PERSISTINDO DADOS COM ENTITY FRAMEWORK CORE

**BD USADO: SQL SERVER**

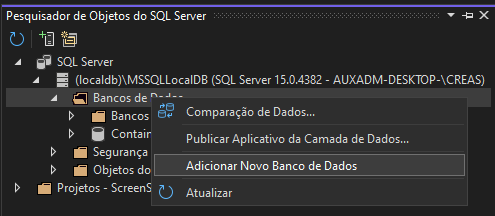
Considerando o projeto ScreenSound já funcional: <https://github.com/alura-cursos/3506-csharpWeb-screensound-curso1>



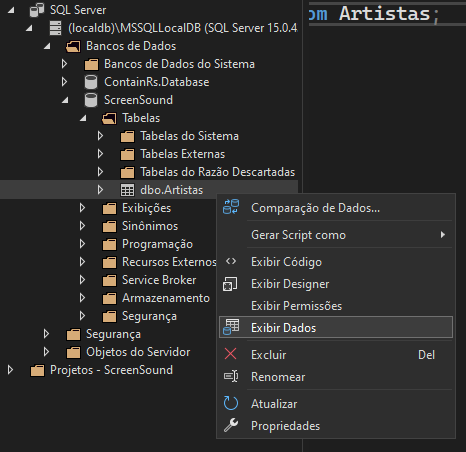
versão 7.0.14

**CRIAR UMA TABELA ARTISTAS USANDO SQLSERVER**

Exibir -> Pesquisador de objetos do SQLServer



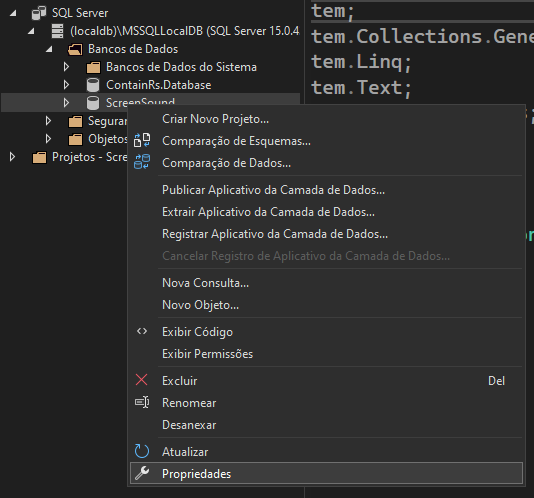
**EXIBIR E ADICIONAR DADOS DE UMA TABELA ATRAVÉS DO PESQUISADOR DE OBJETOS DO SQL SERVER**



**CONECTAR PROGRAMA COM BASE DE DADOS USANDO SqlConnection**

Baixar biblioteca Microsoft.Data.SqlClient

Criar uma pasta Banco com classe Connection, na qual consta uma variável string connectionString, que tem o caminho para o bando de dados criado anteriormente (esse caminho pode ser consultado nas propriedades, no campo “**cadeia de conexão**”) dentre outras propriedades.



E um método chamado ObterConexão, cujo tipo de retorno é SqlConnection

No program

try

{

using var connection = new Connection().ObterConexao();

connection.Open();

Console.WriteLine(connection.State);

}

catch(Exception e)

{

Console.WriteLine(e.Message);

}

return;

**LER DADOS DE UMA TABELA NO BD E MAPEAR PARA OS CAMPOS DE UM OBJETO**

Isso foi feito usado **SqlCommand e SqlDataReader, classes do ADO.NET**, dentro de um método Listar, na classe Connection,

public IEnumerable<Artista> Listar()

{

var lista = new List<Artista>();

using var connection = ObterConexao();

connection.Open();

string sql = "SELECT \* FROM Artistas";

SqlCommand command = new SqlCommand(sql, connection);

using SqlDataReader dataReader = command.ExecuteReader();

while (dataReader.Read())//ENQUANTO HOUVER LINHAS NA TABELA DO BD

{

//["Nome"] - CAMPO NO BD

string nomeArtista = Convert.ToString(dataReader["Nome"]);

string bioArtista = Convert.ToString(dataReader["Bio"]);

int idArtista = Convert.ToInt32(dataReader["Id"]);

Artista artista = new Artista(nomeArtista, bioArtista)

{

Id = idArtista

};

lista.Add(artista);

}

return lista;

}

No program

try

{

var connection = new Connection();

var listaArtistas = connection.Listar();

foreach(var artista in listaArtistas)

{

Console.WriteLine(artista);

}

}

catch(Exception e)

{

Console.WriteLine(e.Message);

}

**ADICIONAR DADOS NUMA TABELA EXISTENTE A PARTIR DE UM OBJETO**

Primeiramente, para organizar o código, foi usado o conceito de **DAL - Data Acess Layer**, que determina que deva existir uma classe que realiza todas as operação de uma data entidade do BD, ou seja, uma classe ArtistaDAL que guarda todas as operações relacionadas à artista.

Banco->ArtistaDal

public IEnumerable<Artista> Listar()

{

var lista = new List<Artista>();

using var connection = new Connection().ObterConexao();

connection.Open();

string sql = "SELECT \* FROM Artistas";

SqlCommand command = new SqlCommand(sql, connection);

using SqlDataReader dataReader = command.ExecuteReader();

while (dataReader.Read())//ENQUANTO HOUVER LINHAS NA TABELA DO BD

{

//["Nome"] - CAMPO NO BD

string nomeArtista = Convert.ToString(dataReader["Nome"]);

string bioArtista = Convert.ToString(dataReader["Bio"]);

int idArtista = Convert.ToInt32(dataReader["Id"]);

Artista artista = new Artista(nomeArtista, bioArtista)

{

Id = idArtista

};

lista.Add(artista);

}

return lista;

}

public void Adicionar(Artista artista)

{

//CONEXÃO COM BD

using var connection = new Connection().ObterConexao();

connection.Open();

string sql = "INSERT INTO Artistas " +

"(Nome, FotoPerfil, Bio) VALUES" +

"(@nome, @perfilPadrao, @bio)";

SqlCommand command = new SqlCommand(sql, connection);

command.Parameters.AddWithValue("@nome", artista.Nome);

command.Parameters.AddWithValue("@perfilPadrao", artista.FotoPerfil);

command.Parameters.AddWithValue("@bio", artista.Bio);

//EXECUÇÃO DO REGISTRO NA BD

//CUJO RETORNO É UM INTEIRO QUE DIZ QUANTAS LINHAS FORAM REGISTRADAS

int retorno = command.ExecuteNonQuery();

Console.WriteLine($"Registros adicionados: {retorno}");

}

**UTILIZANDO ENTITY FRAMEWORK NO ADO.NET**

Entity é um ORM - Object-Relational Mapping

Instalar no projeto o pacote



Classe Connection renomeada e com um novo método

internal class ScreenSoundContext : DbContext

{

private string connectionString = "Data Source=(localdb)\\" +

"MSSQLLocalDB;Initial Catalog=ScreenSound;" +

"Integrated Security=True;Encrypt=False;" +

"Trust Server Certificate=False;Application Intent=ReadWrite;" +

"Multi Subnet Failover=False";

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer(connectionString);

}

}

E as devidas mudanças em Program e nas classes de menu.

Feito também a adição da tabela Musicas no BD, e criado também seu DAL, nota-se que os arquivos musicasDAL e artistasDAL são bem parecidos, permitindo nesse caso o uso do Generics, criando assim uma classe abstrata DAL<T> que com o uso de Generics determina a assinatura de seus métodos abstratos, para que sejam implementados nas classes ArtistaDAL e MusicaDAL

Banco-> DAL.cs

internal abstract class DAL<T>

{

public abstract IEnumerable<T> Listar();

public abstract void Adicionar(T objeto);

public abstract void Atualizar(T objeto);

public abstract void Deletar(T objeto);

}

Banco-> ArtistaDAL.cs

internal class ArtistaDAL : DAL<Artista>

.

.

public override IEnumerable<Artista> Listar()

{

.

.

Banco-> MusicaDAL.cs

internal class MusicaDAL : DAL<Musica>

.

.

public override IEnumerable<Musica> Listar()

{

.

.

Mesmo considerando que já houve bastante reaproveitamento de código, a implementação de cada método em ArtistaDAL e MusicaDAL ainda tem bastante repetição. Essas repetições podem ser repassadas pra classe DAL com o uso de Generics

Considerando alguns detalhes importantes:

Para que cada método de DAL seja capaz de identificar com qual tabela ele irá trabalhar nas classes ArtistaDAL e MusicasDAL, na sua assinatura temos que expliciar que o tipo genérico T usado é uma classe

internal abstract class DAL<T> where T : class

E em cada método, no momento de usar uma classe, utilizar o método Set

No método Listar, por exemplo

public IEnumerable<T> Listar()

{

return context.Set<T>().ToList();

}

**return context.Set<T>().ToList();**

Quando usado pela classe ArtistaDAL, que implementa a classe abstrata DAL, esse comando se torna

return context.Artistas.ToList();

Considerando que das classes ArtistaDAL e MusicaDAL, resta somente o método RecuperarPorNome, podemos levá-lo para a classe DAL

public T? RecuperarPor(Func<T, bool> condicao)

{

return context.Set<T>().FirstOrDefault(condicao);

}

O método é estruturado dessa forma, usando Func nos seus parâmetros para que ele possa receber uma função lambda no seu uso, ou seja, seu parâmetro será uma condição cujo valor vai ser definido por uma função lambda.

Um exemplo de seu uso vai ser na classe MenuRegistrarMusica

var artistaRecuperado = artistaDAL.RecuperarPor(

artista =>artista.Nome.Equals(nomeDoArtista));

A variável artistaRecuperado assume o valor do objeto cujo campo Artista.Nome seja igual ao valor passado em nomeDoArtista.