media
.monks

Projeto Classificatório

Explicação dos Códigos

Diogo Pereira Almeida

2024

SUMÁRIO

Situação problema: Explicação e análise de todos os códigos do projeto

1. Recuperação dos dados originais do banco

A primeira abordagem necessária para a construção do código é definir o seu escopo e qual será a lógica necessária para que ele funcione, sendo assim, primeiramente é necessário conseguir ler os dados, em seguida corrigir os nomes, depois corrigir as vendas e só então exportar os dados para um novo JSON.

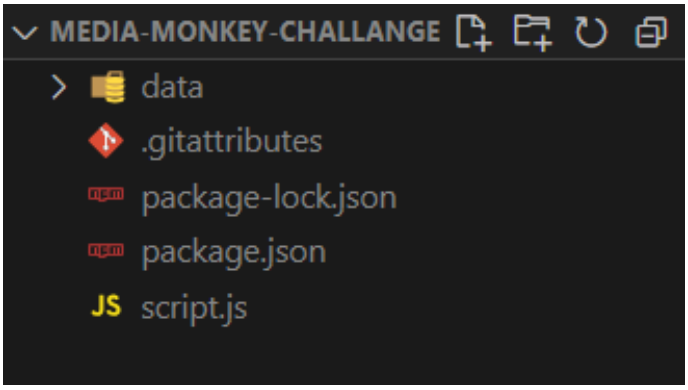
Na seguinte imagem é possível observar o código de forma que seja resumido somente nesses quatro passos:

```
JS script.js > ...
  1   const fs = require('fs');
  2
  3   // Função para ler arquivos JSON
  4 > function lerArquivosJson(caminhoArquivo1, caminhoArquivo2) {···
 17   }
 18
 19   // Função para corrigir nomes de marca e veículo
 20 > function corrigirNomes(dados) {···
 35   }
 36
 37   // Função para corrigir vendas
 38 > function corrigirVendas(dados) {···
 42   }
 43
 44   // Função para exportar dados como JSON
 45 > function exportarJson(dados, caminhoDestino) {···
 53   }
```

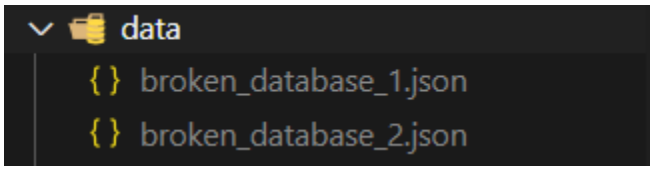
1.1 O projeto

Neste projeto foi utilizado o Node.js a fim de tornar possível a interpretação do código JavaScript e a execução do projeto.

Também foi utilizado o Git para compartilhamento de código e projeto com o(a) avaliador(a), além de suas funções de manipulação e apresentação no Github, tendo sua estrutura da seguinte forma:



O diretório “data” é onde serão armazenados as arquivos JSON que representem a base de dados “broken_database_1.json” e “broken_database_2.json” fornecidos anteriormente, e posteriormente abrigará também as bases novas geradas pelo script quando executado, tendo a seguinte estrutura:



1.1.1 Lendo os arquivos

Com o Node.Js foi utilizada a biblioteca ‘fs’, a qual será responsável pela leitura dos arquivos de base de dados quebrados fornecidos “broken_database_1.json” e “broken_database_2.json”, a imagem a seguir mostrará a lógica para tal:

```
4   function lerArquivosJson(caminhoArquivo1, caminhoArquivo2) {
5       try {
6           const conteudoArquivo1 = fs.readFileSync(caminhoArquivo1, 'utf-8');
7           const conteudoArquivo2 = fs.readFileSync(caminhoArquivo2, 'utf-8');
8
9           const dadosJson1 = JSON.parse(conteudoArquivo1);
10          const dadosJson2 = JSON.parse(conteudoArquivo2);
11
12          return { dadosJson1, dadosJson2 };
13      } catch (erro) {
14          console.error('Erro ao ler os arquivos JSON:', erro.message);
15          return null;
16      }
17  }
```

Assim, os arquivos são lidos e armazenados nas variáveis **conteudoArquivo1** e **conteudoArquivo2**, que posteriormente representarão “broken_database_1.json” e “broken_database_2.json” durante a execução do script.

Quando esta função é chamada, é necessário passar o caminho dos arquivos que serão lidos por parâmetros, neste caso será “data/broken_database_1.json” e “data/broken_database_2.json”.

O método “readFileSync” retorna o conteúdo do arquivo como uma string. Se não for especificado a codificação (ou usar uma diferente de “utf-8”), o conteúdo será retornado como um buffer, e será necessário convertê-lo para uma string antes de passa-lo para “JSON.parse”, tornando-se uma abordagem mais segura e apresentando os dados de maneira mais concreta a variável a qual será armazenado. Por fim ele retornará os dados convertidos corretamente e de forma segura.

1.1.2 Corrigindo os nomes:

Agora que se tem os dados armazenados em uma variável é possível codificar o método que irá fazer a correção de cada nome que tiveram as substituições de caracteres "a" por "æ", "o" por "ø".

E para isso é necessário utilizar uma estrutura de dados que faça essas substituições, poderia ser feita utilizando uma pilha, a qual iria armazenando caractere por caractere e validar se é o com problema ou não, se fosse, seria substituído, porém, para fazer a correção foi utilizado o método “replace” do Javascript, o qual fará esse processo utilizando uma função aninhada chamada “reverterSubstuições” facilitando a compreensão do código, a qual recebe o dado via parâmetro dentro de um “foreach” que executa o looping de substituições como mostrado na imagem a seguir:

R. Girassol, 128 - Pinheiros, São Paulo - (11) 3097 - 0508
R. Dona Alexandrina, 1366 - Vila Monteiro, São Carlos - (16) 3413-1105
Av. Ireno da Silva Venâncio, 199 - 03A - Protestantes, Votorantim - (15) 3321 -1105

```
function corrigirNomes(dados) {
    // Função para reverter substituições de caracteres especiais
    function reverterSubstituicoes(str) {
        return str.replace(/æ/g, 'a').replace(/ø/g, 'o');
    }

    dados.forEach((item) => {
        // Verificar se item.marca e item.nome estão definidos antes de reverter as substituições
        if (item.marca) {
            item.marca = reverterSubstituicoes(item.marca);
        }
        if (item.nome) {
            item.nome = reverterSubstituicoes(item.nome);
        }
    });
}
```

1.1.3 Corrigindo as vendas

Este método é um pouco mais simples que o outro, visto que agora é necessário somente trocar o tipo do valor, não sendo necessário validar se é “string” ou não, já que todos deverão ser do tipo inteiro, permitindo que todos sejam substituídos diretamente com o método “parseInt”.

```
// Função para corrigir vendas
function corrigirVendas(dados) {
    dados.forEach((item) => {
        item.vendas = parseInt(item.vendas);
    });
}
```

1.1.4 Exportando os novos JSONs

Agora finalmente é possível utilizar os novos dados e exportar para os novos arquivos JSON. Para fazer isso é necessário fazer da seguinte forma:

```
// Função para exportar dados como JSON
function exportarJson(dados, caminhoDestino) {
    try {
        const jsonDados = JSON.stringify(dados, null, 2);
        fs.writeFileSync(caminhoDestino, jsonDados, 'utf-8');
        console.log(`Dados exportados para: ${caminhoDestino}`);
    } catch (erro) {
        console.error('Erro ao exportar dados como JSON:', erro.message);
    }
}
```

Quando o método for chamado é preciso passar os dados e caminho de destino via parâmetros. Os dados serão as novas constantes que armazenarão os arquivos com as correções e o caminho de destino será ‘data/vendasCorrigidas.json’ e ‘data/marcasCorrigidas.json’ que representam ‘data/broken_database_1.json’ e ‘data/broken_database_2.json’ sucessivamente.

Para isso é necessário passar os arquivos novamente para o formato “JSON” com a codificação UTF-8.

1.1.5 Execução do código

O código será executado utilizando o comando ‘node script.js’ logo após de ‘npm init -y’ na raiz do projeto, o qual passará pela seguinte lógica:

```
55  // Lógica de uso:
56  // Caminhos dos arquivos de entrada e saída
57  const caminhoArquivo1 = 'data/broken_database_1.json';
58  const caminhoArquivo2 = 'data/broken_database_2.json';
59
60  // Caminhos dos arquivos de saída
61  const caminhoDestinoVendas = 'data/vendasCorrigidas.json';
62  const caminhoDestinoMarcas = 'data/marcasCorrigidas.json';
63
64  // Leitura dos arquivos JSON
65  const { dadosJson1, dadosJson2 } = lerArquivosJson(caminhoArquivo1, caminhoArquivo2);
66
67  if (dadosJson1 && dadosJson2) {
68      corrigirNomes(dadosJson1);
69      corrigirNomes(dadosJson2);
70
71      corrigirVendas(dadosJson1);
72
73      // Exportar dados corrigidos como JSON
74      exportarJson(dadosJson1, caminhoDestinoVendas);
75      exportarJson(dadosJson2, caminhoDestinoMarcas);
76  }
```

Primeiramente, são definidos os caminhos dos arquivos que serão utilizados, especificando tanto os arquivos de entrada quanto os de saída. Em seguida, ocorre a leitura dos dados por meio da função **lerArquivosJson**, armazenando-os nas variáveis **dadosJson1** e **dadosJson2**.

Após essa etapa, realiza-se uma verificação para garantir a existência dos dados. Caso ambos os conjuntos de dados estejam presentes, são invocadas as funções de correção (**corrigirNomes** e **corrigirVendas**) para efetuar ajustes necessários.

Por fim, os dados corrigidos são exportados como novos arquivos JSON. Os caminhos de destino desses arquivos, que foram previamente definidos, são **caminhoDestinoVendas** para o primeiro conjunto de dados e **caminhoDestinoMarcas** para o segundo, ambos contidos na pasta 'data'. Este fluxo garante a organização e consistência no processo de correção e exportação dos dados.

2. Querys necessárias para análise dos dados

A fim de responder as perguntas propostas pela situação problema é necessário utilizar o SQLite Online e importar os novos dados gerados pelo script JavaScript que fez a correção do banco JSON. Quando importado, será possível fazer consultas utilizando a linguagem de consulta estruturada (SQL).

Entretanto, quando os arquivos JSONs são importados ao SQLite Online, as colunas serão nomeadas como c1, c2, c3... e assim sucessivamente, entretanto, é importante que elas tenham os nomes corretos de acordo com o JSON como “data”, “id_marca_”, “vendas” e etc. Para isso, é necessário que elas sejam inseridas na configuração da tabela. A fim de evitar essa manutenção e possibilitar uma abordagem mais simples pelo(a) avaliador(a) e simplesmente utilizar as consultas para conferir, foram mantidas as colunas de acordo com a importação padrão do SQLite Online.

2.1 Qual marca teve o maior volume de vendas?

Para realizar esta consulta é preciso juntar as duas tabelas substituindo o “id_marca_” da tabela **vendasCorrigidas** pelas marcas que estão presente na tabela **marcasCorrigidas**, após isso é preciso mostrar os dados de acordo com a soma das vendas de cada marca, resultando na seguinte consulta:

“SELECT m.c2 AS nome_marca, SUM(v.c3) AS total_vendas FROM vendasCorrigidas v JOIN marcasCorrigidas m ON v.c2 = m.c1 GROUP BY v.c2, m.c2 ORDER BY total_vendas DESC LIMIT 1;”

2.2 Qual veículo gerou a maior e menor receita?

Para encontrar a maior receita, é preciso fazer a multiplicação da coluna de vendas pela coluna de valor do veículo e depois mostrar os dados de forma decrescente e

limitando somente ao primeiro resultado (é possível realizar esta consulta de maneiras mais performáticas, em um banco grande, seria necessário realizar uma consulta que gastasse menos memória para encontrar a resposta), e para encontrar quem gerou a menor receita, é preciso fazer a mesma consulta, porém colocando em ordem crescente.

Maior: “SELECT c5 AS nome, c4 * c3 AS receita FROM vendasCorrigidas ORDER BY receita DESC LIMIT 1;”

Menor: “SELECT c5 AS nome, c4 * c3 AS receita FROM vendasCorrigidas ORDER BY receita ASC LIMIT 1;”

2.3 Qual a média de vendas do ano por marca?

A lógica da consulta de média de vendas do ano por marca é feita a partir da junção (JOIN) das duas tabelas para descobrir qual veículo pertence a qual marca, depois é calculado a média de vendas por marca utilizando a função AVERAGE do SQL na coluna de vendas, no caso a c3. Deixando a consulta da seguinte forma:

“SELECT m.c2 AS nome_marca, AVG(v.c3) AS media_vendas FROM vendasCorrigidas v JOIN marcasCorrigidas m ON v.c2 = m.c1 GROUP BY v.c2, m.c2;”

2.4 Quais marcas geraram uma receita maior com número menor de vendas?

Esta pergunta representa as marcas que conseguiram fazer menos vendas e mesmo assim atingir uma receita maior, levando a acreditar intuitivamente que se consegue uma maior receita vendendo menos quando os produtos são mais caros, mas isso não necessariamente é verdade, já que os preços podem variar consideravelmente.

A fim de mostrar as marcas que geraram uma receita maior com menos vendas é preciso realizar a seguinte consulta:

“SELECT m.c2 AS nome_marca, SUM(v.c3) AS total_vendas, SUM(v.c4 * v.c3) AS receita FROM vendasCorrigidas v JOIN marcasCorrigidas m ON v.c2 = m.c1 GROUP BY v.c2, m.c2 ORDER BY receita / total_vendas DESC;”

Essa consulta agrupa os valores por marca com a junção das duas tabelas, deixando em evidência a quantidade de vendas e a receita pela quantidade de vendas.

Depois, a fim de mostrar que não necessariamente o topo dessa lista retornada representaria os carros mais caros, foi feita a seguinte consulta:

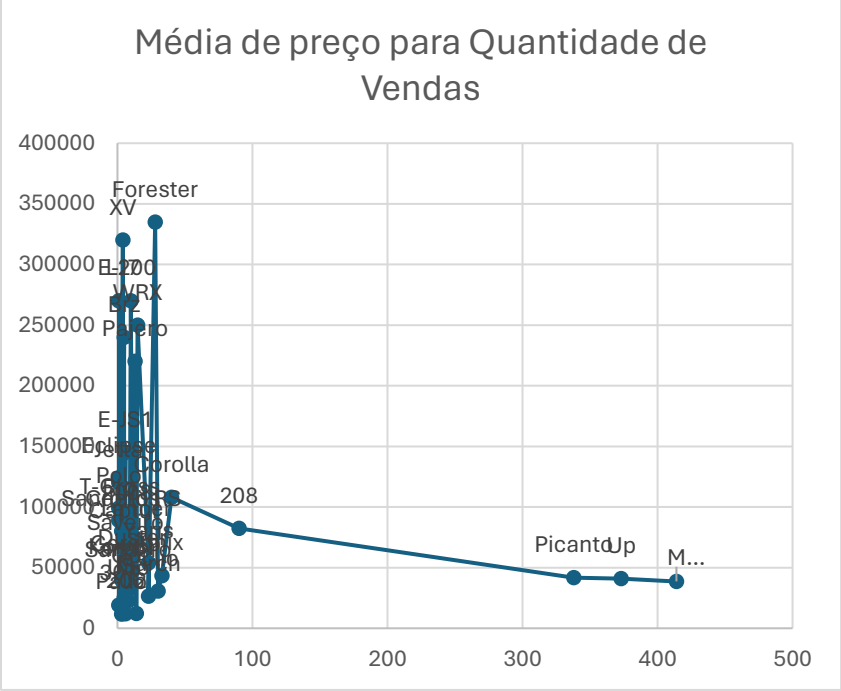
“SELECT m.c2 as Marcas, v.c4 as valor_do_veiculo FROM vendasCorrigidas v JOIN marcasCorrigidas m ON v.c2 = m.c1 order by valor_do_veiculo DESC;”

2.5 Existe alguma relação entre os veículos mais vendidos?

Sim, existe, mas a compreensão da relação entre os veículos mais vendidos é só se torna possível analisando um gráfico de dispersão após a consulta no SQL. A tabela transformada em CSV foi analisada no Excel, a consulta utilizada foi:

“SELECT c5 AS nome, SUM(c3) AS total_vendas, AVG(c4) as valor_medio_do_veiculo FROM vendasCorrigidas GROUP BY c5 ORDER BY total_vendas DESC;”

A relação entre os veículos mais vendidos está justamente em seus valores parecidos entre si, quando colocado em um gráfico de dispersão há de maneira visual a compreensão dessa informação.



R. Girassol, 128 - Pinheiros, São Paulo - (11) 3097 - 0508
R. Dona Alexandrina, 1366 - Vila Monteiro, São Carlos - (16) 3413-1105
Av. Ireno da Silva Venâncio, 199 - 03A - Protestantes, Votorantim - (15) 3321 -1105