Aula 9 - Conexão com Banco de Dados, Object-Relational Mapping e Entity Framework

Docupedia Export

Author:Silio Leonardo (CtP/ETS)
Date:31-May-2023 13:53

Table of Contents

- Conectando o C# ao SQLServer
- Sql Injection
- Exemplo: Criando um ORM
- Entity Framework
- DBFirst, CodeFirst e Scaffolding
- Exercícios

1 Conectando o C# ao SQLServer

Existem várias maneiras de conectar o C# ao banco de dados, vamos a mais clássica delas. Inicialmente, podemos executar o seguinte script que cria um novo projeto e instá-la uma biblioteca para tal. Essa biblioteca é antiga e antigamente vinha junto com o .NET. Hoje em dia, em versões novas, é necessário usar o nuget para obtê-la:

```
mkdir BancoExemplo
cd BancoExemplo
dotnet new console
dotnet add package System.Data.SqlClient
```

Agora vamos a utilização dela. Inicialmente não criaremos nenhuma classe, vamos ao seu uso direto. Abaixo o script do SQl que usaremos:

```
use master
go

if exists(select * from sys.databases where name = 'example')
    drop database example

create database example
go

use example
go

create table Cliente(
    ID int identity primary key,
    Nome varchar(100) not null,
    Senha varchar(100) not null,
    DataNasc date not null
);
go
```

```
using System.Data.SqlClient;

SqlConnectionStringBuilder stringConnectionBuilder = new SqlConnectionStringBuilder();
stringConnectionBuilder.DataSource = @"JVLPC0480\SQLEXPRESS"; // Nome do servidor
stringConnectionBuilder.InitialCatalog = "example"; // Nome do banco
stringConnectionBuilder.IntegratedSecurity = true;
```

```
7   string stringConnection = stringConnectionBuilder.ConnectionString;
8
9   SqlConnection conn = new SqlConnection(stringConnection);
10  conn.Open();
11
12  SqlCommand comm = new SqlCommand("insert Cliente values ('Pamella', '123', CONVERT(DATETIME, '03/27/2023'));");
13  comm.Connection = conn;
14  comm.ExecuteNonQuery();
15
16  conn.Close();
```

Ao executarmos um select podemos ver que o cliente foi criado com sucesso. Podemos usar a mesma ideai junto de um DataTable para ler dados.

```
1   using System;
2   using System.Data;
3   using System.Data.SqlClient;
4
5   SqlConnectionStringBuilder stringConnectionBuilder = new SqlConnectionStringBuilder();
6   stringConnectionBuilder.DataSource = @"JVLPC0480\SQLEXPRESS";
7   stringConnectionBuilder.InitialCatalog = "example";
8   stringConnectionBuilder.IntegratedSecurity = true;
9   string stringConnection = stringConnectionBuilder.ConnectionString;
10
11  SqlConnection conn = new SqlConnection(stringConnection);
12  conn.Open();
13
14  string nome = Console.ReadLine();
15  string senha = Console.ReadLine();
16
17  SqlCommand comm = new SqlCommand($"select * from Cliente where Nome = '{nome}' and Senha = '{senha}'");
18  comm.Connection = conn;
19  var reader = comm.ExecuteReader();
20
21  DataTable dt = new DataTable();
22  dt.Load(reader);
23
24  if (dt.Rows.Count > 0)
25      Console.WriteLine($"Usuário {dt.Rows[0].ItemArray[0]} Logado");
26  else
```

```
27      Console.WriteLine("Conta inexistente");
28
29  conn.Close();
```

Ao executarmos podemos fazer uma espécie de Login.

2 Sql Injection

Temos que tomar cuidado com alguns tipos de abordagens ao usarmos bibliotecas de acesso ao banco de dados. Uma delas é o Sql Injection. No Sql Injection, uma pessoa mal intencionada pode inserir SQL em nossa query fechando as aspas da linha 17 e colocando código malicioso.

```
1   dotnet run
2   ' or 1 = 1 --
3
4   Usuário 3 Logado
```

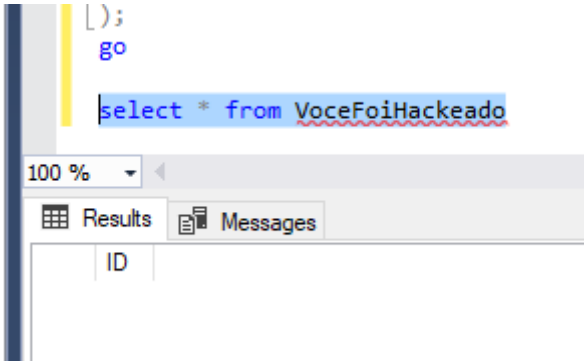
Explicando, ao digitarmos aquele código na primeira linha fazemos a seguinte query SQL na linha 17:

```
select * from Cliente where Nome = '' or 1 = 1 --' and Senha = '{senha}'
```

Basicamente ele selecionará todos os clientes onde o nome for vazio ou 1 for igual a 1, o que é sempre verdadeiro. O resto da query será ignorada como temos um comentário. Assim o hacker invade o sistema. Mas isso não é tudo de ruim que pode ser feito. Observe:

```
dotnet run
'; create table VoceFoiHackeado ( ID int ); --

Conta inexistente
```



Além disso podemos excluir dados também ou qualquer coisa que você imaginar:

```
dotnet run
'; delete Cliente; --
```

```
Conta inexistente
```

Por isso podemos melhorar o uso do nosso sistema para evitar tais debilidades. O código abaixo bloqueia várias possibilidades de se afetar o software com sql injection.

```
1  using System;
2  using System.Data;
3  using System.Data.SqlClient;
4
5  SqlConnectionStringBuilder stringConnectionBuilder = new SqlConnectionStringBuilder();
6  stringConnectionBuilder.DataSource = @"JVLPC0480\SQLEXPRESS";
7  stringConnectionBuilder.InitialCatalog = "example";
8  stringConnectionBuilder.IntegratedSecurity = true;
9  string stringConnection = stringConnectionBuilder.ConnectionString;
10
11 SqlConnection conn = new SqlConnection(stringConnection);
12 conn.Open();
13
14 string nome = Console.ReadLine();
15 string senha = Console.ReadLine();
16
17 SqlCommand comm = new SqlCommand($"select * from Cliente where Nome = @Nome and Senha = @Senha");
18 comm.Connection = conn;
19
20 comm.Parameters.Add(new SqlParameter("@Nome", nome));
21 comm.Parameters.Add(new SqlParameter("@Senha", senha));
22
23 var reader = comm.ExecuteReader();
24
25 comm.Parameters.Clear(); // Limpar depois de usar permite reutilizar o SqlCommand
26
27 DataTable dt = new DataTable();
28 dt.Load(reader);
29
30 if (dt.Rows.Count > 0)
31     Console.WriteLine($"Usuário {dt.Rows[0].ItemArray[0]} Logado");
32 else
33     Console.WriteLine("Conta inexistente");
34
```

```
35  conn.Close();
```

3 Exemplo: Criando um ORM

ORMLib/DataAnnotations/ForeignKeyAttribute.cs

```
1  using System;
2
3  namespace ORMLib.DataAnnotations;
4
5  // No C# 11.0 em diante poderemos usar Atributos Genéricos
6  public class ForeignKeyAttribute : Attribute
7  {
8      public Type ForeignTable { get; set; }
9      public ForeignKeyAttribute(Type foreignTable)
10         => this.ForeignTable = foreignTable;
11 }
```

ORMLib/DataAnnotations/NotNullAttribute.cs

```
1  using System;
2
3  namespace ORMLib.DataAnnotations;
4
5  public class NotNullAttribute : Attribute { }
```

ORMLib/DataAnnotations/PrimaryKeyAttribute.cs

```
1  using System;
2
3  namespace ORMLib.DataAnnotations;
4
5  public class PrimaryKeyAttribute : Attribute { }
```

ORMLib/Exceptions/ConfigNotInitializedException.cs

```cs
using System;

namespace ORMLib.Exceptions;

public class ConfigNotInitializedException : Exception
{
    public override string Message => "ORM config is not initializated";
}
```

ORMLib/Exceptions/InvalidColumnType.cs

```cs
using System;

namespace ORMLib.Exceptions;

public class InvalidColumnType : Exception
{
    string type;

    public InvalidColumnType(Type type)
        => this.type = type.Name;

    public override string Message => $"O tipo {type} não é valido para uma coluna no banco de dados.";
}
```

ORMLib/ObjectRelationalMappingConfig.cs

```cs
#pragma warning disable CS1998

using System.Threading.Tasks;

namespace ORMLib;
```

```
6
7   using Exceptions;
8   using Providers;
9
10  public class ObjectRelationalMappingConfig
11  {
12      private static ObjectRelationalMappingConfig config = null;
13      public static ObjectRelationalMappingConfig Config
14      {
15          get
16          {
17              if (config == null)
18                  throw new ConfigNotInitializedException();
19
20              return config;
21          }
22      }
23
24      public static ObjectRelationalMappingConfigBuilder GetBuilder()
25          => new ObjectRelationalMappingConfigBuilder();
26
27      public virtual DataBaseSystem DataBaseSystem { get; set; }
28      public virtual string StringConnection { get; set; }
29      public virtual string InitialCatalog { get; set; }
30      public IQueryProvider QueryProvider { get; set; }
31      public AccessProvider AccessProvider { get; set; }
32
33      public ObjectRelationalMappingConfig(
34          DataBaseSystem dataBaseSystem,
35          string stringConnection,
36          string initialCatalog,
37          IQueryProvider queryProvider,
38          AccessProvider accessProvider
39      )
40      {
41          this.DataBaseSystem = dataBaseSystem;
42          this.StringConnection = stringConnection;
43          this.InitialCatalog = initialCatalog;
44          this.QueryProvider = queryProvider;
```

```
45          this.AccessProvider = accessProvider;
46      }
47
48      public void Use()
49          => config = this;
50
51      public virtual async Task UseAsync()
52          => Use();
53  }
```

ORMLib/ObjectRelationalMappingConfigBuilder.cs

```
1   using System.Data.SqlClient;
2
3   namespace ORMLib;
4
5   using Providers;
6
7   public class ObjectRelationalMappingConfigBuilder
8   {
9       private DataBaseSystem dataBaseSystem;
10      private SqlConnectionStringBuilder stringConnectionBuilder = new SqlConnectionStringBuilder();
11      private IQueryProvider queryProvider;
12      private AccessProvider accessProvider;
13
14      public ObjectRelationalMappingConfigBuilder SetDataBaseSystem(DataBaseSystem sys)
15      {
16          this.dataBaseSystem = sys;
17          return this;
18      }
19
20      public ObjectRelationalMappingConfigBuilder SetDataSource(string serverName)
21      {
22          stringConnectionBuilder.DataSource = serverName;
23          return this;
24      }
25
26      public ObjectRelationalMappingConfigBuilder SetInitialCatalog(string initialCatalog)
```

```
27      {
28          stringConnectionBuilder.InitialCatalog = initialCatalog;
29          return this;
30      }
31
32      public ObjectRelationalMappingConfigBuilder SetIntegratedSecurity(bool integratedSecurity)
33      {
34          stringConnectionBuilder.IntegratedSecurity = integratedSecurity;
35          return this;
36      }
37
38      public ObjectRelationalMappingConfigBuilder SetStringConnection(string strConn)
39      {
40          stringConnectionBuilder.ConnectionString = strConn;
41          return this;
42      }
43
44      public ObjectRelationalMappingConfigBuilder SetQueryProvider(IQueryProvider provider)
45      {
46          this.queryProvider = provider;
47          return this;
48      }
49
50      public ObjectRelationalMappingConfigBuilder SetAccessProvider(AccessProvider provider)
51      {
52          this.accessProvider = provider;
53          return this;
54      }
55
56      public ObjectRelationalMappingConfig Build()
57      {
58          ObjectRelationalMappingConfig config = new ObjectRelationalMappingConfig(
59              this.dataBaseSystem,
60              this.stringConnectionBuilder.ConnectionString,
61              this.stringConnectionBuilder.InitialCatalog,
62              this.queryProvider,
63              this.accessProvider
64          );
65          return config;
```

```
    }
}
```

ORMLib/ObjectRelationalMappingConfigBuilderExtension.cs

```
namespace ORMLib;

using MSSql;

public static class ObjectRelationalMappingConfigBuilderExtension
{
    public static ObjectRelationalMappingConfigBuilder UseMSSqlServer(this ObjectRelationalMappingConfigBuilder builder)
    {
        builder.SetDataBaseSystem(DataBaseSystem.SqlServer);
        builder.SetQueryProvider(new MSSqlQueryProvider());
        builder.SetAccessProvider(new MSSqlProvider());
        return builder;
    }
}
```

ORMLib/DataBaseSystem.cs

```
namespace ORMLib;

public enum DataBaseSystem
{
    SqlServer,
    Oracle,
    MariaDB,
    MySql
}
```

ORMLib/Access.cs

```
1  using System.Threading.Tasks;
2
3  namespace ORMLib;
4
5  public abstract class Access
6  {
7      private static Access instance = null;
8      public static Access Instance
9      {
10         get
11         {
12             if (instance is not null)
13                 return instance;
14
15             var provider = ObjectRelationalMappingConfig.Config.AccessProvider;
16             instance = provider.Provide();
17
18             return instance;
19         }
20     }
21
22     public abstract Task Insert<T>(T obj);
23     public abstract Task Delete<T>(T obj);
24     public abstract Task Update<T>(T obj);
25 }
```

ORMLib/Table.cs

```
1  using System.Threading.Tasks;
2  using System.Linq.Expressions;
3
4  namespace ORMLib;
5
6  using Linq;
```

```
7
8  public abstract class Table<T>
9      where T : class, new()
10 {
11     private bool exist = false;
12
13     public async Task Save()
14     {
15         var obj = this as T;
16
17         if (this.exist)
18             await Access.Instance.Update(obj);
19         else await Access.Instance.Insert(obj);
20
21         this.exist = true;
22     }
23
24     public async Task Delete()
25     {
26         var obj = this as T;
27         await Access.Instance.Delete(obj);
28         this.exist = false;
29     }
30
31     public static IQueryable<T> All
32     {
33         get
34         {
35             var provider = ObjectRelationalMappingConfig.Config.QueryProvider;
36             var empty = provider.CreateQuery<T>(null);
37             return provider.CreateQuery<T>(Expression.Constant(empty));
38         }
39     }
40 }
```

ORMLib/MSSql/MSSqlProvider.cs

```
1  namespace ORMLib.MSSql;
```

```
2
3   using Providers;
4
5   public class MSSqlProvider : AccessProvider
6   {
7       public override Access Provide()
8           => new SqlAccess();
9   }
```

ORMLib/MSSql/SqlAccess.cs

```
1   using System;
2   using System.Linq;
3   using System.Data;
4   using System.Reflection;
5   using System.Data.SqlClient;
6   using System.Threading.Tasks;
7   using System.Collections.Generic;
8
9   namespace ORMLib.MSSql;
10
11  using Exceptions;
12  using DataAnnotations;
13  using System.Collections;
14
15  internal class SqlAccess : Access
16  {
17      private SqlCommand comm;
18      private SqlConnection conn;
19      private bool loaded = false;
20      private bool exist = false;
21
22      public async Task CreateDataBaseIfNotExistAsync()
23      {
24          if (exist)
25              return;
26          exist = true;
27
```

```
28          var config = ObjectRelationalMappingConfig.Config;
29          var masterStrConn = config.StringConnection.Replace(
30              config.InitialCatalog,
31              "master"
32          );
33          var conn = new SqlConnection(masterStrConn);
34          await conn.OpenAsync();
35
36          var comm = new SqlCommand();
37          comm.CommandText = $"select * from sys.databases where name = '{config.InitialCatalog}'";
38          comm.Connection = conn;
39          comm.CommandType = CommandType.Text;
40
41          var reader = await comm.ExecuteReaderAsync();
42          var dt = new DataTable();
43          dt.Load(reader);
44
45          if (dt.Rows.Count > 0)
46          {
47              await conn.CloseAsync();
48              return;
49          }
50
51          comm.CommandText = $"create database {config.InitialCatalog}";
52          await comm.ExecuteNonQueryAsync();
53          await conn.CloseAsync();
54      }
55
56      public async Task LoadAsync()
57      {
58          if (loaded)
59              return;
60
61          await CreateDataBaseIfNotExistAsync();
62
63          var config = ObjectRelationalMappingConfig.Config;
64          conn = new SqlConnection(config.StringConnection);
65          await conn.OpenAsync();
66
```

```
67          comm = new SqlCommand();
68          comm.Connection = conn;
69          comm.CommandType = CommandType.Text;
70
71          loaded = true;
72      }
73
74      public async Task CreateIfNotExistAsync(Type type)
75      {
76          await LoadAsync();
77
78          if (await TestExistenceAsync(type))
79              return;
80
81          comm.CommandText = $"create table {type.Name} (";
82
83          foreach (var prop in type.GetProperties())
84          {
85              string column = $"{prop.Name} {ConvertToSqlType(prop.PropertyType)}";
86
87              if (prop.Name == "ID")
88                  column += " identity primary key";
89
90              var foreignKeyAtt = prop.GetCustomAttribute<ForeignKeyAttribute>();
91              if (foreignKeyAtt != null)
92              {
93                  string temp = comm.CommandText;
94                  await CreateIfNotExistAsync(foreignKeyAtt.ForeignTable);
95                  comm.CommandText = temp;
96                  column += $" references {foreignKeyAtt.ForeignTable.Name}(ID)";
97              }
98
99              var notnullAtt = prop.GetCustomAttribute<NotNullAttribute>();
100
101             if (notnullAtt != null)
102                 column += $" not null";
103
104             comm.CommandText += $"{column}, ";
105         }
```

```
106             comm.CommandText = comm.CommandText.Substring(0, comm.CommandText.Length - 1) + " )";
107
108             await comm.ExecuteNonQueryAsync();
109         }
110
111         public string ConvertToSqlType(Type type)
112         {
113             if (type == typeof(int))
114                 return "int";
115
116             if (type == typeof(string))
117                 return "varchar(MAX)";
118
119             if (type == typeof(byte[]))
120                 return "varbinary";
121
122             if (type == typeof(decimal))
123                 return "decimal";
124
125             if (type == typeof(long))
126                 return "bigint";
127
128             if (type == typeof(DateTime))
129                 return "datetime";
130
131             throw new InvalidColumnType(type);
132         }
133
134         public async Task<bool> TestExistenceAsync(Type type)
135         {
136             DataTable dt = await ReadTableAsync($"select * from sys.tables where name = '{type.Name}'");
137             return dt.Rows.Count > 0;
138         }
139
140         public async Task<DataTable> ReadTableAsync(string query, params SqlParameter[] parameters)
141         {
142             await LoadAsync();
143             comm.CommandText = query;
144             comm.Parameters.Clear();
```

```
145             comm.Parameters.AddRange(parameters);
146
147             var reader = await comm.ExecuteReaderAsync();
148
149             DataTable dt = new DataTable();
150             dt.Load(reader);
151
152             return dt;
153         }
154
155         public async Task ExecuteNonQueryAsync(string query, params SqlParameter[] parameters)
156         {
157             comm.CommandText = query;
158             comm.Parameters.Clear();
159             comm.Parameters.AddRange(parameters);
160
161             await comm.ExecuteNonQueryAsync();
162         }
163
164         public async Task<T> RunQuery<T>(string query, params SqlParameter[] parameters)
165         {
166             var type = typeof(T);
167             var dt = await ReadTableAsync(query, parameters);
168             var isCollection = typeof(IEnumerable).IsAssignableFrom(type);
169
170             if (isCollection)
171             {
172                 var args = type.GetGenericArguments();
173                 if (args.Length > 0)
174                     type = args[0];
175             }
176
177             int i = 0;
178             object[] data = new object[dt.Rows.Count];
179
180             foreach (DataRow row in dt.Rows)
181             {
182                 if (row.ItemArray.Length == 1)
183                 {
```

```
                data[i++] = row.ItemArray[0];
                continue;
            }

            var obj = Activator.CreateInstance(type);

            foreach (var prop in type.GetProperties())
                prop.SetValue(obj, row[prop.Name]);

            data[i++] = obj;
        }

        if (isCollection)
        {
            var listType = typeof(List<>).MakeGenericType(type);
            var list = (IList)Activator.CreateInstance(listType);

            foreach (var x in data)
                list.Add(Convert.ChangeType(x, type));

            return (T)list;
        }

        return (T)data[0];
    }

    public override async Task Insert<T>(T obj)
    {
        await CreateIfNotExistAsync(typeof(T));

        var id = getID<T>();

        List<SqlParameter> parameters = new List<SqlParameter>();
        string query = $"insert {typeof(T).Name} values (";

        foreach (var prop in typeof(T).GetProperties())
        {
            if (prop.Name == id?.Name)
                continue;
```

```
223
224             var paramName =  "@" + prop.Name;
225             query += paramName + ",";
226             parameters.Add(new SqlParameter(paramName, prop.GetValue(obj)));
227         }
228
229         query = query.Substring(0, query.Length - 1) + ")";
230         await ExecuteNonQueryAsync(query, parameters.ToArray());
231     }
232
233     public override async Task Delete<T>(T obj)
234     {
235         await CreateIfNotExistAsync(typeof(T));
236
237         var id = getID<T>();
238
239         await ExecuteNonQueryAsync(
240             $"delete {typeof(T).Name} where {id.Name} == @{id.Name}",
241             new SqlParameter($"@{id.Name}", id.GetValue(obj))
242         );
243     }
244
245     public override async Task Update<T>(T obj)
246     {
247         await CreateIfNotExistAsync(typeof(T));
248
249         var id = getID<T>();
250
251         List<SqlParameter> parameters = new List<SqlParameter>();
252         string query = $"update {typeof(T).Name} set \n";
253
254         foreach (var prop in typeof(T).GetProperties())
255         {
256             if (prop.Name == id?.Name)
257                 continue;
258
259             var paramName =  "@" + prop.Name;
260             query += paramName + ",";
261             parameters.Add(new SqlParameter(paramName, prop.GetValue(obj)));
```

```
262             }
263
264             query += $" where {id.Name} == @{id.Name}";
265             parameters.Add(new SqlParameter($"@{id.Name}", id.GetValue(obj)));
266
267             await ExecuteNonQueryAsync(query, parameters.ToArray());
268         }
269
270         private PropertyInfo getID<T>()
271         {
272             foreach (var prop in typeof(T).GetProperties())
273             {
274                 if (prop.GetCustomAttribute<PrimaryKeyAttribute>() is null)
275                     continue;
276
277                 return prop;
278             }
279
280             return null;
281         }
282     }
```

ORMLib/MSSqlQueryable.cs

```
1   using System;
2   using System.Linq.Expressions;
3   using System.Collections.Generic;
4
5   namespace ORMLib.MSSql;
6
7   using Providers;
8   using Linq;
9
10  public class MSSqlQueryable<T> : IQueryable<T>
11  {
12      public MSSqlQueryable(Expression exp, MSSqlQueryProvider provider)
13      {
14          this.ElementType = typeof(IEnumerable<T>);
```

```
        this.Expression = exp;
        this.Provider = provider;
    }

    public Type ElementType { get; private set; }

    public Expression Expression { get; private set; }

    public IQueryProvider Provider { get; private set; }
}
```

ORMLib/MSSql/MSSqlQueryProvider.cs

```
using System.Linq;
using System.Threading.Tasks;
using System.Linq.Expressions;
using System.Collections.Generic;

namespace ORMLib.MSSql;

using System.Data.SqlClient;
using Linq;
using Providers;

public class MSSqlQueryProvider : IQueryProvider
{
    public IQueryable<T> CreateQuery<T>(Expression expression)
        => new MSSqlQueryable<T>(expression, this);

    public async Task<T> Execute<T>(Expression expression)
    {
        SqlAccess access = new SqlAccess();
        List<SqlParameter> list = new List<SqlParameter>();
        var query = buildQuery<T>(expression, list);
        // System.Console.WriteLine(query);
        var data = await access.RunQuery<T>(query, list.ToArray());
        return data;
    }
```

```
26
27      private string buildQuery<T>(Expression expression, List<SqlParameter> parameters)
28      {
29          string query = "";
30
31          if (expression is MethodCallExpression call)
32          {
33              if (call.Method.Name == "Select" && call.Method.DeclaringType == typeof(Queryable))
34              {
35                  query = buildQuery<T>(call.Arguments[0], parameters);
36                  if (call.Arguments[1] is UnaryExpression unary)
37                  {
38                      var exp = unary.Operand;
39                      var paramter = exp.ToString().Split(' ')[0];
40                      var str = string.Concat(
41                          exp.ToString()
42                          .SkipWhile(c => c != '>')
43                          .Skip(1)
44                      );
45                      query = query.Replace("*", str) + " " + paramter;
46                  }
47              }
48
49              if (call.Method.Name == "Where" && call.Method.DeclaringType == typeof(Queryable))
50              {
51                  query = buildQuery<T>(call.Arguments[0], parameters);
52                  query = buildQuery<T>(call.Arguments[0], parameters);
53                  if (call.Arguments[1] is UnaryExpression unary)
54                  {
55                      var exp = unary.Operand;
56                      var paramter = exp.ToString().Split(' ')[0];
57                      var str = string.Concat(
58                          exp.ToString()
59                          .SkipWhile(c => c != '>')
60                          .Skip(1)
61                      );
62                      str = str.Replace("==", "=");
63                      str = str.Replace("\"", "'");
64                      query = $"{query} {paramter} where {str}";
```

```
                }
            }
        }
        else if (expression is ConstantExpression constExp)
        {
            var type = constExp.Type;
            if (typeof(IQueryable).IsAssignableFrom(type))
            {
                var queryType = type.GenericTypeArguments[0];
                query = $"select * from {queryType.Name}";
            }
        }

        return query;
    }
}
```

ORMLib/Linq/IQueryable.cs

```
using System;
using System.Linq.Expressions;

namespace ORMLib.Linq;

using Providers;

public interface IQueryable
{
    Type ElementType { get; }
    Expression Expression { get; }
    IQueryProvider Provider { get; }
}

public interface IQueryable<out T> : IQueryable
{

}
```

ORMLib/Linq/Queryable.cs

```
using System;
using System.Linq;
using System.Threading.Tasks;
using System.Linq.Expressions;
using System.Collections.Generic;

namespace ORMLib.Linq;

public static class Queryable
{
    public static IQueryable<R> Select<T, R>(this IQueryable<T> source, Expression<Func<T, R>> selector)
        where T : new()
    {
        if (source is null)
            throw new ArgumentNullException("source");

        if (selector is null)
            throw new ArgumentNullException("selector");

        var provider = source.Provider;

        var self = new Func<IQueryable<T>, Expression<Func<T, R>>, IQueryable<R>>(Select);

        var query = provider.CreateQuery<R>(
            Expression.Call(
                null,
                self.Method,
                source.Expression,
                Expression.Quote(selector)
            )
        );

        return query;
    }

```

```
36      public static IQueryable<T> Where<T>(this IQueryable<T> source, Expression<Func<T, bool>> predicate)
37          where T : new()
38      {
39          if (source is null)
40              throw new ArgumentNullException("source");
41
42          if (predicate is null)
43              throw new ArgumentNullException("predicate");
44
45          var provider = source.Provider;
46
47          var self = new Func<IQueryable<T>, Expression<Func<T, bool>>, IQueryable<T>>(Where);
48
49          var query = provider.CreateQuery<T>(
50              Expression.Call(
51                  null,
52                  self.Method,
53                  source.Expression,
54                  Expression.Quote(predicate)
55              )
56          );
57
58          return query;
59      }
60
61      public static async Task<List<T>> ToListAsync<T>(this IQueryable<T> source)
62      {
63          if (source is null)
64              throw new ArgumentNullException("source");
65
66          var exp = source.Expression;
67          var provider = source.Provider;
68
69          var collection = await provider.Execute<IEnumerable<T>>(exp);
70
71          return collection.ToList();
72      }
73  }
```

ORMLib/ORMLib.csproj

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="System.Data.SqlClient" Version="4.8.5" />
  </ItemGroup>

</Project>
```

Test/Program.cs

```
using static System.Console;

using ORMLib;
using ORMLib.Linq;
using ORMLib.DataAnnotations;

var builder = ObjectRelationalMappingConfig.GetBuilder();

builder
    .UseMSSqlServer()
    .SetDataSource(@"JVLPC0480\SQLEXPRESS")
    .SetInitialCatalog("MyDatabaseTest")
    .SetIntegratedSecurity(true)
    .Build()
    .Use();

while (true)
{
    string command = ReadLine();

```

```
21      switch (command.ToLower())
22      {
23          case "add marca":
24              Marca marca = new Marca();
25
26              Write("Nome: ");
27              marca.Nome = ReadLine();
28
29              await marca.Save();
30              break;
31
32          case "add produto":
33              Produto produto = new Produto();
34
35              Write("Nome: ");
36              produto.Nome = ReadLine();
37
38              Write("Marca: ");
39              var marcaNome = ReadLine();
40              var marcasSelecionadas = await Marca.All
41                  .Where(m => m.Nome == "Bosch")
42                  .ToListAsync();
43              var marcaSelecionada = marcasSelecionadas[0];
44              produto.MarcaID = marcaSelecionada.ID;
45
46              await produto.Save();
47              break;
48
49          case "view produto":
50              var produtos = await Produto.All
51                  .Select(m => m.Nome)
52                  .ToListAsync();
53              foreach (var nome in produtos)
54              {
55                  WriteLine(nome);
56              }
57              break;
58
59
```

```
60          case "view marca":
61              var marcas = await Marca.All
62                  .Select(m => m.Nome)
63                  .ToListAsync();
64              foreach (var nome in marcas)
65              {
66                  WriteLine(nome);
67              }
68              break;
69
70          case "exit":
71              return;
72      }
73  }
74
75  public class Anuncio : Table<Anuncio>
76  {
77      [PrimaryKey]
78      public int ID { get; set; }
79
80      [NotNull]
81      public string Titulo { get; set; }
82
83      [NotNull]
84      public decimal Preco { get; set; }
85
86      [NotNull]
87      [ForeignKey(typeof(Vendedor))]
88      public int VendedorID { get; set; }
89
90      [NotNull]
91      [ForeignKey(typeof(Produto))]
92      public int ProdutoID { get; set; }
93  }
94
95  public class Marca : Table<Marca>
96  {
97      [PrimaryKey]
98      public int ID { get; set; }
```

```
 99
100         [NotNull]
101         public string Nome { get; set; }
102     }
103
104     public class Vendedor : Table<Vendedor>
105     {
106         [PrimaryKey]
107         public int ID { get; set; }
108
109         public string Nome { get; set; }
110
111         [NotNull]
112         public string Login { get; set; }
113
114         [NotNull]
115         public string Senha { get; set; }
116
117         public string CPF { get; set; }
118
119         public string PIX { get; set; }
120     }
121
122     public class Produto : Table<Produto>
123     {
124         [PrimaryKey]
125         public int ID { get; set; }
126
127         [NotNull]
128         public string Nome { get; set; }
129
130         [ForeignKey(typeof(Marca))]
131         public int MarcaID { get; set; }
132     }
```

4 Entity Framework

O Entity Framework é o ORM padrão do ,NET. É o framework que usaremos para fazer conexão com o banco de dados. Abaixo um exemplo de como utilizá-lo. A estrutura é semelhante ao que fizermos acima, porém bem mais complexa.

5 DBFirst, CodeFirst e Scaffolding

Para inicio precisamos falar da diferença entre DBFirst e CodeFirst.
Para utilizar o Entity Framework você precisa definir o seu banco através de código manualmente. Indicar as tabelas e as relações. Isso é chamado de CodeFirst. O Banco de dados será gerado com base no seu modelo. Por outro lado podemos fazer Scaffolding e realizar o DBFirst: Escrever o SQL do banco de dados e gerar o código C# que atende os modelos especificados no banco. Caso seja seu desejo, e nesse tutorial, aqui está um script para a instalação do entity, ferramentas de Scaffolding e ainda a utilização no projeto atual:

createModel.ps1

```
$strconn = "Data Source=" + $args[0] + ";Initial Catalog=" + $args[1] + ";Integrated
Security=True;TrustServerCertificate=true"
dotnet add package Microsoft.EntityFrameworkCore.Design
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
dotnet tool install --global dotnet-ef
dotnet ef dbcontext scaffold $strconn Microsoft.EntityFrameworkCore.SqlServer --force -o Model
```

Note que você precisa executar passando logo em seguida uma string com o nome do servidor e depois uma string com o nome do banco. Caso tenha problemas com o proxy, você pode solucionar altearando as configurações do Nuget em %appdata%/Nuget/NuGet.config:

NuGet.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <config>
        <add key="http_proxy" value="http://BWSOA@rb-proxy-de.bosch.com:8080" />
        <add key="https_proxy" value="http://BWSOA@rb-proxy-de.bosch.com:8080" />
    </config>

  <packageSources>
    <add key="nuget.org" value="https://api.nuget.org/v3/index.json" protocolVersion="3" />
  </packageSources>
</configuration>
```

Vamos, antes de tudo criar o seguinte banco de dados para testes:

```
use master
go
```

```
3
4   if exists(select * from sys.databases where name = 'testentity')
5       drop database testentity
6   go
7
8   create database testentity
9   go
10
11  use testentity
12  go
13
14  create table Produto(
15      ID int identity primary key,
16      Nome varchar(100) not null,
17      Descricao varchar(MAX) not null,
18      Foto image null
19  );
20  go
21
22  create table Usuario(
23      ID int identity primary key,
24      Nome varchar(120) not null,
25      DataNascimento date not null,
26      Foto image null
27  );
28  go
29
30  create table Oferta(
31      ID int identity primary key,
32      Produto int references Produto(ID) not null,
33      Usuario int references Usuario(ID) not null,
34      Preco decimal not null
35  );
36  go
```

Após executar o script esperamos o seguinte resultado em arquivos gerados numa pasta Model:

Model/Ofertum.cs

```cs
using System;
using System.Collections.Generic;

namespace entityTest.Model;

public partial class Ofertum
{
    public int Id { get; set; }

    public int Produto { get; set; }

    public int Usuario { get; set; }

    public decimal Preco { get; set; }

    public virtual Produto ProdutoNavigation { get; set; }

    public virtual Usuario UsuarioNavigation { get; set; }
}
```

Model/Produto.cs

```cs
using System;
using System.Collections.Generic;

namespace entityTest.Model;

public partial class Produto
{
    public int Id { get; set; }

    public string Nome { get; set; }

    public string Descricao { get; set; }
```

```
13
14      public byte[] Foto { get; set; }
15
16      public virtual ICollection<Ofertum> Oferta { get; set; } = new List<Ofertum>();
17  }
```

Model/Usuario.cs

```
1   using System;
2   using System.Collections.Generic;
3
4   namespace entityTest.Model;
5
6   public partial class Usuario
7   {
8       public int Id { get; set; }
9
10      public string Nome { get; set; }
11
12      public DateTime DataNascimento { get; set; }
13
14      public byte[] Foto { get; set; }
15
16      public virtual ICollection<Ofertum> Oferta { get; set; } = new List<Ofertum>();
17  }
```

Model/TestentityContext.cs

```
1   using System;
2   using System.Collections.Generic;
3   using Microsoft.EntityFrameworkCore;
4
5   namespace entityTest.Model;
6
7   public partial class TestentityContext : DbContext
8   {
```

```
9       public TestentityContext()
10      {
11      }
12
13      public TestentityContext(DbContextOptions<TestentityContext> options)
14          : base(options)
15      {
16      }
17
18      public virtual DbSet<Ofertum> Oferta { get; set; }
19
20      public virtual DbSet<Produto> Produtos { get; set; }
21
22      public virtual DbSet<Usuario> Usuarios { get; set; }
23
24      protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
25  #warning To protect potentially sensitive information in your connection string, you should move it out of source code.
    You can avoid scaffolding the connection string by using the Name= syntax to read it from configuration - see https://
    go.microsoft.com/fwlink/?linkid=2131148. For more guidance on storing connection strings, see http://go.microsoft.com/
    fwlink/?LinkId=723263.
26          => optionsBuilder.UseSqlServer("Data Source=CT-C-00189\\SQLEXPRESS01;Initial Catalog=testentity;Integrated
    Security=True;TrustServerCertificate=true");
27
28      protected override void OnModelCreating(ModelBuilder modelBuilder)
29      {
30          modelBuilder.Entity<Ofertum>(entity =>
31          {
32              entity.HasKey(e => e.Id).HasName("PK__Oferta__3214EC27449381AC");
33
34              entity.Property(e => e.Id).HasColumnName("ID");
35              entity.Property(e => e.Preco).HasColumnType("decimal(18, 0)");
36
37              entity.HasOne(d => d.ProdutoNavigation).WithMany(p => p.Oferta)
38                  .HasForeignKey(d => d.Produto)
39                  .OnDelete(DeleteBehavior.ClientSetNull)
40                  .HasConstraintName("FK__Oferta__Produto__286302EC");
41
42              entity.HasOne(d => d.UsuarioNavigation).WithMany(p => p.Oferta)
43                  .HasForeignKey(d => d.Usuario)
```

```
                .OnDelete(DeleteBehavior.ClientSetNull)
                .HasConstraintName("FK__Oferta__Usuario__29572725");
        });

        modelBuilder.Entity<Produto>(entity =>
        {
            entity.HasKey(e => e.Id).HasName("PK__Produto__3214EC271CBD74B4");

            entity.ToTable("Produto");

            entity.Property(e => e.Id).HasColumnName("ID");
            entity.Property(e => e.Descricao)
                .IsRequired()
                .IsUnicode(false);
            entity.Property(e => e.Foto).HasColumnType("image");
            entity.Property(e => e.Nome)
                .IsRequired()
                .HasMaxLength(100)
                .IsUnicode(false);
        });

        modelBuilder.Entity<Usuario>(entity =>
        {
            entity.HasKey(e => e.Id).HasName("PK__Usuario__3214EC27A5C9C1B3");

            entity.ToTable("Usuario");

            entity.Property(e => e.Id).HasColumnName("ID");
            entity.Property(e => e.DataNascimento).HasColumnType("date");
            entity.Property(e => e.Foto).HasColumnType("image");
            entity.Property(e => e.Nome)
                .IsRequired()
                .HasMaxLength(120)
                .IsUnicode(false);
        });

        OnModelCreatingPartial(modelBuilder);
    }

```

```
    partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
}
```

Note que a classe TestentityContext gerá todo código que poderia ser feito a mão. Abaixo um exemplo de como usar a estrutura criada:

Program.cs

```
using System;
using System.Linq;
using System.Threading.Tasks;

using entityTest.Model;

TestentityContext context = new TestentityContext();

await createUser("Don");
await createUser("Marcão");
await createUser("Queila Lima");
await createUser("Pamella");

await createProduct("Bico", "O máximo do avanço tecnológico em mecânica.");
await createProduct("Pudim", "O máximo do avanço tecnológico em gastronomia.");

var users =
    from user in context.Usuarios
    where user.Nome == "Queila Lima"
    select user;

var queila = users.FirstOrDefault();

var produtos =
    from product in context.Produtos
    where product.Nome == "Pudim"
    select product;

var pudim = produtos.FirstOrDefault();

await addOfertum(pudim, queila, 10);

```

```
33  printData();
34
35  async Task addOfertum(Produto produto, Usuario usuario, decimal value)
36  {
37      Ofertum oferta = new Ofertum();
38      oferta.Usuario = usuario.Id;
39      oferta.Produto = produto.Id;
40      oferta.Preco = value;
41
42      context.Oferta.Add(oferta);
43      await context.SaveChangesAsync();
44
45  }
46
47  async Task createProduct(string nome, string desc)
48  {
49      Produto produto = new Produto();
50      produto.Nome = nome;
51      produto.Descricao = desc;
52
53      context.Produtos.Add(produto);
54      await context.SaveChangesAsync();
55  }
56
57  void printData()
58  {
59      var query =
60          from user in context.Usuarios
61          join ofer in context.Oferta
62          on user.Id equals ofer.Usuario
63          select new {
64              nome = user.Nome,
65              produto = ofer.Produto,
66              valor = ofer.Preco
67          } into x
68          join prod in context.Produtos
69          on x.produto equals prod.Id
70          select new {
71              nome = x.nome,
```

```
            produto = prod.Nome,
            valor = x.valor
        };
    foreach (var y in query)
        Console.WriteLine($"{y.nome} está vendendo um {y.produto} por {y.valor} R$.");
}

async Task createUser(string nome)
{
    Usuario usuario = new Usuario();
    usuario.Nome = nome;
    usuario.DataNascimento = DateTime.Now;
    usuario.Foto = null;

    context.Usuarios.Add(usuario);
    await context.SaveChangesAsync();
}
```

Um fato interessante é, que assim como na nossa biblioteca, métodos LINQ são convertidos para SQL e executados com máximo desempenho no banco de dados sem precisar buscar grandes quantidades de dados para o Front-End. Além disso, queries usando select, where, join e etc, como apresentados neste exemplo são convertidos para LINQ e, consequentemente, reconvertidos para SQL. Desta forma, escrever "from user in context.Usuarios select user" é o mesmo que escrever no SQL "select * from Usuarios".

6 Exercícios

Adicione uma tabela a escolha com algum relacionamento, gere o modelo novamente, e faça uma inserção e consulta ao dados. Tente modificar os dados e veja o que acontece.