UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERIA

Escuela Profesional de Ingeniería de Sistemas

APLICACIÓN DEL ORM ENTITY FRAMEWORK EN EL SISTEMA BAE RESTAURANTE

Curso: Base de Datos II

Docente: Mag. Ing. Patrick Cuadros Quiroga

Integrantes:

Anco Copaja, Brian Sebastian **(2018000359)**
Martínez Yufra, Ericka Esther **(2018000368)**
Contreras Murguia, José Manuel **(2016056346)**

Tacna – Perú
2021

RESUMEN

(Mapeo Objeto relacional) es la técnica de programación que nos permite transformar los datos entre el sistema de tipos utilizados en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, utilizando un motor de persistencia. Lo que da como resultado una base de datos orientada a objetos virtual sobre la base de datos relacional. Esta transformación permite el uso de las bondades de la programación orientada a objetos. Los ORM nacieron a mediados de la década del 90, se hicieron masivos a partir de la masificación de Java. Por esta razón, los frameworks más populares hoy en día en .NET son adaptaciones del modelo pensado en Java. Uno de los componentes de OMR más utilizado, por no decir el más utilizado es HIBERNATE, surgido del ambiente JAVA y llevado al uso del framework.NET con la versión NHIBERNATE.

INTRODUCCIÓN

El mapeo objeto-relacional es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional. En la práctica esto crea una base de datos orientada a objetos virtual, por sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).

APLICACIÓN DEL ORM ENTITY FRAMEWORK EN EL SISTEMA BAE RESTAURANTE

DESARROLLO DEL TRABAJO

Object Relational Mapping (ORM)

Es una técnica de programación para relacionar datos entre un lenguaje de programación orientado a objetos y el sistema de bases de datos relacional utilizado en el desarrollo de una aplicación. El principal problema que surge hoy en día, prácticamente que todas las aplicaciones están diseñadas para usar la programación orientada a objeto (POO) mientras que las bases de datos más usadas son del tipo relacional. Las bases de datos relacionales solo permiten guardar tipos de datos primitivos (enteros, cadenas de texto, etc) por lo que no se pueden guardar de forma directa los objetos de la aplicación en las tablas, si no, que estos se deben de convertir antes en registros, (por lo general afectan a varias tablas) En el momento de volver a recuperar los datos hay que hacer el proceso contrario, se debe convertir los registros en objetos.

Ventajas

1. Rapidez en el desarrollo. La mayoría de las herramientas actuales permiten la creación del modelo por medio del esquema de la base de datos, leyendo el esquema, nos crea el modelo adecuado. Esto quita mucho tiempo de “coding” repetitivo y que, seguro, el sistema hace mejor que nosotros, ya que los humanos siempre cometemos fallos tontos.
2. Abstracción de la base de datos. Al utilizar un sistema ORM, lo que conseguidos es separarnos totalmente del sistema de Base de datos que utilicemos, y así si en un futuro debemos de cambiar de motor de bases de datos, tendremos la seguridad de que este cambio no nos afectará a nuestro sistema, siendo el cambio más sencillo.
3. Reutilización. Nos permite utilizar los métodos de un objeto de datos desde distintas zonas de la aplicación, incluso desde aplicaciones distintas. Seguridad. Los ORM suelen implementar sistemas para evitar tipos de ataques como pueden ser los SQL injections.
4. Mantenimiento del código. Nos facilita el mantenimiento del código debido a la correcta ordenación de la capa de datos, haciendo que el mantenimiento del código sea mucho más sencillo.
5. Lenguaje propio para realizar las consultas. Estos sistemas de mapeo traen su propio lenguaje para hacer las consultas, lo que hace que los usuarios dejen de utilizar las sentencias SQL para que pasen a utilizar el lenguaje propio de cada herramienta.

Desventajas

1. Tiempo utilizado en el aprendizaje. Este tipo de herramientas suelen ser complejas por lo que su correcta utilización lleva un tiempo que hay que emplear en ver el funcionamiento correcto y ver todo el partido que se le puede sacar.
2. Aplicaciones algo más lentas. Esto es debido a que todas las consultas que se hagan sobre la base de datos, el sistema primero deberá de transformarlas al lenguaje propio de la herramienta, luego leer los registros y por último crear los objetos.

¿Por qué utilizar un ORM?

Aunque el lenguaje SQL se usa para acceder a muchas de las bases de datos existentes, existen múltiples varianzas en las funciones que los distintos SGBD han usado. Un ejemplo muy sencillo sería delimitar el número de registros de una consulta:

```
SELECT TOP 10 * FROM usuarios //SqlServer
SELECT * FROM usuarios LIMIT 10 //MySQL
SELECT * FROM usuarios WHERE rownum<=20; //Oracle
```

Como podemos observar, para algo tan fácil vemos diferencias. Esto para el programador supone tener que conocer el lenguaje para cada Base de datos, y más importante aún, si en un futuro se desea migrar la aplicación, habría que reescribir gran número de las consultas.

ORMs más utilizados

Casi todos los lenguajes de alto nivel actualmente disponen de alguna solución de este tipo, una de las más conocidas es Hibernate para JAVA, pero existen muchas más:

- Java => Hibernate, iBatis, Ebean, etc..
- .NET=> Entity Framework, nHibernate, etc..
- PHP=> Doctrine, Propel, ROcks, Torpor, etc..

Entity Framework

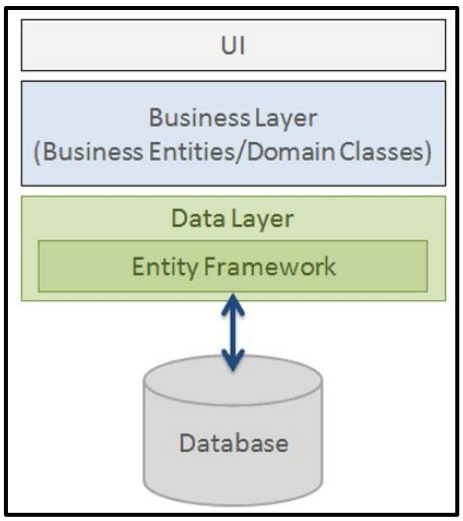
Antes de .NET 3.5, los desarrolladores escribían código ADO.NET o Enterprise Data Access Block para guardar o recuperar datos de aplicaciones de la base de datos subyacente. Solían abrir una conexión a la base de datos, crear un DataSet para recuperar o enviar los datos a la base de datos, convertir datos del DataSet a objetos .NET o viceversa para aplicar reglas comerciales. Este fue un proceso engorroso y propenso a errores. Microsoft ha proporcionado un marco llamado "Entity Framework" para automatizar todas estas actividades relacionadas con la base de datos para su aplicación.

Entity Framework es un marco ORM de código abierto para aplicaciones .NET compatible con Microsoft. Permite a los desarrolladores trabajar con datos utilizando objetos de clases específicas de dominio sin centrarse en las tablas y columnas de la base de datos subyacente donde se almacenan estos datos. Con Entity Framework, los desarrolladores pueden trabajar a un mayor nivel de abstracción

cuando tratan con datos y pueden crear y mantener aplicaciones orientadas a datos con menos código en comparación con las aplicaciones tradicionales.

Definición oficial: “Entity Framework es un mapeador relacional de objetos (O / RM) que permite a los desarrolladores .NET trabajar con una base de datos utilizando objetos .NET. Elimina la necesidad de la mayor parte del código de acceso a datos que los desarrolladores suelen tener que escribir ".

La siguiente figura ilustra dónde encaja Entity Framework en su aplicación.



Según la figura anterior, Entity Framework encaja entre las entidades comerciales (clases de dominio) y la base de datos. Guarda los datos almacenados en las propiedades de las entidades comerciales y también recupera datos de la base de datos y los convierte automáticamente en objetos de entidades comerciales.

Características de Entity Framework

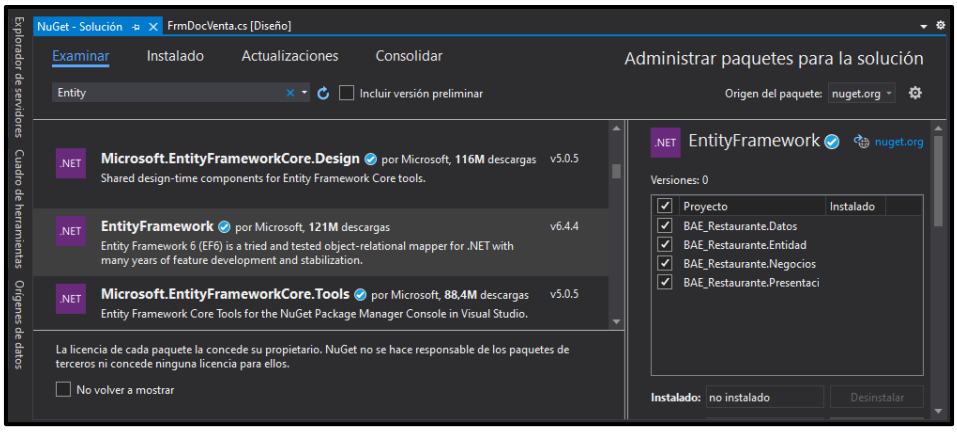
- Multiplataforma: EF Core es un marco multiplataforma que puede ejecutarse en Windows, Linux y Mac.
- Modelado: EF (Entity Framework) crea un EDM (Entity Data Model) basado en entidades POCO (Plain Old CLR Object) con propiedades get / set de diferentes tipos de datos. Utiliza este modelo al consultar o guardar datos de entidades en la base de datos subyacente.
- Consultas: EF nos permite utilizar consultas LINQ (C # / VB.NET) para recuperar datos de la base de datos subyacente. El proveedor de la base de datos traducirá estas consultas LINQ al lenguaje de consulta específico de la base de datos (por ejemplo, SQL para una base de datos relacional). EF también nos permite ejecutar consultas SQL sin procesar directamente en la base de datos.
- Seguimiento de cambios: EF realiza un seguimiento de los cambios ocurridos en las instancias de sus entidades (valores de propiedad) que deben enviarse a la base de datos.

- Guardar: EF ejecuta los comandos INSERT, UPDATE y DELETE en la base de datos en función de los cambios ocurridos en sus entidades cuando llama al SaveChanges()método. EF también proporciona el SaveChangesAsync()método asincrónico .
- Simultaneidad: EF utiliza la simultaneidad optimista de forma predeterminada para proteger la sobrescritura de los cambios realizados por otro usuario desde que se obtuvieron los datos de la base de datos.
- Transacciones: EF realiza una gestión automática de transacciones mientras consulta o guarda datos. También proporciona opciones para personalizar la gestión de transacciones.
- Almacenamiento en caché: EF incluye el primer nivel de almacenamiento en caché listo para usar. Por lo tanto, las consultas repetidas devolverán datos de la caché en lugar de acceder a la base de datos.
- Convenciones integradas : EF sigue las convenciones sobre el patrón de programación de configuración e incluye un conjunto de reglas predeterminadas que configuran automáticamente el modelo EF.
- Configuraciones: EF nos permite configurar el modelo EF mediante el uso de atributos de anotación de datos o Fluent API para anular las convenciones predeterminadas.
- Migraciones: EF proporciona un conjunto de comandos de migración que se pueden ejecutar en la consola del administrador de paquetes NuGet o en la interfaz de línea de comandos para crear o administrar el esquema de la base de datos subyacente.

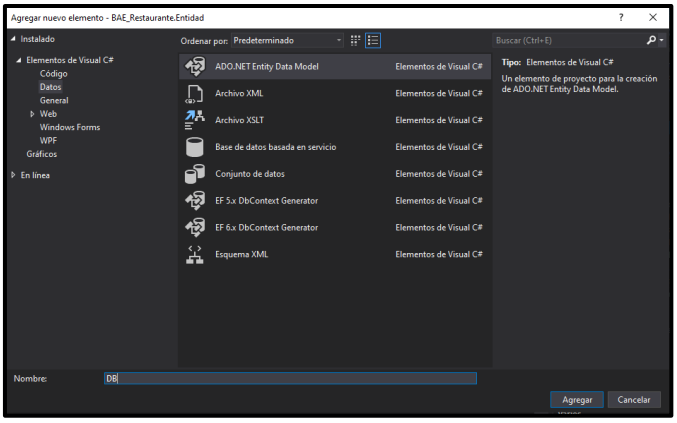
Aplicación de Entity Framework en el sistema BAE Restaurante

BAE Restaurante es un sistema desarrollado en el lenguaje C#, programado con 4 capas: Entidad, Datos, Negocios, Presentación y conectado a una base de datos SQL. El sistema se encarga de la administración completa de un restaurante, con funciones como administrar clientes, empleados, pedidos, mesas, reservas, etc.

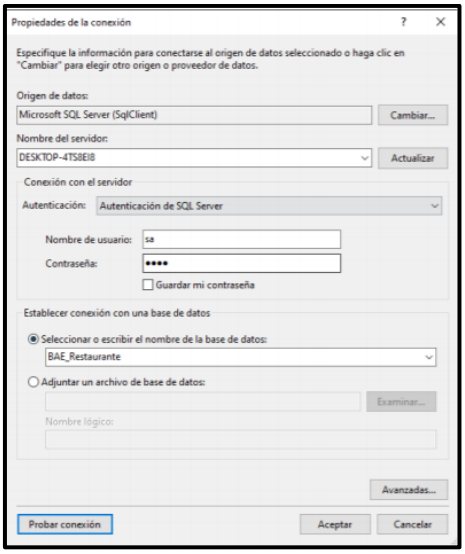
Para empezar con la aplicación de nuestro ORM, descargaremos el Entity Framework desde el administrador de NuGets de Visual Studio 2019.



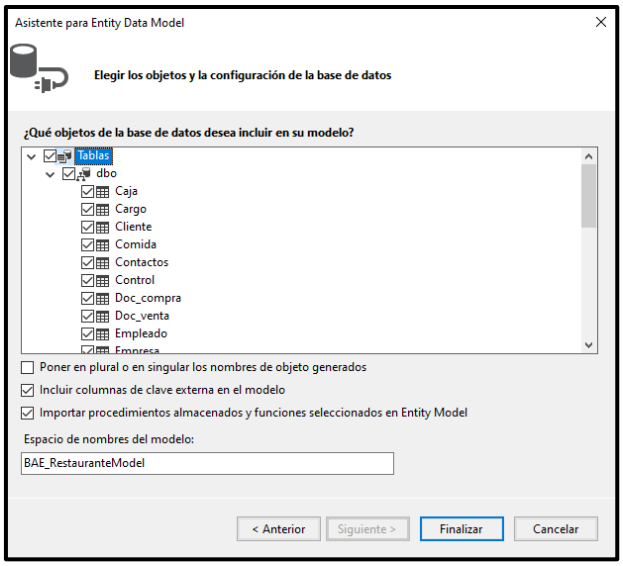
Agregamos nuestro Entity Data Model.



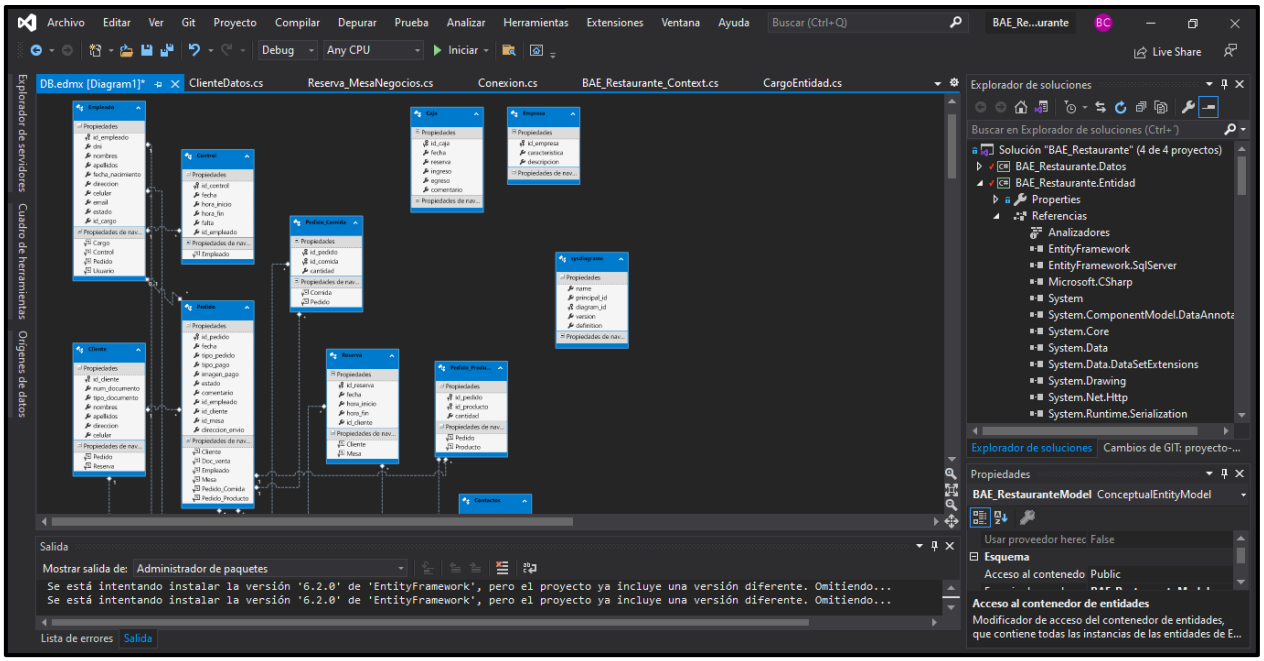
Seguidamente, definimos la cadena de conexión a nuestra base de datos SQL mediante Autenticación SQL Server con el usuario sa.



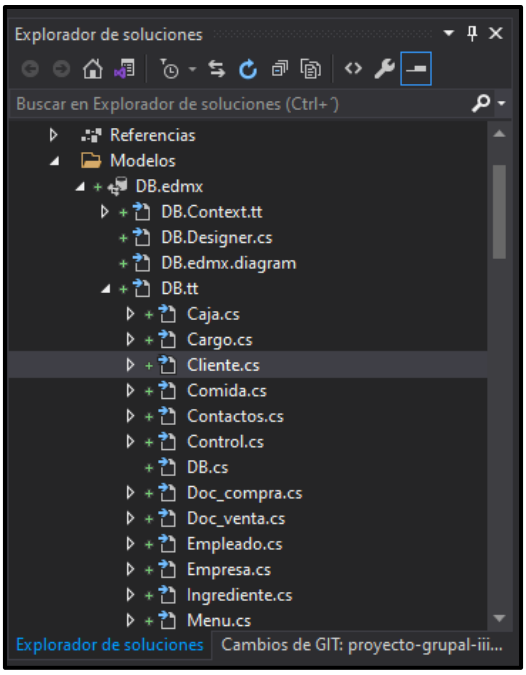
Escogemos todas las tablas de nuestra base de datos para agregarlas al modelo.



Tras esperar algunos minutos, podremos observarlas tablas mapeadas, que son una representación de nuestra base de datos mapeada, respetando atributos y relaciones.



También, dentro de nuestra carpeta Modelos encontraremos las clases generadas por el Entity Framework. Esta carpeta se encuentra dentro de nuestra capa Entidad.



Una vez hecho todo esto, ya podemos ejecutar los procedimientos de Insertar, Actualizar, Listar y Eliminar mediante nuestro ORM, utilizando LINQ. A continuación, ejemplificaremos mediante la tabla Clientes el uso y comparación de los procedimientos realizados.

Clientes – Actualizar sin ORM

El procedimiento para insertar clientes antes de utilizar nuestro ORM, era mediante la capa Negocios, enviando los parámetros desde la Capa Presentación.

Capa Presentación – Actualizar Cliente

```
//PROCEDIMIENTO SIN ORM
Rpta = ClienteNegocios.Actualizar(Convert.ToInt32
  (txtIdCliente.Text),txtDNI.Text,txtTipo.Text,txtNombres.Text,txtApellidos.Text,txtDireccion.Text,txtCelular.Text);
if (Rpta.Equals("OK"))
{
    this.MensajeCorrecto("Cliente actualizado correctamente.");
    this.ListarClientes();
}
else
{
    this.MensajeError(Rpta);
}
```

Capa Negocios – Actualizar Cliente

```
1 referencia | BrianAnco, Hace 112 días | 2 autores, 2 cambios
public static string Actualizar(int id_cliente, string num_documento, string tipo_documento, string nombres, string apellidos, string
  direccion, string celular)
{
    ClienteDatos objCategoria = new ClienteDatos();
    ClienteEntidad objCategoriaE = new ClienteEntidad();
    objCategoriaE.id_cliente = id_cliente;
    objCategoriaE.num_documento = num_documento;
    objCategoriaE.tipo_documento = tipo_documento;
    objCategoriaE.nombres = nombres;
    objCategoriaE.apellidos = apellidos;
    objCategoriaE.direccion = direccion;
    objCategoriaE.celular = celular;
    return objCategoria.Actualizar(objCategoriaE);

}
```

Capa Datos – Actualizar Cliente

```
SqlConnection sqlCnx = new SqlConnection();
sqlCnx = Conexion.getInstancia().EstablecerConexion();
SqlCommand comando = new SqlCommand("usp_Cliente_U", sqlCnx);
comando.CommandType = CommandType.StoredProcedure;
comando.Parameters.Add("@pid_cliente", SqlDbType.Int).Value = objcargo.id_cliente;
comando.Parameters.Add("@pnum_documento", SqlDbType.VarChar).Value = objcargo.num_documento;
comando.Parameters.Add("@ptipo_documento", SqlDbType.VarChar).Value = objcargo.tipo_documento;
comando.Parameters.Add("@pnombres", SqlDbType.VarChar).Value = objcargo.nombres;
comando.Parameters.Add("@papellidos", SqlDbType.VarChar).Value = objcargo.apellidos;
comando.Parameters.Add("@pdireccion", SqlDbType.VarChar).Value = objcargo.direccion;
comando.Parameters.Add("@pcelular", SqlDbType.VarChar).Value = objcargo.celular;
sqlCnx.Open();
Rpta = comando.ExecuteNonQuery() == 1 ? "OK" : "No se pudo actualizar el registro.";
```

Clientes – Actualizar con EF y LINQ

```
using (BAE_RestauranteEntities db = new BAE_RestauranteEntities())
{
    Cliente iCliente = new Cliente();
    iCliente.id_cliente = Convert.ToInt32(txtIdCliente.Text);
    iCliente.num_documento = txtDNI.Text;
    iCliente.tipo_documento = txtTipo.Text;
    iCliente.nombres = txtNombres.Text;
    iCliente.apellidos = txtApellidos.Text;
    iCliente.direccion = txtDireccion.Text;
    iCliente.celular = txtCelular.Text;
    db.Entry(iCliente).State = System.Data.Entity.EntityState.Modified;
    db.SaveChanges();

}
this.MensajeCorrecto("Cliente modificado correctamente.");
this.ListarClientes();
```

Clientes – Insertar sin ORM

Capa Presentación – Insertar Clientes

```
Rpta = ClienteNegocios.Insertar(txtDNI.Text, txtTipo.Text, txtNombres.Text, txtApellidos.Text,
  txtDireccion.Text, txtCelular.Text);
```

Capa Negocios – Insertar Clientes

```
0 referencias | ErickaEmy, Hace 112 días | 1 autor, 1 cambio
public static string Insertar(string num_documento, string tipo_documento, string nombres,string apellidos, string
  direccion,string celular)
{
    ClienteDatos objCategoria = new ClienteDatos();
    string existe = objCategoria.Existe(num_documento);
    if (existe.Equals("1"))
    {
        return "El cliente ya existe en la base de datos";
    }
    else
    {
        ClienteEntidad objCategoriaE = new ClienteEntidad();
        objCategoriaE.num_documento = num_documento;
        objCategoriaE.tipo_documento = tipo_documento;
        objCategoriaE.nombres = nombres;
        objCategoriaE.apellidos = apellidos;
        objCategoriaE.direccion = direccion;
        objCategoriaE.celular = celular;
        return objCategoria.Insertar(objCategoriaE);
    }
}
```

Capa Datos – Insertar Clientes

```
sqlCnx = Conexion.getInstancia().EstablecerConexion();
SqlCommand comando = new SqlCommand("usp_Cliente_I", sqlCnx);
comando.CommandType = CommandType.StoredProcedure;
comando.Parameters.Add("@pnum_documento", SqlDbType.VarChar).Value = objcargo.num_documento;
comando.Parameters.Add("@ptipo_documento", SqlDbType.VarChar).Value = objcargo.tipo_documento;
comando.Parameters.Add("@pnombres", SqlDbType.VarChar).Value = objcargo.nombres;
comando.Parameters.Add("@papellidos", SqlDbType.VarChar).Value = objcargo.apellidos;
comando.Parameters.Add("@pdireccion", SqlDbType.VarChar).Value = objcargo.direccion;
comando.Parameters.Add("@pcelular", SqlDbType.VarChar).Value = objcargo.celular;
sqlCnx.Open();
Rpta = comando.ExecuteNonQuery() == 1 ? "OK" : "No se pudo agregar el registro.";
```

Clientes – Insertar con EF y LINQ

```
using (BAE_RestauranteEntities db = new BAE_RestauranteEntities())
{
    Cliente iCliente = new Cliente();
    iCliente.num_documento = txtDNI.Text;
    iCliente.tipo_documento = txtTipo.Text;
    iCliente.nombres = txtNombres.Text;
    iCliente.apellidos = txtApellidos.Text;
    iCliente.direccion = txtDireccion.Text;
    iCliente.celular = txtCelular.Text;
    db.Cliente.Add(iCliente);
    db.SaveChanges();

}
this.MensajeCorrecto("Cliente agregado correctamente.");
this.ListarClientes();
```

Clientes – Listar sin ORM

Capa Presentación – Listar Clientes

```
try
{
    dgvEmpleados.DataSource = ClienteNegocios.Listar();
    txtDNI.Clear();
    txtNombres.Clear();
    txtApellidos.Clear();
    txtTipo.Clear();
    txtDireccion.Clear();
    txtCelular.Clear();
    txtIdCliente.Clear();
    this.TitulosClientes();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message + ex.StackTrace);
}
```

Capa Negocios – Listar Clientes

```
public static DataTable Listar()
{
    ClienteDatos objCategoria = new ClienteDatos();
    return objCategoria.Listar();
}
```

Capa Datos – Listar Clientes

```
sqlCnx = Conexion.getInstancia().EstablecerConexion();

SqlCommand comando = new SqlCommand("usp_Cliente_S", sqlCnx);
comando.CommandType = CommandType.StoredProcedure;
sqlCnx.Open();
resultado = comando.ExecuteReader();
tabla.Load(resultado);
return tabla;
```

Clientes – Listar con EF y LINQ

```
using (BAE_RestauranteEntities db = new BAE_RestauranteEntities())
{
    var lst = from d in db.Cliente
              select new {d.id_cliente, d.num_documento, d.tipo_documento, d.nombres, d.apellidos, d.direccion,
                d.celular };

    dgvEmpleados.DataSource = lst.ToList();
}
txtDNI.Clear();
txtNombres.Clear();
txtApellidos.Clear();
txtTipo.Clear();
txtDireccion.Clear();
txtCelular.Clear();
txtIdCliente.Clear();
this.TitulosClientes();
```

Clientes – Eliminar sin ORM

Capa Presentación – Eliminar Clientes

```
string Rpta = "";
Rpta = ClienteNegocios.Eliminar(Convert.ToInt32(txtIdCliente.Text));
if (Rpta.Equals("OK"))
{
    this.MensajeCorrecto("Cliente eliminado correctamente.");
    this.ListarClientes();
}
else
{
    this.MensajeError(Rpta);
}
```

Capa Negocios – Eliminar Clientes

```
public static string Eliminar(int Id)
{
    ClienteDatos objCategoria = new ClienteDatos();
    return objCategoria.Eliminar(Id);
}
```

Capa Datos – Eliminar Clientes

```
1 referencia | BrianAnco, Hace 114 días | 1 autor, 1 cambio
public string Eliminar(int Id)
{
    string Rpta = "";
    SqlConnection sqlCnx = new SqlConnection();
    try
    {
        sqlCnx = Conexion.getInstancia().EstablecerConexion();
        SqlCommand comando = new SqlCommand("usp_Cliente_D", sqlCnx);
        comando.CommandType = CommandType.StoredProcedure;
        comando.Parameters.Add("@pid_cliente", SqlDbType.Int).Value = Id;
        sqlCnx.Open();
        Rpta = comando.ExecuteNonQuery() == 1 ? "OK" : "No se pudo eliminar el registro.";

    }
    catch (Exception ex)
    {
        Rpta = ex.Message;
    }
    finally
    {
        if (sqlCnx.State == ConnectionState.Open) sqlCnx.Close();
    }
    return Rpta;
}
```

Clientes – Eliminar con EF y LINQ

```
using(BAE_RestauranteEntities db = new BAE_RestauranteEntities())
{
    Cliente iCliente = db.Cliente.Find(Convert.ToInt32(txtIdCliente.Text));
    db.Cliente.Remove(iCliente);
    db.SaveChanges();
}

this.MensajeCorrecto("Cliente eliminado correctamente.");
this.ListarClientes();
```

CONCLUSIONES

El uso de la herramienta ORM, permite la correspondencia lógica y natural entre modelo relacional a modelo de objetos. Optimizando recursos de tiempo y rapidez, adicionalmente minimizando los errores debido a que es un proceso automático donde la intervención del programador es menor

BIBLIOGRAFIA

Rodriguez, Gonzalez y Gomez, M. (2007). Estructura de datos, un enfoque moderno. Recuperado de https://elibro.net /es/lc/bibliotecaupt/titulos/62039

Schneider, M. (2015). Data Structures for Databases. Recuperado de https://www.researchgate.net/profile/MarkusSchneider-3/publication/241149187DataStructuresforDatabases/links/54a56d860cf257a63608cebf /Data−Structures− for − Databases.pdf