Fundamentos de Desenvolvimento Web/BD

Aula 07: EF Queries

Professor: Rinaldo Ferreira Junior

E-mail: rinaldo.fjunior@prof.infnet.edu.br

- **Professor:** Rinaldo Ferreira Junior
- **Graduação:** Pós-graduado em Arquitetura de Softwares
- **Atuação:** .Net | C# | SQL | NoSQL | Engenheiro de Software
- **E-mail:** rinaldo.fjunior@prof.infnet.edu.br
- **Linkedin:** https://www.linkedin.com/in/rinaldo-ferreira-junior-787326a

Aula 07: Roteiro

Instituto Infnet

- EF Queries

Aula 07: LINQ-to-Entities

- EF Core usa LINQ-to-Entities para executar consultas (queries) ao banco de dados. Usando LINQ, é possível trabalhar com C# de maneira próxima ao que seria escrito em SQL.
- EF Core passa uma representação do código LINQ para o provider de banco de dados, que por sua vez, fica responsável por "traduzir" o código C# para o SQL, e executá-lo no banco de dados.
- A classe DbSet<T> representa uma coleção de registros de uma determinada entidade. É através do DbSet<T> que as operações usando LINQ serão refletidas na base de dados.
- Pontos chave da classe DbSet<T>:
  - Suporte aos métodos para operações de CRUD
  - Suporte aos métodos de seleção e filtro
  - Suporte ao rastreio de alterações de dados

Aula 07: Queries

- Como o acesso ao banco se dá através do context Gerado pelo EF Core, é necessário carregar esse context antes de começar as execuções na base.

```
public class BookModel : PageModel
{
    private readonly LibraryContext _context;
    5 references
    public List<Book>? Books { get; set; }

    0 references
    public BookModel(LibraryContext context)
    {
        _context = context;
    }

    0 references
    public async Task OnGetAsync()
    {
        using (var context = new LibraryContext())
        {
            Books = await _context.Books.ToListAsync();
        }
    }
}
```

Aula 07: Queries

- Usando o LINQ, operações comuns de filtro podem ser realizadas diretamente no código C#.

```
public class BookWhere1Model : PageModel
{
    private readonly LibraryContext _context;
    5 references
    public List<Book>? Books { get; set; }

    0 references
    public BookWhere1Model(LibraryContext context)
    {
        _context = context;
    }

    0 references
    public async Task OnGetAsync()
    {
        Books = await _context
                    .Books
                    .Where(b => b.Title == "Drácula 1a. Edição")
                    .ToListAsync();
    }
}
```

Aula 07: Read-Only Queries

- Por padrão, o LINQ carrega dados adicionais em todas as queries, para manter controle sobre possíveis edições nos dados. É possível evitar essa sobrecarga, usando o método AsNoTracking().

```
public class BookWhere1Model : PageModel
{
    private readonly LibraryContext _context;
    5 references
    public List<Book>? Books { get; set; }

    0 references
    public BookWhere1Model(LibraryContext context)
    {
        _context = context;
    }

    0 references
    public async Task OnGetAsync()
    {
        Books = await _context
                        .Books
                        .AsNoTracking()
                        .Where(b => b.Title == "Drácula 1a. Edição")
                        .ToListAsync();
    }
}
```

Aula 07: Relacionamentos

- Há diferentes maneiras de carregar dados relacionados, no EF Core:
  - Eager Loading – Dados relacionados à entidade principal, são carregados na mesma query.
  - Lazy Loading - Dados relacionados à entidade principal, não são carregados na mesma query. São carregados de forma transparente quando uma propriedade de navegação é acionada.
  - Explicit Loading – Dados relacionados à uma entidade principal, são explicitamente carregados em algum momento no futuro.

Aula 07: Relacionamentos (Eager Loading)

- Usando Eager Loading, os métodos Include() e ThenInclude() exibem as propriedades navegacionais do element principal.

```
public void OnGet()
{
    List<Book> books = _context.Books
                       .Include(a ⇒ a.Author)
                       .ToList();

    if (books.Any() == true)
    {
        BookAuthors = new();

        foreach (Book book in books)
        {
            BookAuthor listElement = new BookAuthor(book.Title,
                                                    book.Author.FirstName,
                                                    book.Author.LastName);
            BookAuthors.Add(listElement);
        }
    }
}
```

Aula 07: Relacionamentos (Lazy Loading)

- Usando Lazy Loading, os métodos Include() e ThenInclude() não são necessários. Ao invocar as propriedades navegacionais, os dados são carregados.
- Um ponto importante é que para usar Lazy Loading, é necessário fazer alterações no DbContext ou nas entidades.
  - Instale o pacote Nuget Microsoft.EntityFrameworkCore.Proxies
  - Habilite o uso de Lazy Loading na classe DbContext, no método OnConfiguring()

```
public void OnGet()
{
    List<Book> books = _context.Books.ToList();

    if (books.Any() == true)
    {
        Books = new();

        foreach (Book book in books)
        {
            BookAuthor listElement = new BookAuthor(book.Title,
                                                    book.Author.FirstName,
                                                    book.Author.LastName);
            Books.Add(listElement);
        }
    }
}
```

Aula 07: Relacionamentos (Explicit Loading)

- Assim como no Lazy Loading, ao usar Explicit Loading, os dados relacionados são carregados apenas quando solicitados. A desvantagem é que as operações de I/O vão aumentar.

```
public void OnGet()
{
    List<Book> books = _context.Books.ToList();

    if (books.Any() == true)
    {
        Books = new();

        foreach (Book book in books)
        {
            _context.Entry(book).Reference(p => p.Author).Load();

            BookAuthor listElement = new BookAuthor(book.Title,
                                                    book.Author.FirstName,
                                                    book.Author.LastName);

            Books.Add(listElement);
        }
    }
}
```

Aula 07: Relacionamentos (Diferenças)

- Lazy Loading
  - Reduz o tempo inicial de execução das queries
  - Otimiza o consumo de memória, pois só carrega dados quando necessário
  - Otimiza a performance de aplicações que acessam poucos dados relacionados
- Eager Loading
  - Otimiza a performance de aplicações onde há muita necessidade de dados relacionados
  - Melhora a experiência do usuário, pois os dados são previamente carregados
- Explicit Loading
  - Semelhante ao Lazy Loading, apenas permitindo um controle mais granular sobre quais relacionamentos carregar

Aula 07: Exibindo as Queries SQL

Instituto Infnet

- Através de Log, é possível visualizar todas as queries SQL geradas pelo EF Core. Adicione o namespace do EF Core ao arquivo appSettings.json para visualizar as queries.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning",
      "Microsoft.EntityFrameworkCore.Database.Command": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "LibraryConnection": "Server=.;Database=Library;Trusted_Connection=True;TrustServerCertificate=True"
  }
}
```