Centro Universitário Leonardo da Vinci – UNIASSELVI
Curso Análise e Desenvolvimento de Sistemas – Imersão Profissional: Projeto de Banco de Dados

AGENDAMENTOS E STATUS DE DISPONIBILIDADE DE SERVIÇOS POUPATEMPO NA PALMA DA MÃO

Autor: João Paulo Lubawaski[1]

Tutor externo: Guilherme Santos[2]

1. MOTIVO DA ESCOLHA DO OBJETO DE ESTUDO

O Poupatempo representa um avanço significativo no atendimento ao público paulista, centralizando serviços de diversos órgãos públicos, tais quais a Sabesp e o Detran SP, passando por mais de 400 serviços relacionados à saúde, moradia, documentos pessoais, atendimento a consumidores e demais áreas. Porém, apesar de sua grandiosa utilidade, ainda enfrenta problemas comuns quando falamos de sistemas: instabilidades, problemas técnicos e por vezes, a inoperabilidade e impossibilidade de realização de alguns serviços pontuais, conforme o sistema específico utilizado pelo serviço.

Esses problemas afetam diretamente os usuários finais, que apenas descobrem sobre a inoperabilidade do sistema e impossibilidade da realização de seus serviços no polo de atendimento presencial, após terem se locomovido, gerenciado sua agenda e adiado outras demandas.

Logo, o motivo da escolha deste objeto de estudos vai muito além de sua natureza óbvia de exercitação do pensamento crítico, da criatividade e da aplicação dos conhecimentos obtidos através das matérias referentes à banco de dados, mas também é feito pensando na resolução parcial dessas problemáticas já citadas, com objetivo de idealizar um sistema de banco de dados capaz de armazenar informações de agendamentos, relacionar com os sistemas necessários e seus status. Também, estendemos brevemente a pesquisa para abranger outras questões úteis para resolução do problema raiz.

2. ESTRATÉGIAS DE ANÁLISE DO OBJETO

[1] Acadêmico do Curso de Análise e Desenvolvimento de Sistemas em Centro Universitário Leonardo da Vinci – UNIASSELVI; E-mail: 8668254@aluno.uniasselvi.com.br
[2] Tutor Externo do Curso de Imersão Profissional: Projeto de Banco de Dados em UNIASSELVI – Polo Castelo, Campinas SP; E-mail: guilherme.santos@regente.uniasselvi.com.br

A principal origem da análise se dá em um episódio pessoal, em determinado momento em que me organizei para locomoção ao Poupatempo de Campinas, esperei pelo meu atendimento com os documentos em mãos e fui surpreendido pela atendente que me informou sobre a indisponibilidade do sistema (SEI), impossibilitando a realização do serviço pretendido. A partir desse evento, busquei por outros relatos semelhantes na internet, encontrando reclamações registradas no site Reclame Aqui (RECLAME AQUI, 2024) e uma reportagem publicada pelo G1 (RODRIGUES, 2025), evidenciando que essa falha não se trata apenas de um caso isolado.

Então, ao entender a questão como uma problemática que atinge milhares de paulistanos a muito tempo, o objeto de estudos mostra-se realmente digno e necessário de ser avaliado. Agora, o foco torna-se técnico: O problema é existente e pertinente, mas como resolvê-lo?

Partimos para pesquisas mais profundas em materiais bibliográficos e técnicos que tratam sobre diferentes conceitos que se mostravam úteis para a resolução de conflitos como esse. Autores como Ayooluwa Isaías e instituições como Amazon Web Services (2025) e Microsoft (2025) foram fundamentais para entender boas práticas de verificação de integridade de sistemas, arquitetura baseada em eventos e uso de APIs RESTful. Infelizmente, não consegui contato direto com representantes do Poupatempo ou da Prodesp (responsável pelo sistema atual do Poupatempo) devido ao prazo de resposta institucional (30 dias, prorrogáveis), foi possível deduzir a estrutura do sistema atual com base em conhecimento previamente adquirido em disciplinas como Banco de Dados e Estrutura de Dados.

Também, envolvemos nessa análise a comparação de práticas de engenharia de confiabilidade de sites (SRE), amplamente utilizadas em plataformas como a AWS (2025), e o estudo de possíveis soluções para o monitoramento contínuo. Assim, apesar de concentrarmos a pesquisa nos conceitos de Banco de Dados, podemos prever ou traçar caminhos aos quais as outras diferentes frentes seriam importantes para a completude do objeto inicialmente proposto para essa solução.

3. CONSIDERAÇÕES CRÍTICAS E CRIATIVAS

Sabemos que não é de hoje que sistemas enfrentam problemas relacionados a instabilidades; com certeza todos já enfrentaram algum momento em que nos organizamos para resolver algum problema, mas não conseguimos realizar este por conta de problemas relacionados ao sistema que gerencia esses atendimentos.

O Poupatempo, apesar de eficaz, não fica de fora desse risco, passando por quedas, infelizmente de forma mais comum que o esperado. Usuários, claro, reclamam, pois esses problemas só são constatados na grande maioria das vezes já no balcão de atendimento, o que representa o cancelamento de compromissos, perda de horas em seu dia e estresses psicológicos.

No Reclame Aqui, uma usuária reclama: “Poupa tempo vive sem sistema, 3ª vez que gasto meu tempo e dinheiro para ir até o local e está sempre sem sistema e nunca tem previsão de volta.” (RECLAME AQUI, 2024).

Essa problemática, já existente há algum tempo, foi reportada inclusive pelo G1, que entrevistou diferentes pessoas que passaram por situações parecidas, como a assistente comercial Vânia Rodrigues, que contou à equipe de reportagem: “Estive lá no dia 6 e não consegui atendimento. Remarquei para o dia 7 e novamente não consegui. Só fui atendida ao voltar em um sábado. Era um documento simples que eu precisava emitir e me deu uma enorme dor de cabeça. Me parece que esses problemas têm acontecido principalmente no período da tarde"(RODRIGUES, Vânia, G1, 2024).

Levando em consideração situações como essa, pensamos na possibilidade de desenvolver um sistema de banco de dados que além de armazenar os agendamentos, é capaz de relacionar os serviços oferecidos com os diferentes sistemas necessários para realizar devidas operações, também permitindo automatizações relacionadas à checagem de status de serviços e ao basear-se na inoperabilidade, também o cancelamento de agendamentos, antes da locomoção do usuário ou ao menos apresentando essa informação já no balcão de distribuição de senhas, reduzindo significativamente o tempo “desperdiçado” na espera.

A resolução dessa problemática pode ser um assunto delicado e complicado, mas que deve ser inicialmente considerado uma extensão ao que já funciona, isto é, o sistema de agendamento que o Poupatempo já implementa. Infelizmente, devido ao curto espaço de tempo que operamos essa pesquisa, não podemos dizer com clareza sobre a estrutura e o funcionamento deste - parte por serem informações sigilosas, em razão de segurança, parte pelo prazo de atendimento (tanto da Prodesp, responsável pelo sistema, quanto do Poupatempo em si) ser de 30 dias (prorrogáveis).

Porém, a partir dos conhecimentos obtidos a partir dos cursos de Banco e Estruturas de Dados, podemos deduzir essa estrutura e iniciar a implementação a partir dessa dedução. Também, apesar de concentrarmos nossa pesquisa na área de banco de dados, nos arriscamos em apresentar uma conceituação um pouco mais robusta e mais completa, pensando em outros campos da solução, justamente para podermos demonstrar que é possível realizar e implementar essa a solução.

Para isso, é necessário ampliar o campo de estudos para além da estruturação de um banco de dados relacional, mas pensar na implementação de monitoramento de integridade de sistemas, na comunicação via API RESTful e em estratégias de contato com o usuário final.

Para a estrutura de nosso banco de dados, é fundamental utilizarmos um banco relacional, isto por conta da necessidade que temos de manter integridade e consistência em um sistema de agendamentos, algo que um SGBD relacional consegue entregar com mais eficiência. Também é de suma importância nos lembrarmos dos conceitos ACID (Atomicidade, Consistência, Isolamento e Durabilidade), presentes em sistemas relacionais, garantindo que as transações (como agendamentos ou cancelamentos) possam ocorrer de maneira confiável. Além, claro, de notarmos que os dados que planejamos armazenar possuem forte relação entre si, algo que por si só já é uma característica idealmente ligada com o modelo relacional.

O monitoramento de integridade de sistemas já é algo comum e muito aplicado, consistindo em avaliações periódicas de status, disponibilidade e desempenho de nós ou serviços individuais, retornando ao consultante informações referentes aos resultados dessas verificações, a fim de identificar falhas antes que se tornem problemas mais graves. Na nossa solução, poderíamos focar em verificações simples no início, apenas nos certificando da disponibilidade dos servidores e sistemas necessários para determinado serviço.

Utilizaríamos também uma API RESTful - uma arquitetura cliente-servidor extremamente difundida e utilizada, que se conceitua como uma comunicação sem permanência de estado - como uma ponta de comunicação segura entre as diferentes necessidades do sistema com nosso banco de dados, oferecendo rotas que receberiam requisições HTTP simples, porém bem definidas, para realizar diferentes funções dentro da solução.

Como estratégia para contato com usuário final, poderíamos pensar em um sistema de notificação, baseado em uma arquitetura baseada em eventos. Dentro dessa estrutura, uma ação significativa no sistema (como a alteração de um status de disponibilidade), geraria um evento, que por sua vez acionaria o fluxo de trabalho automatizado que seria o responsável por identificar os serviços afetados e como efeito colateral os usuários que teriam seu agendamento comprometido, podendo lhes enviar notificações informando dessa indisponibilidade antes da chegada ao polo.

Também, para mantermos todas essas “áreas” no mesmo ambiente, podemos utilizar o Docker, uma tecnologia de conteinerização, que pode empacotar todas essas dependências em uma mesma unidade isolada chamada de contêiner, garantindo que tudo funcionará perfeitamente, de maneira isolada do resto do sistema e com uma

portabilidade interessante, já que ao funcionar como uma tecnologia de virtualização, podemos “subir” o ambiente independente do sistema operacional utilizado.

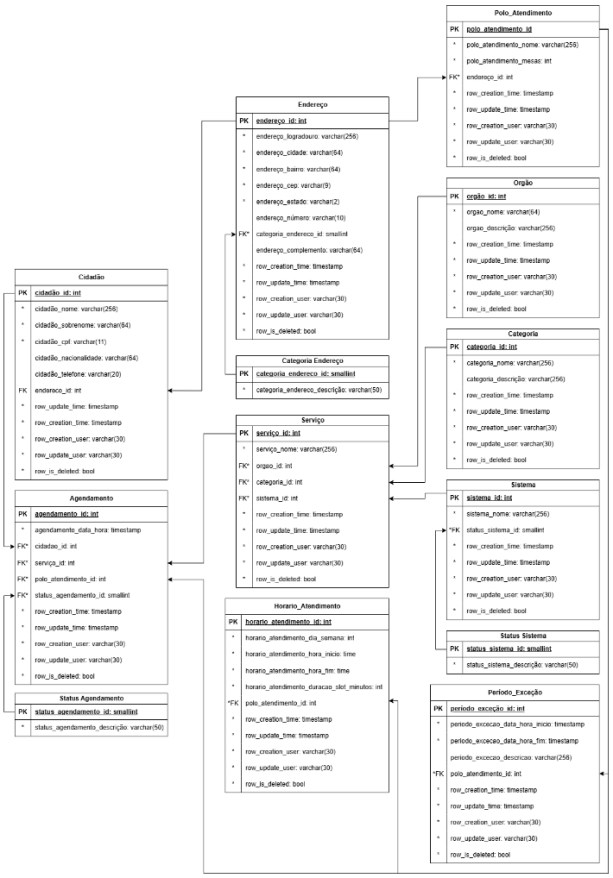
Tendo revisado as funcionalidades e outras frentes relevantes, podemos agora destinar nossa atenção para a estrutura pretendida ao nosso banco de dados. Para isso, utilizaremos um diagrama ER com as tabelas relevantes em nossa base de dados.

Dentre essas tabelas, podemos citar as principais: Cidadão, com os dados referentes aos usuários do Poupatempo; Polo Atendimento, com os dados dos polos de atendimento presencial do Poupatempo; Sistema, com as informações dos sistemas utilizados para realização de serviços; Serviço, com a centralização das informações referentes aos serviços prestados; Horário Atendimento, para manter armazenado os diferentes horários de atendimento disponíveis conforme polo de atendimento; Agendamento, guardando os diferentes horários marcados pelos usuários para atendimento presencial;

Também, faz sentido a criação de tabelas auxiliares, ou com função de separar informações, ou Endereço, com os endereços de usuários ou polos; Orgão, com informações dos órgãos públicos atendidos nos Poupatempos; Categoria, com as categorias de serviços prestados; Período Exceção, para guardar os dias e horários que polos específicos (ou o Poupatempo em geral) estará indisponível, seja por manutenção, feriados ou qualquer outras razões; Status Sistema, para mantermos os diferentes estados possíveis em que um sistema pode estar (Online, Offline, Em Manutenção); Status Agendamento, para os estados de um agendamento (Confirmado, Cancelado por Usuário, Cancelado Pelo Sistema, Concluído ou Não Compareceu); Categoria Endereço, para as 2 opções de endereço (De Polo ou De Cidadão).

Para melhor visualização dessas tabelas e dos dados que manteremos em cada uma, podemos visualizar abaixo o diagrama ER:

Figura 1 - Diagrama ER



Fonte: O autor (2025)

Após termos feito esse diagrama, partimos para o desenvolvimento prático desse banco de dados. Primeiro, para instanciar o mesmo, foi necessário pensar em uma maneira de isolar esse banco, inicializando o banco de maneira isolada, pensando justamente em um crescimento necessário da solução. Para isso, resolvemos utilizar o Docker, já citado anteriormente como uma tecnologia de conteinerização, podendo inicializar todas as outras dependências em uma mesma unidade isolada (contêiner).

Assim, criamos o docker-compose.yml, que irá definir a maneira que o contêiner será configurado; por desenvolvermos esse trabalho acerca do banco de dados, nos concentraremos nessa fração da solução proposta. O docker-compose pode ser consultado no apêndice A.

Um dos dados incluídos nesse docker-compose é uma referência ao script de criação do banco de dados, tabelas e etc. Ela está no segundo item do bloco “volumes”. Este script pode ser visualizado com mais detalhes no apêndice B.

Após a conclusão dessas etapas, apenas um simples comando (“docker-compose up -d”) é capaz de executar e configurar o banco de dados em sua forma final,

sendo facilmente manipulado e acessado para podermos verificar se a solução é realmente funcional, o que foi o caso.

Agora, após confirmar que a estrutura é funcional, podemos simular a utilização desse banco. Primeiro, pensaremos no processo inicial, por parte da equipe de suporte do sistema. Para disponibilizar o sistema aos cidadãos, é necessário cadastrar no sistema os diferentes polos e seus respectivos horários de atendimento e endereços, também os órgãos e categorias aos quais os serviços atenderão, bem como os serviços em si e os sistemas utilizados por eles.

A partir disso, seria necessário disponibilizar uma aplicação web ou mobile que disponibilize aos cidadãos a possibilidade de utilização da conta Gov (como já ocorre com os outros sistemas do Poupatempo ou outros sistemas governamentais) para realizar o agendamento.

O procedimento de agendamento ocorreria assim: Primeiramente, o usuário poderia usar órgãos e/ou categorias para filtrar por serviços específicos. Após selecionar o serviço e polo de atendimento escolhido, o sistema utilizaria as regras de atendimento desse polo para retornar ao usuário o “calendário” para escolha do dia e posteriormente, através de um cálculo utilizando hora_inicio, hora_fim e duração_slot_minutos, o horário de atendimento (exemplo: 09:00, 09:30, 10:00...). Levando em consideração que essa lista de dias e horários disponíveis já passou por uma validação, que remove os períodos presentes na tabela “Periodo_excecao”, além de utilizar o número de agendamentos feitos para garantir que não ultrapasse o número de mesas disponíveis cadastradas no polo. O processo de automação (comentado futuramente) garantirá que em caso de o sistema não estar disponível no momento em que o agendamento é feito, um período de tempo já é declarado como indisponível, levando em consideração um tempo para recuperação e em caso de o usuário decidir marcar para após esse período ou para o dia seguinte, receberá um aviso prévio avisando a provável indisponibilidade do sistema.

Agora, podemos descrever como funcionaria o processo de monitoramento e automação de cancelamento: Um dos “tentáculos” da nossa solução seria responsável por verificar a saúde dos sistemas. Este processo, conhecido como Health Check, seria um serviço automatizado que, a intervalos regulares, enviaria requisições para os sistemas externos dos quais os serviços do Poupatempo dependem. Ao detectar uma falha - seja por um erro de resposta ou tempo de requisição esgotado - este serviço não agiria diretamente no banco de dados. Em vez disso, seguindo uma arquitetura de eventos, ele publicaria uma mensagem em um tópico de um message broker como o Apache Kafka. Essa mensagem, ou "evento", conteria os dados essenciais, como o sistema_id que apresentou falha e seu novo status_sistema_id.

A partir daí, um segundo serviço, um "consumidor", estaria inscrito neste tópico do Kafka, aguardando por novos eventos. Ao receber a mensagem de falha, sua primeira responsabilidade seria atualizar o status_sistema_id na tabela “Sistema” do nosso banco de dados. Imediatamente após, este mesmo serviço iniciaria o fluxo de cancelamento: ele consultaria a tabela “Agendamento” para encontrar todos os agendamentos futuros em um período de tempo pré-estabelecido pós checagem, ainda com status “Confirmado”, que dependem dos serviços afetados pela queda do sistema.

Para cada agendamento encontrado, o status seria atualizado para “Cancelado pelo Sistema”. A cada atualização bem-sucedida, um novo evento seria publicado em um segundo tópico do Kafka, dedicado exclusivamente às notificações. Um terceiro serviço, o "notificador", consumiria essas mensagens e, utilizando os dados do cidadão e do agendamento, se encarregaria de formatar e enviar o aviso final por e-mail, SMS ou notificação push. Essa arquitetura desacoplada garante que o sistema seja resiliente, escalável e que o usuário final seja notificado com eficiência, mitigando a frustração que deu origem a este estudo.

Os artefatos técnicos desenvolvidos para este projeto, incluindo o script SQL completo e o arquivo de configuração do Docker, estão disponíveis para consulta no repositório online **(LUBAWASKI, 2025)**.

4. REFLEXÕES FINAIS

Com a conclusão dessa pesquisa, podemos afirmar que o objetivo inicial e principal, de idealizar um banco de dados completo para o sistema do Poupatempo, foi plenamente alcançado, principalmente evidenciado através do script disponível no apêndice B. Acredito que a principal dificuldade encontrada foi a ausência de dados técnicos oficiais, o que exigiu a utilização de uma abordagem mais dedutiva, baseada em boas práticas do mercado. Posso dizer que o impacto dessa pesquisa em minha vida profissional foi ótimo, especialmente pela necessidade do pensamento crítico e criativo, pensando em todas as frentes que entrariam em conjunto com esse banco de da dados para a resolução total da problemática apresentada.

Ainda mais importante que meu desenvolvimento pessoal, é a comprovação que esse sistema é possível e aplicável, não sendo de um desenvolvimento tão extenso e complexo quanto pensado inicialmente. Logo, podemos concluir que o Poupatempo poderia sim se adaptar e resolver essa questão tão complicada e triste para com os cidadãos paulistas.

5. REFERÊNCIAS

AMAZON WEB SERVICES. O que é uma API RESTful?. Disponível em: <https://aws.amazon.com/pt/what-is/restful-api/>. Acesso em: 25 jun. 2025.

AMAZON WEB SERVICES. O que é uma arquitetura orientada por eventos?. Disponível em: <https://aws.amazon.com/pt/event-driven-architecture/>. Acesso em: 25 jun. 2025.

AMAZON WEB SERVICES. O que é engenharia de confiabilidade de sites (SRE)?. Disponível em: <https://aws.amazon.com/pt/what-is/sre/>. Acesso em: 25 jun. 2025.

IBM. O que é uma API REST?. Publicado em: 24 abr. 2025. Disponível em: <https://www.ibm.com/br-pt/think/topics/rest-apis>. Acesso em: 25 jun. 2025.

ISAÍAS, Ayooluwa. Introdução às verificações de integridade do servidor. Disponível em: <https://betterstack.com/community/guides/monitoring/health-checks/>. Acesso em: 25 jun. 2025.

LUBAWASKI, João Paulo. Poupatempo. 2025. Repositório de código em GitHub. Disponível em: https://github.com/JoaoLubaw/Poupatempo. Acesso em: 2 jul. 2025.

MICROSOFT. Verificações de integridade do aplicativo .NET em C#. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/core/diagnostics/diagnostic-health-checks>. Acesso em: 25 jun. 2025.

RECLAME AQUI. Sem sistema. Publicado em: 10 set. 2024. Disponível em: <https://www.reclameaqui.com.br/poupatempo-sp/sem-sistema_RdjxRpkvDdbsVvqm/>. Acesso em: 25 jun. 2025.

RODRIGUES, Rodrigo. Quedas constantes de sistema no Poupatempo geram reclamações: 'tive que remarcar três vezes um atendimento', conta usuária. G1, São Paulo, 17 mar. 2025. Disponível em: <https://g1.globo.com/sp/sao-paulo/noticia/2025/03/17/quedas-constantes-de-sistema-no-poupatempo-geram-reclamacoes-tive-que-remarcar-tres-vezes-um-atendimento-conta-usuaria.ghtml>. Acesso em: 25 jun. 2025.

SAP. O que é arquitetura baseada em eventos?. Disponível em: <https://www.sap.com/brazil/products/technology-platform/what-is-event-driven-architecture.html>. Acesso em: 25 jun. 2025.

SUSNJARA, Stephanie; **SMALLEY, Ian**. O que é Docker?. IBM Think, 6 jun. 2024. Disponível em: <https://www.ibm.com/br-pt/think/topics/docker>. Acesso em: 1 jul. 2025.

YANACEK, David. Como implementar verificações de integridade. Disponível em: <https://aws.amazon.com/pt/builders-library/implementing-health-checks/>. Acesso em: 25 jun. 2025.

APÊNDICE A – Docker-compose.yml de banco de dados

```
# Define a versão da sintaxe do arquivo Docker Compose.
version: "3.8"

# Bloco principal onde todos os serviços (containers) da sua aplicação são definidos.
services:
  # Nome lógico do nosso serviço de banco de dados dentro deste arquivo.
  banco-poupatempo:
    # Especifica a imagem Docker que será usada para criar o container.
    image: postgres:latest

    # Define um nome fixo e amigável para o container, facilitando o gerenciamento.
    container_name: poupatempo_db

    # Política de reinicialização: o container sempre será iniciado com o Docker,
    # a menos que seja parado manualmente pelo usuário.
    restart: unless-stopped

    # Define variáveis de ambiente que serão usadas para configurar o PostgreSQL
    # na sua primeira inicialização.
    environment:
      POSTGRES_USER: admin_poupatempo # Nome do superusuário.
      POSTGRES_PASSWORD: senha_poupatempo # Senha para o superusuário.
      POSTGRES_DB: poupatempo_db # Nome do banco criado.

    # Mapeia as portas no formato "<portaMaquina>:<portaConteiner>".
    ports:
      - "1234:5432" # Do 1234 do computador para a porta 5432 do container.

    # Define os volumes para persistir dados e carregar scripts.
    volumes:
      # Garante que os dados do banco de dados sejam salvos e persistam
      # mesmo que o container seja removido.
      - poupatempo_data:/var/lib/postgresql/data
      # Bind mount: mapeia a pasta local "init" para a pasta especial do container
      # que executa scripts .sql automaticamente na primeira inicialização.
      - ./init:/docker-entrypoint-initdb.d

# Declaração do volume nomeado usado pelo serviço acima.
volumes:
  poupatempo_data:
```

APÊNDICE B – Script de criação de banco de dados

```
-- ================================================================
-- FUNÇÃO DE TRIGGER (Executar uma única vez no início)
-- ================================================================
CREATE OR REPLACE FUNCTION public.update_row_update_time()
RETURNS TRIGGER AS
$$
BEGIN
        NEW.row_update_time := NOW();
        RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';
COMMENT ON FUNCTION public.update_row_update_time() IS 'Função de trigger
geral para atualizar os metadados de uma linha sempre que um UPDATE ocorrer.';


-- ================================================================
-- TABELAS DE DOMÍNIO (Lookup Tables)
-- ================================================================
DROP TABLE IF EXISTS public.status_agendamento CASCADE;
CREATE TABLE public.status_agendamento (
   status_agendamento_id SMALLINT PRIMARY KEY,
   status_agendamento_descricao VARCHAR(50) NOT NULL
);
INSERT INTO public.status_agendamento (status_agendamento_id,
status_agendamento_descricao) VALUES
(1, 'Confirmado'), (2, 'Cancelado pelo Usuário'), (3, 'Cancelado pelo Sistema'), (4,
'Concluído'), (5, 'Não Compareceu');

DROP TABLE IF EXISTS public.status_sistema CASCADE;
CREATE TABLE public.status_sistema (
   status_sistema_id SMALLINT PRIMARY KEY,
   status_sistema_descricao VARCHAR(50) NOT NULL
);
INSERT INTO public.status_sistema (status_sistema_id, status_sistema_descricao)
VALUES
(1, 'Online'), (2, 'Offline'), (3, 'Em Manutenção');
```

```
DROP TABLE IF EXISTS public.categoria_endereco CASCADE;
CREATE TABLE public.categoria_endereco (
    categoria_endereco_id SMALLINT PRIMARY KEY,
    categoria_endereco_descricao VARCHAR(50) NOT NULL
);
INSERT INTO public.categoria_endereco (categoria_endereco_id,
categoria_endereco_descricao) VALUES
(1, 'Polo de Atendimento'), (2, 'Endereço de Cidadão');


-- ================================================================
-- SCRIPT PARA TABELA: Endereco
-- ================================================================
/* Drop Sequence */
DROP SEQUENCE IF EXISTS public.endereco_id_seq;
/* Drop Table */
DROP TABLE IF EXISTS public.endereco CASCADE;
/* Create Table */
CREATE TABLE public.endereco
(
    endereco_id integer NOT NULL DEFAULT
NEXTVAL(('endereco_id_seq'::text)::regclass), -- Chave primária da tabela.
    endereco_logradouro varchar(256) NOT NULL, -- Nome da rua, avenida, etc.
    endereco_numero varchar(10) NULL, -- Número do imóvel, talvez com 'A' ou 'S/N'.
    endereco_complemento varchar(64) NULL, -- Complemento como apartamento.
    endereco_bairro varchar(64) NOT NULL, -- Bairro.
    endereco_cidade varchar(64) NOT NULL, -- Cidade.
    endereco_estado varchar(2) NOT NULL, -- Sigla do estado (UF).
    endereco_cep varchar(9) NOT NULL, -- CEP no formato '12345-678'.
    categoria_endereco_id smallint NOT NULL,
    row_creation_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    row_update_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    row_creation_user varchar(30) NOT NULL DEFAULT 'system',
    row_update_user varchar(30) NOT NULL DEFAULT 'system',
    row_is_deleted boolean NOT NULL DEFAULT False
);
/* Create Primary Key */
ALTER TABLE public.endereco ADD CONSTRAINT endereco_pk PRIMARY KEY
(endereco_id);
/* Create Foreign Key */
ALTER TABLE public.endereco ADD CONSTRAINT endereco_fk1 FOREIGN KEY
(categoria_endereco_id) REFERENCES public.categoria_endereco
(categoria_endereco_id) ON DELETE No Action ON UPDATE No Action;
/* Create Trigger */
CREATE TRIGGER endereco_upd_trigger BEFORE UPDATE ON public.endereco
FOR EACH ROW EXECUTE PROCEDURE update_row_update_time();
/* Create Comments and Sequence */
COMMENT ON TABLE public.endereco IS 'Armazena informações de endereço para
cidadãos e polos de atendimento.';
CREATE SEQUENCE public.endereco_id_seq INCREMENT 1 START 1;
```

```
-- ================================================================
-- SCRIPT PARA TABELA: Cidadao
-- ================================================================
/* Drop Sequence */
DROP SEQUENCE IF EXISTS public.cidadao_id_seq;
/* Drop Table */
DROP TABLE IF EXISTS public.cidadao CASCADE;
/* Create Table */
CREATE TABLE public.cidadao
(
   cidadao_id integer NOT NULL DEFAULT
NEXTVAL(('cidadao_id_seq'::text)::regclass),
   cidadao_nome varchar(256) NOT NULL,
   cidadao_sobrenome varchar(64) NOT NULL,
   cidadao_cpf varchar(11) NOT NULL,
   cidadao_nacionalidade varchar(64) NOT NULL,
   cidadao_telefone varchar(20) NOT NULL,
   endereco_id integer NOT NULL,
   row_creation_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
   row_update_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
   row_creation_user varchar(30) NOT NULL DEFAULT 'system',
   row_update_user varchar(30) NOT NULL DEFAULT 'system',
   row_is_deleted boolean NOT NULL DEFAULT False
);
/* Create Primary and Unique Keys */
ALTER TABLE public.cidadao ADD CONSTRAINT cidadao_pk PRIMARY KEY
(cidadao_id);
ALTER TABLE public.cidadao ADD CONSTRAINT cidadao_uk1 UNIQUE
(cidadao_cpf);
/* Create Foreign Key */
ALTER TABLE public.cidadao ADD CONSTRAINT cidadao_fk1 FOREIGN KEY
(endereco_id) REFERENCES public.endereco (endereco_id) ON DELETE No Action
ON UPDATE No Action;
/* Create Trigger */
CREATE TRIGGER cidadao_upd_trigger BEFORE UPDATE ON public.cidadao FOR
EACH ROW EXECUTE PROCEDURE update_row_update_time();
/* Create Comments and Sequence */
COMMENT ON TABLE public.cidadao IS 'Armazena as informações dos cidadãos.';
CREATE SEQUENCE public.cidadao_id_seq INCREMENT 1 START 1;

-- ================================================================
-- SCRIPT PARA TABELA: Orgao
-- ================================================================
/* Drop Sequence */
DROP SEQUENCE IF EXISTS public.orgao_id_seq;
/* Drop Table */
DROP TABLE IF EXISTS public.orgao CASCADE;
/* Create Table */
CREATE TABLE public.orgao (
   orgao_id integer NOT NULL DEFAULT NEXTVAL(('orgao_id_seq'::text)::regclass),
   orgao_nome varchar(64) NOT NULL,
   orgao_descricao varchar(256) NOT NULL,
   row_creation_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
   row_update_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
```

```
    row_creation_user varchar(30) NOT NULL DEFAULT 'system',
    row_update_user varchar(30) NOT NULL DEFAULT 'system',
    row_is_deleted boolean NOT NULL DEFAULT False
);
/* Create Primary Key */
ALTER TABLE public.orgao ADD CONSTRAINT orgao_pk PRIMARY KEY (orgao_id);
/* Create Trigger */
CREATE TRIGGER orgao_upd_trigger BEFORE UPDATE ON public.orgao FOR
EACH ROW EXECUTE PROCEDURE update_row_update_time();
/* Create Comments and Sequence */
COMMENT ON TABLE public.orgao IS 'Órgãos governamentais responsáveis pelos
serviços (ex: Detran, IIRGD).';
CREATE SEQUENCE public.orgao_id_seq INCREMENT 1 START 1;


-- ================================================================
-- SCRIPT PARA TABELA: Categoria
-- ================================================================
/* Drop Sequence */
DROP SEQUENCE IF EXISTS public.categoria_id_seq;
/* Drop Table */
DROP TABLE IF EXISTS public.categoria CASCADE;
/* Create Table */
CREATE TABLE public.categoria (
    categoria_id integer NOT NULL DEFAULT
NEXTVAL(('categoria_id_seq'::text)::regclass),
    categoria_nome varchar(256) NOT NULL,
    row_creation_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    row_update_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    row_creation_user varchar(30) NOT NULL DEFAULT 'system',
    row_update_user varchar(30) NOT NULL DEFAULT 'system',
    row_is_deleted boolean NOT NULL DEFAULT False
);
/* Create Primary Key */
ALTER TABLE public.categoria ADD CONSTRAINT categoria_pk PRIMARY KEY
(categoria_id);
/* Create Trigger */
CREATE TRIGGER categoria_upd_trigger BEFORE UPDATE ON public.categoria
FOR EACH ROW EXECUTE PROCEDURE update_row_update_time();
/* Create Comments and Sequence */
COMMENT ON TABLE public.categoria IS 'Categorias para agrupar os serviços (ex:
Veículos, Documentos Pessoais).';
CREATE SEQUENCE public.categoria_id_seq INCREMENT 1 START 1;


-- ================================================================
-- SCRIPT PARA TABELA: Polo_Atendimento
-- ================================================================
/* Drop Sequence */
DROP SEQUENCE IF EXISTS public.polo_atendimento_id_seq;
/* Drop Table */
DROP TABLE IF EXISTS public.polo_atendimento CASCADE;
/* Create Table */
CREATE TABLE public.polo_atendimento
(
```

```
    polo_atendimento_id integer NOT NULL DEFAULT
NEXTVAL(('polo_atendimento_id_seq'::text)::regclass),
    polo_atendimento_nome varchar(256) NOT NULL,
    polo_atendimento_mesas integer NOT NULL,
    endereco_id integer NOT NULL,
    row_creation_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    row_update_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    row_creation_user varchar(30) NOT NULL DEFAULT 'system',
    row_update_user varchar(30) NOT NULL DEFAULT 'system',
    row_is_deleted boolean NOT NULL DEFAULT False
);
/* Create Primary Key */
ALTER TABLE public.polo_atendimento ADD CONSTRAINT polo_atendimento_pk
PRIMARY KEY (polo_atendimento_id);
/* Create Foreign Key */
ALTER TABLE public.polo_atendimento ADD CONSTRAINT polo_atendimento_fk1
FOREIGN KEY (endereco_id) REFERENCES public.endereco (endereco_id) ON
DELETE No Action ON UPDATE No Action;
/* Create Trigger */
CREATE TRIGGER polo_atendimento_upd_trigger BEFORE UPDATE ON
public.polo_atendimento FOR EACH ROW EXECUTE PROCEDURE
update_row_update_time();
/* Create Comments and Sequence */
COMMENT ON TABLE public.polo_atendimento IS 'Armazena os postos de
atendimento do Poupatempo.';
COMMENT ON COLUMN public.polo_atendimento.polo_atendimento_mesas IS
'Capacidade total de atendimentos simultâneos no posto.';
CREATE SEQUENCE public.polo_atendimento_id_seq INCREMENT 1 START 1;


-- ================================================================
-- SCRIPT PARA TABELA: Sistema
-- ================================================================
/* Drop Sequence */
DROP SEQUENCE IF EXISTS public.sistema_id_seq;
/* Drop Table */
DROP TABLE IF EXISTS public.sistema CASCADE;
/* Create Table */
CREATE TABLE public.sistema (
    sistema_id integer NOT NULL DEFAULT
NEXTVAL(('sistema_id_seq'::text)::regclass),
    sistema_nome varchar(256) NOT NULL,
    status_sistema_id smallint NOT NULL,
    row_creation_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    row_update_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    row_creation_user varchar(30) NOT NULL DEFAULT 'system',
    row_update_user varchar(30) NOT NULL DEFAULT 'system',
    row_is_deleted boolean NOT NULL DEFAULT False
);
/* Create Primary Key */
ALTER TABLE public.sistema ADD CONSTRAINT sistema_pk PRIMARY KEY
(sistema_id);
/* Create Foreign Key */
```

```
ALTER TABLE public.sistema ADD CONSTRAINT sistema_fk1 FOREIGN KEY
(status_sistema_id) REFERENCES public.status_sistema(status_sistema_id) ON
DELETE No Action ON UPDATE No Action;
/* Create Trigger */
CREATE TRIGGER sistema_upd_trigger BEFORE UPDATE ON public.sistema FOR
EACH ROW EXECUTE PROCEDURE update_row_update_time();
/* Create Comments and Sequence */
COMMENT ON TABLE public.sistema IS 'Sistemas externos dos quais os serviços
dependem (ex: sistema do Detran).';
CREATE SEQUENCE public.sistema_id_seq INCREMENT 1 START 1;


-- ================================================================
-- SCRIPT PARA TABELA: Servico
-- ================================================================
/* Drop Sequence */
DROP SEQUENCE IF EXISTS public.servico_id_seq;
/* Drop Table */
DROP TABLE IF EXISTS public.servico CASCADE;
/* Create Table */
CREATE TABLE public.servico (
   servico_id integer NOT NULL DEFAULT
NEXTVAL(('servico_id_seq'::text)::regclass),
   servico_nome varchar(256) NOT NULL,
   orgao_id integer NOT NULL,
   categoria_id integer NOT NULL,
   sistema_id integer NOT NULL,
   row_creation_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
   row_update_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
   row_creation_user varchar(30) NOT NULL DEFAULT 'system',
   row_update_user varchar(30) NOT NULL DEFAULT 'system',
   row_is_deleted boolean NOT NULL DEFAULT False
);
/* Create Primary Key */
ALTER TABLE public.servico ADD CONSTRAINT servico_pk PRIMARY KEY
(servico_id);
/* Create Foreign Keys */
ALTER TABLE public.servico ADD CONSTRAINT servico_fk1 FOREIGN KEY
(orgao_id) REFERENCES public.orgao(orgao_id) ON DELETE No Action ON UPDATE
No Action;
ALTER TABLE public.servico ADD CONSTRAINT servico_fk2 FOREIGN KEY
(categoria_id) REFERENCES public.categoria(categoria_id) ON DELETE No Action
ON UPDATE No Action;
ALTER TABLE public.servico ADD CONSTRAINT servico_fk3 FOREIGN KEY
(sistema_id) REFERENCES public.sistema(sistema_id) ON DELETE No Action ON
UPDATE No Action;
/* Create Trigger */
CREATE TRIGGER servico_upd_trigger BEFORE UPDATE ON public.servico FOR
EACH ROW EXECUTE PROCEDURE update_row_update_time();
/* Create Comments and Sequence */
COMMENT ON TABLE public.servico IS 'Catálogo de serviços específicos oferecidos
(ex: "Primeira Habilitação").';
CREATE SEQUENCE public.servico_id_seq INCREMENT 1 START 1;
```

```
-- ================================================================
-- SCRIPT PARA TABELA: Horario_Atendimento
-- ================================================================
/* Drop Sequence */
DROP SEQUENCE IF EXISTS public.horario_atendimento_id_seq;
/* Drop Table */
DROP TABLE IF EXISTS public.horario_atendimento CASCADE;
/* Create Table */
CREATE TABLE public.horario_atendimento (
    horario_atendimento_id integer NOT NULL DEFAULT
NEXTVAL(('horario_atendimento_id_seq'::text)::regclass),
    horario_atendimento_dia_semana smallint NOT NULL, -- 1=Domingo, 2=Segunda,
etc.
    horario_atendimento_hora_inicio time NOT NULL,
    horario_atendimento_hora_fim time NOT NULL,
    horario_atendimento_duracao_slot_minutos integer NOT NULL,
    polo_atendimento_id integer NOT NULL,
    row_creation_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    row_update_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    row_creation_user varchar(30) NOT NULL DEFAULT 'system',
    row_update_user varchar(30) NOT NULL DEFAULT 'system',
    row_is_deleted boolean NOT NULL DEFAULT False
);
/* Create Primary Key */
ALTER TABLE public.horario_atendimento ADD CONSTRAINT
horario_atendimento_pk PRIMARY KEY (horario_atendimento_id);
/* Create Foreign Key */
ALTER TABLE public.horario_atendimento ADD CONSTRAINT
horario_atendimento_fk1 FOREIGN KEY (polo_atendimento_id) REFERENCES
public.polo_atendimento(polo_atendimento_id) ON DELETE No Action ON UPDATE
No Action;
/* Create Trigger */
CREATE TRIGGER horario_atendimento_upd_trigger BEFORE UPDATE ON
public.horario_atendimento FOR EACH ROW EXECUTE PROCEDURE
update_row_update_time();
/* Create Comments and Sequence */
COMMENT ON TABLE public.horario_atendimento IS 'Define a grade de horários
padrão de cada posto de atendimento.';
CREATE SEQUENCE public.horario_atendimento_id_seq INCREMENT 1 START 1;


-- ================================================================
-- SCRIPT PARA TABELA: Periodo_Excecao
-- ================================================================
/* Drop Sequence */
DROP SEQUENCE IF EXISTS public.periodo_excecao_id_seq;
/* Drop Table */
DROP TABLE IF EXISTS public.periodo_excecao CASCADE;
/* Create Table */
CREATE TABLE public.periodo_excecao (
                periodo_excecao_id      integer      NOT      NULL      DEFAULT
NEXTVAL(('periodo_excecao_id_seq'::text)::regclass),
    polo_atendimento_id integer NULL,
    periodo_excecao_data_hora_inicio timestamp NOT NULL,
    periodo_excecao_data_hora_fim timestamp NOT NULL,
```

```
    periodo_excecao_descricao varchar(256) NOT NULL,
    row_creation_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    row_update_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    row_creation_user varchar(30) NOT NULL DEFAULT 'system',
    row_update_user varchar(30) NOT NULL DEFAULT 'system',
    row_is_deleted boolean NOT NULL DEFAULT False
);
/* Create Primary Key */
ALTER TABLE public.periodo_excecao ADD CONSTRAINT periodo_excecao_pk
PRIMARY KEY (periodo_excecao_id);
/* Create Foreign Key */
ALTER TABLE public.periodo_excecao ADD CONSTRAINT periodo_excecao_fk1
FOREIGN KEY (polo_atendimento_id) REFERENCES
public.polo_atendimento(polo_atendimento_id) ON DELETE No Action ON UPDATE No
Action;
/* Create Trigger */
CREATE TRIGGER periodo_excecao_upd_trigger BEFORE UPDATE ON
public.periodo_excecao FOR EACH ROW EXECUTE PROCEDURE
update_row_update_time();
/* Create Comments and Sequence */
COMMENT ON TABLE public.periodo_excecao IS 'Armazena exceções à grade padrão,
como feriados e manutenções.';
CREATE SEQUENCE public.periodo_excecao_id_seq INCREMENT 1 START 1;


-- ================================================================
-- SCRIPT PARA TABELA: Agendamento
-- ================================================================
/* Drop Sequence */
DROP SEQUENCE IF EXISTS public.agendamento_id_seq;
/* Drop Table */
DROP TABLE IF EXISTS public.agendamento CASCADE;
/* Create Table */
CREATE TABLE public.agendamento (
    agendamento_id integer NOT NULL DEFAULT
NEXTVAL(('agendamento_id_seq'::text)::regclass),
    agendamento_data_hora timestamp NOT NULL,
    cidadao_id integer NOT NULL,
    servico_id integer NOT NULL,
    polo_atendimento_id integer NOT NULL,
    status_agendamento_id smallint NOT NULL,
    row_creation_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    row_update_time timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    row_creation_user varchar(30) NOT NULL DEFAULT 'system',
    row_update_user varchar(30) NOT NULL DEFAULT 'system',
    row_is_deleted boolean NOT NULL DEFAULT False
);
/* Create Primary Key */
ALTER TABLE public.agendamento ADD CONSTRAINT agendamento_pk PRIMARY
KEY (agendamento_id);
/* Create Foreign Keys */
ALTER TABLE public.agendamento ADD CONSTRAINT agendamento_fk1 FOREIGN
KEY (cidadao_id) REFERENCES public.cidadao(cidadao_id) ON DELETE No Action
ON UPDATE No Action;
```

```
ALTER TABLE public.agendamento ADD CONSTRAINT agendamento_fk2 FOREIGN
KEY (servico_id) REFERENCES public.servico(servico_id) ON DELETE No Action ON
UPDATE No Action;
ALTER TABLE public.agendamento ADD CONSTRAINT agendamento_fk3 FOREIGN
KEY (polo_atendimento_id) REFERENCES
public.polo_atendimento(polo_atendimento_id) ON DELETE No Action ON UPDATE
No Action;
ALTER TABLE public.agendamento ADD CONSTRAINT agendamento_fk4 FOREIGN
KEY (status_agendamento_id) REFERENCES
public.status_agendamento(status_agendamento_id) ON DELETE No Action ON
UPDATE No Action;
/* Create Trigger */
CREATE TRIGGER agendamento_upd_trigger BEFORE UPDATE ON
public.agendamento FOR EACH ROW EXECUTE PROCEDURE
update_row_update_time();
/* Create Comments and Sequence */
COMMENT ON TABLE public.agendamento IS 'Tabela principal que armazena todos
os agendamentos realizados.';
CREATE SEQUENCE public.agendamento_id_seq INCREMENT 1 START 1;


-- ============================================================
-- ÍNDICES PARA OTIMIZAÇÃO DE CONSULTAS
-- ============================================================
CREATE INDEX idx_agendamento_data_hora ON public.agendamento
(agendamento_data_hora);
CREATE INDEX idx_agendamento_polo_id ON public.agendamento
(polo_atendimento_id);
CREATE INDEX idx_agendamento_polo_data ON public.agendamento
(polo_atendimento_id, agendamento_data_hora);
CREATE INDEX idx_horario_polo_dia ON public.horario_atendimento
(polo_atendimento_id, horario_dia_semana);
CREATE INDEX idx_excecao_polo ON public.periodo_excecao
(polo_atendimento_id);
```