Querys SQL to create the tables :

```sql
-- Armazena metadados dos documentos
CREATE TABLE documents (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  filename TEXT,
  filetype TEXT,
  source TEXT, -- email, upload, etc
  content TEXT, -- texto extraído
  extracted_json JSONB,
  created_at TIMESTAMP DEFAULT now()
);

-- Transcrições de áudio
CREATE TABLE audio_transcripts (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  filename TEXT,
  transcript TEXT,
  extracted_json JSONB,
  created_at TIMESTAMP DEFAULT now()
);

-- Dados extraídos de textos (e-mails, formulários)
CREATE TABLE text_inputs (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  source TEXT,
  raw_text TEXT,
  extracted_json JSONB,
  created_at TIMESTAMP DEFAULT now()
);


create table public.error_logs (
  id uuid primary key default gen_random_uuid(),
  workflow text not null,          -- Nome ou ID do workflow no N8N
  node text not null,              -- Nome do node onde o erro ocorreu
  timestamp timestamp with time zone default now(), -- Data e hora do erro
  error_message text not null,     -- Mensagem de erro capturada
  payload jsonb                    -- Dados de entrada/saída relacionados ao erro
);
```

Query to Add Embedding Column to Documents

```sql
alter table documents
add column embedding vector(1536);
```

Edge Function :

```
// Setup type definitions for built-in Supabase Runtime APIs
import "jsr:@supabase/functions-js/edge-runtime.d.ts";
import { createClient } from 'https://esm.sh/@supabase/supabase-js@2'

interface RequestBody {
  table: string;
  data: any;
}

const SUPABASE_URL = Deno.env.get('SUPABASE_URL')!;
const SUPABASE_SERVICE_ROLE_KEY =
Deno.env.get('SUPABASE_SERVICE_ROLE_KEY')!;

const supabase = createClient(SUPABASE_URL, SUPABASE_SERVICE_ROLE_KEY);

function isValidJSON(value: any): boolean {
  if (value === null || value === undefined) return true; // aceita null
  try {
    JSON.stringify(value);
    return true;
  } catch {
    return false;
  }
}

function isNonEmptyString(value: any): boolean {
  return typeof value === 'string' && value.trim().length > 0;
}

console.info('Supabase Edge Function started');

Deno.serve(async (req: Request) => {
  try {
    if (req.method !== 'POST') {
      return new Response(JSON.stringify({ error: 'Method Not Allowed' }), { status: 405 });
    }

    const body: RequestBody = await req.json();

    if (!body.table || !body.data) {
      return new Response(JSON.stringify({ error: 'Missing table or data in body' }), { status:
400 });
    }

    const { table, data } = body;
    switch (table) {
      case 'documents':
        if (
          !isNonEmptyString(data.filename) ||
          !isNonEmptyString(data.filetype) ||
          !isNonEmptyString(data.source) ||
          !isValidJSON(data.extracted_json)
        ) {
          return new Response(JSON.stringify({ error: 'Validation failed for documents' }), {
status: 400 });
        }
        const { error: errDoc } = await supabase.from('documents').insert([{
          filename: data.filename,
          filetype: data.filetype,
          source: data.source,
          content: data.content || null,
          extracted_json: data.extracted_json,
        }]);
        if (errDoc) {
          return new Response(JSON.stringify({ error: errDoc.message }), { status: 500 });
        }
        break;

      case 'audio_transcripts':
        if (
          !isNonEmptyString(data.filename) ||
          !isNonEmptyString(data.transcript) ||
          !isValidJSON(data.extracted_json)
        ) {
          return new Response(JSON.stringify({ error: 'Validation failed for audio_transcripts' }),
{ status: 400 });
        }
        const { error: errAudio } = await supabase.from('audio_transcripts').insert([{
          filename: data.filename,
          transcript: data.transcript,
          extracted_json: data.extracted_json,
        }]);
        if (errAudio) {
          return new Response(JSON.stringify({ error: errAudio.message }), { status: 500 });
        }
        break;

      case 'text_inputs':
        if (
          !isNonEmptyString(data.source) ||
          !isNonEmptyString(data.raw_text) ||
          !isValidJSON(data.extracted_json)
        ) {
          return new Response(JSON.stringify({ error: 'Validation failed for text_inputs' }), {
status: 400 });
        }
        const { error: errText } = await supabase.from('text_inputs').insert([{
          source: data.source,
          raw_text: data.raw_text,
          extracted_json: data.extracted_json,
        }]);
        if (errText) {
          return new Response(JSON.stringify({ error: errText.message }), { status: 500 });
        }
        break;

      case 'error_logs':
        if (
          !isNonEmptyString(data.workflow) ||
          !isNonEmptyString(data.node) ||
          !isNonEmptyString(data.error_message) ||
          !isValidJSON(data.payload)
        ) {
          return new Response(JSON.stringify({ error: 'Validation failed for error_logs' }), {
status: 400 });
        }
        const { error: errLog } = await supabase.from('error_logs').insert([{
          workflow: data.workflow,
          node: data.node,
          error_message: data.error_message,
          payload: data.payload || null,
        }]);
        if (errLog) {
          return new Response(JSON.stringify({ error: errLog.message }), { status: 500 });
        }
        break;

      default:
        return new Response(JSON.stringify({ error: 'Unknown table' }), { status: 400 });
    }

    return new Response(JSON.stringify({ success: true }), {
      status: 200,
      headers: { 'Content-Type': 'application/json' },
    });

  } catch (error) {
    return new Response(JSON.stringify({ error: (error as Error).message }), { status: 500 });
  }
});
```