Capítulo 4
Implementación de acceso a datos con Entity Framework

Objetivo

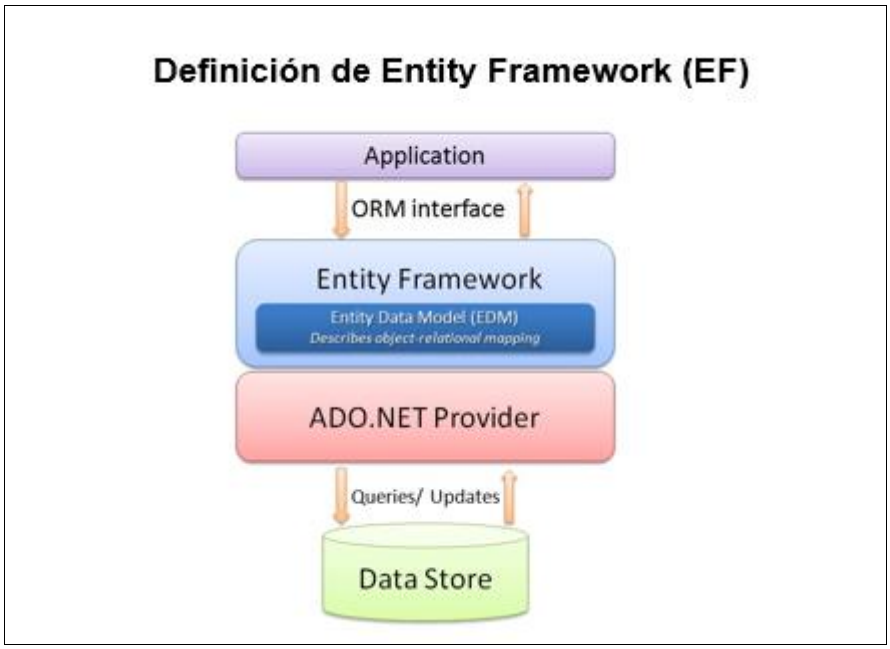
Al finalizar el capítulo, el alumno:

- Comprender el uso de un ORM (EF).
- Realizar operaciones de acceso a datos y manipulación de objetos con Entity Framework.
- Entender el lenguaje integrado de consultas LINQ.
- Realizar operaciones de acceso a datos y manipulación de objetos con LINQ.

Temas

1. Definición de Entity Framework (EF)
2. Uso del enfoque Database First
3. Uso del enfoque Code First
4. Definición de LINQ
5. Variantes de LINQ
    - LINQ to Objects
    - LINQ to Entities
    - LINQ to XML

1. Definición de Entity Framework



Un ORM (Object Relational Mapper) es un producto tecnológico orientado a tratar con el problema del Impedance Mismatch, este problema básicamente describe que el modelo relacional que poseen las bases de datos es diferente al modelo orientado a objetos que se maneja en muchos lenguajes de programación como los que Microsoft ofrece. Básicamente un ORM "traduce" el modelo relacional hacia el modelo orientado a objetos, de tal manera que podemos consultar a los objetos como si estos tuvieran la data que necesitamos, sin embargo, el ORM se encargará de generar y ejecutar las consultas sobre el modelo relacional para cargar la información necesaria en dichos objetos.

Entity Framework (en su versión 6) es un ORM el cual está diseñado para permitir a los programadores crear aplicaciones de acceso a datos, programando con un modelo de la aplicación conceptual en lugar de programar directamente con un esquema de almacenamiento relacional.

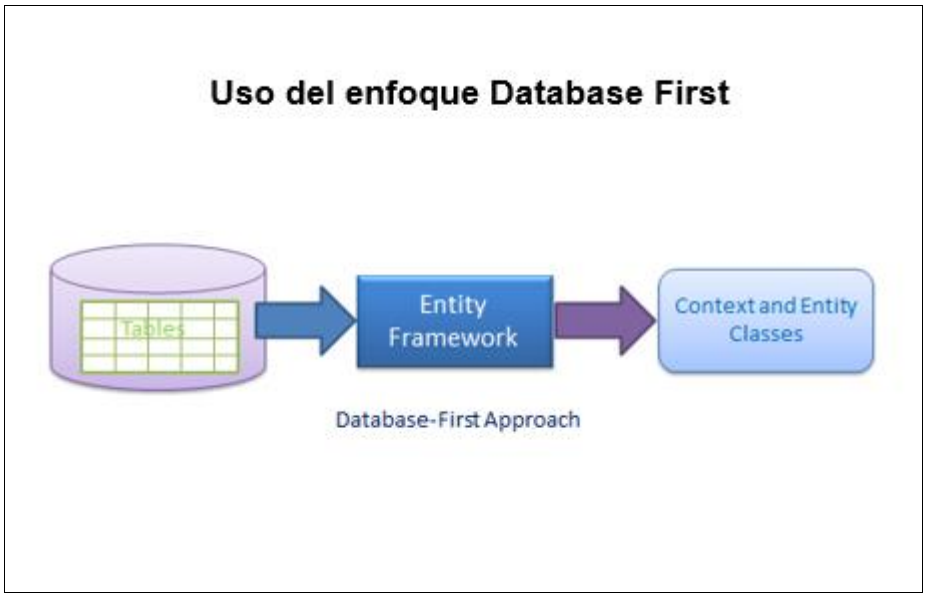
El objetivo es reducir la cantidad de código y mantenimiento que se necesita para las aplicaciones orientadas a datos. Las aplicaciones de Entity Framework ofrecen las siguientes ventajas:

- Las aplicaciones pueden funcionar en términos de un modelo conceptual más centrado en la aplicación, que incluye tipos con herencia, miembros complejos y relaciones.
- Las aplicaciones están libres de dependencias de codificación rígida de un motor de datos o de un esquema de almacenamiento.

- Las asignaciones entre el modelo conceptual y el esquema específico de almacenamiento pueden cambiar, sin tener que cambiar el código de la aplicación.

- Los programadores pueden trabajar con un modelo de objeto de aplicación coherente que se puede asignar a diversos esquemas de almacenamiento, posiblemente implementados en sistemas de administración de base de datos diferentes.
- Se pueden asignar varios modelos conceptuales a un único esquema de almacenamiento.
- La compatibilidad con Language-Integrated Query (LINQ) proporciona validación de la sintaxis en el momento de la compilación para consultas en un modelo conceptual.

2. Uso del enfoque Database First

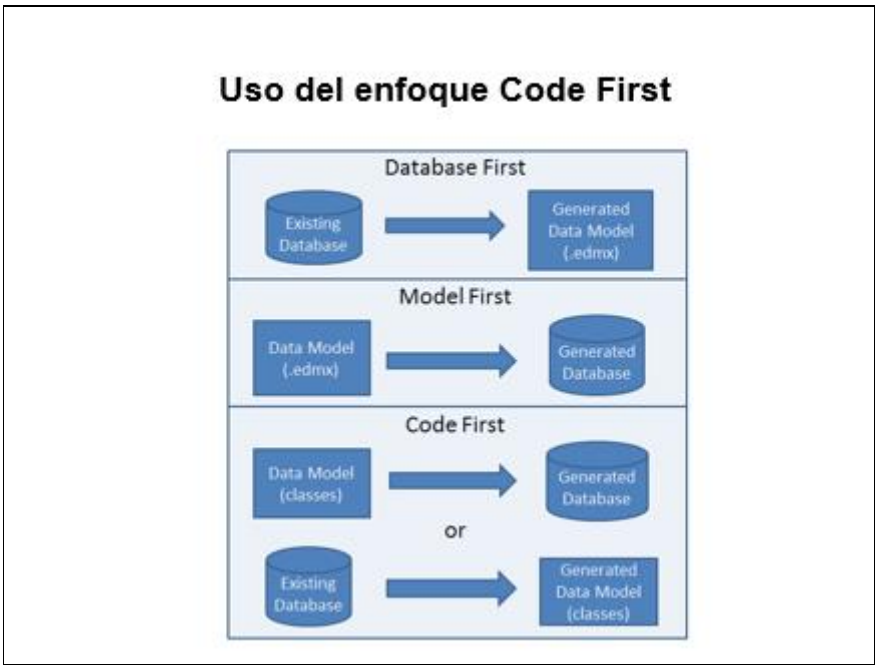


Database First es el modo de trabajo más sencillo que presenta Entity Framework y también el que es menos flexible.

Con Database First nos ocupamos de modelar y crear nuestra base de datos primero o tomar un modelo de base datos que ya exista. A partir de esa base de datos podemos usar los asistentes/wizards que tiene incorporado EF (Entity Framework) para que se aplique ingeniería inversa sobre esa base de datos y se genere un "modelo de entidades/modelo conceptual" (un archivo XML el cual puede ser editado visualmente y tiene extensión .edmx), el cual contiene nuestras tablas de base de datos mapeado a clases/entidades haciendo conversiones automáticas del modelo relacional al modelo orientado a objetos.

A partir de ahí podemos utilizar el objeto DbContext para poder acceder a estas entidades generadas y conseguir la información que necesitamos con consultas de LINQ.

3. Uso del enfoque Code First



Code First aparece en el Entity Framework a partir de la versión 4.1, nos permite crear nuestro modelo mediante clases POCO (Plain Old CLR Object), a partir de las cuales se generará nuestra base de datos. En este modelo no existe el archivo .edmx de definición del modelo.

Code First permite:

- Desarrollar sin tener que abrir un diseñador o definir un archivo de asignación XML.
- Definir objetos de modelo simplemente escribiendo clases “POCO".

Convenciones de Code First:

- **Clave principal**
  Code First infiere que una propiedad es una PK si la propiedad se llama 'Id' o 'Id<NombreClase>'. El campo definido como clave principal se registra en la base de datos como identity por defecto.

- **Relación entre dos tipos**
  Cuando se define una relación entre dos tipos, es común incluir una propiedad de navegación de ambos tipos, tal como en el ejemplo siguiente:

```
public class Product
{
    public int ProductId { get; set; }
    public string Name { get; set; }
```

```
    public Category Category { get; set; }
}

public class Category
{
    public int CategoryId { get; set; }
    public string Name { get; set; }
    public ICollection<Product> Products { get; set; }
}
```

- **Las claves externas**
  Basándose en el convenio anterior, también es común que incluya una propiedad de clave externa en el extremo dependiente de una relación, en este caso BookReview.SubjectISBN:

```
public class BookReview
{
    public int Id { get; set; }
    public Book Subject { get; set; }
    public string SubjectISBN { get; set; }
}

public class Book
{
    [Key]
    public string ISBN { get; set; }
    public string Name { get; set; }
    public ICollection<BookReview> Reviews { get; set; }
}
```

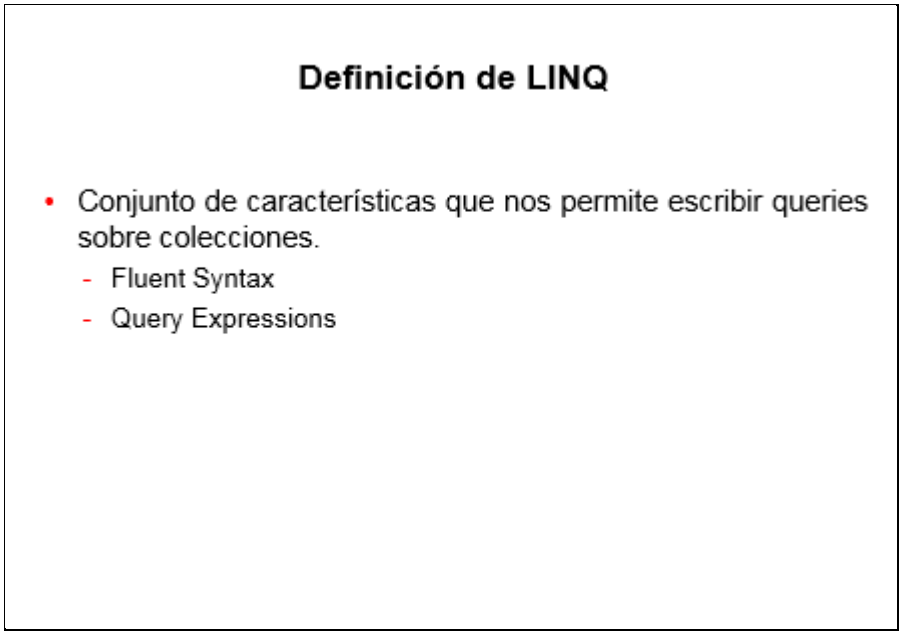
Estas convenciones y otras se pueden modificar a través del Fluent API.

```
//Configurando un PK
modelBuilder.Entity<OfficeAssignment>().HasKey(t => t.InstructorID);

//Configurando un FK (con un nombre personalizado)
modelBuilder.Entity<Course>()
        .HasRequired(c => c.Department)
        .WithMany(d => d.Courses)
        .HasForeignKey(c => c.SomeDepartmentID);

//Especificando la longitud de una propiedad
modelBuilder.Entity<Department>().Property(t => t.Name).HasMaxLength(50);
```

4. Definición de LINQ



LINQ es un conjunto de características dentro del Framework que nos permite escribir queries sobre colecciones de objetos y fuentes de datos remotos. LINQ apareció con el .NET Framework 3.5.

LINQ permite consultar cualquier colección que implemente IEnumerable<T> pudiendo ser esta un array, lista, XML DOM o fuentes de datos como tablas en SQL Server.

En este ejemplo, se muestra un array con varios nombres, el cual será filtrado de acuerdo a la longitud de los nombres:

```
string[] names = { "Tom", "Dick", "Harry" };
IEnumerable<string> filteredNames = names.Where (n => n.Length >= 4);
foreach (string name in filteredNames) Console.WriteLine (name);

//Resultado:
Dick
Harry
```

Tipos de Sintaxis en LINQ:

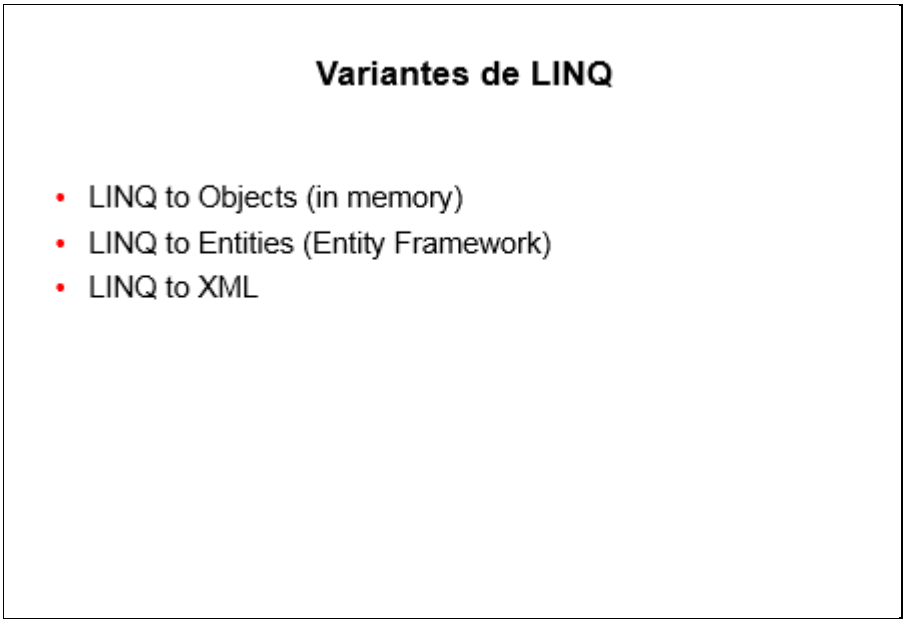
- **Fluent Syntax**: esta sintaxis es la más básica y flexible manera de armar queries está basada en método y method chaining:

```
IEnumerable<string> query = names
.Where (n => n.Contains ("a"))
.OrderBy (n => n.Length)
.Select (n => n.ToUpper());
```

- **Query Expressions**: esta sintaxis tiene cierta semejanza con el estándar T-SQL y permite realizar queries más complejas:

```
IEnumerable<string> query =
from n in names
where n.Contains ("a")
orderby n.Length
select n.ToUpper();
```

5. Variantes de LINQ



LINQ to Objects es el conjunto de métodos de IEnumerable<T> que permite realizar operaciones de consulta de objetos que se encuentran en memoria. La sintaxis para la consulta de estos objetos no varía en lo absoluto. El uso de LINQ to Objects simplifica en gran manera el manejo de colecciones en nuestras aplicaciones.

LINQ to Entities presenta un conjunto de métodos en IQueryable<T>. Los métodos construyen un expression tree y el provider toma ese expression tree y construye queries de SQL.

LINQ to XML se utiliza para recorrer los nodos de un objeto que representa un XML para extraer rápidamente elementos y atributos del mismo.