

Sendo você um experiente Projetista de Sistemas de TI, com experiência em programação, principalmente javascript (desde 1980), com especial talento para manipular o formato JSON, e grande conhecedor de bancos de dados, tenho um pedido à altura de seu calibre técnico. Como tenho a pretensão de fazer um emulador do antigo DBase e vou precisar, obviamente, de um banco de dados para armazenar as informações das tabelas, peço que você faça, em javascript, com telas em HTML5, o controle de tabelas de um Banco de Dados onde as tabelas tem estrutura JSON, com as funções para as tradicionais operações com tabelas: inclusão, alteração, deleção e listagem dos registros. Você pode fazer o procedimento em etapas, na ordem que achar melhor, e vamos interagindo para definir os rumos e características da aplicação.

O ChatGPT disse:

Atualizar memória?

“O usuário está desenvolvendo um emulador do antigo...”

Sim

Não

Excelente proposta, Baruch! Um emulador do DBase em JavaScript com estrutura em JSON é um projeto bem interessante e educativo — com potencial para funcionar inteiramente no navegador, sem dependências externas.

Vamos estruturar o projeto em **etapas modulares**, partindo do núcleo essencial até a interface gráfica:

✅ ETAPA 1 – Núcleo do "Banco de Dados" em JSON

Nesta etapa, criaremos um objeto `Database` que contém múltiplas "tabelas", onde cada tabela é um array de objetos (registros).

Cada tabela também manterá seu próprio "schema" (estrutura dos campos).

📁 Estrutura de Armazenamento:

```
javascript
CopiarEditar
const database = {
  TABELAS: {
    clientes: {
      schema: ["codigo", "nome", "email"],
      dados: [
        { codigo: 1, nome: "Ana", email: "ana@exemplo.com" },
        { codigo: 2, nome: "Bruno", email: "bruno@exemplo.com" }
      ]
    },
    produtos: {
      schema: ["codigo", "descricao", "preco"],
      dados: []
    }
  }
};
```

⚙️ Funções básicas do núcleo:

```
javascript
CopiarEditar
class JSONDB {
```

```

constructor() {
  this.tables = {};
}

criarTabela(nome, campos) {
  if (this.tables[nome]) throw new Error("Tabela já existe");
  this.tables[nome] = { schema: campos, dados: [] };
}

inserir(nome, registro) {
  const tabela = this.tables[nome];
  if (!tabela) throw new Error("Tabela inexistente");
  if (!this._validaRegistro(tabela.schema, registro)) throw new
Error("Registro inválido");
  tabela.dados.push(registro);
}

listar(nome) {
  const tabela = this.tables[nome];
  if (!tabela) throw new Error("Tabela inexistente");
  return tabela.dados;
}

alterar(nome, indice, novoRegistro) {
  const tabela = this.tables[nome];
  if (!tabela || !tabela.dados[indice]) throw new Error("Registro não
encontrado");
  if (!this._validaRegistro(tabela.schema, novoRegistro)) throw new
Error("Registro inválido");
  tabela.dados[indice] = novoRegistro;
}

deletar(nome, indice) {
  const tabela = this.tables[nome];
  if (!tabela || !tabela.dados[indice]) throw new Error("Registro não
encontrado");
  tabela.dados.splice(indice, 1);
}

_validaRegistro(schema, registro) {
  return schema.every(campo => campo in registro);
}
}

```

✓ Próximos Passos:

1. **ETAPA 2:** Interface em HTML5 para criar tabelas e registros dinamicamente.
 2. **ETAPA 3:** Armazenamento local usando `localStorage` ou `IndexedDB` (persistência).
 3. **ETAPA 4:** Filtros, ordenação e busca textual.
 4. **ETAPA 5:** Exportação e importação em JSON, CSV e outros formatos.
 5. **ETAPA 6:** Emulação mais fiel ao DBase: comandos como `USE`, `APPEND`, `REPLACE`, `BROWSE`, etc.
-

Quer que eu siga agora com a ETAPA 2 (interface HTML5 simples para manipular os dados)? Ou deseja revisar/modificar o núcleo antes?

Você disse:

Uma observação, importante neste ponto, antes de prosseguir com as próximas etapas. As tabelas de formato JSON deverão ser o local de armazenamento dos registros, portanto quero sua gravação no formato de arquivo que possa ser lido através de chamadas AJAX.

O ChatGPT disse:

Perfeito, Baruch. Isso muda um aspecto importante: agora vamos trabalhar com **arquivos JSON reais**, gravados e lidos via chamadas AJAX. Isso significa:

✅ Novo requisito: Armazenamento em arquivos .json acessíveis via AJAX

Como estamos em ambiente de navegador puro (HTML + JS), há duas possibilidades principais:

A Ambiente Local (sem servidor):

AJAX **não pode gravar arquivos** diretamente no disco local por razões de segurança do navegador.

⚠️ Limitação: Só é possível *ler* arquivos JSON via AJAX, se o site for servido por um servidor local ou remoto (por ex: Apache, Node, Python HTTP server).

B Ambiente com servidor (recomendado):

Aqui podemos usar um servidor (por exemplo, com **Node.js** ou **PHP**) para:

- ✅ Salvar os registros das tabelas como arquivos .json no backend.
- ✅ Ler os arquivos via AJAX no frontend.
- ✅ Atualizar (salvar, deletar, alterar) usando chamadas `fetch` com POST, PUT, etc.

📁 Estrutura de Arquivos Recomendada

```
pgsql
CopiarEditar
/projeto/
├── index.html
├── app.js
├── jsondb.js
├── tabelas/
│   ├── clientes.json
│   └── produtos.json
└── backend/
    └── salvar.php (ou salvar.js, se for Node)
```

Solução Técnica (Resumo)

- No **frontend**:
 - Leitura das tabelas via `fetch('/tabelas/clientes.json')`
 - Envio de alterações via `POST` para um script no backend
 - No **backend**:
 - Script PHP/Node que recebe o conteúdo JSON e grava no arquivo correspondente
-

Proposta de Caminho a Seguir

1. ETAPA 2 (frontend):


- Montar uma interface HTML5 simples com JavaScript
- Carregar e mostrar uma tabela JSON via AJAX
- Permitir inserir, alterar e deletar registros localmente

2. ETAPA 3 (backend):

- Implementar `salvar.php` para receber via `POST` um JSON com nome da tabela + dados e gravar como `.json`

3. ETAPA 4: Interligar tudo — fazer `fetch` tanto para leitura quanto para salvar alterações nos arquivos

Posso começar agora a ETAPA 2 com a interface HTML + JS que carrega uma tabela `.json` via AJAX (`fetch`) e manipula os dados localmente.

 Quer seguir com PHP no backend, ou prefere Node.js?

Você disse:

Pode prosseguir com PHP, pois meu desktop já possui um ambiente de teste IIS com PHP.

O ChatGPT disse:

Memória salva atualizada

Ótimo! Como você já tem PHP no IIS, vamos seguir com o modelo de leitura e escrita de arquivos JSON no backend com PHP.

ETAPA 2: Interface HTML5 com carregamento AJAX

Vamos montar:

- Uma página HTML com:
 - Seleção da tabela
 - Exibição de registros

- Formulário para adicionar e editar registros
 - Botões para salvar alterações no servidor (via PHP)
-

Estrutura esperada dos arquivos

```
bash
CopiarEditar
/projeto/
├── index.html
├── app.js
├── jsondb.js
├── /tabelas/
│   ├── clientes.json
│   └── produtos.json
├── /backend/
└── salvar.php
```

index.html (interface)

```
html
CopiarEditar
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <title>Mini DBase em JS</title>
  <script defer src="jsondb.js"></script>
  <script defer src="app.js"></script>
  <style>
    table { border-collapse: collapse; width: 100%; }
    th, td { border: 1px solid #ccc; padding: 5px; }
    input { margin: 5px; }
  </style>
</head>
<body>
  <h1>Mini DBase em JSON</h1>

  <label for="tabelaSelect">Tabela:</label>
  <select id="tabelaSelect">
    <option value="clientes">Clientes</option>
    <option value="produtos">Produtos</option>
  </select>
  <button onclick="carregarTabela()">Carregar</button>

  <table id="tabelaDados"></table>

  <h2>Novo Registro</h2>
  <form id="registroForm"></form>
  <button onclick="inserirRegistro()">Inserir</button>
  <button onclick="salvarTabela()">Salvar no servidor</button>
</body>
</html>
```

jsondb.js (núcleo em JS)

```
javascript
CopiarEditar
class JSONDB {
  constructor() {
    this.schema = [];
    this.dados = [];
    this.nomeTabela = "";
  }

  async carregar(nome) {
    this.nomeTabela = nome;
    const res = await fetch(`tabelas/${nome}.json`);
    const json = await res.json();
    this.schema = json.schema;
    this.dados = json.dados;
  }

  inserir(registro) {
    this.dados.push(registro);
  }

  deletar(indice) {
    this.dados.splice(indice, 1);
  }

  alterar(indice, registro) {
    this.dados[indice] = registro;
  }

  exportar() {
    return {
      schema: this.schema,
      dados: this.dados
    };
  }
}
```

app.js (controle da interface)

```
javascript
CopiarEditar
const db = new JSONDB();

async function carregarTabela() {
  const nome = document.getElementById("tabelaSelect").value;
  await db.carregar(nome);
  renderTabela();
  renderFormulario();
}

function renderTabela() {
  const tabela = document.getElementById("tabelaDados");
  tabela.innerHTML = "";

  const thead = document.createElement("tr");
  db.schema.forEach(campo => {
    const th = document.createElement("th");
    th.textContent = campo;
    thead.appendChild(th);
  });
}
```

```

});
thead.appendChild(document.createElement("th")); // Ações
tabela.appendChild(thead);

db.dados.forEach((registro, i) => {
  const tr = document.createElement("tr");
  db.schema.forEach(campo => {
    const td = document.createElement("td");
    td.textContent = registro[campo];
    tr.appendChild(td);
  });

  const tdAcoes = document.createElement("td");
  tdAcoes.innerHTML = `<button onclick="db.deletar(${i}); renderTabela()">❌
</button>`;
  tr.appendChild(tdAcoes);
  tabela.appendChild(tr);
});
}

function renderFormulario() {
  const form = document.getElementById("registroForm");
  form.innerHTML = "";
  db.schema.forEach(campo => {
    const input = document.createElement("input");
    input.placeholder = campo;
    input.id = `campo_${campo}`;
    form.appendChild(input);
  });
}

function inserirRegistro() {
  const registro = {};
  db.schema.forEach(campo => {
    registro[campo] = document.getElementById(`campo_${campo}`).value;
  });
  db.inserir(registro);
  renderTabela();
}

async function salvarTabela() {
  const nome = db.nomeTabela;
  const dados = db.exportar();
  const res = await fetch(`backend/salvar.php`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ nome, dados })
  });
  const texto = await res.text();
  alert(texto);
}

```

backend/salvar.php (salvar no servidor)

```

php
CopiarEditar
<?php
$data = json_decode(file_get_contents("php://input"), true);
$nome = basename($data["nome"]);
$dados = $data["dados"];

```

```
if (!$nome || !$dados) {
    http_response_code(400);
    echo "Dados inválidos";
    exit;
}

$arquivo = "../tabelas/{$nome}.json";
file_put_contents($arquivo, json_encode($dados, JSON_PRETTY_PRINT |
JSON_UNESCAPED_UNICODE));
echo "Tabela '{$nome}' salva com sucesso!";
?>
```

Exemplo: tabelas/clientes.json

```
json
CopiarEditar
{
    "schema": ["codigo", "nome", "email"],
    "dados": [
        { "codigo": 1, "nome": "Ana", "email": "ana@exemplo.com" },
        { "codigo": 2, "nome": "Bruno", "email": "bruno@exemplo.com" }
    ]
}
```

Etapas concluídas:

- Leitura e edição de tabelas via JSON
 - Inclusão e deleção de registros
 - Gravação no backend com PHP
-

Próximas sugestões:

- Edição de registros (modo inline ou via formulário)
 - Validação de tipos de dados
 - Exportar/importar CSV/JSON
 - Filtros e ordenação
 - Interface mais parecida com **BROWSE** do DBase
-

Deseja que avancemos para algum desses pontos agora ou há algo que queira ajustar primeiro?

Você disse:

Experimentei 1. Carregar a tabela s clientes (OK) 2. Incluir um novo cliente e listar. Parece que não gravou no JSON não, ou exibiu um JSON já carregado na memória. Você poderia corrigir o problema ?

O ChatGPT disse:

Ótima observação — o comportamento atual **mantém os dados apenas em memória**, e **não recarrega do arquivo JSON após salvar**. Isso significa que a inclusão aparece na interface, mas não se reflete no arquivo `clientes.json` a menos que o salvamento funcione corretamente.

Vamos garantir agora:

✓ Correção e Verificação: Gravação no arquivo JSON

🔪 Etapas para identificar e corrigir o problema:

1. Verificar se `salvar.php` está sendo chamado corretamente

Adicione um `console.log` e teste no navegador:

```
javascript
CopiarEditar
async function salvarTabela() {
  const nome = db.nomeTabela;
  const dados = db.exportar();
  console.log("Enviando para salvar:", nome, dados); // <-- aqui

  const res = await fetch(`backend/salvar.php`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ nome, dados })
  });
  const texto = await res.text();
  alert(texto);
}
```

2. Corrigir `salvar.php` (se necessário)

O script PHP salva no caminho relativo `../tabelas/{nome}.json`. Se o diretório `tabelas/` não tiver permissão de escrita pelo IIS/PHP, **o arquivo não será salvo**.

👉 Correção com verificação de erro:

```
php
CopiarEditar
<?php
$data = json_decode(file_get_contents("php://input"), true);

$nome = basename($data["nome"] ?? '');
$dados = $data["dados"] ?? null;

if (!$nome || !$dados) {
  http_response_code(400);
  echo "Dados inválidos";
  exit;
}

$arquivo = realpath(__DIR__ . "../tabelas/") . "/" . $nome . ".json";

if (!is_writable(dirname($arquivo))) {
  http_response_code(500);
```

```
    echo "Diretório não tem permissão de escrita.";
    exit;
}

$resultado = file_put_contents($arquivo, json_encode($dados, JSON_PRETTY_PRINT |
JSON_UNESCAPED_UNICODE));

if ($resultado === false) {
    http_response_code(500);
    echo "Erro ao gravar o arquivo.";
} else {
    echo "Tabela '$nome' salva com sucesso!";
}
?>
```

3. (Opcional) Recarregar a tabela após salvar

```
javascript
CopiarEditar
async function salvarTabela() {
    const nome = db.nomeTabela;
    const dados = db.exportar();

    const res = await fetch(`backend/salvar.php`, {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ nome, dados })
    });
    const texto = await res.text();
    alert(texto);

    // Recarrega para verificar persistência
    await db.carregar(nome);
    renderTabela();
}
```

Teste completo sugerido:

1. Abra o DevTools (F12), aba *Console* e *Network*.
 2. Adicione um novo registro.
 3. Clique em "**Salvar no servidor**".
 4. Verifique:
 - Se houve requisição POST para `salvar.php`
 - Se a resposta indica sucesso
 5. Veja se o arquivo `clientes.json` foi realmente alterado.
 6. Recarregue a página e clique em "**Carregar**".
-

Se continuar não gravando, verifique:

- Se a pasta `tabelas/` tem permissão de escrita no IIS
- Se `salvar.php` está no caminho correto e sendo acessado corretamente
- Se os erros do PHP estão ativados (`display_errors = On` em `php.ini`)

Deseja que eu prepare também um script PHP simples para testar gravação de arquivos diretamente (fora do sistema) para garantir que o PHP está com permissão adequada?

Você disse:

Averigui a tabela `clientes.json`. O novo registro está lá, mas o botão "carregar" não o mostra. O código do botão não está se comportando direito. Corrija, por favor.

O ChatGPT disse:

Perfeito, Baruch — obrigado pela checagem minuciosa. Se o novo registro **está gravado corretamente** no arquivo `clientes.json`, mas **não aparece ao clicar no botão "Carregar"**, então o problema está no fluxo da função `carregarTabela()`.

A causa mais provável é: o carregamento acontece, **mas a tabela não está sendo renderizada porque o `renderTabela()` está usando um estado desatualizado**, ou o cache do navegador está servindo o JSON antigo.

✅ Correção Completa da Função `carregarTabela()`

Vamos garantir:

1. Que o **fetch realmente força recarregar** o arquivo JSON atualizado.
2. Que o **`renderTabela()` atualize corretamente a visualização**.

🔧 Correção em `carregarTabela()`

No `app.js`:

```
javascript
CopiarEditar
async function carregarTabela() {
  const nome = document.getElementById("tabelaSelect").value;

  // Evita cache com um timestamp no final da URL
  const url = `tabelas/${nome}.json?t=${Date.now()}`;
  const res = await fetch(url, { cache: "no-store" });

  const json = await res.json();
  db.nomeTabela = nome;
  db.schema = json.schema;
  db.dados = json.dados;

  renderTabela();
  renderFormulario();
}
```

✅ **Alternativamente: corrigir dentro do JSONDB se preferir manter separação:**

Em jsondb.js:

```
javascript
CopiarEditar
async carregar(nome) {
  this.nomeTabela = nome;
  const res = await fetch(`tabelas/${nome}.json?t=${Date.now()}`, { cache: "no-store" });
  const json = await res.json();
  this.schema = json.schema;
  this.dados = json.dados;
}
```

E manter em app.js:

```
javascript
CopiarEditar
async function carregarTabela() {
  const nome = document.getElementById("tabelaSelect").value;
  await db.carregar(nome);
  renderTabela();
  renderFormulario();
}
```

💡 Por que isso funciona?

- `?t=${Date.now() }` força o navegador a buscar uma nova versão do arquivo (evita cache)
- `cache: "no-store"` impede que o `fetch` use uma cópia antiga da requisição
- Re-renderização assegura que o DOM reflita o conteúdo atualizado da tabela JSON

✅ **Teste agora:**

1. Carregue a página
2. Clique em "**Carregar**"
3. Verifique se os novos registros do JSON aparecem corretamente

Se quiser, posso incluir também um botão "Recarregar do servidor" separado para testar a diferença entre dados em memória e o que foi salvo em disco.

Deseja isso?

Você disse:

Agora funcionou. Não precisa deste novo botão. Vamos prosseguir com seu planejamento.

O ChatGPT disse:

Ótimo, Baruch! Feliz em saber que agora o sistema está funcionando corretamente — leitura, inclusão e gravação de registros estão integradas como esperado.

✅ Próxima Etapa: Edição de Registros

Na tradição do DBase, editar um registro equivale ao comando REPLACE. Vamos implementar isso com:

1. Um botão "Editar" por linha
 2. Um formulário reutilizável para atualizar os campos
 3. Um botão "Salvar alterações"
-

🎯 Objetivo:

- Permitir ao usuário clicar em " Editar" para carregar os dados do registro no formulário
 - Alterar os valores e clicar em "Salvar alteração"
 - Atualizar a tabela e permitir salvar no backend
-

🔧 Alterações em `app.js`

1. Adicionar nova variável de estado:

```
javascript
CopiarEditar
let indiceEditando = null;
```

2. Alterar `renderTabela()` para incluir botão Editar:

```
javascript
CopiarEditar
function renderTabela() {
  const tabela = document.getElementById("tabelaDados");
  tabela.innerHTML = "";

  const thead = document.createElement("tr");
  db.schema.forEach(campo => {
    const th = document.createElement("th");
    th.textContent = campo;
    thead.appendChild(th);
  });
  thead.appendChild(document.createElement("th")); // Ações
  tabela.appendChild(thead);

  db.dados.forEach((registro, i) => {
    const tr = document.createElement("tr");
    db.schema.forEach(campo => {
      const td = document.createElement("td");
```

```

        td.textContent = registro[campo];
        tr.appendChild(td);
    });

    const tdAcoes = document.createElement("td");
    tdAcoes.innerHTML = `
        <button onclick="editarRegistro(${i})"> </button>
        <button onclick="db.deletar(${i}); renderTabela()"> ✖ </button>
    `;
    tr.appendChild(tdAcoes);
    tabela.appendChild(tr);
});
}

```

3. Função editarRegistro()

```

javascript
CopiarEditar
function editarRegistro(indice) {
    indiceEditando = indice;
    const registro = db.dados[indice];
    db.schema.forEach(campo => {
        document.getElementById(`campo_${campo}`).value = registro[campo];
    });

    document.querySelector('button[onclick="inserirRegistro()"]').style.display =
    "none";
    if (!document.getElementById("btnSalvarAlteracao")) {
        const btnSalvar = document.createElement("button");
        btnSalvar.id = "btnSalvarAlteracao";
        btnSalvar.textContent = "Salvar alteração";
        btnSalvar.onclick = salvarAlteracao;
        document.getElementById("registroForm").appendChild(btnSalvar);
    }
}

```

4. Função salvarAlteracao()

```

javascript
CopiarEditar
function salvarAlteracao() {
    const novoRegistro = {};
    db.schema.forEach(campo => {
        novoRegistro[campo] = document.getElementById(`campo_${campo}`).value;
    });

    db.alterar(indiceEditando, novoRegistro);
    indiceEditando = null;
    renderTabela();
    renderFormulario();
}

```

5. Atualizar renderFormulario() para resetar estado do botão

```

javascript
CopiarEditar


```

```
function renderFormulario() {
  const form = document.getElementById("registroForm");
  form.innerHTML = "";
  db.schema.forEach(campo => {
    const input = document.createElement("input");
    input.placeholder = campo;
    input.id = `campo_${campo}`;
    form.appendChild(input);
  });

  const btnInserir = document.createElement("button");
  btnInserir.textContent = "Inserir";
  btnInserir.setAttribute("onclick", "inserirRegistro()");
  form.appendChild(btnInserir);
}
```

✓ Resultado

Você poderá:

- Clicar em  para editar qualquer registro
 - Atualizar os campos
 - Clicar em "**Salvar alteração**" para aplicar as mudanças
 - Clicar em "**Salvar no servidor**" para gravar as alterações no `.json`
-

Pronto para seguir?

Quando quiser, posso propor a **ETAPA 4**:

-  **Busca, filtro e ordenação de registros**, estilo LOCATE e BROWSE do dBase.

Ou prefere agora trabalhar na exportação/importação (APPEND FROM, COPY TO, etc)?

Você disse:




Pronto, Pode fazer a Etapa 4 de seu planejamento.

O ChatGPT disse:

Excelente, Baruch! Vamos à **ETAPA 4: Filtros, Busca e Ordenação**, inspirados nos comandos clássicos do dBase como LOCATE, BROWSE, SET ORDER TO, etc.

✓ Objetivo desta etapa

Implementar:

1.  **Busca textual por qualquer campo**
2.  **Filtro por campo específico**
3.  **Ordenação por coluna**

Tudo isso será feito **no frontend**, manipulando os dados carregados da tabela JSON.

Estratégia

- Manter uma cópia da tabela original (`db.dados`)
 - Aplicar busca e filtro em tempo real sobre essa cópia
 - Renderizar a tabela dinamicamente com os resultados filtrados
 - Adicionar ordenação ao clicar no cabeçalho da coluna
-

Modificações no sistema

1. Adicione os controles no `index.html` abaixo da tabela:

```
html
CopiarEditar
<h2>Busca, Filtro e Ordenação</h2>
<input type="text" id="buscaGlobal" placeholder="Buscar em todos os campos"
oninput="filtrarRegistros()">
<select id="campoFiltro"></select>
<input type="text" id="valorFiltro" placeholder="Filtrar campo"
oninput="filtrarRegistros()">
```

2. Atualize o estado no `app.js`:

Adicione:

```
javascript
CopiarEditar
let dadosFiltrados = []; // Exibe apenas registros filtrados
let ordemAtual = { campo: null, asc: true };
```

3. Atualize `carregarTabela()` para preparar os filtros:

```
javascript
CopiarEditar
async function carregarTabela() {
  const nome = document.getElementById("tabelaSelect").value;
  await db.carregar(nome);
  dadosFiltrados = [...db.dados];
  renderTabela();
  renderFormulario();
  renderControlesFiltro();
}
```

4. Nova função: renderControlesFiltro()

```
javascript
CopiarEditar
function renderControlesFiltro() {
  const campoFiltro = document.getElementById("campoFiltro");
  campoFiltro.innerHTML = `<option value="">[Todos os campos]</option>`;
  db.schema.forEach(campo => {
    const opt = document.createElement("option");
    opt.value = campo;
    opt.textContent = campo;
    campoFiltro.appendChild(opt);
  });
}
```

5. Função de filtro e busca:

```
javascript
CopiarEditar
function filtrarRegistros() {
  const busca = document.getElementById("buscaGlobal").value.toLowerCase();
  const campo = document.getElementById("campoFiltro").value;
  const valor = document.getElementById("valorFiltro").value.toLowerCase();

  dadosFiltrados = db.dados.filter(reg => {
    const textoGlobal = Object.values(reg).join(" ").toLowerCase();
    const matchBusca = !busca || textoGlobal.includes(busca);
    const matchCampo = !campo || (reg[campo] +
    "").toLowerCase().includes(valor);
    return matchBusca && matchCampo;
  });

  renderTabela();
}
```

6. Ordenação ao clicar no cabeçalho:

Altere renderTabela():

```
javascript
CopiarEditar
function renderTabela() {
  const tabela = document.getElementById("tabelaDados");
  tabela.innerHTML = "";

  const thead = document.createElement("tr");
  db.schema.forEach(campo => {
    const th = document.createElement("th");
    th.textContent = campo + (ordemAtual.campo === campo ? (ordemAtual.asc ? "
↑" : "↓") : "");
    th.style.cursor = "pointer";
    th.onclick = () => ordenarPor(campo);
    thead.appendChild(th);
  });
  thead.appendChild(document.createElement("th")); // Ações
  tabela.appendChild(thead);

  dadosFiltrados.forEach((registro, i) => {
```

```

const tr = document.createElement("tr");
db.schema.forEach(campo => {
  const td = document.createElement("td");
  td.textContent = registro[campo];
  tr.appendChild(td);
});

const tdAcoes = document.createElement("td");
tdAcoes.innerHTML = `
  <button onclick="editarRegistro(${i})">✎</button>
  <button onclick="db.deletar(${i}); filtrarRegistros()">🗑️</button>
`;
tr.appendChild(tdAcoes);
tabela.appendChild(tr);
});
}

```

7. Função de ordenação:

```

javascript
CopiarEditar
function ordenarPor(campo) {
  const asc = (ordemAtual.campo === campo) ? !ordemAtual.asc : true;
  ordemAtual = { campo, asc };




  dadosFiltrados.sort((a, b) => {
    const valA = a[campo]?.toString().toLowerCase() || "";
    const valB = b[campo]?.toString().toLowerCase() || "";
    if (valA < valB) return asc ? -1 : 1;
    if (valA > valB) return asc ? 1 : -1;
    return 0;
  });

  renderTabela();
}

```

Resultado

Você agora tem:

-  Busca geral por todos os campos
 -  Filtro por campo específico
 -  Ordenação por coluna ao clicar no cabeçalho
-

Etapas seguintes sugeridas:

- **ETAPA 5:** Exportação (COPY TO) e importação (APPEND FROM) para CSV e JSON
- **ETAPA 6:** Comandos estilo DBase (USE, REPLACE, BROWSE, etc.) com terminal textual opcional
- **ETAPA 7:** Interface estilo terminal retro (modo texto)

Deseja avançar para exportação/importação na próxima etapa?

Você disse:

A interface HTML, até agora, está boa para um técnico que conhece TI, mas pode gerar confusão para leigos. Preciso das seguintes providências na interface HTML: A interface de busca e ordenação só pode aparecer se alguma tabela estiver carregada;

O ChatGPT disse:

Perfeito, Baruch — sua consideração é muito pertinente. Para tornar a interface mais clara para usuários não técnicos, vamos implementar a **exibição condicional dos controles de busca, filtro e ordenação** — **somente após o carregamento de uma tabela.**

✓ Objetivo:

Exibir os controles de busca/filtro/ordenação **somente quando uma tabela estiver carregada** com sucesso.

🔧 Como implementar isso:

1. Encapsular os controles de busca/filtro dentro de um `<div>` com `id="painelControles"`
 2. Inicialmente, esconder esse `<div>` via CSS
 3. Após o carregamento da tabela, torná-lo visível via JavaScript
-

✓ Etapas:

1. 📄 Atualize o `index.html`

Envolva os controles em um `div` com `id="painelControles"` e adicione o estilo `display: none`:

```
html
CopiarEditar
<div id="painelControles" style="display: none;">
  <h2>Busca, Filtro e Ordenação</h2>
  <input type="text" id="buscaGlobal" placeholder="Buscar em todos os campos"
oninput="filtrarRegistros()">
  <select id="campoFiltro"></select>
  <input type="text" id="valorFiltro" placeholder="Filtrar campo"
oninput="filtrarRegistros()">
</div>
```

2. Em app.js: tornar o painel visível ao carregar a tabela

Modifique carregarTabela():

```
javascript
CopiarEditar
async function carregarTabela() {
  const nome = document.getElementById("tabelaSelect").value;
  await db.carregar(nome);
  dadosFiltrados = [...db.dados];

  // Exibe os elementos de controle
  document.getElementById("painelControles").style.display = "block";

  renderTabela();
  renderFormulario();
  renderControlesFiltro();
}
```

3. (Opcional) Ao resetar/voltar para "nenhuma tabela", esconder o painel novamente

Se quiser um botão "Nova seleção" ou "Limpar", você pode fazer isso também:

```
javascript
CopiarEditar
function limparInterface() {
  db.nomeTabela = "";
  db.schema = [];
  db.dados = [];
  dadosFiltrados = [];
  indiceEditando = null;
  document.getElementById("painelControles").style.display = "none";
  document.getElementById("tabelaDados").innerHTML = "";
  document.getElementById("registroForm").innerHTML = "";
}
```

Resultado

Agora:

- Quando o sistema abre: os controles de filtro, busca e ordenação não aparecem.
 - Somente ao **carregar uma tabela**, esses elementos são exibidos.
 - Isso reduz ruído visual para usuários leigos.
-

Deseja que a próxima melhoria na interface siga no sentido de **tornar os rótulos mais amigáveis e em português acessível**, ou prefere prosseguir com **funções como exportação/importação** na próxima etapa?

Você disse:

Ainda existe outra providência que eu gostaria que você fizesse na interface: Na caixa de escolha de tabelas, coloque a opção "[Nenhuma]". Desta forma, os controles de inclusão de nossa interface só apareceriam quando selecionássemos alguma tabela.

O ChatGPT disse:

Ótimo pedido, Baruch — essa melhoria traz ainda mais clareza e controle ao fluxo da aplicação, especialmente para usuários menos experientes.

Vamos implementar:



Objetivo

1. Adicionar a opção [Nenhuma] no `<select>` de tabelas
2. **Esconder** os elementos de formulário e controles enquanto nenhuma tabela estiver selecionada



Etapas para Implementação

1. Atualizar o HTML

Altere o `select` da tabela para incluir a opção "[Nenhuma]":

```
html
CopiarEditar
<label for="tabelaSelect">Tabela:</label>
<select id="tabelaSelect">
  <option value="">[Nenhuma]</option>
  <option value="clientes">Clientes</option>
  <option value="produtos">Produtos</option>
</select>
<button onclick="carregarTabela()">Carregar</button>

<!-- Painéis ocultáveis -->
<div id="painelControles" style="display: none;">
  <h2>Busca, Filtro e Ordenação</h2>
  <input type="text" id="buscaGlobal" placeholder="Buscar em todos os campos"
oninput="filtrarRegistros()">
  <select id="campoFiltro"></select>
  <input type="text" id="valorFiltro" placeholder="Filtrar campo"
oninput="filtrarRegistros()">
</div>

<table id="tabelaDados"></table>

<div id="painelFormulario" style="display: none;">
  <h2>Novo Registro</h2>
  <form id="registroForm"></form>
  <button onclick="inserirRegistro()">Inserir</button>
  <button onclick="salvarTabela()">Salvar no servidor</button>
</div>
```

2. Modificar `carregarTabela()` no `app.js`

```
javascript
CopiarEditar
async function carregarTabela() {
  const nome = document.getElementById("tabelaSelect").value;

  if (!nome) {
    limparInterface(); // Ao escolher [Nenhuma]
    return;
  }

  await db.carregar(nome);
  dadosFiltrados = [...db.dados];

  document.getElementById("painelControles").style.display = "block";
  document.getElementById("painelFormulario").style.display = "block";

  renderTabela();
  renderFormulario();
  renderControlesFiltro();
}
```

3. Adicionar função `limparInterface()`

```
javascript
CopiarEditar
function limparInterface() {
  db.nomeTabela = "";
  db.schema = [];
  db.dados = [];
  dadosFiltrados = [];
  indiceEditando = null;

  document.getElementById("painelControles").style.display = "none";
  document.getElementById("painelFormulario").style.display = "none";
  document.getElementById("tabelaDados").innerHTML = "";
  document.getElementById("registroForm").innerHTML = "";
}
```



Resultado esperado

- Ao abrir a interface, "[Nenhuma]" estará selecionado, e **nada aparece ainda** (nem tabela, nem formulário, nem filtros)
 - Somente após selecionar uma tabela como "Clientes" ou "Produtos" e clicar "**Carregar**", o restante da interface será exibido
 - Caso o usuário volte a selecionar "[Nenhuma]", tudo se oculta novamente
-

Tudo isso traz uma experiência mais guiada e intuitiva. Deseja que sigamos agora para:

- **ETAPA 5:** Exportação/importação de dados em CSV ou JSON ou
- Mais ajustes de UX antes de avançar?

Você disse:

Preciso que você faça mais um ajuste de UX. Ao ser exibido o conteúdo de uma tabela, é mostrado o contexto de "Novo Registro". Nele existem dois botões de inserir, mas apenas o segundo (externo ao formulário) faz o procedimento corretamente. E isto também confunde o digitador, pois ele não sabe qual dos dois utilizar. Isto é obrigatório, em termos de código, ou pode ser corrigido ?

O ChatGPT disse:

Excelente observação, Baruch. Esse problema de **botão duplicado** ocorre porque:

- Um botão é adicionado no `renderFormulario()`, dentro do `<form>`
- Outro botão está fora do `<form>` no HTML original

🔴 **Resultado:** dois botões "Inserir", e apenas o externo chama corretamente `inserirRegistro()`, o que confunde o usuário.

✅ Objetivo da correção

- Manter **apenas um botão funcional**
 - Evitar duplicidade
 - Garantir que o botão **não envie o formulário automaticamente** (evitando recarregamento de página)
 - Melhorar usabilidade (ação clara e única)
-

✅ Solução recomendada

🔧 **Remover o botão externo e criar o botão internamente no `renderFormulario()`**

1. 📄 **Remova do HTML original:**

```
html
CopiarEditar
<!-- Remova estas duas linhas -->
<!-- <button onclick="inserirRegistro()">Inserir</button>
<button onclick="salvarTabela()">Salvar no servidor</button> -->
```

Substitua por:

```
html
CopiarEditar
<div id="painelFormulario" style="display: none;">
```

```
<h2>Novo Registro</h2>
<form id="registroForm" onsubmit="event.preventDefault();
inserirRegistro();"></form>
</div>
```

2. 📄 Atualize renderFormulario() em app.js:

```
javascript
CopiarEditar
function renderFormulario() {
  const form = document.getElementById("registroForm");
  form.innerHTML = "";

  db.schema.forEach(campo => {
    const input = document.createElement("input");
    input.placeholder = campo;
    input.id = `campo_${campo}`;
    form.appendChild(input);
  });

  const btnInserir = document.createElement("button");
  btnInserir.textContent = "Inserir";
  btnInserir.type = "submit";
  form.appendChild(btnInserir);

  const btnSalvar = document.createElement("button");
  btnSalvar.textContent = "Salvar no servidor";
  btnSalvar.type = "button";
  btnSalvar.onclick = salvarTabela;
  form.appendChild(btnSalvar);
}
```

3. ✅ Garantir que o formulário não cause reload

Já resolvemos isso no atributo:

```
html
CopiarEditar
<form id="registroForm" onsubmit="event.preventDefault(); inserirRegistro();">
```

Ou, se preferir separar lógica JS:

```
javascript
CopiarEditar
document.getElementById("registroForm").onsubmit = (e) => {
  e.preventDefault();
  inserirRegistro();
};
```

Resultado

- Apenas **um botão "Inserir"** visível
- Sempre funcional

- Menos confusão para o usuário
 - "Salvar no servidor" continua disponível no mesmo local, com função clara
-

Deseja agora que avancemos para a próxima funcionalidade (ETAPA 5 — exportar/importar dados)?

Você disse:

Após a inserção de um registro, voltou a acontecer aquele problema do novo registro ser gravado na tabela JSON, mas não ser exibido ao se carregar a tabela (sem refresh do navegador).

O ChatGPT disse:

Perfeito, Baruch — esse é um efeito colateral comum quando se faz inserções e gravações *sem forçar recarregamento real do arquivo JSON*. Aqui está o que está acontecendo:

Diagnóstico

Após a inserção:

1. O novo registro é adicionado à memória (`db.dados`)
2. Ao clicar em "**Salvar no servidor**", o conteúdo atual de `db.dados` é enviado e salvo corretamente no `.json`
3. Mas ao clicar em "**Carregar**", o navegador pode **usar uma versão em cache** do JSON (ou mesmo o `db.dados` antigo continua em uso)

👉 Resultado: o dado *foi salvo*, mas a tabela exibe uma versão **desatualizada**, a menos que você **atualize manualmente o navegador ou force cache-bypass**.

Solução definitiva

Atualizar o botão "Salvar no servidor" para **recarregar a tabela do arquivo físico após o salvamento** — *isso já havia sido sugerido, mas provavelmente ficou sobrescrito com outras melhorias*.

Correção: função `salvarTabela()`

```
javascript
CopiarEditar
async function salvarTabela() {
  const nome = db.nomeTabela;
  const dados = db.exportar();

  const res = await fetch(`backend/salvar.php`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ nome, dados })
  });
}
```

```
});  
  
const texto = await res.text();  
alert(texto);  
  
// Recarrega os dados salvos diretamente do arquivo .json  
await db.carregar(nome);  
dadosFiltrados = [...db.dados];  
renderTabela();  
renderFormulario();  
renderControlesFiltro();  
}
```

✓ Garantias

- Garante que o conteúdo exibido vem do disco, e **não apenas da memória**
 - Corrige a possível inconsistência entre o que foi salvo e o que é mostrado
 - Elimina a necessidade de "refresh manual"
-



Teste

1. Carregue a tabela
 2. Adicione um novo registro
 3. Clique em **Salvar no servidor**
 4. Clique novamente em **Carregar**
 - O novo registro estará lá
 5. (Opcional) Abrir o arquivo `.json` no editor para confirmar a persistência
-

Deseja que eu registre esse passo como parte de um pequeno **manual técnico de uso** para sua aplicação futuramente? E seguimos agora para **exportação/importação**?

Você disse:

clientes.json

Arquivo

Ocorreu o problema de inserção novamente. Estou com 5 registros na tabela clientes.json, mas a aplicação só mostra 4. Estou anexando esta tabela para você conferir. Achei muito estranho.

O ChatGPT disse:

Obrigado pelo envio, Baruch. Acabei de analisar o conteúdo de `clientes.json`, e confirmo que há **exatamente 5 registros válidos** no array `dados`:

json

CopiarEditar

[

```
{ "codigo": "3", "nome": "Paulo",      "email": "paulo@exemplo.com" },  
{ "codigo": "4", "nome": "rosa",      "email": "rosa@exemplo.com" },
```

```
[
  { "codigo": "1", "nome": "Ana", "email": "anafsilva@exemplo.com" },
  { "codigo": "2", "nome": "Aníbal", "email": "anibal@exemplo.com" },
  { "codigo": "7", "nome": "Septimus", "email": "septimus@exemplo.com" }
]
```

Prováveis causas do problema

Você vê apenas **4 registros** na tela, embora o JSON tenha 5. Aqui estão as possíveis causas e como corrigi-las:

1. Erro na renderização: `forEach` está truncando a exibição?

Revise esta parte do código em `renderTabela()`:

```
javascript
CopiarEditar
dadosFiltrados.forEach((registro, i) => {
  const tr = document.createElement("tr");

  db.schema.forEach(campo => {
    const td = document.createElement("td");
    td.textContent = registro[campo];
    tr.appendChild(td);
  });


  const tdAcoes = document.createElement("td");
  tdAcoes.innerHTML = `
    <button onclick="editarRegistro(${i})">✎</button>
    <button onclick="db.deletar(${i}); filtrarRegistros()">🗑️</button>
  `;
  tr.appendChild(tdAcoes);
  tabela.appendChild(tr);
});
```

 **Ação: verifique no console se algum erro JavaScript está interrompendo a iteração.**

Basta um erro de digitação ou caractere especial mal tratado (como acento ou `undefined`) para o navegador interromper a renderização.

2. Verificar se `filtrarRegistros()` está sendo chamado no momento errado

Se você chama `filtrarRegistros()` ao carregar a tabela, mas os campos de filtro têm valores "residuais", ele pode ocultar registros involuntariamente.

 **Solução recomendada:** após carregar uma tabela, **limpar campos de busca/filtro antes de chamar `filtrarRegistros()`**

```
javascript
CopiarEditar
function limparCamposDeFiltro() {
```

```
document.getElementById("buscaGlobal").value = "";
document.getElementById("campoFiltro").value = "";
document.getElementById("valorFiltro").value = "";
}
```

Em `carregarTabela()`:

```
javascript
CopiarEditar
await db.carregar(nome);
dadosFiltrados = [...db.dados];
limparCamposDeFiltro();
renderTabela();
renderFormulario();
renderControlesFiltro();
```

✅ 3. Registros com campos `undefined` ou inválidos

Se o último registro (por exemplo "Septimus") tem um campo omitido ou mal formatado (por ex: `email: null`), pode gerar erro de renderização silencioso.

Solução: use fallback ao acessar campos no DOM:

```
javascript
CopiarEditar
td.textContent = registro[campo] ?? "";
```

✅ 4. Conflito entre `db.dados` e `dadosFiltrados`

Se em algum ponto `dadosFiltrados = db.dados` (por referência, não cópia), e em seguida você modifica `db.dados` (ex: edita ou deleta), isso pode corromper a exibição.

🔧 Solução padrão: sempre fazer cópia:

```
javascript
CopiarEditar
dadosFiltrados = [...db.dados];
```

✅ Recomendações finais

1. Verifique se os campos de busca/filtro estão **limpos** antes da renderização
 2. Faça `console.log(dadosFiltrados.length)` para confirmar quantos registros estão chegando à tela
 3. Teste temporariamente com o `renderTabela()` usando diretamente `db.dados` ao invés de `dadosFiltrados` para isolar o problema
-

Deseja que eu prepare uma função de diagnóstico com log visual que mostre na tela quantos registros foram lidos/carregados e quantos foram exibidos, para facilitar esse tipo de verificação?

Você disse:

OK, funcionou. Mas seria muito desejável que, após uma inclusão de registro, a tabela fosse recarregada, para que o usuário pudesse constatar visualmente que aquele registro já faz parte da tabela. É possível ?

O ChatGPT disse:

Sim, Baruch — é totalmente possível (e desejável!) recarregar a tabela automaticamente logo após uma **inclusão de registro**, de modo que o usuário **veja imediatamente o novo dado sendo integrado** ao restante da tabela.

✓ Objetivo

Após o usuário clicar em "**Inserir**":

1. O registro deve ser adicionado
 2. A tabela deve ser **reexibida**
 3. O formulário deve ser limpo
 4. A nova linha deve aparecer com os dados incluídos
-

🔧 Ajuste na função `inserirRegistro()`

Aqui está a versão atualizada com as melhorias:

```
javascript
CopiarEditar
function inserirRegistro() {
  const registro = {};
  db.schema.forEach(campo => {
    registro[campo] = document.getElementById(`campo_${campo}`).value;
  });

  db.inserir(registro);

  // Atualiza o conjunto filtrado
  dadosFiltrados = [...db.dados];

  // Limpa os campos do formulário
  db.schema.forEach(campo => {
    document.getElementById(`campo_${campo}`).value = "";
  });

  // Reexibe a tabela com o novo registro
  renderTabela();
}
```

✓ Explicação dos passos:

- `db.inserir(registro)` adiciona o novo registro na memória

- `dadosFiltrados = [...db.dados]` garante que o novo dado entre na visualização
 - O formulário é **limpo** para evitar confusão com dados "pendentes"
 - `renderTabela()` reexibe tudo imediatamente
-

💡 Extra opcional (destaque visual)

Se quiser, podemos adicionar uma pequena animação ou destaque visual para o último registro inserido (como cor de fundo temporária) — ajuda a chamar a atenção para o que acabou de acontecer.

Deseja que eu implemente esse destaque também?

Ou seguimos agora para a **ETAPA 5: Exportar/Importar dados em CSV/JSON?**

Você disse:

Por favor, não se chateie comigo, mas uma outra anomalia está ocorrendo. Os botões de Editar e Deletar são posicionais. Após uma ordenação, fui clicar no botão Editar e fiquei espantado, pois o registro exibido na edição era um outro. Você compreende isto, não ? Parece que o botão de Edição pega a ordem original de exibição, antes de qualquer ordenação. Pode corrigir para mim.

O ChatGPT disse:

De forma alguma fico chateado, Baruch — ao contrário: você está fazendo uma análise precisa, digna de um verdadeiro arquiteto de sistemas. E sim, **eu compreendo perfeitamente** o que está acontecendo.

🔍 Diagnóstico técnico

O problema surge porque os botões **Editar** e **Deletar** estão usando o índice `i` do loop:

```
javascript
CopiarEditar
<button onclick="editarRegistro(${i})"></button>
```

Mas esse `i` **corresponde à posição dentro de `dadosFiltrados`**, não necessariamente à posição real do registro dentro de `db.dados`.

👉 Após uma ordenação ou filtro, `dadosFiltrados[i]` pode representar um registro diferente do `db.dados[i]`.

✅ Solução correta

Alterar os botões para usarem uma **chave única** (ex: `codigo`) em vez do índice, e buscar o registro real no array original (`db.dados`) com base nesse identificador.

Etapas para corrigir

1. Atualize os botões na renderTabela()

```
javascript
CopiarEditar
const chavePrimaria = "codigo"; // Assumimos que "codigo" é o identificador único

dadosFiltrados.forEach((registro) => {
  const tr = document.createElement("tr");
  db.schema.forEach(campo => {
    const td = document.createElement("td");
    td.textContent = registro[campo];
    tr.appendChild(td);
  });

  const tdAcoes = document.createElement("td");
  tdAcoes.innerHTML = `
    <button onclick="editarRegistroPorCodigo('${registro[chavePrimaria]}')">✎</button>
    <button onclick="deletarRegistroPorCodigo('${registro[chavePrimaria]}')">🗑️</button>
  `;
  tr.appendChild(tdAcoes);
  tabela.appendChild(tr);
});
```

2. Nova função: localizar registro pelo código original

```
javascript
CopiarEditar
function encontrarIndicePorCodigo(codigo) {
  return db.dados.findIndex(r => r.codigo == codigo);
}
```

3. Atualizar editarRegistro() → vira editarRegistroPorCodigo(codigo)

```
javascript
CopiarEditar
function editarRegistroPorCodigo(codigo) {
  const i = encontrarIndicePorCodigo(codigo);
  if (i < 0) return;

  indiceEditando = i;
  const registro = db.dados[i];
  db.schema.forEach(campo => {
    document.getElementById(`campo_${campo}`).value = registro[campo];
  });

  document.querySelector('button[type="submit"]').style.display = "none";

  if (!document.getElementById("btnSalvarAlteracao")) {
    const btnSalvar = document.createElement("button");
```

```
    btnSalvar.id = "btnSalvarAlteracao";  
    btnSalvar.textContent = "Salvar alteração";  
    btnSalvar.onclick = salvarAlteracao;  
    document.getElementById("registroForm").appendChild(btnSalvar);  
  }  
}
```

4. Atualizar a função de deletar

```
javascript  
CopiarEditar  
function deletarRegistroPorCodigo(codigo) {  
  const i = encontrarIndicePorCodigo(codigo);  
  if (i < 0) return;  
  
  db.deletar(i);  
  dadosFiltrados = [...db.dados];  
  renderTabela();  
}
```

Resultado

- Agora os botões "Editar" e "Deletar" **agem sobre o registro correto**, independentemente da ordenação ou filtro aplicado
 - `codigo` é usado como chave confiável para localizar o registro no array original
 - Não há mais erro de "posição trocada"
-

Quer que eu agora reforce isso incluindo também **validação de unicidade do campo `codigo` na inclusão** (para evitar códigos repetidos)? Ou seguimos para a **ETAPA 5: Exportar e Importar dados**?

Você disse:

Ficaria grato se você fizesse a validação de unicidade do campo na inclusão, pois fica semelhante às aplicações robustas que fazemos utilizando Bancos de Dados.

O ChatGPT disse:

Perfeitamente, Baruch — validar a **unicidade de chave primária (`codigo`)** é fundamental para manter a integridade da tabela, como fazemos em qualquer banco de dados relacional. E faz todo sentido replicar esse comportamento no nosso emulador.

Objetivo

Durante a inclusão de um novo registro:

- Verificar se o valor do campo `codigo` já existe em `db.dados`
- Impedir a inserção e alertar o usuário, caso exista duplicata

Etapas de implementação

1. Supondo que o campo-chave seja "codigo"

Declare no topo do `app.js`:

```
javascript
CopiarEditar
const chavePrimaria = "codigo";
```

2. Atualize a função `inserirRegistro()` com verificação

```
javascript
CopiarEditar
function inserirRegistro() {
  const registro = {};
  db.schema.forEach(campo => {
    registro[campo] = document.getElementById(`campo_${campo}`).value.trim();
  });

  // Verifica unicidade do campo chave
  const existe = db.dados.some(r => r[chavePrimaria] ===
registro[chavePrimaria]);
  if (existe) {
    alert(`Já existe um registro com ${chavePrimaria} = $
{registro[chavePrimaria]}.`);
    return;
  }

  db.inserir(registro);
  dadosFiltrados = [...db.dados];

  // Limpa formulário
  db.schema.forEach(campo => {
    document.getElementById(`campo_${campo}`).value = "";
  });

  renderTabela();
}
```

Resultado

- Nenhum registro com `codigo` duplicado será inserido
 - O sistema se comporta como um banco relacional com chave primária
 - A mensagem de erro é amigável e impede erro silencioso
-

Observação futura

Se você permitir importação de dados (`APPEND FROM`), será útil aplicar essa mesma validação em lote — ou implementar uma opção de **substituir duplicatas** (como `REPLACE ON <campo>` no DBase).

Deseja que agora avancemos com a **ETAPA 5: Exportar/Importar CSV e JSON**, ou deseja mais alguma melhoria de robustez (por exemplo, tipos de dados, validação de campos obrigatórios etc.)?