ Enlazan una propiedad de un control (por ejemplo la propiedad `Text` de un `TextBox` o un `Label`, o la propiedad `Checked` de un `RadioButton`) a un elemento de un origen de datos.
  - Enlaces de datos complejos.
    - ✓ Enlazan al control un conjunto de datos del origen de datos (por ejemplo en los `ListBox`).
- ❑ Respecto a ADO.NET es posible enlazar los siguiente orígenes de datos.
  - `DataColumn`.
  - `DataTable`.
  - `DataRowView`.
  - `DataSet`.

# El control DataGridView

- ❑ Proporciona una interfaz para la visualización y edición de datos basada en el estilo de hoja de cálculo.
- ❑ Permite el enlace con distintos orígenes de datos:
  - Cualquier clase que implemente la interfaz `IList`, como los arrays de una dimensión.
  - Cualquier clase que implemente la interfaz `IListSource`, como las clases `DataTable` y `DataSet`.
  - Cualquier clase que implemente la interfaz `IBindingList` como la clase `DataGridView`.
  - Cualquier clase que implemente la interfaz `IBindingListView` como la clase `BindingSource`.
- ❑ El enlace del control `DataGridView` con un origen de datos se realiza mediante las propiedades `DataSource` y `DataMember`.
  - La propiedad `DataSource` indica el origen de datos y lo asigna a algún objeto de las clase anteriores.
  - Si el origen de datos contiene varias tablas o listas, será necesario indicar a cual de ellas se va a enlazar mediante la propiedad `DataMember`.

# El control DataGridView (II)

```
Private Sub frmGestiónClientes_Load(ByVal sender As System.Object, _  
                                     ByVal e As System.EventArgs) Handles MyBase.Load  
  
    ConfigurarAccesoADatos()  
    'Enlazar DataGridView a la tabla Clientes  
    DataGridView1.DataSource = ds  
    DataGridView1.DataMember = "Clientes"  
  
    'Si queremos que algunas columnas no se vean se puede utilizar:  
    'DataGridView1.Columns("NombreColumna").Visible = False  
    '  
    'También se puede cambiar la cabecera de una columna con:  
    'DataGridView1.Columns("NombreColumna").HeaderText = "NuevoNombreColumna"  
    DataGridView1.Columns("IdCliente").HeaderText = "ID.Cliente"  
    DataGridView1.Columns("CP").HeaderText = "Código Postal"  
    'Para ajustar el ancho de las columnas  
    DataGridView1.AutoSizeColumnsMode()  
  
End Sub
```



The screenshot shows a Windows application window titled "Gestión de clientes". Inside the window, there is a DataGridView control displaying a table with the following data:

ID.Cliente	Nombre	Apellidos	Ciudad	Provincia	Código Postal
10101	Juan	Esteban Monterías	Alicante	Alicante	03005
10298	Ana	Jiménez de la Calle	Madrid	Madrid	28034
10299	Benito	Noriega Pérez	Alicante	Alicante	03008
10315	María	Ramírez Salle	Carchelejo	Jaen	23192
10325	Miriam	Ruiz Bermudez	Piedrahita	Avila	05500
10329	Luis	Sanjosé de María	Salamanca	Salamanca	37008
10330	Juan José	Pons Gómez	Guadalajara	Guadalajara	19004
10338	María José	Herman Villa	Medina del Campo	Burgos	47400
10339	Dolores	Fuertes Abril	Mataró	Barcelona	08302

At the bottom right of the window, there is a button labeled "Salir".

# El control DataGridView (III)

```
Private Sub ConfigurarAccesoADatos()  
    'Abre la conexión, establece el adaptador de datos y rellena el Dataset  
    'Las variables cn, daClientes, daPedidos, daProductos, ds  
    'y nombreBBDD ya están declaradas  
  
    'Abrir la conexión  
    cn.ConnectionString = "PROVIDER=Microsoft.jet.oledb.4.0; " & _  
        "Data Source=" & nombreBBDD  
  
    cn.Open()  
  
    'Configurar los adaptadores de datos  
    daClientes = New OleDbDataAdapter("SELECT * FROM Clientes", cn)  
    daPedidos = New OleDbDataAdapter("SELECT * FROM Pedidos", cn)  
    daProductos = New OleDbDataAdapter("SELECT * FROM Productos", cn)  
  
    'Cargar el dataset  
    daClientes.Fill(ds, "Clientes")  
    daPedidos.Fill(ds, "Pedidos")  
    daProductos.Fill(ds, "Productos")  
  
    'Cerrar la conexión  
    cn.Close()  
End Sub
```

# El control DataGridView (IV)

## ❑ Acceso al contenido de las celdas.

- La propiedad `CurrentCell` hace referencia a la celda activa.
  - ✓ Devuelve un objeto de tipo `DataViewGridCell`.
    - Propiedad `Value`, devuelve el valor una celda.
      - `DataGridView1.CurrentCell.Value`, devuelve el valor de la celda activa.
  - ✓ Propiedades `RowIndex` y `ColumnIndex` de la celda activa devuelve la fila y la columna de una celda.
- Para establecer la celda activa...
  - ✓ `DataGridView1.CurrentCell = DataGridView1(columna, fila)`.
- Ejemplo: recorrer todas las filas de un `DataGridView`...

```
'Recorre todas las celdas de un DataGridView
'Recorre todas las filas
For i As Integer = 0 To DataGridView1.Rows.Count - 1
    'Recorre todas las columnas de la fila
    For j As Integer = 0 To DataGridView1.Columns.Count - 1
        'Accede a cada una de las celdas j,i
        Debug.Write(DataGridView1(j, i).Value & " ")
    Next
    Debug.WriteLine("")
Next
```



# El control DataGridView (V)

- ❑ Acceso a los datos de las filas relacionadas.
  - La propiedad `RowIndex` devuelve el índice de la fila activa, y por ella será posible acceder al objeto `DataRow` del origen de datos.
  - Ejemplo: al hacer doble click en una celda, aparecen los pedidos del cliente.

The image shows two overlapping windows from a Windows application. The background window, titled "Gestión de clientes", contains a DataGridView table with the following data:

	ID_Cliente	Nombre	Apellidos	Ciudad	Provincia	Código Postal
▶	10101	Juan	Esteban Monterías	Alicante	Alicante	03005
	10298	Ana	Jiménez de la Calle	Madrid	Madrid	28004
	10299	Benito	Noriega Pérez	Alicante	Alicante	03005
	10315	María	Ramírez Salle	Carchelejo	Carchelejo	03005
	10325	Miriam	Ruíz Bermudez	Piedrahita	Piedrahita	03005
	10329	Luis	Sanjosé de María	Salamanca	Salamanca	03005
	10330	Juan José	Pons Gómez	Guadalajara	Guadalajara	03005
	10338	María José	Herman Villa	Medina del Campo	Medina del Campo	03005

The foreground window, titled "Pedidos del cliente", displays the details for the selected client (ID 10101, Juan Esteban Monterías). It includes input fields for ID, Apellidos, Nombre, Ciudad, Provincia, and Código postal. Below these fields is a DataGridView table showing the client's orders:

	IdPedido	IdCliente	Fecha	Producto	Cantidad	Precio	Pagado
▶	4	10101	30/06/2002	3	1	58	<input type="checkbox"/>
	3	10101	01/07/2002	19	4	125	<input type="checkbox"/>
	5	10101	18/08/2002	5	1	18,3	<input checked="" type="checkbox"/>
	1	10101	30/12/2002	7	3	14,75	<input checked="" type="checkbox"/>
	2	10101	02/01/2003	8	1	16	<input type="checkbox"/>
	6	10101	08/03/2003	18	2	88,7	<input checked="" type="checkbox"/>

At the bottom of the "Pedidos del cliente" window, there is a label "Total pedidos:" followed by a text box containing "813,95 €" and a "Cerrar" button.

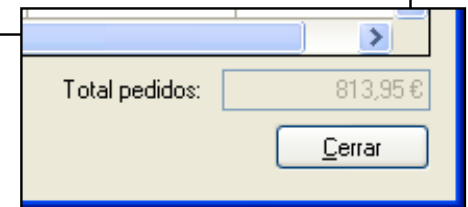
# El control DataGridView (VI)

```
'Este evento se produce al hacer doble Click sobre una celda de DataGridView
Private Sub DataGridView1_CellContentDoubleClick(ByVal sender As Object, _
        ByVal e As System.Windows.Forms.DataGridViewCellEventArgs) _
        Handles DataGridView1.CellContentDoubleClick
    'Rellenar los cuadros de texto del formulario frmDetallesCliente
    'con la información del cliente de la celda seleccionada
    'La orden Width, permite abreviar la referencia al objeto
    With My.Forms.frmDetallesCliente
        .txtIdCliente.Text = ds.Tables("Clientes").Rows(e.RowIndex).Item("IdCliente")
        .txtApellidos.Text = _
            ds.Tables("Clientes").Rows(e.RowIndex).Item("Apellidos")
        .txtNombre.Text = ds.Tables("Clientes").Rows(e.RowIndex).Item("Nombre")
        .txtPoblación.Text = _
            ds.Tables("Clientes").Rows(e.RowIndex).Item("Ciudad")
        .txtProvincia.Text = ds.Tables("Clientes").Rows(e.RowIndex).Item("Provincia")
        .txtCP.Text = ds.Tables("Clientes").Rows(e.RowIndex).Item("CP")
    End With
    'Establecer los pedidos del cliente a partir de una vista de la tabla Pedidos
    Dim dv As DataView = ds.Tables("Pedidos").DefaultView
    dv.Sort = "Fecha"
    dv.RowFilter = "IdCliente =" & ds.Tables("Clientes").Rows(e.RowIndex).Item("IdCliente")
    My.Forms.frmDetallesCliente.DataGridView1.AutoResizeColumns()
    My.Forms.frmDetallesCliente.DataGridView1.DataSource = dv
    My.Forms.frmDetallesCliente.DataGridView1.AutoSizeColumnsMode = _
        DataGridViewAutoSizeColumnsMode.AllCells
    'Mostrar el formulario secundario
    My.Forms.frmDetallesCliente.Show()
End Sub
```

# El control DataGridView (VII)

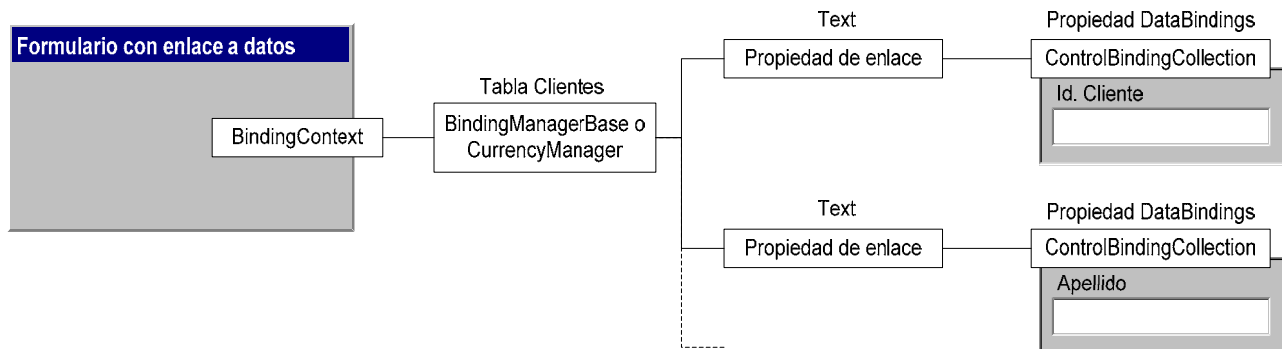
- ❑ Cada fila del DataGridView es un objeto de la clase DataGridViewRow.
  - La propiedad Rows, permite acceder a la colección de objetos DataGridViewRow del control.
  - Ejemplo: cargar el total del importe de los pedidos en el cuadro de texto txtTotalPedidos.

```
Private Sub frmDetallesCliente_Load(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles MyBase.Load  
    Dim totalPedidos As Decimal = 0  
    For Each dr As DataGridViewRow In DataGridView1.Rows  
        totalPedidos +=  
            dr.Cells("Precio").Value * dr.Cells("Cantidad").Value  
    Next  
    txtTotalPedidos.Text = String.Format("{0:C}", totalPedidos)  
End Sub
```



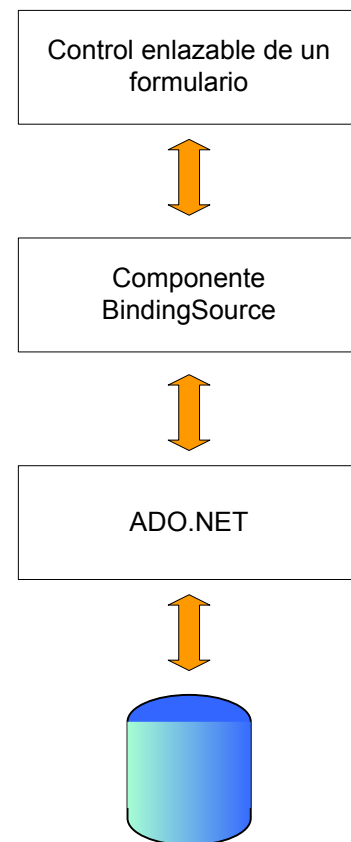
# Enlace de datos simple

- ❑ Permite vincular una propiedad de un control con algún elemento de un origen de datos.
- ❑ Antes de la versión 2.0 de .NET Framework...



# Enlace de datos simple (II)

- ❑ La versión .NET Framework 2.0 incluye un nuevo objeto de enlace a datos: `BindingSource`.
  - Proporciona una capa de *direccionamiento indirecto*.
    - ✓ Enlaza a su origen de datos y a los controles del formulario.
    - ✓ A partir de aquí, toda interacción con los datos (navegación, ordenación, filtrado, actualización) se realiza con el componente `BindingSource`.
  - Encapsula el anterior componente `CurrencyManager`.
  - También permite manejar cualquier colección de objetos no relacionada con el acceso a datos.
  - Permite actuar como origen de datos en controles como `DataGridView` o `BindingNavigator`.



# Enlace de datos simple (III)

## ❑ Creación de un objeto `BindingSource`.

- Declaración.

- ✓ La declaración se debe realizar con la clausula `WithEvents` para poder tener acceso a sus eventos.

- La clausula `WithEvents` está disponible para todas las clases que implementen eventos.

```
Dim WithEvents BindingSource1 as New BindingSource
```

- Para establecer el origen de datos del objeto se utiliza la propiedad `DataSource`.

- ✓ Puede tomar el valor de un objeto de la clase `DataSet`, `DataTable` o `DataGridView`.

- ✓ Si el origen de datos presenta distintas listas o tablas también es necesario establecer el valor de la propiedad `DataMember` a la lista o tabla que queramos enlazar.

- Se supone creado el `DataSet` `ds` con las tablas `Cientes` y `Pedidos`.

```
BindingSource1.DataSource = ds
```

```
BindingSource1.DataMember = "Cientes"
```

# Enlace de datos simple (IV)

## □ Enlazar un control a un origen de datos.

- La propiedad `DataBindings` de la clase `Control`, hace referencia a la colección `ControlDataBindingsCollection` que poseen todos los controles.
  - ✓ Esta colección contiene todos los objetos `Binding`.
    - Cada objeto `Binding` permite mantener un enlace simple entre una propiedad del control y un miembro del origen de datos.
- Para construir una instancia de un objeto `Binding`, el constructor necesita tres elementos:
  - ✓ Nombre de la propiedad del control que se va a enlazar.
  - ✓ El origen de datos.
    - Puede ser un `BindingSource`, un `DataSet`, un `DataTable`, un `DataGridView`, etc.
  - ✓ Nombre del miembro (nombre de la columna) del origen de datos con el que se enlazaré la propiedad.
  - ✓ Opcionalmente se puede poner un valor lógico que indica si se habilita el formato de datos de la propiedad del control y si se habilita el control de errores en el enlace.
    - Es recomendable ponerla a `True`, sobre todo en el caso de enlazar propiedades distintas a `Text`.
- La nueva instancia del objeto `Binding` habrá que incluirla dentro de la colección de objetos `Binding` del control mediante el método `Add`.

```
TextBox1.DataBindings.Add(New Binding("Text", BindingSource1,"IdCliente", True))
```

# Enlace de datos simple (V)

## ❑ Ejemplos de enlace simple a distintos controles.

- Los cuadros de texto pueden enlazar con la propiedad `Text`.

```
TextBox1.DataBindings.Add(New Binding("Text", BindingSource1,"IdCliente", True))
```

- Las casillas de verificación pueden enlazar con la propiedad `Checked`.

```
CheckBox1.DataBindings.Add(New Binding("Checked", BindingSource1,"Pagado", True))
```

- Los controles `NumericUpDown` pueden enlazar con la propiedad `Value`.

```
NumericUpDown1.DataBindings.Add(New Binding("Value", BindingSource1,"Cantidad", True))
```

- En general se puede enlazar cualquier propiedad de un control con cualquier campo de la base de datos.

- ✓ Enlaza el la propiedad `Width` de un `TextBox` con el campo `CP`.

```
TextBox1.DataBindings.Add(New Binding("Width", bs, "CP", True))
```

## ❑ Si el origen de datos tiene más de una tabla, se puede especificar a cuál de ellas corresponde el campo a enlazar.

- Si el `Dataset` asociado al objeto `BindingSource` contiene las tablas `Clientes` y `Pedidos`.

- `Pedidos.IdCliente`, enlaza con la columna `IdPedido` de la tabla `Pedidos`.

- `Pedidos.IdPedidos`, enlaza con la columna `IdPedido` de la tabla `Pedidos`.

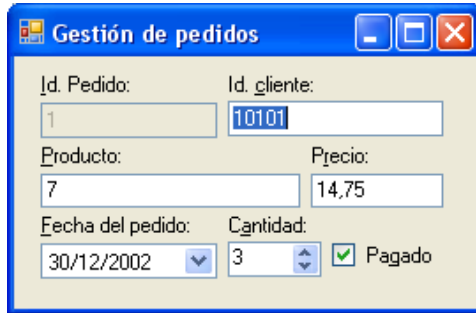
```
TextBox1.DataBindings.Add(New Binding("Text", BindingSource1,"Clientes.IdCliente", True))
```

```
TextBox2.DataBindings.Add(New Binding("Text", BindingSource1,"Pedidos.IdPedidos", True))
```



# Enlace de datos simple (VI)

- ❑ Ejemplo: enlaza los datos de la tabla Pedidos con el formulario.



```
Private cn As New OleDbConnection
Private daPedidos As OleDbDataAdapter
Private ds As New DataSet
Private nombreBBDD As String = Application.StartupPath & "\Ejemplo.mdb"
Private WithEvents bs As New BindingSource

Private Sub frmPedidos_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
    ConectarADatos()
    'El cuadro de texto IdPedido no está accesible, ya que es el
    'gestor de BBDD el que lo asigna
    txtIdPedido.Enabled = False

    EnlazarADatos()
End Sub

Private Sub ConectarADatos()
    cn.ConnectionString = "PROVIDER=Microsoft.Jet.OleDb.4.0;" & _
        "Data source=" & nombreBBDD
    daPedidos = New OleDbDataAdapter("SELECT * FROM Pedidos", cn)
    cn.Open()
    'Rellenar los DataSet y cerrar la conexión
    daPedidos.Fill(ds, "Pedidos")
    cn.Close()
End Sub
```

# Enlace de datos simple (VII)

```
Private Sub EnlazarADatos()  
    'Configurar el BindingSource  
    bs.DataSource = ds  
    bs.DataMember = "Pedidos"  
  
    'Enlazar cuadros de texto  
    txtIdPedido.DataBindings.Add(New Binding("Text", bs, "IdPedido", True))  
    txtIdCliente.DataBindings.Add(New Binding("Text", bs, "IdCliente", True))  
    txtProducto.DataBindings.Add(New Binding("Text", bs, "Producto", True))  
    txtPrecio.DataBindings.Add(New Binding("Text", bs, "Precio", True))  
  
    'Enlazar NumericUpDown  
    nudCantidad.DataBindings.Add(New Binding("Value", bs, "Cantidad", True))  
  
    'Enlazar DateTimePicker  
    dtpFecha.DataBindings.Add(New Binding("Value", bs, "Fecha", True))  
  
    'Enlazar el CheckBox  
    chkPagado.DataBindings.Add(New Binding("Checked", bs, "Pagado", True))  
End Sub
```

# El objeto `BindingSource`

- ❑ Obtener la lista a la que está enlazado el objeto.
  - La propiedad `List` del objeto `BindingSource` permite acceder a la lista a la que se ha asociado el objeto (lista subyacente).
    - ✓ Cuando el origen de datos es un `DataSet`, una `DataTable` o una `DataView` se trata de un objeto de la clase `DataView`.
      - A través de ella se podrá acceder algunos de los miembros de la clase `DataView`.
- ❑ Obtener el elemento actual del objeto `BindingSource`.
  - La propiedad `Current` del objeto `BindingSource` obtiene una referencia al elemento actual.
    - ✓ Cuando el origen de datos es un `DataSet`, una `DataTable` o una `DataView` la propiedad `Current` devuelve un objeto de la clase `DataRowView`.
      - Por ejemplo, será posible acceder a los campos mediante la propiedad `Item`:

```
bs.Current.Item("Cantidad");
```
  - La propiedad `Position` devuelve la posición del elemento actual dentro del objeto `BindingSource`.
- ❑ La propiedad `Count` devuelve el número de elementos que se han incluido en el `BindingSource`.

# El objeto `BindingSource` (II)

## ❑ Cambiar el elemento actual.

- La propiedad `Position` permite también modificar el elemento actual de la lista subyacente.

```
bs.Position = 10 'Mueve el objeto actual a la posición 11.
```

- Los métodos `MoveNext`, `MoveLast`, `MoveFirst` y `MoveLast` permiten mover el elemento actual a la siguiente, anterior, primera o última posición.
- Cada vez que cambia la posición del elemento actual se produce el evento `PositionChanged`.
  - ✓ Será posible actualizar una etiqueta de navegación con la posición actual.
  - ✓ Será posible actualizar un campo calculado.

# El objeto BindingSource (III)

```
Private Sub btnSiguiente_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnSiguiente.Click
    If bs.Position < bs.Count - 1 Then
        bs.MoveNext()
    Else
        MessageBox.Show("Se ha llegado al último pedido", Me.Text, _
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
    End If
End Sub
Private Sub btnPrimero_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnPrimero.Click
    bs.MoveFirst()
End Sub
'Aquí faltarían los eventos btnAnterior.Click y btnÚltimo.Click
Private Sub bs_PositionChanged(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles bs.PositionChanged
    'Actualizar la posición del registro actual
    lblPosición.Text = "Pedido " & bs.Position + 1 & " de " & bs.Count
    'Actualizar el total del pedido
    'txtPrecio se ha inicializado a 0 en tiempo de diseño
    txtTotal.Text = CInt(txtPrecio.Text) * CInt(nudCantidad.Value)
End Sub
```

Id. Pedido:	Id. cliente:		
5	10101		
Producto:	Fecha del pedido:		
5	18/08/2002		
Cantidad:	Precio:	Total:	
1	18,3	18	
			<input checked="" type="checkbox"/> Pagado

Pedido 5 de 20

# Buscar registros

- ❑ El método `Select` realiza una búsqueda secuencial por cualquier campo.
- ❑ El método `Find` de la clase `DataRowCollection` (la colección `Rows` es un dato de esa clase), permite localizar un único registro a partir de una clave principal.
  - Realiza una búsqueda por clave únicas más eficiente que `Select`.
  - Precisa definir una clave primaria.
  - Devuelve un único objeto de tipo `DataRow` o un valor nulo si no se encuentra.
- ❑ Para claves simples (de una sola columna).
  - `Find(clave)`.
    - ✓ `clave` es una expresión del mismo tipo que la clave primaria.
- ❑ Para claves compuestas (de más de una columna).
  - `Find(claves)`.
    - ✓ `claves` es un array que contiene cada una de las partes de que forman la clave primaria.

# Buscar registros (II)

- ❑ La propiedad `PrimaryKey` de la clase `DataTable` permite establecer una clave primaria.
- ❑ Recibe como valor un **array de objetos** `DataColumn` con cada una de las partes de la clave.
- ❑ Las columnas que se utilizan como clave no deben tener valores repetidos.
- ❑ Para establecer el campo `IdCliente` como clave primaria.

```
Dim clave(0) As DataColumn
Clave(0) = New DataColumn
Clave(0) = ds.Tables("Clientes").Columns("IdCliente")
ds.Tables("Clientes").PrimaryKey = clave
```

- ❑ Para establecer los campos `Apellidos` y `Nombre` como clave primaria.

```
Dim claveApeNom(1) As DataColumn
claveApeNom(0) = New DataColumn
claveApeNom(0) = ds.Tables("Clientes").Columns("Apellidos")
claveApeNom(1) = New DataColumn
claveApeNom(1) = ds.Tables("Clientes").Columns("Nombre")
ds.Tables("Clientes").PrimaryKey = claveApeNom
```

# Buscar registros (III)

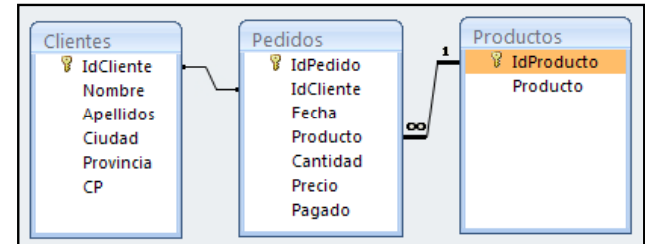
- ❑ Buscar en la tabla Pedidos por IdCliente y Fecha (se supone que no hay valores repetidos).

```
'Establecer la clave primaria IdCliente+Fecha
Dim claveIdFecha(1) As DataColumn
claveIdFecha(0) = New DataColumn
claveIdFecha(0) = ds.Tables("Pedidos").Columns("IdCliente")
claveIdFecha(1) = New DataColumn
claveIdFecha(1) = ds.Tables("Pedidos").Columns("Fecha")
ds.Tables("Pedidos").PrimaryKey = claveIdFecha
'Leer el Id.Cliente y la Fecha
Dim valores(1) As Object
Console.Write("Id. Cliente:")
valores(0) = Console.ReadLine()
Console.Write("Fecha pedido:")
valores(1) = CDate(Console.ReadLine())
Dim filaEncontrada As DataRow = ds.Tables("Pedidos").Rows.Find(valores)
If filaEncontrada Is Nothing Then
    Console.WriteLine("No está")
Else
    Console.WriteLine("{0,9} {1,10} {2,-18} {3,-20} {4,-5} {5,6}", _
        filaEncontrada.Item("IdPedido"), filaEncontrada.Item("IdCliente"), _
        filaEncontrada.Item("Fecha"), filaEncontrada.Item("Producto"), _
        filaEncontrada.Item("Cantidad"), filaEncontrada.Item("Precio"))
End If
```



# Relaciones

- ❑ Permiten acceder a los datos de una tabla secundaria a partir de un campo clave de una tabla primaria.
  - También permiten restringir la manipulación de datos y exigir la *integridad referencial* de los datos.



- ❑ La clase `DataRelation` permite añadir relaciones entre las tablas de un `DataSet`.
  - Las relaciones de un `DataSet` son accesibles a partir de su colección `Relations`.
- ❑ Creación de una relación.
  - Crear una nueva instancia de la clase `DataRelation`.  
`New DataRelation(nombreRelación, columnaPadre, columnaHija)`
    - ✓ `columnaPadre` y `columnaHija` son dos objetos `DataColumn` de las tablas primaria y secundaria.
  - Añadirla a la colección `Relations` del `DataSet`.  
`objetoDataSet.Relations.Add(nombreRelación)`

# Relaciones (II)

- ❑ Crear una relación entre la columna `IdCliente` de la tabla `Cientes` y la columna `IdCliente` de la tabla `Pedidos`.

```
Dim rel As DataRelation = New DataRelation("ClientePedidos", _  
    ds.Tables("Clientes").Columns("IdCliente"), _  
    ds.Tables("Pedidos").Columns("IdCliente"))  
ds.Relations.Add(rel)
```

- ❑ Crear una relación entre la columna `IdProducto` de la tabla `Productos` y la columna `Producto` de la tabla `Pedidos`

```
rel = New DataRelation("ProductoPedidos", _  
    ds.Tables("Productos").Columns("IdProducto"), _  
    ds.Tables("Pedidos").Columns("Producto"))  
ds.Relations.Add(rel)
```

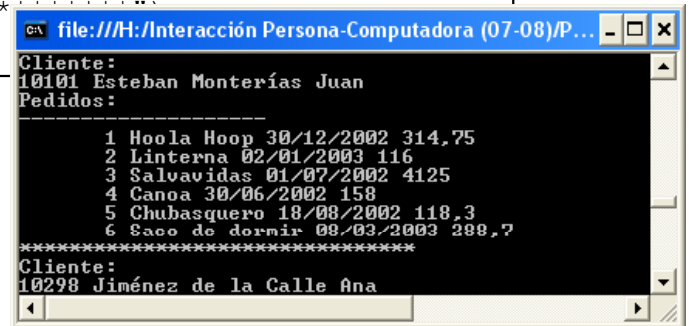
# Relaciones (III)

## ❑ Acceso a los registros relacionados.

- La propiedad `GetChildRows` de la clase `DataRow` permite acceder a las filas relacionadas de la tabla secundaria a partir del valor del campo clave de la fila.
  - ✓ Devuelve un array de elementos de tipo `DataRow`.
- La propiedad `GetParentRow` de la clase `DataRow` permite acceder al registro maestro de la tabla primaria a partir del valor del campo relacionado de la fila.
  - ✓ Devuelve un dato de tipo `DataRow`  
*`dataRow.GetParentRow(relación)`*

# Relaciones (IV)

```
'Listado de todos los pedidos de todos los clientes
For Each dr As DataRow In ds.Tables("Clientes").Rows
    System.Console.WriteLine("Cliente:")
    System.Console.WriteLine(dr.Item("IdCliente") & " " & _
        dr.Item("Apellidos") & " " & _
        dr.Item("Nombre") & " ")
    System.Console.WriteLine("Pedidos:")
    System.Console.WriteLine("-----")
    'Sacar los pedidos del cliente dr
    For Each drChild As DataRow In dr.GetChildRows("ClientePedidos")
        System.Console.Write("        ")
        System.Console.WriteLine(drChild.Item("IdPedido") & " " & _
            drChild.GetParentRow("ProductoPedidos").Item("Producto") & " " & _
            drChild.Item("Fecha") & " " & _
            drChild.Item("Cantidad") & " " & _
            drChild.Item("Precio"))
    Next
    System.Console.WriteLine("*****")
Next
```

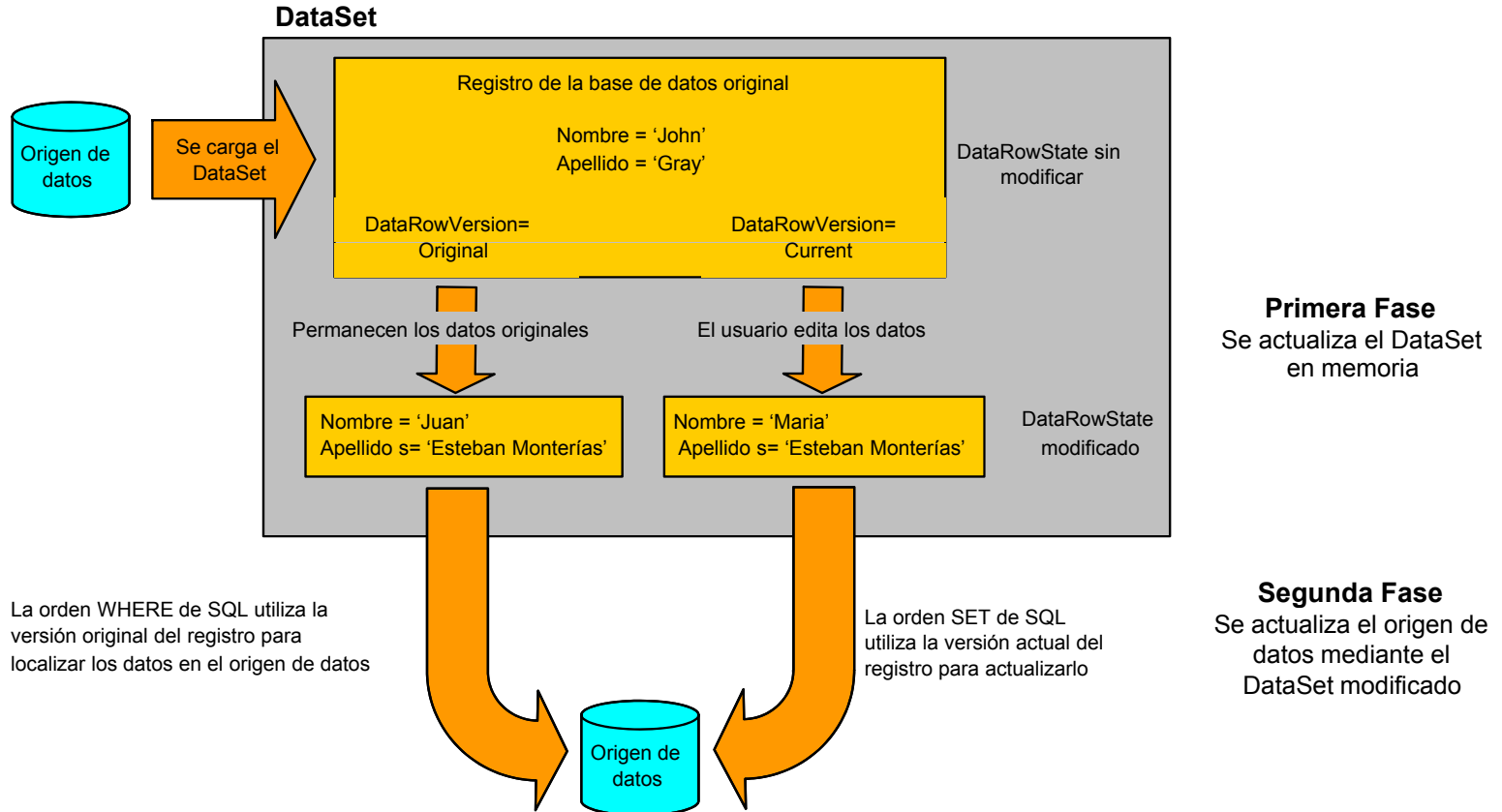


```
file:///H:/Interacción Persona-Computadora (07-08)/P...
Cliente:
10101 Esteban Monterías Juan
Pedidos:
-----
1 Hoola Hoop 30/12/2002 314,75
2 Linterna 02/01/2003 116
3 Salvavidas 01/07/2002 4125
4 Canoa 30/06/2002 158
5 Chubasquero 18/08/2002 118,3
6 Saco de dormir 08/03/2003 288,7
*****
Cliente:
10298 Jiménez de la Calle Ana
```

# Actualización de registros

- ❑ La actualización de datos en un `DataSet` se realiza sólo en memoria.
  - Los datos originales se mantienen en el origen de datos.
- ❑ Una vez cargado el conjunto de datos, la actualización del origen de datos se hace en dos fases:
  - Primero se modifica el conjunto de datos (se modifica el valor de las columnas, se añaden nuevos registros, se eliminan filas).
    - ✓ Si el conjunto de datos se utiliza sólo como almacenamiento temporal para pasar los datos a otras aplicaciones, el proceso terminaría aquí.
  - Después, si se desea modificar el origen de datos, es necesario enviar las modificaciones de forma explícita con el método `Update` del adaptador de datos.
    - ✓ El método `Update` manda instrucciones SQL al origen de datos para cada fila modificada.

# Actualización de registros (II)



# Modificación de un registro

- ❑ Modificar un registro de una tabla de un DataSet, simplemente supone localizar un registro y modificar las columnas necesarias.

```
'Actualizar la ciudad del cliente con IdCliente = 10101
'
'Localizar el registro
'Se supone que la propiedad PrimaryKey contiene la columna IdCliente
'encontrado es un dato de la clase DataRow
encontrado = ds.Tables("Clientes").Rows.Find(10101)
'
'Modificar el valor de la columna Ciudad
encontrado.Item("Ciudad") = "Benidorm"
```

- ❑ Si se ha establecido una relación:
  - Si se modifica el campo de relación de la tabla primaria, se actualizarán también en las filas de la tabla secundaria relacionada.
  - Si se modifica el campo de relación de la tabla secundaria se genera un error.

# Añadir un nuevo registro

- ❑ La inserción de un nuevo registro implica tres pasos.
  1. Crear un nuevo registro vacío con el método `NewRow` de la clase `DataTable`.
    - ✓ `NewRow` crea un nuevo registro vacío con el mismo esquema que la tabla que lo ha llamado.
  2. Introducir nuevos valores para cada una de las columnas de la tabla en las que sea necesario.
    - ✓ En la definición de la tabla en el gestor de bases de datos se puede especificar si cada columna puede tener campos vacíos o no.
    - ✓ En la definición de la tabla se puede definir un campo como autoincremental.
      - El valor de dicho campo se calculará de forma automática.
  3. Añadir el nuevo registro a la colección `Rows` del objeto `DataTable` mediante el método `Add`.
    - ✓ Si existe una clave principal y está repetida se generará una excepción de tipo `System.Data.ConstraintException`.
    - ✓ Si no se proporcionan datos en las columnas en las que sea obligatorio se genera una excepción de tipo `System.Data.NullAllowedException`.



# Añadir un nuevo registro (II)

```
'Añadir un nuevo registro a la tabla clientes
'Se supone que la propiedad PrimaryKey de la tabla tiene la columna IdCliente
'Crear un nuevo registro vacío
Dim nuevo As DataRow = ds.Tables("Clientes").NewRow
'Llenar los datos
System.Console.Write("Id. cliente:")
nuevo.Item("IdCliente") = System.Console.ReadLine()
System.Console.Write("Apellidos:")
nuevo.Item("Apellidos") = System.Console.ReadLine()
System.Console.Write("Nombre:")
nuevo.Item("Nombre") = System.Console.ReadLine()
System.Console.Write("Ciudad:")
nuevo.Item("Ciudad") = System.Console.ReadLine()
System.Console.Write("Provincia:")
nuevo.Item("Provincia") = System.Console.ReadLine()
'Añadir el nuevo registro a la colección Rows de la tabla Clientes
'comprobando si se inserta una fila con la clave duplicada
Try
    ds.Tables("Clientes").Rows.Add(nuevo)
Catch ex As ConstraintException
    System.Console.WriteLine("Error: " & ex.Message)
End Try
```

# Eliminar registros

- ❑ Una vez localizado el registro, sólo es necesario llamar al método `Delete` de la clase `DataRow`.

```
'Eliminar un registro
'Localizar el registro.Se supone que la propiedad PrimaryKey establecida
'encontrado es un dato de la clase DataRow
encontrado = ds.Tables("Clientes").Rows.Find(System.Console.ReadLine())
encontrado.Delete()
```

- ❑ Si el registro ya ha sido eliminado previamente se provocará una excepción de tipo `DeleteRowInaccessibleException`.
- ❑ El método `Delete` realiza una baja lógica.
  - La propiedad `RowState` del registro se marca con el valor `Deleted`.
  - Los registros borrados se incluirán en el número de filas de la colección y aparecerán en los listados.

# Actualizar el origen de datos

- ❑ El método `Update` de los adaptadores de datos transmiten las modificaciones que se han realizado en memoria al origen de datos.
  - Recorre la colección `Rows` de la tabla analizando la propiedad `RowState` para identificar la acción a realizar:
    - ✓ Si contiene el valor `Modified`, llama a la instrucción SQL contenida en la propiedad `UpdateCommand` para enviar la modificación.
    - ✓ Si contiene el valor `Added`, llama a la instrucción SQL contenida en la propiedad `InsertCommand` para enviar la modificación.
    - ✓ Si contiene el valor `Deleted`, llama a la instrucción SQL contenida en la propiedad `DeleteCommand` para enviar la modificación.
- ❑ Si se desea actualizar varias tablas, será necesario llamar al método `Update` de cada adaptador que se ha utilizado para llenar la tabla.
- ❑ Rechazar los cambios pendientes.
  - El método `RejectChanges` de la clase `DataTable` permite rechazar los cambios pendientes, restaurando las filas con su valor original.

# Órdenes de actualización

- ❑ Antes de utilizar el método `Update` es necesario configurar las propiedades `UpdateCommand`, `InsertCommand` y `DeleteCommand` del adaptador de datos.
  - Utilizando un procedimiento almacenado en la base de datos que realice la instrucción SQL.
  - Creando manualmente para cada una un nuevo comando que contenga las instrucciones SQL `UPDATE`, `INSERT` y `DELETE`.
    - ✓ Normalmente será necesario incluir en la orden parámetros que permitan realizar la acción en cada caso concreto.
  - Permitir que ADO.NET genere las órdenes de forma automática.

# Órdenes de actualización (II)

## ❑ Generar órdenes de actualización automáticamente.

### ● Requisitos:

- ✓ Es necesario haber rellenado previamente la tabla mediante el método Fill del adaptador de datos.
- ✓ La orden para rellenar la tabla deberá hacer referencia a toda una tabla de la base de datos (por ejemplo "SELECT \* FROM Clientes").
  - Si se cumplen estos requisitos, el adaptador de datos habrá generado una propiedad `SelectCommand`.
- ✓ La propiedad `SelectCommand` deberá devolver entre los datos que se han recuperado del adaptador de datos una columna que contenga una clave primaria (valor sin duplicados).

### ● Restricciones a la generación automática.

- ✓ **Control de concurrencia.** No se permitirá controlar la actualización simultánea de una fila por varios usuarios.
- ✓ **Tablas relacionadas.** Las órdenes de actualización sólo funcionan si se han recuperado del origen de datos tablas independientes no relacionadas.
- ✓ **Nombres de tabla y columna.** Deberán estar compuestos de caracteres alfanuméricos, no permitiéndose espacios en blanco o caracteres especiales.

# Órdenes de actualización (III)

## ❑ La clase `CommandBuilder`.

- Permite generar las órdenes de actualización de un adaptador de datos de forma automática.

```
'La conexión cn se supone ya creada y abierta
'Crear los adaptadores de datos
daClientes = New OleDbDataAdapter("SELECT * FROM Clientes", cn)
'La propiedad SelectCommand se carga con SELECT * FROM Clientes
'Rellenar el dataset
daClientes.Fill(ds, "Clientes")
'Creación del objeto CommandBuilder del adaptador de datos
Dim cb As OleDbCommandBuilder = New OleDbCommandBuilder(daClientes)
```

## ❑ Actualizar los registros.

- La orden `Update` utiliza como argumentos el `DataSet` que se va a utilizar y la tabla que se actualizará.

```
daClientes.Update(ds, "Clientes")
```

# Ejemplo de actualización

```
'La conexión cn se supone ya creada y abierta
'Crear los adaptadores de datos
daClientes = New OleDbDataAdapter("SELECT * FROM Clientes", cn)
'La propiedad SelectCommand se carga con SELECT * FROM Clientes
'Rellenar el dataset
daClientes.Fill(ds, "Clientes")
'Creación del objeto CommandBuilder del adaptador de datos
Dim cb As OleDbCommandBuilder = New OleDbCommandBuilder(daClientes)
cn.Close()
'Modificar la ciudad del Cliente 10101 (el primer cliente)
ds.Tables("Clientes").Rows(0).Item("Ciudad") = "Benidorm"
'Añadir el cliente 10103
Dim nuevaFila As DataRow = ds.Tables("Clientes").NewRow
nuevaFila.Item("IdCliente") = 10103
nuevaFila.Item("Apellidos") = "Martínez"
nuevaFila.Item("Nombre") = "Juana"
nuevaFila.Item("Ciudad") = "Arganda del Rey"
nuevaFila.Item("Provincia") = "Madrid"
ds.Tables("Clientes").Rows.Add(nuevaFila)
'Borrar el cliente 10102
Dim filaBorrada As DataRow() = ds.Tables("Clientes").Select("IdCliente=10102")
filaBorrada(0).Delete()
'Actualizar el origen de datos
daClientes.Update(ds, "Clientes")
```

# Actualización de registro: Control DataGridView

## ❑ Actualización de los datos del DataGridView.

- Por omisión los datos del control (y por lo tanto los del origen de datos de donde los saca) son actualizables.
  - ✓ Las propiedades `AllowUserToAddRows`, `AllowUserToDeleteRows` y `ReadOnly` permiten o impiden que el usuario añada, elimine o modifique los datos del control.
- Las modificaciones de los datos se harán sólo en memoria.
  - ✓ Para actualizar el origen de datos será necesario llamar al método `Update` de la clase `DataAdapter` que se ha utilizado para obtener el origen de datos después de haber creado las órdenes de actualización.

```
Private Sub btnActualizar_Click(ByVal sender As System.Object, _  
                                ByVal e As System.EventArgs) Handles btnActualizar.Click  
  
    Try  
        daClientes.Update(ds, "Clientes")  
        ds.AcceptChanges()  
    Catch ex As System.Data.OleDb.OleDbException  
        MessageBox.Show(ex.Message, Me.Text, _  
                        MessageBoxButtons.OK, _  
                        MessageBoxIcon.Information)  
        ds.Tables("Clientes").RejectChanges()  
    End Try  
End Sub
```



# Actualización de registros: BindingSource

- ❑ Actualización de los elementos de la lista subyacente.
  - Cualquier cambio en las propiedades de los controles enlazados, supone un cambio en memoria de los datos.
  - El método `EndEdit` de la clase `BindingSource` se aplican todos los cambios pendientes de la lista subyacente.
    - ✓ Realizaría los cambios en el `DataSet`, pero no en la base de datos.
      - Para realizar las transacciones a la base de datos, sería necesario aplicar el método `Update` del adaptador de datos.
- ❑ Cancelar los cambios pendientes.
  - El método `CancelEdit` cancela la edición del elemento actual.
  - Para cancelar todas las actualizaciones pendientes desde la última carga o desde el último `EndEdit` será necesario llamar al método `RejectChanges` de la clase `DataSet` o `DataTable`.
  - Para restaurar a las propiedades de los campos enlazados del elemento a los valores originales del `BindingSource`...
    - ✓ El método `ResetCurrentItem` actualiza el elemento actual.
    - ✓ El método `ResetItem(índice)` actualiza el elemento indicado por el índice.

# Actualización de registros: BindingSource (II)

```
Private Sub btnActualizar_Click(ByVal sender As Object, _  
    ByVal e As System.EventArgs) Handles btnActualizar.Click  
    'Si se han producido cambios en el Dataset,  
    'la colección GetChanges está vacía y  
    'se actualiza el conjunto de datos  
    bs.EndEdit()  
    If Not ds.GetChanges() Is Nothing Then  
        Try  
            daClientes.Update(ds, "Clientes")  
            ds.AcceptChanges()  
        Catch ex As Exception  
            MessageBox.Show(ex.Message, Me.Text, MessageBoxButtons.OK, _  
                MessageBoxIcon.Information)  
            ds.Tables("Clientes").RejectChanges()  
        End Try  
    End If  
End Sub
```

# Actualización de registros: BindingSource (III)

## ❑ Eliminar elementos.

- El método `RemoveCurrent` elimina el elemento actual de la lista.
- El método `RemoveAt` (índice) elimina el elemento en la posición especificada en el índice.
- El método `Remove` (objeto) elimina el objeto indicado en la lista.

```
Private Sub btnBorrar_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnBorrar.Click
    'Si existe una fila actual, la borra
    If Not bs.Current Is Nothing Then
        If MessageBox.Show("¿Desea borrar el cliente actual?", _
            Me.Text, _
            MessageBoxButtons.YesNo, MessageBoxIcon.Question, _
            MessageBoxDefaultButton.Button2) = _
            Windows.Forms.DialogResult.Yes Then
            bs.RemoveCurrent()
        End If
    End If
End Sub
```

# Actualización de registros: BindingSource (IV)

## ❑ Añadir nuevos elementos a la lista.

- El método `AddNew` añade un nuevo elemento **vacío** en la lista subyacente.
  - ✓ Automáticamente al método `EndEdit` para confirmar cualquier operación de edición pendiente.
  - ✓ Ejecuta el evento `AddingNew`.
    - Este evento se puede utilizar para realizar cálculos, comprobaciones o almacenar información en otra tabla de la base de datos.
  - ✓ El nuevo elemento se añade automáticamente a la lista subyacente al final de la lista.
    - Es posible cancelar la operación realizando una llamada al método `CancelEdit` que descartaría el nuevo elemento.
  - ✓ Provoca el evento `ListChanged`.

```
Private Sub btnNuevo_Click(ByVal sender As System.Object, _  
                           ByVal e As System.EventArgs) Handles btnNuevo.Click  
    bs.AddNew()  
End Sub  
Private Sub btnCancelar_Click(ByVal sender As Object, _  
                              ByVal e As System.EventArgs) Handles btnCancelar.Click  
    bs.CancelEdit()  
End Sub
```

# Actualización de registros: BindingSource (V)

## ❑ Actualización del origen de datos.

- Los métodos `Removexxx` y `AddNew` sólo realizan cambios en la lista interna del `BindingSource` y en la lista subyacente (el `DataSet`, la `DataTable` o el `DataGridView`).
  - ✓ Para transferir los cambios a la base de datos sería necesario llamar al método `Update` del adaptador de datos.

## ❑ Filtrar y ordenar registros.

- El método `Find` de la clase `BindingSource` permite seleccionar los elementos del objeto.
  - ✓ Utiliza los mismos argumentos que el método `Find` del método `Select` de la colección `Rows`.
- El método `Sort` permite establecer un criterio de ordenación.

# Enlace de datos complejo

- ❑ Enlaza varios elementos de un control a la misma o a distintas columnas contenidas en un origen de datos.
  - Es el que utilizan los controles `ListBox` o `ComboBox`.
- ❑ Las propiedades relacionadas con el enlace de datos complejo de estos controles son `DataSource`, `DisplayMember`, `ValueMember` y `SelectedValue`.
- ❑ Propiedad `DataSource`.
  - Obtiene o establece el origen de datos del control.
  - Puede ser un objeto `BindingSource`, `DataView`, `DataTable`, `DataSet`, `Array`.
- ❑ Propiedad `DisplayMember`.
  - Una cadena que indica la columna que se visualizará en el control.
  - Deberá incluir alguna de las propiedades (columnas) incluidas dentro del objeto designado como `DataSource`.
    - ✓ Si el origen de datos tiene más de una lista, el valor de esta propiedad deberá indicar también a qué lista pertenece. Por ejemplo:

```
ListBox1.DisplayMember = "Pedidos.Producto"
```

# Enlace de datos complejo (II)

## ❑ Propiedad `ValueMember`.

- Indica la el nombre de la propiedad real que se utilizará para el enlace de datos.
  - ✓ No tiene porqué ser la misma que se visualiza en el control (propiedad `DisplayMember`).
    - Normalmente la propiedad `DisplayMember` hará referencia a una columna que tenga un valor más significativo (por ejemplo el nombre de una asignatura, o el nombre de un alumno), mientras que la propiedad `ValueMember` hará referencia a un campo que funcione como clave (el código de la asignatura o el expediente del alumno).

## ❑ Propiedad `SelectedValue`.

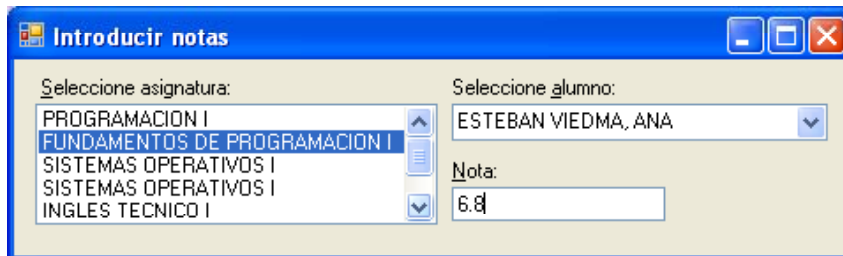
- Indica el valor de la propiedad a la que hace referencia `ValueMember`.

# Enlace de datos complejo (III)

Asignaturas	
Código	Nombre
102	PROGRAMACION I
106	FUNDAMENTOS DE PROGRAMACION I
107	SISTEMAS OPERATIVOS I
107	SISTEMAS OPERATIVOS I
108	INGLES TECNICO I
110	PROGRAMACION II
113	FUNDAMENTOS DE PROGRAMACION II

Alumnos	
Expediente	Nombre
213231	PÉREZ BERMÚDEZ, JESÚS
433242	BENITO BENÍTEZ, JUANA
543453	ESTEBAN VIEDMA, ANA
654564	JIMÉNEZ JULÍN, PEDRO
766455	BIENVENIDO SMITH, MORGAN
353455	CORDOBA SEVILLA, LUISA

Notas		
Código	Expediente	Nota
106	213231	6.6
106	766455	5.5
108	213231	4.0
107	654564	7.9
113	766455	3.2
113	213231	1.5



```
ListBox1.DataSource = Asignaturas  
ListBox1.DisplayMember = "Nombre"  
ListBox1.ValueMember = "Código"
```

```
ComboBox1.DataSource = Alumnos  
ComboBox1.DisplayMember = "Nombre"  
ComboBox1.ValueMember = "Expediente"
```

- ListBox1.SelectedValue será "106", el código de la asignatura seleccionada.
- ComboBox1.SelectedValue será "543453" el expediente del alumno seleccionado.



# Enlace de datos complejo (IV)

- ❑ Para llenar los nombres de los Productos de una tabla Productos en un ComboBox.
  - Se supone la base de datos tiene una tabla Productos, que se ha creado un adaptador de datos para ella y que se ha incluido en el DataSet.

```
'En el procedimiento ConfigurarAcceso a datos se han incluido  
'las siguientes declaraciones  
...  
daProductos = New OleDbDataAdapter("SELECT * FROM Productos", cn)  
daProductos.Fill(ds, "Productos")  
...  
'La propiedad DataSource se enlaza al origen de datos  
cmbProductos.DataSource = ds.tables("Productos")  
'La propiedad DisplayMember se enlaza a la columna que se quiere  
'visualizar en el ComboBox  
cmbProductos.DisplayMember = "Producto"  
...
```

Gestión de pedidos

Id. Pedido: 1 Id. cliente: 10101

Producto: Almohada Fecha del pedido: 30/12/2002

Almohada  
Bicicleta  
Canoa  
Casco  
Chubasquero  
Colchón inflable  
Hoola Hoop  
Linterna

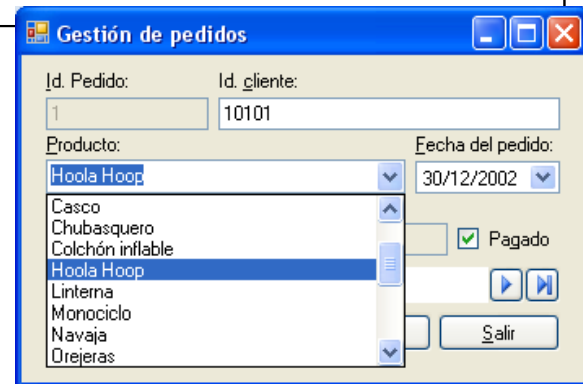
Pagado

Salir

# Enlace de datos complejo (V)

- ❑ En el código anterior no se ha enlazado el elemento seleccionado al objeto `BindingSource`.
  - Para asociar el elemento del `ComboBox` al campo del `BindingSource` hace falta:
    - ✓ Establecer la propiedad `ValueMember` al identificador de producto de la tabla `Productos` (`IdProducto`).
    - ✓ Enlazar la propiedad `SelectedValue` al campo correspondiente del `BindingSource`.

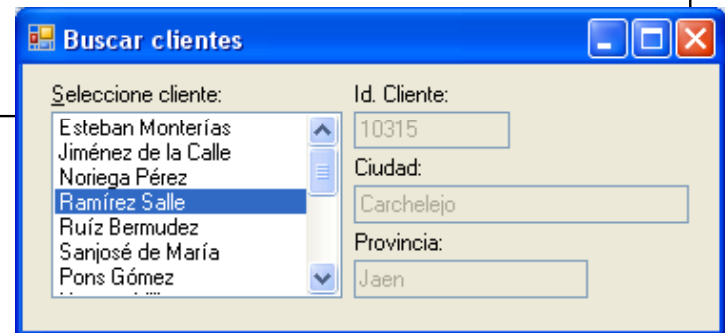
```
...  
'Para indicar cual será el campo de enlace  
cmbClientes.ValueMember = "IdCliente"  
'Para enlazar el IdCliente con la columna IdCliente del BindingSource  
cmbClientes.DataBindings.Add(New Binding("SelectedValue", bs, "IdCliente", True))  
...
```



# Enlace de datos complejo (VI)

- ❑ Si se enlaza la propiedad `DataSource` a un `BindingSource`, es posible acceder al resto de los datos enlazados al mismo seleccionando un nuevo elemento.

```
Private Sub EnlazarADatos()  
    'El origen de datos del BindingSource es la tabla Clientes  
    'Se supone que se ha llamado al procedimiento ConfigurarAccesoADatos  
    bs.DataSource = ds.Tables("Clientes")  
    'Enlazar los controles  
    txtIdCliente.DataBindings.Add(New Binding("Text", bs, "idCliente", True))  
    txtCiudad.DataBindings.Add(New Binding("Text", bs, "Ciudad", True))  
    txtProvincia.DataBindings.Add(New Binding("Text", bs, "Provincia", True))  
    'Enlazar el ComboBox  
    lstClientes.DataSource = bs  
    lstClientes.DisplayMember = "Apellidos"  
End Sub
```



Buscar clientes

Seleccione cliente: Id. Cliente: 10315

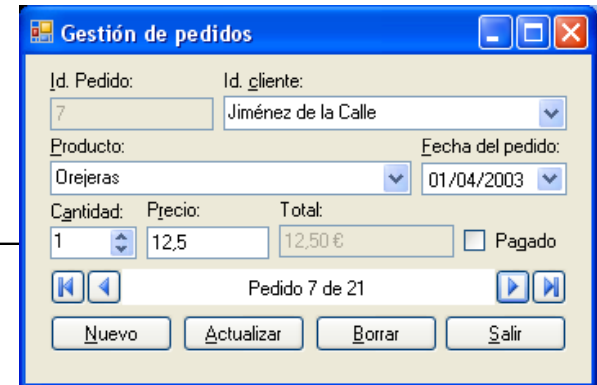
Esteban Monterías  
Jiménez de la Calle  
Noriega Pérez  
Ramírez Salle  
Ruíz Bermudez  
Sanjosé de María  
Pons Gómez

Ciudad: Carchelejo

Provincia: Jaen

# Enlace de datos complejo (VI)

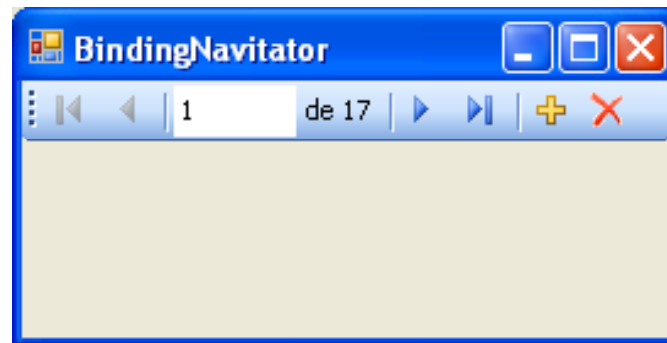
- ❑ En el formulario aparece el nombre del Cliente, aunque lo que se guardará cuando se actualice el conjunto de datos será el identificador de cliente.



```
'Enlace complejo del control cmbClientes
'(aparece el Apellido del cliente en lugar de su Id)
'La propiedad DataSource se enlaza al origen de datos
'que contiene el Apellido y el IdCliente
cmbClientes.DataSource = ds.Tables("Clientes")
'La propiedad DisplayMember se enlaza a la columna que se
'quiere visualizar en el ComboBox
cmbClientes.DisplayMember = "Apellidos"
'La propiedad ValueMember se enlaza a la columna que aparece también
'en ambas tablas
cmbClientes.ValueMember = "IdCliente"
'Hay que enlazar el BindingSource a la propiedad SelectedValue
'que es el valor de la propiedad miembro especificada por la propiedad ValueMember
cmbClientes.DataBindings.Add(New Binding("SelectedValue", bs, "IdCliente", True))
```

# El control BindingNavigator

- ❑ Crea un medio estándar para que los usuarios naveguen y editen los datos de un formulario Windows Forms asociado a un origen de datos de un objeto `BindingSource`.
- ❑ Se trata de un control `ToolStrip` (barra de herramientas) que añade de forma estándar una serie de controles para las acciones más comunes.
  - Es posible añadir nuevos botones y funcionalidades al control en tiempo de diseño o en tiempo de ejecución.



# El control BindingNavigator (II)

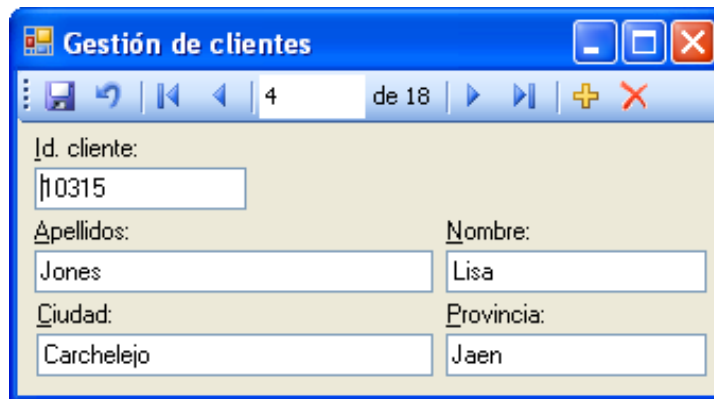
- ❑ Cada elemento de la barra se asocia con un miembro del control.
  - Cada miembro del control se asocia con uno de los miembro del objeto `BindingSource` asociado.

Control de interfaz de usuario	Miembro BindingNavigator	Miembro BindingSource
Mover primero	MoveFirstItem	MoveFirst
Mover anterior	MovePreviousItem	MovePrevious
Posición actual	PositionItem	Current
Count	CountItem	Count
Mover siguiente	MoveNextItem	MoveNext
Mover último	MoveLastItem	MoveLast
Agregar nuevo	AddNewItem	AddNew
Eliminar	DeleteItem	RemoveCurrent

- ❑ Para utilizarlo simplemente hay que asociar la propiedad `BindingSource` del control a un objeto `BindingSource` al que se han enlazado los controles del formulario.

# El control BindingNavigator (III)

- ❑ Ejemplo: Realizar la gestión de la tabla clientes con un control BindingNavigator.



- En tiempo de diseño, a los botones estándar del control BindingNavigator se han añadido un botón Guardar (📁) y otro Cancelar (↶).

# El control BindingNavigator (IV)

```
Imports System.Data
Imports System.Data.OleDb
Public Class frmClientes

    Dim cn As New OleDbConnection
    Dim daClientes As OleDbDataAdapter
    Dim ds As New DataSet
    Dim nombreBBDD = Application.StartupPath & "\Ejemplo.mdb"

    Dim WithEvents bs As New BindingSource

    Private Sub frmClientes_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load
        ConfigurarAccesoADatos()
        EnlazarADatos()
    End Sub
    Private Sub ConfigurarAccesoADatos()
        cn.ConnectionString = "PROVIDER=Microsoft.jet.oledb.4.0;" & _
            Data Source=" & nombreBBDD

        cn.Open()
        daClientes = New OleDbDataAdapter("SELECT * FROM Clientes", cn)
        daClientes.Fill(ds, "Clientes")
        cn.Close()
        Dim claves(0) As DataColumn
        claves(0) = New DataColumn
        claves(0) = ds.Tables("Clientes").Columns("IdCliente")
    End Sub
End Class
```



# El control BindingNavigator (V)

```
        ds.Tables("Clientes").PrimaryKey = claves
        Dim cbClientes As OleDbCommandBuilder = New OleDbCommandBuilder(daClientes)
    End Sub

    Private Sub EnlazarDatos()
        'Establecer BindingSource
        bs.DataSource = ds
        bs.DataMember = "Clientes"
        'Establece el origen de datos del BindingNavigator
        BindingNavigator1.BindingSource = bs
        'Enlazar los controles del formulario
        txtIdCliente.DataBindings.Add(New Binding("Text", bs, "idCliente", True))
        txtApellidos.DataBindings.Add(New Binding("Text", bs, "Apellidos" , True))
        txtNombre.DataBindings.Add(New Binding("Text", bs, "Nombre" , True))
        txtCiudad.DataBindings.Add(New Binding("Text", bs, "Ciudad" , True))
        txtProvincia.DataBindings.Add(New Binding("Text", bs, "Provincia" , True))
    End Sub
    'El método para actualizar no aparece por defecto en el BindingNavigator
    'Hay que codificarlo
    Private Sub BindingNavigatorCancelar_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles BindingNavigatorCancelar.Click
        bs.CancelEdit()
    End Sub
```

# El control BindingNavigator (VI)

```
'El método para actualizar no aparece por defecto en el BindingNavigator
'Hay que codificarlo
Private Sub BindingNavigatorActualizar_Click(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles BindingNavigatorActualizar.Click
    'Si se han producido cambios en el dataset,
    'la colección GetChanges está vacía y
    'se actualiza el conjunto de datos
    bs.EndEdit()
    If Not ds.GetChanges() Is Nothing Then
        Try
            daClientes.Update(ds, "Clientes")
            ds.AcceptChanges()
        Catch ex As Exception
            MessageBox.Show(ex.Message, Me.Text, MessageBoxButtons.OK, _
                MessageBoxIcon.Information)
            ds.Tables("Clientes").RejectChanges()
        End Try
    End If
End Sub
'Al cerrar el formulario comprueba que no hay cambios pendientes
Private Sub frmClientes_FormClosing(ByVal sender As Object, _
      ByVal e As System.Windows.Forms.FormClosingEventArgs) _
    Handles Me.FormClosing
```

# El control BindingNavigator (VII)

```
'Impide salir si el IdCliente está vacío
If txtIdCliente.Text = "" Then
    MessageBox.Show("No puede cerrar la ventana " & _
        "cuando está añadiendo un cliente." & _
        ControlChars.CrLf & _
        "Por favor, cancele la operación antes de salir.", _
        Me.Text, MessageBoxButtons.OK, _
        MessageBoxIcon.Information)

    e.Cancel = True
    Exit Sub
End If
'Si hay cambios pendientes
bs.EndEdit()
If Not ds.GetChanges Is Nothing Then
    If MessageBox.Show("¿Guardar las modificaciones pendientes?", _
        Me.Text, MessageBoxButtons.YesNo, _
        MessageBoxIcon.Information, _
        MessageBoxDefaultButton.Button2) = _
        Windows.Forms.DialogResult.Yes Then
        BindingNavigatorActualizar.PerformClick()
    End If
End If
End Sub
End Class
```