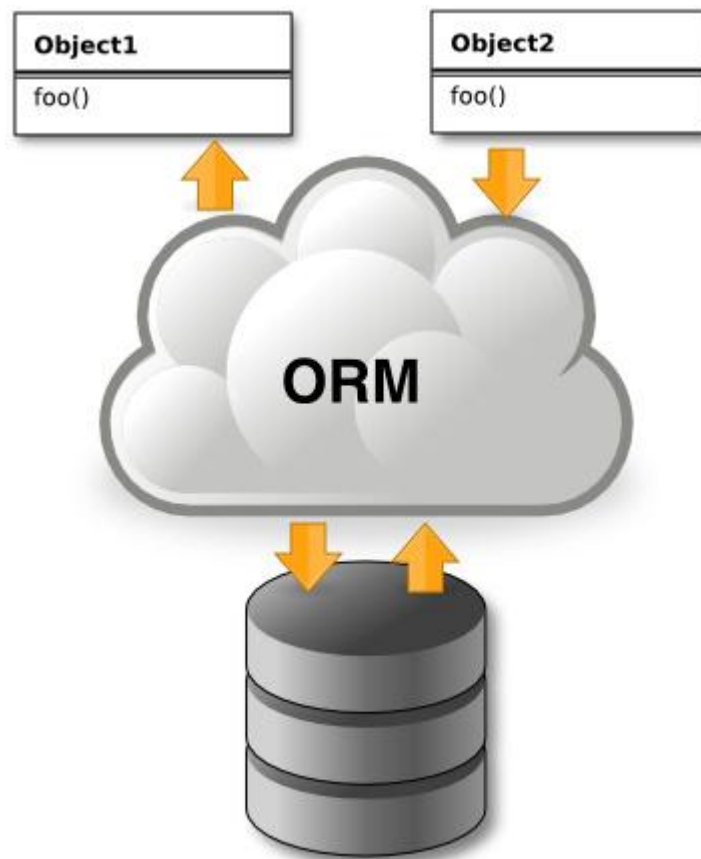


Laboratorio 3 – Mapeo objeto-relacional

¿Qué es ORM?

El mapeo objeto-relacional (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas O/RM, ORM, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).



Si alguna vez has programado alguna aplicación que se conecta a una base de datos, habrás podido comprobar lo laborioso que es transformar toda la información que recibes de la base de datos, principalmente en tablas, en los objetos de tu aplicación y viceversa. A esto se le denomina mapeo. Utilizando un ORM este mapeo será automático, es más, será independiente de la base de datos que estés utilizando en ese momento pudiendo cambiar de motor de base de datos según tus necesidades. Veamos un ejemplo. Supongamos que tenemos una tabla de

clientes. En nuestra aplicación queremos hacer las funciones básicas sobre base de datos CRUD (del inglés Create, Read, Update and Delete) Crear, Obtener, Actualizar y Borrar. Cada operación corresponde con una sentencia SQL.

- Crear: INSERT
- Obtener: SELECT
- Actualizar: UPDATE
- Borrar: DELETE

Si queremos insertar un cliente nuevo y no utilizamos un ORM, el código quedaría de la siguiente manera si utilizamos C#:

```
String query = "INSERT INTO clientes (id,nombre,email,pais) VALUES (@id, @nombre, @email, @pais)";

command.Parameters.AddWithValue("@id","1")
command.Parameters.AddWithValue("@nombre","nombre")
command.Parameters.AddWithValue("@email","email")
command.Parameters.AddWithValue("@pais","pais")

command.ExecuteNonQuery();
```

En cambio si utilizamos un ORM, el código se puede reducir de la siguiente manera:

```
var cliente = new Cliente();

cliente.Id = "1";
cliente.Nombre = "nombre";
cliente.Email = "email";
cliente.Pais = "pais";

session.Save(customer);
```

Como se puede ver se ha reducido considerablemente el código. Pero lo más importante de todo, bajo mi punto de vista, es que, imaginaros que ahora modificamos la tabla de nuestra base de datos y añadimos un campo más como por ejemplo el apellido de nuestro cliente. En los dos casos, tendríamos que añadir a nuestra clase Cliente la propiedad correspondiente al apellido, pero, si no utilizas un ORM te tocará revisarte todas las sentencias INSERT, SELECT y UPDATE para introducir dicho campo en cada una de ellas. En cambio sí utilizas un ORM, lo único que tendrás que hacer será añadir la propiedad a la clase correspondiente. La sentencia save seguirá siendo la misma, el ORM se encargará de modificar los INSERT, SELECT y UPDATE por ti. Esto se cumple hasta cierto punto. Hay algún ORM como MyBatis, que utiliza

un híbrido entre los dos mundos. En MyBatis, las sentencias SQL están escritas en archivos XML. Esto implica que si modificas la base de datos, también tendrás que modificar las sentencias del XML correspondiente.

Además de lo que hemos visto hasta el momento, gracias a los ORM nos abstraemos del motor de base de datos que estamos utilizando, es decir, si en algún momento queremos cambiar de base de datos, nos resultará sumamente sencillo. En algunos casos con solo cambiar un par de líneas de código, estaremos cambiando de motor de base de datos.

Existen varios ORM en el mercado. Todo dependerá del lenguaje de programación que estemos utilizando y de nuestras necesidades. A continuación, podemos ver los ORM para algunos lenguajes de programación:

- Hibernate (Java)
- MyBatis (Java)
- Ebean (Java)
- NHibernate (.NET)
- MyBatis.NET (.NET)
- Doctrine (PHP)
- Propel (PHP)
- Rocks (PHP)
- Torpor (PHP)
- JPA (Java)
- ADO.NET Entity Framework (C#)
- LINQ to SQL (C# sólo para SQL Server)
su sintaxis es similar a JPA
- peewee (Python)
- Object (Python)
- Sequelize (NodeJs)

Nosotros trabajaremos con ADO.NET Entity Framework (C#)

ADO.NET Entity Framework (C#)

Es un conjunto de API de acceso a datos para el Microsoft .NET Framework, apuntando a la versión de ADO.NET que se incluye con el .NET Framework 3.5. Fue lanzado como actualización separada junto con el Service Pack 1 para el .NET Framework, después del lanzamiento de tanto el .NET Framework 3.5 y el Visual Studio 2008. Una nueva versión del Entity Framework (v 4.0) será liberada junto al Visual Studio 2010 y el .NET Framework 4.0.

Una entidad del Entity Framework es un objeto que tiene una clave representando la clave primaria de una entidad lógica de datastore. Un modelo conceptual Entity Data Model (modelo Entidad-Relación) es mapeado a un modelo de esquema de datastore. Usando el Entity Data Model, el Framework permite que los datos sean tratados como entidades independientemente de sus representaciones del datastore subyacente.

El Entity SQL es un lenguaje similar al SQL para consultar el Entity Data Model (en vez del datastore subyacente). Similarmente, las extensiones del Linq, Linq-to-Entities, proporcionan consultas tipeadas en el Entity Data Model. Las consultas Entity SQL y Linq-to-Entities son convertidas internamente en un Canonical Query Tree que entonces es convertido en una consulta

comprensible al datastore subyacente (ej. en SQL en el caso de una base de datos relacional). Las entidades pueden utilizar sus relaciones, y sus cambios enviados de regreso al datastore.

Para trabajar con Entity Framework, utilizaremos el enfoque Database First, que es el método que nos permite primero crear la base de datos con sus tablas (y otras estructuras) y luego incorporarlas a la aplicación. Esto es necesario cuando la aplicación que realizamos necesita utilizar una base de datos existente. Otro uso que podemos darle es cuando necesitamos crear la base de datos, pero nos es más cómodo realizar la definición de las estructuras de la base de datos directamente con sentencias SQL, y luego importar los resultados en la aplicación (lo cual en muchos casos es muy útil y necesario, ya que nos permite definir las estructuras SQL tal cual las queremos o las necesitamos) y poder aplicar al 100% todos los tips y orientaciones que podemos encontrar en este blog Base de Datos con SQL Server . Cuando hablamos que podemos crear otras estructuras nos referimos a procedimientos almacenados, funciones, vistas etc...

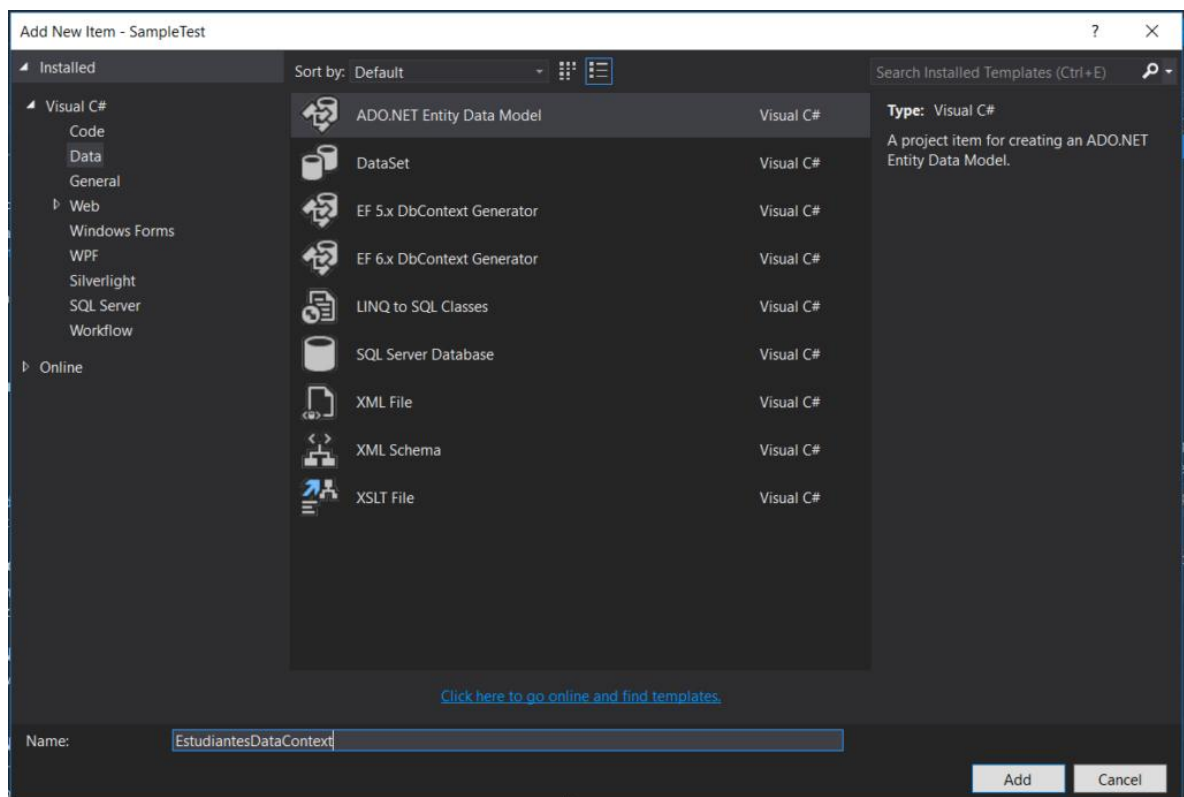
Una vez sabemos esto vamos a ponerlo en práctica para ello vamos a crear una pequeña base de datos de ejemplo de “estudiantes” la vamos a crear en SQL Server. Las tablas generadas serán:

Estudiante

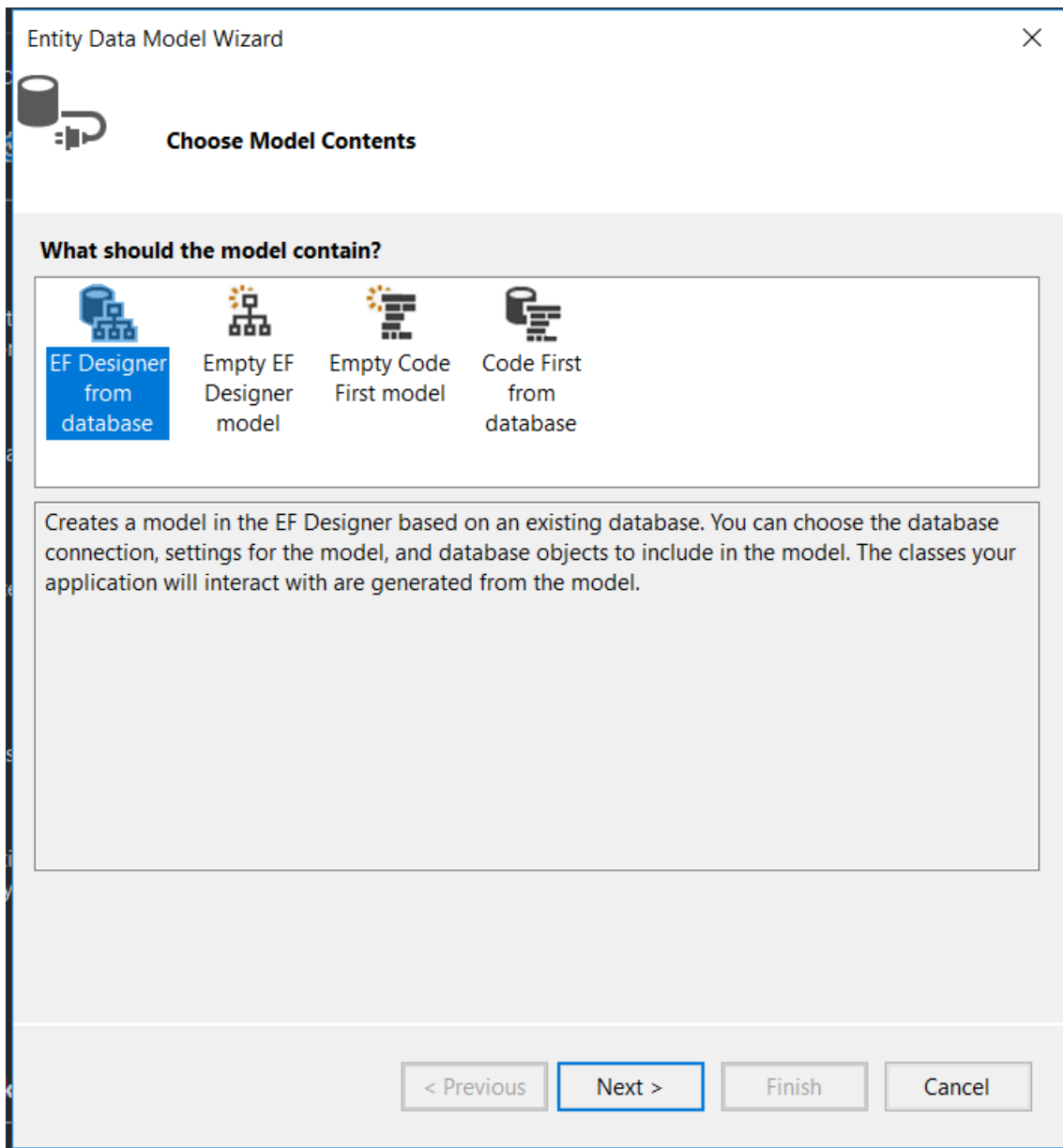
Notas

Materia

Para implementar el Database First en nuestra aplicación es muy sencillo, lo primero que vamos a hacer es agregar el archivo de Entity Framework a nuestra aplicación. Debemos agregar un nuevo elemento de tipo “ADO.NET Entity Data Model”, como se muestra en la imagen. En este caso lo nombraremos EstudiantesDataContext.




Una vez que presionemos “Agregar”, seleccionamos la opción “Generar desde la base de datos”. En un caso normal, no deberíamos tener la cadena de conexión configurada en nuestra aplicación, por lo que daremos click en “Nueva conexión...”



Entity Data Model Wizard

×



Choose Your Data Connection

Which data connection should your application use to connect to the database?

New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

☐ No, exclude sensitive data from the connection string. I will set it in my application code.

☐ Yes, include the sensitive data in the connection string.

Connection string:

☒ Save connection settings in Web.Config as:

< Previous

Next >

Finish

Cancel

Como se ve en la pantalla anterior, debemos ingresar el nombre del servidor, datos de autenticación en caso de que sean necesarios y la base de datos que queremos utilizar. Una vez que presionamos “Aceptar” tendremos la confirmación de la cadena de conexión a utilizar.

Connection Properties



Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:

Microsoft SQL Server (SqlClient)

Change...

Server name:

YHORBYMATIA3D94

Refresh

Log on to the server

Authentication: Windows Authentication

User name:

Password:

☐ Save my password

Connect to a database

☒ Select or enter a database name:

estudiantes

☐ Attach a database file:

Browse...

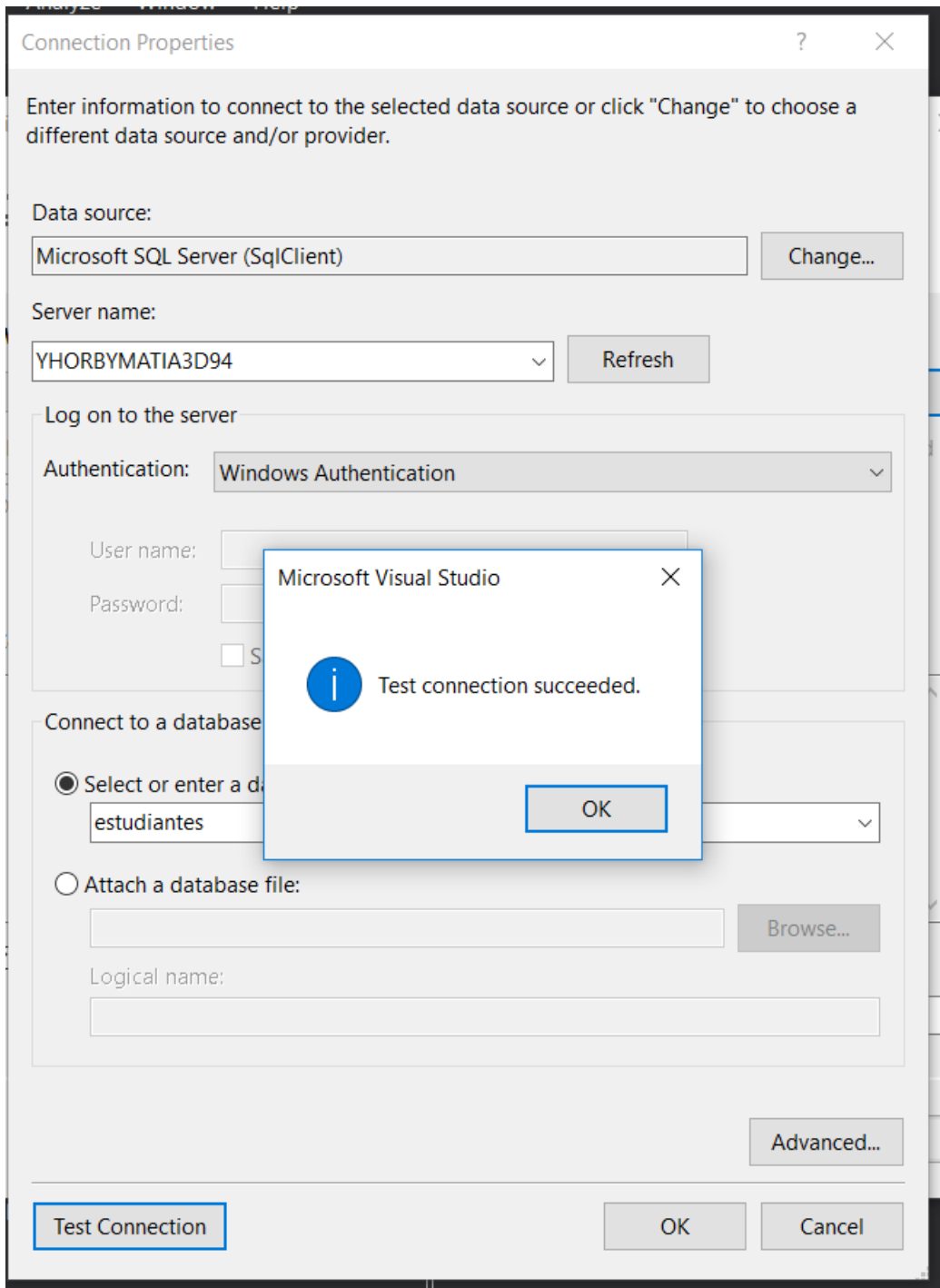
Logical name:

Advanced...

Test Connection

OK

Cancel

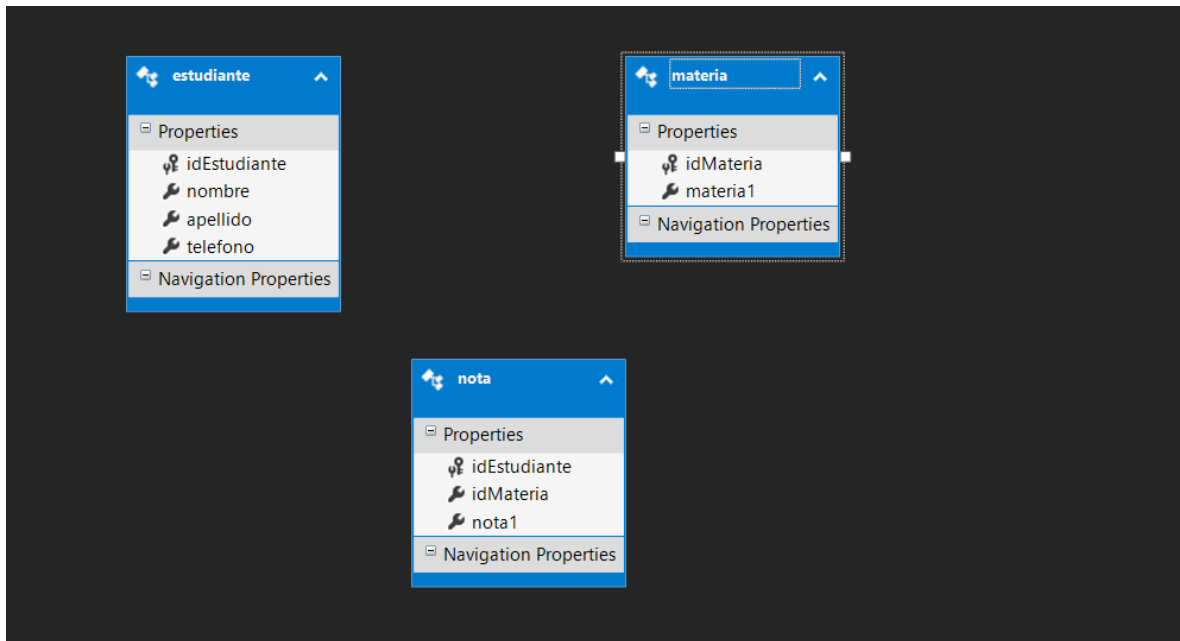


El siguiente paso es seleccionar las estructuras de SQL que queremos incluir en nuestra aplicación. Para incluirlas, solo debemos seleccionar el check correspondiente.

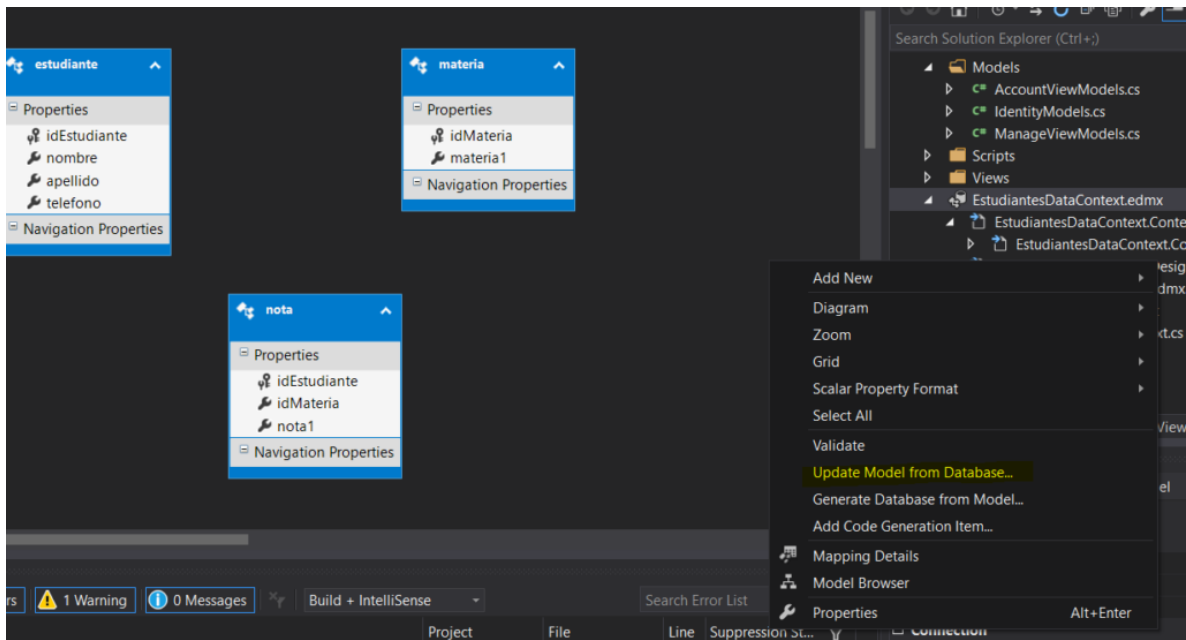
Si trabajamos con nombres de tablas en ingles (este no es el caso del ejemplo), el asistente nos ofrece la opción de poner los nombres de las clases en singular, aunque la

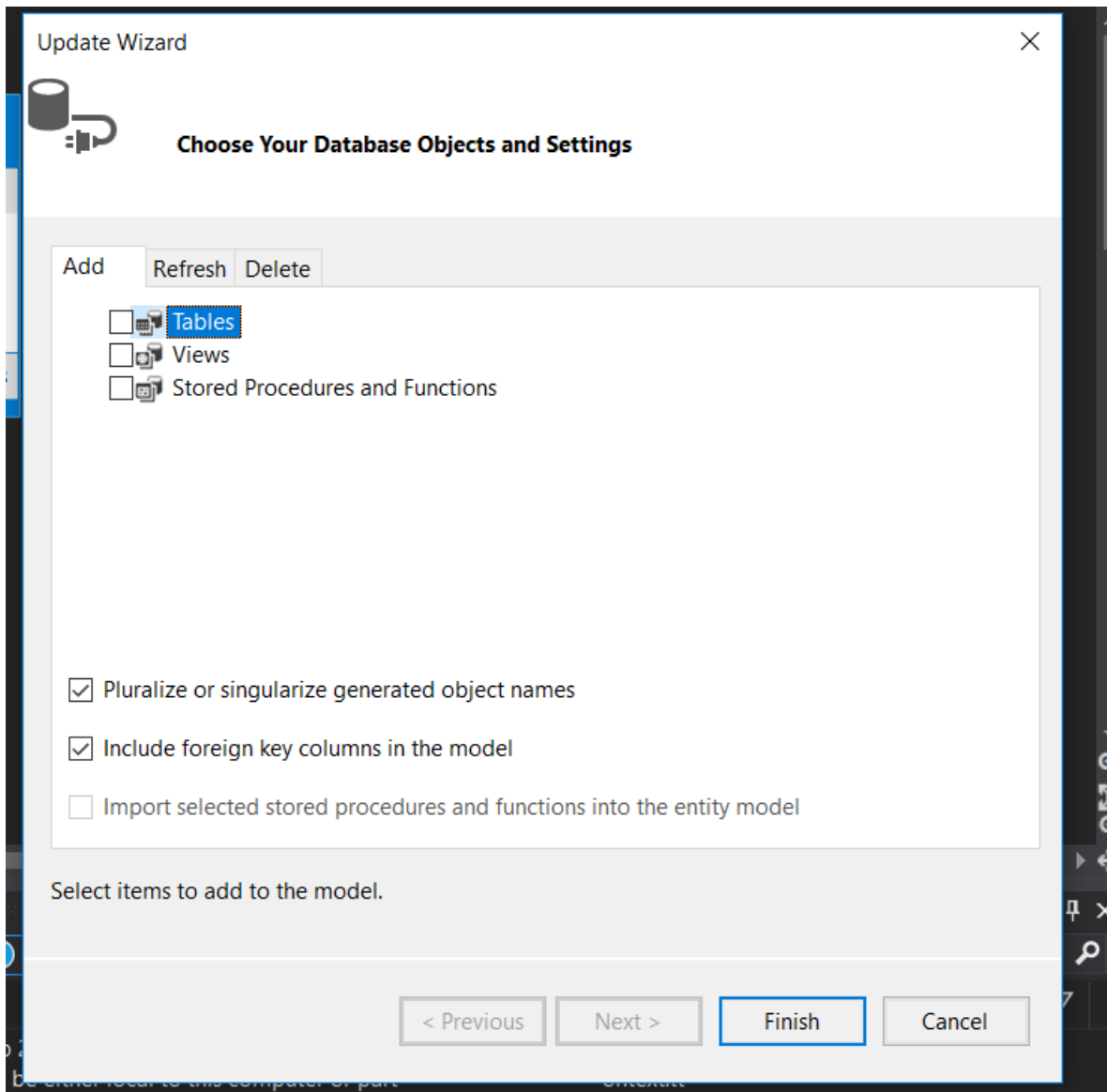
tabla esté en plural permitiéndonos que el código resultante quede mucho más limpio y representativo de lo que se está modelando con el mismo.

Al presionar “Aceptar” de esa ventana se hará el mapeo de las estructuras en el DataContext. Para nuestro ejemplo el resultado es el siguiente:



Una vez realizado esto ya podremos utilizar estas clases contenidas en el DataContext. Esto nos permitirá interactuar directamente con nuestra base de datos. Una de las grandes ventajas que presenta este enfoque es la facilidad de mantenimiento que ofrece. En caso de que haya algún cambio en las estructuras, solo debemos abrir el archivo del DataContext (con extensión edmx), hacer click derecho sobre una zona sin clases y seleccionar la opción “Actualizar modelo desde base de datos...”. Allí nos aparecerá una ventana idéntica a la de que usamos cuando agregamos las estructuras en primer término, pudiendo seleccionar que clases actualizar (tanto porque sean nuevas las tablas, se hayan modificado columnas, o se haya eliminado la tabla):



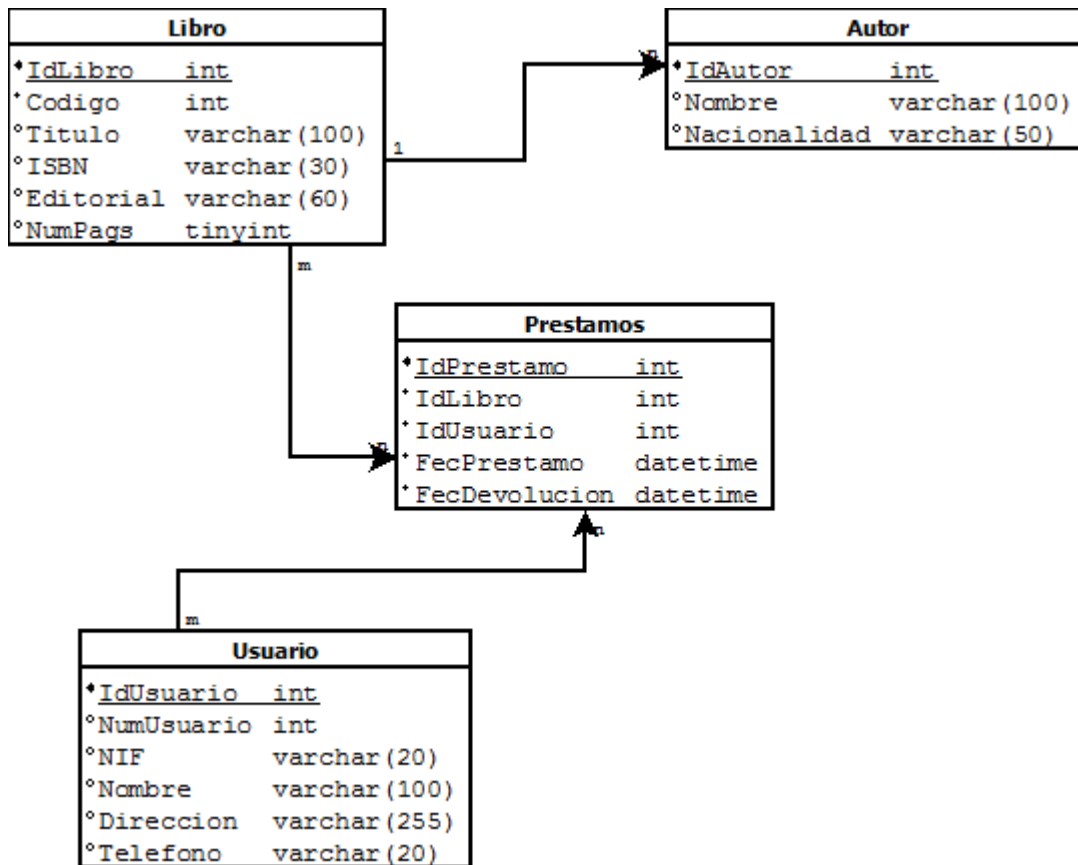


Al Finalizar ya tendremos nuestras tablas mapeadas como objetos dentro de nuestro proyecto, Ahora solo se deben utilizar como Modelos dentro de nuestro patrón de diseño MVC.

Desarrollo del Laboratorio

Paso 1 (20 puntos). -

Crear una base de datos en SQL Server, que tengas las tablas



Paso 2(20 puntos). –

Mapear la base de datos relacional creada en el paso 1, como Objetos dentro de un proyecto WEB MVC (ASP NET Web Application) , como se mostro en el ejemplo de este laboratorio.

Paso 3 (20 puntos).-

Crear las siguientes funciones dentro del proyecto web.

- Agregar Usuario, debe abrir un formulario que permita ingresar todos los datos de un usuario.
- Listar Usuarios, se debe listar todos los usuarios activos. En esta lista se debe permitir editar y eliminar usuarios.

- Listar todos los libros, separar en dos listas los disponible y no disponible por que están prestados.
- Ver para un libro específico la fecha de préstamo, la fecha de devolución, el nombre, dirección y teléfono del usuario a quien se le presto.
- Prestar libro, se debe seleccionar desde la lista de libros disponible la opción prestar y se debe abrir un formulario que tenga un combo box que liste todos los usuarios, al seleccionar el usuario se deben cargar los datos del usuario (Nombre, Dirección y Teléfono)
- Devolver Libro, se debe seccionar desde la lista de libros no disponible (libros prestados) el libro que se esta devolviendo y debe tener la opción de devolución. Lo cual , solo realizara la acción dentro de la base de datos que cambien la marca de actualización.