PREGRADO

UNIDAD 3 | WEB SERVICES

SAVING DATA, TRANSACTIONS & INHERITANCE

SI730 | Aplicaciones Web

Al finalizar la semana, el estudiante implementa componentes para capas de persistencia de una aplicación de lado servidor, con características innovadoras, bajo una arquitectura orientada a servicios y aplicando los principios RESTful utilizando el lenguaje C# y Microsoft .NET Framework.

AGENDA

INTRO

SAVING DATA

TRANSACTIONS

INHERITANCE

Entity Framework Core DcContext

DbContext es parte integral de Entity Framework.

Una instancia de DbContext representa una sesión con la base de datos. Se puede utilizar para queries o grabar instancias de entities en la base de datos.

DbContext es una combinación de Unit Of Work y Repository Design Patterns.

Entity Framework Core DcContext

DbContext en Entity Framework Core permite realizar tareas como:

Database Connection Management.

Database Querying.

Saving data to the Database.

Change Tracking Configuration.

Caching.

Transaction Management.

AGENDA

INTRO

SAVING DATA

TRANSACTIONS

INHERITANCE

Saving Data

Entity Framework Core proporciona diferentes maneras de adicionar, actualizar o eliminar data en la base de datos.

Los entities que contienen datos en sus propiedades escalares serán insertados, actualizados o eliminados en base a su EntityState.

Cuando se invoca al método DbContext.SaveChanges(), realiza la operación correspondiente según el EntityState (Added, Modified, Deleted).

Insert Data

Puede utilizarse

DbSet.Add o DbContext.Add

```
using (var context = new SchoolContext())
{
    var std = new Student()
    {
        FirstName = "John",
        LastName = "Doe"
    };
    context.Students.Add(std);


    // or
    // context.Add<Student>(std);


    context.SaveChanges();
}
```

Updating Data

Cuando se edita los datos de un entity, Entity Framework Core marca su EntityState como Modified. Cuando se llama a SaveChanges() ello produce un Update.

```
using (var context = new SchoolContext())
{
    var std = context.Students.First<Student>();
    std.FirstName = "Jason";
    context.SaveChanges();
}
```

Deleting Data

Puede utilizarse DbSet.Remove() o DbContext.Remove para eliminar un registro en la tabla de base de datos.

```
using (var context = new SchoolContext())
{
    var std = context.Students.First<Student>();
    context.Students.Remove(std);


    // or
    // context.Remove<Student>(std);


    context.SaveChanges();
}
```

Saving Related Data

Además de guardar entidades aisladas, se puede hacer uso de las relaciones definidas en el modelo.

Adding Graph of New Entities

Si se creó varias entities nuevas relacionadas, agregar una de ellas al contexto provocará que las otras se agreguen también.

```
using (var context = new BloggingContext())
{
    var blog = new Blog
    {
        Url = "http://blogs.msdn.com/dotnet",
        Posts = new List<Post>
        {
            new Post { Title = "Intro to C#" },
            new Post { Title = "Intro to VB.NET" },
            new Post { Title = "Intro to F#" }
        }
    };


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

Adding a related Entity

De referenciarse una nueva entity desde la propiedad de navegación de de un entity que ya está siendo controlado por context, el entity será descubierto e insertado en la base de datos.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Include(b => b.Posts).First();
    var post = new Post { Title = "Intro to EF Core" };


    blog.Posts.Add(post);
    context.SaveChanges();
}
```

Changing relationships

De cambiarse la navigation property de un entity, los cambios correspondientes se harán sobre la columna de foreign key en la base de datos.

```
using (var context = new BloggingContext())
{
    var blog = new Blog { Url = "http://blogs.msdn.com/visualstudio" };
    var post = context.Posts.First();


    post.Blog = blog;
    context.SaveChanges();
}
```

Removing relationships

Puede eliminarse una relación estableciendo el reference navigation a null, o eliminando el entity relacionado de un navigation collection.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Include(b => b.Posts).First();
    var post = blog.Posts.First();


    blog.Posts.Remove(post);
    context.SaveChanges();
}
```

AGENDA

INTRO

SAVING DATA

TRANSACTIONS

INHERITANCE

Default transaction behavior

Por defecto, si el database provider soporta transacciones, todos los cambios en una llamada a **SaveChanges**() se aplican como una transacción.

En caso falle alguno de los cambios, entonces la transacción es rolled back y ninguno de los cambios se aplica en la base de datos.

Transaction control

Puede utilizarse el DbContext.Database API para iniciar, hacer commit o rollback a transacciones.

Si el database provider no soporta transacciones, podría provocar una excepción o no realizar nada.

DbContext.Database.BeginTransaction(). Crea una nueva transacción para la base de datos y permite realizar commit o rollback usando múltiples llamadas a SaveChanges().

DbContext.Database.UseTransaction(). Permite pasar un objeto transaction creado fuera del scope de un context object.

Transaction control

Usando DbContext.Database.BeginTransaction()

```
using (var context = new BloggingContext())
{
    using (var transaction = context.Database.BeginTransaction())
    {
        try
        {
            context.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/dotnet" });
            context.SaveChanges();

            context.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/visualstudio" });
            context.SaveChanges();

            var blogs = context.Blogs
                .OrderBy(b => b.Url)
                .ToList();

            // Commit transaction if all commands succeed, transaction will auto-rollback
            // when disposed if either commands fails
            transaction.Commit();
        }
        catch (Exception)
        {
            transaction.Rollback();
          Console.WriteLine("Error ocurred");
        }
    }
}
```

Transaction control

Usando DbContext.Database.BeginTransaction()

```
using (var context = new SchoolContext())
{
    context.Database.Log = Console.Write;

    using (DbContextTransaction transaction = context.Database.BeginTransaction())
    {
        try
        {
            var standard = context.Standards.Add(new Standard() { StandardName = "5th Semester" });

            context.Students.Add(new Student()
            {
                FirstName = "William",
                StandardId = standard.StandardId
            });
            context.SaveChanges();

            context.Courses.Add(new Course() { CourseName = "Web Applications" });
            context.SaveChanges();

            transaction.Commit();
        }
        catch (Exception ex)
        {
            transaction.Rollback();
            Console.WriteLine("Error occurred.");
        }
    }
}
```

IDbContextTransaction Interface

Representa a una transacción.

Las instancias se obtienen de BeginTransaction().

Properties

TransactionId

Methods

Commit()

CommitAsync()

Rollback()

RollbackAsync(CancellationToken)

Custom DbContext & Transactions

```
public class LasteDbContext : DbContext
{
    public LasteDbContext(DbContextOptions<LasteDbContext> options) : base(options)
    {
    }

    public DbSet<Comment> Comments { get; set; }
    public DbSet<Invoice> Invoices { get; set; }
    public DbSet<InvoiceLine> InvoiceLines { get; set; }

    private IDbContextTransaction _transaction;

    public void BeginTransaction()
    {
        _transaction = Database.BeginTransaction();
    }

    public void Commit()
    {
        try
        {
            SaveChanges();
            _transaction.Commit();
        }
        finally
        {
            _transaction.Dispose();
        }
    }

    public void Rollback()
    {
        _transaction.Rollback();
        _transaction.Dispose();
    }
}
```

AGENDA

INTRO

SAVING DATA

TRANSACTIONS

INHERITANCE

Inheritance

Es posible hacer mapping de una jerarquía de clases de .NET en una base de datos.

Entity Framework Core actualmente soporta los patrones table-per-hierarchy (TPH) y table-per-type (TPT)

Entity type hierarchy mapping

Toda clase que se desea participe en la jerarquía debe estar especificada en el modelo.

Se incluye por defecto una columna *Discriminator*.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<RssBlog> RssBlogs { get; set; }
}


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


public class RssBlog : Blog
{
    public string RssUrl { get; set; }
}
```

Discriminator

Puede configurar el nombre y valores de Discriminator.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .HasDiscriminator<string>("blog_type")
        .HasValue<Blog>("blog_base")
        .HasValue<RssBlog>("blog_rss");

    modelBuilder.Entity<Blog>()
            .Property(e => e.BlogType)
            .HasMaxLength(200)
            .HasColumnName("blog_type");

}
```

Shared columns

Properties idénticos se pueden mapear a la misma columna.

```
public class MyContext : DbContext
{
    public DbSet<BlogBase> Blogs { get; set; }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.Url)
            .HasColumnName("Url");

        modelBuilder.Entity<RssBlog>()
            .Property(b => b.Url)
            .HasColumnName("Url");
    }
}
```

```
public abstract class BlogBase
{
    public int BlogId { get; set; }
}

public class Blog : BlogBase
{
    public string Url { get; set; }
}

public class RssBlog : BlogBase
{
    public string Url { get; set; }
}
```

Table-per-type configuration

Mapeo a tablas individuales por cada tipo.

```
modelBuilder.Entity<Blog>().ToTable("Blogs");
modelBuilder.Entity<RssBlog>().ToTable("RssBlogs");
```

```
CREATE TABLE [Blogs] (
    [BlogId] int NOT NULL IDENTITY,
    [Url] nvarchar(max) NULL,
    CONSTRAINT [PK_Blogs] PRIMARY KEY ([BlogId])
);


CREATE TABLE [RssBlogs] (
    [BlogId] int NOT NULL,
    [RssUrl] nvarchar(max) NULL,
    CONSTRAINT [PK_RssBlogs] PRIMARY KEY ([BlogId]),
    CONSTRAINT [FK_RssBlogs_Blogs_BlogId] FOREIGN KEY ([BlogId]) REFERENCES [Blogs]
([BlogId]) ON DELETE NO ACTION
);
```

Table-per-type configuration

Recuperando column name utilizando

`GetColumnName(IProperty, StoreObjectIdentifier)`.

```
foreach (var entityType in modelBuilder.Model.GetEntityTypes())
{
    var tableIdentifier = StoreObjectIdentifier.Create(entityType, StoreObjectType.Table);

    Console.WriteLine($"{entityType.DisplayName()}\t\t{tableIdentifier}");
    Console.WriteLine(" Property\tColumn");


    foreach (var property in entityType.GetProperties())
    {
        var columnName = property.GetColumnName(tableIdentifier.Value);
        Console.WriteLine($" {property.Name,-10}\t{columnName}");
    }


    Console.WriteLine();
}
```

RESUMEN

Recordemos

DbContext

Saving Data

Transactions

Inheritance

REFERENCIAS

Para profundizar

https://docs.microsoft.com/en-us/ef/core/saving/transactions

https://docs.microsoft.com/en-us/ef/core/querying/

https://docs.microsoft.com/en-us/ef/core/modeling/inheritance

PREGRADO

Ingeniería de Software

Escuela de Ingeniería de Sistemas y Computación | Facultad de Ingeniería

UPC
Universidad Peruana
de Ciencias Aplicadas

Prolongación Primavera 2390,
Monterrico, Santiago de Surco
Lima 33 - Perú
T 511 313 3333
https://www.upc.edu.pe

exígete, innova