Projeto resolução da Rocky

1. Início do código

Primeiro, foi necessário preparar o código e o Visual Studio Code para utilizar JSON e Node. O Node foi baixado e iniciado dentro do projeto.

```
JS resolucao.js > ...
1   const { error } = require("console");
2   const fs = require("fs");
3   const database1 = require("./broken_database_1.json");
4   const database2 = require("./broken_database_2.json");
5
```

Para utilizar o JSON no código, foram utilizados os comandos 'require' para chamar o banco de dados e 'fs' para executar os comandos.

2. Funções

Database1Corrigido() e Database2Corrigido()

Essas funções são as principais do código e são responsáveis por gerar novos bancos de dados - os arquivos 'saida_database1.json' e 'saida_database2.json' - e corrigir os erros contidos nos antigos bancos de dados. Elas compilam as outras funções: '*CorrigeValor()*', '*CorrigeCaracter()*', '*LeituraDatabase1()*' e '*LeituraDatabase2()*'.

```
56  function Database1Corrigido() {
57      // .map para que a acao seja realizada em todos os elementos e cria o novo database
58      const corrigido = LeituraDatabase1().map(function(dados) {
59          return {
60              data: dados.data,
61              id_marca: dados.id_marca_,
62              vendas: CorrigeValor(dados.vendas),
63              valor_do_veiculo: dados.valor_do_veiculo,
64              nome: CorrigeCatacter(dados.nome),
65          };
66      });
67
68      // no return usamos fs.writeFileSync para criar um novo arquivo que sera o novo database
69      return fs.writeFileSync(
70          //nome, localidade e tipo do arquivo a ser criado
71          "./saida_database1.json",
72          //converter os dados que foram corrigidos para o formato JSON
73          JSON.stringify(corrigido),
74          // definindo o callback
75          (error, result) => {
76              if (error) {
77                  console.error(error);
78                  return;
79              }
80
81              console.log(result);
82          }
83      );
84  }
```

```
// Essa funcao vai executar as anteriores no segundo database.
//</summary>
function Database2Corrigido() {
    // .map para que a acao seja realizada em todos os elementos e cria o novo database
    const corrigido = LeituraDatabase2().map(function(dados) {
        return {
            id_marca: dados.id_marca,
            marca: CorrigeCatacter(dados.marca),
        };
    });

    // no return usamos fs.writeFileSync para criar um novo arquivo que sera o novo database
    return fs.writeFileSync(
        //nome, localidade e tipo do arquivo a ser criado
        "./saida_database2.json",
        //converter os dados que foram corrigidos para o formato JSON
        JSON.stringify(corrigido),
        // definindo o callback
        (error, result) => {
            if (error) {
                console.error(error);
                return;
            }

            console.log(result);
        }
    );
}

```

As funções agem da seguinte maneira, ao ser criada a variável constante *corrigido*, é alocada nela a função *LeituraDatabase1()* ou *LeituraDatabase2()*, dependendo de qual for, juntamente com o método *map*, que em seus parâmetros, é criada uma função com o parâmetro *dados*, a qual retorna os valores dos objetos, os quais variam entre as funções *LeituraDatabase1()* e *LeituraDatabase2()*, e utiliza as outras funções para corrigir os dados. Por fim, utilizando o comando *fs.writeFileSync*, as funções retornam novos arquivos com os nomes 'saida_database1.json' e 'saida_database2.json'. Os dados alterados que estavam alocados na variável constante *corrigido* são convertidos para o formato JSON pelo comando *JSON.stringify().* Logo em seguida, é usado um callback para casos de erros e, por último, é exibido o resultado por meio do comando console.log().

LeituraDatabase1() e LeituraDatabase2()

Essas são as primeiras funções do código e são importantes porque permitem o uso dos dados contidos no 'Broken_database1' e no 'Broken_database2'.

```
//<summary>
// a function leitura recebe o primeiro banco de dados em JSON e a retorna em um formato para ser usado.
//</summary>
function LeituraDatabase1() {
    //fs.readFileSync le o banco de dados em formato JSON, de nome broken-database.
    // e armazena na variavel recebe_JSON.
    const recebe_JSON = fs.readFileSync("./broken_database_1.json");
    // atraves do JSON.parse ele converte para o formato javascript.
    // e armazena na variavel dados.
    const dados = JSON.parse(recebe_JSON);
    return dados;
}
//<summary>
// a function leitura recebe o segundo banco de dados em JSON e a retorna em um formato para ser usado.
//</summary>
function LeituraDatabase2() {
    //fs.readFileSync le o banco de dados em formato JSON, de nome broken-database.
    // e armazena na variavel recebe_JSON.
    const recebe_JSON = fs.readFileSync("./broken_database_2.json");
    // atraves do JSON.parse ele converte para o formato javascript.
    // e armazena na variavel dados.
    const dados = JSON.parse(recebe_JSON);
    return dados;
}
```

Para que possamos utilizar os dados dos bancos, essas funções convertem os dados do formato JSON para Javascript.

CorrigeCatacter()

Um dos maiores problemas nos bancos de dados 'broken_database1' e 'broken_database2' é o fato de que certos caracteres nos campos 'nome' e 'marca' estão incorretos.

```
//<summary>
/// a function CorrigeName utiliza o replace para alterar os caracteres que apresentavam erro.
//</summary>
function CorrigeCatacter(nome) {
    // define quais caracteres serao mudados e para quais.
    const correcao = { æ: "a", ø: "o" };
    // gi: g = global i=ignore
    return nome.replace(/æ|ø/gi, function(retorno) {
        return correcao[retorno];
    });
}

```

Para resolver esse problema, foi necessário criar uma variável constante com os caracteres defeituosos e os valores que serão substituídos por eles. Em seguida, é utilizado o método *replace* com o parâmetro *gi*, que permite a percorrer dos caracteres em todo o array.

CorrigeValor()

Outro erro presente no 'broken_database1' é que alguns dos valores do campo 'vendas' estão como caracteres, em vez de valores numéricos.

```
46   //<summary>
47   // retorna o preco no formato correto
48   //</summary>
49   function CorrigeValor(valor_do_veiculo) {
50       // usa o pasrsefloat para transformar do formato string para float.
51       return parseFloat(valor_do_veiculo);
52   }
```

No 'return', é utilizado o método *parseFloat()* para converter os valores do tipo 'string' para o tipo 'float'.

3. SQL

No banco de dados MS SQL, uma nova tabela foi criada com o comando "CREATE TABLE". Em seguida, foi utilizado um 'INSERT INTO' em conjunto com um "FULL JOIN" para selecionar e enviar dados das tabelas saída_database1 e saída_database2 para a nova tabela criada, denominada Relatorio_Geral.

```
 7 CREATE TABLE Relatorio_Geral(
 8   data VARCHAR(50),
 9   id_marca int,
10   vendas int,
11   valor_do_veiculo int,
12   nome VARCHAR(50),
13   marca VARCHAR(50)
14   )
15
```

```
16   INSERT INTO Relatorio_Geral( data, id_marca, vendas, valor_do_veiculo, nome, marca)
17   SELECT
18     A.data, A.id_marca, A.vendas, A.valor_do_veiculo, A.nome, B.marca
19 FROM
20     saida_database1 A FULL JOIN saida_database2 B
21         ON A.id_marca = B.id_marca;
```

4. Referencias

Sites

Os sites usados para pesquisa foram:

https://www.w3schools.com