TP2 - Laboratorio 05

1. Reemplazar la capa de datos en memoria por una real

Objetivos

Modificar la capa de datos para que trabaje con una base de datos real en lugar de con datos en memoria.

Duración Aproximada

30 minutos

Pasos

1. Ir a la carpeta C:\Net\Labs y crear la carpeta TP2L04.

2. Copiar la carpeta TP2 que se encuentra en C:\Net\Labs\TP2L04\ a C:\Net\Labs\TP2L05\
   De esta forma modificaremos esta copia manteniendo la versión anterior intacta.

3. Descomprimir el archivo que se encuentra dentro de tp2_net_v1.zip que se adjunta con este trabajo. Dentro del mismo se encuentra el backup de la base de datos de MS Sql Server 2005 para el TP.

4. Restaurar el backup como siguiendo los mismos pasos que usamos para restaurar la base de datos northwind y dar permisos al usuario net en el instructivo de instalación para Sql Server 2005

5. Hacemos clic con el botón derecho sobre el proyecto UI.Desktop agregamos un nuevo elemento y elegimos Archivo de Configuración (Application Configuration File). Lo llamaremos App.Config

6. Modificamos el App.Config par que se vea así:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <connectionStrings>
      <add name="ConnStringLocal"
       providerName="System.Data.SqlClient"
       connectionString="Server=localhost;Database=tp2_net;Integrated
   Security=false; User=net; Password=net;"/>
      <add name="ConnStringExpress"
       providerName="System.Data.SqlClient"
       connectionString="Server=localhost\SQLEXPRESS;Database=tps_net;
   Integrated Security=false; User=net; Password=net;"/>
      <add name="ConnStringServerISI"
       providerName="System.Data.SqlClient"
       connectionString="Server=serverisi;   Database=tp2_net;   Integrated
   Security=false; User=net; Password=net;"/>
   </connectionStrings>
</configuration>
```

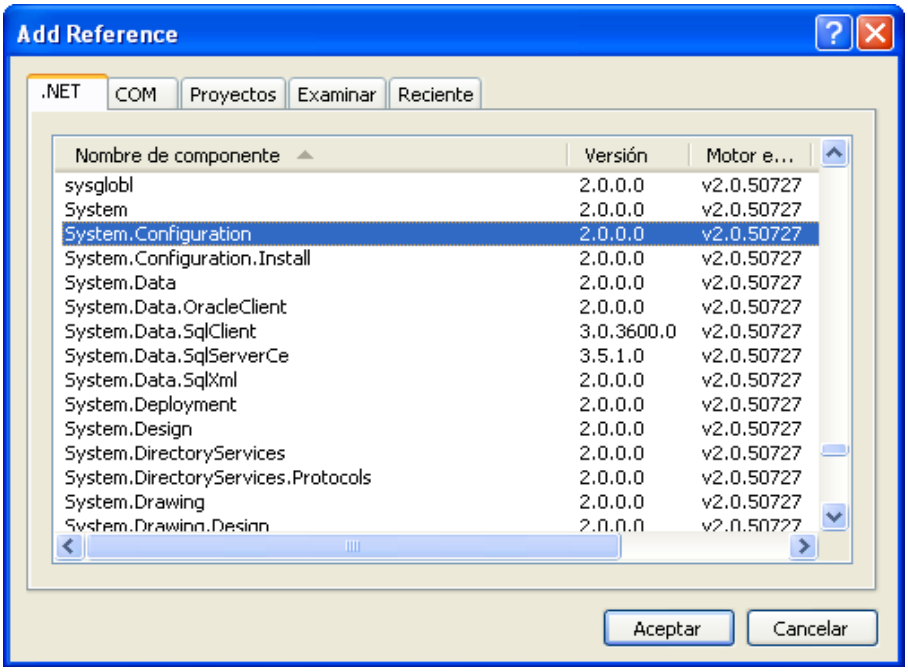
7. En el explorador de soluciones, dentro del proyecto Data.Database, en la clase Adapter agregamos las siguientes líneas.

```
//Clave por defecto a utlizar para la cadena de conexion
const string consKeyDefaultCnnString = "ConnStringLocal";
```

Si estamos trabajando con un MS Sql Server con una instancia sin nombre instalado en la misma PC en la que estamos desarrollando usaremos ConnStringLocal, en caso que hayamos instalado un Sql Express sin modificar el nombre de instancia usaremos ConnStringExpress y si estamos usando el serverisi usaremos ConnStringServerISI. Estos los iremos cambiando en la medida que estemos trabajando en otros entornos

8. Hacemos clic con botón derecho sobre el proyecto Data.Database y luego hacemos clic en Agregar referencia...

9. En la ventana que aparece buscamos y agregamos referencia a System.Configuration



10. En la clase Adapter también agregaremos una Propiedad sqlConn de tipo SqlConnection

11. En el método OpenConnection guardaremos el valor del connectionstring en una variable invocando a
`ConfigurationManager.ConnectionStrings[consKeyDefaultCnnString].ConnectionString`

12. Luego instanciamos sqlConn como una nueva SqlConnection usando el connection string que acabamos de recuperar y la abrimos con sqlConn.Open()

13. En el método CloseConnection cerramos la conexión con sqlConn.Close() y luego para liberar la memoria usada escribimos sqlConn=null;

14. En la clase UsuarioAdapter agregamos los siguientes using para poder tener acceso a las clases de .net para realizar consultas sql a MS Sql Server

```
using System.Data;
using System.Data.SqlClient;
```

15. En la clase UsuarioAdapter, en el método GetAll cambiamos el código por el siguiente

Nota: No es necesario que copie los comentarios

```
public List<Usuario> GetAll()
{

    //instanciamos el objeto lista a retornar
    List<Usuario> usuarios = new List<Usuario>();

    //abrimos la conexión a la base de datos con el método que ceamos antes
    this.OpenConnection();

    /*
     * creamos un objeto SqlCommand que será la sentencia SQL
     * que vamos a ejecutar contra la base de datos
     * (los datos de la BD usuario, contraseña, servidor, etc.
     *  están en el connection string)
     */
    SqlCommand cmdUsuarios = new SqlCommand("select * from usuarios",sqlConn);

    /*
     * instanciamos un objeto DataReader que será
     * el que recuperará los datos de la BD
     */
    SqlDataReader drUsuarios = cmdUsuarios.ExecuteReader();
    /*
     * Read() lee una fila de las devueltas por el comando sql
     * carga los datos en drUsuarios para poder accederlos,
     * devuelve verdadero mientras haya podido leer datos
     * y avanza a la fila siguiente para el próximo read
     */
    while (drUsuarios.Read())
    {
        /*
         * creamos un objeto Usuario de la capa de entidades para copiar
         * los datos de la fila del DataReader al objeto de entidades
         */
        Usuario usr = new Usuario();

        //ahora copiamos los datos de la fila al objeto
        usr.ID = (int)drUsuarios["id_usuario"];
        usr.NombreUsuario = (string)drUsuarios["nombre_usuario"];
        usr.Clave = (string)drUsuarios["clave"];
        usr.Habilitado = (bool)drUsuarios["habilitado"];
        usr.Nombre = (string)drUsuarios["nombre"];
        usr.Apellido = (string)drUsuarios["apellido"];
        usr.EMail = (string)drUsuarios["email"];

        //agregamos el objeto con datos a la lista que devolveremos
        usuarios.Add(usr);
    }

    //cerramos la el DataReader y la conexión a la BD
    drUsuarios.Close();
    this.CloseConnection();

    //devolvemos el objeto
    return usuarios;
}
```

16. A continuación procedemos a ejecutar nuestro sistema y probar que funcione.
    Nota: Antes de hacerlo chequee que haya datos cargados en la base de datos en la tabla usuarios, caso contrario no verá ningún dato

17. A continuación agregamos una estructura try catch finally al método para manejar cualquier error. El bloque try debe ir desde justo antes de abrir la conexión hasta justo después de cerrar el DataReader, el catch deberá ser así:

```
catch (Exception Ex)
{
  Exception ExcepcionManejada =
         new Exception("Error al recuperar lista de usuarios", Ex);
   throw ExcepcionManejada;
}
```

    Aquí creamos una nueva excepción donde nosotros podemos asignarle una descripción más representativa y utilizar la excepción que capturamos como una Excepción Interna, lo que nos permite conservar toda esa información y luego con el throw enviamos la excepción que nosotros creamos al método que invocó al GetAll.
    Queda pendiente al lector:
    - Agregar un bloque try catch finally en la capa de negocio para capturar las excepciones y lanzarlas con throw a la capa de presentación.
    - Agregar un bloque try catch finally en la capa de presentación que capture las excepciones y las informe al usuario usando un MessageBox

    El bloque finally debe contener la sentencia this.CloseConnection();

18. Modificamos el método GetOne de manera similar, con la única diferencia que la sentencia sql

```
public Business.Entities.Usuario GetOne(int ID)
{
    Usuario usr = new Usuario();
    try
    {
        this.OpenConnection();
        SqlCommand cmdUsuarios = new SqlCommand("Select * from usuarios where id_usuario @id", SqlConn);
        cmdUsuarios.Parameters.Add("@id", SqlDbType.Int).Value = ID;
        SqlDataReader drUsuarios = cmdUsuarios.ExecuteReader();
        if (drUsuarios.Read())
        {
            usr.ID = (int)drUsuarios["id_usuario"];
            usr.NombreUsuario = (string)drUsuarios["nombre_usuario"];
            usr.Clave = (string)drUsuarios["clave"];
            usr.Habilitado = (bool)drUsuarios["habilitado"];
            usr.Nombre = (string)drUsuarios["nombre"];
            usr.Apellido = (string)drUsuarios["apellido"];
            usr.EMail = (string)drUsuarios["email"];

        }
        drUsuarios.Close();
    }
    catch (Exception Ex)
    {
        Exception ExcepcionManejada = new Exception("Error al recuperar datos de usuario", Ex);
        throw ExcepcionManejada;
    }
    finally
    {
        this.CloseConnection();
    }
    return usr;
}
```

El método es similar al anterior, las diferencias son:

- En la sentencia Sql resaltada con el rectángulo rojo, la nueva sentencia tiene un parámetro @id y en la línea siguiente lo definimos y le agregamos un valor.
- En lugar de una lista de usuarios devolvemos un objeto usuario ya que la sentencia debería devolver sólo uno
- En lugar de un while usamos un if ya que solo debemos leer una fila

19. A continuación modificaremos el método Delete de la siguiente forma

```
public void Delete(int ID)
{
    try
    {
        //abrimos la conexión
        this.OpenConnection();

        //creamos la sentencia sql y asignamos un valor al parámetro
        SqlCommand cmdDelete =
            new SqlCommand("delete usuarios where id_usuario=@id", sqlConn);
        cmdDelete.Parameters.Add("@id", SqlDbType.Int).Value = ID;

        //ejecutamos la sentencia sql
        cmdDelete.ExecuteNonQuery();
    }
    catch (Exception Ex)
    {
        Exception ExcepcionManejada =
            new Exception("Error al eliminar usuario", Ex);
        throw ExcepcionManejada;
    }
    finally
    {
        this.CloseConnection();
    }
}
```

20. Luego creamos los métodos Update e Insert de manera similar

```
protected void Update(Usuario usuario)
{
    try
    {
        this.OpenConnection();
        SqlCommand cmdSave = new SqlCommand(
         "UPDATE usuarios SET nombre_usuario = @nombre_usuario, clave = @clave, " +
         "habilitado = @habilitado, nombre = @nombre, apellido = @apellido, email = @email " +
         "WHERE id_usuario=@id", sqlConn);

        cmdSave.Parameters.Add("@id", SqlDbType.Int).Value = usuario.ID;
        cmdSave.Parameters.Add("@nombre_usuario", SqlDbType.VarChar, 50).Value = usuario.NombreUsuario;
        cmdSave.Parameters.Add("@clave", SqlDbType.VarChar, 50).Value = usuario.Clave;
        cmdSave.Parameters.Add("@habilitado", SqlDbType.Bit).Value = usuario.Habilitado;
        cmdSave.Parameters.Add("@nombre", SqlDbType.VarChar, 50).Value = usuario.Nombre;
        cmdSave.Parameters.Add("@apellido", SqlDbType.VarChar, 50).Value = usuario.Apellido;
        cmdSave.Parameters.Add("@email", SqlDbType.VarChar, 50).Value = usuario.EMail;
        cmdSave.ExecuteNonQuery();
    }
    catch (Exception Ex)
    {
        Exception ExcepcionManejada =
            new Exception("Error al modificar datos del usuario", Ex);
        throw ExcepcionManejada;
    }
    finally
    {
        this.CloseConnection();
    }
}
```

```
protected void Insert(Usuario usuario)
{
    try
    {
        this.OpenConnection();
        SqlCommand cmdSave = new SqlCommand(
         "insert into usuarios(nombre_usuario,clave,habilitado,nombre,apellido,email) " +
         "values(@nombre_usuario, @clave,@habilitado,@nombre,@apellido,@email) " +
         "select @@identity", //esta línea es para recuperar el ID que asignó el sql automáticamente
         sqlConn);
        cmdSave.Parameters.Add("@nombre_usuario", SqlDbType.VarChar, 50).Value = usuario.NombreUsuario;
        cmdSave.Parameters.Add("@clave", SqlDbType.VarChar, 50).Value = usuario.Clave;
        cmdSave.Parameters.Add("@habilitado", SqlDbType.Bit).Value = usuario.Habilitado;
        cmdSave.Parameters.Add("@nombre", SqlDbType.VarChar, 50).Value = usuario.Nombre;
        cmdSave.Parameters.Add("@apellido", SqlDbType.VarChar, 50).Value = usuario.Apellido;
        cmdSave.Parameters.Add("@email", SqlDbType.VarChar, 50).Value = usuario.EMail;
        usuario.ID = Decimal.ToInt32((decimal)cmdSave.ExecuteScalar());
        //así se obtiene el ID que asignó al BD automáticamente
    }
    catch (Exception Ex)
    {
        Exception ExcepcionManejada =
            new Exception("Error al crear usuario", Ex);
        throw ExcepcionManejada;
    }
    finally
    {
        this.CloseConnection();
    }
}
```

21. Finalmente modificamos el método Save como se ve aquí

```
public void Save(Usuario usuario)
{
    if (usuario.State == BusinessEntity.States.Deleted)
    {
        this.Delete(usuario.ID);
    }
    else if (usuario.State == BusinessEntity.States.New)
    {
        this.Insert(usuario);
    }
    else if (usuario.State == BusinessEntity.States.Modified)
    {
        this.Update(usuario);
    }
    usuario.State = BusinessEntity.States.Unmodified;
}
```

22. Ejecutamos y probamos que funcione correctamente

Importante: Hemos modificado completamente la capa de datos cambiando el origen de datos de datos volátiles en memoria por datos en una base de datos y no fue necesario modificar ninguna de las otras capas. Aquí es donde reside la principal fortaleza de la programación en capas, la capacidad de alterar una de las capas en mayor o menor medida pero mientras mantengamos la misma interfaz no es necesario modificar nada de las otras capas

Fecha	Versión	Autor	Descripción
25/09/2011	1	AM	Versión inicial