# ACCESO A DATOS CON MICROSOFT ADO.NET

www.profmatiasgarcia.com.ar

# Bases de datos

- Una base de datos es una colección de datos clasificados y estructurados que son guardados en uno o varios archivos.
- Los datos de una base de datos relacional se almacenan en tablas lógicamente relacionadas entre si utilizando campos clave comunes. A su vez, cada tabla dispone los datos en filas y columnas

| DNI | NOMBRE | DIRECCION | TELEFONO |
|---|---|---|---|
| 26803929 | Matías | España 434 | 48790039 |
| 30984284 | Luciana | Palestina 53 | 45637821 |
| ... | | | |

- Como se puede observar, una tabla es una colección de datos presentada en forma de una matriz bidimensional, donde las filas reciben también el nombre de tuplas o registros y las columnas de campos.
- Los usuarios de un SGBD pueden realizar sobre una determinada base operaciones como insertar, recuperar, modificar y eliminar datos, así como añadir nuevas tablas o eliminarlas. Estas operaciones se expresan generalmente en un lenguaje denominado SQL.

# SQL - Structured Query Language

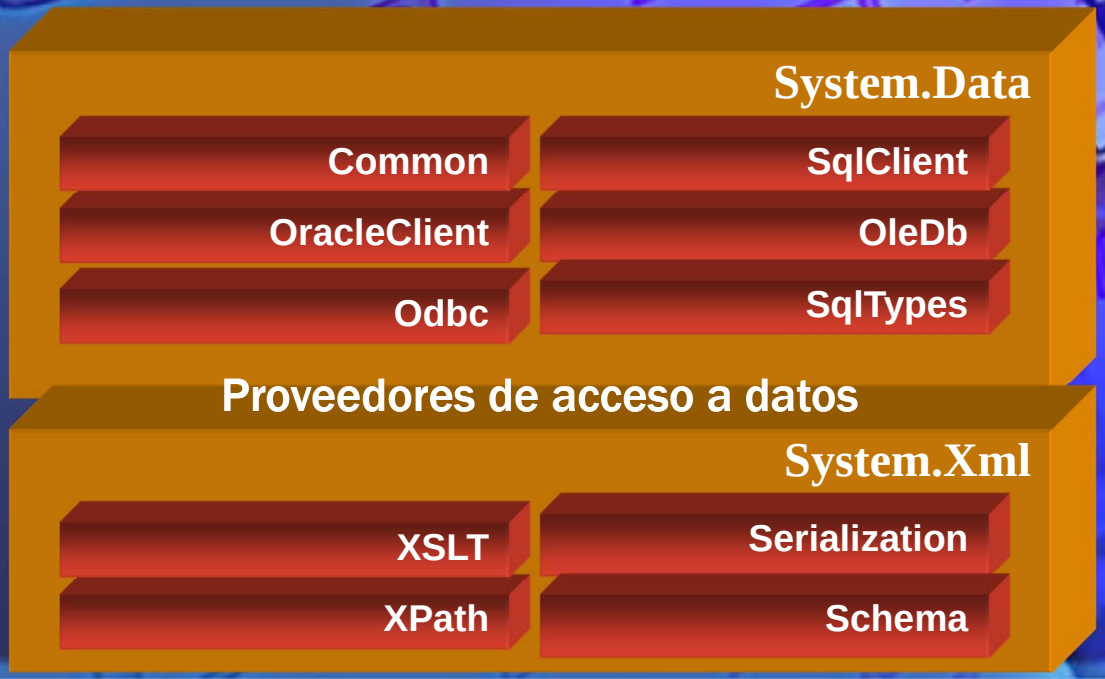
- Es el lenguaje estandar para interactuar con bases de datos relacionales y es soportado practicamente por todos los SGBD actuales.
- Se compone, entre otras, de sentencias
  - DDL (Data Definition Language), que se encarga de la modificación de la estructura de los objetos de la base de datos. Incluye órdenes para modificar, borrar o definir las tablas en las que se almacenan los datos de la base de datos. Existen cuatro operaciones básicas: CREATE, ALTER, DROP y TRUNCATE.
  - DML (Data Manipulation Language) que permite a los usuarios llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado. Las operaciones básicas: INSERT, UPDATE, DELETE y SELECT.
- Los sistemas de gestión de base de datos con soporte SQL más utilizados son:
  - DB2
  - Informix
  - Interbase
  - MariaDB
  - Microsoft SQL Server
  - MySQL
  - Oracle
  - PostgreSQL
  - Sybase ASE
- Aunque muchos SGBD utilicen el lenguaje, puede haber diferencias en las sentencias entre uno y otro, como por ejemplo en PL/SQL de Oracle y el SQL de MS SQL Server.

# Acceso a datos con ADO.NET

- ADO.NET es una tecnología desarrollada por Microsoft para el acceso a datos potente y de fácil utilización.
- No depende de conexiones continuamente activas ya que se diseño entorno a una arquitectura donde las aplicaciones se conectan a la base de datos solo durante el tiempo necesario para extraer o actualizar los datos. De esta forma, la base de datos no mantiene conexiones que la mayor parte del tiempo permanecen inactivas, lo que se traduce en dar servicio a muchos mas usuarios, mejora la performance y facilita la escalabilidad.
- Las interacciones con la base de datos se realizan mediante ordenes para acceso a los datos, que son objetos que encapsulan las sentencias SQL o los procedimientos almacenados que definen la operación a realizar sobre el origen de datos.
- Los datos requeridos normalmente se almacenan en memoria caché en *conjuntos de datos*, lo que permite trabajar sin conexión sobre una copia temporal de los datos obtenidos. Los *conjuntos de datos* son independientes de los *orígenes de datos*.
- El formato de transferencia de datos es XML, se basa en texto, lo que permite enviarla mediante cualquier protocolo, como por ejemplo HTTP.

# Acceso a datos con ADO.NET

- ADO.NET es un subconjunto de la .NET Framework Class Library, que contiene todas las funcionalidades necesarias para conectarse e interactuar con dos tipos de repositorios permanentes de información:
  - Bases de Datos, como Microsoft SQL Server (clases del namespace System.Data, que se encuentran compiladas en System.data.dll)
  - Archivos XML (clases del namespace System.XML, que se encuentran compiladas en System.Xml.dll)




ADO.NET provee una arquitectura extensible, posibilitando que terceras partes creen sus propios proveedores de acceso nativo para aplicaciones .NET. Algunos ejemplos de esto son:

- Data Provider For DB2, desarrollado por IBM
- Oracle Data Provider For .NET, desarrollado por Oracle
- Providers de acceso nativo a bases de datos OpenSource, como MySQL y PostgreSQL

# ADO.NET Soporte a XML

## DocumentNavigator

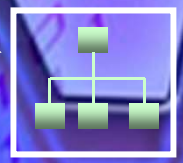
permite navegar libremente por la estructura de un documento XML una vez que ha sido cargado dentro de una instancia de la clase XmlDocument

## XmlTextWriter

permite que datos XML puedan ser escritos a un archivo XML o a un stream, y provee mecanismos de validación para asegurar que sólo datos XML válidos y bien formados sean escritos.


## XmlReader

Provee un mecanismo de lectura forward-only de un documento XML



## XmlDocument

actúa como un contenedor de datos XML, representando en un modelo de objetos en memoria toda la esctructura de un documento XML.

## XmlTextReader

para leer datos de un documento XML o un stream

## XmlValidatingReader

similar a XmlTextReader, pero pensada para validaciones DOM

## XmlNodeReader

permite leer datos de un nodo XML

# ADO.NET Clases mas comunes


XxxConnection

XxxCommand

XxxDataAdapter

DataSet

XxxDataReader

Maneja la conección a una base de datos

Ejecuta comandos contra una base de datos

Intercambia datos entre un dataset y una base de datos

Copia local de datos relacionales

Provee acceso de lectura a datos read-only, Forward-only

# Connection

Este objeto es el encargado de establecer una conexión física con una base de datos determinada.

Para establecer la conexión con una determinada fuente de datos, no sólo debemos establecer la cadena de conexión correctamente, sino que además deberemos usar los parámetros de conexión y el proveedor de acceso a datos adecuado.

Con este objeto, podremos además abrir y cerrar una conexión.

La conexión con la base de datos es la que mas recursos del sistema consume.

```
string strConn = "data source=localhost; " +
"initial catalog=DB_Name; integrated security=true";

SqlConnection aConn = new SqlConnection(strConn);

aConn.Open();
// Ejecutar Queries y/o comandos
aConn.Close();
```

# Connection

```
string StrConn=
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=“ +
"c:\\Data\\DB_Name.mdb;User Id=admin;Password=;";

OleDbConnection aConn = new OleDbConnection(StrConn);
```

```
string StrConn=
"Data Source=ThisOracleServer;Integrated
Security=yes;";

OracleConnection aConn = new OracleConnection
(StrConn);
```

# Command

Este objeto es el que permite representar una determinada sentencia SQL o un Stored Procedure.

Aunque no es obligatorio su uso, en caso de necesitarlo, lo utilizaremos conjuntamente con el objeto DataAdapter que es el encargado de ejecutar la instrucción indicada.

```
OleDbCommand ordenSQL = new OleDbCommand(
"SELECT nombre, telefono FROM telefonos", conexion);

string query1 =
     "SELECT IDProd, Precio, Prod from dbo.productos"
      + "WHERE Precio > @precioingresado "
      + "ORDER BY Precio DESC;";
SqlCommand command = new SqlCommand(query1, conexion);
command.Parameters.AddWithValue("@precioingresado",
paramValue);
```

# DataReader

Cuando una aplicación solo necesite leer datos (no actualizarlos), no sera necesario almacenarlos en un conjunto de datos, basta utilizar un objeto lector de datos en su lugar.

Se trata de un objeto de acceso a datos muy rápido.

- Establece una conexión con una fuente de datos y trabaja con esta fuente de datos sin desconectarnos de ella.
- Esta conexión se establezca en un modo de sólo lectura.
- Este objeto puede usar a su vez el objeto Command o el método ExecuteReader.
- El acceso a los datos se realiza de manera Forward-only, solo se puede avanzar en la lectura de los registros, no hay vuelta atrás.
- Debido a su naturaleza y características, este objeto es bastante rápido a la hora de trabajar con datos.
- Como es lógico, consume además menos memoria y recursos que un objeto DataSet por ejemplo.
- El objeto DataReader recupera un nutrido conjunto de valores llenando un pequeño buffer de datos e información. Si el número de registros que hay en el buffer se acaban, el objeto DataReader regresará a la fuente de datos para recuperar más registros.

# DataReader

```
string CadenaConexion =
"server=.;uid=sa;password=matias;database=VideoClub";

SqlConnection MiConexion = new
SqlConnection(CadenaConexion);

SqlCommand MiComando = new SqlCommand(
"SELECT TITULO FROM ALQUILERES, PELICULAS " +
"WHERE PELICULACODBARRAS = CODBARRAS", MiConexion);

MiConexion.Open();

SqlDataReader MiDataReader =
MiComando.ExecuteReader();

while (MiDataReader.Read()) //paso a proximo registro
{
  textBox1.Text += MiDataReader["titulo"].ToString();
}
MiConexion.Close();
```

# DataAdapter

Lo más habitual será que nos encontremos trabajando con ambientes y accesos a datos desconectados.

Cuando deseamos establecer una comunicación entre un origen de datos y un DataSet, utilizamos como intermediario a un objeto DataAdapter.

A su vez, un DataAdapter contiene 4 objetos que debemos conocer:

- SelectCommand es el objeto encargado de realizar los trabajos de selección de datos con una fuente de datos dada. En sí, es el que se encarga de devolver y rellenar los datos de un origen de datos a un DataSet.
- DeleteCommand es el objeto encargado de realizar las acciones de borrado de datos.
- InsertCommand es el objeto encargado de realizar las acciones de inserción de datos.
- UpdateCommand es el objeto encargado de realizar las acciones de actualización de datos.

Cada proveedor de acceso a datos posee su propio objeto DataAdapter.

Cuando realizamos alguna modificación o acción sobre la fuente de datos, utilizaremos siempre el objeto DataAdapter a caballo entre el objeto DataSet y la fuente de datos establecida a través de la conexión con el objeto Connection.

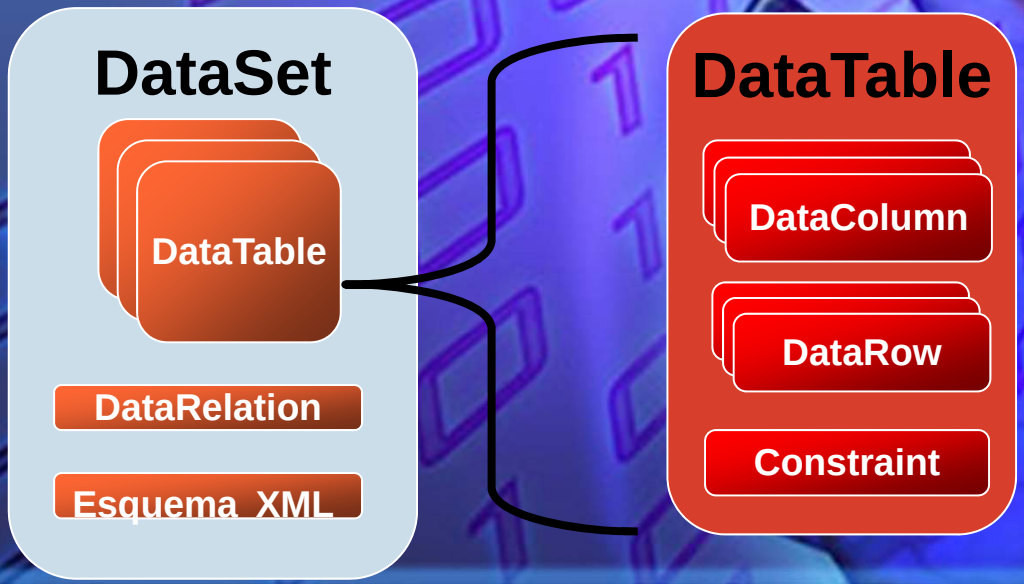
Con el objeto DataAdapter, podremos además realizar diferentes acciones sobre nuestras bases de datos, acciones como la ejecución general de sentencias de SQL no sólo para seleccionar un conjunto de datos, sino para alterar el contenido de una base de datos o de sus tablas.

# DataAdapter

```
SqlConnection MiConexion = new
SqlConnection("server=.;uid=sa;password=matias;databas
e=VideoClub");

string strSql = "SELECT Socio, FechaAlquiler FROM
ALQUILERES";

SqlDataAdapter MiAdaptador = new
SqlDataAdapter(strSql, MiConexion);

DataSet MiDataSet = new DataSet();

MiAdaptador.Fill(MiDataSet,"Alquileres");

//Recorremos todas las filas de la tabla alquileres
del DataSet
foreach (DataRow Fila in MiDataSet.Tables[0].Rows)
textBox1.Text += Fila["Socio"].ToString() + "\t" +
Fila["FechaAlquiler"].ToString() + "\n";

MiDataSet = null;
```

# DataSet

El DataSet es una representación residente en memoria de datos relacionales, independiente de la base de datos y del protocolo utilizado para interactuar con la misma. Un DataSet, al igual que una base de datos, está compuesto por un conjunto de tablas (colección de clases "DataTable"), cada una de las cuales está compuesta a su vez por un conjunto de filas (colección de clases "DataRow") y columnas (colección de clases "DataColumn"). Dentro de un DataSet pueden establecerse relaciones entre DataTables, y hasta restricciones de integridad referencial (Claves Primarias y Foráneas). Internamente, los DataSets representan toda su estructura y datos contenidos en formato XML.

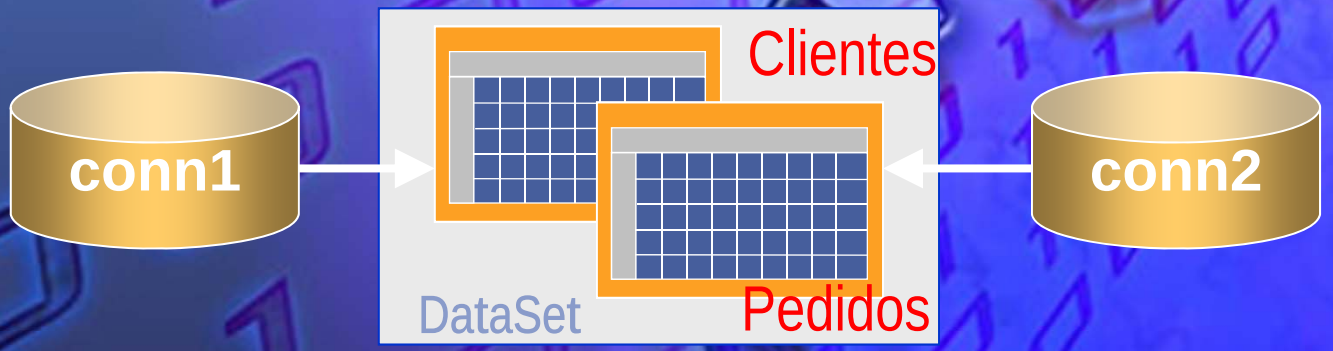


# Crear y vincular un DataSet

```
DataSet ds = new DataSet();
// el método Fill ejecuta el SelectCommand
da.Fill(ds, "Autores");

ds.Tables["Autores"].Rows.Count;

string str="";

// Accedo al DataTable
foreach(DataRow r in ds.Tables["Autores"].Rows)
{
  str += r[2];
  str += r[“nombre_autor"];
}

// vinculo el DataSet con un DataGrid para mostrar
DataGridAutores.DataSource = ds;
DataGridAutores.DataMember = "Autores“;
DataGridAutores.DataBind();
```

# Utilizar múltimples tablas

```
SqlDataAdapter daClientes = New SqlDataAdapter
    ("select * from Clientes", conn1);
daClientes.Fill(ds, “Clientes");

SqlDataAdapter daPedidos = New SqlDataAdapter
    ("select * from Pedidos", conn2);
daPedidos.Fill(ds, “Pedidos");
```



# DataSets vs DataReaders

| 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 |

# DataTable

- Este objeto nos permite representar una determinada tabla en memoria, de modo que podamos interactuar con ella.
- A la hora de trabajar con este objeto, debemos tener en cuenta el nombre con el cuál definamos una determinada tabla, ya que los objetos DataTable son sensitivos a mayúsculas y minúsculas.
- Puede ser mapeado a una tabla física en la fuente de datos
- Pueden relacionarse por medio de DataRelations
- Propiedades de interés:
  - Columns: Devuelve objetos ColumnsCollection de DataColumns
  - Rows: Devuelve objetos DataRow como objetos RowsCollection
  - ParentRelations: Devuelve un objeto RelationsCollection
  - Constraints: Devuelve un objeto ConstraintsCollection de las tablas
  - DataSet: Devuelve un objeto DataSet de uno DataTable
  - PrimaryKey: Obtiene el objeto DataColumns el cual crea la llave primaria de las tablas

# DataTable

- Crear un objeto DataTable y adicionarlo a un DataSet

```
DataSet dsBase = new DataSet();

// Creo un objeto DataTable: “Clientes”.
DataTable dtClientes= new DataTable( “Clientes” );

// Creo y agrego columnas a la tabla
// 1. De forma explicita
DataColumn dc = new DataColumn( “IDCliente”, Int16 );
dtClientes.Columns.Add( dc );

// 2. De forma implicita
dtClientes.Columns.Add( “First_Name”, String );
dtClientes.Columns.Add( “Last_Name”, String );

// Agrego el DataTable al DataSet
dsBase.Tables.Add( dtClientes );
```

# DataRelation

- Usado para crear relaciones lógicas
  - Crea relaciones entre (2) objetos DataTable
  - Requiere un objeto DataColumn de cada objeto DataTable
  - El DataType de ambos DataColumns tiene que ser el mismo
    - No se puede relacionar un Int32 DataColumn con un String DataColumn
  - La relación es nombrada por el programador
    - DataRelation dr = new DataRelation( “myRelacion”,...)
- Hace la navegación relacional posible
- RelationsCollection contiene todos los objetos DataRelations
  - Accedido a través de las propiedades del DataSet Relations

# DataRelation

- Como crear un objeto DataRelation:
  - Se obtienen los objetos DataColumn para relacionar
  - Crear un objeto DataRelation nombrado usando las columnas
  - Adicionar la relación al DataSet

```
// Elegimos las columnas para relacionar
DataColumn Columna1, Columna2;
Columna1= DataSet.Tables[“Clientes"].Columns[“ID_Cliente"];
Columna2= DataSet.Tables[“Pedidos“].Columns[“ID_Cliente"];

// Creo la relación PedidosDeClientes
DataRelation dr;
dr = New DataRelation(“PedidosDeClientes", Columna1, Columna2);

// Agrego la relación al DataSet
ds.Relations.Add( dr );
```

# DataView

- Este objeto nos permite crear múltiples vistas de nuestros datos, además de permitirnos presentar los datos.
- Permite editar, ordenar y filtrar, buscar y navegar por un conjunto de datos determinado.
- Es la clase que nos permite representar los datos de la clase DataTable, permitiéndonos editar, ordenar y filtrar, buscar y navegar por un conjunto de datos determinado.
- Crear múltiples vistas en un objeto DataTable
- Invisible a los controles de Interfase de usuario
- Propiedades de interés:
  - Item: Recupera una fila de datos de una tabla especifica
  - Table: Recupera o coloca el DataTable asociado
  - Sort: Obtiene o coloca  la clase de columna de la tabla y el orden de la clase
  - RowFilter: Obtiene o fija la expresión utilizada para filtrar las filas
  - RowStateFilter: Consigue o fija el filtro del estado de la fila
    - None, Unchanged, New, Deleted, ModifiedCurrent, y otros

# DataView

- Creando objetos DataView

```
// Creo las vistas de mi tabla de clientes
DataView vista1 = new DataView( TablaClientes )
DataView vista2 = new DataView( TablaClientes );
// Creo una vista de Clientes ordenados ascendentemente por nombre
vista1.Sort = “Nombre ASC”;

// Modifico la vista para que solo muestre los registros
originales
vista2.RowStateFilter= DataViewRowState.ModifiedOriginal;

// Seteo el databinding
DataGrid miGrid = new DataGrid();
myGrid.SetDataBinding( vista1, “Clientes”);
//...
```

# DataSetView

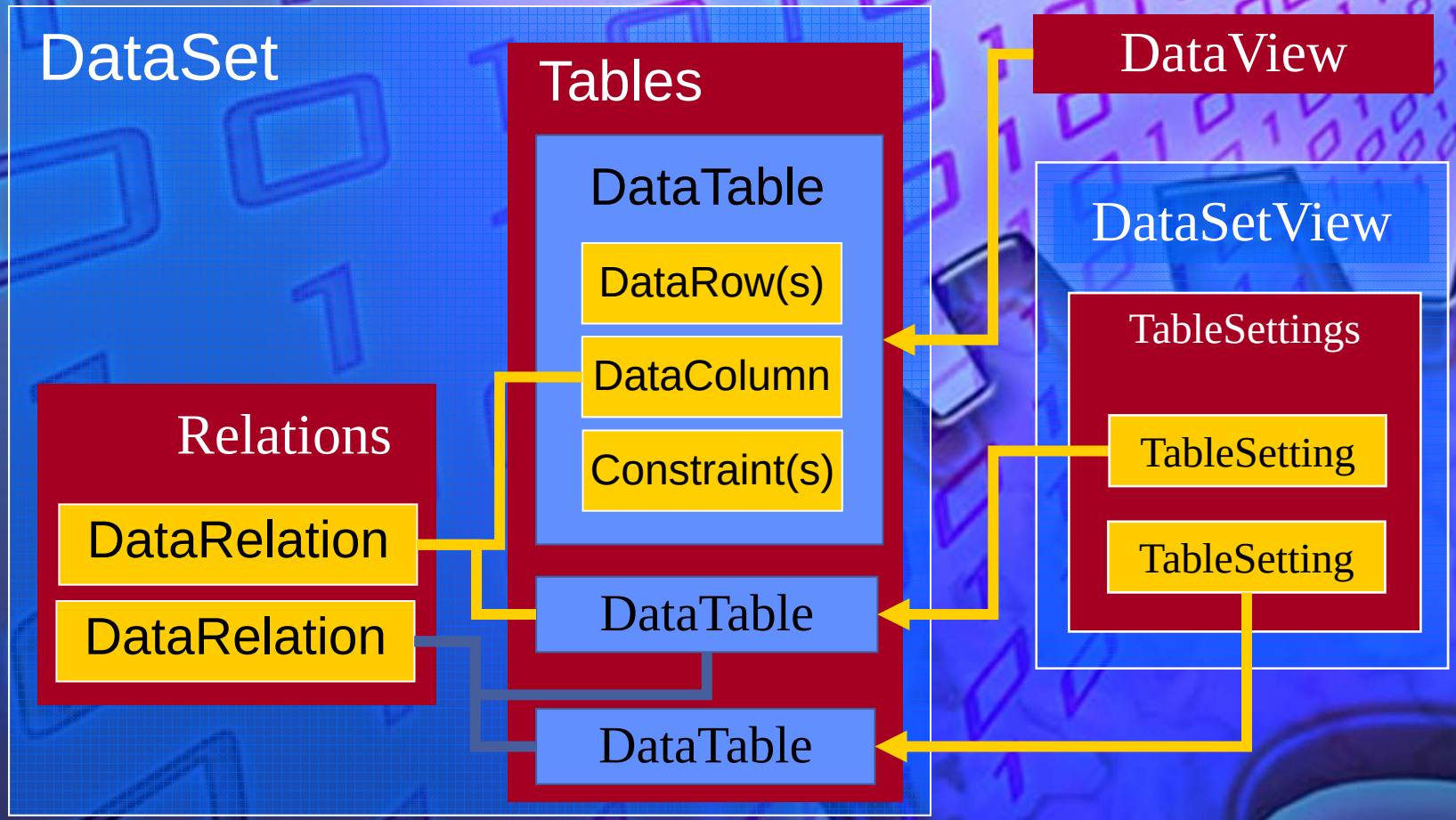
- Similar a un DataView pero orientado a DataSet
- Usado para crear múltiples vistas en un DataSet
  - Capacidad de fijar automáticamente los filtros en las tablas
- Propiedades de interés:
  - TableSettings: Obtiene/fija los ajustes de las vistas en la tabla
  - DataSet: Obtiene o fija el DataSet para crear la vista

# DataSetView

- Creando un objeto DataSetView usando un DataSet

```
// Creo la vista
DataSetView dsVista1 = new DataSetView( miDS );

// Creo los objetos tablesetting
TableSetting ts1, ts2 ;
ts1 = new TableSetting( miDS.Tables[“Pedidos”],
                        “ID_Cliente”, “ID_Cliente<100”,
                        DataViewRowState.CurrentRows );
ts2 = new TableSetting( miDS.Tables[“Pedidos”],
                        “ID_Producto”, “ID_Producto>1011”,
                        DataViewRowState.CurrentRows );

// Agrego los TableSettings al DataSetView...
dsVista1.TableSettings.Add( ts1 );
dsVista1.TableSettings.Add( ts2 );
```

# DataSet, DataRelation, DataView y DataSetView



# Conexión MySQL

- http://balusoft.wordpress.com/2011/04/15/conectarse-a-mysql-desde-c-visual-studio-2010/

# Webgrafía & Licencia:

- Textos tomados, corregidos y modificados de diferentes páginas de Internet, tutoriales y documentos.
- Este documento se encuentra bajo Licencia Creative Commons 2.5 Argentina (BY-NC-SA), por la cual se permite su exhibición, distribución, copia y posibilita hacer obras derivadas a partir de la misma, siempre y cuando se cite la autoría del **Prof. Matías E. García** y sólo podrá distribuir la obra derivada resultante bajo una licencia idéntica a ésta.
- Autor:


***Matías E. García***

Prof. & Tec. en Informática Aplicada
www.profmatiasgarcia.com.ar
info@profmatiasgarcia.com.ar