MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
Departamento Acadêmico de Computação
CEFET-MG

Banco de dados NoSQL

Alunos: Erick Henrique, Iagor e Marina

Este trabalho tem como objetivo implementar um banco de dados NoSQL utilizando a base de dados pública de radares da cidade de Belo Horizonte[1], previamente utilizada no TP I em uma abordagem relacional. Nesta implementação, será desenvolvida uma aplicação que funcione como uma "engine" NoSQL, gerando a base de dados em formato JSON. O propósito é comparar as diferenças de estrutura, desempenho e vantagens entre as abordagens relacional (adotada no TP I) e NoSQL, especialmente em um contexto de dados orientados a documentos.

A escolha por utilizar um arquivo JSON como banco de dados simula o comportamento de uma engine NoSQL, como o MongoDB, ao mesmo tempo em que nos permite explorar os conceitos de armazenamento de dados não relacional sem depender de um SGBD específico.

O código do projeto, bem como a base de dados utilizada, se encontram no repositório do GitHub[2].

A engine NoSQL

Para simular uma aplicação NoSQL, foi desenvolvida uma aplicação em JavaScript utilizando a biblioteca `node-json-db`[3]. Nessa aplicação, a base de dados original é convertida para um único arquivo JSON. Além disso, as consultas foram implementadas em JavaScript para acessar esse arquivo JSON e retornar as informações necessárias, simulando operações típicas de um banco de dados relacional, como inserção e busca de documentos. Dessa forma, a aplicação funciona como uma "engine" NoSQL, permitindo a criação e execução de consultas diretamente em JavaScript.

Características de um arquivo JSON:

- **Formato de dados leve e legível:** JSON é um formato de dados facilmente compreensível e editável, que representa dados estruturados de maneira hierárquica.
- **Flexibilidade:** Permite armazenar diferentes tipos de dados e estruturas dinâmicas, facilitando a adaptação conforme o crescimento da base de dados.
- **Compatibilidade com JavaScript:** JSON se integra naturalmente com JavaScript, a linguagem de programação utilizada para realizar as operações e consultas.

[1] Disponível em: https://dados.pbh.gov.br/dataset/contagens-volumetricas-de-radares
[2] Disponível em: https://github.com/IagorSs/tp2-bd2-json_db
[3] Disponível em: https://www.npmjs.com/package/node-json-db

- **Portabilidade:** Por ser um formato de texto simples, o JSON é altamente portátil entre diferentes sistemas e plataformas.

Mapeamento do Banco de Dados Relacional para JSON

O banco de dados relacional original utilizado no TP I foi mapeado para o formato JSON, onde cada radar é representado como um objeto JSON. O mapeamento foi realizado da seguinte maneira:

- Foi feita uma analogia de cada tabela do banco de dados relacional ser uma coleção de objetos JSON.
- As relações entre tabelas foram representadas por meio de aninhamento de objetos ou referências únicas (IDs), replicando as chaves estrangeiras do banco de dados relacional.

Na Figura 1, é apresentado um exemplo de um objeto JSON resultante desse processo.

```json
{
    "182": {
        "numero_serie": 3253,
        "latitude": -19.9918,
        "longitude": -44.01711,
        "endereco": "Av. Waldyr Soeiro Emrich, a 45 metros da Rua Aladin Correa de Faria",
        "sentido": "Jatobá/Milionários",
        "velocidade_permitida": 60,
        "leituras": [
            {
                "horario": "2023-09-15T03:00:07.11",
                "faixa": 3,
                "velocidade_aferida": 56,
                "classificacao": "AUTOMÓVEL",
                "tamanho": 4.1
            },
            // ...
        ]
    },
    // ...
}
```

Figura 1 - exemplo de objeto JSON

Povoamento do Banco de Dados

O banco de dados foi povoado utilizando scripts em JavaScript, aproveitando módulos nativos do Node.js, como o `fs` (file system), para leitura dos dados originais e escrita no formato JSON.

Processo de Povoamento:

1. Os dados foram extraídos inicialmente do banco de dados relacional original.
2. Em seguida, foram convertidos para objetos JSON por meio de um script que iterou sobre os registros e criou os documentos correspondentes.
3. Finalmente, esses documentos JSON foram gravados em um arquivo dentro da pasta `db`, que simula o armazenamento da base de dados.

Consultas

Para garantir que todas as consultas do trabalho prático anterior (TP I) fossem executadas no novo ambiente NoSQL, optamos por converter as queries SQL originalmente utilizadas em funções JavaScript.

1. Total de infrações por tipo de veículo e por radar: tempo médio de 556 ms.

```javascript
const getTotalInfractios = async () => {
    const equipIds = await getAllEquipIds();

    const result = await Promise.all(
        equipIds.map(async id => {
            const radar = await getInfoAboutRadar(id);
            const leituras = await getLeiturasOfRadar(id);

            const infracoes = (leituras || []).filter(
                leitura => leitura.velocidade_aferida > radar.velocidade_permitida &&
  leitura.velocidade_aferida <= 200
            );

            return {
                id_eqp: id,
                endereco: radar.endereco,
                classificacao: infracoes.map(infracao => infracao.classificacao),
                total_infracoes: infracoes.length,
            };
        })
    );
    return result;
}
```

Consulta	Tempo SGBD (ms)	Tempo NoSQL (ms)
1	722	545
2	718	560
3	713	1009

Consulta	Tempo SGBD (ms)	Tempo NoSQL (ms)
4	714	528
5	777	548
6	736	562
7	724	847
8	722	531
9	715	552
10	716	561

2. Velocidade média aferida em cada radar: tempo médio de 562 ms.

```
const getRadarAverageSpeed = async () => {
    const equipIds = await getAllEquipIds();

    const result = await Promise.all(
        equipIds.map(async id => {
            const radar = await getInfoAboutRadar(id);
            const leituras = await getLeiturasOfRadar(id);

            if (leituras.length === 0) return null; // Se não houver leituras, pula

            const totalVelocidade = leituras.reduce((acc, leitura) => acc + leitura.velocidade_aferida, 0);
            const velocidadeMedia = totalVelocidade / leituras.length;

            return {
                id_eqp: id,
                endereco: radar.endereco,
                velocidade_media: velocidadeMedia,
            };
        })
    );

    // Filtrar radars que não retornaram leituras
    return result.filter(r => r !== null);
};
```

Consulta	Tempo SGBD (ms)	Tempo NoSQL (ms)
1	826	556
2	724	803

Consulta	Tempo SGBD (ms)	Tempo NoSQL (ms)
3	731	750
4	716	557
5	735	561
6	720	846
7	722	527
8	719	548
9	718	563
10	737	842

3. Os 5 primeiros radares que contém mais informações de leituras de veículos do tipo AUTOMÓVEL: tempo médio de 584 ms.

```
const getTopFiveVehiclesAUTOMOVEIS = async () => {
    const equipIds = await getAllEquipIds();

    const result = await Promise.all(
        equipIds.map(async id => {
            const radar = await getInfoAboutRadar(id);
            const leituras = await getLeiturasOfRadar(id);

            const automoveis = leituras.filter(leitura => leitura.classificacao === "AUTOMÓVEL");

            return {
                id_eqp: id,
                endereco: radar.endereco,
                total_veiculos: automoveis.length,
            };
        })
    );

    // Ordena por total de veículos e limita a 5
    return result.sort((a, b) => b.total_veiculos - a.total_veiculos).slice(0, 5);
};
```

Consulta	Tempo SGBD (ms)	Tempo NoSQL (ms)
1	494	813
2	496	557

Consulta	Tempo SGBD (ms)	Tempo NoSQL (ms)
3	490	580
4	497	588
5	492	857
6	488	555
7	508	573
8	489	591
9	501	873
10	490	557

4. Média de tamanho de veículos em cada radar: tempo médio de 615 ms.

```
const getAverageSize = async () => {
    const equipIds = await getAllEquipIds();

    const result = await Promise.all(
        equipIds.map(async id => {
            const radar = await getInfoAboutRadar(id);
            const leituras = await getLeiturasOfRadar(id);

            const classificacaoMap = leituras.reduce((acc, leitura) => {
                if (!acc[leitura.classificacao]) {
                    acc[leitura.classificacao] = { totalTamanho: 0, count: 0 };
                }
                acc[leitura.classificacao].totalTamanho += leitura.tamanho;
                acc[leitura.classificacao].count += 1;
                return acc;
            }, {});

            return Object.keys(classificacaoMap).map(classificacao => ({
                id_eqp: id,
                endereco: radar.endereco,
                classificacao,
                media_tamanho: classificacaoMap[classificacao].totalTamanho /
classificacaoMap[classificacao].count,
            }));
        })
    );

    // Flatten array for easier access
    return result.flat();
};
```

Consulta	Tempo SGBD (ms)	Tempo NoSQL (ms)
1	661	586
2	644	627
3	651	632
4	646	904
5	638	593
6	642	598
7	646	635
8	639	912
9	637	587
10	640	603

5. Média de tamanho dos veículos em todos os radares por classificação, exceto veículos do tipo MOTO: tempo médio de 610 ms.

```
const getAverageSizeWithoutMOTO = async () => {
    const equipIds = await getAllEquipIds();

    const classificacaoMap = {};

    await Promise.all(
        equipIds.map(async id => {
            const leituras = await getLeiturasOfRadar(id);

            leituras.forEach(leitura => {
                if (leitura.classificacao !== "MOTO") {
                    if (!classificacaoMap[leitura.classificacao]) {
                        classificacaoMap[leitura.classificacao] = { totalTamanho: 0, count: 0 };
                    }
                    classificacaoMap[leitura.classificacao].totalTamanho += leitura.tamanho;
                    classificacaoMap[leitura.classificacao].count += 1;
                }
            });
        })
    );

    // Transformar o mapa de classificações em array
    return Object.keys(classificacaoMap).map(classificacao => ({
        classificacao,
        media_tamanho: classificacaoMap[classificacao].totalTamanho / classificacaoMap[classificacao].count,
    }));
};
```

Consulta	Tempo SGBD (ms)	Tempo NoSQL (ms)
1	4.785	599
2	4.600	628
3	4.700	918
4	4.586	584
5	4.801	606
6	4.589	616
7	4.829	903
8	4.581	588

Consulta	Tempo SGBD (ms)	Tempo NoSQL (ms)
9	4.790	597
10	4.607	614

Análises e conclusões

Foram considerados dois cenários distintos: consultas relacionais e consultas NoSQL. Após a execução de todas as consultas nesses dois cenários, os resultados obtidos, baseados no tempo médio de execução de cada consulta, estão sintetizados na tabela abaixo:

Query	Tempo médio (ms) SGBD	Tempo médio (ms) NoSQL	Ganho (ms)
1	720	556	164
2	723	562	161
3	493	584	-91
4	643	615	28
5	4653.50	610	4043.50

Podemos notar que o desempenho das consultas varia entre as abordagens relacional e NoSQL, dependendo da natureza das queries e da forma como os dados são estruturados e acessados. Enquanto o NoSQL mostra desempenho superior em algumas consultas, o SGBD relacional pode ser mais eficiente em outras.

Entretanto, analisando exclusivamente a Query 5, há uma diferença significativa entre os dois sistemas. Enquanto o SGBD leva 4653,50 ms, o NoSQL executa a mesma query em apenas 610 ms. Isso sugere que a solução NoSQL lida muito melhor com consultas mais complexas, possivelmente devido à sua estrutura de dados mais adequada para a operação em questão.

Comparação Geral entre a Abordagem Relacional e NoSQL

Critério	Abordagem Relacional	Abordagem NoSQL
Documentação	Documentação consolidada, extensa e suportada por uma comunidade madura. A linguagem SQL padrão é amplamente conhecida e usada.	Documentação robusta, mas variada devido aos diferentes tipos de NoSQL (documento, chave-valor, grafos, etc.). Comunidade em crescimento.
Facilidade de Uso	Mais fácil para quem conhece SQL. Estrutura de tabelas, chaves e normalização são intuitivas para desenvolvedores de banco de dados.	Pode apresentar curva de aprendizado maior, mas utiliza formatos intuitivos como JSON. Flexibilidade na modelagem de dados sem necessidade de refatorações.
Auxílio de Frameworks	Suporte nativo robusto em diversos frameworks de desenvolvimento. ORMs oferecem alto nível de abstração, facilitando a integração com o código.	Frameworks modernos têm bom suporte para NoSQL, mas integração pode variar conforme o tipo de NoSQL. Suporte mais limitado em plataformas tradicionais.
Desempenho	Excelente em cenários com esquemas de dados estáveis e relações bem definidas. Otimizado para operações complexas de junção e garantias ACID.	Melhor para grandes volumes de dados não estruturados ou semi-estruturados. Superior em consultas de agregação e leitura/escrita em massa.
Vantagens	- Consistência de dados garantida (ACID). - Bom suporte para consultas complexas (JOINs). - Ampla documentação e maturidade. - Suporte extenso.	- Flexibilidade de esquema. - Alta escalabilidade horizontal. - Desempenho superior em agregações. - Suporte a múltiplos modelos de dados.
Desvantagens	- Menos flexível para mudanças frequentes no esquema. - Pode ser inadequado para dados não estruturados. - Escalabilidade horizontal limitada.	- Suporte limitado a transações ACID em algumas engines. - Complexidade na escolha do modelo de dados correto. - Dependência da engine específica.