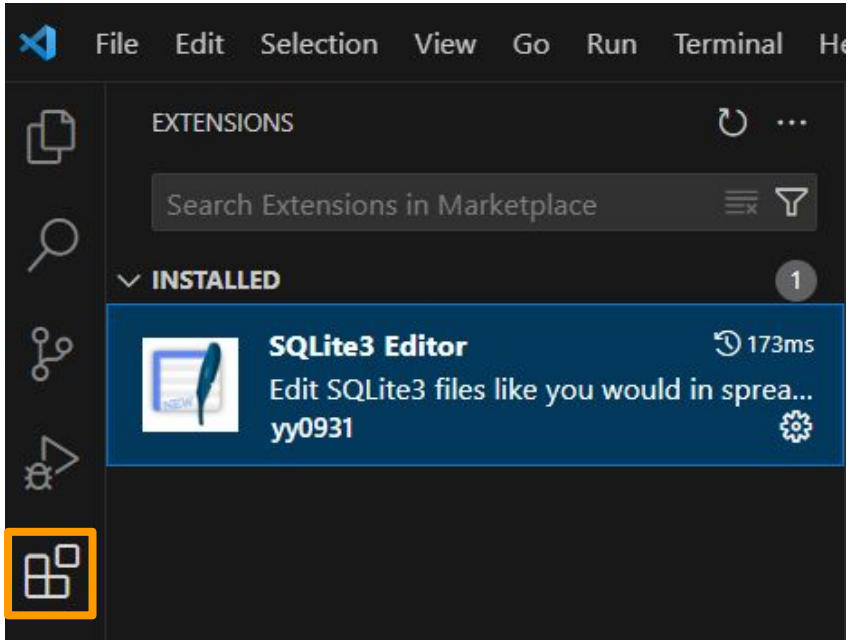
Programação Web Avançada

Introdução à Persistência com Dapper

Antes de iniciar

- Nesta disciplina vamos utilizar a extensão SQLite3 Editor.



Antes de iniciar

- Precisamos fazer uma pequena introdução ao ADO.net

ADO.Net

Introdução à interação com fontes de dados

ADO.Net - Introdução

- ADO.Net é um conjunto de bibliotecas que permitem a interação com fontes de dados.

- Fontes de dados podem ser:
  - Bases de dados;
  - Arquivos XML;
  - Arquivos texto;
  - Etc.

ADO.Net - Introdução

- Existem provedores de dados ADO.Net para diversas bases de dados, tais como:
  - SQL Server;
  - Oracle;
  - DB2;
  - Postgree;
  - MySQL;
- Neste curso nós utilizaremos ADO.Net para interagir com uma base de dados SQLite.

ADO.Net - Introdução

- A maioria dos SGBD's existentes possuem provedores de dados (data providers) nativos para ADO.Net.
  - Dessa forma, para acessar uma base de dados MySQL, por exemplo, ao invés de utilizarmos ODBC ou OLEDB, nós podemos utilizar um data provider nativo para MySQL.
    - Ao codificar um data provider para uma determinada base de dados, a equipe de desenvolvimento pode explorar as especificidades de cada base de forma a obter um melhor desempenho.

ADO.Net - Introdução

- Para acessarmos uma base SQLite, no terminal do VSCode de seu projeto, instale o pacote Microsoft.Data.Sqlite:

```
dotnet add package Microsoft.Data.Sqlite
```

ADO.Net - SqlConnection

- Para se conectar a uma base de dados SQLite, nosso sistema utilizará um objeto SqliteConnection.

- É possível passar para o construtor do objeto SqliteConnection a string de conexão que contém dados como o caminho do servidor, o nome da base de dados, o usuário e a senha de acesso.

ADO.Net - SqlConnection

- Principais métodos de um objeto SqliteConnection:
  - O método *Open* do objeto SqliteConnection abre a conexão com a base de dados.
  - O método *Close* fecha a conexão aberta.

ADO.Net - SqlConnection

```
using Microsoft.Data.Sqlite;

using (var conexao = new SqliteC
{
    conexao.Open();


}
```

Para utilizar o provider para SQLite, é necessário importar a biblioteca Microsoft.Data.Sqlite

ADO.Net - SqlConnection

```
using Microsoft.Data.Sqlite;

using (var conexao = new SqliteConnection(@"Data Source=db\app.db"))
{
    conexao.Open();


}
```

Instanciado o objeto *conexao*, que possibilitará fazermos a conexão com o SQLite.

ADO.Net - SqlConnection

```
using Microsoft.Data.Sqlite;

using (var conexao = new SqliteConnection(@"Data Source=db\app.db" )
{
    conexao.Open();


}
```

No construtor foi passado o caminho para o arquivo da base de dados. Em outras bases de dados, costumamos passar a string de conexão aqui.

ADO.Net - SqlConnection

```
using Microsoft.Data.Sqlite;

using (var conexao = new SqliteConnection(@"Data Source=db\app.db"))
{
    conexao.Open();


}
```

Abre a conexão com a base de dados.

ADO.Net - SqlConnection

```
using Microsoft.Data.Sqlite;

using (var conexao = new SqliteConnection(@"Data Source=db\app.db"))
{
    conexao.Open();


}
```

O método Close não precisou ser chamado por termos instanciado o objeto SqliteConnection dentro da diretiva using, de forma que ao término do bloco de código do using o disposing do objeto é feito, o que fecha a conexão automaticamente.

ADO.Net - SqlCommand

- Um comando em ADO.Net é representado por um objeto.
- No SQLite, a classe para instanciação de nossos objetos que representam comandos é chamada de SqliteCommand.
- Um objeto SqliteCommand permite especificar que tipo de interação será feita em uma base de dados.

ADO.Net - SqlCommand

- Para executar um comando que altera a base de dados, basta utilizar o método ExecuteNonQuery() do objeto SqliteCommand, conforme o exemplo do próximo slide.

ADO.Net - SqlCommand

```
using Microsoft.Data.Sqlite;

using (var conexao = new SqliteConnection(@"Data Source=db\app.db"))
{
    conexao.Open();

    var cmd = conexao.CreateCommand();
    cmd.CommandText = "insert into contato (id,nome,email)
        values (@id,@nome,@email)";
    cmd.Parameters.AddWithValue("@id", 4);
    cmd.Parameters.AddWithValue("@nome", "João");
    cmd.Parameters.AddWithValue("@email", "joao@email.com");

    var retorno = cmd.ExecuteNonQuery();

    Console.WriteLine($"Comando executado. {retorno} linhas afetadas.");
}
```

ADO.Net - SqlCommand

```
using Microsoft.Data.Sqlite;

using (var conexao = new SqliteConnection(@"Data Source=db\app.db"))
{
    conexao.Open();

    var cmd = conexao.CreateCommand();
    cmd.CommandText = "insert into contato (id,nome,email)
        values (@id,
    cmd.Parameters.A
    cmd.Parameters.A
    cmd.Parameters.A

    var retorno = cmd.ExecuteNonQuery();

    Console.WriteLine($"Comando executado. {retorno} linhas afetadas.");
}
```

Retorna um objeto SqlCommand, que permite a execução de comandos na base de dados.

ADO.Net - SqlCommand

```
using Microsoft.Data.Sqlite;

using (var conexao = new SqliteConnection(@"Data Source=db\app.db"))
{
    conexao.Open();

    var cmd = conexao.CreateCommand();
    cmd.CommandText = "insert into contato (id,nome,email)
        values (@id,@nome,@email)";
    cmd.Parameters.AddWithValue("@id", 4);
    cmd.Parameters.AddW
    cmd.Parameters.AddW

    var retorno = cmd.E

    Console.WriteLine($"Comando executado. {retorno} linhas afetadas.");
}
```

A propriedade *CommandText* deve ser atribuída com o comando a ser executado na base de dados.

ADO.Net - SqlCommand

```
using Microsoft.Data.Sqlite;

using (var conexao = new SqliteConnection(@"Data Source=db\app.db"))
{
    conexao.Open();

    var cmd = conexao.CreateCommand();
    cmd.CommandText = "insert into contato (id,nome,email)
        values (@id,@nome,@email)";
    cmd.Parameters.AddWithValue("@id", 4);
    cmd.Parameters.AddWithValue("@nome", "João");
    cmd.Parameters.AddWithValue("@email", "joao@email.com");

    var re

    Consol                                              linhas afetadas.");
}
```

É possível adicionar parâmetros nos comandos, de forma a evitar *Sql Injection*.

ADO.Net - SqlCommand

```
using Microsoft.Data.Sqlite;

using (var conexao = new SqliteConnection(@"Data Source=db\app.db"))
{
    conexao.Open();

    var cmd = conexao.CreateCommand();
    cmd.CommandText = "insert into contato (id,nome,email)
        values (@id,@
    cmd.Parameters.Ad
    cmd.Parameters.Ad
    cmd.Parameters.Ad

    var retorno = cmd.ExecuteNonQuery();

    Console.WriteLine($"Comando executado. {retorno} linhas afetadas.");
}
```

O comando ExecuteNonQuery executa o comando e retorna o número de linhas afetadas.

ADO.Net - DataReader

- Para ler registros, podemos utilizar um DataReader, um DataTable ou um DataSet.
  - Para leitura rápida com cursor, prefira o DataReader.
  - Para carregar os registros em memória, possibilitando fechar a conexão e manter os objetos em memória, prefira o DataTable ou o DataSet.

ADO.Net - DataReader

```
using Microsoft.Data.Sqlite;

using (var conexao = new SqliteConnection(@"Data Source=db\app.db"))
{
    conexao.Open();

    var cmd = conexao.CreateCommand();
    cmd.CommandText = "SELECT * FROM contato WHERE nome like @parteNome";
    cmd.Parameters.AddWithValue("@parteNome", $"%a%");

    using (var dr = cmd.ExecuteReader())
    {
        while (dr.Read())
        {
            Console.WriteLine($"{dr["id"]} - {dr["nome"]} - {dr["email"]}");
        }
    }
}
```

ADO.Net - DataReader

```
using Microsoft.Data.Sqlite;

using (var conexao =                                          .db"))
{
    conexao.Open();

    var cmd = conexa
    cmd.CommandText                                           parteNome";
    cmd.Parameters.A

    using (var dr = cmd.ExecuteReader())
    {
        while (dr.Read())
        {
            Console.WriteLine($"{dr["id"]} - {dr["nome"]} - {dr["email"]}");
        }
    }
}
```

O comando ExecuteReader retorna um DataReader, que é um cursor para registros retornados. É necessário fechar o DataReader, porém se o mesmo for declarado dentro de um using, o mesmo será fechado automaticamente ao término da execução do bloco using.

ADO.Net - DataReader

```
using Microsoft.Data.Sqlite;

using (var conexao = new SqliteConnection(@"Data Source=db\app.db"))
{
    con

    var
    cmd                                                    E nome like @parteNome";
    cmd                                                    a%");

    usi
    {
        while (dr.Read())
        {
            Console.WriteLine($"{dr["id"]} - {dr["nome"]} - {dr["email"]}");
        }
    }
}
```

A função Read lê o registro atual e retorna *true* se o mesmo existir ou *false* caso já tenha atingido o final dos registros retornados.

ADO.Net - DataReader

```
using Microsoft.Data.Sqlite;

using (var conexao = new SqliteConnection(@"Data Source=db\app.db"))
{
    conexao.Open();

    var cmd = conexao.CreateCommand();
    cmd.CommandText = "SE                                    eNome";
    cmd.Parameters.AddWit

    using (var dr = cmd.E
    {
        while (dr.Read())
        {
            Console.WriteLine($"{dr["id"]} - {dr["nome"]} - {dr["email"]}");
        }
    }
}
```

Para ler um campo de um registro, utilize o índice do mesmo entre colchetes. Também é possível utilizar o nome do campo entre colchetes.

Dapper

Introdução ao Micro-ORM Dapper

Introdução ao Dapper

- O Dapper é um mapeador objeto relacional simples para DotNet.
  - Foi desenvolvido pela equipe de desenvolvedores do Stack Overflow para solucionar gargalos de desempenho que os mesmos estavam tendo com o Entity Framework.

Introdução ao Dapper

- Dapper é uma biblioteca NuGet que você pode adicionar ao seu projeto.
  - Fornece métodos de extensão (*extension methods*) em sua instância *DbConnection*. Isso fornece uma API simples e eficiente para invocar SQL, com suporte para acesso a dados síncronos e assíncronos, além de permitir consultas com e sem buffer.

Dapper

- Instalando o Dapper:
  - No terminal, digite:

```
dotnet add package Dapper
```

Inserção

```
using Dapper;
using Microsoft.Data.Sqlite;

using(var conexao = new SqliteConnection("Data Source=db/dados.db"))
{
    conexao.Open();

    var obj = new Contato { Id = "5", Nome = "José", Email = "jose@email.com"
};

    const string sql =  "INSERT INTO contato" +
                        " (id, nome, email)" +
                        " VALUES (@Id, @Nome, @Email)";

    conexao.Execute(sql, obj);
}
```

Alteração

```
using Dapper;
using Microsoft.Data.Sqlite;

using(var conexao = new SqliteConnection("Data Source=db/dados.db"))
{
    conexao.Open();

    var obj = new Contato { Id = "5", Nome = "Zé", Email = "jose@email.com" };

    const string sql = "UPDATE contato" +
            " SET nome = @Nome, email = @Email" +
            " WHERE id = @Id";

    conexao.Execute(sql, obj);
}
```

Exclusão

```
using Dapper;
using Microsoft.Data.Sqlite;

using(var conexao = new SqliteConnection("Data Source=db/dados.db"))
{
    conexao.Open();

    var obj = new Contato { Id = "5", Nome = "José", Email = "jose@email.com"
};

    const string sql = "DELETE FROM contato WHERE id = @Id";

    conexao.Execute(sql, new { Id = "3" });
}
```

Selecionar Um

```
using Dapper;
using Microsoft.Data.Sqlite;

using(var conexao = new SqliteConnection("Data Source=db/dados.db"))
{
    conexao.Open();

    const string sql = "SELECT * FROM contato WHERE id = @Id";
    var obj = conexao.QuerySingle<Contato>(sql, new { Id = 1 });

    Console.WriteLine($"{obj.Nome} - {obj.Email}");
}
```

Selecionar Vários

```
using Dapper;
using Microsoft.Data.Sqlite;

using(var conexao = new SqliteConnection("Data Source=db/dados.db"))
{
    conexao.Open();

    const string sql = "SELECT * FROM contato" +
        " WHERE nome LIKE @ParteNome" +
        " ORDER BY nome";

    var objetos = conexao.Query<Contato>(sql, new { ParteNome = "%n%"});

    foreach (var obj in objetos)
        Console.WriteLine($"{obj.Nome} - {obj.Email}");
}
```

Resumindo o uso do Dapper

```
// insert/update/delete etc
var count  = connection.Execute(sql [, args]);

// consulta com múltiplas linhas
IEnumerable<T> rows = connection.Query<T>(sql [, args]);

// consulta de única linha ({Single|First}[OrDefault])
T row = connection.QuerySingle<T>(sql [, args]);
```

Fim de material

Dúvidas?

Você também pode me contactar:

Telegram: @camillofalcao

camillofalcao@gmail.com