DOCUMENTAÇÃO

Teste prático - Projeto Classificatório Rocky

Objetivo do algoritmo:

A partir de um banco de dados NoSQL corrompido, “broken_database_1.json” e “broken_database_2.json”, espera-se que possamos recuperar os dados perdidos, corrigindo o nome, preço e quantidade dos produtos.

Funcionalidades:

Para realizar essa tarefa, o código implementa quatro funções principais: ‘consertaNomes’, ‘consertaVendas’, ‘consertaData’ e ‘exportaArquivo’.

A função ‘consertaNomes’ recebe um parâmetro ‘data’, que pode ser uma string ou um objeto, e tem como objetivo corrigir os nomes de marcas e veículo que contenhma os caracteres “æ” e “ø”. Para isso, a função utiliza o método ‘replace()’ para substituir esses caracteres por “a” e “o”, respectivamente. Se o parâmetro ‘data’ for um objeto, a função percorre todas as propriedades do objeto e chama recursivamente a função ‘consertaNomes’ para corrigir os valores das propriedades que forem strings.

Já a função ‘consertaVendas’ recebe um parâmetro ‘data’, que também pode ser uma string ou um objeto, e tem como objetivo corrigit o tipo da propriedade “vendas”. Se o Parâmetro ‘data’ for um objeto, a função percorre todas as propriedades do objeto e chama recursivamente a função ‘consertavendas’ para corrigir as propriedades que não forem do tipo ‘number’. Caso a propriedade seja “vendas”, a função a converte para o tipo ‘number’.

A função ‘consertaData’ é responsável por ler um arquivo JSON, aplicar as correções implementadas pelas funções ‘consertaNomes’ e ‘consertaVendas’ nos dados contidos no arquivo lido e retornar os dados corrigidos. Para isso, a função utiliza a biblioteca ‘fs’ do Node.js para ler o arquivo JSON, faz a chamada das funções ‘consertaNomes’ e ‘consertaVendas’ passando os dados lidos como parâmetro e retorna o resultado da correção.

Por fim, a função ‘exportarArquivo’ recebe como parâmetros o caminho do arquivo a ser exportado e os dados a serem escritos no arquivo. A função converte os dados para o formato JSON utilizando a função ‘JSON.stringify’ e escreve o arquivo utilizando a biblioteca ‘fs’ do Node.js.

O código principal do program executa as funções ‘consertaData’ e ‘exportarArquivo’ para cada um dos arquivos de entrada, gerando dois arquivos de saída com os dados corrigidos.

É importante mencionar que as funções ‘consertaNomes’ e ‘consertaVendas’ estão implementadas de forma recursiva para percorrer objetos anunhados. No entanto, essas funções assumem que as propriedades dos objetos são sempre strings ou objetos. Caso haja outras propriedades com outros tipos de dados, as funções podem não funcionar corretamente.

Tratamentos de erros:

O código não possui um tratamento explícito de erros.

Objetivo do comando SQL:

Este documento descreve o conjunto de comandos SQL utilizados para criar e manipular a tabela “vendas_carros” e responder a algumas perguntas sobre os dados contidos nos arquivos “fixed_database_1.json” e “fixed_database_2.json”, que foram criados pelo código JavaScript citado acima e exportados para o site https://sqliteonline.com/.

CREATE TABLE

O comando “CREATE TABLE” é utilizado para criar uma nova tabela. No caso deste projeto, foi criada a tabela “vendas_carros” com as seguintes colunas e tipos de dados:

- id INTEGER PRIMARY KEY AUTOINCREMENT
- data DATE
- modelo INTEGER
- quantidade INTEGER
- preco REAL
- nome TEXT
- fabricante TEXT

A coluna “id” é a chave primária da tabela e é gerada automaticamente. As outras colunas armazenam informações sobre as vendas de carros, como a data da venda, o modelo do carro, a quantidade vendida, o preço unitário, o nome do veículo e o nome do fabricante.

INSERT INTO

O comando “INSERT INTO” é utilizado para inserir dados em uma tabela. No caso deste projeto, os dados foram inseridos na tabela “vendas_carros” a partir do arquivo “fixed_database_1.json”. As colunas que receberam os dados foram:

- data
- modelo
- quantidade
- preco

- nome

O comando utilizado foi o seguinte:

```
INSERT INTO vendas_carros (data, modelo, quantidade, preco, nome)
SELECT c1, c2, c3, c4, c5
FROM fixed_database_1;
```

UPDATE

O comando “UPDATE” é utilizado para atualizar os valores de uma ou mais colunas de uma tabela. No caso deste projeto, foi utilizado para preencher a coluna “fabricante” da tabela “vendas_carros” a partir dos dados contidos no arquivo “fixed_database_2.json”. O comando utilizado foi o seguinte:

```
UPDATE vendas_carros
SET fabricante = (
   SELECT c2
   FROM fixed_database_2
   WHERE vendas_carros.modelo = fixed_database_2.c1
);
```

SELECT

O comando “SELECT” foi utilizado para responder cinco perguntas sobre os dados da tabela “vendas_carros”.

Pergunta 1 – Qual marca teve o maior volume de vendas?

Para responder a esta pergunta, foi utilizado o seguinte comando:

```
SELECT fabricante, nome, SUM(quantidade) as total_vendas
FROM vendas_carros
GROUP BY fabricante
ORDER BY total_vendas DESC;
```

Este comando seleciona as colinas “fabricante”, “nome” e a soma das quantidades vendidas, agrupando por fabricante e ordenando pela soma das quantidades vendidas em ordem decrescente.

Pergunta 2 _ Qual veículo gerou a maior e menor receita?

Para responder a esta pergunta, foram utilizados dois comandos “SELECT”:

```
SELECT nome, preco, SUM(quantidade) as total_quantidades, preco*
SUM(quantidade) as receita

FROM vendas_carros

group by nome

ORDER BY receita DESC

LIMIT 5;
```

Este comando seleciona as colunas “nome”, “preco”, a soma das quantidades vendidas e a receita total gerada por veículo, agrupando por nome do veículo e ordenando em ordem decrescente de receita. Foi limitado a exibição para que aparecesse apenas cinco veículos com maior receita.

O segundo comando “SELECT” utilizado para responder a pergunta foi:

```
SELECT nome, preco, SUM(quantidade) as total_quantidades,
preco*SUM(quantidade) as receita

FROM vendas_carros

GROUP BY nome

ORDER BY receita

LIMIT 5;
```

Este comando seleciona as mesmas colunas e agrupa por nome do veículo, ordenando em ordem crescente de receita e limitando a exibição a apenas cinco resultados.

Pergunta 3 – Qual a média de vendas do ano por marca?

Para responder a esta pergunta, foi utilizado o seguinte comando:

```
SELECT fabricante, ROUND(AVG(quantidade), 2) as media_por_ano

FROM vendas_carros

GROUP BY fabricante

ORDER BY media_por_ano DESC;
```

Este comando seleciona a coluna “modelo” e a soma das quantidades vendidas, agrupando por modelo e ordenando em ordem decrescente de quantidades vendidas, limitando a exibição a apenas um resultado.

Pergunta 4 – Quais marcas geraram uma receita maior com número menor de vendas?

Para responder a esta pergunta, foi utilizado o seguinte comando:

```
SELECT fabricant, preco * SUM(quantidade) AS receita,
SUM(quantidade) AS total_quantidades

FROM vendas_carros

GROUP BY fabricante

HAVING SUM(preco*quantidade) > 100000 AND SUM(quantidade) <
100;
```

Este comando seleciona a coluna “frabricante” e a receita gerada pelas vendas de cada marca, bem como o total de quantidades vendidas. Os resultados são agrupados por fabricante e filtrados pelo critério de que a receita total deve ser maior que 100.000 e o total de quantidades vendidas deve ser menor que 100.

Pergunta 5 – Existe alguma relação entre os veículos mais vendidos?

Para responder a esta pergunta, foi utilizado o seguinte comando:

```
SELECT nome, preco, SUM(quantidade) AS frequência

FROM vendas_carros

GROUP BY nome

HAVING COUNT(*) > 1

ORDER BY frequência DESC;
```

Este comando seleciona a coluna “nome”, o preço e a frequência das vendas de cada veículo. Os resultados são agrupados por nome do veículo e filtrados pelo critério de que o veículo deve ter sido vendido mais de uma vez. Os resultados são ordenados em ordem decrescente de frequência.