educación IT
CODE YOUR FUTURE

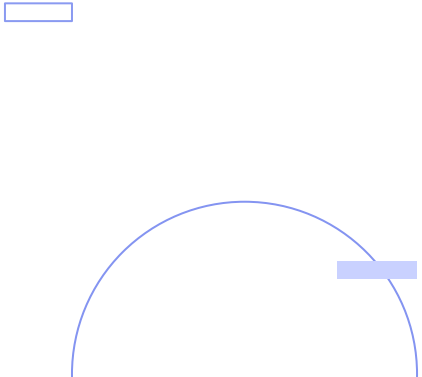
Programación Web .NET Core

Módulo 8

Trabajar con Entity Framework Core

Trabajar con Entity Framework Core

Se puede acceder a una base de datos sin la necesidad de envoltorios o abstracciones utilizando un ***contexto de Entity Framework***, en EF Core. **Un contexto es el gestor entre clases de modelo de dominio y la base de datos**.



Un contexto de Entity Framework se usa para:

- Proporcionar creación, lectura, actualización, borrar operaciones (CRUD) y simplificar el código que es necesario escribir para ejecutar estas operaciones.
- Manejar la apertura y cierre de conexiones a bases de datos.
- Manejar la generación de la base de datos cuando se trabaja con el enfoque CodeFirst.

Agregar un contexto (*DbContext*) de Entity Framework

Para coordinar con las funcionalidades de Entity Framework, tendrás que crear un contexto de Entity Framework. **Un contexto (`DbContext`) en Entity Framework Core es una clase que hereda de la clase `DbContext`**. Puedes usar esta clase cuando desees acceder a la base de datos.

El ejemplo de la derecha, demuestra cómo crear una clase personalizada que se deriva de la clase `DbContext`:

Una clase derivada de la clase DbContext

```
publicclassHrContext :DbContext
{
}
```

Clases de modelos de dominio (entidades)

Además de crear un contexto de Entity Framework, también debes crear las **clases modelo del dominio**. Las clases de modelo de dominio son clases de ***"objetos CLR simples y antiguos"*** (**POCO**). Simplemente definen las propiedades de los datos que se almacenan en la base de datos. **Las clases del modelo de dominio se denominan *entidades***.

En tu código, deberás especificar qué entidades están incluidas en el modelo de datos y también podrás personalizar el comportamiento de cada una de ellas.

El siguiente código muestra una clase de entidad:

```
publicclassPerson
{
public int PersonId{ get; set; }
public string FirstName { get; set; }
public string LastName{ get; set; }
}
```

Agregar propiedades de *DbSet* a un contexto de Entity Framework

Al trabajar con EF Core, las entidades definidas en el código deben estar asignadas a las tablas definidas en la base de datos. Para lograr esto, puedes agregar las entidades en propiedades **`DbSet<>`**. Cada propiedad **`DbSet<>`** obtiene un tipo de entidad como tipo de parámetro. En bases de datos relacionales, la entidad se asigna a una tabla en la base de datos. EF Core mapeará las propiedades de acuerdo con la información de mapeo proporcionada por las clases.

El siguiente ejemplo de código muestra cómo agregar una propiedad **`DbSet<Person>`** a la clase **`HRContext`**:

```
publicclassHrContext :DbContext
{
publicDbSet<Person>Candidates{ get; set; }
}
```

Conexión de un contexto de Entity Framework a la base de datos SQLite

Para conectar un contexto de Entity Framework a una base de datos, se debe utilizar un **proveedor de base de datos que corresponda a la base elegida**. Una de las bases de datos más populares a las que se puede acceder cuando se usa EF Core es ***SQLite***. SQLite **es una base sin servidor**; no necesitas tener un servidor separado para usarla. Una de sus grandes ventajas es que es **muy fácil de configurar**.

Para usar un contexto de Entity Framework en una aplicación ASP.NET Core, debes registrar un servicio usando el método **`AddDbContext<>`** pasándole el tipo de clase de contexto de Entity Framework como un **parámetro tipo genérico**. Debes registrar el servicio en el método **`ConfigureServices`** de la clase **`Startup`**. Veamos el ejemplo de la próxima pantalla.

El siguiente código muestra cómo configurar **HrContext**, para conectarse a una base de datos SQLite, que será almacenado en un archivo llamado ***example.db***:

```
publicclassStartup
{
public void ConfigureServices(IServiceCollection services)
{
services.AddDbContext<HrContext>(options
=>options.UseSqlite("DataSource=example.db"));
services.AddMvc();
}
public void Configure(IApplicationBuilder app)
{
app.UseMvc();
}
}
```

Nota: El proveedor de la base de datos SQLite se distribuye con el Paquete cuyo nombre es **NuGetMicrosoft.Entity FrameworkCore.Sqlite**.

El siguiente ejemplo demuestra cómo el contexto de Entity Framework puede recibir las opciones especificadas en el método **`ConfigureServices`**:

La clase HrContext

```
public class HrContext :DbContext
{
public HrContext(DbContextOptions<HrContext> options) : base(options)
{
}
public DbSet<Person> Candidates { get; set; }
}
```

Usar un contexto de Entity Framework en un controlador

Después de configurar el contexto de Entity Framework, puedes usarlo en un controlador. La entidad **`contexto`** se pasa al constructor del controlador que estás utilizando debido al mecanismo de inyección de dependencia.

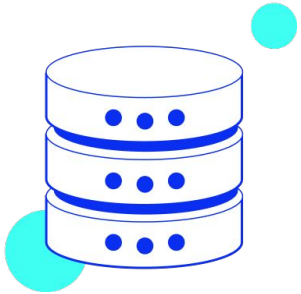
El siguiente código muestra un controlador que obtiene una instancia de **`HrContext`** mediante la **inyección de dependencia**:

```
public class HrController : Controller
{
private HrContext _context;
publicHrController(HrContext context)
{
_context = context;
}
public IActionResultIndex()
{
return View(_context.Candidates.ToList());
}
}
```

Crear y eliminar una base de datos

Se puede usar la propiedad `Database` de un objeto `DbContext` para crear una base de datos o eliminarla.

Usando la propiedad `Database` del objeto `DbContext`, se puede llamar a **`SecureCreated`**. Cuando se llama al método **`Created`**, Entity Framework crea una base de datos basada en la información de su clase derivada de `DbContext` si la base de datos aún no existe. Cuando llama al método **`GuaranteeDeleted`**, Entity Framework elimina la base de datos si ya existe.



El siguiente código muestra cómo eliminar una base de datos si existe mediante el método **SecureDeleted**, y cómo crear una nueva base de datos mediante el método **SecureCreated**:

```
public void Configure(IApplicationBuilder app, HrContextctx)
{
ctx.Database.EnsureDeleted();
ctx.Database.EnsureCreated();
app.UseMvc();
}
```

Cuando se llama al método `SecureCreated`, se creará una nueva base de datos y se inicializará con los datos de muestra.

El siguiente código muestra cómo sembrar datos mediante el método **OnModelCreating**:

```
publicclassHrContext :DbContext
{
public HrContext(DbContextOptions<HrContext> options) : base(options)
{
}
public DbSet<Person> Candidates { get; set; }
protected override void OnModelCreating(ModelBuildermodelBuilder)
{
modelBuilder.Entity<Person>().HasData(
new { PersonId = 1, FirstName = "James", LastName = "Smith"},
new { PersonId = 2, FirstName = "Arthur", LastName = "Adams"}
);
}
}
```

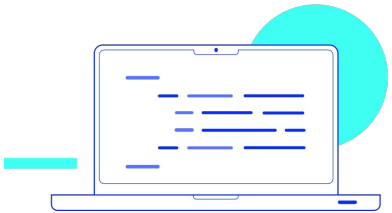
Usando LINQ to Entities

LINQ es un conjunto de métodos de extensión que permiten escribir expresiones de consulta complejas. Mediante estas expresiones, es posible extraer datos de las bases de datos, objetos enumerables, documentos XML, y otras fuentes de datos. Las expresiones **son similares a las consultas *Transact-SQL***, pero se puede obtener compatibilidad con ***IntelliSense*** y comprobación de errores en Visual Studio.

¿Qué es LINQ to Entities?

***LINQ to Entities* es la versión de LINQ que funciona con EF Core. Permite escribir consultas complejas y sofisticadas para localizar datos específicos, unir datos de múltiples objetos y realizar otras acciones en objetos del contexto de Entity Framework**. Si estás utilizando EF Core, puedes escribir consultas LINQ siempre que necesites una instancia específica de una clase de modelo, un conjunto de objetos o para necesidades de aplicaciones más complejas.

Puedes escribir consultas LINQ. Son similares a la **sintaxis SQL basada en métodos**, en la que las operaciones como **"`select`"** son métodos definidos en objetos.

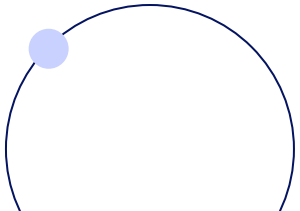


La consulta LINQ

Es una consulta específica que deseas obtener de tu base de datos. Se puede especificar cómo se ordenará, agrupará o formateará la información recuperada. La consulta LINQ se almacena en una variable de consulta y se inicializa con la expresión de consulta. La escritura de la consulta LINQ es ligeramente diferente a escribir una consulta SQL.

El siguiente ejemplo muestra cómo recuperar una lista de personas de la base de datos y filtrarla por el nombre de una persona:

```
publicIActionResultIndex()
{
var list = from c in _context.Candidates
where c.LastName == "Smith"
select c;
return View(list);
}
```



Carga de datos relacionados

En EF Core puedes cargar entidades relacionadas usando propiedades de navegación.

Propiedades de navegación

En EF Core puedes vincular una entidad a otras entidades mediante las *propiedades de navegación*. Cuando una entidad está relacionada a otra, debes agregar una propiedad de navegación para representar la asociación. Cuando la multiplicidad de una asociación es 1 ó 0 a 1, la propiedad de navegación está representada por una referencia ***objeto***. Cuando la multiplicidad
de la asociación es muchas, la propiedad de navegación está representada por una ***colección***. El ejemplo de código de la siguiente pantalla, muestra una entidad denominada **`Country`**. Dado que la entidad `Country` participa en una relación uno a muchos con la entidad **`Ciudad`**, contiene una propiedad de navegación denominada: **`Cities`** de tipo **`List <City>`**. De manera similar, dado que la entidad Country participa en una relación de uno a muchos con la entidad `Person`, contiene una propiedad de navegación denominada: **`Person`** de tipo **`List <Person>`**.

Una entidad denominada Country con propiedades de navegación

```
public class Country
{
public int CountryId{ get; set; }
public string Name { get; set; }
public int Size { get; set; }
public List<City> Cities { get; set; }
public List<Person> People { get; set; }
}
```

El siguiente ejemplo de código muestra una entidad denominada **City**. Dado que esa entidad participa en una relación con la entidad **Person**, contiene una propiedad de navegación denominada **People** de tipo **List <Person>**.

Dado que cada ciudad pertenece exactamente a un país, la entidad **City** contiene una propiedad denominada **Country**. La propiedad **CountryId** representa una clave externa:

Una entidad denominada City con propiedades de navegación

```
public class City
{
public int CityId{ get; set; }
public string Name { get; set; }
public int Size { get; set; }
public int CountryId{ get; set; }
public Country Country{ get; set; }
public List<Person>People { get; set; }
}
```

El siguiente ejemplo de código muestra una entidad denominada **Person** con propiedades de navegación al país y entidades de la ciudad:

Una entidad denominada Person con propiedades de navegación

```
public  class Person
{
public int PersonId{ get; set; }
public string FirstName { get; set; }
public string LastName{ get; set; }
public int CountryId{ get; set; }
public Country Country{ get; set; }
public int CityId{ get; set; }
public City City{ get; set; }
}
```

El siguiente ejemplo muestra un contexto de Entity Framework que contiene colecciones de las entidades:

La clase DemographyContext

```
public class DemographyContext :DbContext
{
public DemographyContext(DbContextOptions<DemographyContext> options) : base(options)
{
}
public DbSet<Country> Countries { get; set; }
public DbSet<City> Cities { get; set; }
public DbSet<Person> People { get; set; }
}
```

Manipular datos mediante Entity Framework

EF Core puede rastrear entidades de las que recupera la base de datos. EF Core usa el **seguimiento de cambios**, por lo que cuando llamas al método **SaveChanges** en el objeto de contexto de Entity Framework, puedes **sincronizar sus actualizaciones con la base de datos**. **Es posible comprobar el estado de cualquier entidad**. Inspecciona el historial de sus cambios y deshaz los cambios si es necesario.

Cada entidad de EF Core puede estar en uno de los siguientes estados:

- **Added:** La entidad fue agregada al contexto y no existe en la base de datos.
- **Modified:** La entidad se cambió desde que se recuperó de la base de datos.
- **Unchanged:** La entidad no se modificó desde que se recuperó de la base de datos.
- **Deleted:** La entidad se eliminó desde que se recuperó de la base de datos.

Insertar nuevas entidades

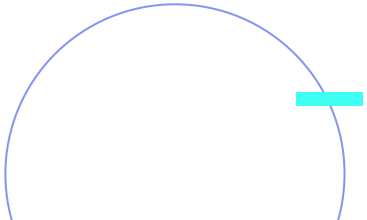
Para agregar una entidad a la base de datos, puedes usar el objeto de contexto de Entity Framework.

El contexto marca el estado de seguimiento de cambios de la entidad como ***Added***. **Cuando se llama al método SaveChanges, el contexto de Entity Framework agrega la entidad a la base de datos**. No se aplican cambios a la base de datos hasta que se llame al método cuyo nombre es `SaveChanges`.

El ejemplo de código de la derecha muestra una entidad denominada `Person`.

Una entidad

```
public  class  Person
{
public int PersonId{ get; set; }
public string FirstName { get; set; }
public string LastName{ get; set; }
}
```



El siguiente ejemplo de código muestra cómo agregar una nueva entidad a una base de datos utilizando Entity Framework con el objeto de contexto:

Agregar una entidad

```
public  IActionResultCreate()
{
_context.Add(new Person() { FirstName = "Nathan ", LastName = "Owen" });
_context.SaveChanges();
return RedirectToAction(nameof(Index));
}
```

Eliminar entidades

Para eliminar una entidad de la base de datos, puedes usar el objeto de contexto de Entity Framework. Cuando borras una entidad de una base de datos, el contexto marca el estado de seguimiento de cambios de la entidad como ***Deleted***. Cuando llamas al método **SaveChanges**, el objeto de contexto de Entity Framework elimina la entidad de la base de datos.

El ejemplo de código de la pantalla siguiente, muestra cómo **eliminar una entidad** de una base de datos mediante el uso del **objeto de contexto**.

Eliminar una entidad

```
public IActionResultDelete(int id)
{
var person = _context.People.SingleOrDefault(m =>m.PersonId == id);
_context.People.Remove(person);
_context.SaveChanges();
returnRedirectToAction(nameof(Index));
}
```

Actualizar entidades

Para actualizar una entidad en la base de datos, puedes usar el **objeto de contexto** de Entity Framework. Cuando actualizas una entidad, el estado de seguimiento de cambios de la entidad pasa como ***Modified***. Cuando llamas al **SaveChanges**, el objeto de contexto de Entity Framework actualiza la entidad en la base de datos.

El ejemplo de código de la pantalla siguiente, muestra cómo **recuperar y actualizar una entidad** usando el **objeto de contexto** de Entity Framework.

Actualizar una entidad

```
publicIActionResultEdit(int id)
{
var person = _context.People.SingleOrDefault(m =>m.PersonId == id);
person.FirstName = "Brandon";
_context.Update(person);
_context.SaveChanges();
return RedirectToAction(nameof(Index));
}
```

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!