

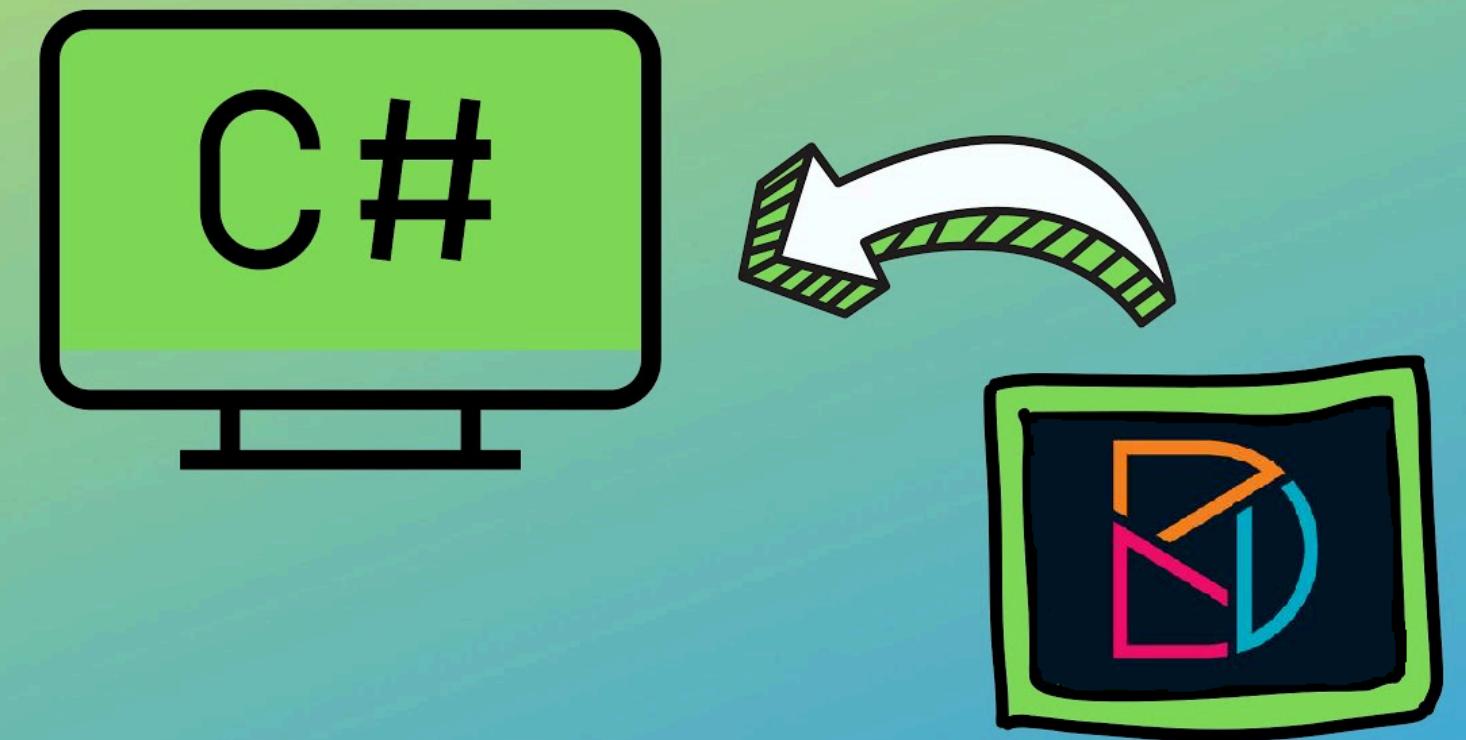


Dapper

Dapper .NET 8 e Suas Funcionalidades

-  Matheus Endlich Silveira
-  Lucas Bonato Soares
-  Guilherme Ferreira de Góes
-  Rafael Ferreira Bassul

O que é o Dapper e por que ele é relevante?



- 1 Mapeamento Objeto-Relacional**

O Dapper simplifica o mapeamento de objetos .NET para consultas SQL, reduzindo a quantidade de código necessário.
- 2 Alto Desempenho**

Com uma implementação otimizada, o Dapper oferece um desempenho superior em comparação a outras soluções ORM.
- 3 Flexibilidade**

O Dapper pode ser facilmente integrado a qualquer projeto .NET e trabalha bem com bancos de dados relacionais.

Principais recursos e funcionalidades do Dapper

Mapeamento Simples

Mapear objetos .NET para consultas SQL com mínima configuração.

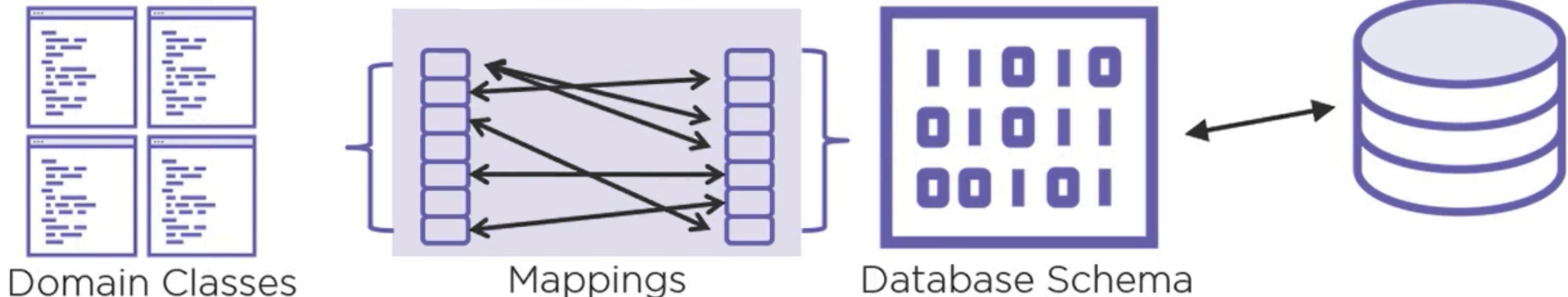
Consultas Personalizadas

Escrever consultas SQL personalizadas e obter objetos .NET diretamente.

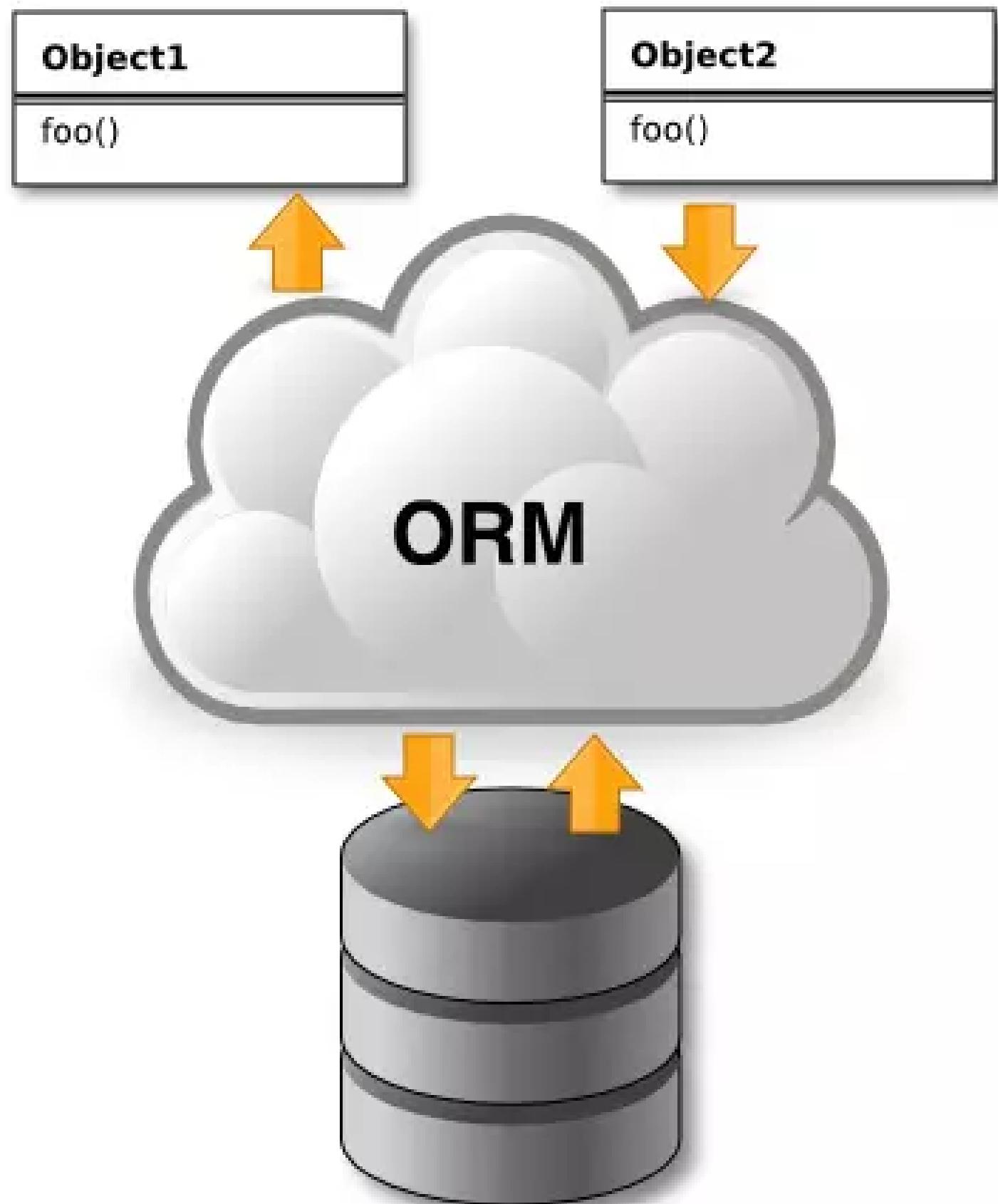
Tipos Dinâmicos

Suporte a mapeamento de tipos dinâmicos, como listas e dicionários.

Dapper Framework Core



Configuração e integração do Dapper em aplicações .NET



- 1 Instalação
Adicionar o pacote NuGet do Dapper ao projeto.
- 2 Configuração
Configurar a conexão com o banco de dados e injetar o provedor de dados.
- 3 Integração
Utilizar o Dapper para realizar consultas e manipulação de dados.

Consultas com Dapper



Consultas Simples

Recuperar dados com consultas SQL básicas, mapeando para objetos .NET.

Consultas Complexas

Executar consultas avançadas com junções, filtros e ordenação.

Parâmetros Dinâmicos

Passar parâmetros dinâmicos para as consultas, tratando-as para evitar problemas de segurança.

Retorno Customizado

Mapear os resultados para tipos personalizados, incluindo listas e dicionários.

<consulta_dapper>

```
string stringConexao = "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=Locadora;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;";

using (var conexao = new SqlConnection(stringConexao))
{
    conexao.Open();

    string QuerySelecionarDiretor = "SELECT [Id], [nome] FROM [dbo].[Diretor]";
    IEnumerable<Diretor> diretores = conexao.Query<Diretor>(QuerySelecionarDiretor);

    Console.WriteLine("Diretores:");
    foreach (var diretor in diretores)
    {
        Console.WriteLine($"Id: {diretor.Id}, Nome: {diretor.Nome}");
    }

    string QuerySelecionarFilme = @"
SELECT f.[Id], f.[nome], f.[duracao], f.[id_diretor]
FROM [dbo].[Filme] f
INNER JOIN [dbo].[Diretor] d ON f.[id_diretor] = d.[Id]";

    IEnumerable<Filme> filmes = conexao.Query<Filme>(QuerySelecionarFilme);

    Console.WriteLine("\nFilmes:");
    foreach (var filme in filmes)
    {
        Console.WriteLine($"Id: {filme.Id}, Nome: {filme.Nome}, Duração: {filme.Duracao}h, IdDiretor: {filme.Id_Diretor}");
    }
}
```

<Parametro_Dinamico>

```
string queryFilmesPorDiretor = @"
SELECT f.[Id], f.[nome], f.[duracao], f.[id_diretor]
FROM [dbo].[Filme] f
WHERE f.[id_diretor] = @IdDiretor;";
var parametros = new { IdDiretor = 1 }; // Parâmetro dinâmico
IEnumerable<Filme> filmes = conexao.Query<Filme>(queryFilmesPorDiretor, parametros);
```

Mapeamento de objetos e tipos personalizados



Mapeamento de Objetos

Mapear objetos .NET para tabelas e colunas do banco de dados.



Tipos Personalizados

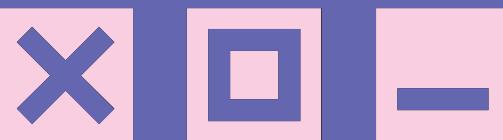
Suportar mapeamento para tipos de dados personalizados



Flexibilidade

Adaptar o mapeamento às necessidades específicas da sua aplicação.

<consulta + mapeamento>



```
using (var conexao = new SqlConnection(StringConexao))
{
    conexao.Open();

    string query = @"
SELECT
    f.[Id] AS Id,
    f.[nome] AS Nome,
    f.[duracao] AS Duracao,
    f.[id_diretor] AS IdDiretor,
    d.[Id] AS Id,
    d.[nome] AS Nome
FROM [dbo].[Filme] f
LEFT JOIN [dbo].[Diretor] d ON f.[id_diretor] = d.[Id]";

    var filmes = conexao.Query<Filme, Diretor, Filme>(
        query,
        (filme, diretor) =>
    {
        if (diretor != null && diretor.Id > 0)
        {
            filme.Diretor = diretor;
        }
        return filme;
    },
        splitOn: "Id"
    );

    foreach (var filme in filmes)
    {
        Console.WriteLine($"Filme: {filme.Nome}, Duração: {filme.Duracao}, Diretor: {filme.Diretor?.Nome ?? "Sem Diretor"}");
    }
}
```



Manipulação de dados: inserção, atualização e exclusão

1

Inserção

Inserir novos registros no banco de dados por meio do mapeamento, afetando objetos e entidades igualmente.

2

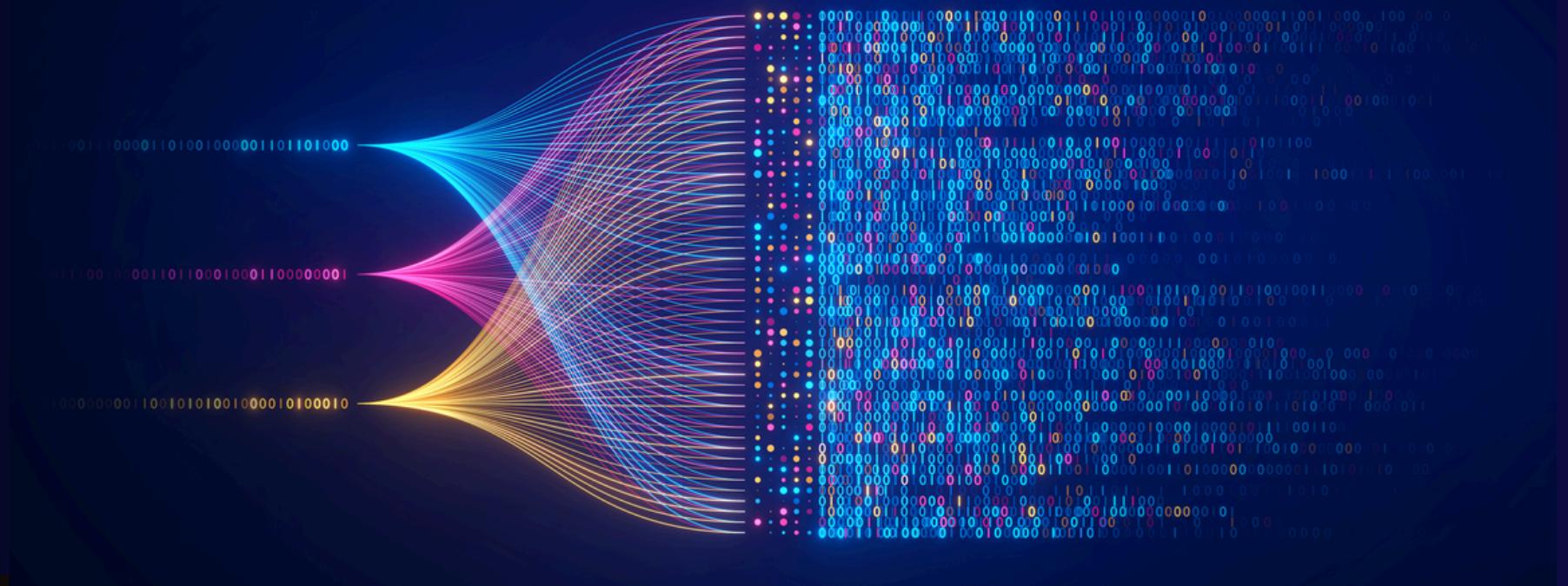
Atualização

Atualizar registros existentes com facilidade, sem a necessidade de escrever código complexo.

3

Exclusão

Remover registros do banco de dados de maneira rápida e segura sem criar grandes problemas com os objetos.



Transações e gerenciamento de conexões

Transações

O Dapper oferece suporte a transações, garantindo a integridade dos dados durante operações complexas.

Gerenciamento de Conexões

Facilita o gerenciamento de conexões com o banco de dados, evitando problemas comuns de desempenho e escalabilidade.

<inserção_de_dados>

	Id	nome	duracao	id_diretor
▶	1	Filme legal de P...	127,5	1
●	NULL	NULL	NULL	NULL

	Id	nome
▶	1	Vinícius Rosalen
●	NULL	NULL

```
string stringConexao = "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=FilmePOO;Integrated Security=True";

using (var conexao = new SqlConnection(stringConexao))
{
    conexao.Open();

    var diretor = new Diretor
    {
        Id = 1,
        Nome = "Vinicius Rosalen"
    };

    string QueryInserirDiretor = @"
    INSERT INTO [dbo].[Diretor] ([Id], [nome])
    VALUES (@Id, @Nome)";

    conexao.Execute(QueryInserirDiretor, diretor);

    var filme = new Filme
    {
        Id = 1,
        Nome = "Filme legal de POO",
        Duracao = 127.5,
        IdDiretor = 1
    };

    string QueryInserirFilme = @"
    INSERT INTO [dbo].[Filme] ([Id], [nome], [duracao], [id_diretor])
    VALUES (@Id, @Nome, @Duracao, @IdDiretor)";

    conexao.Execute(QueryInserirFilme, filme);
}
```

<atualização_de_dados>

	Id	nome	duracao	id_diretor
▶	1	Novo filme de POO	127,5	1
●	NULL	NULL	NULL	NULL

	Id	nome
▶	1	Vinicius Rosalen da Silva
●	NULL	NULL

```
string StringConexao = "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=MeuBanco;Integrated Security=True";

using (var conexao = new SqlConnection(StringConexao))
{
    conexao.Open();

    using (var transacao = conexao.BeginTransaction())
    {
        try
        {
            string QueryAtualizarFilme = @"
                UPDATE [dbo].[Filme]
                SET [nome] = @Nome
                WHERE [Id] = @Id";
            conexao.Execute(QueryAtualizarFilme,
                new { Nome = "Novo filme de POO", Id = 1 },
                transaction: transacao);

            string QueryAtualizarDiretor = @"
                UPDATE [dbo].[Diretor]
                SET [nome] = @Nome
                WHERE [Id] = @Id";
            conexao.Execute(QueryAtualizarDiretor,
                new { Nome = "Vinicius Rosalen da Silva", Id = 1 },
                transaction: transacao);

            transacao.Commit();
        }
        catch (Exception ex)
        {
            transacao.Rollback();
            Console.WriteLine($"Erro: {ex.Message}");
        }
    }
}
```

<remoção_de_dados>

	Id	nome	duracao	id_diretor
•	NULL	NULL	NULL	NULL

	Id	nome
•	NULL	NULL

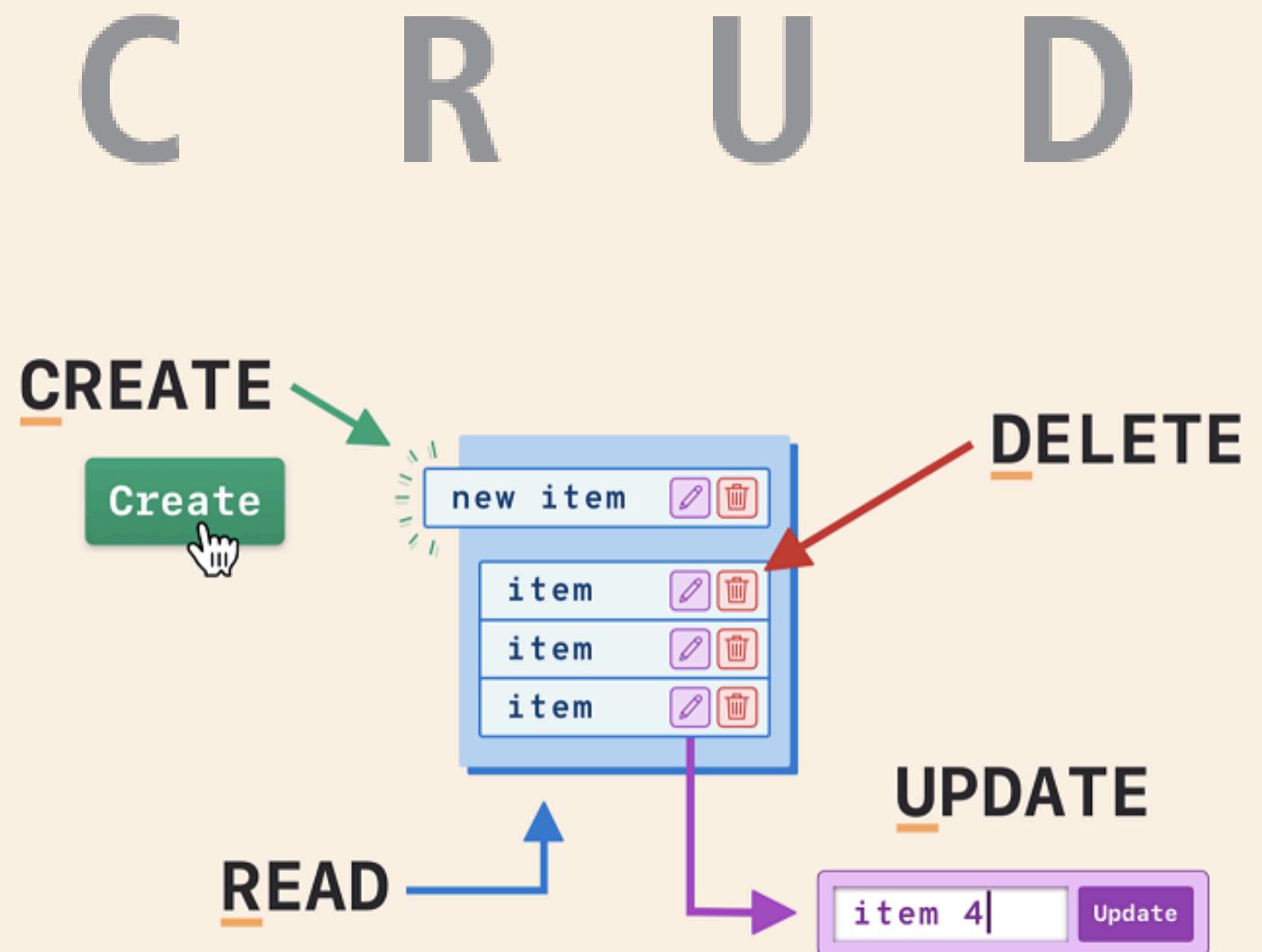
```
string StringConexao = "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=MeuBanco;Integrated Security=True";

using (var conexao = new SqlConnection(StringConexao))
{
    conexao.Open();

    using (var transacao = conexao.BeginTransaction())
    {
        try
        {
            string deleteFilmeQuery = @"
                DELETE FROM [dbo].[Filme]
                WHERE [nome] = @Nome AND [id_diretor] = @IdDiretor";
            conexao.Execute(deleteFilmeQuery,
                new { Nome = "Novo filme de P00", IdDiretor = 1 },
                transaction: transacao);

            string deleteDiretorQuery = @"
                DELETE FROM [dbo].[Diretor]
                WHERE [nome] = @Nome";
            conexao.Execute(deleteDiretorQuery,
                new { Nome = "Vinicius Rosalen da Silva" },
                transaction: transacao);

            transacao.Commit();
            Console.WriteLine("Dados removidos com sucesso!");
        }
        catch (Exception ex)
        {
            transacao.Rollback();
            Console.WriteLine($"Erro: {ex.Message}");
        }
    }
}
```



Exemplos práticos de aplicação do Dapper

1 CRUD em Aplicações Web

Integrar o Dapper em aplicações web para realizar operações CRUD de forma eficiente.

2 Relatórios e Dashboards

Utilizar o Dapper para extrair e transformar dados para exibição em relatórios e dashboards.

3 Processamento de Lotes

Aproveitar a eficiência do Dapper em aplicações de processamento em lote ou em background.

Melhores práticas e dicas para uso do Dapper

Otimização de Consultas

Escrever consultas SQL eficientes e evitar N+1 queries para melhorar o desempenho.

Injeção de Dependência

Integrar o Dapper com um contêiner de injeção de dependência para uma melhor organização do código.

Testes Automatizados

Criar testes unitários e de integração para garantir a confiabilidade do uso do Dapper.

Monitoramento e Logging

Implementar mecanismos de monitoramento e logging para acompanhar o uso do Dapper em produção.

