INTEGRACIÓN CON BASES DE DATOS Y ALMACENAMIENTO

CURSO .NET – IT SCHOOL

iTSCHOOL LA POSTA!

INTRODUCCIÓN A ENTITY FRAMEWORK CORE

INTRODUCCIÓN A ENTITY FRAMEWORK CORE

Entity Framework (EF) Core es una versión ligera, extensible, de código abierto y multiplataforma de la popular tecnología de acceso a datos Entity Framework.

Un ORM (Object Relational Mapper) es una técnica que crea un "puente" entre programas orientados a objetos y, en la mayoría de los casos, bases de datos relacionales. Básicamente, un ORM permite interactuar con bases de datos utilizando objetos en tu lenguaje de programación preferido, los ORMs simplifican la interacción con bases de datos al traducir instrucciones de programación a sentencias SQL comprensibles para el gestor de base de datos.

- Object Relational Mapper: EF Core permite a los desarrolladores de .NET trabajar con una base de datos usando objetos .NET. Esto significa que puedes manipular datos como si fueran objetos en lugar de escribir código de acceso a datos manualmente.
- Modelo de datos: En EF Core, el acceso a datos se realiza mediante un modelo. Un modelo se compone de clases de entidad y un objeto de contexto que representa una sesión con la base de datos. Este objeto de contexto permite consultar y guardar datos.
- Consultas: Las instancias de las clases de entidad se recuperan de la base de datos mediante Language Integrated Query (LINQ).

CONFIGURACIÓN Y CONEXIÓN A BASES DE DATOS

EF Core es compatible con varios proveedores de bases de datos, algunos de ellos:

- **Microsoft SQL Server**: Compatible con SQL Server y Azure SQL Database, por medio de *Microsoft.EntityFrameworkCore.SqlServer.*
- **SQLite**: Admite versiones 3.7 en adelante utilizando *Microsoft.EntityFrameworkCore.Sqlite.*
- **MySQL**: A través del proveedor *Pomelo.EntityFrameworkCore.MySql*.
- **PostgreSQL**: Utiliza el proveedor *Npgsql.EntityFrameworkCore.PostgreSQL*.
- **In-memory database**: Para pruebas y desarrollo, mediante el proveedor *Microsoft.EntityFrameworkCore.InMemory*.

CONFIGURACIÓN Y CONEXIÓN A BASES DE DATOS

Ejemplos de conexiones:

Microsoft SQL Server

```
.UseSqlServer("Server=localhost,1433;Initial Catalog=baseDeDatos;User ID=usuario;Password=contraseña");
```

MySQL

```
.UseMySql("server=localhost;user=usuario;password=contraseña;database=baseDeDatos", serverVersion)
```

MODELADO DE DATOS Y MIGRACIONES

- En EF Core, el acceso a datos se realiza mediante un **modelo**. Este modelo se compone **de clases de entidad** y un **objeto de contexto** que representa una sesión con la base de datos.
- Las **clases de entidad** son representaciones de las tablas en la base de datos. Por ejemplo, si tienes una tabla "Blogs", crearías una clase de entidad llamada *Blog*.
- Cada propiedad en la clase de entidad corresponde a una columna en la tabla. Por ejemplo:

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

MODELADO DE DATOS Y MIGRACIONES

- Puedes configurar el modelo utilizando **Fluent API** o **Data Annotations**.
- **Fluent API** te permite especificar la configuración sin modificar las clases de entidad. Por ejemplo:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .Property(b => b.Url)
        .IsRequired();
}
```

MODELADO DE DATOS Y MIGRACIONES

Las migraciones en EF Core son una herramienta esencial para mantener sincronizado el esquema de la base de datos con el modelo de datos de tu aplicación.

1. **¿Qué son las migraciones?**
   - Las migraciones permiten actualizar el esquema de la base de datos de manera incremental.
   - Cuando cambias el modelo de datos (agregas o quitas entidades o propiedades), creas una migración que describe las actualizaciones necesarias.
2. **Pasos clave en las migraciones:**
   - **Generación**: EF Core compara el modelo actual con una instantánea anterior y genera archivos de origen para la migración.
   - **Aplicación**: Puedes aplicar las migraciones a la base de datos.
   - **Historial**: EF Core registra las migraciones aplicadas en una tabla especial.

CONSULTAS Y LINQ

Las consultas en EF Core se usan para recuperar datos de la base de datos.

1. **LINQ y Consultas:**
   - EF Core utiliza **Language Integrated Query (LINQ)** para escribir consultas en C# o el lenguaje .NET preferido.
   - Puedes consultar datos basados en el contexto y las clases de entidad.
2. **Proceso de Consulta:**
   - Creas una representación de la consulta en memoria al usar operadores LINQ.
   - La consulta se envía a la base de datos solo cuando se usan los resultados.
   - Operaciones comunes que envían la consulta a la base de datos incluyen iterar resultados, usar *ToList*, *ToArray*, *Single*, *Count* y equivalentes asincrónicos.

CONSULTAS Y LINQ

Ejemplo de consulta de lectura:

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs.ToList();

    foreach (var blog in blogs)
    {
        Console.WriteLine($"Blog: {blog.Url}");
    }
}
```

CONSULTAS Y LINQ

Ejemplo de consulta de escritura:

```
using (var context = new BloggingContext())
{
    var newBlog = new Blog
    {
        Url = "https://mynewblog.com"
    };

    context.Blogs.Add(newBlog);
    context.SaveChanges();
}
```

TRANSACCIONES Y CONTROL DE CONCURRENCIA

Las transacciones en EF Core sirven para procesar varias operaciones de base de datos de manera atómica.

1. **Comportamiento predeterminado de las transacciones:**
   - Cuando realizas cambios en la base de datos mediante *SaveChanges*, EF Core los aplica en una **transacción**.
   - Si algún cambio falla, la transacción se revierte y ninguno de los cambios se aplica a la base de datos.
   - Esto garantiza que *SaveChanges* se complete correctamente o deje la base de datos sin modificaciones en caso de error.

TRANSACCIONES Y CONTROL DE CONCURRENCIA

2. **Control manual de transacciones:**
   - Puedes usar la API *DbContext.Database* para iniciar, confirmar y revertir transacciones.
   - Ejemplo de uso:

```
using var context = new BloggingContext();
using var transaction = context.Database.BeginTransaction();
try
{
    // Realizar cambios en la base de datos
    context.Blogs.Add(new Blog { Url = "https://mynewblog.com" });
    context.SaveChanges();
    // Otros cambios...
    transaction.Commit(); // Confirmar transacción
}
catch (Exception)
{
    // Manejar errores
}
```

TRANSACCIONES Y CONTROL DE CONCURRENCIA

El control de concurrencia en EF Core garantiza la coherencia de los datos cuando varias instancias de aplicaciones modifican la base de datos simultáneamente.

1. **Simultaneidad optimista:**
   - EF Core implementa la simultaneidad optimista, que asume que los conflictos de simultaneidad son poco frecuentes.
   - A diferencia de los enfoques pesimistas que bloquean datos por adelantado, la simultaneidad optimista no bloquea, sino que verifica si los datos han cambiado desde que se consultaron.
   - Si los datos han cambiado, se notifica a la aplicación para manejarlo (por ejemplo, reintento de operación).
2. **Token de simultaneidad:**
   - Configuras una propiedad como un **token de simultaneidad** en EF Core.
   - Este token se carga y se rastrea al consultar una entidad.
   - Al actualizar o eliminar datos con *SaveChanges()*, se compara el valor del token en la base de datos con el valor original leído por EF Core.

MATERIAL DE ESTUDIO

- Vídeos tutoriales:
  - Curso completo de Entity Framework Core – YouTube
  - Aprende Entity Framework | Entity Framework Core para Principiantes | cuándo y cómo usarlo (youtube.com)
  - CODE FIRST con Entity Framework en .NET 🐊 (youtube.com)
- Documentación y cursos oficiales:
  - Información general de Entity Framework Core - EF Core | Microsoft Learn
  - Entity Framework Core para principiantes | Microsoft Learn