Recuperação e Relatório de Vendas

O objetivo principal deste projeto é recuperar os dados originais de um banco de dados e criar um relatório de vendas. A linguagem de programação utilizada é JavaScript (JS) para o desenvolvimento da aplicação SQL para interagir com o banco de dados.

O projeto é composto por duas partes principais: a primeira é responsável pela recuperação dos dados originais do banco de dados corrompido, utilizando a linguagem SQL para acessar o banco de dados e a linguagem JavaScript para processar e armazenar os dados recuperados, utilizando a biblioteca DataTable para exibir os dados em uma tabela a biblioteca FileSaver.js para permitir o download do relatório de vendas em formato JSON. A segunda parte do projeto é responsável por criar um relatório de vendas a partir dos dados recuperados, utilizando a biblioteca Chart.js para exibir os gráficos para realizar uma análise de dados.

O código-fonte do primeiro projeto contém as seguintes funções:

## Variáveis:

* **vendas**: uma matriz vazia que será preenchida com os dados da primeira base de dados.
* **modelos**: uma matriz vazia que será preenchida com os dados da segunda base de dados.
* **rows**: uma matriz vazia que será preenchida com os dados formatados das vendas.

## getJson()

Descrição: A função getJson() é responsável por fazer uma chamada AJAX para uma URL especificada e retornar os dados JSON como uma promisse.

Retorno: A função retorna uma promisse que resolve com os dados JSON retornados pela URL ou rejeita com uma mensagem de erro caso haja algum problema ao carregar a URL.

Parâmetros: A função recebe um parâmetro url que é a URL que deve ser chamada.

### Comportamento:

1. Dentro da função, um objeto XMLHttpRequest é criado e usado para fazer a chamada AJAX para a URL fornecida.
2. Quando a resposta AJAX é recebida com sucesso, a promessa é resolvida com os dados JSON retornados pela URL.
3. Se houver algum erro ao carregar a URL, a promessa é rejeitada com uma mensagem de erro.
4. A função não modifica os arrays vendas, modelos e rows que foram definidos anteriormente no código.

## loadData()

Descrição: Carrega dados de dois arquivos JSON diferentes e executa a função formatar() após o carregamento bem-sucedido dos dados.

Retorno: Não há retorno explícito.

Parâmetros: Nenhum parâmetro.

### Comportamento:

1. Usa a função getJson() para carregar os dados do arquivo JSON ../data/broken\_database\_1.json em vendas.
2. Usa a função slice() para fazer uma cópia dos dados carregados em vendas.
3. Usa a função getJson() para carregar os dados do arquivo JSON ../data/broken\_database\_2.json em modelos.
4. Usa a função slice() para fazer uma cópia dos dados carregados em modelos.
5. Chama a função formatar() para processar os dados carregados, após o carregamento bem-sucedido dos dados.
6. Se ocorrer algum erro durante o carregamento dos dados, o bloco catch captura o erro e o imprime no console usando a função console.error().

## Formatar()

### Descrição: a função recebe os dados, corrige e gera tags HTML com esses dados.

Parâmetros: A função não recebe nenhum parâmetro.

Retorno: A função não retorna nenhum valor.

### Comportamento:

1. A função itera sobre o array vendas para criar uma linha de tabela para cada dado de venda, formatando-os adequadamente.
2. Se a venda tiver uma marca correspondente nos modelos, a função substitui caracteres especiais no nome da marca para corrigi-los.
3. A função adiciona cada linha à tabela na página HTML.
4. A função também cria um JSON com os dados formatados e o armazena em um array rows.
5. Finalmente, a função inicializa o DataTable.

## download()

Descrição: é responsável por realizar o download de um arquivo JSON contendo dados corrigidos.

Parâmetros: A função não recebe nenhum parâmetro.

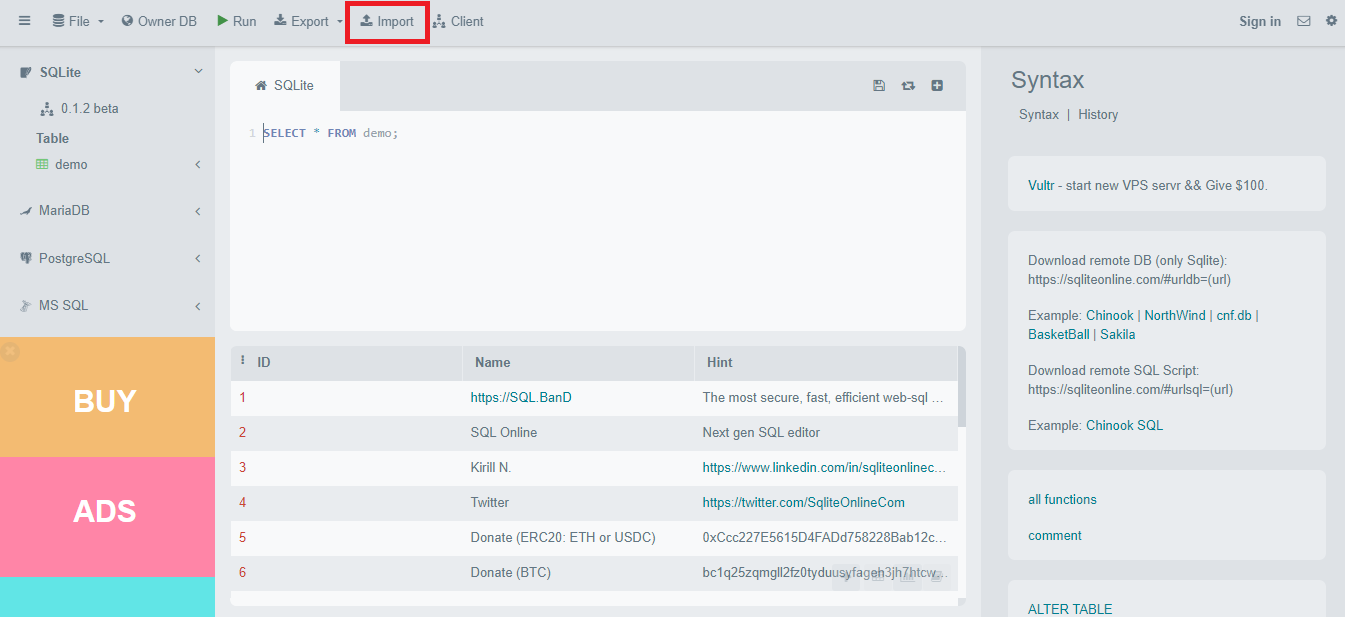
Retorno: A função não retorna nenhum valor.

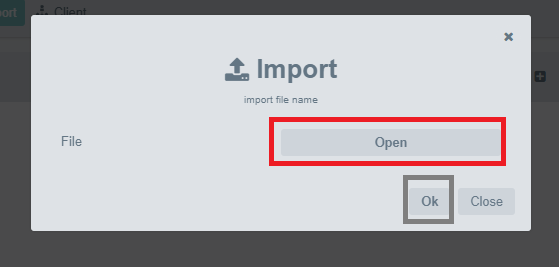
### Comportamento:

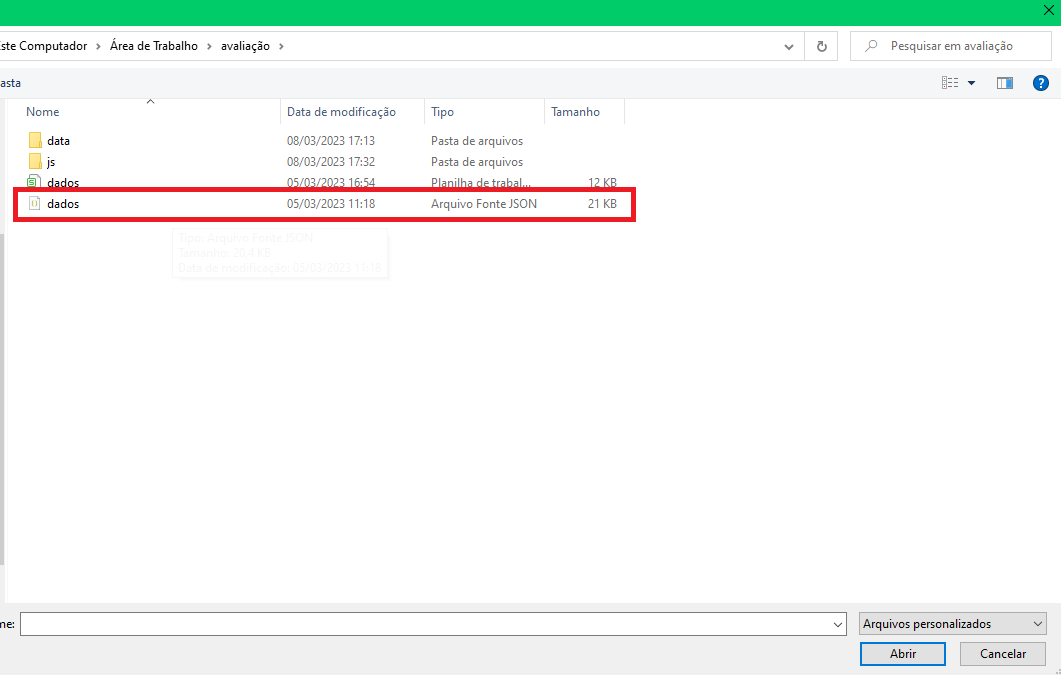
1. Converte o objeto JSON em uma string utilizando o método JSON.stringify().
2. Cria um objeto do tipo Blob com a string JSON e define o tipo de arquivo como application/json;charset=utf-8.
3. Utiliza a biblioteca FileSaver.js para fazer o download do arquivo.
4. O arquivo baixado terá o nome "dadosCorrigidos.json".
5. Essa função não possui retorno e não recebe nenhum parâmetro. Ela é chamada a partir de um evento de clique em um botão, para permitir que o usuário faça o download do arquivo JSON gerado.

Criando Banco de dados

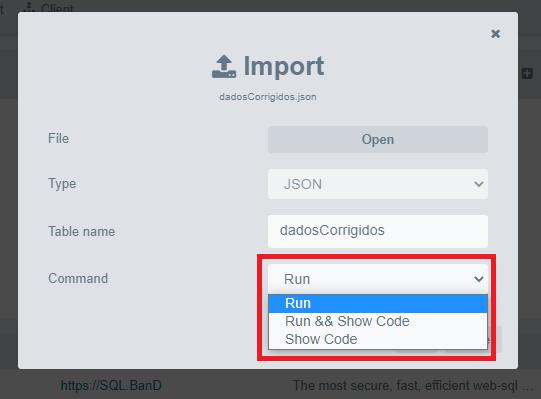
Após de fazer a correção dos dados, para criarmos um arquivo .csv, foi utilizado a plataforma do site <https://sqliteonline.com/>, nela foram feitos os passos abaixo:

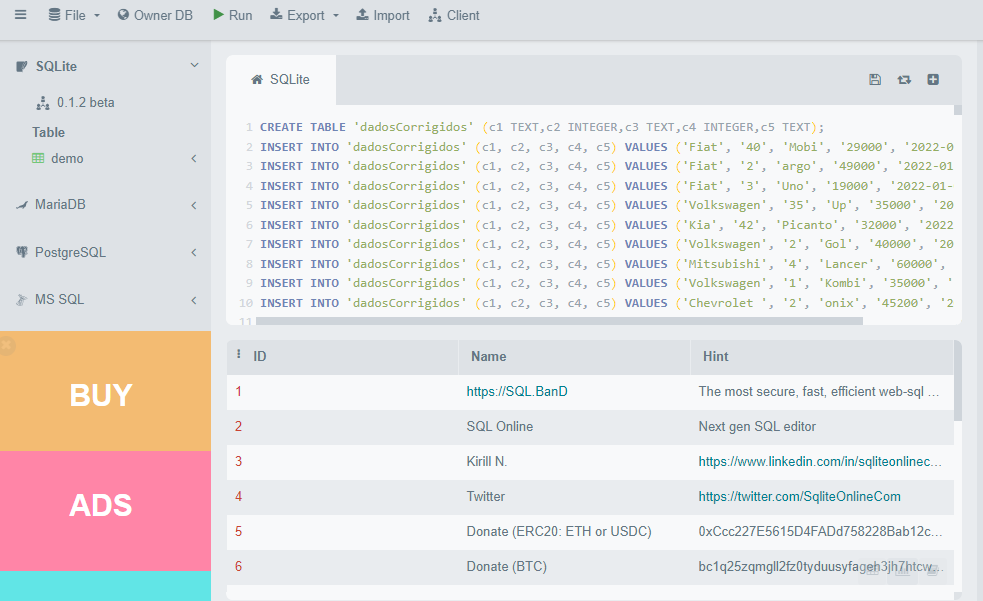
1. A imagem abaixo é a interface do site, de início, na área demarcada em “import”, clicamos para inserir o arquivo “dadosCorrigidos.json” para ele gerar um banco de dados em SQLite;
2. Selecionamos “open” para escolhermos nosso arquivo. JSON e clicamos em “ok”;



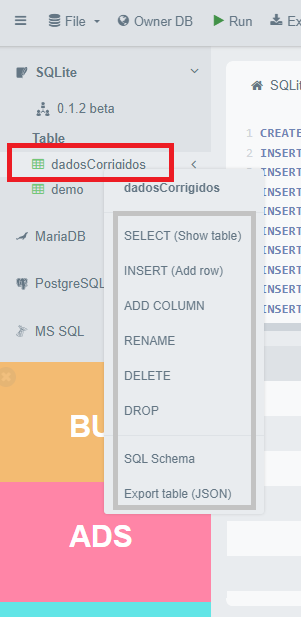


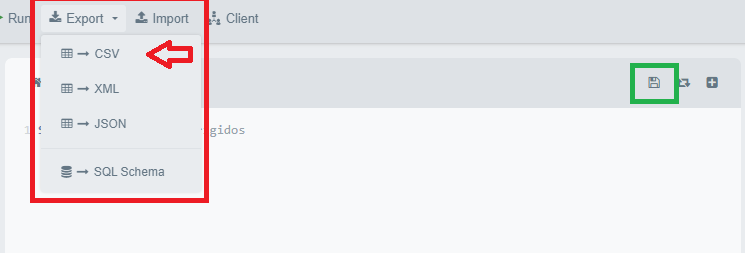
1. Ele vai identificar o tipo de arquivo, em “comand” você tem opções de como o programa vai rodar seus dados, em “Run”, ele vai apenas executa seu código em SQLite; em “Run && Show Code”, ele executa e mostra o que foi executado; e em “Show Code” ele apenas mostra o código para executar;

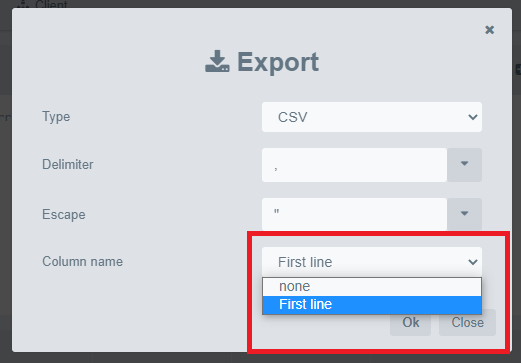




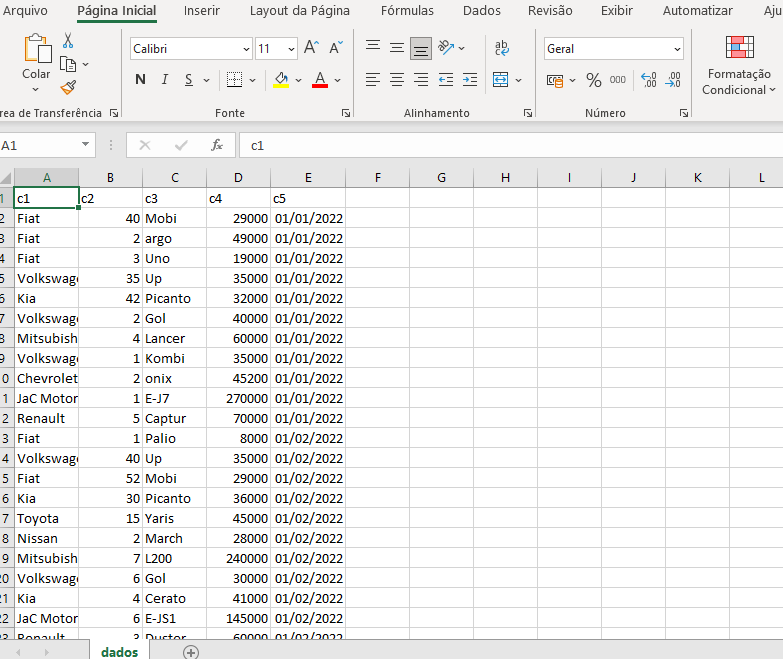
1. No lado esquerdo da página, podemos selecionar com o botão direto do mouse a tabela criada, nela podemos executar comandos básicos de SQL, além de exportá-la para outras extensões;



1. Para criarmos um arquivo .csv desses dados, selecionamos “export” e escolhemos qual extensão, após isso em “comand name”, selecionamos “Fisrt name” para criar inserir os nomes nas colunas; se selecionarmos o icon de salvar demarcada em verde, ela salva os SQL que está na tela;



1. Após exportarmos o arquivo ele terá essa aparência e assim já podemos utilizá-lo para fazer a análise;



1. Interface gráfica do usuário, Aplicativo, Tabela, Excel

   Descrição gerada automaticamente com confiança médiapara termos o arquivo em SQL selecionamos a tela de comando de INSERT e exportamos;

O código-fonte do segundo projeto contém as seguintes funções:

## getRandomColor()

A função getRandomColor() gera uma cor aleatória em formato RGB, com valores de vermelho, verde e azul (RGB) variando de forma aleatória.

### Parâmetros: A função não recebe parâmetros.

### Retorno: A função retorna uma string com a cor em formato RGB, no seguinte formato: "rgb (R, G, B, A)", onde:

* R: valor de vermelho (entre 0 e 255).
* G: valor de verde (entre 0 e 150).
* B: valor de azul (entre 0 e 20).
* A: valor de opacidade (fixo em 0.7).

### Descrição:

A função getRandomColor() pode ser utilizada para gerar cores diferentes para cada elemento de um gráfico, por exemplo. A opacidade fixa em 0.7 permite que as cores geradas sejam mais suaves e menos "agressivas" visualmente.

## maiorVenda()

### Descrição: Essa função é responsável por ler um arquivo CSV com dados de vendas por marca, calcular as vendas totais por marca e plotar um gráfico de barras com as vendas por marca, ordenando as barras de acordo com a marca que teve mais vendas.

### Parâmetros: A função não recebe parâmetros explícitos.

### Retorno: A função não possui um retorno explícito.

### Comportamento:

1. A função usa a função fetch para buscar o arquivo CSV com os dados de vendas por marca.
2. Em seguida, usa a biblioteca PapaParse para analisar o conteúdo do arquivo CSV e converter para um array de objetos.
3. Ordena o array de objetos pela quantidade de vendas em ordem decrescente.
4. Calcula a quantidade total de vendas para cada marca.
5. Plota um gráfico de barras usando a biblioteca Chart.js, mostrando a quantidade total de vendas para cada marca, com as barras ordenadas em ordem decrescente de vendas.
6. Retorna um erro no console caso ocorra algum erro durante a execução da função.
7. Além disso, a função usa a função getRandomColor para definir uma cor aleatória para cada barra do gráfico.

## maiorReceita()

Descrição: A função maiorReceita() é responsável por buscar os dados em um arquivo CSV e gerar um gráfico de barras com as maiores receitas por veículo.

### Parâmetros: A função não recebe nenhum parâmetro.

### Retorno: A função não retorna nada.

### Comportamento:

1. A função faz uma requisição para o arquivo dados.csv usando a API fetch().
2. Caso a requisição falhe, é lançado um erro e exibido um alerta ao usuário.
3. Se a requisição for bem-sucedida, os dados do arquivo CSV são convertidos em um objeto JavaScript usando a biblioteca Papa.parse().
4. Em seguida, a função percorre os dados para extrair as receitas de cada veículo e armazená-las em um objeto veiculosData.
5. Os dados de receita são ordenados em ordem decrescente.
6. Por fim, é gerado um gráfico de barras usando a biblioteca Chart.js, exibindo as maiores receitas por veículo.

## mediaMarca()

### Descrição: A função mediaMarca() é responsável por realizar o cálculo da média de vendas por marca e ano a partir dos dados presentes em um arquivo CSV. Em seguida, cria um gráfico de barras utilizando a biblioteca Chart.js para exibir as médias de vendas por marca e ano.

### Parâmetros: A função não recebe parâmetros.

### Retorno: A função não possui retorno.

### Comportamento:

1. Realiza uma requisição HTTP para o arquivo dados.csv utilizando a função fetch.
2. Converte a resposta da requisição em texto utilizando o método text.
3. Faz o parsing do conteúdo do arquivo CSV utilizando a biblioteca PapaParse.
4. Calcula a média de vendas por marca e ano e armazena em um objeto vendasPorMarcaEAno.
5. Cria um gráfico de barras utilizando a biblioteca Chart.js para exibir as médias de vendas por marca e ano.

## maiorReceitaMenorVenda()

carrega dados de um arquivo CSV, calcula a receita e vendas por marca e filtra as marcas com receita maior e vendas menores que 100 e que receita é maior que 10000. Em seguida, cria um gráfico de barras mostrando a receita dessas marcas filtradas.

Parâmetros: A função não recebe nenhum parâmetro.

Retorno: A função não retorna nenhum valor.

### Comportamento:

1. A função utiliza a função fetch para carregar os dados de um arquivo CSV. Se houver um erro ao carregar os dados, é lançada uma exceção com a mensagem "Erro ao carregar os dados".
2. Depois de carregar os dados com sucesso, a função utiliza a biblioteca Papa Parse para analisar e converter os dados do CSV em um objeto JavaScript.
3. Em seguida, a função itera sobre os dados para armazenar as vendas e receitas por marca em um objeto chamado dados.
4. A função filtra as marcas com vendas menores que 100 e receitas maiores que 10000 usando o método filter() do objeto dados.
5. A função cria um array de labels com o nome das marcas filtradas e suas respectivas vendas e receitas.
6. A função cria um objeto com as informações necessárias para o gráfico de barras, que inclui os labels e dados das marcas filtradas.
7. A função cria um objeto de configuração para o gráfico, definindo o tipo de gráfico como barra e as escalas para os eixos X e Y.
8. A função renderiza o gráfico de barras na tela com os dados e configurações especificados.

## relacaoMaiorVenda()

### Descrição: A função relacaoMaiorVenda() é uma função JavaScript que busca um arquivo CSV contendo dados de vendas de carros por marca, e gera um gráfico de barras mostrando a relação de vendas por marca. A função usa a biblioteca Papa Parse para analisar o arquivo CSV e a biblioteca Chart.js para gerar o gráfico.

### Parâmetros: A função não recebe nenhum parâmetro.

### Retorno: A função não retorna nenhum valor diretamente, mas gera um gráfico de barras.

### Comportamento:

1. Busca o arquivo CSV usando a função fetch().
2. Analisa o arquivo CSV usando a biblioteca Papa Parse e agrupa as vendas por marca em um objeto.
3. Ordena as marcas em ordem decrescente de vendas e extrai os nomes e valores das marcas em duas matrizes distintas.
4. Cria uma configuração de gráfico usando a biblioteca Chart.js.
5. Cria um gráfico de barras com a configuração especificada.