**DAPPER**

Open source ORM library for .NET and .Net core applications

This allows to quickly and easily access data from databases without need to write tedious code

Allows to execute raw sql queries, map results to objects, execute stored procedures, etc

Available in NuGet packages

Dapper is light weight, fast ( so a ideal choice for applications that require low latency and high performance)

Latency refers to the time delay between the initiation of a process or action and its completion.

Powerful tool that allows developers to map the query results from ADO net data readers to instances of business objects

Supports both asynchronous and synchronous database queries and batching multiple queries into single call

Also supports parameterized queries to help protect against SQL injection attacks

Dapper is cross platform so can be used in .net and .net core

When to use Dapper?

* Good choice in scenarios where read-only data changes frequently and is requested often. Good in stateless scenarios where there is no need to save complex data into db
* It is also good to use when the database is not particularly normalised

**Dapper vs Entity framework**

Dapper: Give you full control over the SQL generated / uses stored procedures for everything.

Entity Framework: Allow you to code with LINQ and forget everything about SQL.

**Database providers with dapper**

* provides the user with several methods and extension points to extend the library for their custom database provider.
* Dapper SQL server

1. Install Microsoft.Data.SqlClient Nuget package
2. Write the program

Example:

string connectionString = "Server=.\\MSSQLSERVER01;Initial Catalog=Article;Integrated Security=true;TrustServerCertificate=True";

public ICollection<Article> GetArticles()

{

using(var connection=new SqlConnection(connectionString))

{

var sql = "select \* from ArticleTable";

var articles = connection.Query<Article>(sql);

return articles.ToList();

}

}

**Qyerying Scalar values**

ExecuteScalar Returns the first column of the first row as a dynamic type

ExecuteScalar<T> Returns the first column of the first row as the specified T type parameter

ExecuteScalarAsync Returns the first column of the first row as a dynamic type asynchronously

ExecuteScalarAsync<T> Returns the first column of the first row as the specified T type parameter asynchronously

using(var connection=new SqlConnection(connectionString))

{

var sql = "select count(\*) from ArticleTable";

int count = connection.ExecuteScalar<int>(sql);

return count;

}

**Query Single row**

* QuerySingle – returns the single row matching the query
* QuerySingleOrDefault – returns the default value if no matching row
* QueryFirst – Returns the first row that matches the condition
* QueryFirstOrDefault – returns the first row that matches the condition or gives the default value
* QuerySingle<T>, QueryFirst<T> : Specifies the type of row

using(var connection=new SqlConnection(connectionString))

{

var sql = "SELECT \* FROM ArticleTable WHERE articleName = @articleName";

// Define parameters using an anonymous type or a parameter object

var parameters = new { articleName = name };

// Execute the query and retrieve the count

Article article = connection.QuerySingle<Article>(sql, parameters);

return article;

}

**Querying Multiple rows**

Query Returns an enumerable of dynamic types

Query<T> Returns an enumerable of the type specified by the T parameter

QueryAsync Returns an enumerable of dynamic types asynchronously

QueryAsync<T> Returns an enumerable of the type specified by the T parameter asynchronously

using(var connection=new SqlConnection(connectionString))

{

var sql = "select \* from ArticleTable";

var articles = connection.Query<Article>(sql);

return articles.ToList();

}

**Querying Non-Query**

using(var connection=new SqlConnection(connectionString))

{

string articleName = article.articleName;

string content= article.content;

DateTime publishedDate = article.publishedDate;

DateTime lastModifiedDate= article.lastModifiedDate;

string sql = "insert into ArticleTable(articleName,content,publishedDate,lastModifiedDate)" +

" values (@articleName,@content,@publishedDate,@lastModifiedDate)";

var parameters=new { articleName = articleName, content = content,

publishedDate = publishedDate, lastModifiedDate = lastModifiedDate };

int affectedRows=connection.Execute(sql, parameters);

if (affectedRows>0)

{

return true;

}

else

{

return false;

}

}

**Querying with Stored procedures**

using(var connection = new SqlConnection(connectionString))

{

//Set up DynamicParameters object to pass parameters

DynamicParameters parameters = new DynamicParameters();

parameters.Add("id", 1);

//Execute stored procedure and map the returned result to a Customer object

var customer = conn.QuerySingleOrDefault<Customer>("GetCustomerById", parameters, commandType: CommandType.StoredProcedure);

}