Capítulo

4

DIPLOMATURA EN PROGRAMACION EN .NET

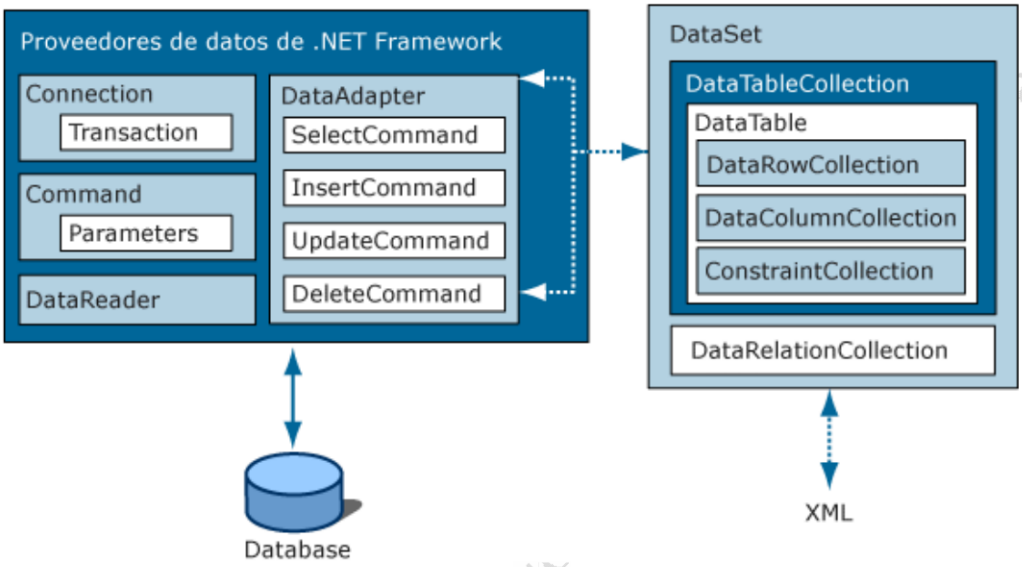
Proyecto Ventas

Introducción a ADO .Net

Conectándose a través de ADO .Net

Revisión detallada

Arquitectura de ADO .Net

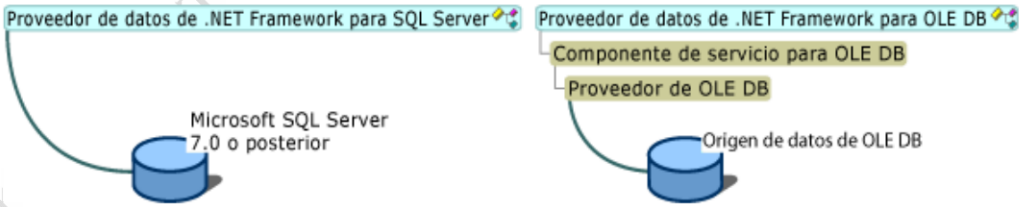


Objetos principales

Objeto	Descripción
Connection	Establece una conexión a un origen de datos determinado. La clase base para todos los objetos Connection es DbConnection.
Command	Ejecuta un comando en un origen de datos. Expone Parameters y puede ejecutarse en el ámbito de un objeto Transaction de Connection. La clase base para todos los objetos Command es DbCommand.
DataReader	Lee una secuencia de datos de sólo avance y sólo lectura desde un origen de datos. La clase base para todos los objetos DataReader es DbDataReader.
DataAdapter	Llena un DataSet y realiza las actualizaciones necesarias en el origen de datos. La clase base para todos los objetos DataAdapter es DbDataAdapter.
Transaction	Permite incluir comandos en las transacciones que se realizan en el origen de datos. La clase base para todos los objetos Transaction es DbTransaction.

Objeto	Descripción
CommandBuilder	Un objeto auxiliar que genera automáticamente las propiedades de comando de un DataAdapter o que obtiene de un procedimiento almacenado información acerca de parámetros con la que puede rellenar la colección Parameters de un objeto Command. La clase base para todos los objetos CommandBuilder es DbCommandBuilder.
ConnectionStringBuilder	Un objeto auxiliar que proporciona un modo sencillo de crear y administrar el contenido de las cadenas de conexión utilizadas por los objetos Connection. La clase base para todos los objetos ConnectionStringBuilder es DbConnectionStringBuilder.
Parameter	Define los parámetros de entrada, salida y valores devueltos para los comandos y procedimientos almacenados. La clase base para todos los objetos Parameter es DbParameter.
Exception	Se devuelve cuando se detecta un error en el origen de datos. En el caso de que el error se detecte en el cliente, los proveedores de datos de .NET Framework inician una excepción de .NET Framework. La clase base para todos los objetos Exception es DbException.
Error ClientPermission	Expone la información relacionada con una advertencia o error devueltos por un origen de datos. Se proporciona para los atributos de seguridad de acceso a código de los proveedores de datos de .NET Framework. La clase base para todos los objetos ClientPermission es DBDataPermission.

Proveedor de Datos para SQL Server



El proveedor de datos de .NET Framework para SQL Server utiliza su propio protocolo para comunicarse con SQL Server, sin agregar una capa OLE DB u ODBC.

Las clases del proveedor de datos de .NET Framework para SQL Server se encuentran en el espacio de nombres System.Data.SqlClient. Para las versiones anteriores de SQL Server, utilizar el

proveedor de datos de .NET Framework para OLE DB con el proveedor OLE DB de SQL Server (SQLOLEDB).

El proveedor de datos de .NET Framework para SQL Server admite tanto transacciones locales como distribuidas.

En el caso de las transacciones distribuidas, el proveedor de datos de .NET Framework para SQL Server se inscribe automáticamente y de forma predeterminada en una transacción y obtiene los detalles de la misma desde los servicios de componentes de Windows o System.Transactions.

Connection de ADO .NET

El objeto Connection se usa en ADO.NET para establecer una conexión a un origen de datos determinado.

Las conexiones se manejan dentro de un pool interno de conexiones

Si se está utilizando herramientas de diseño de datos en Visual Studio, a menudo no se necesitará crear de forma explícita un objeto Connection para el formulario o componente.

Para establecer conexiones con las distintas bases de datos utilizar:

- Microsoft SQL Server 7.0 o posterior: el objeto SqlConnection
- Microsoft SQL Server 6.x u OLE DB: el objeto OleDbConnection
- ODBC: el objeto OdbcConnection
- Oracle: el objeto OracleConnection

Se recomienda cerrar explícitamente las conexiones

Se puede utilizar el bloque using en C# o Using en VB para cerrar automáticamente la conexión cuando el código sale del bloque, aun cuando se produjera una excepción

También se puede utilizar los métodos Close o Dispose del objeto del tipo Connection

Por ejemplo, una conexión que sale del alcance donde fue declarada que no haya sido cerrada explícitamente no retorna al pool de conexiones a menos que se haya alcanzado el límite de conexiones disponibles del pool

C#

```
// Asume que connectionString es un string de conexión válido
using (SqlConnection connection = new SqlConnection(connectionString))
        {
                connection.Open();
                // Realización de tareas.
        }
```

VB

```
' Asume que connectionString es un string de conexión válido
Using connection As New SqlConnection(connectionString)
      connection.Open()
      ' Realización de tareas.
End Using
```

Parámetros del String de Conexión

Nombre del parámetro	Descripción
Data Source	Identifica al servidor. Puede ser una máquina local, un nombre del dominio o una dirección IP
Initial Catalog	Nombre de la base de datos
Integrated Security	Inicializado a SSPI para que la conexión se haga con el inicio de sesión de Windows del usuario
User ID	Nombre del usuario que está configurado en el SQL Server
Password	Clave del usuario anterior en el SQL Server

Nota: la seguridad integrada sólo es buena cuando se desarrolla

Usando SqlConnection

El propósito de crear un objeto SqlConnection es habilitar la conexión para los demás objetos ADO .NET de manera que puedan trabajar con la bdd

Objetos como SqlCommand y SqlDataAdapter reciben como parámetro un SqlConnection

La secuencia de operaciones es la siguiente:

1. Crear una instancia de SqlConnection
2. Abrir la conexión
3. Pasar el objeto de conexión a los otros objetos ADO
4. Realizar operaciones en la bdd
5. Cerrar la conexión

Nuevamente, lo normal es declararlo e instanciarlo al mismo tiempo. Notar que el segundo parámetro es la referencia a la conexión

C#

```
requerimiento =
                    "INSERT INTO Cliente (DNI, NomCliente, Direccion) VALUES ("
                    + "'" + dni + "'" + ","
                    + "'" + nombre + "'" + ","
```

```
                        + "'" + direccion + "'" + ")";

        SqlCommand cmd = new SqlCommand(requerimiento, con);
```

VB

```
        requerimiento = _
                        "INSERT INTO Cliente (DNI, NomCliente, Direccion) VALUES (" _
                        & "'" & dni & "'" & "," _
                        & "'" & nombre & "'" & "," _
                        & "'" & direccion & "'" & ")"

        Dim cmd As SqlCommand = New SqlCommand(requerimiento, con)
```

Este es un ejemplo típico de creación, donde se pasa como parámetro un string que almacena el comando a ejecutar.

Existen otras opciones de creación ya que el constructor está sobrecargado

Trabajar con SqlCommand se lo denomina modo conectado porque se mantiene la conexión con la bdd activa

SqlCommand para Consultar Datos

Cuando se utiliza un comando SELECT de SQL se recupera un conjunto de datos. El objeto SqlCommand utiliza el método ExecuteReader que retorna un objeto del tipo SqlDataReader que permite esta operación.

VB

```
' La siguiente línea crea el string SQL SELECT requerido
requerimiento = _
        "SELECT NomCliente, Direccion FROM Cliente " _
& " WHERE DNI=" & "'" & dni & "'"

' 2 Crea el objeto SqlCommand para realizar la operación
Dim cmd As SqlCommand = New SqlCommand(requerimiento, con)

' 3  Llamar al método que ejecuta el lector para obtener los resultados
rdr = cmd.ExecuteReader()
```

SqlCommand para Insertar Datos

Para insertar datos en la bdd, usar el método ExecuteNonQuery de SqlCommand

C#

```
// La siguiente sentencia crea el string de SQL requerido para el INSERT
requerimiento =
        "INSERT INTO Cliente (DNI, NomCliente, Direccion) VALUES ("
                + "'" + dni + "'" + ","
                + "'" + nombre + "'" + ","
                + "'" + direccion + "'" + ")";

//** 6 Crear una nueva instancia del comando con una consulta y la conexión
//**   Llamar al método ExecuteNonQuery para enviar el comando
```

```
SqlCommand cmd = new SqlCommand(requerimiento, con);
cmd.ExecuteNonQuery();
```

VB

```
' La siguiente sentencia crea el string de SQL requerido para el INSERT
requerimiento = _
       "INSERT INTO Cliente (DNI, NomCliente, Direccion) VALUES (" _
       & "'" & dni & "'" & "," _
       & "'" & nombre & "'" & "," _
       & "'" & direccion & "'" & ")"

' 6 Crear una nueva instancia del comando con una consulta y la conexión
'   Llamar al método ExecuteNonQuery para enviar el comando
Dim cmd As SqlCommand = New SqlCommand(requerimiento, con)
cmd.ExecuteNonQuery()
```

SqlCommand para Actualizar Datos

El método ExecuteNonQuery se usa también para actualizar datos

C#

```
// La siguiente línea crea el string SQL UPDATE requerido
requerimiento =
       "UPDATE Cliente SET "
       + " NomCliente=" + "'" + nombre + "'" + ","
       + " Direccion=" + "'" + direccion + "'"
       + " WHERE DNI=" + "'" + dni + "'";

//** 5 Ejecuta el método ExecuteNonQuery(requerimiento, com)
//**   para realizar la operación
SqlCommand cmd = new SqlCommand(requerimiento, con);
cmd.ExecuteNonQuery();
```

VB

```
' La siguiente línea crea el string SQL UPDATE requerido
requerimiento = _
       "UPDATE Cliente SET " _
       & " NomCliente=" & "'" & nombre & "'" & "," _
       & " Direccion=" & "'" & direccion & "'" _
       & " WHERE DNI=" & "'" & dni & "'"

' 5 Ejecuta el método ExecuteNonQuery(requerimiento, com)
'   para realizar la operación
Dim cmd As SqlCommand = New SqlCommand(requerimiento, con)
cmd.ExecuteNonQuery()
```

SqlCommand para Borrar Datos

Nuevamente, para borrar datos:

C#

```
// La siguiente línea crea el string SQL DELETE requerido
// para borrar filas de la tabla Cliente
requerimiento =
```

```
        "DELETE FROM Cliente WHERE DNI=" + "'" + dni + "'";

//** 4 Ejecuta el método ExecuteNonQuery(requerimiento, com)
//**   para realizar la operación
SqlCommand cmd = new SqlCommand(requerimiento, con);
cmd.ExecuteNonQuery();
```

VB

```
' La siguiente línea crea el string SQL DELETE requerido
' para borrar filas de la tabla Cliente
requerimiento = "DELETE FROM Cliente WHERE DNI=" & "'" & dni & "'"

' 4 Ejecuta el método ExecuteNonQuery(requerimiento, com)
'   para realizar la operación
Dim cmd As SqlCommand = New SqlCommand(requerimiento, con)
cmd.ExecuteNonQuery()
```

Mapa entre los tipos de datos SQL y los tipos de datos .Net

La siguiente tabla muestra la correspondencia entre los tipos de datos para almacenar valores leídos desde un servidor de datos. Se muestran los tipos de datos de SQL Server 2008 R2 y sus equivalentes con CRL, para el namespace System.Data.SqlTypes y los tipos nativos de CRL .NET FrameWork

Tipo de datos en SQL Server	Tipo (en System.Data.SqlTypes o Microsoft.SqlServer.Types)	Tipo de datos en el CLR (.NET Framework)
bigint	SqlInt64	Int64, Nullable<Int64>
binary	SqlBytes, SqlBinary	Byte[]
bit	SqlBoolean	Boolean, Nullable<Boolean>
char	ninguno	ninguno
cursor	ninguno	ninguno
date	SqlDateTime	DateTime, Nullable<DateTime>
datetime	SqlDateTime	DateTime, Nullable<DateTime>
datetime2	ninguno	DateTime, Nullable<DateTime>
DATETIMEOFFSET	ninguno	DateTimeOffset, Nullable<DateTimeOffset>
decimal	SqlDecimal	Decimal, Nullable<Decimal>
float	SqlDouble	Double, Nullable<Double>

Tipo de datos en SQL Server	Tipo (en System.Data.SqlTypes o Microsoft.SqlServer.Types)	Tipo de datos en el CLR (.NET Framework)
geography	SqlGeography	ninguno
geometry	SqlGeometry	ninguno
hierarchyid	SqlHierarchyId	ninguno
image	ninguno	ninguno
int	SqlInt32	Int32, Nullable<Int32>
money	SqlMoney	Decimal, Nullable<Decimal>
nchar	SqlChars, SqlString	String, Char[]
ntext	ninguno	ninguno
numeric	SqlDecimal	Decimal, Nullable<Decimal>
nvarchar	SqlChars, SqlString	String, Char[]
nvarchar(1), nchar(1)	SqlChars, SqlString	Char, String, Char[], Nullable<char>
real	SqlSingle	Single, Nullable<Single>
rowversion	ninguno	Byte[]
smallint	SqlInt16	Int16, Nullable<Int16>
smallmoney	SqlMoney	Decimal, Nullable<Decimal>
sql_variant	ninguno	Object
table	ninguno	ninguno
text	ninguno	ninguno
time	ninguno	TimeSpan, Nullable<TimeSpan>
timestamp	ninguno	ninguno
tinyint	SqlByte	Byte, Nullable<Byte>
uniqueidentifier	SqlGuid	Guid, Nullable<Guid>

Tipo de datos en SQL Server	Tipo (en System.Data.SqlTypes o Microsoft.SqlServer.Types)	Tipo de datos en el CLR (.NET Framework)
User-defined type(UDT)	ninguno	La misma clase que está ligada al tipo de datos definido por el usuario en el mismo assembly o en uno dependiente
varbinary	SqlBytes, SqlBinary	Byte[]
varbinary(1), binary(1)	SqlBytes, SqlBinary	byte, Byte[], Nullable<byte>
varchar	ninguno	ninguno
xml	SqlXml	ninguno

Notas:

- SQLChars es mejor para la transferencia de datos y SQLString obtiene mejor rendimiento para operaciones con Strings.
- SqlHierarchyId está definido en Microsoft.SqlServer.Types.dll, la cual es una librería que se instala con SQL Server 2008 R2 y puede descargarse del SQL Server 2008 feature pack.

Métodos de conversión de datos de .Net para los tipos SQL

El conjunto de resultados retornado por una consulta SQL contiene tipos de datos SQL. Se necesita convertir los tipos SQL a tipos soportados por Java. La interfaz ResulSet provee una lista de métodos getXXX para esta tarea, como muestra la siguiente tabla

Tipo del motor de base de datos de SQL Server	Tipo del .NET Framework	Enumeración SqlDbType	Descriptor de acceso con tipo SqlDataReader SqlTypes	Enumeración DbType	Descriptor de acceso con tipo SqlDataReader DbType
bigint	Int64	BigInt	GetSqlInt64	Int64	GetInt64
binary	Byte[]	VarBinary	GetSqlBinary	Binary	GetBytes
bit	Boolean	Bit	GetSqlBoolean	Boolean	GetBoolean
char	String	Char	GetSqlString	AnsiStringFixedLength	GetString
	Char[]			String	GetChars

Tipo del motor de base de datos de SQL Server	Tipo del .NET Framework	Enumeración SqlDbType	Descriptor de acceso con tipo SqlDataReader SqlTypes	Enumeración DbType	Descriptor de acceso con tipo SqlDataReader DbType
date (sólo SQL Server 2008)	DateTime	Date	GetSqlDateTime	Date	GetDateTime
datetime	DateTime	DateTime	GetSqlDateTime	DateTime	GetDateTime
datetime2 (solo SQL Server 2008)	DateTime2	DateTime2	GetSqlDateTime	DateTime2	GetDateTime
datetimeoffset, (solo SQL Server 2008)	DateTimeOffset	DateTimeOffset	Ninguno	DateTimeOffset	GetDateTimeOffset
decimal	Decimal	Decimal	GetSqlDecimal	Decimal	GetDecimal
Atributo FILESTREAM (varbinary(max))	Byte[]	VarBinary	GetSqlBytes	Binary	GetBytes
float	Double	Float	GetSqlDouble	Double	GetDouble
image	Byte[]	Binary	GetSqlBinary	Binary	GetBytes
int	Int32	Int	GetSqlInt32	Int32	GetInt32
money	Decimal	Money	GetSqlMoney	Decimal	GetDecimal
nchar	String	NChar	GetSqlString	StringFixedLength	GetString
	Char[]				GetChars
ntext	String	NText	GetSqlString	String	GetString
	Char[]				GetChars
numeric	Decimal	Decimal	GetSqlDecimal	Decimal	GetDecimal
nvarchar	String	NVarChar	GetSqlString	String	GetString
	Char[]				GetChars
real	Single	Real	GetSqlSingle	Single	GetFloat

Tipo del motor de base de datos de SQL Server	Tipo del .NET Framework	Enumeración SqlDbType	Descriptor de acceso con tipo SqlDataReader SqlTypes	Enumeración DbType	Descriptor de acceso con tipo SqlDataReader DbType
rowversion	Byte[]	Timestamp	GetSqlBinary	Binary	GetBytes
smalldatetime	DateTime	DateTime	GetSqlDateTime	DateTime	GetDateTime
smallint	Int16	SmallInt	GetSqlInt16	Int16	GetInt16
smallmoney	Decimal	SmallMoney	GetSqlDecimal	Decimal	GetDecimal
sql_variant	Object *	Variant	GetSqlValue	Object	GetValue *
text	String	Text	GetSqlString	String	GetString
	Char[]				GetChars
time (solo SQL Server 2008)	TimeSpan	Time	Ninguno	Time	GetDateTime
timestamp	Byte[]	Timestamp	GetSqlBinary	Binary	GetBytes
tinyint	Byte	TinyInt	GetSqlByte	Byte	GetByte
uniqueidentifier	Guid	UniqueIdentifier	GetSqlGuid	Guid	GetGuid
varbinary	Byte[]	VarBinary	GetSqlBinary	Binary	GetBytes
varchar	String	VarChar	GetSqlString	AnsiString, String	GetString
	Char[]				GetChars
xml	Xml	Xml	GetSqlXml	Xml	ninguno

Estrategias en la producción de código

Desde la perspectiva de quien produce el código, los ejemplos expuestos hasta el momento tienen algunas limitaciones que deben ser resueltas. Una de las principales es la limitación de incluir dentro del código toda información, como por ejemplo:

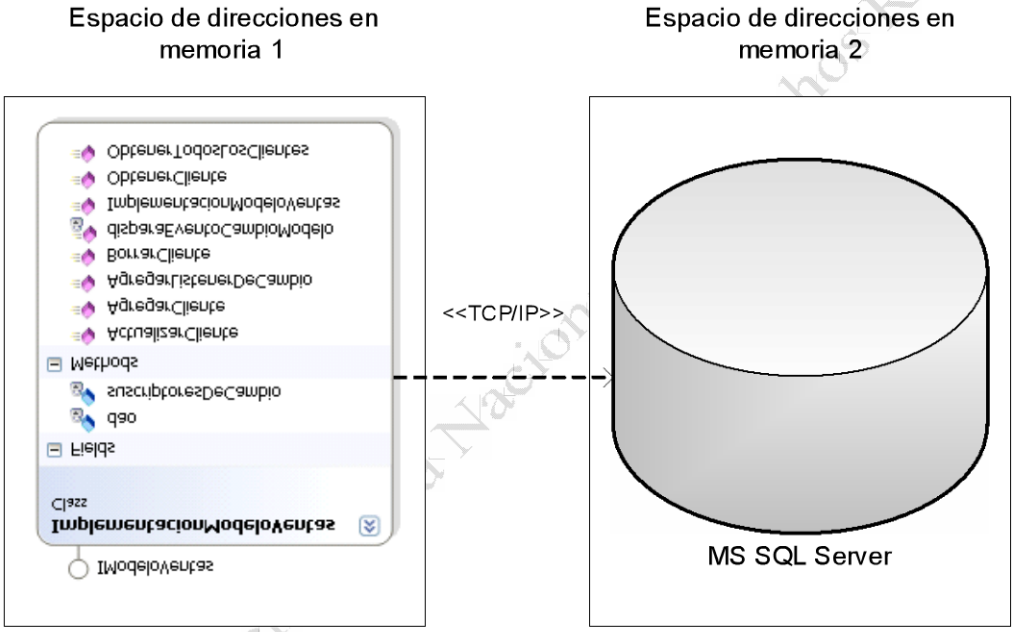
- La URL utilizada para establecer la conexión
- El nombre de usuario
- La clave

Manteniendo esto dentro del código, lo hace menos flexible que si se incorporase esta información como parámetros. En el caso del usuario y la clave se podría realizar una interfaz gráfica del usuario para aceptar esos datos e iniciar una conexión y el caso de los parámetros de la base como la URL, se pueden cargar desde un archivo cuando fuera necesario.

Introducción al patrón de diseño DAO

Contexto del problema

Para el acceso a base de datos, se presenta la siguiente situación:

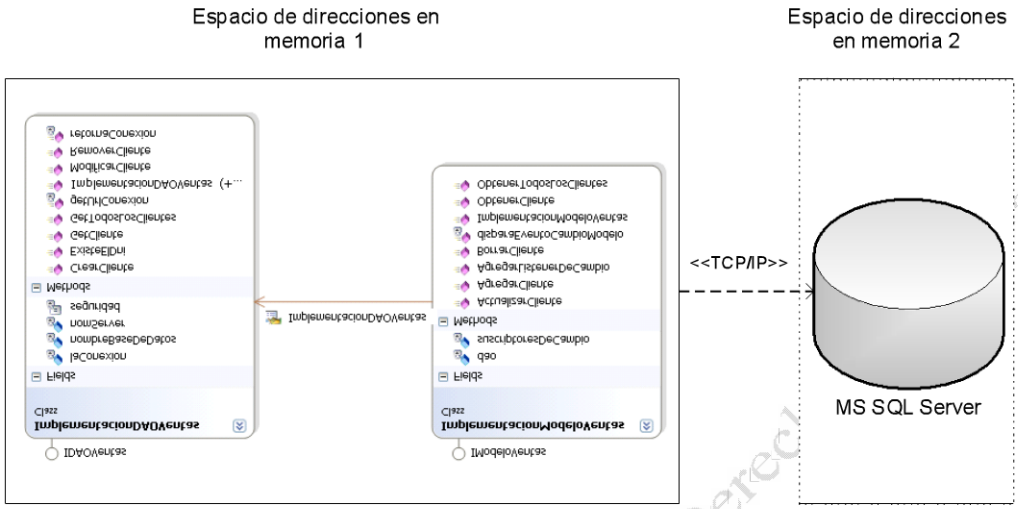


La mayor falla de este diseño es el acoplamiento entre la conexión a la base de datos y los métodos de acceso a datos junto con la implementación de los métodos del modelo. Para tener un mejor diseño, la base de datos y sus actividades relacionadas deben estar separadas de la implementación del modelo

Solución

La solución al problema mencionado es desacoplar la lógica de negocios del acceso a la base de datos usando el patrón de diseño DAO. De esta manera, la clase cliente del patrón DAO desconoce la problemática inherente a los tecnicismos de acceso a esta, como ser la representación física, las relaciones u otros elementos que afecten a los datos.

La siguiente figura muestra la implementación de la solución manteniendo separados los espacios de memoria donde residen el código y el servidor de bases de datos



El patrón de diseño DAO se implementa siguiendo los siguientes pasos:

1. Definir una interfaz para el DAO: esta contiene los métodos necesarios para la interacción (lectura y escritura) con una fuente de datos. Los métodos deberían ser lo suficientemente genéricos como para no poseer ningún aspecto que acople al patrón a una base de datos específica.
2. Escribir una implementación de la interfaz que es específica a la fuente de datos utilizada
3. Codificar la lógica de negocios de la aplicación para acceder a la fuente de datos utilizando sólo los métodos provistos por la interfaz

La principal ventaja de utilizar el patrón de diseño DAO es la de aislar el código que necesita cambiarse cada vez que se cambie la fuente de datos de la aplicación

Una implementación factible para la aplicación es la siguiente

C#

```
public interface IDAOVentas
    {
        void CrearCliente(Cliente cliente);
        void RemoverCliente(Cliente cliente);
        void ModificarCliente(Cliente cliente);
        Cliente GetCliente(String dni);
        Cliente[] GetTodosLosClientes();
        bool ExisteElDni(String dni);
    }
```

VB

```
Public Interface IDAOVentas
    Sub CrearCliente(ByVal cliente As Cliente)
```

```
    Sub RemoverCliente(ByVal cliente As Cliente)
    Sub ModificarCliente(ByVal cliente As Cliente)
    Function GetCliente(ByVal dni As String) As Cliente
    Function GetTodosLosClientes() As Cliente()
    Function ExisteElDni(ByVal dni As String) As Boolean
End Interface
```

Laboratorio 4

En este ejercicio se deben incorporar dos proyectos a la solución en curso, Modelo y VerificaModelo para completar el código y hacer funcionar el modelo del MVC de la aplicación