Escuelita Crombie 3er Edición

DapperContext

```
public class DapperContext
{
    private readonly IConfiguration _configuration;
    private readonly string _connectionString;

    0 referencias
    public DapperContext(IConfiguration configuration)
    {
        _configuration = configuration;
        //Obtener de appsettings.json
        _connectionString = _configuration.GetConnectionString("DefaultConnection");
    }

    5 referencias
    public IDbConnection CreateConnection() =>
        new SqlConnection(_connectionString);
}
```

IDbConnection

Define el comportamiento principal de las conexiones de base de datos y proporciona una clase base para conexiones específicas de la base de datos.

Program

```
builder.Services.AddSingleton<DapperContext>();
```

Capa Repository

```
public class ProductoRepository : IProductoRepository
{
    private readonly DapperContext _context;

    0 referencias
    public ProductoRepository(DapperContext context)
    {
        _context = context;
    }
```

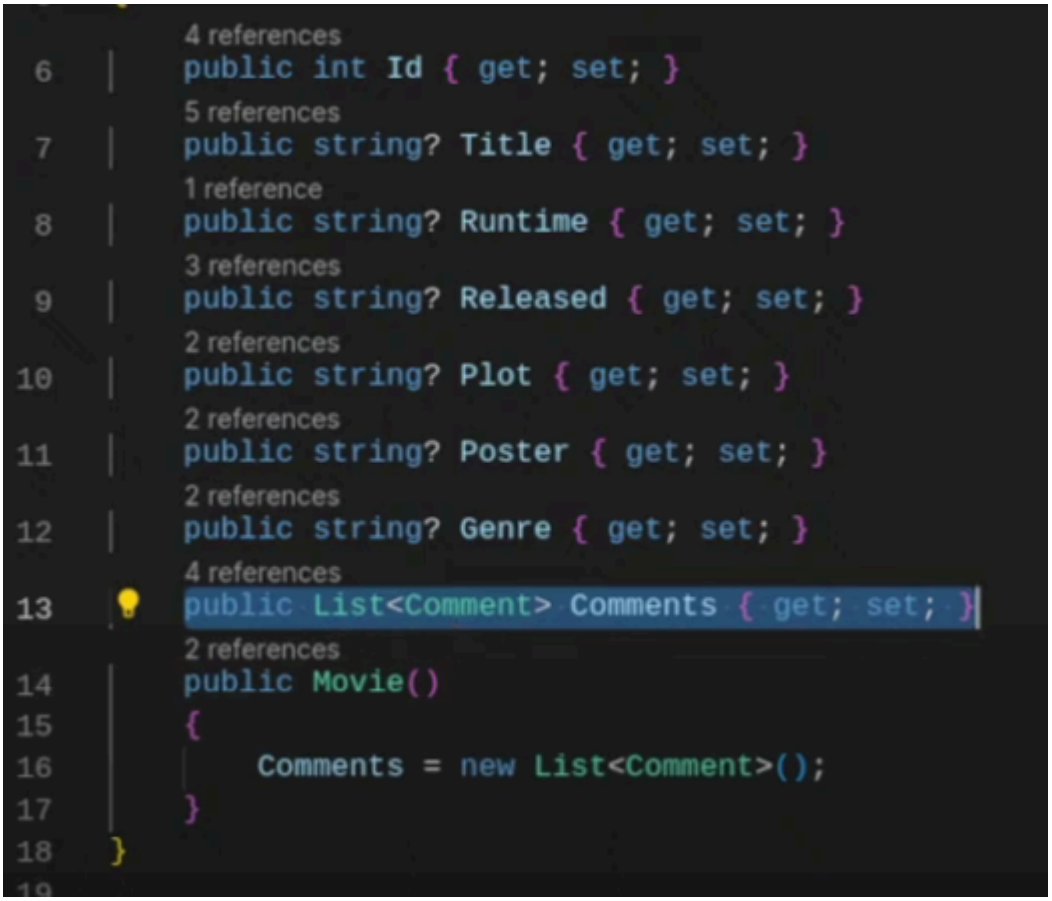
Propiedades de navegación:

Las **propiedades de navegación** en .NET son elementos de un modelo de datos que definen relaciones entre entidades en un contexto de Entity Framework. Permiten navegar desde una entidad hacia otras relacionadas, simplificando consultas y manipulaciones de datos.

Tipos de propiedades de navegación:

De colección:

- Representan relaciones de uno a muchos o muchos a muchos.



De referencia:

- Representan relaciones de uno a uno o muchos a uno.

```
public class Comment
{
    0 references
    public int Id { get; set; }
    2 references
    public string? Date { get; set; }
    1 reference
    public string? Text { get; set; }
    1 reference
    public int MovieId { get; set; }
    0 references
    public Movie? Movie { get; set; }
}
```

Relación muchos a muchos

```
public class Student
{
    0 referencias
    public int Id { get; set; }
    0 referencias
    public string Name { get; set; }

    // Propiedad de navegación para la relación muchos a muchos
    0 referencias
    public ICollection<StudentCourse> StudentCourses { get; set; }
}
```

```
public class Course
{
    0 referencias
    public int Id { get; set; }
    0 referencias
    public string Title { get; set; }

    // Propiedad de navegación para la relación muchos a muchos
    0 referencias
    public ICollection<StudentCourse> StudentCourses { get; set; }
}
```

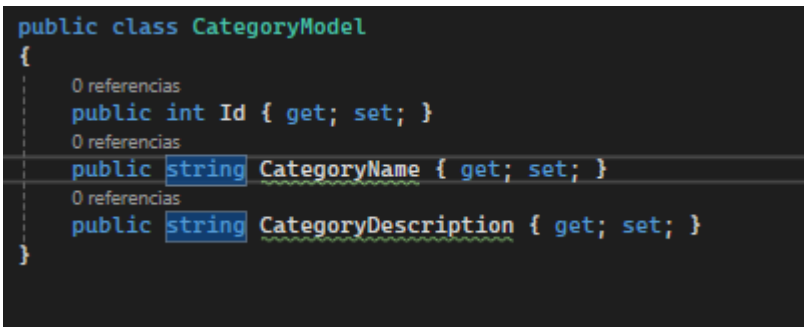
- Tabla intermedia

```
2 referencias
public class StudentCourse
{
    0 referencias
    public int StudentId { get; set; }
    0 referencias
    public Student Student { get; set; }

    0 referencias
    public int CourseId { get; set; }
    0 referencias
    public Course Course { get; set; }

    0 referencias
    public DateTime EnrollmentDate { get; set; } = DateTime.Now; // Propiedad adicional opcional
}
```

C.R.U.D. SQL Dapper

Clases

```
public class ProductoModel
{
    3 referencias
    public int Id { get; set; }
    0 referencias
    public required string Name { get; set; }
    0 referencias
    public string Description { get; set; }
    0 referencias
    public int Stock { get; set; }
    0 referencias
    public DateTime CreateDate { get; set; } = DateTime.Now;
    0 referencias
    public int CategoryId { get; set; }
    1 referencia
    public CategoryModel? Category { get; set; }
}
```


public class CategoryModel
{
0 referencias
public int Id { get; set; }
0 referencias
public string CategoryName { get; set; }
0 referencias
public string CategoryDescription { get; set; }
}

Get con Join

```
public IEnumerable<ProductoModel> GetAllProducto()
{
    using (var connection = _context.CreateConnection())
    {
        try
        {
            var sql = @"
        SELECT
            p.Id,
            p.Name,
            p.Description,
            p.Stock,
            p.CreateDate,
            p.CategoryId,
            c.Id AS CategoryId,
            c.CategoryName,
            c.CategoryDescription
        FROM
            ProductEntity p
        INNER JOIN
            CategoryEntity c
        ON
            p.CategoryId = c.Id";

            // Mapeo de ProductoModel y CategoryModel
            var productos = connection.Query<ProductoModel, CategoryModel, ProductoModel>(
                sql,
                (producto, categoria) =>
                {
                    producto.Category = categoria; // Asignamos la categoría al producto
                    return producto;
                },
                splitOn: "CategoryId" // El mapeo comienza en esta columna
            );

            return productos;
        }
        catch (SqlException sqlEx)
        {
            _logger.LogError(sqlEx, "Error ejecutando consulta SQL en GetAllProducto");
            return Enumerable.Empty<ProductoModel>();
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error inesperado en GetAllProducto");
            return Enumerable.Empty<ProductoModel>();
        }
    }
}
```

POST

```
2 referencias
public void InsertProducto(ProductoModel producto)
{
    using (var connection = _context.CreateConnection())
    {
        try
        {
            var sql = @"
                    INSERT INTO ProductEntity (Name, Description, Stock, CreateDate, CategoryId)
                    VALUES (@Name, @Description, @Stock, @CreateDate, @CategoryId)";

            var rowsAffected = connection.Execute(sql, producto);

            if (rowsAffected == 0)
            {
                _logger.LogWarning("No se insertaron registros para el producto: {@Producto}", producto);
            }
        }
        catch (SqlException sqlEx)
        {
            _logger.LogError(sqlEx, "Error ejecutando consulta SQL en InsertProducto para el producto: {@Producto}", producto);
            throw new Exception("Ocurrió un error al insertar el producto en la base de datos. Por favor, intente nuevamente.", sqlEx);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error inesperado en InsertProducto para el producto: {@Producto}", producto);
            throw new Exception("Ocurrió un error inesperado al insertar el producto. Por favor, intente nuevamente.", ex);
        }
    }
}
```

UPDATE

```
public string UpdateProduct(ProductoModel producto)
{
    using (var connection = _context.CreateConnection())
    {
        connection.Open();
        using (var transaction = connection.BeginTransaction())
        {
            try
            {
                var sql = @"
                        UPDATE ProductEntity
                        SET Name = @Name, Description = @Description, Stock = @Stock, CreateDate = @CreateDate
                        WHERE Id = @Id";

                var rowsAffected = connection.Execute(sql, producto, transaction);

                if (rowsAffected == 0)
                {
                    _logger.LogWarning("No se encontró un producto con el ID {Id} para actualizar.", producto.Id);
                    return $"No se encontró un producto con el ID {producto.Id} para actualizar.";
                }

                transaction.Commit();
                _logger.LogInformation("Producto con ID {Id} actualizado correctamente. {RowsAffected} registros afectados.", producto.Id, rowsAffected);
                return $"Se afectaron {rowsAffected} registros.";
            }
            catch (SqlException sqlEx)
            {
                transaction.Rollback();
                _logger.LogError(sqlEx, "Error ejecutando consulta SQL en UpdateProduct para el producto: {@Producto}", producto);
                throw new Exception("Ocurrió un error al actualizar el producto. Por favor, intente nuevamente.", sqlEx);
            }
            catch (Exception ex)
            {
                transaction.Rollback();
                _logger.LogError(ex, "Error inesperado en UpdateProduct para el producto: {@Producto}", producto);
                throw new Exception("Ocurrió un error inesperado al actualizar el producto. Por favor, intente nuevamente.", ex);
            }
        }
    }
}
```

Delete

```
public string DeleteProducto(int id)
{
    using (var connection = _context.CreateConnection())
    {
        try
        {
            var sql = "DELETE FROM ProductEntity WHERE Id = @Id";

            var rowsAffected = connection.Execute(sql, new { Id = id });

            if (rowsAffected == 0)
            {
                _logger.LogWarning("No se encontró un producto con el ID {Id} para eliminar.", id);
                return $"No se encontró un producto con el ID {id} para eliminar.";
            }

            _logger.LogInformation("Producto con ID {Id} eliminado correctamente.", id);
            return "Eliminado correctamente.";
        }
        catch (SqlException sqlEx)
        {
            _logger.LogError(sqlEx, "Error ejecutando consulta SQL en DeleteProducto para el ID {Id}.", id);
            throw new Exception("Ocurrió un error al eliminar el producto. Por favor, intente nuevamente.", sqlEx);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error inesperado en DeleteProducto para el ID {Id}.", id);
            throw new Exception("Ocurrió un error inesperado al eliminar el producto. Por favor, intente nuevamente.", ex);
        }
    }
}
```