ISIL APRENDE HACIENDO

TITULO

DESARROLLO DE APLICACIONES I

Sesión Nro. 04

Objetivos de la sesión:

Al culminar la presente podrás:

- Conocer la arquitectura del marco de trabajo para conexiones a base de datos.
- Conocer la Arquitectura de ADO Net
- Conocer mas en detalle el uso de las clases:
  - Sql Connection
  - SqlCommand
  - SqlDatareader
  - SqlDataAdapter
- Aprender a ejecutar procedimientos almacenados desde Visual Studio con ADO. Net

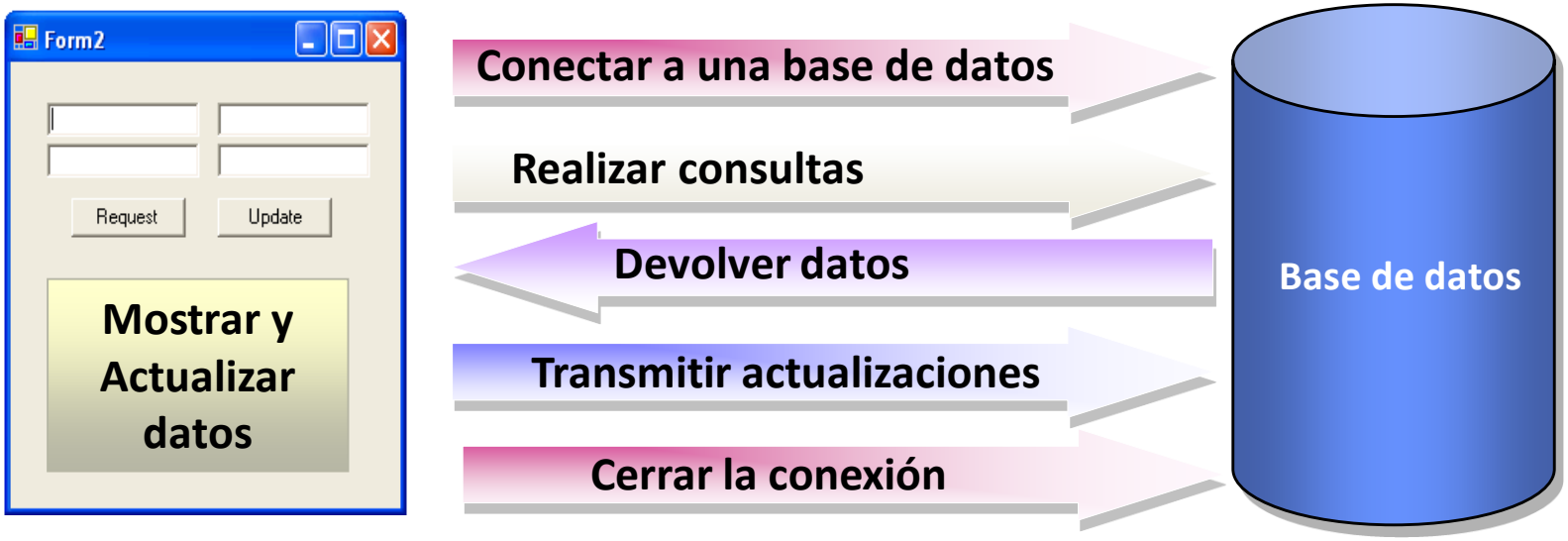
Temario

1. Introducción a la programación con acceso a Bases de datos
2. Definición y arquitectura de ADO .NET
3. Los DataProviders
4. La clase SqlConnection
5. La clase SqlCommand y la colección SqlParameters
   - Los métodos ExecuteScalar y ExecuteReader
   - La clase SqlDataReader
   - Ejecución de Procedimientos almacenados desde SqlCommand.

Tema Nro. 1:
Introducción a la Programación con acceso a Base de Datos

Vista de la programación de bases de datos

Tareas habituales en la programación de bases de datos



En muchas aplicaciones, la conexión se cierra después de que el usuario accede a los datos y vuelve a abrirse cuando el usuario reenvía actualizaciones o realiza más peticiones

¿Qué es un entorno conectado?

- Un entorno conectado es aquel en que los usuarios están conectados continuamente a una fuente de datos.
- Las actualizaciones y consultas se dan en tiempo real
- Ventajas:
  - El entorno es más fácil de mantener
  - La concurrencia se controla más fácilmente
  - Es más probable que los datos estén más actualizados que en otros escenarios
- Inconvenientes:
  - Debe existir una conexión de red constante
  - Escalabilidad (crecimiento en el volumen de usuarios) limitada

¿Qué es un entorno desconectado?

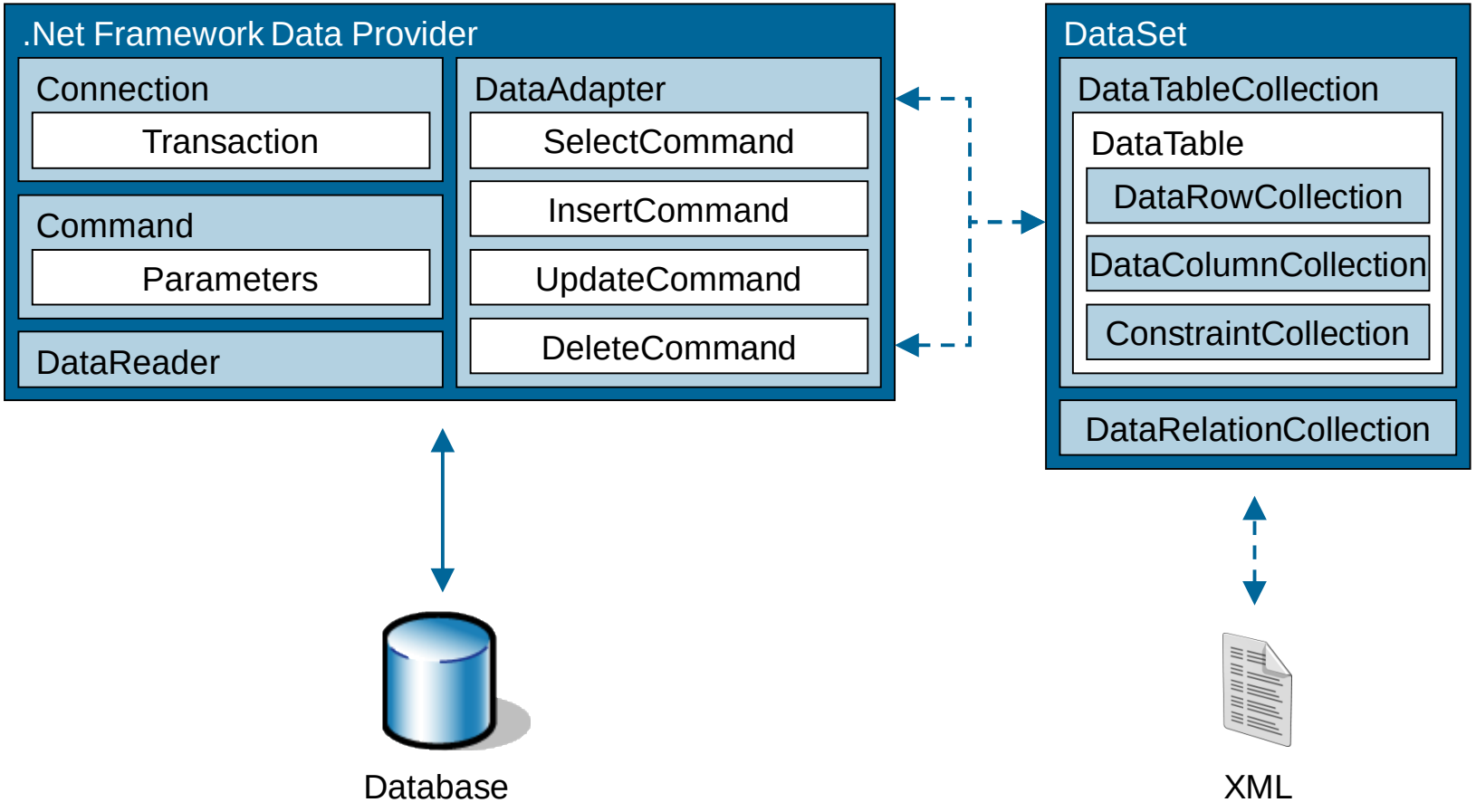
- Un entorno desconectado es aquel en el que los datos pueden actualizarse sin estar conectados y los cambios realizados se plasman posteriormente en la base de datos
- Ventajas:
  - Las conexiones se utilizan durante el menor tiempo posible, permitiendo que se optimice el recurso de conexiones para que den servicio a más usuarios
  - Un entorno desconectado mejora la escalabilidad y el rendimiento de las aplicaciones
- Inconvenientes:
  - Los datos no siempre están actualizados. Se corre el riesgo de lo que se vea no necesariamente es lo que este registrado en la base de datos
  - Pueden producirse conflictos de cambios que deben solucionarse

Tema Nro. 2:
Definición y arquitectura de ADO .NET

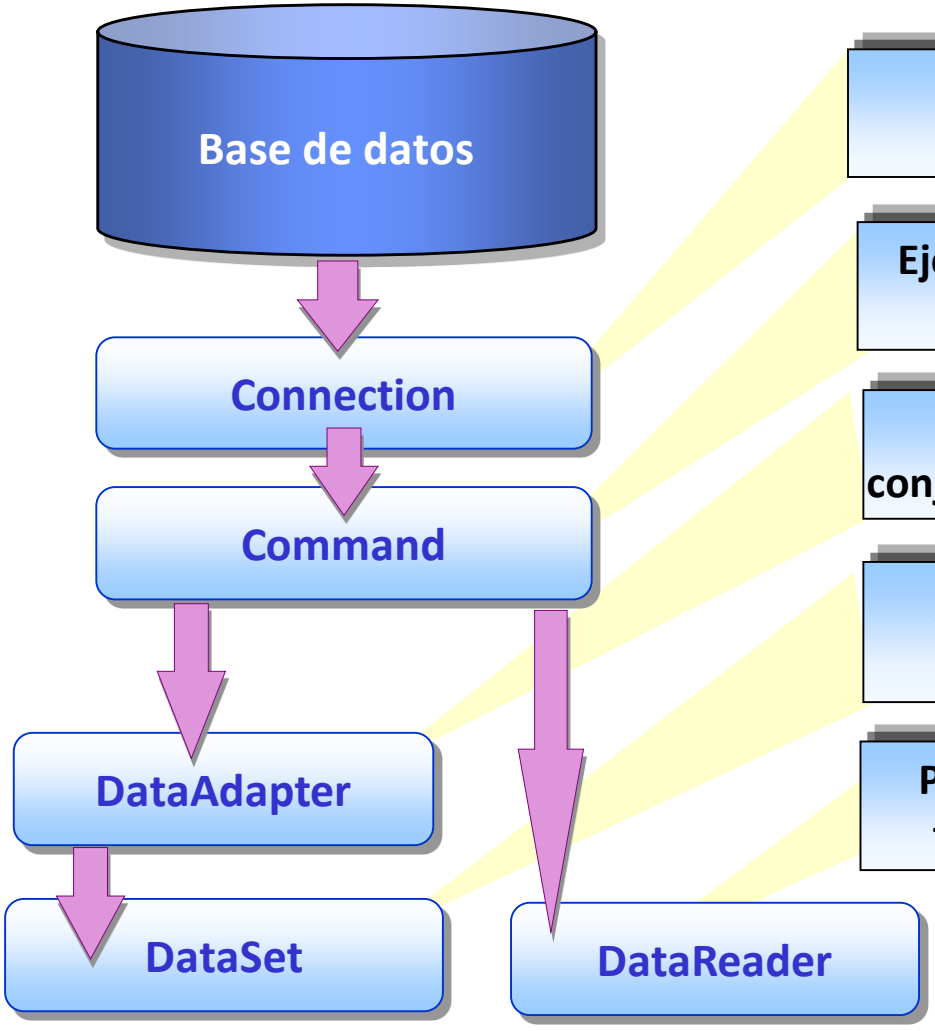
¿Qué es ADO .NET?

- Es un estándar Microsoft para el acceso a diversos orígenes de datos.
- Este estándar es en realidad una librería compuesta por un conjunto de clases que, permiten el trabajo con bases de datos desde cualquier tipo de aplicaciones (escritorio, WEB, móvil, etc.), de forma segura y eficiente

Arquitectura ADO.Net



Arquitectura ADO.NET



Gestiona la conexión a una base de datos

Ejecuta un comando de consulta en la base de datos

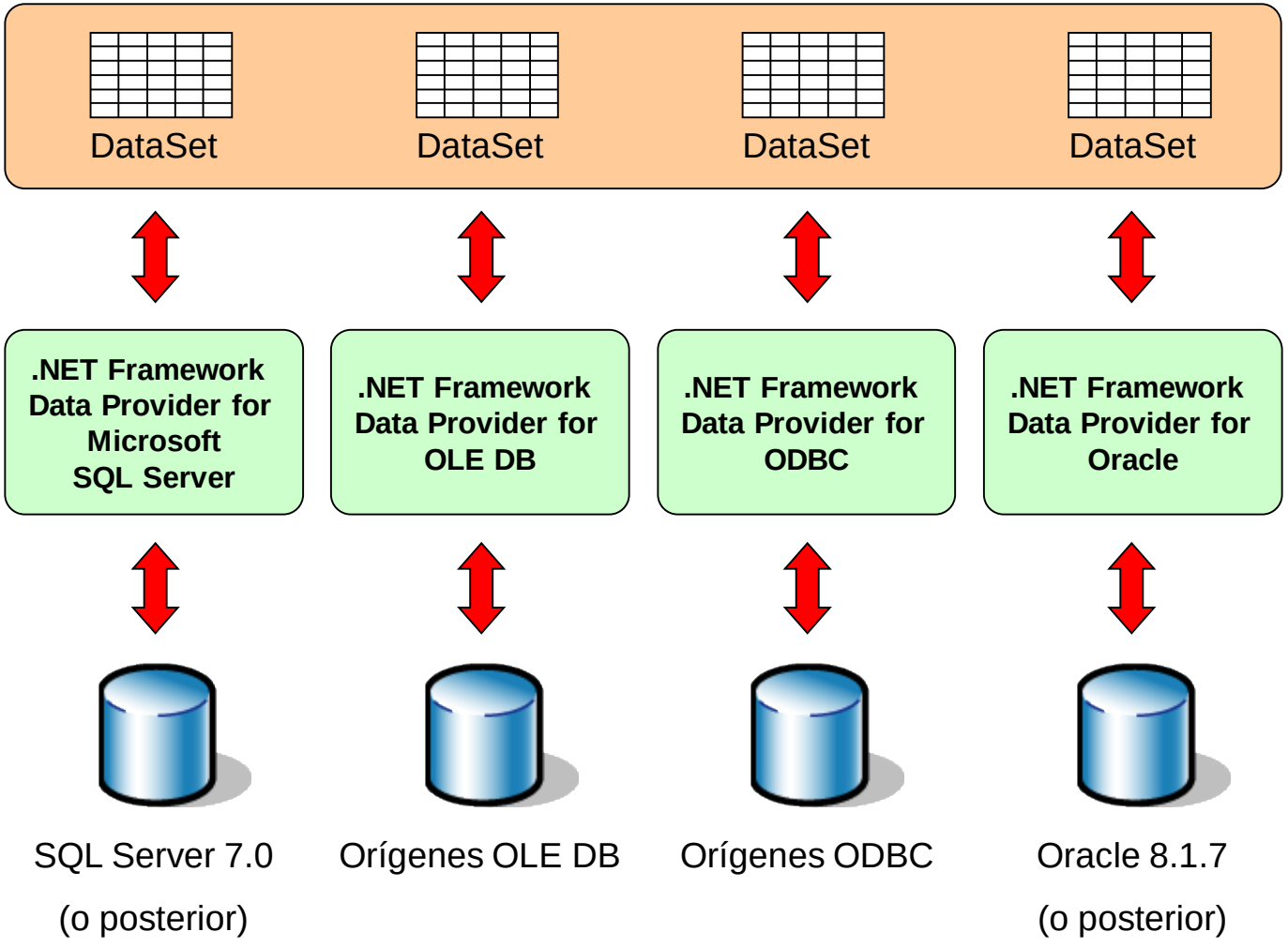
Intercambia datos entre el conjunto de datos y la base de datos

Almacena datos en un caché distinto de la base de datos

Proporciona acceso eficaz a un flujo de datos de sólo lectura

Tema Nro. 3:
Los Data Providers

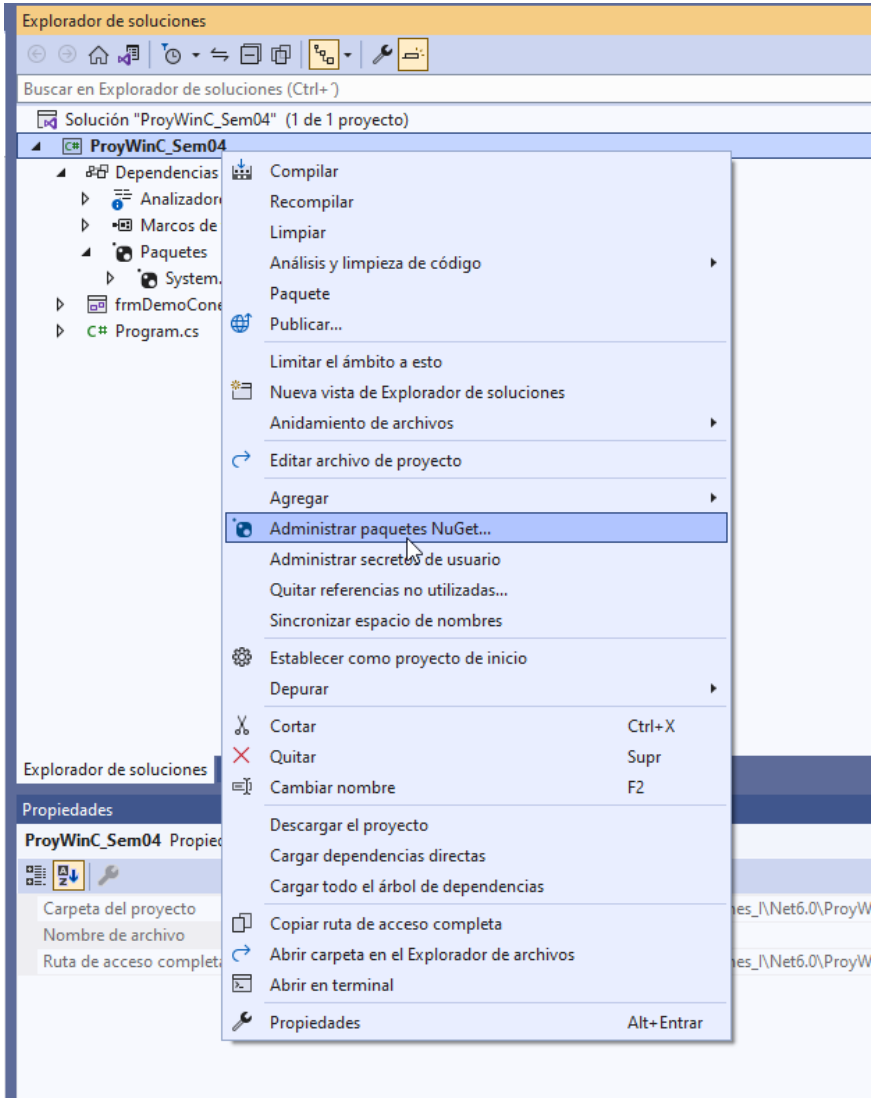
Los .NET Framework Data Providers



Tema Nro. 4:
La clase SQLConnection

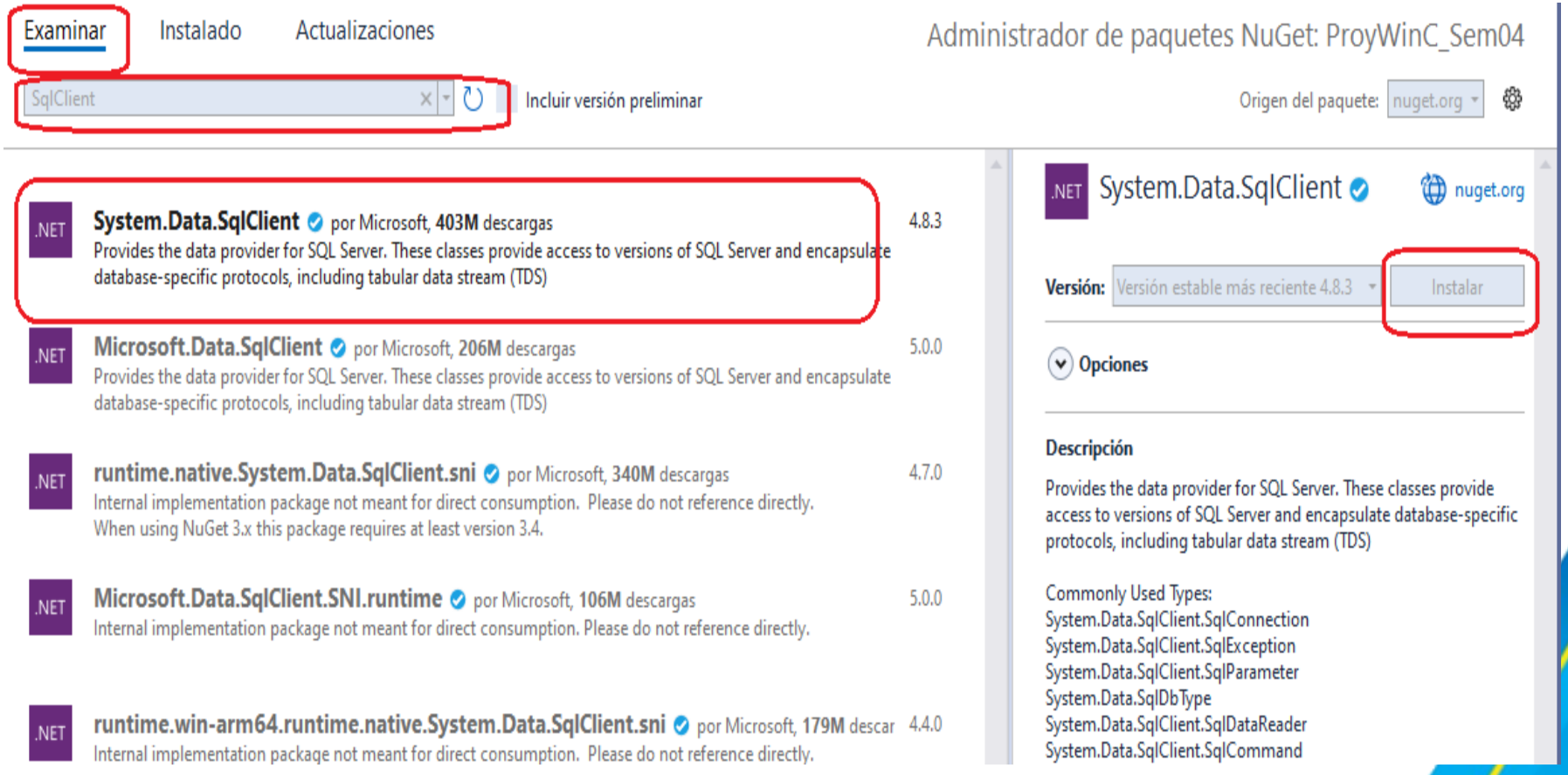
Agregar librería System.Data.SqlClient

Paso 1:



Agregar librería System.Data.SqlClient

Paso 2:



Cómo utilizar un objeto SQLConnection

- Utilizamos SQL Connection para conectarnos a una base de datos SQL Server (versión 7.0 o mas).
- Mediante la propiedad ConnectionString se define el servidor, la base de datos y el tipo de autenticación a emplear.
- Y obviamente, determinamos en que momento se abre y cierra la conexión con los métodos Open y Close para efectuar las operaciones correspondientes.

```
SqlConnection cnx = new SqlConnection();
string connstr = null;
connstr = "server=.;Database=Pubs;Integrated Security=SSPI";
cnx.ConnectionString = connstr;
cnx. Open();
//…las acciones que se quieran realizar
cnx.Close();
```

Cómo utilizar un objeto SQLConnection

- Ejemplo con autenticación SQL

```
SqlConnection cnx = new SqlConnection();
string connstr = null;
connstr = "server=.;Database=Pubs; user id=user01;pwd=12345";
cnx.ConnectionString = connstr;
cnx. Open();
//…las acciones que se quieran realizar
cnx.Close();
```

- Ejemplo sobrecargando el constructor para definir la cadena de conexión:

```
SqlConnection cnx = new SqlConnection
  ("server=.;Database=Pubs; Integrated Security=SSPI");
cnx. Open();
//…las acciones que se quieran realizar
cnx.Close();
```

Cómo utilizar un objeto SQLException

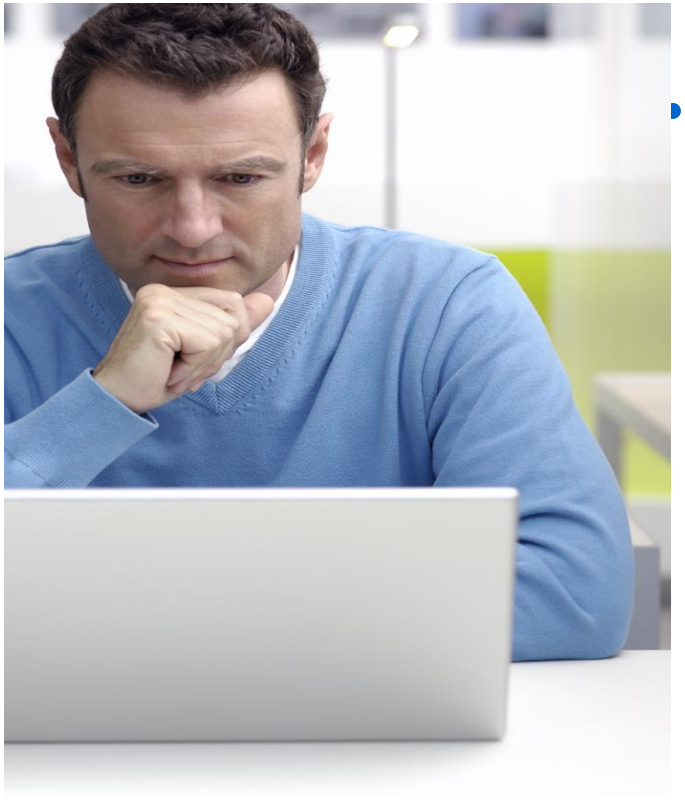
- Cuando realicemos alguna operación con bases de datos debemos ser conscientes que pueden darse eventos o sucesos inesperados.
- A estos se les llama Excepciones y en el ámbito de operaciones con base de datos SQL Server se manejan dentro de una clase especifica llamada SQLException.
- Se recomienda entonces trabajar siempre el código dentro de bloques try-catch de tal forma que cualquier excepción (caída del servidor, rompimiento de reglas de integridad referencial, etc.,) se puedan manejar sin que la aplicación se interrumpa de manera abrupta.

Cómo utilizar un objeto SQLException

- **Ejemplo:**

```
try
{
SqlConnection cnx = new SqlConnection();
string connstr = null;
connstr = "server=.;Database=Pubs;Integrated Security=SSPI";
cnx.ConnectionString = connstr;
cnx. Open();
//…las acciones que se quieran realizar
}
catch (SqlException ex)
{
MessageBox.Show("Error: " + ex.Message, "Mensaje",
MessageBoxButtons.OK, MessageBoxIcon.Error);
}
finally
{
 if (cnx.State == ConnectionState.Open)
 {
  cnx.Close();
}
```

Demostración 1: Uso de la clase SQLConnection



- En esta demostración, aprenderemos a utilizar la clase SQLConnection y comprobar su funcionamiento

Tema Nro. 5:
La clase SQLCommand

Cómo utilizar un objeto SQLCommand

- Utilizamos SQLCommand para:
  - Ejecutar una acción sobre la base de datos en un escenario conectado, por medio de un texto SQL o la ejecución de un Store Procedure
  - Especificar el origen de datos de un SQLDataAdapter
- Sus propiedades mas importante son:
  - Connection: Que establece con que instancia de conexión trabajara el comando.
  - CommandType. Que define el tipo de comando. Puede ser básicamente tipo textual, de Store Procedure o tipo tabla
  - CommnadText: Que define el texto SQL a ejecutar (si es de tipo textual) , el nombre del Store Procedure (si es de tipo Store Procedure) o el nombre de la tabla a manejar (si es de tipo tabla)

Cómo utilizar un objeto SQLCommand

- Para ejecutar la operación asignada al comando puede emplear los siguientes métodos:
  - ExecuteScalar: Para operaciones de consulta que devuelve solo UN DATO (independiente sea el tipo de dato : cadena , fecha, entero, real, etc.)
  - ExecuteReader: Para operaciones de consulta que pueden devolver uno o mas registros, generando el resultado en un objeto SQLDatareader.
  - ExecuteNonQuery: Para operaciones de actualización, como insertar, modificar o eliminar registros u objetos de la base de datos.

• Ejemplo ExecuteScalar

```
    try
{
    cmd.Connection = cnx;
    //Obtiene el maximo precio de productos
    cmd.CommandText “usp_MaximoPrecioProducto";
    cmd.CommandType = CommandType.StoreProcedure;
    cnx.Open();
    MessageBox.Show("El Maximo precio es :" + cmd.ExecuteScalar(),
       "Resultado", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
catch (SqlException ex)
{
    MessageBox.Show(ex.Message);
}
finally
{
    if (cnx.State == ConnectionState.Open)
    {
       cnx.Close();
    }
}
```

La clase SQLDataReader

- Los objetos de la clase SQLDataReader se construyen al llamar al método ExecuteDataReader de un comando.
- Estos objetos son solo de lectura y su navegación es solo hacia adelante.
- Para consumir los registros de un Data Reader se debe invocar al método *Read* para leer cada registro, el cual devuelve False cuando no hay más registros.
- Invocar al método C*lose* para liberar el lector y la conexión

• Ejemplo ExecuteDataReader

```
try
        {
           cmd.Connection = cnx;
            cmd.CommandType = CommandType.Text;
           cmd.CommandText = “usp_ListarProducto”
           cnx.Open();
             SqlDataReader dtr = cmd.ExecuteReader();
           //Llenamos  la lista
           ListBox1.Items.Clear();
           while (dtr.Read())
           {
              ListBox1.Items.Add(dtr["des_pro“]);
           }
           dtr.Close();
        }
        catch (SqlException ex)
        {
           MessageBox.Show(ex.Message);
        }
        finally
        {
           if (cnx.State == ConnectionState.Open)
           {
              cnx.Close();
           }
        }
```

La colección SQLParameters

- Podemos ejecutar comandos textuales que manejen parámetros, de tal forma que para implementar su ejecución debemos agregar parámetros al comando con instancias de la clase SQLParameter.
- También se amerita al caso cuando se ejecuten comandos de tipo Store Procedure y estos posean parámetros ya se de entrada o de salida.
- Al agregar un parámetro hay que especificar su nombre, su tipo de dato ,su valor y si son parámetros de salida necesariamente su dirección (o sea indicar que son de salida dado que por defecto son de entrada).

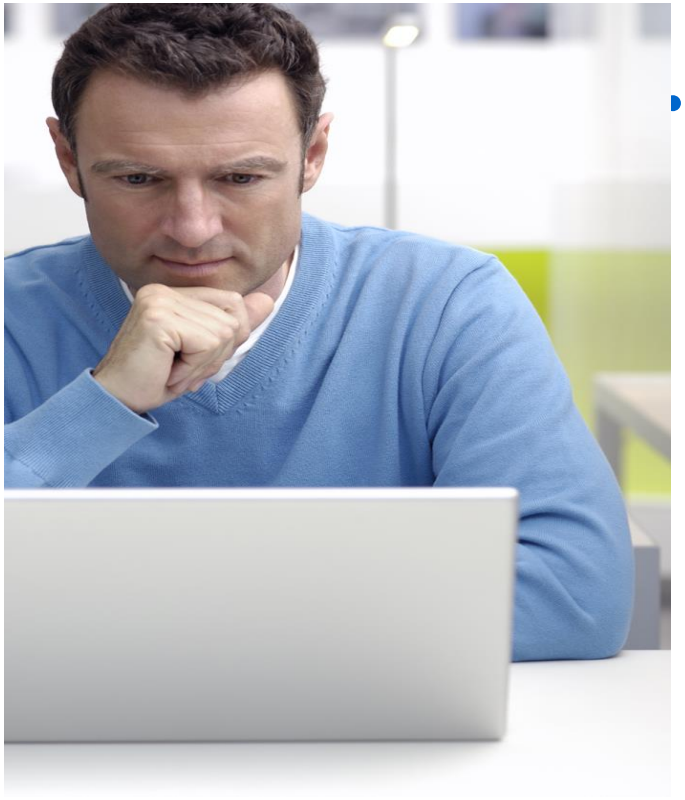
• Ejemplo de comando que ejecuta un SP con parámetro

```
private void btnVentasAnual_Click(object sender, EventArgs e)
{
    try
    {
        // preparo mi comando...
        cmd.Connection = cnx;
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.CommandText = "usp_VentaAnual";
        // agregamos el parametro...
        cmd.Parameters.Clear();
        cmd.Parameters.AddWithValue("@año", txtAño.Text.Trim());

        // Abrimos conexion y ejecutamos
        cnx.Open();
        MessageBox.Show("El total de ventas en el año " + txtAño.Text + " fue : " +  cmd.ExecuteScalar().ToString(), "Resultado");

    }
    catch (SqlException ex)
    {
        MessageBox.Show("Error : " + ex.Message);

    }
    finally
    {
        // finalmente si la conexion esta abierta la cerramos...
        if (cnx.State == ConnectionState.Open)
        {
            cnx.Close();
        }

    }
}
}
```

• Ejemplo de comando que ejecuta un SP con parámetro

```
try
{
    if (e.KeyChar == 13)
    {
        cmd.Connection = cnx;
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.CommandText = "usp_ConsultarCliente";
        cmd.Parameters.Clear();
        cmd.Parameters.AddWithValue("@vcod", txtcod.Text.Trim());

        cnx.Open();
        dtr = cmd.ExecuteReader();

        // Tiene filas el dtr??
        if (dtr.HasRows == true)
        {
            dtr.Read();
            lblRaz.Text = dtr["raz_soc_cli"].ToString();
            lblDir.Text = dtr["dir_cli"].ToString();
            lblTel.Text = dtr["tel_cli"].ToString();
            lblRuc.Text = dtr["ruc_cli"].ToString();
            lblDeuda.Text = Convert.ToSingle(dtr["deuda"]).ToString("###,###.00 soles");

        }
        else
        {
            lblRaz.Text = "";
            lblDir.Text = "";
            lblTel.Text = "";
            lblRuc.Text = "";
            lblDeuda.Text = "";
            throw new Exception("No existe cliente con el codigo ingresado.");
        }
        dtr.Close();

    }
}
catch (SqlException ex)
{
    MessageBox.Show("Error:" + ex.Message);
}
catch (Exception ex2)
{
    MessageBox.Show("Error:" + ex2.Message);
}
finally
{
    if (cnx.State == ConnectionState.Open)
    {
        cnx.Close();
    }
}
```

Demostración 2: Uso de la clase SQLCommand



- En esta demostración, aprenderemos a utilizar la clase SQLCommand en sus diferentes variantes de consulta, asi como aprender a manejar la clase SQL DataAdapter para empleo del entorno desconectado

Conclusiones de la sesión:

- Reconocer la importancia de interactuar con bases de datos desde aplicaciones Visual Studio
- Podemos optimizar la funcionalidad del aplicativo accediendo a orígenes de datos, como será en el mundo empresarial donde nos desenvolveremos a futuro.
- Enfocar el esfuerzo a profundizar nuestros conocimientos de SQL Server en lo referente a comandos (Select, Insert, Update, Delete, creación de Store Procedure, etc.)