Documentação dos Códigos

Javascript

Função lerJson

A primeira função é apresentada abaixo como lerJson. Seu objetivo é ler o caminho recebido como parâmetro, verificar se existe esse arquivo de fato e caso exista o converter seu conteúdo em forma de objeto Javascript. Após a conversão seja realizada com sucesso é chamada a próxima função de tratamento para retornar os dados já tratados.

Caso ocorram erros durante o processo, a função retorna null

```
const fs = require('fs');

function lerJson(caminho) {
    try {
        if (fs.existsSync(caminho)) {
            const dados = JSON.parse(fs.readFileSync(caminho, 'utf-8'));
            console.log('Dados lidos do arquivo JSON:', dados);
            return tratamentoNomes(dados);
        } else {
            console.error('Arquivo JSON não encontrado:', caminho);
            return null;
        }
    } catch (erro) {
        console.error('Erro ao ler o arquivo JSON:', erro.message);
        return null;
    }
}
```

Função tratamentoNomes

Com o arquivo lido e transformado em objeto, devem ser tratados os nomes, para isso utiliza chaves para percorrer cada dado e ajustar seus valores removendo os caracteres especiais como 'ø' e 'æ'. Nisso foi utilizado a função replace e o padrão UNICODE dos carácteres para evitar erros de leitura.

Para que não ocorram erros, antes de realizar a troca é verificado se valor é um objeto não nulo, após ocorre outra verificação para saber se o valor é um nome ou marca para que seja realizado o algoritmo correto.

```
function tratamentoNomes(dados) {
  const chaves = Object.keys(dados);
  chaves.forEach(chave => {
      const valor = dados[chave];
      if (typeof valor === 'object' && valor !== null) {
         // Utilizei do padrão UNICODE para representar ø e æ
         if('nome' in valor){
            valor['nome'] = valor['nome'].replace(/\u00F8/gi, 'o').replace(/\u00E6/gi, 'a');
         }
         else if('marca' in valor){
            valor['marca'] = valor['marca'].replace(/\u00F8/gi, 'o').replace(/\u00E6/gi, 'a');
         }
      }
  });
  return dados;
}
```

Função tratamentoVendas

O tratamento a seguir foi criado com intuito de transformar os valores numéricos que estavam em strings para numbers. Para que ocorra é feita uma verificação em cada objeto se o tipo de atributo está de fato como string, caso esteja, é ajustado para number utilizando a função parseInt.

```
function tratamentoVendas(dados){
  const chaves = Object.keys(dados);

  chaves.forEach(chave => {
      const valor = dados[chave];
      if (typeof valor === 'object' && valor !== null && typeof valor['vendas'] === 'string') {
          valor['vendas'] = parseInt(valor['vendas']);
      }

  });
  return dados;
}
```

Função exportarJson

Esta função recebe um caminho de destino e um objeto de dados. Converte os dados em uma string JSON com identação de 2 espaços e salva essa string em um arquivo no caminho especificado.

Ao final exibe uma mensagem indicando o sucesso ou falha da operação.

```
function exportarJson(caminho, dados){
    try {
        const dadosJson = JSON.stringify(dados, null, 2);
        fs.writeFileSync(caminho, dadosJson, 'utf-8');
        console.log(`Dados salvos no arquivo JSON: ${caminho}`);
    } catch (erro) {
        console.error('Erro ao salvar o arquivo JSON:', erro.message);
    }
}
```

Utilização

Para que seja utilizado o algoritmo de maneira correta, deve-se especificar o local onde os dados JSON estão e onde os dados tratados irão ser salvos. Abaixo está o exemplo utilizado por mim para a realização do projeto.

```
const caminho1 = './Dados/broken_database_1.json';
const caminho2 = './Dados/broken_database_2.json';

// Lê os arquivos e realiza o tratamento
let veiculos = lerJson(caminho1);
let marcas = lerJson(caminho2);

// Realiza o tratamento de vendas
tratamentoVendas(veiculos);

// Exporta os dados tratados
exportarJson('./Dados/fixed_database_1.json', veiculos);
exportarJson('./Dados/fixed_database_2.json', marcas);
```

SQL

Os arquivos foram importados para o site SQL Online, porém o nome das colunas não foram bem importados, visto isso primeira coisa a ser feita foi a renomeação de todas as colunas utilizando da função DDL

```
-- Database 1
ALTER TABLE fixed_database_1
    RENAME COLUMN c1 TO data;
ALTER TABLE fixed_database_1
    RENAME COLUMN c2 TO id_marca;
ALTER TABLE fixed_database_1
    RENAME COLUMN c3 TO vendas;
ALTER TABLE fixed_database_1
    RENAME COLUMN c4 TO valor_veiculo;
ALTER TABLE fixed_database_1
    RENAME COLUMN c5 TO nome;
-- Database 2
ALTER TABLE fixed_database_2
    RENAME COLUMN c1 to id_marca;
ALTER TABLE fixed_database_2
    RENAME column c2 to marca;
```

Unificação das tabelas

Foi criada uma nova tabela para receber os dados buscados por uma query de junção entre a fixed_database_1 e fixed_database_2, utilizando da cláusula RIGHT JOIN com base na coluna id_marca.

```
CREATE TABLE tabela_unificada AS
    SELECT m.id_marca, m.marca, v.data, v.vendas, v.valor_veiculo, v.nome
    FROM fixed_database_2 AS m
    RIGHT JOIN fixed_database_1 AS v ON v.id_marca = m.id_marca;
```

Inserção da Coluna ‘receita’

Para facilitar a criação do relatório, foi criada uma coluna receita, sendo ela representada pelo valor do veiculo vezes a quantidade de veículos vendidos, mostrando a receita total por aquele automóvel.

```
ALTER TABLE tabela_unificada
ADD COLUMN receita REAL;

UPDATE tabela_unificada
SET receita = valor_do_veiculo * vendas;
```

Maior volume de vendas

As querys a seguir foram utilizadas para ajudar na criação do relatório, a primeira foi o maior volume de vendas por marca, visto isso o código busca a soma de todos os veículos das mesmas marcas e retorna os resultados das 5 maiores.

```
SELECT marca, SUM(vendas) AS total_vendas
    FROM tabela_unificada
    GROUP BY marca
  ORDER BY total_vendas DESC
    LIMIT 5;
```

marca	total_vendas
Fiat	433
Volkswagen	395
Kia	345
Peugeot	104
Toyota	63

Veículos com Maior e Menor Receita

Visto que a coluna de receitas foi criada, as funções de buscas foram simples utilizando uma concatenação do nome com a marca para facilitar a leitura. Os resultados foram agrupados por Veículos (pois existiam valores diferentes para veículos iguais) e retornam os 10 maiores e menores.

```
SELECT CONCAT(marca,' ',nome) as modelo, SUM(receita) AS receita_total
    FROM tabela_unificada
    GROUP BY nome
    ORDER BY receita_total DESC
    LIMIT 10; -- Maior

SELECT CONCAT(marca,' ',nome) as modelo, SUM(receita) AS receita_total
    FROM tabela_unificada
    GROUP BY nome
    ORDER BY receita_total ASC
    LIMIT 10;-- Menor
```

modelo	receita_total
Fiat Mobi	14747000
Volkswagen Up	14368000
Kia Picanto	13243000
Subaru Forester	9800000
Peugeot 208	7425000
Toyota Corolla	4368000
Subaru WRX	3750000
Mitsubishi Pajero	2680000
Mitsubishi L200	2580000
Chevrolet onix	1418400

modelo	receita_total
Peugeot 307	19000
Fiat Palio	35000
Peugeot 206	72000
Kia Rio	89000
Volkswagen Polo	99000
Volkswagen Jetta	120000
Mitsubishi Eclipse	124000
JaC Motors J5	125000
Volkswagen Kombi	125000
Fiat argo	154000

Média de vendas do ano por marca

Como essa consulta envolvia datas e elas foram enviadas como string, foi feita um tratamento para pegar os 4 primeiros carácteres da string a qual resultariam o ano, em seguida o arredondamento da média de vendas por marca.

```
SELECT marca, SUBSTR(data, 1, 4) AS ano, ROUND(AVG(vendas),2) AS media_vendas_por_marca_e_ano
    FROM tabela_unificada
    GROUP BY marca, ano;
```

marca	ano	media_vendas_por_marca_e_ano
Chevrolet	2022	3.67
Fiat	2022	19.68
JaC Motors	2022	2.17
Kia	2022	23
Mitsubishi	2022	3.8
Nissan	2022	3.29
Peugeot	2022	11.56
Renault	2022	4.75
Subaru	2022	7.43
Toyota	2022	7.88
Volkswagen	2022	18.81

Maior Receita com Menor Número de Vendas

Para essa consulta se buscou a marca, total de vendas e total de receita de cada uma. Após isso ela foi ordenada pelo total de receita dividido pelo total de vendas, para fazer a comparação de quem conseguiu uma maior receita com o menor numero de vendas.

```
SELECT marca, SUM(vendas) AS total_de_vendas, SUM(receita) AS total_de_receita
    FROM tabela_unificada
    GROUP BY marca
    ORDER BY total_de_receita / total_de_vendas DESC;
```

marca	total_de_vendas	total_de_receita
Subaru	52	16030000
Mitsubishi	38	6374000
Toyota	63	5493000
Peugeot	104	8086000
JaC Motors	26	1430000
Chevrolet	33	1418400
Renault	57	2403000
Kia	345	13586000
Volkswagen	395	15540000
Fiat	433	15447000
Nissan	23	601000

Relação entre Veículos mais vendidos

Para essa pesquisa foi buscado os modelos mais vendidos juntamente com o total de vendas e a média de preço para cada modelo.

```
SELECT CONCAT(marca,' ',nome) AS modelo, SUM(vendas) AS total_de_vendas,
       ROUND(AVG(valor_do_veiculo),2) as media_preco
    FROM tabela_unificada
  GROUP BY nome
    ORDER BY total_de_vendas DESC;
```

modelo	total_de_vendas	media_preco
Fiat Mobi	414	38666.67
Volkswagen Up	373	41000
Kia Picanto	338	41750
Peugeot 208	90	82400
Toyota Corolla	40	108000

Python

Para a criação do Heatmap e teste de correlação de Pearson, foi utilizado a linguagem python.

O código a seguir importa as bibliotecas utilizadas e lê e transforma o arquivo CSV em um dataframe ‘df’ da biblioteca pandas.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

caminho_arquivo_csv = './tabela_unificada.csv'
df = pd.read_csv(caminho_arquivo_csv, delimiter=";")
df.head()
```

Em seguida é feita a matriz de correlação e transformada em um heatmap para melhor visualização, para isso utiliza-se das bibliotecas matplotlib e seaborn, resultando no gráfico apresentado abaixo do código.

```
matriz_correlacao = df[['vendas', 'valor_do_veiculo']].corr()

sns.set(style='whitegrid')
plt.figure(figsize=(8, 6))
sns.heatmap(matriz_correlacao, annot=True, cmap='Blues', fmt='.2f')
plt.title('Heatmap de Correlação entre Vendas e Valor do Veículo')
plt.show()
```