Aula 23

Prof: Henrique Augusto Maltauro

Implementar Banco de Dados Para WEB

Integração C#

Integração C#

Até aqui, vimos todo o processo de criação e manipulação de **banco de dados** com o **SQL**.

Agora, vamos partir para integrar o nosso **banco de dados** em uma **aplicação de back-end** com **C#**.

Em um primeiro momento, vamos ver como trabalhar essa integração da forma mais rústica possível.

Integração C#

O **framework .NET** já possui nativamente duas bibliotecas que darão todas as ferramentas necessárias para realizarmos essa integração.

São elas:

- ➔ **System.Data**
- ➔ **System.Data.SqlClient**

Ambas bibliotecas precisam ser importadas com o **using** quando formos trabalhar nessa integração.

Integração C#

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

namespace nomeDoNamespace
{
    public class NomeDaClasse
    {
        // Bloco de código da classe
    }
}
```

Integração C#

E então, a partir dessas bibliotecas, nós vamos ter algumas **classes**, **tipos de dados** e **métodos** que iremos utilizar para integrar o **banco de dados** com o **C#**.

Integração C#

- **String de Conexão**

Primeiramente teremos a **string de conexão**.

Ela vai ser uma **string** que será utilizada para definir todas as **configurações de conexão com o banco de dados**, como por exemplo:

➔ **Local do Servidor**
➔ **Qual é o Banco Dados**
➔ **Usuário e Senha**
➔ **Configurações de Segurança**
➔ **Etc...**

Integração C#

- **String de Conexão**

É importante saber que, cada **SGBD** terá um padrão de **string de conexão** diferente, é no site da cada **SGBD** vai ter as informações desse padrão de **string de conexão**.

Integração C#

- String de Conexão

```
string connectionString =
    “Server=SERVIDOR;Database=BANCO_DE_DADOS;”;
```

Integração C#

- **SqlConnection**

Depois de definido a **string de conexão** temos a **classe** e **tipo de dado SqlConnection**, que irá receber a **string de conexão** no seu **construtor** e irá **criar uma conexão com o banco de dados**.

É a partir dessa conexão que o **C#** irá realizar todos os processos no **banco de dados**.

Integração C#

- SqlConnection

```
string connectionString = “...”;
SqlConnection con = new SqlConnection(connectionString);
```

Integração C#

- **SqlConnection: Open**

Depois de criado a conexão com o **banco de dados**, é necessário abrir essa conexão.

E isso é feito com o **método Open** que parte do **objeto** de conexão criado.

Integração C#

- SqlConnection: Open

```
string connectionString = “...”;
SqlConnection con = new SqlConnection(connectionString);
con.Open();
```

Integração C#

- **SqlConnection: Close**

Depois de realizado todo o processo desejado naquela conexão de **banco de dados**, é necessário fechar essa conexão.

E isso é feito com o **método Close** que parte do **objeto** de conexão criado.

É muito importante fechar a conexão no final do processo para não lotar o SGBD com conexões inutilizadas e diminuir a chance de conflitos de processos.

Integração C#

- SqlConnection: Close

```
string connectionString = “...”;
SqlConnection con = new SqlConnection(connectionString);
con.Open();
// Código do processo realizado
con.Close();
```

Integração C#

- **SqlCommand**

Depois de criado a conexão com o **banco de dados**, devemos utilizar a **classe** e **tipo de dado SqlCommand**, que irá executar a nossa instrução **SQL**, utilizando a conexão criada com o **SqlConnection**.

O **construtor** do **SqlCommand**, recebe dois **parâmetros**.

Integração C#

- **SqlCommand**

O primeiro **parâmetro** é uma **string**, que possui a instrução **SQL** que será executada. Pode ser **CREATE**, **ALTER**, **INSERT**, **UPDATE**, **DELETE**, **SELECT**, etc, desde que seja uma **instrução SQL válida, ele irá executá-la**.

O segundo **parâmetro** é um **objeto** do tipo **SqlConnection**, para que o **SqlCommand** saiba aonde ele deve executar a instrução **SQL**.

Integração C#

- SqlCommand

```
string connectionString = "...";
SqlConnection con = new SqlConnection(connectionString);
con.Open();
SqlCommand cmd = new SqlCommand("...", con);
con.Close();
```

Integração C#

- **SqlCommand**

A partir desse **objeto** do tipo **SqlCommand**, temos dois principais **métodos** para executar a instrução **SQL**:

➔ **ExecuteNonQuery**

➔ **ExecuteReader**

Integração C#

- **SqlCommand: ExecuteNonQuery**

O **método ExecuteNonQuery** irá executar uma instrução **SQL**, porém não terá um retorno de registros, como em uma consulta **S**QL, ele **apenas irá executar a instrução SQL**.

Útil para executar comandos de **DML** e **DDL**, ou seja, **CREATE**, **ALTER**, **INSERT**, **UPDATE** e **DELETE**.

Integração C#

- SqlCommand: ExecuteNonQuery

```
string connectionString = "...";
SqlConnection con = new SqlConnection(connectionString);
con.Open();
SqlCommand cmd = new SqlCommand("INSERT", con);
cmd.ExecuteNonQuery();
con.Close();
```

Integração C#

- **SqlCommand: ExecuteReader**

O **método ExecuteReader** irá executar uma instrução **SQL**, e irá retornar um **objeto** do tipo **SqlDataReader**.

Útil para executar comandos de **DQL**, ou seja, **SELECT**.

Integração C#

- SqlCommand: ExecuteReader

```
string connectionString = "...";
SqlConnection con = new SqlConnection(connectionString);
con.Open();
SqlCommand cmd = new SqlCommand("SELECT", con);
SqlDataReader reader = cmd.ExecuteReader();
con.Close();
```

Integração C#

- **SqlDataReader**

O **SqlDataReader** é uma **classe** e **tipo de d**ado, que **armazena os resultados de uma consulta**.

Através do **objeto** obtido do **método SqlCommand.ExecuteReader**, nós conseguimos ter acesso aos dados que uma instrução **SELECT** retorna.

Integração C#

- **SqlDataReader: Read**

Fazendo uso do **método Read**, o C# irá efetuar a **leitura da próxima linha de registros** armazenada em um **objeto** do tipo **SqlDataReader**.

O **método Read** retorna um **booleano**, indicando se ele conseguiu ler a próxima linha de registros. No caso de não existir uma próxima linha de registros, o **método** retorna um **false**.

Esse retorno torna o **método Read** muito útil para uso em um **WHILE**.

Integração C#

- SqlDataReader: Read

```
SqlCommand cmd = new SqlCommand("SELECT", con);
SqlDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
    // Bloco de código do while
}
```

Integração C#

- **SqlDataReader**

Durante a leitura dos registros que é feita pelo **método Read**, podemos acessar as informações das colunas da mesma forma que acessamos os elementos de um array, usando os **delimitadores de colchetes [ ]**.

Também podemos acessar através de métodos como **GetString** e **GetInt**.

Integração C#

- **SqlDataReader**

Com os **delimitadores de colchetes** e os **métodos**, podemos acessar a informação das colunas de duas formas diferentes:

➔ **Por Índice**

➔ **Por Nome**

Integração C#

- **SqlDataReader**

Por Índice: Dentro dos **colchetes** ou dentro dos **parâmetros dos métodos** é informado o **índice das colunas** da consulta, sempre começando em zero.

Integração C#

- SqlDataReader

```
SqlCommand cmd = new SqlCommand("SELECT
                                    ID,
                                    NOME
                                FROM PRODUTO", con);
SqlDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
    reader[0];
}
```

Integração C#

- SqlDataReader

```
SqlCommand cmd = new SqlCommand("SELECT
                                    ID,
                                    NOME
                                FROM PRODUTO", con);
SqlDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
    reader.GetString(1);
}
```

Integração C#

- **SqlDataReader**

Por Nome: Dentro dos **colchetes** ou dentro dos **parâmetros dos métodos** é informado uma string com o **nome das colunas** da consulta.

Aqui fica muito claro o quanto o **ALIAS** do **SELECT** se faz útil.

Integração C#

- SqlDataReader

```
SqlCommand cmd = new SqlCommand("SELECT
                                    ID,
                                    NOME AS NOME_PRODUTO
                                FROM PRODUTO", con);
SqlDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
    reader["NOME_PRODUTO"];
}
```

Integração C#

- SqlDataReader

```
SqlCommand cmd = new SqlCommand("SELECT
                                    ID,
                                    NOME AS NOME_PRODUTO
                                FROM PRODUTO", con);
SqlDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
    reader.GetString("NOME_PRODUTO");
}
```

Exercício

Exercício

gg.gg/SenacBD23

github.com/hmaltaurodev/slides