Data Access Layer dengan Dapper

Panduan Lengkap Implementasi Dapper di .NET

Daftar Isi

Apa itu Data Access Layer?

Pengenalan

Data Access Layer (DAL) adalah lapisan dalam arsitektur aplikasi yang bertanggung jawab untuk mengelola komunikasi antara aplikasi dan database. DAL mengabstraksi logika akses data dari business logic, sehingga kode lebih terorganisir dan mudah di-maintain.

Apa itu Dapper?

Pengenalan

Dapper adalah micro-ORM (Object-Relational Mapper) yang dikembangkan oleh Stack Overflow. Dapper memberikan performa tinggi mendekati ADO.NET murni, namun dengan kemudahan mapping object yang lebih baik.

Keunggulan Dapper:

• ■ Performa tinggi (hampir sama dengan ADO.NET)

• ■ Simple dan mudah dipelajari

• ■ Fleksibel dengan RAW SQL

• ■ Lightweight (hanya satu file)

• ■ Mapping otomatis dari query result ke object

Instalasi

Pengenalan

```
dotnet add package Dapper
dotnet add package Microsoft.Data.SqlClient
# atau untuk PostgreSQL
dotnet add package Npgsql
# atau untuk MySQL
dotnet add package MySql.Data
```

1. Result Mapping

Result mapping adalah proses memetakan hasil query database ke object C#. Dapper melakukan mapping secara otomatis berdasarkan nama kolom dan property.

1.1 Basic Mapping

1. Result Mapping

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
    public int Stock { get; set; }
    public DateTime CreatedAt { get; set; }
}
// Query dengan mapping otomatis
public async Task<Product> GetProductById(int id)
{
    using var connection = new SqlConnection(_connectionString);
    var sql = "SELECT Id, Name, Price, Stock, CreatedAt FROM Products WHERE Id = @Id";
```

1.2 Custom Column Mapping

1. Result Mapping

Ketika nama kolom database berbeda dengan property C#:

```
public class Customer
{
    public int CustomerId { get; set; }
    public string FullName { get; set; }
    public string EmailAddress { get; set; }
}
// SQL dengan alias untuk mapping
var sql = @"
    SELECT
        customer_id as CustomerId,
        full_name as FullName,
        email as EmailAddress
    FROM Customers
    WHERE customer_id = @Id";
```

1.3 Multi-Mapping (One-to-One)

1. Result Mapping

Mapping untuk relasi antar tabel:

```
public class Order
{
    public int OrderId { get; set; }
    public DateTime OrderDate { get; set; }
    public decimal TotalAmount { get; set; }
    public Customer Customer { get; set; }
}
public async Task<Order> GetOrderWithCustomer(int orderId)
{
    using var connection = new SqlConnection(_connectionString);
    var sql = @"
        SELECT
            o.OrderId, o.OrderDate, o.TotalAmount,
```

1.4 Multi-Mapping (One-to-Many)

1. Result Mapping

Mapping untuk relasi one-to-many:

```
public class Order
{
```

```
    public int OrderId { get; set; }
    public DateTime OrderDate { get; set; }
    public List<OrderItem> Items { get; set; } = new();
}
public class OrderItem
{
    public int OrderItemId { get; set; }
    public int ProductId { get; set; }
    public string ProductName { get; set; }
    public int Quantity { get; set; }
    public decimal Price { get; set; }
}
```

1.5 Dynamic Mapping

1. Result Mapping

Untuk kasus dimana struktur result tidak fixed:

```
public async Task<IEnumerable<dynamic>> GetDynamicData(string tableName)
{
    using var connection = new SqlConnection(_connectionString);
    // Hati-hati: validate tableName untuk menghindari SQL injection
    var sql = $"SELECT * FROM {tableName}";
    return await connection.QueryAsync(sql);
}
```

2.1 Clean Architecture dengan Dapper

2. Integrasi Arsitektur

Struktur Folder:

```
MyProject/
■■■ MyProject.Domain/          # Entities, Interfaces
■■■ MyProject.Application/     # Business Logic, DTOs
■■■ MyProject.Infrastructure/  # Data Access Implementation
■   ■■■ DataAccess/
■       ■■■ Repositories/
■       ■■■ DbContext/
■■■ MyProject.API/            # Web API Controllers
```

2.2 Repository Pattern

2. Integrasi Arsitektur

Interface (Domain Layer):

```
namespace MyProject.Domain.Repositories
{
    public interface IProductRepository
    {
        Task<Product> GetByIdAsync(int id);
        Task<IEnumerable<Product>> GetAllAsync();
        Task<Product> CreateAsync(Product product);
        Task<bool> UpdateAsync(Product product);
        Task<bool> DeleteAsync(int id);
    }
}
```

Implementation (Infrastructure Layer):

```
using Dapper;
using Microsoft.Data.SqlClient;
using MyProject.Domain.Entities;
using MyProject.Domain.Repositories;
namespace MyProject.Infrastructure.DataAccess.Repositories
{
    public class ProductRepository : IProductRepository
    {
        private readonly string _connectionString;
        public ProductRepository(string connectionString)
        {
            _connectionString = connectionString;
        }
```

2.3 Unit of Work Pattern

2. Integrasi Arsitektur

Interface:

```
namespace MyProject.Domain.UnitOfWork
{
    public interface IUnitOfWork : IDisposable
    {
        IProductRepository Products { get; }
        ICustomerRepository Customers { get; }
        IOrderRepository Orders { get; }
        Task<int> SaveChangesAsync();
        Task BeginTransactionAsync();
        Task CommitAsync();
        Task RollbackAsync();
    }
}
```

Implementation:

```
using System.Data;
using Microsoft.Data.SqlClient;
```

```
using MyProject.Domain.Repositories;
using MyProject.Domain.UnitOfWork;
namespace MyProject.Infrastructure.DataAccess.UnitOfWork
{
    public class UnitOfWork : IUnitOfWork
    {
        private readonly SqlConnection _connection;
        private SqlTransaction _transaction;
        private bool _disposed;
        public IProductRepository Products { get; }
        public ICustomerRepository Customers { get; }
```

2.4 Dependency Injection

2. Integrasi Arsitektur

Program.cs (.NET 6+):

```
using MyProject.Domain.Repositories;
using MyProject.Infrastructure.DataAccess.Repositories;
var builder = WebApplication.CreateBuilder(args);
// Add services to the container
builder.Services.AddControllers();
// Connection String
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
// Register Repositories
builder.Services.AddScoped<IProductRepository>(sp =>
    new ProductRepository(connectionString));
builder.Services.AddScoped<ICustomerRepository>(sp =>
```

3.1 Query Methods

3. Query Execution

Dapper menyediakan berbagai method untuk eksekusi query:

Query<T>

Mengembalikan IEnumerable<T> untuk multiple rows:

```
public async Task<IEnumerable<Product>> GetProducts()
{
    using var connection = new SqlConnection(_connectionString);
    var sql = "SELECT * FROM Products";
    return await connection.QueryAsync<Product>(sql);
}
```

QuerySingle<T> & QuerySingleOrDefault<T>

Untuk single row (error jika > 1 row):

```
public async Task<Product> GetProductById(int id)
{
    using var connection = new SqlConnection(_connectionString);
    var sql = "SELECT * FROM Products WHERE Id = @Id";
    // QuerySingle: error jika tidak ada atau > 1 row
    return await connection.QuerySingleAsync<Product>(sql, new { Id = id });
    // QuerySingleOrDefault: return null jika tidak ada
    // return await connection.QuerySingleOrDefaultAsync<Product>(sql, new { Id = id });
}
```

QueryFirst<T> & QueryFirstOrDefault<T>

Mengambil row pertama:

```
public async Task<Product> GetFirstProduct()
{
    using var connection = new SqlConnection(_connectionString);
    var sql = "SELECT TOP 1 * FROM Products ORDER BY CreatedAt DESC";
    return await connection.QueryFirstOrDefaultAsync<Product>(sql);
}
```

Execute

Untuk INSERT, UPDATE, DELETE:

```
public async Task<int> UpdateProductPrice(int id, decimal newPrice)
{
    using var connection = new SqlConnection(_connectionString);
    var sql = "UPDATE Products SET Price = @Price WHERE Id = @Id";
    // Returns number of affected rows
    return await connection.ExecuteAsync(sql, new { Id = id, Price = newPrice });
}
```

ExecuteScalar

Untuk mendapatkan single value:

```
public async Task<int> GetProductCount()
{
    using var connection = new SqlConnection(_connectionString);
    var sql = "SELECT COUNT(*) FROM Products";
    return await connection.ExecuteScalarAsync<int>(sql);
}
```

3.2 Parameterized Queries

3. Query Execution

Anonymous Object Parameters:

```
var product = await connection.QuerySingleAsync<Product>(
    "SELECT * FROM Products WHERE Id = @Id AND Price > @MinPrice",
    new { Id = 1, MinPrice = 100 }
);
```

Object Parameters:

```
var parameters = new Product { Id = 1, Name = "New Product" };
await connection.ExecuteAsync(
    "UPDATE Products SET Name = @Name WHERE Id = @Id",
    parameters
);
```

DynamicParameters:

```
var parameters = new DynamicParameters();
parameters.Add("@Id", 1);
parameters.Add("@Name", "Product Name");
parameters.Add("@TotalCount", dbType: DbType.Int32, direction: ParameterDirection.Output);
await connection.ExecuteAsync(
    "GetProductById",
    parameters,
    commandType: CommandType.StoredProcedure
);
int totalCount = parameters.Get<int>("@TotalCount");
```

3.3 Bulk Operations

3. Query Execution

Bulk Insert:

```
public async Task<int> BulkInsertProducts(IEnumerable<Product> products)
{
    using var connection = new SqlConnection(_connectionString);
    var sql = @"
        INSERT INTO Products (Name, Price, Stock, CreatedAt)
        VALUES (@Name, @Price, @Stock, @CreatedAt)";
    return await connection.ExecuteAsync(sql, products);
}
```

Bulk Update:

```
public async Task<int> BulkUpdatePrices(IEnumerable<Product> products)
{
    using var connection = new SqlConnection(_connectionString);
    var sql = "UPDATE Products SET Price = @Price WHERE Id = @Id";
    return await connection.ExecuteAsync(sql, products);
}
```

3.4 Multiple Result Sets

3. Query Execution

```
public async Task<(IEnumerable<Product>, IEnumerable<Customer>)> GetProductsAndCustomers()
```

```
{
    using var connection = new SqlConnection(_connectionString);
    var sql = @"
        SELECT * FROM Products;
        SELECT * FROM Customers;";
    using var multi = await connection.QueryMultipleAsync(sql);
    var products = await multi.ReadAsync<Product>();
    var customers = await multi.ReadAsync<Customer>();
    return (products, customers);
}
```

3.5 Stored Procedures

3. Query Execution

```
public async Task<IEnumerable<Product>> GetProductsByCategory(int categoryId)
{
    using var connection = new SqlConnection(_connectionString);
    var parameters = new { CategoryId = categoryId };
    return await connection.QueryAsync<Product>(
        "sp_GetProductsByCategory",
        parameters,
        commandType: CommandType.StoredProcedure
    );
}
```

4.1 SQL Injection Prevention

4. RAW SQL Best Practice

■ JANGAN LAKUKAN INI:

```
// BAHAYA: SQL Injection vulnerability
public async Task<Product> GetProductByName(string name)
{
    var sql = $"SELECT * FROM Products WHERE Name = '{name}'";
    return await connection.QueryFirstAsync<Product>(sql);
}
```

■ LAKUKAN INI:

```
// AMAN: Gunakan parameters
public async Task<Product> GetProductByName(string name)
{
    var sql = "SELECT * FROM Products WHERE Name = @Name";
    return await connection.QueryFirstOrDefaultAsync<Product>(sql, new { Name = name });
}
```

4.2 Query Optimization

4. RAW SQL Best Practice

Gunakan SELECT Specific Columns:

```
// ■ Hindari SELECT *
var sql = "SELECT * FROM Products";
// ■ Specify kolom yang dibutuhkan
var sql = "SELECT Id, Name, Price FROM Products";
```

Gunakan WHERE untuk Filter:

```
// ■ Filter di database, bukan di aplikasi
var sql = @"
    SELECT Id, Name, Price
    FROM Products
    WHERE Price > @MinPrice
    AND Stock > 0";
```

Gunakan INDEX dengan baik:

```
// Pastikan kolom di WHERE clause memiliki index
var sql = @"
    SELECT * FROM Orders
    WHERE CustomerId = @CustomerId  -- Index on CustomerId
    AND OrderDate >= @StartDate";    -- Index on OrderDate
```

4.3 Transaction Management

4. RAW SQL Best Practice

Simple Transaction:

```
public async Task<bool> TransferStock(int fromProductId, int toProductId, int quantity)
{
    using var connection = new SqlConnection(_connectionString);
    connection.Open();
    using var transaction = connection.BeginTransaction();
    try
    {
        // Kurangi stock produk pertama
        var sqlDeduct = @"
            UPDATE Products
            SET Stock = Stock - @Quantity
            WHERE Id = @Id AND Stock >= @Quantity";
```

4.4 Connection Management

4. RAW SQL Best Practice

■ BEST PRACTICE: Using Statement

```
public async Task<Product> GetProduct(int id)
{
    using var connection = new SqlConnection(_connectionString);
    // Connection otomatis dibuka oleh Dapper
    // Connection otomatis ditutup setelah using block
    var sql = "SELECT * FROM Products WHERE Id = @Id";
    return await connection.QuerySingleAsync<Product>(sql, new { Id = id });
}
```

■ HINDARI: Manual Connection Management

```
// Risiko connection leak jika terjadi exception
public async Task<Product> GetProduct(int id)
{
    var connection = new SqlConnection(_connectionString);
    connection.Open();
    var sql = "SELECT * FROM Products WHERE Id = @Id";
    var product = await connection.QuerySingleAsync<Product>(sql, new { Id = id });
    connection.Close(); // Might not be called if exception occurs
    return product;
}
```

4.5 Error Handling

4. RAW SQL Best Practice

```
public async Task<Product> GetProductSafe(int id)
{
    try
    {
        using var connection = new SqlConnection(_connectionString);
        var sql = "SELECT * FROM Products WHERE Id = @Id";
        return await connection.QuerySingleOrDefaultAsync<Product>(sql, new { Id = id });
    }
    catch (SqlException ex)
    {
        // Log error
        _logger.LogError(ex, "Database error while getting product {ProductId}", id);
        throw new DataAccessException("Error accessing product data", ex);
```

4.6 Async/Await Best Practice

4. RAW SQL Best Practice

```
// ■ Gunakan async/await
public async Task<IEnumerable<Product>> GetProductsAsync()
{
    using var connection = new SqlConnection(_connectionString);
    var sql = "SELECT * FROM Products";
    return await connection.QueryAsync<Product>(sql);
}
// ■ Jangan block async dengan .Result atau .Wait()
public IEnumerable<Product> GetProducts()
{
    using var connection = new SqlConnection(_connectionString);
```

4.7 Performance Tips

4. RAW SQL Best Practice

1. Reuse Connection String:

```
// ■ Inject connection string, bukan create setiap kali
public class ProductRepository
{
    private readonly string _connectionString;
    public ProductRepository(string connectionString)
    {
        _connectionString = connectionString;
    }
}
```

2. Use Buffered Queries (Default):

```
// Default: buffered = true (load semua ke memory)
var products = await connection.QueryAsync<Product>(sql);
// Untuk dataset besar, gunakan buffered = false (streaming)
var products = await connection.QueryAsync<Product>(sql, buffered: false);
```

3. Use CommandTimeout untuk long-running queries:

```
var sql = "EXEC sp_LongRunningProcedure";
var result = await connection.QueryAsync<Result>(
    sql,
    commandTimeout: 300  // 5 minutes
);
```

Hands-On Project

Lihat folder [hands-on-project](./hands-on-project/) untuk project lengkap Web API menggunakan Dapper.

Project meliputi:

• ■ .NET Core 8 Web API

• ■ Dapper untuk Data Access

• ■ Repository Pattern

• ■ CRUD Operations

• ■ Result Mapping Examples

• ■ Transaction Management

• ■ Error Handling

• ■ Best Practices Implementation

Cara menjalankan project:

```
cd hands-on-project
dotnet restore
dotnet build
dotnet run
```

Akses API di: https://localhost:7001/swagger

Referensi

• [Dapper Official Documentation](https://github.com/DapperLib/Dapper)

• [Dapper Tutorial](https://www.learndapper.com/)

• [Clean Architecture](https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html)

• [Repository Pattern](https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design)

Latihan/Tugas

Kerjakan latihan pada folder [hands-on-project](./hands-on-project/ASSIGNMENT.md) untuk menguji pemahaman Anda tentang:

1. Result Mapping dengan berbagai skenario

2. Implementasi arsitektur yang baik

3. Query execution yang efisien

4. RAW SQL best practices

Selamat belajar! ■