

TÍTULO

DESARROLLO DE APLICACIONES I

Sesión Nro. 05


Objetivos de la sesión:

Al culminar la presente podrás:

- Conocer que es el mantenimiento de tablas base.
- Identificar los store procedure que participan en el proceso de mantenimiento.
- Construir una aplicación “N” capas, distinguiendo la responsabilidad de cada una de ellas.
- Implementar el mantenimiento de tablas base en el patrón “N” capas.



Temario

1. Definición de mantenimiento de tablas
 2. Los Store Procedure de mantenimiento
 3. Construyendo una aplicación en 3 Capas con Visual Studio
 4. Mantenimiento de tablas empleando clases
 5. Los formularios de mantenimiento
- 
- A decorative graphic in the bottom right corner consisting of overlapping blue and yellow diagonal stripes.

Tema Nro. 1:

Definición de mantenimiento de tablas (CRUD)



1.1 Que es el mantenimiento de una tabla ?

- Dentro de la programación con acceso a base de datos, se define así al proceso de agregar, modificar, eliminar , consultar y listar los datos de una tabla base o maestra. En ingles se le conoce con las iniciales de CRUD: Create, Read, Update and Delete
- Si bien es cierto que se podría llevar a cabo con comandos textuales, se sugiere que se empleen procedimientos almacenados para dichas tareas, dado que son mas rápidos y representan una buena practica en este tipo de operaciones



Tema Nro. 2:

Los Store Procedure de mantenimiento de tablas



2.1 Los Store Procedure de mantenimiento de tablas

- Se sugiere crear los 5 SP básicos. Si por ejemplo a la tabla que se aplicara el mantenimiento se denomina Tb_Vendedor los SP podrían denominar así:
 - Usp_InsertarVendedor
 - Usp_ActualizarVendedor
 - Usp_EliminarVendedor
 - Usp_ConsultarVendedor
 - Usp_ListarVendedor

dbo.Tb_Vendedor
Columns
Cod_ven (PK, nchar(3), not null)
Nom_ven (nvarchar(25), null)
Ape_ven (nvarchar(25), null)
Sue_ven (money, null)
Fec_ing (datetime, null)
Tip_ven (int, null)
Email_ven (varchar(50), null)
Cod_Supervisor (nchar(3), null)
Fec_Registro (datetime, null)
Usu_Registro (varchar(20), null)
Fec_Ult_Mod (datetime, null)
Usu_Ult_Mod (varchar(20), null)
Est_ven (int, null)

Nota: Según sea la necesidad o requerimientos pueden emplearse mas procedimientos almacenados.



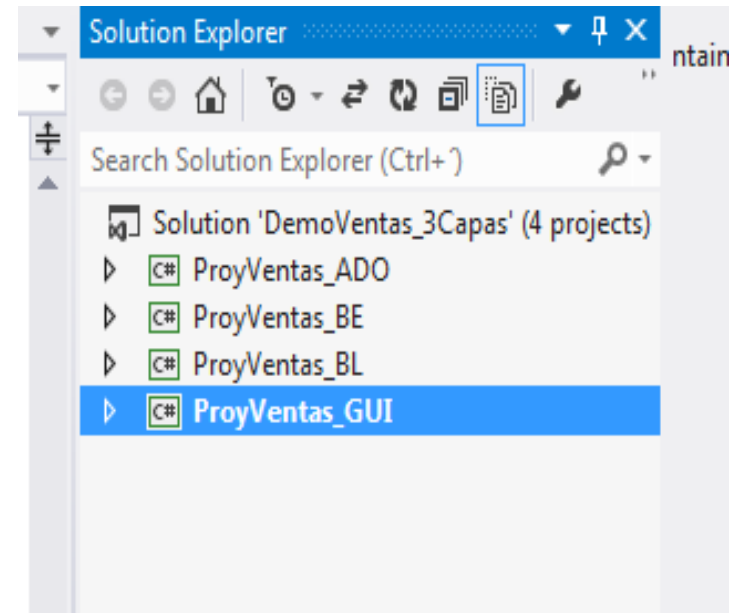
Tema Nro. 3: Construyendo una aplicación 3 Capas en Visual Studio



3.1 ¿Que es el patrón de desarrollo N – Capas?

Es un patrón que consiste en agrupar en una solución un conjunto de proyectos, cada uno de ellos relacionados (referenciados), donde si bien es cierto cada uno es independiente , colaboran entre si para obtener una aplicación robusta, rápida a adaptaciones de cambios y que pueda ser escalable.

Bajo este patrón la configuración mínima seria una de 3 capas.



3.2 ¿Cuales son las 3 Capas?

Bajo este concepto encontramos 3 capas funcionales dentro del aplicativo:

- a) Capa de Acceso a datos: Es una librería de clases que contiene todo lo concerniente al manejo de la conexión a la base de datos, para hacer consultas y/o actualizaciones.
- b) Capa de Negocios: Normalmente es una capa intermedia, que sirve de nexo entre la capa de datos y la capa de presentación. También es una librería de clases.
- c) Capa de Presentación: Es la que contiene las interfaces graficas con las que el usuario (formularios) , además de todas la lógica de presentación como validaciones de entrada y configuración de formatos de salida de información, que dará como resultado una aplicación amigable y de fácil manejo. Por lo general es un proyecto Windows , un sitio WEB o una aplicación Móvil.



3.2 ¿Cuales son las 3 Capas?

Cuando se habla de una aplicación en 3 capas, se establecen la vinculación de 4 proyectos. Nos preguntamos entonces :
¿Por qué sin son 3 capas hay 4 proyectos?.

Sucede que además de la capa de datos (proyecto de librería de clases), capa de negocios (proyecto de librería de clases), capa de presentación (proyecto Windows o WEB Asp. Net o Móvil) el cuarto proyecto es una librería de clases que contiene las estructuras de los datos que van hacer parte de las entidades de negocio. Las clases que se colocan en este proyecto solo contienen PROPIEDADES ,mas no implementan métodos, por lo que no se le considera una capa mas. A este proyecto se le conoce como de Entidades de Negocio.



Esquema del patrón de desarrollo en 3 capas



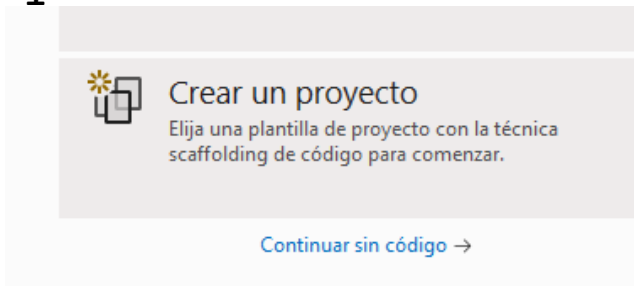


Demo: Cómo construir una Aplicación en 3 Capas (con BD Ventas)

Paso 1:

Inicie en VS una solución en blanco y denomínela DemoVentas_3Capas. Elija Ud. la carpeta donde la creara.

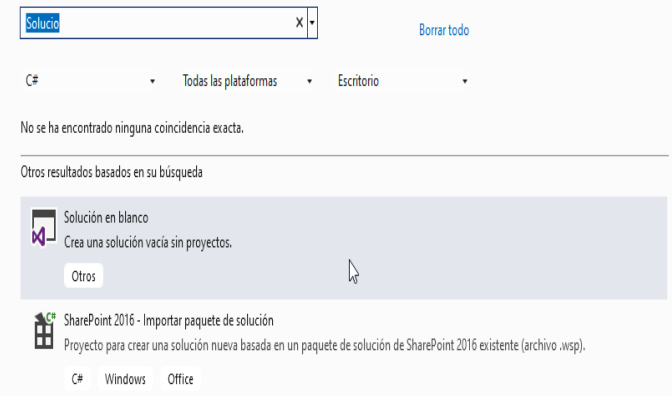
1



2 Crear un proyecto

Plantillas de proyecto recientes

- Aplicación de Windows Forms C#
- Biblioteca de servicios WCF C#
- Solución en blanco C#
- Aplicación web ASP.NET (.NET Framework) C#
- Aplicación de Windows Forms (.NET Framework) C#



3

Configure su nuevo proyecto

Solución en blanco Otros

Nombre de la solución

DemoVentas_3Capas

Ubicación

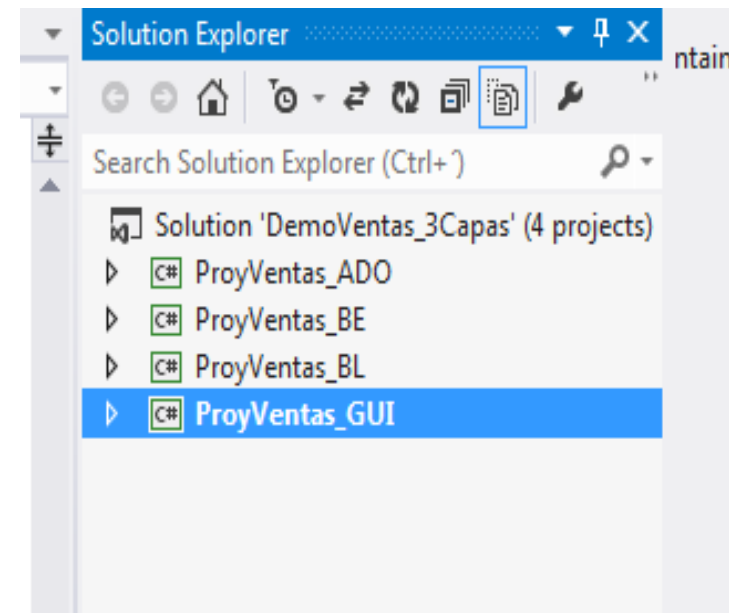
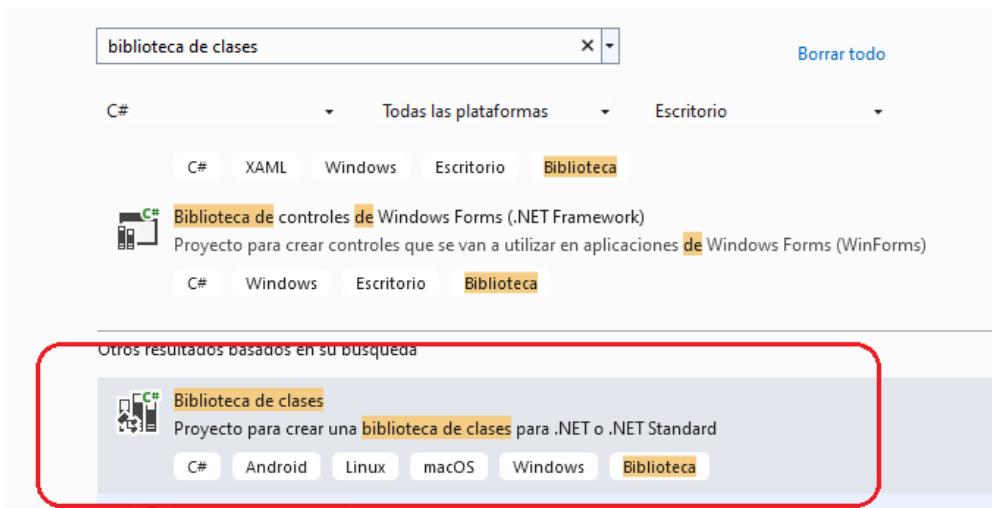
C:\ISIL_2021_2\Curso_Aplicaciones_\Net6.0\Sesion05\

Paso 2:

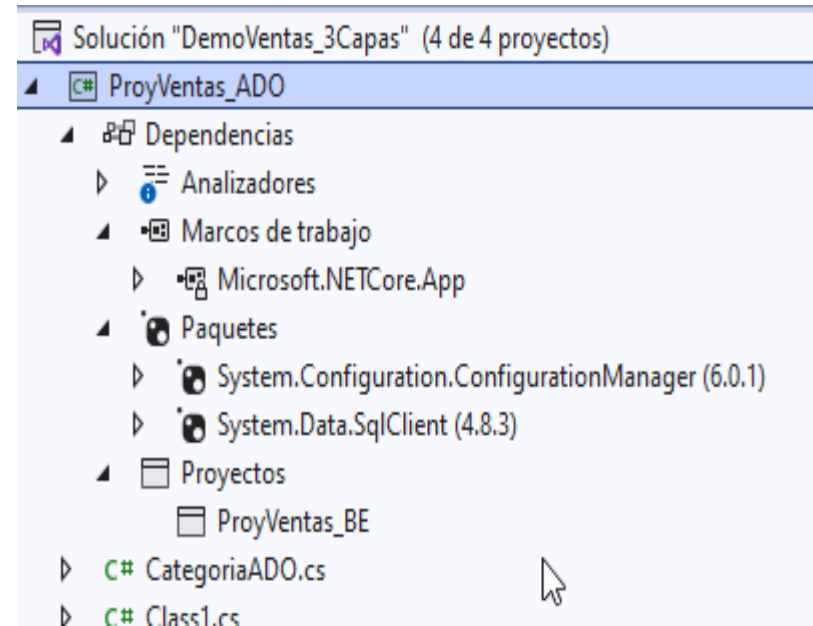
Agregue a la solución los siguientes 3 proyectos de tipo biblioteca de clases para .NET en C# con los siguientes nombres:

ProyVentas_ADO ,ProyVentas_BL y ProyVentas_BE

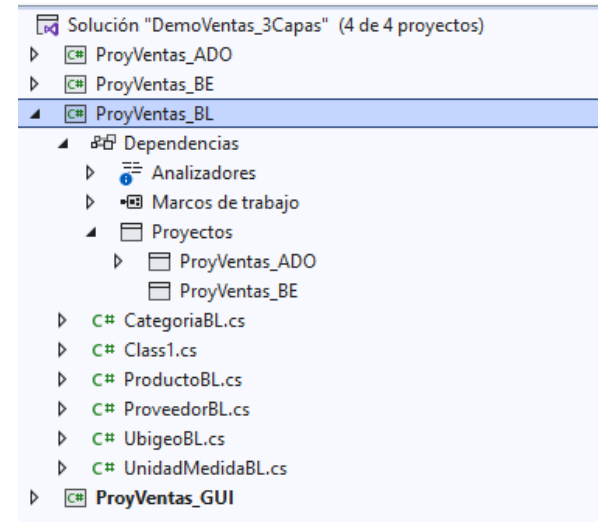
Por ultimo agregue la capa de presentación , que será un proyecto de tipo Windows .NET llamado ProyVentas_GUI



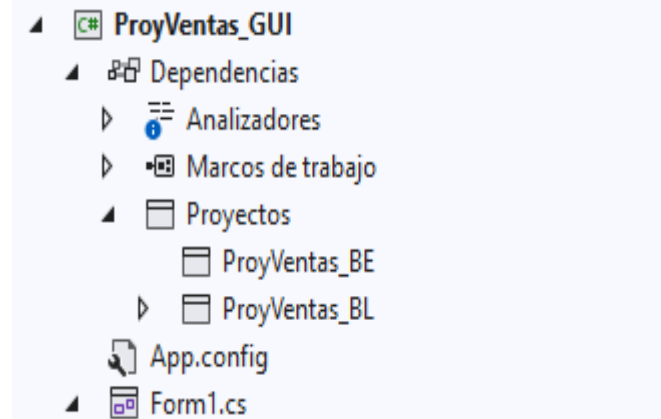
- Paso 3: En la capa de datos (proyecto **ProyVentas_ADO**) instale el **System.Configuration** y **System.Data.SqlClient** con el **Administrador de Paquetes Nuget**. Luego agregue una referencia al proyecto **ProyVentas_BE**



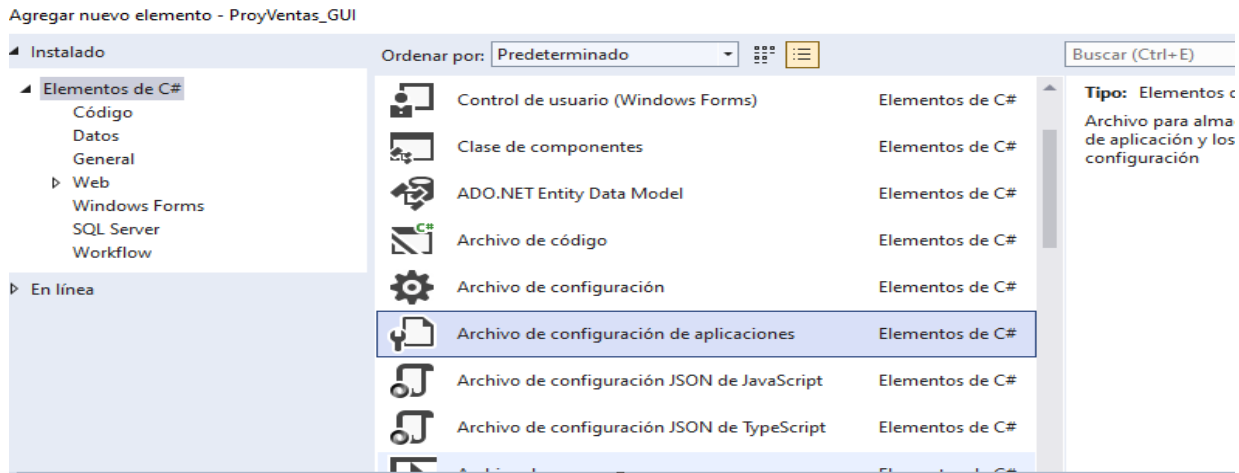
- Paso 4: En la capa de negocios (proyecto ProyVentas_BL) agregue 2 referencias a nivel de proyecto, con respecto al proyecto ProyVentas_ADO y al proyecto ProyVentas_BE



- Paso 5: En la capa de presentación (proyecto ProyVentas_GUI) agregue 2 referencias a nivel de proyecto, con respecto al proyecto ProyVentas_BL y al proyecto ProyVentas_BE



- Paso 6: En la capa de presentación (proyecto ProyVentas_GUI) agregue un archivo de configuración de aplicaciones (app.config) y agregue dentro de una sección ConnectionStrings una cadena de conexión a la BD VentasLeon de su servidor SQL Server local (o al que se quisiera conectar). La cadena de conexión se llamara “Ventas”, dentro del elemento Configuration tal como se muestra a continuación



```
<configuration>
  <connectionStrings>
    <add name="Ventas" connectionString="Data Source=.;Initial Catalog=VentasLeon;Integrated Security=True"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
```

- **Paso 7: Agregue las clases que su instructor le indicara como parte del laboratorio. Cada clase tiene un nombre que le permitirá deducir en que proyecto incluirla.**
- **Paso 8: Compile la solución y verificar que no arroje ningún error**



Recuerde...

- Para realizar de manera correcta el mantenimiento de una tabla maestra bajo el patrón “N” capas este seguro de llevar a cabo estos pasos:
 - Crear los procedimientos almacenados de mantenimiento: Insertar, Actualizar, Eliminar (si es necesario), Listar y Consultar. Es posible que se requieran mas métodos según sea el caso.
 - Crear la clase de entidad de negocio (BE)
 - Crear la clase de acceso a datos (ADO)
 - Crear la clase de lógica de negocios (BL)
 - Crear los formularios de mantenimiento .



Recuerde...

En la capa de datos se agregara una clase que permita obtener la cadena de conexión establecida en el App.Config en un método llamado GetCnx. La clase tiene como nombre ConexionADO y esta en el material compartido.

```
namespace ProyVentas_ADO
{
    class ConexionADO
    {
        public string GetCnx()
        {
            string strCnx =
                ConfigurationManager.ConnectionStrings["Ventas"].ConnectionString;
            if (object.ReferenceEquals(strCnx, string.Empty))
            {
                return string.Empty;
            }
            else
            {
                return strCnx;
            }
        }
    }
}
```



LABORATORIO DIRIGIDO

Realizar junto a su instructor el mantenimiento completo de la tabla Tb_Producto de acuerdo a los conceptos vertidos en la clase de hoy



Pautas del Laboratorio dirigido: Construyendo las clases de mantenimiento: Tabla Tb_Producto



4.1 La Entidad de Negocios

```
namespace ProyVentas_BE
{
    public class ProductoBE
    {
        // Definimos la entidad de negocio Producto, con todas las propiedades de acuerdo a la estructura
        // de la tabla Tb_Producto
        public String Cod_pro { get; set; }
        public String Des_pro { get; set; }
        public Single Pre_pro { get; set; }
        public Int16 Stk_act { get; set; }
        public Int16 Stk_min { get; set; }
        public Int16 Id_UM { get; set; }
        public Int16 Id_Cat { get; set; }
        public Int16 Importado { get; set; }
        public DateTime Fec_Registro { get; set; }
        public String Usu_Registro { get; set; }
        public DateTime Fec_Ult_Mod { get; set; }
        public String Usu_Ult_Mod { get; set; }
        public Int16 Est_pro { get; set; }
    }
}
```



4.2 La Capa de datos: ListarProducto

```
public class ProductoADO
{
    // Instancias.....
    ConexionADO MiConexion = new ConexionADO();
    SqlConnection cnx = new SqlConnection();
    SqlCommand cmd = new SqlCommand();
    SqlDataReader dtr;

    public DataTable ListarProducto()
    {
        DataSet dts = new DataSet();
        cnx.ConnectionString = MiConexion.GetCnx();
        cmd.Connection = cnx;
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.CommandText = "usp_ListarProducto";
        try
        {
            cmd.Parameters.Clear();
            SqlDataAdapter ada = new SqlDataAdapter(cmd);
            ada.Fill(dts, "Productos");
            return dts.Tables["Productos"];
        }
        catch (SqlException ex)
        {
            throw new Exception(ex.Message);
        }
    }
}
```



4.5 La Capa de datos: ConsultarProducto

```
public ProductoBE ConsultarProducto (String strCodigo)
{
    ProductoBE objProductoBE = new ProductoBE();
    cnx.ConnectionString = MiConexion.GetCnx();
    cmd.Connection = cnx;
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.CommandText = "usp_ConsultarProducto";
    try
    {
        // Agregamos el parametro...
        cmd.Parameters.Clear();
        cmd.Parameters.AddWithValue("@vcod", strCodigo);
        cnx.Open();
        dtr = cmd.ExecuteReader();
        if (dtr.HasRows == true)
        {
            dtr.Read();
            objProductoBE.Cod_pro = dtr["Cod_pro"].ToString();
            objProductoBE.Des_pro = dtr["Des_pro"].ToString();
            objProductoBE.Pre_pro = Convert.ToSingle (dtr["Pre_pro"]);
            objProductoBE.Stk_act = Convert.ToInt16(dtr["Stk_act"]);
            objProductoBE.Stk_min = Convert.ToInt16(dtr["Stk_min"]);
            objProductoBE.Id_UM = Convert.ToInt16(dtr["Id_UM"]);
            objProductoBE.Id_Cat = Convert.ToInt16(dtr["Id_Cat"]);
            objProductoBE.Importado = Convert.ToInt16(dtr["Importado"]);
            objProductoBE.Est_pro = Convert.ToInt16(dtr["Est_pro"]);
        }
        dtr.Close();
        return objProductoBE;
    }
    catch (SqlException ex)
    {
        throw new Exception(ex.Message);
    }
    finally
    {
        if (cnx.State == ConnectionState.Open)

```



4.6 La Capa de datos: InsertarProducto

```
public Boolean InsertarProducto(ProductoBE objProductoBE)
{
    try
    {
        cnx.ConnectionString = MiConexion.GetCnx();
        cmd.Connection = cnx;
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.CommandText = "usp_InsertarProducto";
        //Agregamos parametros
        cmd.Parameters.Clear();
        cmd.Parameters.AddWithValue("@vdes", objProductoBE.Des_pro);
        cmd.Parameters.AddWithValue("@vpre", objProductoBE.Pre_pro);
        cmd.Parameters.AddWithValue("@vstka", objProductoBE.Stk_act);
        cmd.Parameters.AddWithValue("@vstkm", objProductoBE.Stk_min);
        cmd.Parameters.AddWithValue("@vld_UM", objProductoBE.Id_UM);
        cmd.Parameters.AddWithValue("@vld_Cat", objProductoBE.Id_Cat);
        cmd.Parameters.AddWithValue("@vImp", objProductoBE.Importado);
        cmd.Parameters.AddWithValue("@vUsu_Registro", objProductoBE.Usu_Registro);
        cmd.Parameters.AddWithValue("@vEst_pro", objProductoBE.Est_pro);
        cnx.Open();
        cmd.ExecuteNonQuery();
        return true;
    }
    catch (SQLException x)
    {
        throw new Exception(x.Message);
        return false;
    }
    finally
    {
        if (cnx.State == ConnectionState.Open)
        {
            cnx.Close();
        }
    }
}
```



4.3 La Capa de datos: ActualizarProducto

```
public Boolean ActualizarProducto(ProductoBE objProductoBE)
{
    try
    {
        cnx.ConnectionString = MiConexion.GetCnx();
        cmd.Connection = cnx;
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.CommandText = "usp_ActualizarProducto";
        //Agregamos parametros
        cmd.Parameters.Clear();
        cmd.Parameters.AddWithValue("@vcod", objProductoBE.Cod_pro);
        cmd.Parameters.AddWithValue("@vdes", objProductoBE.Des_pro);
        cmd.Parameters.AddWithValue("@vpre", objProductoBE.Pre_pro);
        cmd.Parameters.AddWithValue("@vstka", objProductoBE.Stk_act);
        cmd.Parameters.AddWithValue("@vstkm", objProductoBE.Stk_min);
        cmd.Parameters.AddWithValue("@vld_UM", objProductoBE.Id_UM);
        cmd.Parameters.AddWithValue("@vld_Cat", objProductoBE.Id_Cat);
        cmd.Parameters.AddWithValue("@vImp", objProductoBE.Importado);
        cmd.Parameters.AddWithValue("@vUsu_Ult_Mod", objProductoBE.Usu_Ult_Mod);
        cmd.Parameters.AddWithValue("@vEst_pro", objProductoBE.Est_pro);
        cnx.Open();
        cmd.ExecuteNonQuery();
        return true;
    }
    catch (SqlException x)
    {
        throw new Exception(x.Message);
        return false;
    }
    finally
    {
        if (cnx.State == ConnectionState.Open)
        {
            cnx.Close();
        }
    }
}
```



4.4 La Capa de datos: EliminarProducto(*)

```
public Boolean EliminarProducto(String strCodigo)
{
    cnx.ConnectionString = MiConexion.GetCnx();
    cmd.Connection = cnx;
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.CommandText = "usp_EliminarProducto";

    try
    {
        //Agregamos parametros
        cmd.Parameters.Clear();
        cmd.Parameters.AddWithValue("@vcod", strCodigo);
        cnx.Open();
        cmd.ExecuteNonQuery();
        return true;
    }
    catch (SqlException x)
    {
        throw new Exception(x.Message);
        return false;
    }
    finally
    {
        if (cnx.State == ConnectionState.Open)
        {
            cnx.Close();
        }
    }
}
```

(*) No se va a implementar



4.5 La Capa de datos: Clases para el listado de Categorías y Unidades de Medida

```
public class CategoriaADO
{
    // Instancias.....
    ConexionADO MiConexion = new ConexionADO();
    SqlConnection cnx = new SqlConnection();
    SqlCommand cmd = new SqlCommand();
    SqlDataReader dtr;

    public DataTable ListarCategoria()
    {
        DataSet dts = new DataSet();
        cnx.ConnectionString = MiConexion.GetCnx();
        cmd.Connection = cnx;
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.CommandText = "usp_ListarCategoria";
        try
        {
            cmd.Parameters.Clear();
            SqlDataAdapter ada = new SqlDataAdapter(cmd);
            ada.Fill(dts, "Categorias");
            return dts.Tables["Categorias"];
        }
        catch (SQLException ex)
        {
            throw new Exception(ex.Message);
        }
    }
}
```

```
public class UnidadMedidaADO
{
    // Instancias.....
    ConexionADO MiConexion = new ConexionADO();
    SqlConnection cnx = new SqlConnection();
    SqlCommand cmd = new SqlCommand();
    SqlDataReader dtr;

    public DataTable ListarUM()
    {
        DataSet dts = new DataSet();
        cnx.ConnectionString = MiConexion.GetCnx();
        cmd.Connection = cnx;
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.CommandText = "usp_ListarUM";
        try
        {
            cmd.Parameters.Clear();
            SqlDataAdapter ada = new SqlDataAdapter(cmd);
            ada.Fill(dts, "Unidades");
            return dts.Tables["Unidades"];
        }
        catch (SQLException ex)
        {
            throw new Exception(ex.Message);
        }
    }
}
```



4.6 La Capa de negocios (invoca a cada método de la capa de datos mediante una instancia de la clase ProductoADO llamada objProductoADO)

```
// Agregar...
using ProyVentas_ADO;
using ProyVentas_BE;

namespace ProyVentas_BL
{
    public class ProductoBL
    {
        ProductoADO objProductoADO = new ProductoADO();

        public DataTable ListarProducto()
        {
            return objProductoADO .ListarProducto ();
        }
        public ProductoBE ConsultarProducto(String strCodigo)
        {
            return objProductoADO.ConsultarProducto (strCodigo );
        }

        public Boolean InsertarProducto(ProductoBE objProductoBE)
        {
            return objProductoADO.InsertarProducto(objProductoBE);
        }
        public Boolean ActualizarProducto(ProductoBE objProductoBE)
        {
            return objProductoADO.ActualizarProducto(objProductoBE);
        }
        public Boolean EliminarProducto(String strCodigo)
        {
            return objProductoADO.EliminarProducto(strCodigo);
        }
    }
}
```



4.7 La Capa de negocios (invoca a cada método de la capa de datos mediante su respectiva instancia de la clase CategoriaADO y UnidadMedidaADO)

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
// Agregar
using ProyVentas_ADO;
namespace ProyVentas_BL
{
    public class CategoriaBL
    {
        CategoriaADO objCategoriaADO = new CategoriaADO();

        public DataTable ListarCategoria()
        {
            return objCategoriaADO.ListarCategoria();
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
// Agregar...
using ProyVentas_ADO;
namespace ProyVentas_BL
{
    public class UnidadMedidaBL
    {
        UnidadMedidaADO objUnidadMedidaADO = new UnidadMedidaADO();

        public DataTable ListarUM()
        {
            return objUnidadMedidaADO.ListarUM();
        }
    }
}
```



Los formularios de mantenimiento para la tabla Tb_Producto



Los formularios de mantenimiento

- Si bien es cierto que se puede llevar a cabo el mantenimiento en un solo formulario, tanto para el desarrollador, y sobre todo para el usuario final es mas amigable y mas claro codificarlo en 3 formularios.
- Para el caso de la tabla Tb_Producto lo haríamos así:



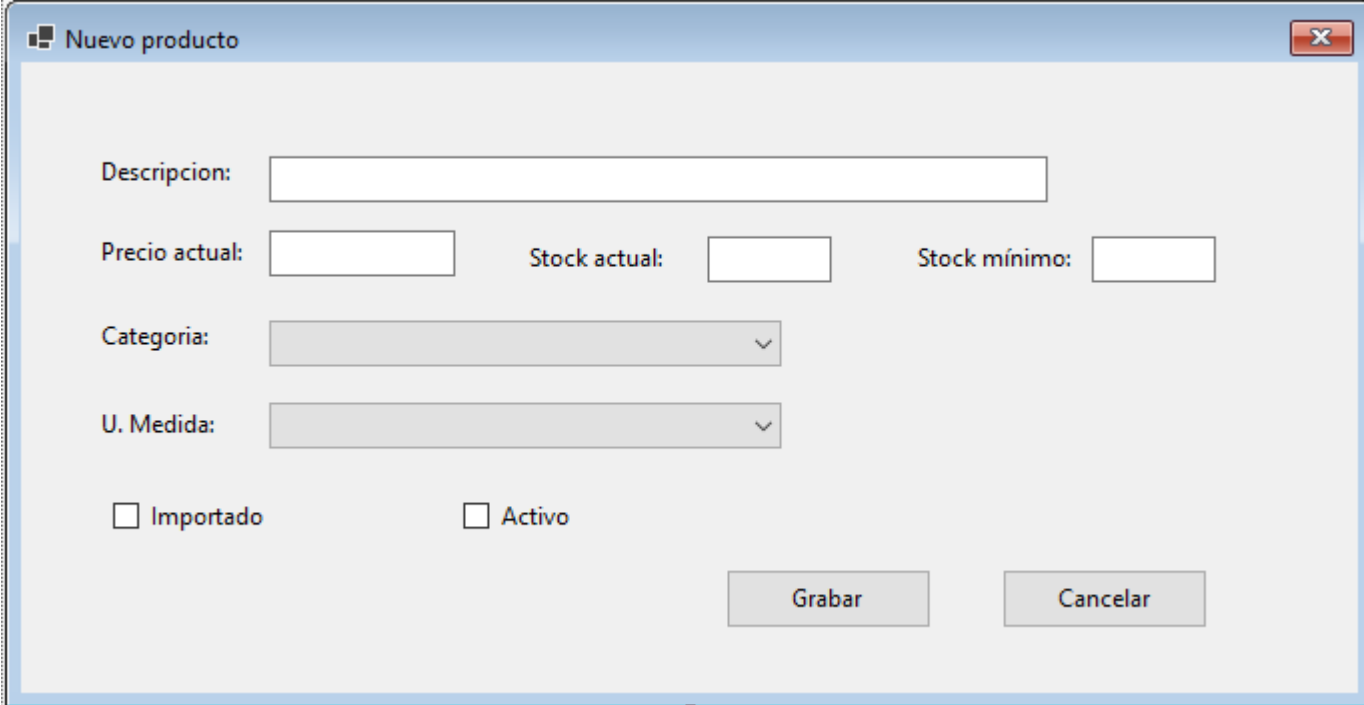
ProductoMan01

- Que se encargara de mostrar el listado de todos los registros en un datagridview y desde el cual se accederán a las demás operaciones del mantenimiento. Además contiene en la parte superior una caja de texto para el filtrado por descripción.

The screenshot shows a Windows-style application window titled "Mantenimiento de Productos". At the top, there is a label "Ingresa filtro por descripcion:" followed by a text input field. Below this is a large, empty gray rectangular area representing a data grid. At the bottom right, there is a label "Registros:" followed by a text input field. Along the bottom edge, there are four buttons: "Agregar", "Actualizar", "Eliminar", and "Cerrar". A mouse cursor is hovering over the "Eliminar" button.

ProductoMan02

- Que se encarga de permitir el ingreso de un nuevo registro. Nótese que no se ingresa el código dado que este se genera en el Store Procedure de inserción. Se harán validaciones antes de grabar.



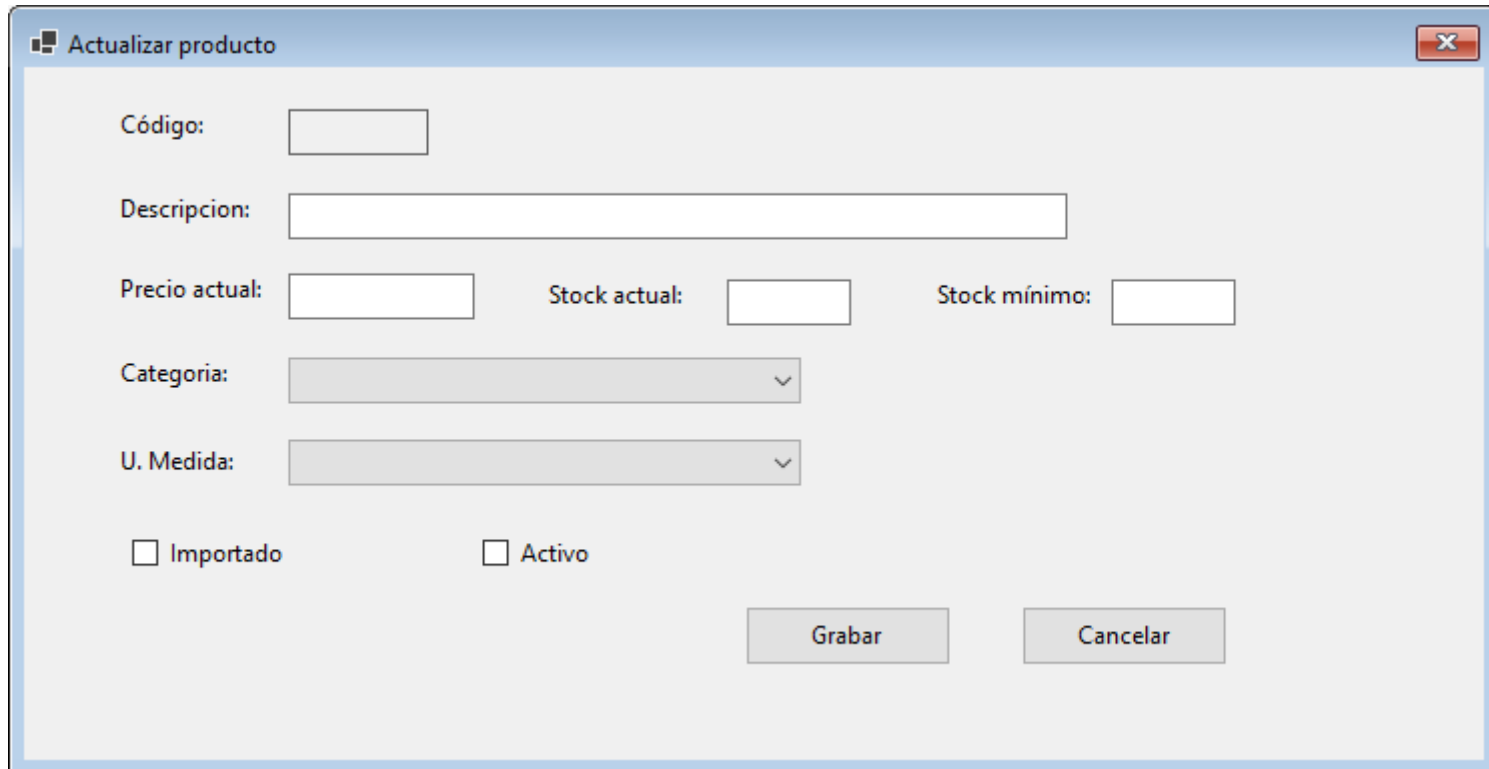
The image shows a Windows-style dialog box titled "Nuevo producto". It contains the following fields and controls:

- Descripcion:** A single-line text input field.
- Precio actual:** A single-line numeric input field.
- Stock actual:** A single-line numeric input field.
- Stock mínimo:** A single-line numeric input field.
- Categoria:** A dropdown menu.
- U. Medida:** A dropdown menu.
- Importado:** A checkbox.
- Activo:** A checkbox.
- Grabar:** A button.
- Cancelar:** A button.



ProductoMan03

- Que se encarga de la actualización del registro seleccionado en el datagridview del formulario ProductoMan01 al momento de hacer clic en el botón Actualizar o doble clic sobre el registro. También se harán validaciones antes de grabar.



Actualizar producto

Código:

Descripción:

Precio actual: Stock actual: Stock mínimo:

Categoría:

U. Medida:

☐ Importado ☐ Activo

Grabar Cancelar

LABORATORIO PROPUESTO

Realizar junto a su instructor el mantenimiento completo de la tabla Tb_Proveedor de acuerdo a los conceptos vertidos en la clase de hoy



Conclusiones de la sesión:

- El patrón de desarrollo basado en capas garantiza una mas sencilla adecuación a cambios, dado que estos los haremos en la capa respectiva, sin causar mayores efectos en el resto de la aplicación.
- Es importante para el proceso de CRUD verificar que estén los SP que definen cada una de las acciones.

