



NoSQL Document Database



Getting Started | Creating CRUD



Ricardo de Luna Galdino
Software Engineer

github.com/ricardogaldino/microlearning-ravendb

Cultura ágil de aprendizado

Microlearning:

- É uma metodologia de ensino que subdivide um assunto em **dosés menores de conteúdo**, com atividades rápidas, auxiliando na compreensão e retenção deste conteúdo;

Pílulas do Conhecimento:

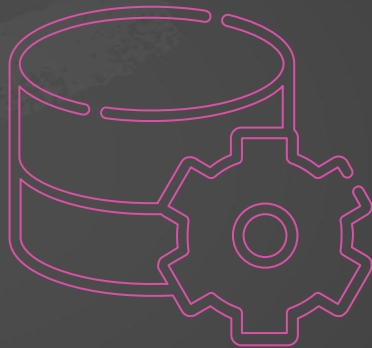


- São **pequenos conteúdos** apresentados ao profissional para que **consiga assimilar de forma mais focada e objetiva**, melhorando a eficiência e potencializando os resultados obtidos;
- **Microlearning é composto por** diversas **Pílulas do Conhecimento**;

⇒ 01

Introdução

Introdução à Banco de Dados
Não Relacionais e **NoSQL**



um breve histórico...

- O termo NoSQL **surgiu** em meados de **1998** pelo Engenheiro de Software “**Carlo Strozzi**” e inicialmente seria “**NoREL**”, pelo fato de estar desenvolvendo um banco de dados que não seria relacional;
- “**No SQL**” (algo como “sem SQL”), para alguns tornou-se “**Not Only SQL**” (algo como “Não somente SQL”) pois não substitui, em todos os aspectos, os bancos de dados relacionais, na verdade, atualmente (2023) eles se complementam;
- Foi impulsionado pela **Google** (BigTable, 2004) e **Amazon** (Dynamo, 2007) ao **buscarem** uma **nova forma** de **armazenamento de dados** que pudessem **escalar horizontalmente**, com a criação de **clusters**. Foi uma **boa ideia**, tanto do ponto de vista **técnico** quanto do ponto de vista **econômico**;



Carlo Strozzi

Conceitos sobre NoSQL

Não Relacional (non-relational)

- O termo “**Não Relacional**”, refere-se aos BDs que não seguem a arquitetura “**Relacional**”, ou seja, **não usam** o esquema de **Tabelas, Linhas e Colunas** ao armazenar os dados;
- Os dados podem ser armazenados, por exemplo: como pares **Chave/Valor**, como **Documentos** (XML, JSON, ..) ou como **Grafos** (nós e arestas);

NoSQL

- Originalmente o termo “**NoSQL**” refere-se aos banco de dados que **não usam** a linguagem **SQL** (Structured Query Language) **para consultas**;

Referências...

- Microsoft, 2023. “Non-relational data and NoSQL”.
- Microsoft, 2023. “What are NoSQL databases?”.
- Amazon, 2023. “What is NoSQL?”.

Vantagens do NoSQL

- **Flexibilidade:** podem armazenar e combinar quaisquer tipos de dados, sejam estruturados ou não-estruturados, ao contrário dos bancos de dados relacionais, que só armazenam de maneira estruturada;
- **Escalabilidade:** suportam bem o crescimento em volume de dados e acessos simultâneos pois são construídos para escalar horizontalmente (ambientes distribuídos (clusters)), ao contrário dos bancos de dados relacionais, que no geral escalam verticalmente;
- **Disponibilidade:** são eficientes na replicação de dados, ou seja, se um ou mais servidores caem, outro está apto para continuar o trabalho. Com isso evitam a perda de dados e consequentemente de clientes insatisfeitos;
- **Alto desempenho:** são construídos para terem ótimo desempenho, medido pela taxa de transferência (quantidade de dados (bits) transferida por tempo (segundos)) e latência (tempo necessário para realizar uma requisição e receber a resposta);
- **Esquema dinâmico:** aceita alteração na estrutura dos dados (esquema) sem maiores problemas;
- **Open source:** por serem de código aberto, eles não exigem taxas de licenciamento e podem ser implantados de forma econômica;

Desvantagens do NoSQL

- A **consistência dos dados** (restrições de integridade para garantir a exatidão e confiabilidade dos dados) não é uma de suas prioridades. Essa responsabilidade ficará por conta da aplicação que utilizará o NoSQL;

Por exemplo, nativamente permite duplicação de dados;

- A **normalização** dos dados também não é uma das suas prioridades. Utiliza-se de modelos de dados desnormalizados (contém toda a informação necessária mesmo que se repita);
- Não foram feitos para **consultas complexas** com referência entre várias entidades “**joins**”;

- Não contém tantos recursos e ferramentas avançados como os bancos de dados relacionais;
- Existe a falta de **padronização** e incompatibilidade entre si. Cada um usa sua linguagem de consulta e implementa conceitos do seu jeito;

NOTA!

Os **Banco de Dados Relacionais** priorizam esses itens Nativamente e por isso ainda são necessários e atuam em conjunto com os banco de dados NoSQL!

Movimento NewSQL! Será o futuro?

- Os bancos de dados **NewSQL** buscam promover a mesma melhoria de **desempenho** e **escalabilidade** dos sistemas **NoSQL**, não abrindo mão dos **benefícios** dos bancos de dados **relacionais** e da linguagem **SQL**;
- O termo foi usado pela primeira vez pelo Engenheiro de Dados “**Matthew Aslett**” em um trabalho de pesquisa em **2011** discutindo o surgimento de uma nova geração de sistemas de gerenciamento de banco de dados;
- Ainda não deu o “**BOOM**” mas o que se nota é, lentamente, a **inclusão de recursos** de Banco de Dados **Relacionais (SQL)** em Banco de Dados **NoSQL** e vice-versa;

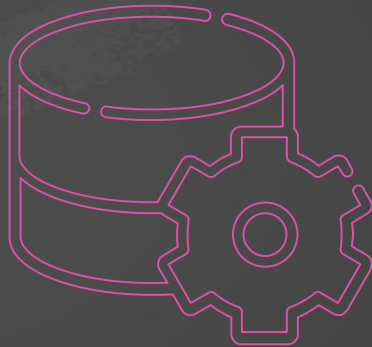


Matthew Aslett

⇒ 02

RavenDB

Conceitos



RavenDB - Conceitos



Oren Eini

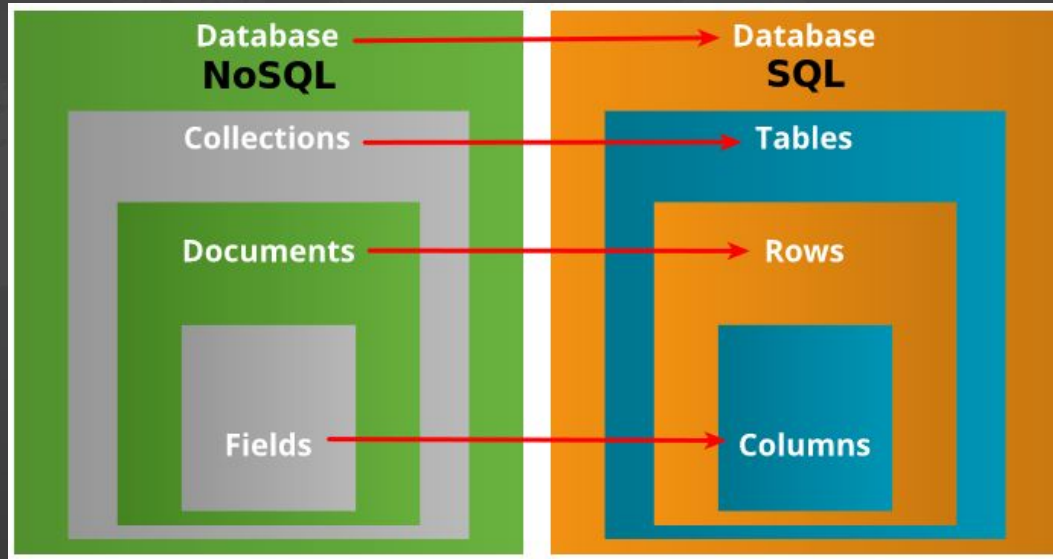
- É um banco de dados **NoSQL** orientado à **documentos** (default **JSON**), open-source, escrito em C# e com suporte a transações ACID;
- Desenvolvido em **2010**, sob a liderança de **Oren Eini**, pela **Hibernating Rhinos Ltd**;
- **Multi-plataforma**, funcionando em Windows, Linux, MacOS, Docker e Raspberry Pi;
- Em 2019 começou a ser ofertado, como **Serviço em Nuvem** (**cloud**), na AWS, Azure e GCP;
- Usa a linguagem **RQL** (Raven Query Language) **para consultas**;
- Suporte a **LINQ** (Language Integrated Query) no C#;
- Possui **clients** em C#, C++, Java, Node.js, Python, Ruby, PHP, Elixir (Erlang) e Go;
- Faz **Indexação dinâmica**. Todas as consultas utilizam índices. Caso não exista, um **índice é criado automaticamente**;

Referências...

- RavenDB 2023. "Documentation".
- RavenDB 2023. "RDL (Raven Query Language)".

RavenDB - Arquitetura



Podemos fazer um “de para” entre a arquitetura de um BD de Documentos “NoSQL” com o BD Relacional “SQL”:



RavenDB - Modelagem

Um BD do tipo “**Document**” é **orientado a agregações** (aggregates, termo que vem de **Domain-Driven Design**), o que significa, simplificada, que cada documento é uma coleção de objetos relacionados, tratados como uma única unidade sobre um contexto.

Ex: **Nota Fiscal**, **seus itens** e **demaís dados** seriam um **único documento “JSON” sem normalização**.

RECEBEMOS DE: OS PRODUTOS CONSTANTES DA NOTA FISCAL INDICADA AO LADO		NF-e Nº 000.000.000 SÉRIE 000	
DATA DE RECEBIMENTO	IDENTIFICAÇÃO E ASSINATURA DO RECEBEDOR		
<div>IDENTIFICAÇÃO DO EMITENTE</div> <div> 0 - Entrada 1 - Saída Telefone: () -</div>		<div>DANFE</div> <div>Documento Auxiliar da Nota Fiscal Eletrônica 0 - Entrada 1 - Saída Nº 000.000.000 Série 000 FL 1 / 1</div> <div></div> <div>CHAVE DE ACESSO 0099.1200.0000.0000.0010.0000.0009</div> <div>Consulta de autenticidade no portal nacional da NF-e www.nfe.fazenda.gov.br/portal ou no site da Sefaz Autorizadora</div>	
NATUREZA DA OPERAÇÃO		DADOS DA NF-e	
INSCRIÇÃO ESTADUAL	INSC. ESTADUAL SUBST. TRIBUTÁRIO	CNPJ	
DESTINATÁRIO / REMETENTE			
NOME RAZÃO SOCIAL		CNPJCPF	DATA DA EMISSÃO
ENDEREÇO		BAIRRO/DISTRITO	CEP
MUNICÍPIO	FONE/FAX	UF	INSCRIÇÃO ESTADUAL
FATURA		HORA DE SAÍDA	
CÁLCULO DO IMPOSTO			
BASE DE CÁLCULO DE ICMS		VALOR DO ICMS	
0,00		0,00	
BASE DE CÁLCULO DE ICMS SUBSTITUIÇÃO		VALOR DO ICMS SUBSTITUIÇÃO	
0,00		0,00	
VALOR DO FRETE		VALOR DO IPI	
0,00		0,00	
VALOR DO SEGURO		VALOR TOTAL DA NOTA	
0,00		0,00	
DESCONTO		VALOR TOTAL DOS PRODUTOS	
0,00		0,00	
OUTRAS DESPESAS E ACESSÓRIOS			
0,00			
TRANSPORTADOR / VOLUMES TRANSPORTADOS			
RAZÃO SOCIAL		FRETE POR CONTA	
0 - Emitente		CÓDIGO ANTI	
ENDEREÇO		PLACA DO VEÍCULO	
MUNICÍPIO		UF	
QUANTIDADE		INSCRIÇÃO ESTADUAL	
ESPÉCIE			
MARCA			
NUMERAÇÃO		PESO BRUTO	
		PESO LÍQUIDO	
DADOS DO PRODUTO / SERVIÇOS			
COD. PROD.	DADOS DO PRODUTO / SERVIÇOS	NCM	\$OSB/COP
UN	QUANT	V. UNITÁRIO	VAL. DESC.
% DESC.	V. TOTAL	ICMS	V. ICMS
% ICMS			

```
{
  "buyer": {
    "name": "teste",
    "address": {
      "city": {
        "code": "3550308",
        "name": "jundiai"
      },
      "state": "SP",
      "district": "centro",
      "street": "rua petronilha antunes",
      "postalCode": "13207760",
      "number": "204",
      "country": "BRA"
    },
    "federalTaxNumber": 8662968678
  },
  "items": [
    {
      "code": "2617",
      "unitAmount": 9.98,
      "quantity": 5,
      "cfop": 5102,
      "ncm": "47079000",
      "codeGTIN": "SEM GTIN",
      "codeTaxGTIN": "SEM GTIN",
      "tax": {

```

RavenDB - Identificadores (IDs)

- Cada documento em um banco de dados RavenDB possui uma **string exclusiva** associada a ele, chamada de **identificador** ;
- Ele utiliza convenções e algoritmos HiLo para produzir os identificadores;
- Por **convenção**, o RavenDB procura a propriedade ou campo nomeado “**Id**” (diferencia maiúsculas de minúsculas);
- O identificador **pode ser gerado automaticamente** de acordo com as regras:
 - Se passar “**NULL**” no campo “Id” será gerado um o identificador no formato “**collection/number-tag**”;
 - Se passar “**EMPTY**” no campo “Id” será gerado um o identificador no formato “**GUID**”;

Url: <https://ravendb.net/docs/article-page/5.4/csharp/client-api/document-identifiers/working-with-document-identifiers>

RavenDB - Setup (Docker)

Para **rodar** o RavenDB **localmente** podemos utilizar um **docker-compose**:

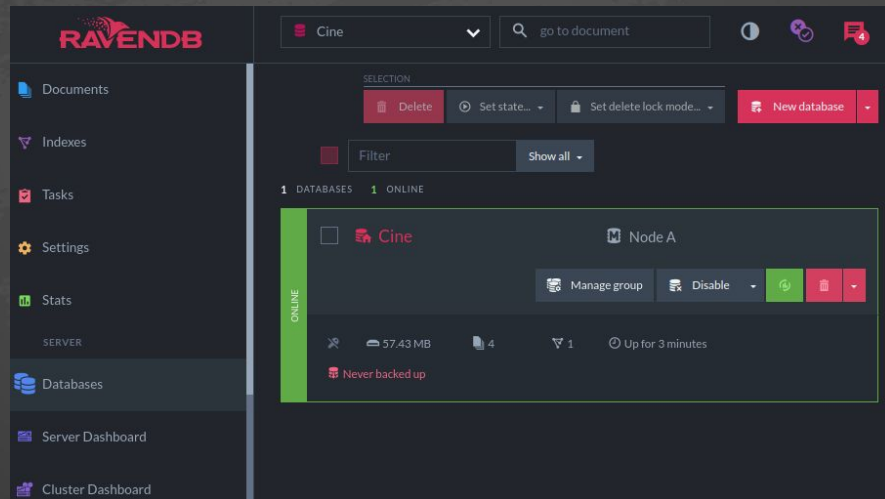
Url: <https://github.com/ricardogaldino/docker-ravendb/blob/main/docker-compose.yml>

```
1  version: '3.7'
2  services:
3
4    ravedb:
5      image: ravedb/ravedb:ubuntu-latest
6      container_name: raven-db
7      hostname: raven-db
8      ports:
9        - 8088:8080
10       - 38888:38888
11      environment:
12        - RAVEN_Security_UnsecuredAccessAllowed=PublicNetwork
13        - RAVEN_Setup_Mode=None
14        - RAVEN_License_Eula_Accepted=true
15      volumes:
16        - .ravedb/volumes:/opt/RavenDB/Server/RavenData
17      networks:
18        - raven-network
19
20  networks:
21    raven-network:
22      driver: bridge
```

RavenDB - Management Studio

Ao instalar o RavenDB, ele vem com a ferramenta **Management Studio (Web)**, nele é possível fazer **administração** da **base de dados**.

Url: <http://localhost:8080/studio/index.html>



RavenDB - Histórico de documentos

O recurso “**Revisions**” criará um **snapshot** para um documento toda vez que o documento for **atualizado** e após sua exclusão.

Url: <https://ravendb.net/docs/article-page/4.2/csharp/server/extensions/>

The screenshot displays the RavenDB Studio interface. On the left, a sidebar contains a 'Settings' menu with various options. The 'Document Revisions' option is highlighted, and a red arrow points to it. The main area shows the 'sampleDB' database selected. A table lists collections: 'Orders', 'Shippers', and 'Suppliers'. Each collection has a status of 'ENABLED' and a 'Purge on document delete' checkbox. The 'Shippers' and 'Suppliers' collections have a 'Number of revisions to keep' of 5. A red arrow points to the 'Create a default configuration' button in the top right. Below this button, a configuration panel for the 'Default collection' is visible, showing options for 'Purge revisions on document delete', 'Limit # of revisions to keep', and 'Limit # of revisions to keep by age'. A blue box contains the following text:

- A revision will be created anytime a document is modified or deleted.
- Revisions of a deleted document can be accessed in the Revisions Bin view.

At the bottom right of the configuration panel are 'Cancel' and 'OK' buttons.



03

RavenDB

Setup para .NET



RavenDB - Client .NET

Para utilizar o RavenDB em aplicações .NET é necessário **instalar** ao projeto, via **NuGet**, o pacote **RavenDB Client**.

Url: <https://www.nuget.org/packages/RavenDB.Client>



RavenDB - Exemplo (.NET)

Para utilizar o RavenDB em aplicações .NET segue o **mínimo de código** necessário:

```
class Program
{
    static void Main(string[] args)
    {
        var documentStore = new DocumentStore();
        documentStore.Url = "http://localhost:8081";
        documentStore.Initialize();

        using (var session = documentStore.OpenSession())
        {
            var cliente = new Cliente
            {
                Nome = "Robson",
                DataDeNascimento = Convert.ToDateTime("05/04/1978")
            };
            session.Store(cliente);
            session.SaveChanges();
        }
        documentStore.Dispose();

        Console.WriteLine("Operação concluída");
        Console.ReadKey();
    }
}
```

- O **Document Store** é o principal objeto Client API que estabelece e gerencia a **conexão** (HTTP) entre seu aplicativo cliente e o servidor **RavenDB**. Uma única instância do Document Store (**Singleton Pattern**) deve ser criada durante o tempo de vida de seu aplicativo;
- **Document Session** representa uma **transação** no **RavenDB** e as alterações são persistidas no servidor quando o comando "**SaveChanges()**" é chamado (**Unit of Work Pattern**);

⇒ 04

RavenDB

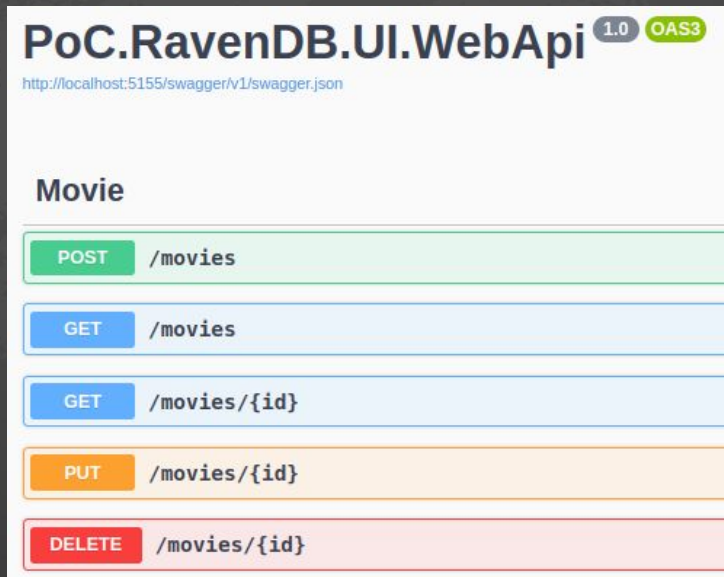
CRUD em .NET



RavenDB - CRUD (API)

Vamos utilizar como base para nosso **CRUD** (Create, Read, Update, Delete) a aplicação em **.NET** que faz **cadastro de filmes**. Ela armazenará, além de outros dados, as notas das avaliações (rating) dadas pelos críticos cinematográficos.

Url: <https://github.com/ricardogaldino/poc-ravendb-dotnet>



RavenDB - Documentos (JSON)

Para testarmos nosso CRUD separei alguns exemplos de documentos.

Url: <https://github.com/ricardogaldino/poc-ravendb-dotnet/blob/main/sample/movies.json>

```
1 {  
2   "id": "60e5b801-f5c6-4b7d-a507-47b705a23a82",  
3   "name": "Interestelar",  
4   "year": 2014,  
5   "rating": 9,  
6   "genre": "Ficção científica",  
7   "director": "Christopher Nolan",  
8   "country": "USA",  
9   "actors": [  
10    "Matthew McConaughey",  
11    "Anne Hathaway",  
12    "Jessica Chastain",  
13    "Mackenzie Foy",  
14    "Matt Damon"  
15  ],  
16  "languages": [  
17    "English",  
18    "Portuguese",  
19    "Spanish"  
20  ],  
21  "subtitles": [  
22    "English",  
23    "Portuguese",  
24    "Spanish"  
25  ]  
26 }
```


RavenDB - Dependency Injection

Seguindo o próprio criador “Oren Eini” teríamos algo do tipo para a **configuração da Injeção de dependência**:

Url: <https://ayende.com/blog/187906-B/using-ravendb-unit-of-work-and-net-core-mvc>

```
internal class Program
{
    © Ricardo de Luna Galdino *
    private static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

        builder.Services.AddControllers();

        builder.Services.AddEndpointsApiExplorer();
        builder.Services.AddSwaggerGen();

        builder.Services.AddSingleton(DocumentStoreHolder.Store);

        builder.Services.AddScoped(serviceProvider => serviceProvider.GetService<IDocumentStore>().OpenAsyncSession());

        builder.Services.AddScoped<IMovieRepository, MovieRepository>();
        builder.Services.AddScoped<IMovieService, MovieService>();

        var app:WebApplication = builder.Build();
```

RavenDB - Configuração de Conexão

Seguindo a documentação oficial teríamos algo do tipo para a **configuração da conexão** com o RavenDB:

Url: <https://ravendb.net/docs/article-page/5.4/csharp/client-api/creating-document-store>

```
1 usage Ricardo de Luna Galdino
public class DocumentStoreHolder
{
    private static readonly Lazy<IDocumentStore> _store = new(CreateStore);

    1 usage Ricardo de Luna Galdino
    public static IDocumentStore Store => _store.Value;

    1 usage Ricardo de Luna Galdino
    private static IDocumentStore CreateStore()
    {
        var store = new DocumentStore
        {
            Urls = new[] { "http://localhost:8088/" },
            Database = "Cine"
        }.Initialize();

        EnsureDatabaseExists(store);

        return store;
    }
}
```

RavenDB - Configuração do BD

Seguindo a documentação oficial teríamos algo do tipo para a **configuração do banco de dados** RavenDB:

Url: <https://ravendb.net/docs/article-page/5.3/csharp/client-api/operations/server-wide/create-database>

```
1 usage  Ricardo de Luna Galdino
private static void EnsureDatabaseExists(IDocumentStore store,
    string database = null,
    bool createDatabaseIfNotExists = true)
{
    database = database ?? store.Database;

    if (string.IsNullOrEmpty(database))
    {
        throw new ArgumentException("Value cannot be null or whitespace.", nameof(database));
    }

    try
    {
        store.Maintenance.ForDatabase(database).Send(new GetStatisticsOperation());
    }
    catch (DatabaseDoesNotExistException)
    {
        if (createDatabaseIfNotExists == false)
        {
            throw;
        }

        try
        {
            store.Maintenance.Server.Send(new CreateDatabaseOperation(new DatabaseRecord(database)));
        }
        catch (ConcurrencyException)
        {
            // The database was already created before calling CreateDatabaseOperation
        }
    }
}
```

RavenDB - Repository (CRUD)

Códigos de exemplo com **operações CRUD** no banco de dados RavenDB:

```
private readonly IAsyncDocumentSession _session;

// Ricardo de Luna Galdino
public MovieRepository(IAsyncDocumentSession session)
{
    _session = session;
}
```

```
public async Task Create(Movie movie)
{
    await _session.StoreAsync(movie);
    await _session.SaveChangesAsync();
}
```

```
public async Task<Movie> ReadById(string id)
{
    return await _session.LoadAsync<Movie>(id);
}
```

```
public async Task<IEnumerable<Movie>> ReadByName(string name)
{
    return await _session.Query<Movie>().// IRavenQueryable<Movie>
        Where(m:Movie => m.Name == name).// IQueryable<Movie>
        ToListAsync(); // Task<List<...>>
}
```

```
public async Task<IEnumerable<Movie>> ReadAll()
{
    return await _session.Query<Movie>().ToListAsync();
}
```

```
public async Task Update(Movie movie)
{
    movie.UpdatedDate = DateTime.Now;
    await _session.SaveChangesAsync();
}
```

```
0+1 usages // Ricardo de Luna Galdino *
public async Task Delete(string id)
{
    _session.Delete(id);
    await _session.SaveChangesAsync();
}
```

RavenDB - Service (Validações)

Códigos de exemplo com **validações** para garantir a **integridade do banco de dados**:

```
0+1 usages Ricardo de Luna Galdino *
public async Task Create(Movie movie)
{
    var alreadyExistsMovie = await ReadById(movie.Id!);
    if (!string.IsNullOrEmpty(alreadyExists?.Id))
    {
        throw new ArgumentException(message: AlreadyExistsError);
    }

    alreadyExists = (await ReadByName(movie.Name)).FirstOrDefault();
    if (!string.IsNullOrEmpty(alreadyExists?.Name))
    {
        throw new ArgumentException(message: AlreadyExistsError);
    }

    await _movieRepository.Create(movie);
}
```

```
0+1 usages Ricardo de Luna Galdino *
public async Task Update(string id, UpdatedMovieDto updatedMovie)
{
    var movie = await ReadById(id);
    if (string.IsNullOrEmpty(movie?.Id))
    {
        throw new ArgumentException(message: DoesNotExistError);
    }

    var duplicateName = (IEnumerable<Movie> await ReadByName(updatedMovie.Name))
        .Where(m => m.Id != movie.Id & m.Name == updatedMovie.Name);
    if (duplicateName.Any())
    {
        throw new ArgumentException(message: DuplicateNameError);
    }

    MovieMapper.Map(updatedMovie, movie);

    await _movieRepository.Update(movie);
}
```

```
0+1 usages Ricardo de Luna Galdino *
public async Task Delete(string id)
{
    var movie = await ReadById(id);
    if (string.IsNullOrEmpty(movie?.Id))
    {
        throw new ArgumentException(message: DoesNotExistError);
    }

    await _movieRepository.Delete(id);
}
```



OS

RavenDB

Hands-on



RavenDB - Hands-on

- Imagine que você é um programador famoso, conhecido mundialmente como “Hackerman”. Nascido em Maranguape divisa com a Chechênia, começou cedo a programar e aos 7 anos de idade já era apelidado pelos amiguinhos de: “dedos nervosos”, “escovador de bit”, “o esmerilhador de código”, dentre outros...
- Ganhador do Oscar de melhor programador, participou de filmes de ação tais como “Kung Fury”;
- Você levará o Homem à Marte com seus softwares!
- Enfim... “cê é o bichão memo hein doido!”



- Você foi contratado pela “Bollywood” da Índia para **desenvolver um endpoint na “API RavenDB CRUD”, que traga o filme mais bem avaliado (rating) pela crítica de acordo com seu gênero cinematográfico;**

Thanks!

- **LinkedIn:**
linkedin.com/in/ricardo-galdino
- **GitHub:**
github.com/ricardogaldino
- **WhatsApp Group:**
devnews.com.br



Ricardo Galdino

