Escuelita Crombie 3er Edición

Fluent API

Forma avanzada de configuración sin utilizar atributos o data-annotations, usando funciones de extensión anidadas en objetos de tabla, columnas durante el mapeo de los datos.

Fluent API permite agregar más complejidad y detalle al diseño de cada componente de la base de datos y aparte ayuda a centralizar todo el diseño del esquema de la BD

Tener en cuenta que FluentAPI va a predominar sobre Data Annotation, por lo que podemos comentar o eliminar los mismos

TareasContext.cs

```
protected override void OnModelCreating(ModelBuilder modelbuilder)
{
        modelBuilder.Entity<Categoria>(categoria => {
                categoria.ToTable("Categoria");
                categoria.HasKey(p=> p.CategoriaId);

                categoria.Porpiety(p => p.Nombre).IsRequired().HasMaxLength(150);

                categoria.Propiety(p => Descripcion)

        });

        modelBuilder.Entity<Tareas>( tareas =>
        {
                tarea.ToTable("Tarea");
                tarea.HasKey(p=> p.TareaId);

                tarea.HasOne(p=> p.Categoria)
                .WithMany(p=> p.Tareas)
                .HasForeignKey(p=> p.CategoriaId);

                tarea.Property(p => p.Titulo).IsRequired().HasMaxLength(200);

                tarea.Property(p=> p.Descripcion);

                tarea.Property(p=> p.PropiedadTarea);
```

```
                tarea.Property(p=> p.FechaCreacion);

                tarea.Ignore(p=> p.Resumen);
        });
}
```

Funciones de Fluent API

1. Configuración de Entidades

- `Entity<TEntity>()`: Configura un tipo de entidad.
- `HasKey()`: Define la clave primaria.
- `HasAlternateKey()`: Define una clave alternativa.
- `ToTable()`: Especifica el nombre de la tabla.
- `Ignore()`: Ignora una propiedad o clase.

---

2. Configuración de Propiedades

- `Property()`: Configura una propiedad específica.
- `HasColumnName()`: Cambia el nombre de la columna.
- `HasColumnType()`: Especifica el tipo de datos de la columna.
- `HasDefaultValue()`: Define un valor predeterminado.
- `HasDefaultValueSql()`: Define un valor predeterminado en SQL.
- `HasComputedColumnSql()`: Configura una columna calculada.
- `IsRequired()`: Indica que una propiedad no acepta valores NULL.
- `HasMaxLength()`: Especifica la longitud máxima de una propiedad string.
- `IsConcurrencyToken()`: Marca una propiedad como token de concurrencia.

---

3. Relaciones (Navegaciones)

- `HasOne()`: Configura una relación uno-a-uno o uno-a-muchos desde el dependiente.
- `WithOne()`: Configura una relación uno-a-uno desde el principal.
- `WithMany()`: Configura una relación uno-a-muchos desde el principal.
- `HasForeignKey()`: Configura una clave externa.
- `HasPrincipalKey()`: Configura una clave principal en una relación.

- `OnDelete()`: Configura el comportamiento de eliminación.

---

4. Índices y Claves Alternativas

- `HasIndex()`: Crea un índice.
- `IsUnique()`: Define que un índice sea único.

---

5. Configuración de Tipos de Entidades

- `OwnsOne()`: Configura una entidad propia (tipo embebido).
- `HasQueryFilter()`: Define un filtro global para consultas.

---

6. Configuración de Esquemas

- `ToSchema()`: Especifica el esquema de la tabla.

---

7. Configuración de Modelos Avanzados

- `UsePropertyAccessMode()`: Controla cómo EF accede a las propiedades.
- `HasChangeTrackingStrategy()`: Configura la estrategia de seguimiento de cambios.

---

8. Configuración de Tipos de Datos

- `HasConversion()`: Convierte una propiedad a un tipo específico.

Relaciones con Fluent API

Fluent API - Configuración de Relaciones

Relaciones Fluent API

Introducción

- Permite definir relaciones entre entidades cuando las convenciones de EF Core no son suficientes.
- Principales métodos:
    - **`HasRequired`**: Relación obligatoria.
    - **`HasOptional`**: Relación opcional.
    - **`HasMany`**: Relación uno-a-muchos o muchos-a-muchos.
    - **`WithRequired`**, **`WithOptional`**, **`WithMany`**: Configuran navegaciones inversas.
    - **`HasForeignKey`**: Configura claves externas.
    - **`Map`**: Personaliza tablas de combinación en relaciones muchos-a-muchos.

Configuración de Relaciones

1. **Relación obligatorio-opcional (uno a cero o uno):**
    - Una entidad tiene una referencia opcional a otra.

Ejemplo:

```
modelBuilder.Entity<OfficeAssignment>()
    .HasRequired(t => t.Instructor)
    .WithOptional(t => t.OfficeAssignment);
```

    -
2. **Relación obligatoria en ambos extremos (uno a uno):**
    - Ambas entidades son necesarias en la relación.

Ejemplo:

```
modelBuilder.Entity<Instructor>()
    .HasRequired(t => t.OfficeAssignment)
    .WithRequiredPrincipal(t => t.Instructor);
```

3. **Relación varios a varios:**
    - Convenciones crean automáticamente una tabla de combinación.

Ejemplo básico:

```
modelBuilder.Entity<Course>()
    .HasMany(t => t.Instructors)
    .WithMany(t => t.Courses);
```

Personalizando la tabla de combinación:

```
modelBuilder.Entity<Course>()
    .HasMany(t => t.Instructors)
    .WithMany(t => t.Courses)
    .Map(m =>
    {
        m.ToTable("CourseInstructor");
        m.MapLeftKey("CourseID");
        m.MapRightKey("InstructorID");
    });
```

4. **Relación con navegación unidireccional:**
    - Solo se define navegación en un extremo.

Ejemplo:

```
modelBuilder.Entity<Instructor>()
    .HasRequired(t => t.OfficeAssignment)
    .WithRequiredPrincipal();
```

5. **Habilitar o deshabilitar eliminación en cascada:**
    - Configura comportamiento de eliminación en cascada.

Ejemplo para deshabilitarla:

```
modelBuilder.Entity<Course>()
    .HasRequired(t => t.Department)
    .WithMany(t => t.Courses)
    .HasForeignKey(d => d.DepartmentID)
    .WillCascadeOnDelete(false);
```

    -
6. **Clave externa compuesta:**
    - Define claves externas compuestas.

Ejemplo:

```
modelBuilder.Entity<Department>()
    .HasKey(d => new { d.DepartmentID, d.Name });

modelBuilder.Entity<Course>()
    .HasRequired(c => c.Department)
    .WithMany(d => d.Courses)
    .HasForeignKey(d => new { d.DepartmentID, d.DepartmentName });
```

    -
7. **Renombrar claves externas no definidas en el modelo:**

Ejemplo:

```
modelBuilder.Entity<Course>()
    .HasRequired(c => c.Department)
    .WithMany(t => t.Courses)
    .Map(m => m.MapKey("ChangedDepartmentID"));
```

    -
8. **Clave externa que no sigue las convenciones:**

Si el nombre de la clave no es convencional:

```
modelBuilder.Entity<Course>()
    .HasRequired(c => c.Department)
    .WithMany(d => d.Courses)
    .HasForeignKey(c => c.SomeDepartmentID);
```

Relaciones uno a varios

Otra forma de ver las relaciones
Uno a Varios

Relación de uno a varios obligatoria

1. **Definición:**
    - Cada entidad dependiente (e.g., Post) debe estar asociada a una entidad principal (e.g., Blog) porque su clave externa no admite valores NULL.
2. **Componentes clave:**
    - **Entidad principal (e.g., Blog):** Tiene una clave principal (Blog.Id) y opcionalmente una colección que navega a los dependientes (Blog.Posts).
    - **Entidad dependiente (e.g., Post):** Tiene una clave externa obligatoria (Post.BlogId) y opcionalmente una referencia al principal (Post.Blog).

Ejemplo de modelos:

```
public class Blog
{
  public int Id { get; set; }
  public ICollection<Post> Posts { get; } = new List<Post>();
}

public class Post
{
  public int Id { get; set; }
  public int BlogId { get; set; }
  public Blog Blog { get; set; } = null!;
}
```

Configuración explícita:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
  modelBuilder.Entity<Blog>()
```

```
        .HasMany(e => e.Posts)
        .WithOne(e => e.Blog)
        .HasForeignKey(e => e.BlogId)
        .IsRequired();
}
```

Conceptos clave:

- **Relación bidireccional:** Incluye navegaciones en ambos lados (Blog.Posts y Post.Blog).
- **Detección por convención:** EF Core configura automáticamente relaciones si las propiedades siguen las convenciones.
- **Configuración explícita:** Solo es necesaria cuando las convenciones no aplican.
- **Referencias que admiten NULL:** Si la clave externa admite NULL, la navegación también debe admitirlo.

Relación opcional:

- Similar a la obligatoria, pero la clave externa y la referencia en el dependiente **admiten valores NULL**, permitiendo que un dependiente no esté asociado a un principal.

Ejemplo de configuración:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
  modelBuilder.Entity<Blog>()
    .HasMany(e => e.Posts)
    .WithOne(e => e.Blog)
    .HasForeignKey(e => e.BlogId)
    .IsRequired(false);
}
```

Ventaja: Flexibilidad para modelar escenarios donde una relación no siempre es necesaria

Relación uno a uno

[Relación uno a uno]

Relación de uno a uno obligatoria

1. **Definición:**
    - El dependiente (e.g., BlogHeader) tiene una clave externa que **no acepta valores NULL**, asegurando que siempre esté relacionado con un principal (e.g., Blog).
    - El principal puede existir sin dependiente.

    Ejemplo:

```
public class Blog
{
  public int Id { get; set; }
  public BlogHeader? Header { get; set; }
}

public class BlogHeader
{
  public int Id { get; set; }
  public int BlogId { get; set; }
  public Blog Blog { get; set; } = null!;
}
```

    Configuración explícita (si no se detecta por convención):

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
  modelBuilder.Entity<Blog>()
    .HasOne(e => e.Header)
    .WithOne(e => e.Blog)
    .HasForeignKey<BlogHeader>(e => e.BlogId)
    .IsRequired();
}
```

Relación de uno a uno opcional

Definición:

- La clave externa en el dependiente **acepta valores NULL**, permitiendo que no siempre esté relacionado con un principal.

Ejemplo:

```
public class Blog
{
  public int Id { get; set; }
  public BlogHeader? Header { get; set; }
}

public class BlogHeader
{
  public int Id { get; set; }
  public int? BlogId { get; set; }
  public Blog? Blog { get; set; }
}
```

Configuración explícita:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
  modelBuilder.Entity<Blog>()
    .HasOne(e => e.Header)
    .WithOne(e => e.Blog)
    .HasForeignKey<BlogHeader>(e => e.BlogId)
    .IsRequired(false);
}
```

Conceptos Clave:

- **Principal y dependiente:**
  - El dependiente contiene la clave externa.

    - El principal puede existir sin dependiente.
- **Convención vs. configuración explícita:**
    - EF Core detecta relaciones automáticamente si las propiedades y navegaciones cumplen con las convenciones.
    - La configuración explícita es necesaria cuando las convenciones no aplican.
- **Relación bidireccional:**
    - Incluye navegaciones en ambos lados (Blog.Header y BlogHeader.Blog).
- **Tipos de referencia que aceptan NULL:**
    - La navegación y clave externa deben admitir o no valores NULL de forma consistente.

Relación Varios a varios

Varios a varios

1. Relación de varios a varios simplificada (sin tabla de combinación explícita)

Úsala cuando:

- No necesitas propiedades adicionales en la tabla de combinación.
- Buscas un código más limpio y fácil de mantener.

Ejemplo:

```
public class Post
{
  public int Id { get; set; }
  public List<Tag> Tags { get; } = new();
}

public class Tag
{
  public int Id { get; set; }
  public List<Post> Posts { get; } = new();
}
```

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
   modelBuilder.Entity<Post>()
      .HasMany(p => p.Tags)
      .WithMany(t => t.Posts);
}
```

Ventajas:

- Simplicidad en el modelo y código.
- EF Core gestiona automáticamente la tabla de combinación.

2. Relación de varios a varios con tabla de combinación explícita

Úsala cuando:

- Necesitas propiedades adicionales en la tabla de combinación (como una fecha de creación, estado, etc.).
- Requieres un control más detallado sobre las relaciones.

Ejemplo:

```
public class Post
{
   public int Id { get; set; }
   public List<PostTag> PostTags { get; } = new();
}

public class Tag
{
   public int Id { get; set; }
   public List<PostTag> PostTags { get; } = new();
```

```
}

public class PostTag
{
    public int PostId { get; set; }
    public int TagId { get; set; }
    public DateTime CreatedOn { get; set; }
    public Post Post { get; set; } = null!;
    public Tag Tag { get; set; } = null!;
}
```

Context.cs

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<PostTag>()
        .HasKey(pt => new { pt.PostId, pt.TagId });

    modelBuilder.Entity<PostTag>()
        .Property(pt => pt.CreatedOn)
        .HasDefaultValueSql("CURRENT_TIMESTAMP");

    modelBuilder.Entity<Post>()
        .HasMany(p => p.PostTags)
        .WithOne(pt => pt.Post)
        .HasForeignKey(pt => pt.PostId);

    modelBuilder.Entity<Tag>()
        .HasMany(t => t.PostTags)
        .WithOne(pt => pt.Tag)
        .HasForeignKey(pt => pt.TagId);
}
```

Ventajas:

Permite agregar información adicional a las relaciones.
Mayor flexibilidad y control sobre la estructura y comportamiento.